H.A.S. Project

Group 5
Fall 2015

Jeffrey Benoit
D'Voran McIntosh
Roneal Valmonte
Zachary Zapasnik

Sponsored By Leidos

Table of Contents

1. Executive Summary

In this age we have more electronic devices in our homes than ever before. People are also busier than ever before, being either wrapped up in work, absorbed in some form of entertainment, or overloaded with the responsibilities of daily life. With all of these things going on, sometimes you forget to turn off or unplug one of your many electronic devices, or lights at home. Maybe you can't remember if you locked the door on your way out. Maybe you just got into bed and realized you forgot to turn off the lights in the kitchen. Perhaps you're energy conscious and want to make sure you've set your air conditioning to turn off while you aren't at home.

Our electronic home automation system seeks to fix all of those problems with the help of the prevalence of smart devices and computers in everyone's lives today. We are developing a number of devices to add to your home that will give you control over the electricity you use every day.

Our project is a home automation and power monitoring system comprised of four different physical devices connected to and controlled by a web based system. The four physical devices are: a wall outlet, a wall switch, an HVAC controller, and an electronic door lock hereafter referred to as a powerlock. Primary interaction with all four devices is done through a smartphone and web browser application. This application will let you see how much power any outlet or wall switch controlled fixture is using. The application will also show you details on your heating and air conditioning usage, and whether the powerlock is currently locked or unlocked. The application enables the user to fully control all of these devices from a smart device or any computer with access to the internet. A user will be able to turn on or off any wall outlet or wall switch that is connect, lock or unlock the power lock, and set the thermostat settings of the HVAC controller. On top of this users will be able to set weekly and daily schedules for these events to happen automatically. This will be achieved through the use of microcontrollers integrated in the circuits of each of the devices that will bridge the gap between hardware and the online software.

2. Project Description

2.1. Motivations

This project started with the idea of being able to control the electricity in your home from anywhere, this was our initial motivation. The idea of the smart wall outlet came first, a simple device that would let you turn on and off appliances and devices plugged into your home's wall outlets from a simple app on your smartphone or computer. Not everything in the home is plugged into a wall outlet however, and we latched onto the thought of the simple paranoia of forgetting

whether you turned something off. That is our main motivation: not having to worry about leaving devices and appliances in your home turned on when you leave. Our main motivation then lead the way for us to decide to make the electronic powerlock, as the anxiety of forgetting to lock your home is very similar to forgetting to turn off a device.

As our thoughts on the project solidified, we had a realization about another motivation. With how integrated into the home our project had become, we decided to also focus on some aspects of home automation. Being able to set schedules for when your appliances and devices are on and off, monitoring your power usage, and being able to control your heating and air conditioning as well became another part of the project. Our two main motivations now are: not having to worry about leaving electronics on, and managing the electricity in your house from anywhere.

2.2. Goals

We have a number of goals that we would like to accomplish with this project. Firstly, we are striving to make products that are compact and easily fit where the devices they are replacing already are. There should ideally be no need to move furniture or change the wiring of a building to accommodate our devices. Secondly, our devices should not require a substantial amount of electrical power. Thirdly, our devices should be easy and simple to use, with an interface that can be quickly learned by anyone. Fourthly, the response time of our devices should be quick, with little noticeable delay between selecting an action and that action occurring. Fifthly, we aim to make our products inexpensive and little to no costs required for upkeep. Sixthly, our products should be both easy to install and not difficult to remove or replace. Seventhly, adding additional devices to the system should be not only possible, but simple.

2.3. Objectives

Our objectives for this project help us to maintain our motivations and comply with our goals. The size of our devices, how quickly they respond to changes made by the mobile and web applications, and their cost are all quantifiable aspects of our devices that will have an effect on the ability of our project to meet our goals.

Size is important for making our devices fit well within homes. For our wall outlet device our objective is to keep it to a size of 5” W x 5” H x 1.5” D at most. We have the same objective for the size of the wall switch: a maximum size of 5” W x 5” H x 1.5” D. Our objective for the size of the HVAC controller is a maximum of 6” W x 5” H x 2” D. Our objective for the size of the powerlock is a maximum size of 5” W x 4” H x 1.5” D.

Response time and cost are more universal to our devices than size is. For all of our devices our objective is to have a response time under 2 seconds. A response time of less than 1 second would be more desirable, but 2 seconds is the longest response time we feel is acceptable for our products. For the cost of our devices, no individual device should cost more than 30 US dollars in parts. $20 is more ideal, however $30 is our highest acceptable price to meet our goal of being able to easily replace or add devices in or to our system.

3. Related Standards and Constraints

3.1. Standards

3.1.1. Hardware Standards

The following sections contain standards relating to the hardware components of the various devices within the project. Each device with relevant standards will be listed in its own section.

3.1.1.1. Wall Outlet

The wall outlet hardware will use the NEMA 5-15 grounded plug standard, also referred to as type B. This plug type accommodates all forms of plugs used in our primary market of North America. This plug standard is also recommended by IEC standard 60906-2 for installations of 120 volts and 60 Hz, as is the case with standard North American installations. In addition to this, as our wall outlet device is intended for indoor use, for the enclosure of the wall outlet we will use the standard for a NEMA Type 1 enclosure.

3.1.1.2. Wall Switch

The wall switch will adhere to IEC standard 60669 Switches for household and similar fixed-electrical installations. Our wall switch falls within the requirements range of this standard, which is a voltage not exceeding 440 volts, and a current not exceeding 63 amps. In addition to this, as our wall switch device is intended for indoor use, for the enclosure of the wall switch we will use the standard for a NEMA Type 1 enclosure.

3.1.1.3. HVAC Controller

For the HVAC controller we will use the NEMA DC 3-2013 standards and guidelines for Residential Controls–Electrical Wall-Mounted Room Thermostats. As we are aiming for a smart programmable system, and power management is

a core philosophy for us, we will also use NEMA DC3, Annex A-2013, Energy-Efficiency Requirements for Programmable Thermostats.

3.1.2. Software Standards and Libraries

The following sections contain any standards and/or libraries relevant to the software components of the project that will be adhered to. The sections are split between the microcontroller and the web and mobile application. The web/mobile application section also contains all standards and libraries relevant to all web based systems used within the project not just those for the base application.

3.1.2.1. Microcontroller Software Libraries

In all microcontrollers, the C standard library will be used. The ATSAMC21J18A firmware will also be used, which allows for the use of the features included on the microcontroller. In order to obtain an encrypted TLS connection with the online server the TLS mbed open source SSL library 2.2.0 will be used. The TLS mbed open source SSL library is compliant with the TLS version 1.2 standards, and the SSL version 3 standards.

3.1.2.2. Mobile/Web Application Libraries

The home automation system will be controlled using a custom cloud-hosted API with a user friendly interface. In order to provide a seamless cross device user experience, we will be using libraries and standards which can work on all modern day devices.

1. AngularJS – angularjs.org

   AngularJS is an open source model-view-controller (MVC) framework sponsored by Google that allows in-client rendering of dynamic HTML views rendered from data models served directly by an application program interface (API) by a JavaScript controller. In layman's terms, an MVC framework like AngularJS allows for client-side applications in HTML and JavaScript. It's similar to jQuery, which just handles view manipulation in JavaScript. AngularJS will allow for code reusability across all web and mobile devices.

2. PhoneGap – phonegap.com

   PhoneGap is an open source framework sponsored by Adobe that allows development of HTML, CSS, and JavaScript mobile applications for Apple's iOS, Google's Android OS, and Microsoft's Windows Phone OS. It primarily serves as a built script that makes use of each phone's existing operating system APIs for building web views. This allows for code reusability across all

mobile devices, and will allow all mobile apps to look and function identically regardless of the operating system.

3. Bootstrap – getbootstrap.com

   Bootstrap is an open source HTML and CSS starter framework sponsored by Twitter that allows for websites to more easily develop modern, responsive interfaces that work on both desktop and mobile sized devices.

4. Node JS – nodejs.org

   Node JS is an open source JavaScript runtime sponsored by the Linux Foundation. It allows for server-side execution of JavaScript to develop event driven and non blocking IO requiring applications, like a home automation system may need. Runtimes like Java and PHP are IO blocking, which means that incoming requests would block until completion, severely limiting the throughput a single server could handle.

5. Express JS – expressjs.com

   Express JS is an open source HTTP routing framework for Node JS sponsored by StrongLoop, an IBM company. It allows for authentication, parameters, and other data to be assigned to routes, which is useful for making a robust API. A cloud-based API will be developed using Node JS and Express JS to serve data to the client-side applications.

6. MongoDB – mongodb.com

   MongoDB is an open source document storage database sponsored by MongoDB, Inc. It's a highly scalable database primarily used for real-time apps that require data logging and aggregation, which is useful for a home automation app that is trying to monitor power usage and serve multiple devices from the cloud in real-time.

3.2. Constraints

3.2.1. Hardware Constraints

The following sections contain constraints that will be considered during the creation of the hardware components of the project.

3.2.1.1. Wall Outlet

Wall outlets are placed in numerous different locations, often behind furniture, and this leads to limitations in form factor. Size is a very important limitation, and to fit with our goals and objectives we have a constraint on the wall outlet device that it should extend out no further than 1.5 inches from the wall when attached to an existing wall outlet. The wall outlet device must be able to handle the North American standard 120 volt, 60Hz while also accounting for the 2.7 to 5.5 volt operating range of the microcontroller.

3.2.1.2. Wall Switch

The wall switch device, unlike the wall outlet, replaces the currently installed device, so it must account for being set into a wall and mounted. The wall switch microcontroller software and enclosure must account for the device to be operated manually, as well as with the mobile or web application. The design of the device must account for the required operating voltage range of 2.7 to 5.5 volts for the microcontroller.

3.2.1.3. HVAC Controller

The HVAC controller device is the most complex of the devices in the project. The microcontroller software and device enclosure must account for the device to be operated directly as well as operated by the mobile and web applications. The enclosure must also account for the device to be mounted on the wall, but due to the standard location of HVAC controllers being on open walls there are no size constraints of the full device. The HVAC controller must have a display that is accurate to the current conditions of the HVAC system, and the display must update from either manual input, or input from the mobile or web application. The power consumption of the HVAC controller must account for the required operating voltage range of 2.7 to 5.5 volts of the microcontroller, and the operating voltage of the display.

3.2.1.4. Powerlock

The powerlock is unique amongst the devices in this project in that it is not directly mounted on a wall. A major limitation for this device is that, since it is mounted on a door, it needs either proper cabling connected to a power source, or a battery. The power source must account for the operation of the mechanical features that will activate or deactivate the lock when prompted by the online service, and the microcontroller. The powerlock must be able to be operated manually, and the microcontroller software must account for that possibility as well as the enclosure and mechanical components. The microcontroller has a required operating voltage range of 2.7 to 5.5 volts. Another constraint of this device being mounted on a door is that the enclosure needs to account for proper mounting of the entire device, rather than the device being able to attach to a wall fixture.

3.2.2. Software Constraints

The following sections contain constraints relating to the software components of the project. The sections are split between the microcontroller and the web and mobile application. The web/mobile application section also contains all constraints relevant to all web based systems used within the project.

3.2.2.1. Microcontroller Software Constraints

The main constraints for the microcontroller software come from the limitations of working on an embedded system. The Atmel Sam C ARM Cortex ATSAMC21J18A device has limitations in memory, RAM, and CPU that affect any code to be written for it.

The Atmel Sam C ARM Cortex ATSAMC21J18A has a flash memory size of 256KB. This limits the amount of code that can be written for the device and stored on it. 256KB is the largest memory size available for the Sam C ARM Cortex series of microcontrollers. 256KB allows for a comparatively large amount of data to be stored on the microcontroller; however it still has influence on the length of any code to be written for the device.

The Atmel Sam C ARM Cortex ATSAMC21J18A has 32KB of SRAM for programs to us while running, limiting the number of simultaneous processes and variables. 32KB is the largest RAM size available for the Sam C ARM Cortex series of microcontrollers. This largest amount of RAM is desirable for the multithreaded approaches of the microcontroller software design, but it is still limiting in its size.

Although The Atmel Sam C ARM Cortex ATSAMC21J18A supports multithreading, it only has a single core CPU to do so with, which limits the actual implementation of multithreading. The CPU has a speed of 48MHz, and a maximum operating temperature of 85 C. These limit the maximum load that should be placed on the processor by the code.

3.2.2.2. Mobile/Web Application Constraints

The home automation system will be controlled using a custom cloud-hosted API with a user friendly interface rendered on the client side. In order to provide a secure, seamless, and reliable cross device user experience, we will must conform to the following standards:

- Screen Resolution

Screen resolutions play a large factor in the design of user interfaces. For mobile devices, there is only a small area of space to work with. For many mobile devices the resolution is 640px by 360px, although newer devices with larger screens have allowed for a slightly larger resolution. The application must be responsive to different size displays, which means it must scale UI elements like menus, buttons, and forms to be mobile capable.

- Cross-Platform

  Because potential users use a myriad of different devices, we must support both iOS, Android OS, as well as a web-based control panel in order to support all possible use cases.

- Security

  With cloud-based services suffering from constant hacking attempts and other forms of malicious attacks, security is of most importance for a cloud-based home automation system. All servers will be firewalled with the exception of web and ssh ports, and ssh will require key-based authentication rather than passwords. All server communications with devices will be encrypted with TLS. This may prove troubling for devices, since the microcontrollers also need to encrypt and decrypt in real-time, which could use additional processing power and prove difficult to program.

- Cloud Limitations

  Since this is a cloud-based service, the service cannot execute arbitrary commands on devices. Each device must be programmed to accept a signal from the service to execute specific actions. This increases development time since every separate device must be separately programmed to accept these specific signals to perform specific actions.

- Response Times & Reliability

  The service must have some form of redundancy in case of hardware or networking outages, since all clients are dependent upon the cloud-based API this service provides.

4. Project Research

In order to determine what kind of competition exists, how each aspect of our project should perform, and to choose parts that will allow us to accomplish this, we begin by looking at already existing projects and products that are similar to our project. There are many products that perform tasks similar to our project, but

not many are as expansive as ours seeks to be. The different products that we compare will serve as a starting point for our project, and will show how we plan to merge several products into one cohesive Home Automation System.

4.1. Research of Similar/Related Projects and Products

In this day and age, people wish to be able to control more and more aspects of their lives from the palm of their hand. So it is no wonder that smart homes are becoming more and more desired. One product we will look at is the ConnectSense Smart Outlet, which allows the user to remotely monitor appliance usage, turn them on or off, or even schedule times when appliances should be turned on or off. Another product we will look at is the Belkin WeMo Insight Switch, which allows the user to monitor the actual energy usage of their appliances, as well as control whether or not they are turned on or off. Next for similar wall switches, we will look at the GE Z-Wave Wireless Lighting On/Off Switch , which is used to wirelessly control the lights in one's home. We will also look at the Belkin WeMo Light Switch, which can function in tandem with the Belkin WeMo Insight Switch. Next, we will discuss the Kwikset Kevo Smart Lock developed by Unikey, which allows residents to unlock their front door once their smartphone is in range of the receiver, and also allows them to send keys to other users as well. We will also look at the August Smart Lock, a similar product. After, we will look at the Honeywell Lyric Smart WiFi thermostat. This device allows users to control their home's central cooling system via their smart device, and also has built in protocols to help adjust for energy conservation for the user during times such as when they are sleeping or not in their home. We will also look at the Ecobee Smart WiFi thermostat as well, which is very similar to the Honeywell Lyric thermostat. Finally, we will look at two similar previous projects done by Senior Design groups, the Smart Home Energy Monitoring System, and the Smart Home Management System, which both sought to provide users with more control over their household while simultaneously helping them to lower energy consumption.

4.1.1. Wall Outlet

We will begin by doing an in depth analysis of the two smart outlets we have researched, the ConnectSense Smart Outlet, and the Belkin WeMo Insight Switch. We will seek to determine the advantages and disadvantages of each device, and determine the aspects of each device that we wish to maintain in our design and final product. To start, the ConnectSense Smart Outlet works by first plugging it into a standard outlet and then connecting any appliances you wish to monitor to it. The user then can open the application, which allows them to configure the Smart Outlet. The outlet displays key information about each appliance connected to it, such as whether it is on or off, how long it has been on for, etc. as well as recent activity, such as when the appliances were turned off or

on, whether it was manually or through the application. Users can also set timers for when their appliances should be turned on or off, as well as rules that can alert the user when some activity happens. The ConnectSense Smart Outlet also supports voice control using Siri on any compatible Apple device. Some key differences with our planned device and the ConnectSense Smart Outlet are that the ConnectSense Smart Outlet is only compatible with Apple devices, while our application will be able to run on both Android and Apple operating systems, as well as on any personal computer. The ConnectSense Smart Outlet also does not display any information on energy usage whatsoever, which is crucial to our project as we seek to help users lower energy consumption with our product. Finally, their device does not support connections via Bluetooth, only wireless. With our device we plan to support both WiFi and Bluetooth. The specifications for the ConnectSense Smart Outlet follow. With our outlet, we plan to only have one outlet instead of two, so our dimensions should be significantly smaller, hopefully with a height of around 2.5 inches instead of 4.9. Figure 4.1.A shows the specifications for the ConnectSense Smart Outlet.

Input Power	120 VAC, 60 Hz
Max Current	15 Amps
Max Voltage	120 Volts
Max Power	1800 Watts (resistive)
Dimensions	4.9 inches tall x 3.0 inches wide x 1.2 inches thick
Weight	11.2 ounces

Figure 4.1.A - Specifications: ConnectSense Smart Outlet

The Belkin WeMo is another smart outlet that is similar to the ConnectSense outlet, but with the added benefit of allowing users to see their actual energy usage, which is a key aspect to our project. It works very similarly to the ConnectSense outlet, as all you have to do is plug it into any standard outlet and then plug in your appliance. It also has a free application that allows the user to monitor and manage any outlet, allowing them to set schedules for appliances as well as get a reading on how much energy that appliance is using. The user can also see the cost generated by that appliance for the day, and get a monthly estimate for how much that appliance will use. This product has appliances for both Android and Apple devices, which is more in line with or project, although it is still missing a web based application like we plan to have. A very important aspect of the Belkin WeMo is that it is fully modular, meaning that you can add as many smart outlets as you like, as well as other smart products by Belkin, like the WeMo LED Lighting and WeMo Light Switch. We plan to develop an application

that will allow for full modularity as well. As both the ConnectSense and Belkin WeMo have the same specifications when it comes to things like input power and current, we plan to match our design to these specifications as our research shows it to be the de-facto standard for smart outlets. The dimensions for the WeMo more closely match what we plan to do for our design, as it fits and covers only one outlet instead of two. The Belkin WeMo also does not support Bluetooth, a key aspect in our design plan. Specifications are shown in Figure 4.1.B.

Input Power	120 VAC, 60 Hz
Max Current	15 Amps
Max Voltage	120 Volts
Max Power	1800 Watts (resistive)
Dimensions	1.5 inches tall x 2.9 inches wide x 2.9 inches thick
Weight	3.7 ounces

Figure 4.1.B - Specifications: Belkin WeMo Insight Switch

4.1.2. Wall Switch

The next component to our project is the wall switch we plan to design. The first wall switch we will look at is the GE Z-Wave Wireless Lighting On/Off Switch. Installing this wall switch requires a bit more work from the user, as in order for it to operate with your in home lighting you will need to do an in wall installation with hard wired connections. However, once installed the GE Wall Switch allows users to wirelessly control their lighting via their smart phone, tablet or computer. The application allows users to see what lights may be turned on, and wirelessly turn them off and vice versa. The user also has the ability to create custom lighting scenes for different situations, such as having the lights turn off at a certain time in night for sleep or even a scene for when the user is away to have the lights intermittently turn on or off to deter thieves. Instead of using WiFi has the communication protocol, the GE wall switch uses a communication specification called Z-Wave. While this approach does minimize power consumption compared to WiFi and Bluetooth, having a standardized communication protocol is important for our project to allow for easier communication between the devices and our application. The GE wall switch also supports all light bulbs, but assuming that your light socket supports these light bulbs that is to be expected. The dimensions of our device should somewhat closely match the dimension of the GE wall switch. The specifications are shown in Figure 4.1.C.

Next is the Belkin WeMo Light Switch. Installation is similar to the GE wall switch, requiring the user to remove their existing wall switch and wire the connections to this instead. It shares many key features with the GE wall switch, like an application that allows the users to see what lights are currently turned on, wirelessly turn them on or off, and set schedules for when the lights should be turned on or off. One unique feature that the WeMo switch has that the GE switch doesn't is that the user can enter their city on the application which will set the user's porch lights to turn on as the sun sets and to turn back on again once the sun rises. The WeMo light switch also operates on WiFi, which is what we plan for our wall switch to use. The WeMo does not however support a web based application as the GE wall switch does. Neither product supports Bluetooth communication either. As was the case with the smart outlet, both products have the same electric specifications. As such, we will design our wall switch around these specifications. We plan to have the dimensions for our wall switch be somewhere between the two existing wall switches discussed. The specifications are shown in Figure 4.1.D.

Max Current	15 Amps
Max Voltage	120 Volts
Max Power	1800 Watts (resistive)
Maximum Incandescent Load	600 Watts
Dimensions	8.2 inches tall x 7.3 inches wide x 2.3 inches thick
Weight	8 ounces

Figure 4.1.C - Specifications: GE Z-Wave Wireless Lighting On/Off Switch

Max Current	15 Amps
Max Voltage	120 Volts
Max Power	1800 Watts (resistive)
Maximum Incandescent Load	600 Watts
Dimensions	5.1 inches tall x 3.3 inches wide x 2.1 inches thick
Weight	4.8 ounces

Figure 4.1.D - Specifications: Belkin WeMo Light Switch

4.1.3. Powerlock

The smart powerlock that we will analyze first is the Kwikset Kevo Smart Lock. Installation is somewhat simple, only requiring a screwdriver and four AA batteries. After installing, the product comes with a mobile application download that provides the user with their eKeys, which are used to unlock the door. Whenever a user comes into close proximity with the door, if they are in possession of either the included FOB or have the application downloaded and are in possession of an eKey, the door will unlock upon touching the lock. Through the application, the user can also share eKeys with other owners of the application to grant them access. The application is compatible with both Apple and select Android devices. One major difference between our intended project design and the Kevo smart lock is that the Kevo does not support remote locking and unlocking, as the only way for the door to unlock is through someone with a valid eKey or FOB to touch the lock when in proximity, while we had planned for users to be able to wirelessly unlock their door through our application. The dimensions for this lock are also much larger than we intend for our lock to be. Specifications are shown in Figure 4.1.E.

Power Source	4 AA Batteries
Dimensions (interior)	7.4 inches tall x 3.25 inches wide x 1.75 inches thick
Dimensions (exterior)	2.75 inches tall x 2.75 inches wide x 1.2 inches thick
Lock Type	Single cylinder deadbolt

Figure 4.1.E - Specifications: Kwikset Kevo Smart Lock

Power Source	4 AA Batteries
Dimensions	3.3 inches tall x 3.3 inches wide x 2.2 inches thick
Lock Type	Single cylinder deadbolt
Weight	16 ounces

Figure 4.1.F - Specifications: August Smart Lock

The next powerlock we will look at is the August Smart Lock. Installation is similar to the Kevo smart lock, not requiring any tools aside from a screwdriver. After installation, the user can download the application to begin configuring their

settings. The application allows the user to remotely lock and unlock their door via Bluetooth. The application also allows the user to actively track who enters and exits your home at different times of the day. Since each user has a unique key, you will also know who is opening the door and when. The lock uses the user's location to determine whether or not you have left home, and will automatically unlock when you approach the door. Our implementation of the powerlock will be much less dynamic, simply allowing the user to remotely lock and unlock the door via the application. Our reasoning for this is that many users state that because of the application running in the background of their smartphone constantly, they are finding their phone batteries dying more quickly. The dimensions for this lock more closely match what we plan for our design to be. Figure 4.1.F shows the specifications.

4.1.4. HVAC Controller

The final component of our project is the HVAC Controller. The first one we will look at is the Honeywell Lyric thermostat. Installation for this component will be the most complex of all four, as it requires wiring to your HVAC system, and different systems will have different wiring. However, once connected, the Lyric is an extremely user friendly thermostat. The Lyric does allow the user to control via application when the system is turned on/off and allows for scheduling. The Lyric also can use the user's smartphone location to determine to adjust the temperature for comfort while the user is home, and for decreasing energy consumption while the user is away. The Lyric also considers both inside and outside temperatures and humidity, so that the user is more comfortable in their home. The thermostat comes with a built in LCD display to allow users without a smartphone or tablet to still control their system. This thermostat does not support a web application, which is a key difference from our planned design, nor does it support Bluetooth communication. The Lyric also does not actually show the user how much energy their system may be consuming, which is a major aspect of our project. The specifications are shown in Figure 4.1.G.

Display	Digital
Dimensions	7 inches tall x 7.2 inches wide x 4 inches thick
Weight	22.4 ounces

Figure 4.1.G - Specifications: Honeywell Lyric Thermostat

The next HVAC controller we will look at is the Ecobee Smart WiFi Thermostat. Installation for the Ecobee thermostat is similar to that of the Honeywell Lyric. Once installed, the application allows the user to monitor and control the temperature from their smartphone, tablet, or computer. It also can adjust to

either heat or cool depending on the weather outside to assure the user is comfortable. A major difference between this HVAC controller and the Honeywell Lyric, as well as our proposed design, is that the ecobee thermostat comes with an included remote sensor that can detect temperatures and motion in separate rooms to determine which rooms are occupied so that you can save energy when appropriate. The built in LCD display allows users to control the temperature from the thermostat as intuitively as the application. Figure 4.1.H shows specifications for the Ecobee Smart Wifi Thermostat.

Power Consumption	< 3.5 Watts
Temperature Sensitivity	+/- 1°F
Humidity Sensitivity	+/- 5% RH
Dimensions	3.93 inches tall x 3.93 inches wide x 0.9 inches thick

Figure 4.1.H - Specifications: Ecobee Smart WiFi Thermostat

4.1.5. Related Projects

One previous Senior Design group's project was called the Smart Home Energy Management System. Their project motivation was very similar to ours, but they chose to focus more singularly on a product that could be compared to our smart outlet. Their product is a number of small devices that can be used to monitor and track the power usage of your appliances. The devices would all then send information to a centralized HUB, which would analyze all of this information, calculate the energy used by each appliance, and display it to the user on an LCD screen. These devices would also be able to remotely disconnect your appliances from power if you chose to do so, via a Bluetooth mesh network. For our project, we opted to have an application instead of a centralized HUB, as it is more economic for users, since they can run the app on their smartphone, tablet, and/or computer. This application also allows for a more modular product, as adding additional modules can be made very simple. We also expanded much more on the energy conservation idea that they had, including both a smart wall switch for light fixtures, and an HVAC controller to regulate the air conditioning. These additions make for a much more robust energy conservation system. Our project also includes WiFi in addition to Bluetooth, which makes our web application possible.

Another Senior Design group's project was the Smart Home Management System, with the same motivation as ours, to monitor and reduce the user's energy consumption. They use multiple modules that will monitor current, as well as have the ability halt or allow power to flow to the attached device. They then

have a central unit which will have a touch screen display, which will be one way in which the user can interface with the system. This central unit would allow the user to view information on all the modules as well as turn on or off the appliances connected to them. Each module will send information about the appliance connected, such as whether or not it is turned on, and how much power it may be drawing. They also planned on having this central unit connected to the internet via an Ethernet cable, allowing the unit to be controlled remotely via the internet. They then planned on having sensors stationed that sense the presence of someone in some area of the household. Having a central hub with an LCD display that can communicate to a computer seems extraneous, which is why we forgo that for an application instead. They also do not allow for the central hub to communicate with a computer wirelessly, as their hub requires an Ethernet connection in order to communicate with computers. Once again this group also only focused on simpler appliances for their energy conservation tasks, while we seek to conserve in multiple aspects of the home, like lighting and cooling. We also choose not to include sensors for our design, as it adds a level of difficulty to our project that would not have an effect worth the effort, since we allow for the users to schedule when appliances should be turned on or off.

4.2. Relevant Technology

After analyzing a number of similar products and projects, and comparing their advantages and disadvantages, we begin to determine exactly what components and technologies we will need in order for this project to be a success. We know that for the smart outlet, we will need a few components. We will need AC to DC converters in order to use the AC current coming in from the outlets. We will need a power sensing IC in order to measure the current flowing out to the appliance, a WiFi and Bluetooth module in order for the outlet to communicate with the application, a relay in order to stop the current or allow the current to flow, and a microcontroller that will allow all of the components to communicate with each other. For the wall switch, the needs will be basically the same, as it will have a similar functionality to the smart outlet. For the powerlock, we will need a WiFi and Bluetooth module, a simple microcontroller, a relay, and a power source. Then, for the HVAC controller, we will need a microcontroller, WiFi and Bluetooth module, a relay, and a power meter to measure how much energy the HVAC system is using.

4.2.1. Power Monitoring

One of the most important aspects for our project is the ability to monitor exactly how much energy any particular appliance is using at a given time. For the smart outlet, we have chosen to go with a Power Sensing IC in order to determine the voltage and current being drawn from the appliances. As we are trying to help

users lower their energy consumption, we would like for the IC to be extremely accurate in its measurements, with an error rate of at least < 0.5%. The IC would then have to be able to send the measured data to the microcontroller, which would then process that data, and finally send it to the application via the WiFi and Bluetooth module to be displayed on the application. The power usage will be displayed in different time variations, such as amount of usage per hour, or usage per day.

4.2.2. Wireless Communication

Another important aspect of our project is the ability for all the separate components to be able to communicate wirelessly to any device running the application. We plan on using both WiFi and Bluetooth in order for communication. WiFi allows users to communicate with the system even when they are at extended ranges, such as work or school, while Bluetooth allows for low power consumption at closer ranges. WiFi also allows us to build a web application for users to access on their computers without the need for a Bluetooth dongle. A low power alternative could be Z-wave, which was designed with home automation in mind. However, it is a younger technology with not much open source development support and this would add another layer of difficulty to our project, so Bluetooth is the preferred option.

4.2.3. Power Supplies

All aspects of our project will need some kind of power supply in order to function correctly. For the smart outlet, the power supply will be the electricity coming from the outlet they are plugged into, after being transformed and converted with our AC to DC converters. The wall switches will use the neutral wire as a power supply. For the electric strike, we will use a simple AC power supply. For the HVAC controller, the power supply will be the wiring for the existing thermostat. The power supplies will have to be enough to drive all components for each product. This should not be a problem in for any product though, as the highest energy draw will be the microcontroller in most cases.

4.2.4. Relays

Yet another important aspect of our project is for the user to have the ability to turn on and off any appliance, as well as schedule when an appliance should turn on and off. Whether this is down wirelessly through the application or manually with a button on either the smart outlet or HVAC controller, these features will be accomplished using a relay. Since both products will be able to wirelessly communicate with the application, a schedule can be set up with the application, and when the time comes for the appliance to be either turned on or off, a signal

will be sent to the proper component and the relay will either halt the flow of energy or allow it to flow once more.

4.2.5. Electric Strike

In order for the powerlock to work, we will need to incorporate an electric strike as a component. An electric strike is a simple component that can either lock or unlock when a current is sent to it depending on its configuration. This will be used in tandem with a power supply, a WiFi and Bluetooth module, a relay, and some simple microcontroller in order to receive a signal to either lock or unlock, then have the relay allow for power to be sent from the power supply to the strike to lock or unlock the door.

4.2.6. LCD Displays

For our HVAC controller, the user will be able to monitor and configure the settings via the application, or the LCD display. The interface should display the current temperatures, and should update accordingly based on whatever options the user may select via the buttons below. It should also be able to display whether or not the HVAC system is operating at this time, as well as the power consumption of the system, and the average cost of the system. The LCD Display should also be able to have some kind of sleep function so that it does not display anything while it is not currently being used.

4.2.7. Temperature Sensor

In order for the HVAC controller to function as specified, we will need a temperature sensor. This sensor should be able to send the temperature to the microcontroller, which processes it and sends it to both the application and the LCD display. It is important that the temperature sensor is very accurate, so that the HVAC controller can properly determine when the user should turn on or off their system so that they can save on energy consumption. Ideally, the temperature sensor would have an accuracy of +/- 1°F, to match the accuracy of the Ecobee Smart WiFi thermostat.

4.2.8. Microcontroller

All these previously stated components are equally important to the proper function of our project; however they would not be able to work together at all if we did not have some type of microcontroller. The microcontroller will be the component that allows for the other components to communicate with each other, as well as process the data that will be sent via the WiFi and Bluetooth module to the application. The microcontroller will have to be able to do all of this simultaneously to allow for timely updates to the user.

5. Project Design

5.1. Hardware Design Plan

The following sections will discuss the specific hardware components needed to run and implement the various aspects of the home automation system. This includes the designs for the wall outlet, HVAC controller, powerlock, and wall switch. The various advantages and disadvantages will be discussed for each component as well as comparisons to other existing modules available. Constraints and Standards were taken into consideration when building the components, making sure the correct AC and DC voltages were according to IEEE standards.

5.1.1. WiFi/Bluetooth Module

For a good, well established communication between the home automation hardware and the smartphone app, a choice of wireless communication needed to be made. There are many out there with certain advantages and disadvantages. For the home automation system, the wireless communication module must be able to have a constant connection while not using much power consumption when not transmitting. Because this specific home automation system is for an average sized room, the module doesn't need to have a wide range of access. There is also only need for one antenna in the module since transmission/receiving is only going to one place, which in this case is the central hub.

5.1.1.1. WL18xxMOD 8 Single-Band Combo Module

In a new line of products from Texas Instruments for combo WiFi/Bluetooth modules, the WL18xxMOD WiLink 8 Single-Band Combo Module incorporates all of the UART wireless communication needed for instant transmission and receiving while consuming low amounts of power.Having both WiFi and Bluetooth allows for a more stable connection between the central hub and the smartphone. A constantly on Bluetooth pairing to the home automation components allows for fast and easier access as well as easier connect and disconnect to the central hub. Figure 5.1.A shows the functional block diagram of the module. The basic input for WLAN and Bluetooth are shown as well as the process the inputs go into. The pinout schematic is shown in Figure 5.1.B, where the inputs are shown being wirelessly received and transmitted using UART communication. Most of the pins are grounded and unused. The important parts of the schematic include pins 50-53 (connecting to host HCI interface), pins 56-60 (connecting to host Bluetooth PSM Bus), and pins 6-14 (connecting to host

SDIO interface). All of the pins stated are needed for a transmit/receive I/O between the digital and bluetooth interfaces.

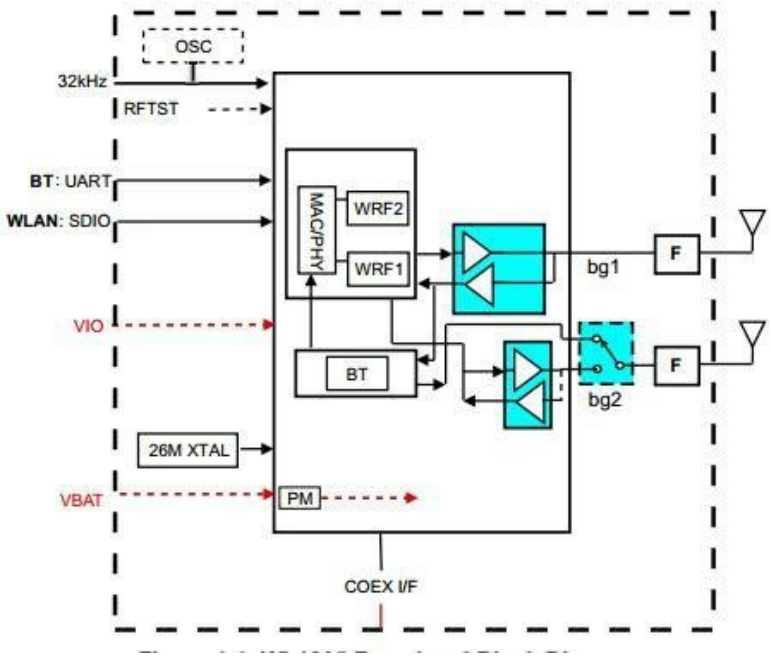


Figure 5.1.A - WL18xxMOD Functional Block Diagram
(permission granted from Texas Instruments)

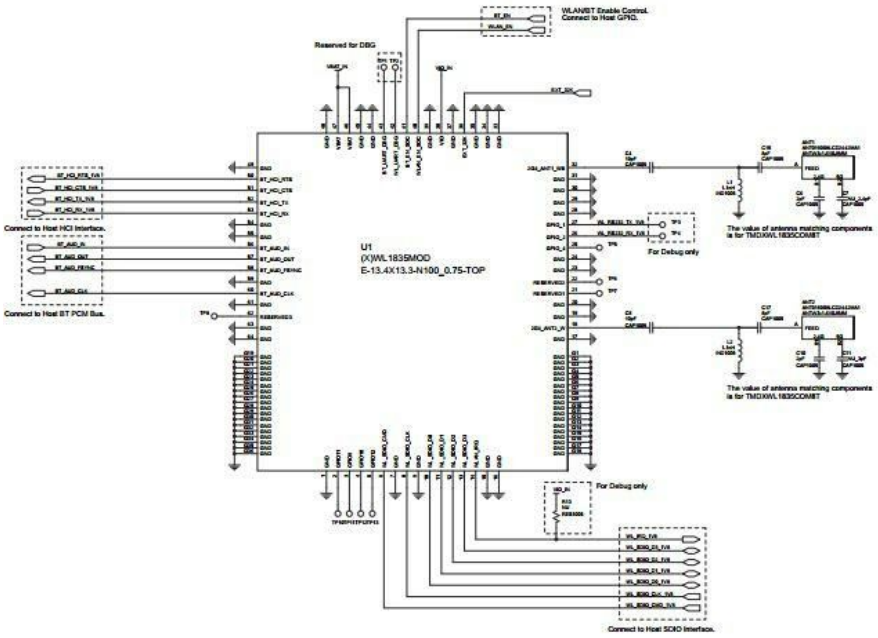


Figure 5.1.B - WL18xxMOD Pinout Schematic
(permission granted from Texas instruments)

5.1.1.1.1. WLAN and Bluetooth

The module offers integrated 2.4 GHz power amplifiers. This assures that the module is able to fulfill all TX/RX signals without any need for other components or modules.This radio also includes a low noise amplifier, which is important if the signal received is very weak due to the smartphone transmitting at a large distance or away from the main room. The WLAN also contains a small embedded ARM CPU. This CPU contains encryption and decryption for the module, allowing for a WiFi with special encryption (either WPA or WPA2) to be fully compatible with the WLAN part of the module.

5.1.1.1.2. Power Requirement and Dimensions

The module occupies a very small portion of the breadboard, only 13.3 × 13.4 × 2 mm in size, allowing for more space for other components, such as the relay and power monitoring chip. This allows for use of a smaller PCB for all of the components, which is vital for the requirement of small and non-invasive components that won't be too noticeable in a standard sized room.

According to the Texas Instruments datasheet for the WiFi/Bluetooth module, there is a minimum startup voltage of 4.8 V. Each pin on the module requires a 1.8V start up voltage. This is done by connecting three internal DC/DC voltage converters already included in the module, allowing for a controlled voltage to access the pins. The functional block diagram can be seen below, showing the top level design of the DC/DC power supplies. In order to obtain an efficient power management system, VBAT and VIO are off while no signals are sent to the module, aside from the low power timer that acts like an LPM4 Timer. WL_EN is used to power-up the module and the DC/DC internal converter, clocks, and LDO's are slowly enabled and stabilized. These processes can be shown in Figure 5.1.C

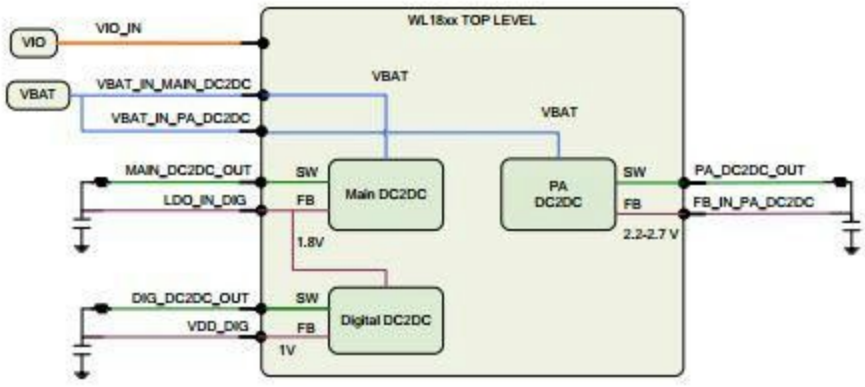


Figure 5.1.C - DC/DC Power Supply Functional Block Diagram

5.1.1.1.3. Comparison to Other Modules

This module was chosen specifically because of the utility and ease of having both Bluetooth and WiFi inside a signal small module that operates on low power. There are many modules out there now currently that use either WiFi or Bluetooth; there are not many combo modules that utilize both. A few have been found and, in comparison to the WL18xxMOD WiLink Low Energy Combo Module, they either lack in performance or do not fit the ideal specifications of the overall final PCB layout. In total there were three Wifi/Bluetooth combo modules that were considered. The price, range, dimensions, and other features of the module were all considered. Figure 5.1.D shows the comparison with each module.

	Price	Dimensions	Range	Startup/Signaling Voltage I/O	Special Features
LSR TiWi-BLE Bluetooth Classic & Bluetooth Low Energy + WiFi 802.11 b/g/n Module	$26.77	18 x 13 x 1.9 mm	2.4 GHz	1.8 / 4.8 VDC	Full support for ANT Reverse polarity antenna
TI WL18xxMOD WiLink™ 8 Single-Band Combo Module	$23.33	13.3 × 13.4 × 2 mm	2.4 GHz	1.8 / 4.8 VDC	Integrates RF, PA, Power Management, etc. Used specifically for Automation Systems
Summit SDC-SSD40NBT	N/A	15 x 15 x 2 mm	2.4 GHz	1.8 / 3.3 VDC	Collision Avoidance

Figure 5.1.D - WiFi/Bluetooth Module Comparison Table

Summit SDC-SSD40NBT

The SDC-SSD40NBT was not only too big for the size requirement, but this specific module has been discontinued and is no longer sold. The module came out around 2013 and it was a first of its kind. Not many modules have both WiFi and Bluetooth in a single package that was also power efficient as well as cost efficient. The physical interface was a typical quad flat pack with no leads. However, the most standout feature would have to be the carrier sense multiple

access with collision avoidance. This allows for multiple signals coming in and out at the same time without interference or distortion caused by signals mixing with each other. The input voltage was only 3.3 VDC and contained up to 5GHz frequency bands, which was unnecessary for the home automation system, but is a very nice feature to have regardless. Back then this would have been a great module to use. However, there was no method of sampling to purchasing this module.

LSR TiWi-BLE Bluetooth Classic & Bluetooth Low Energy + WiFi 802.11 b/g/n Module

With the Summit SDC-SSD40NBT available for consideration, the only two options were the TI WL18xxMOD WiLink™ 8 Single-Band Combo Module and the LSR TiWi-BLE Bluetooth Classic & Bluetooth Low Energy + WiFi 802.11 b/g/n Module. Upon inspection, these modules are seemingly similar and identical. They both offer the same amount of practical application, close dimension specs, superb wi-fi connectivity as well as bluetooth connectivity, and most importantly low power operations for standby mode. However, the LSR module also utilizes the ANT wireless sensor network technology, which is a wireless communications protocol stack used primarily for sensors. The home automation system does not need this type for wireless technology and would not be used. The dimensions and cost also played a minor role in choosing the module. the LSR module was minutely bigger in length, which could be a problem in fitting the components unto the breadboard. The cost, although insignificant, drove the decision to use the TI Combo Module. This is why the LSR module was not considered in the design of the home automation components.

5.1.2. Microcontroller Module

For most of the components of the home automation system, an MCU is needed to operate the component. All of the other modules in the PCB (relay, power sensing IC, bluetooth module) will be connected to this MCU to operate these modules. Inside the MCU will be a pre-made program used to interface with the smartphone and the central hub. The MCU also needs to be able to easily migrate with other peripheral modules and store a considerable amount of data using flash as well as SRAM.

Figure 5.1.E is the basic layout that will be used for all microcontrollers used in each of the devices. The microcontroller will receive data from the device as to its current state, and send that data to the wifi module to be sent to the cloud server, and in turn also receives data from the cloud, via the wifi module, as to which state the processor will tell the device to be in. The microcontroller will also receive input from mobile devices via the bluetooth module, however this will only be input and will only be used to configure the wifi module for connecting to

the desired wifi network and allow connection to a private or protected wifi network. The processor will convert the received data to the correct format for output, and send the output to the correct destination, while still being able to receive new data. Efficiency and accuracy of the process will be the most valued metrics, there should be very little if any delay between a command being entered from the app/cloud and the microcontroller, less than 1 second of delay is the desired goal.

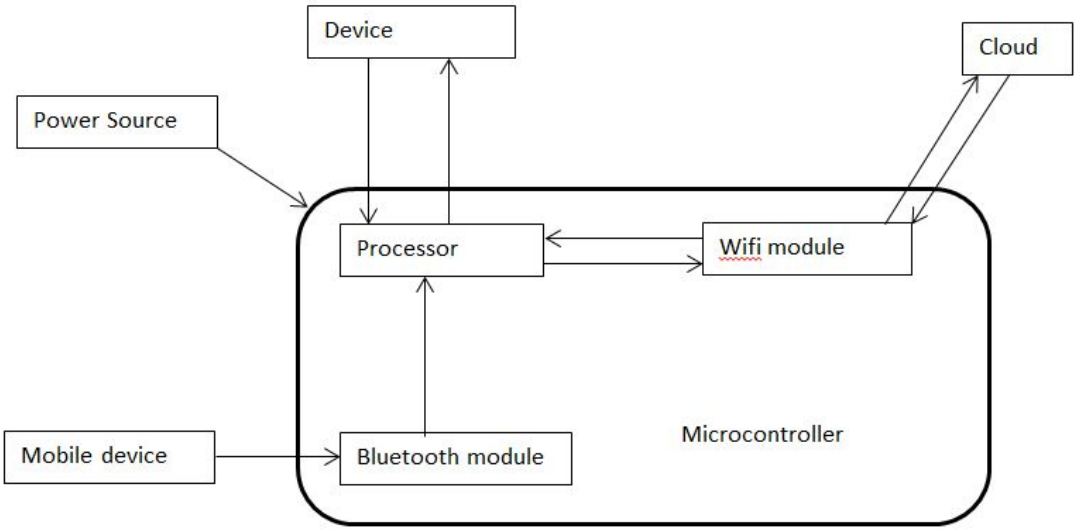


Figure 5.1.E - Microcontroller Layout

There are many MCU's out there that can perform all of these actions and more. However, for the purpose of this project, the MCU will most importantly need to be universal with all of the components of the home automation system. This means that the MCU will need to be practical for the wall outlet, wall switch, powerlock, and HVAC Controller and be able to accomplish of the needed duties without wasting unnecessary power or being too big/small for the PCB. With all of this in mind, typical IEEE standards and constraints need to be followed when interfacing the MCU with the other modules on the PCB.

5.1.2.1. Atmel SAM C ARM Cortex MCU

In order to have an ideal home automation system, it is important to have an MCU that is pre-optimized for tasks brought upon by home automated peripherals. The Atmel SAM C ARM Cortex MCU is the latest in MCU's that are designed specifically for industrial automation, appliances, and other 5V applications related to home automation. One of the main features of this MCU that stand out from other MCU's is the support for capacitive touch button, slider and wheel user interfaces. This allows the use for a smartphone app that can utilize these capabilities for a more user-friendly interface that can be used by anyone.

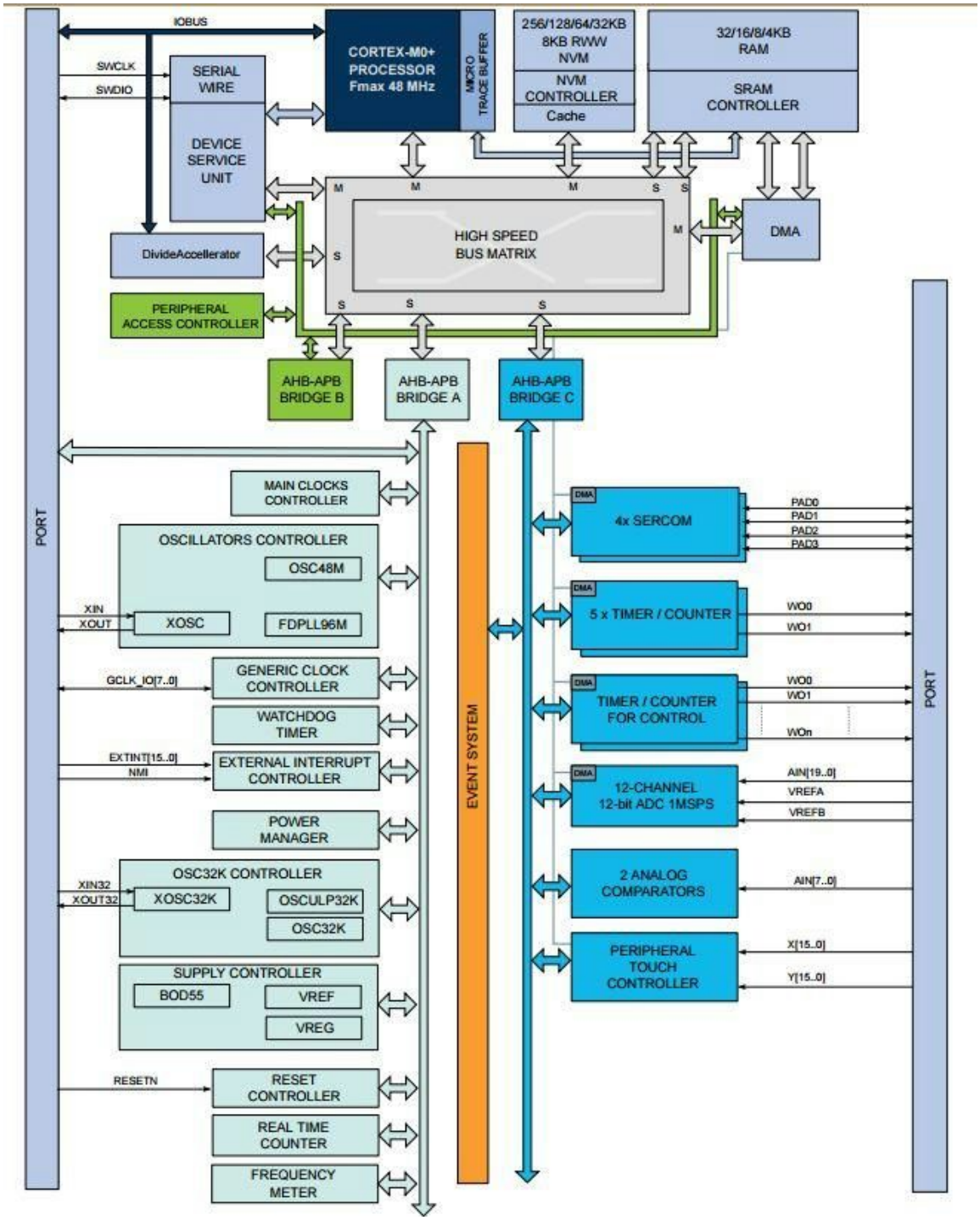


Figure 5.1.F - Atmel SAM C Arm Cortex MCU Functional Block Diagram
(permission granted from Atmel)

The MCU system contains both internal and external clock options with a 48 MHz to 96MHz Fractional Digital Phase Locked Loop. This PLL has an adjustable proportional integral controller that allows for increase/decrease to the frequency.

The reference clock can go from 32 kHZ to 2 MHz that can be chosen from multiple sources. This is important for the output of the signal that will be sent to the WiFi/Bluetooth module and relay.

In regards to the processor, with a peak speed of 48MHz, the ARM Cortex is able to process and complete tasks at a very fast rate. The processor contains a single-cycle hardware multiplier, which means that completing a binary operation only takes about 10 nanoseconds. Certain operations such as division use early termination in order to minimize cycles. The processor utilizes both Little-Endian and Big-Endian for storing into memory, although Little-Endian is mostly used in this case. There are 32 interrupts for stacked processing, which allows for a number of operations to be completed via smartphone. This is unnecessary for the purpose of this project because the workload of the MCU will be minimal. Figure 5.1.F shows a functional block diagram of the MCU. The power sources and timers can be shown in the figure, as well as the necessary comparators and controllers for the system to attain the high speed multiplier.

5.1.2.1.1. Power Requirement

The MCU contains several power supply pins that operate on different voltages. The functional block diagram for the power supply is shown in the figure below.

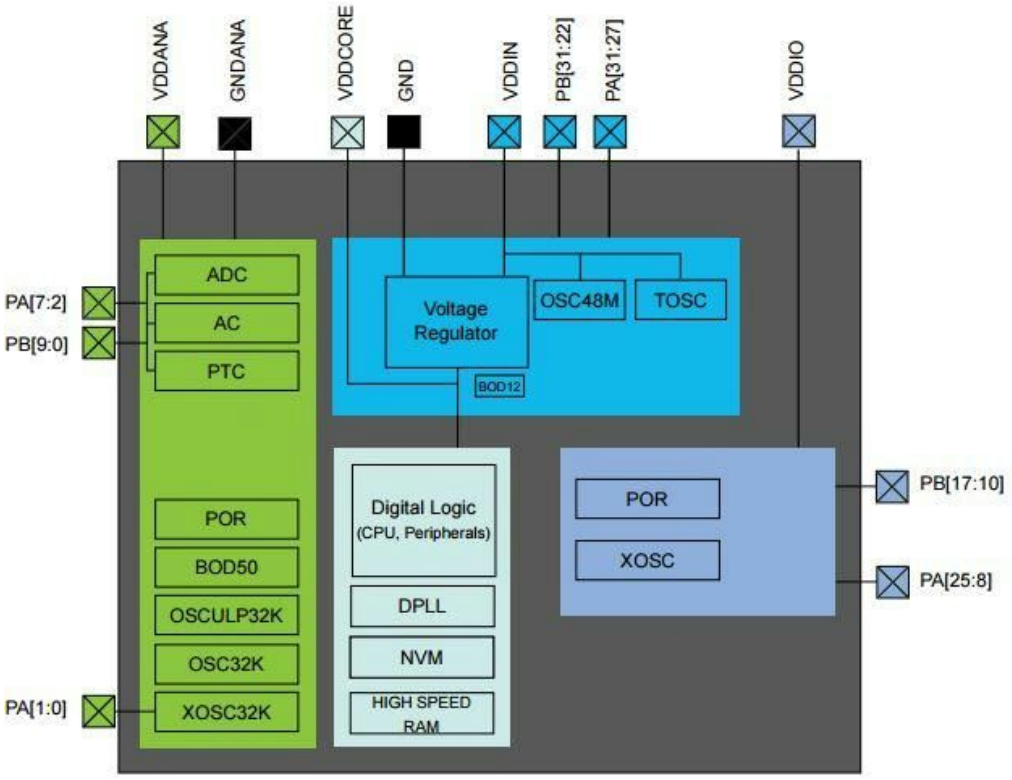


Figure 5.1.G - Atmel SAM C Arm Cortex MCU Power Supply Pinout Schematic (permission granted from Atmel)

Operating voltages are 2.7 V to 5.5 V. The main supply could either be a single supply or a dual supply that is taken from VDDIN. This voltage is applied to 4 pins to power the core, memory, peripherals, oscillators, etc. This is done with a built in voltage regulator that is on standby low power mode until CPU and peripherals start running. Failsafe features are also included, such as power-on reset and brown-out detection. This assures that, in case of a power outage or error, there is no damage to the MCU. The pinouts and components can be seen in Figure 5.1.G.

5.1.2.1.2. Communication with Software

The Atmel Sam C ARM Cortex microcontroller does not have built in WiFi communication capabilities. To be able to connect to WiFi a separate module is needed. To do this we are using the TI WiLink 8 series WL1835MOD module. This module will be connected to the microcontroller directly and the software will be used to configure and operate the module. The configuration data will be received via a Bluetooth connection between a Bluetooth module attached to the microcontroller, and a Bluetooth enabled input device, most likely a smartphone or tablet computer.

We plan to use an encrypted TLS connection to send and receive data between the microcontroller and server. In order to do this, the TLS mbed open source SSL library 2.2.0 will be used. TLS mbed open source SSL library 2.2.0 is designed to work with ARM Cortex microcontrollers, and is purposefully very small in size so that it can be used directly from embedded processors. This library is TLS 1.2 compliant and contains encryption algorithms suitable for embedded systems.

Data is sent and received from the server every cycle of the software to maintain the connection. Data is sent to the server first and then received from the server in every instance of the communication. Data will be encrypted by the TLS mbed library before being sent, and decrypted once received, with no steps between encryption and sending or between receiving and decryption.

5.1.2.1.3. Comparison to Other Modules

When beginning to search for compatible MCU's for home automation, the Atmel SAM C ARM Cortex MCU was the first MCU to pop up. Because this specific MCU contained everything needed and built almost exclusively for home automation, this was the obvious MCU of choice for the project. No other MCU's were considered and no other MCU's are needed since the SAM C ARM Cortex MCU is universal with all of the home automation's components.

5.1.3. Power/Energy Management

Because one of the main purposes of the home automation project is to better conserve energy and power by measuring them across certain intervals, every component that is controlled under the smartphone app needs to incorporate some sort of power measuring tool. The measurement then needs to be stored and sent to the smartphone after a certain interval i.e. at the end of the week or end of the month. In order to have a constant measurement of power, a measurement of current and voltage need to be obtained first.

As far as the hardware module is concerned, the requirements for a quality power sensor needs to be optimal for saving space as well as simple connection and implementation. There needs to be a considerable amount of storage, preferably flash storage, that can be configured, processed, formatted, and interfaced onto the smartphone.

5.1.3.1. 78M6613 Single-Phase AC Power-Measurement IC

The Maxim Integrated 78M6613 Single-Phase AC Power-Measurement IC is an integrated circuit that implements single-phase AC power measurement and computes using a 32-bit compute engine and a 22-bit delta-sigma ADC. This component is used specifically in power supplies and appliances that can be wirelessly controlled. There are four analog inputs that interface to voltage and current sensors. Also included is an integrated microprocessor core that post-processes any input from the voltage and current sensors. A temperature sensor is also included in the IC, but is not necessary for the purposes of this project. 32 KB of flash memory is included in the package, which is plenty of storage for data receiving, processing, and transmission.

As stated in ‘Power/Energy Management’, the power sensor needed to be very small and discreet so there would be more space on the PCB. The 78M6613 Single-Phase AC Power-Measurement IC is 5 x 5 mm, which is incredibly small and compact. Included in the small form factor are 32 pins for a variety of input and output options. For the purposes of this project, most of the pins will not be used and the important pins, such as voltage sensor and current sensor input, will be used. Figure 5.1.H is a functional block diagram of the power measurement IC, which includes pinouts and internal converters and communication peripherals. There are many components inside the IC that are important for power operation, such as the 32 bit compute engine and the mini MPU. The steps on how the power is achieved can be seen under the “Power Management Operation” section.

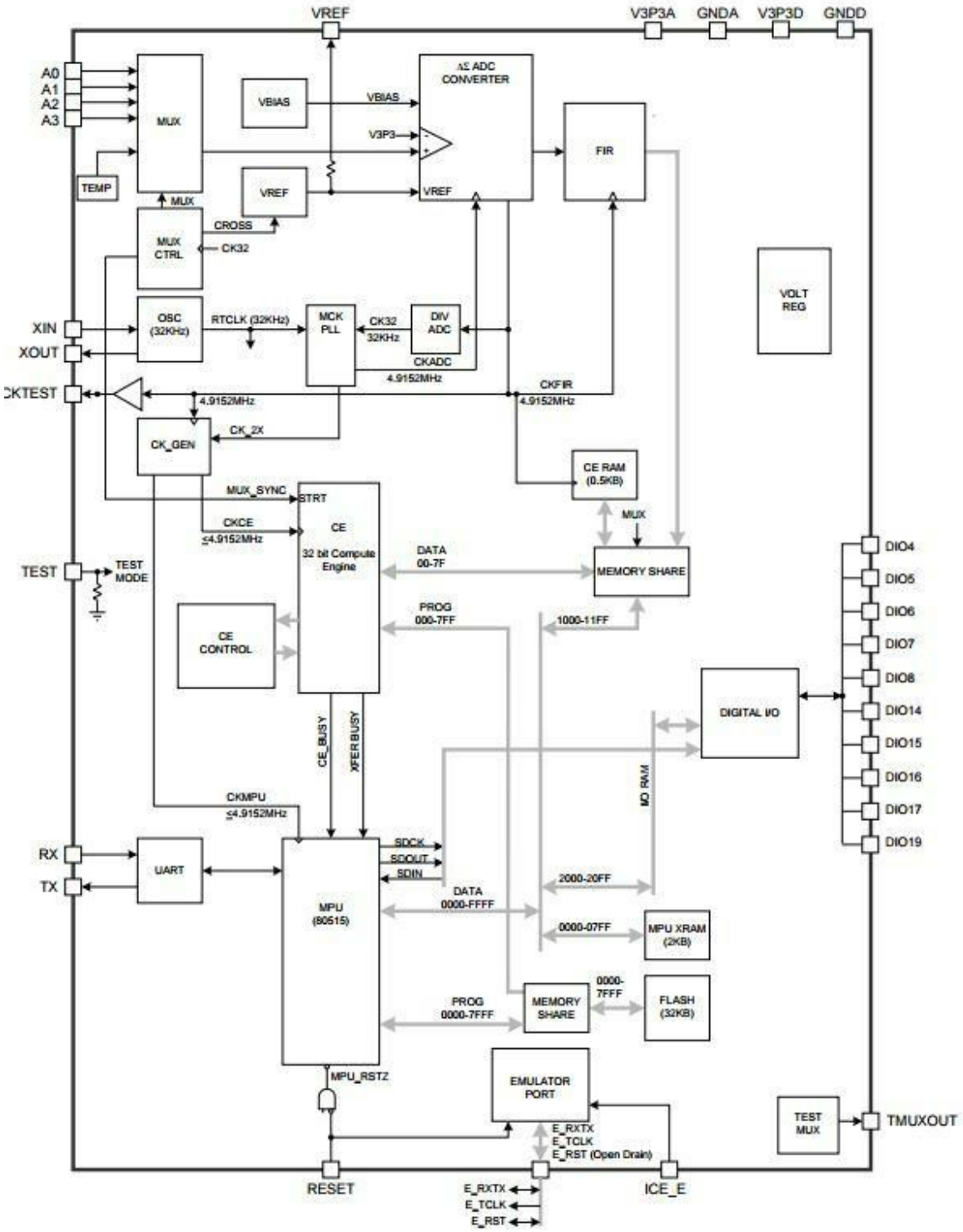


Figure 5.1.H - 78M6613 Single-Phase AC Power-Management IC Functional Block Diagram

5.1.3.1.1. Power Management Operation

Power output for the 78M6613 is given by the following expression:

$$E = \int_0^t V(t)I(t)dt$$

At constant intervals, the sensor captures constant changes in voltage, current, phase, and harmonic current. In order to provide an accurate reading, the 78M6613 processes sample of current and voltage through an ADC under a frequency ω, which is constant. These samples are then multiplied by the sampling period to get a stable voltage and current, which is then multiplied and integrated as the above expressions shows and a momentary energy is yielded. These energies can be added up to obtain an accumulated energy, which is then transmitted from the 78M6613 to the output. Figure 5.1.I is a graph showing the possible trends in voltage, current, momentary and accumulated energy.

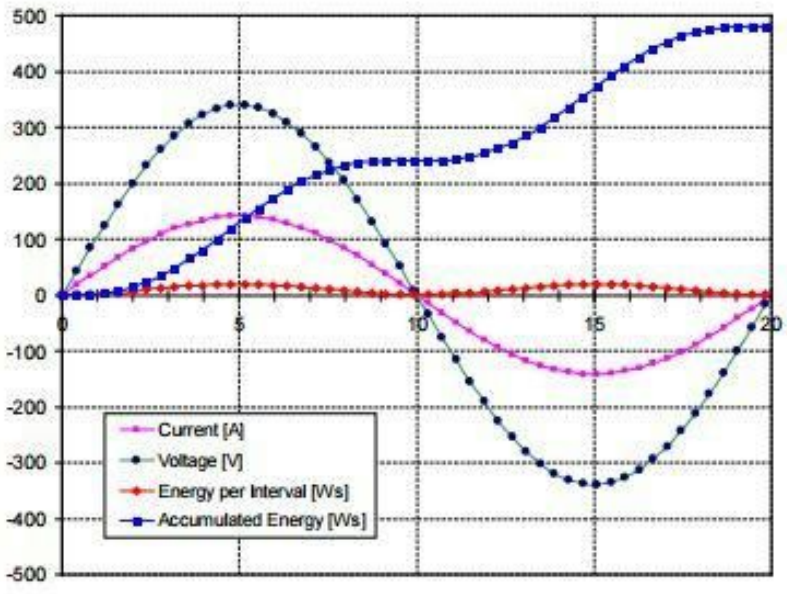


Figure 5.1.I - 78M6613 Single-Phase AC Power-Management IC Power Trends

5.1.4. Wall Outlet

One of the main components of this home automation system is the smart wall outlet. Many smart wall outlets have been made, as shown in the Related Projects section. One of the main influences of the design of the smart wall outlet was the WeMo Insight Switch. Its small form factor and compact design made it stand out from the other bulky smart wall outlets that other competitors offer. However, the WeMo Insight Switch only accounted for one electrical appliance per module. Having dual wall outlet that can both be programmable while containing a slim and discreet profile was the goal in designing our wall outlet.

PLACEHOLDER FOR FINISHED WALL OUTLET DESIGN

Figure 5.1.J - Wall Outlet Final Design

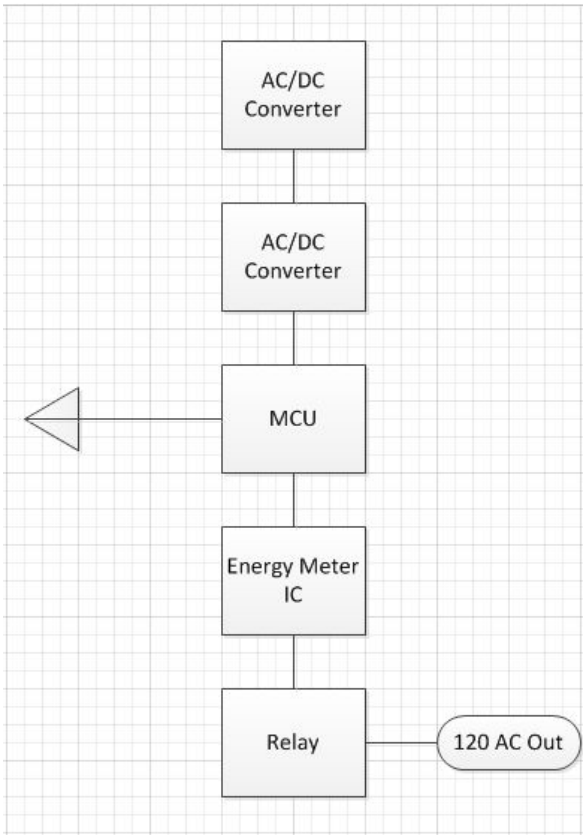


Figure 5.1.K - Wall Outlet Hardware Block Diagram

Figure 5.1.J shows the final design of the smart wall outlet. The design requires a minimum voltage of 7.8 V and must be able to handle temperatures up to 100°F. Although temperatures should realistically never go up to this much, it is

important to ensure the casing and modules inside should not malfunction under extremely high temperatures. The same is asserted for temperatures going as low as 40°F, although the casing as well as the modules inside should not be affected as easily as under hot temperatures.

An input of 120 V goes through an AC/DC converter (and the resulting DC current is degraded to the exact voltage needed to activate the microcontroller (the exact model is still being picked out). The microcontroller, at rest, would not output an AC voltage unless a wireless signal from the app on the smartphone or the tablet turns on the microcontroller. If so then a 120V AC is output, thus turning on the wall outlet and the energy meter IC records this continuously until the state is changed. Figure 5.1.K. is a basic visual hardware block diagram of the above process.

5.1.4.1. Dimensions

Using the Eagle Software, a PCB design was created that contains all of the necessary circuitry needed to power and maintain the wall outlet. It is important that the PCB is small enough to be packaged inside a small casing as desired in the form factor of the wall outlet. The plug is a standard plug of dimensions 1.34375 x 1.125 inches and the size of the entire casing is 4.5 x 2.75 inches. The reasoning for these dimensions was to create an almost inconspicuous piece of hardware that melds into an existing wall outlet without any bulky pieces sticking out. For this reason the PCB will be using a single sided copper layer of size 4 x 2 inches. All of the resistor and capacitor values can be seen in the PCB design below in Figure 5.1.L.

Figure 5.1.L - Wall Outlet PCB Design

Figure 5.1.M displays a table of components that are on the PCB along with the purpose of that component. Also given below in Figure 5.1.N is a table of the electrical characteristics of the wall outlet as well as general info regarding the wall outlet, such as operating temperatures, operating voltages, dropout voltage, etc.

Placeholder for Table of PCB Components

Figure 5.1.M - Wall Outlet Table for PCB Components

Placeholder for Datasheet and Electrical Characteristics

Figure 5.1.N - Wall Outlet Datasheet and Electrical Characteristics

5.1.4.2. Power Requirement

The smart wall outlet will replace any existing wall outlet, meaning that the power requirement for the smart wall outlet is the same as a normal wall outlet. The smart wall outlet does not use a grounded pin, meaning the standard NEMA 1-15 ungrounded plug is used in the smart wall outlet to cover up both plugs on the wall. An ungrounded plug is used because some houses as well as some building have wall outlets that don't have grounded pins. This way the smart wall outlet is compatible with all houses and buildings. The NEMA 1-15 ungrounded plug uses up to 15 A current and 125 V volts.

5.1.4.3. Communication with Software

The wall outlet device's communication with the server and web and mobile applications is handled through the software in the microcontroller using the WiFi module. The procedure for the wall outlet device to send and receive data to and from the server and mobile and web applications has 2 forms. These two forms are an upkeep form and an update form.

The upkeep form of the procedure occurs every cycle of the microcontroller software process. The purpose of this form is to maintain a constant stream of data between the device and the server to prevent the connection from being dropped by the ISP for being idle. The second purpose of the upkeep cycle is to maintain up to date power usage data. The final purpose of the upkeep form is to check for updates to the state from the mobile and web applications.

The state data sent by the upkeep form is always the same as the previous data that was sent to the server. This allows the software to save cycles by not updating all of the information being sent every time. Since the upkeep form is used far more often than the update form, the few cycles saved by splitting the two forms and excluding updating the data add up significantly. The data on the power usage through the device is sent as part of this form, in order to keep that information current and up to date.

The data received by the upkeep form is always checked against the data that was just sent to the server. The received data is then decrypted. If the received data is different, the software will go through its update process, which is dependent on and unique to the design of the software, as detailed in section 5.2.1. The update process always includes updating the data to be sent to the server during the next upkeep communication.

The update form of the procedure occurs only when the state of the device has changed since the last time data was sent to or received from the server. This case is specifically for when the device's state was changed manually. This designation allows the server to know that the device was changed manually.

This second form also allows for information to be sent at a constant rate using the upkeep form and still be able to instantly send an update to the server when the state of a device is changed manually.

The data sent by the update form is always the same as the previous info sent to the server, except for the state which will always be different. The update form updates the current state variable in the information being sent to the server, and then sends that data to the server. The only new data being sent is the updated state of the wall outlet. The reason the rest of the data besides the state does not need to be updated in this form is because regardless of whenever this form is triggered the upkeep form still occurs in the cycle, which will update that other data.

The data received by the update form is largely the same as the upkeep form, the difference being the update form only pays attention to the state information. A fast response time to any changes made using the mobile and web applications is an important part of our goals and objectives. As such, checking for updates more often improves this response time, and fits with our goals and objectives. After the data is decrypted it is checked against the current state data for the device. If the newly received state data is different, the wall outlet goes through its update process, which is different depending on the software design used. The different software designs being considered for the wall outlet are detailed in section 5.2.1.

5.1.5. Wall Switch

After we completed our preliminary research, we have decided on a single wall switch that will allow users to wirelessly control their lighting. The wall switch will be connected directly to the neutral wire that is usually standard with all light switches. The wall switch will also be connected to a wireless network via the WiFi and Bluetooth module. The wall switch will also be able to measure the amount of current and voltage the lighting source will use, and transmit this data so that we can calculate the power that the lights use.

Components/features of the wall switch will include:

AC-DC Conversion

In order for the microcontroller, WiFi module, and energy measuring IC to function correctly, the incoming AC voltage will have to be converted into a DC voltage.

Energy Measurement Circuitry

An energy measuring integrated circuit will be used in order to measure the continuous current and voltage signals so that we can calculate the amount of power the lighting uses.

Wireless Module

The WiFi and Bluetooth module will allow for the wall switch to transmit data to the application, and allow the wall switch to receive transmissions to turn on or off the lighting. It will also use low energy to transmit data.

Relay

In order to actually turn the lights on or off, a relay will be used. The relay will act as a switch that will either allow a current to flow through the circuit, or cease any flow that is currently happening.

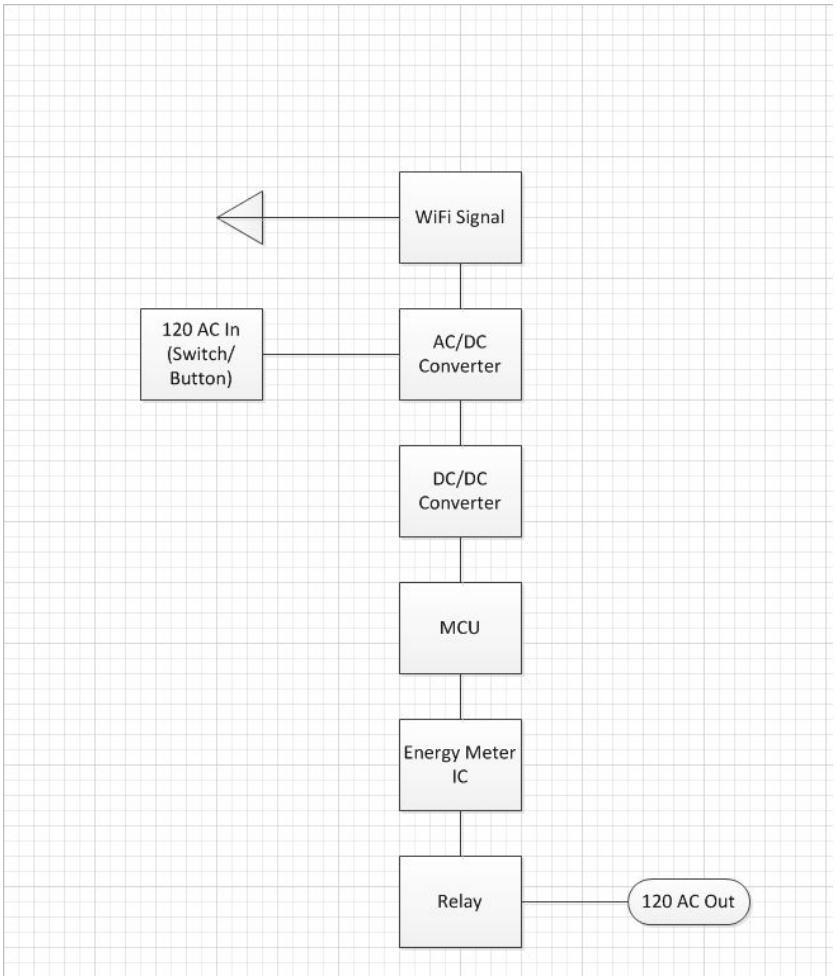


Figure 5.1.O - Smart Wall Switch Hardware Block

Microcontroller

The microcontroller will be responsible for allowing all the other components of the wall switch to operate in tandem. It will also be responsible for any basic calculations we may need done, such as talking the current and voltage and calculating power consumption.

Manual Button

The wall switch will also include a manual button to turn on or off the lights if the user does not wish to use the application.

Figure 5.1.O shows a basic input/output of the wall switch of the home automation system. After the user presses the wall switch or toggles the switch in the application, the signal goes through an AC/DC converter and through a DC/DC converter to reduce voltage. The MCU accepts the signal and goes through an energy meter IC that records the voltage being put it over an infinite period of time or until the wall switch is turned off. The output inverts the wall switch's current state.

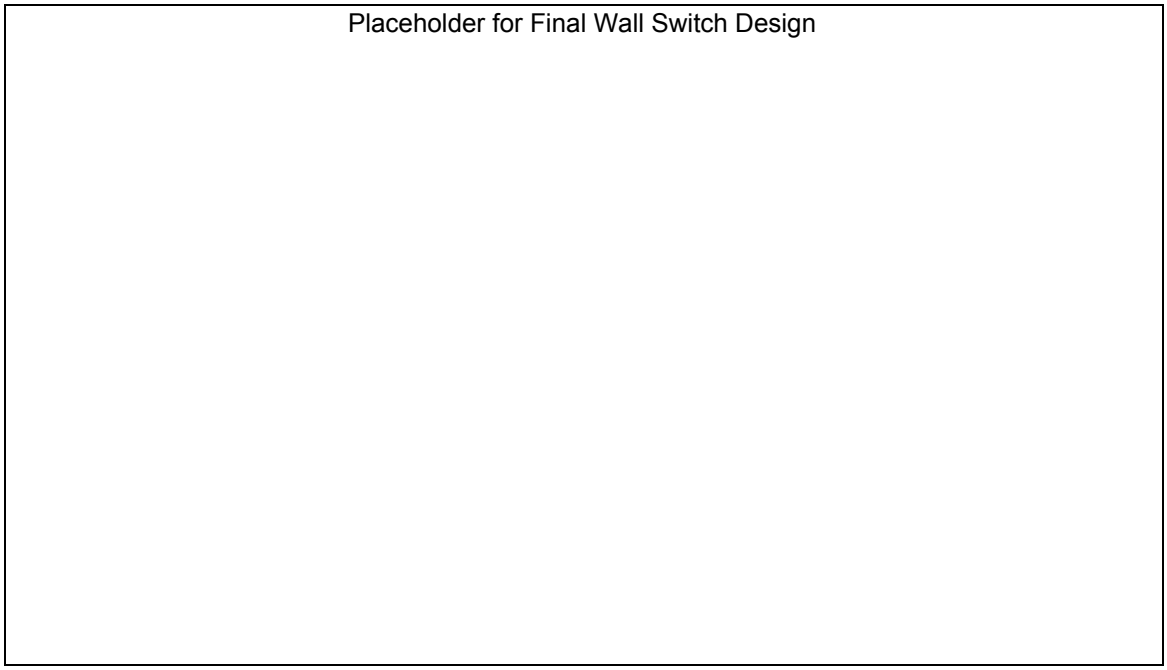


Figure 5.1.P - Final Wall Switch Design

5.1.5.1. Requirements

The wall switch will have the following requirements:

1. Allow for an input of 120V AC

2. Connect to the user's existing switch and neutral wires and use these connections as a power supply
3. Output the correct voltage to operate the lighting
4. Measure the current and voltage that travels to the lighting source
5. Turn the lighting source on or off
6. Interface wirelessly with the application via WiFi or Bluetooth

5.1.5.2. Specifications

Listed in Figures 5.1.Q - 5.1.U are the specifications for the wall switch device.

Input Voltage	120V AC
Input Frequency	60 Hz
Max Current	15 A
Connection	Neutral Wire
Maximum Incandescent Load	600 W

Figure 5.1.Q - Electric Specifications

Current	Yes
Voltage	Yes
Temperature	No
Humidity	No

Figure 5.1.R - Sensor Specifications

Wired	No
Wireless	Yes
Protocols	WiFi, Bluetooth
Band Frequency	2.4 GHz
MAC/PHY specifications	802.11bgn

Figure 5.1.S - Communication Specifications

Dimensions	2.75 inches wide x 4.5 inches high x 1.5 inches thick
Weight	3.5 ounces
Color	White

Figure 5.1.T - Physical Specifications

Temperature	32°F to 122°F
Humidity	5% to 85% Relative Humidity
Elevation	0 - 6000 feet

Figure 5.1.U - Operation Conditions

5.1.5.3. Components

AC to DC Converter (Rectifier)

Since the input voltage coming from the neutral wire will be an AC voltage, the components of the wall switch will not be able to use this directly as a power source. In order for the components to be powered, the AC voltage will have to be converted to a regulated DC voltage, and to do that, we will need to employ --

Forward Voltage	1.3V
Forward Current	1 A
Leakage Current	1 mA
Reverse Recovery	15 ns
Forward Recovery	30 ns
Junction Capacitance	70 pF
Minimum Operating Temperature	0°C
Maximum Operating Temperature	70°C
Dimensions	9.81 mm length x 6.35 mm x 4.57 mm height
Weight	0.019 ounces

Figure 5.1.V - UC3610N Specifications

-- some type of circuitry that will perform this. One method that we can use is an AC to DC converter, or as it is also known, a rectifier.

In order for the wall switch to work correctly, the rectifier will first have to take in the 120V input voltage, convert that into a DC voltage, and then filter that DC voltage so that it is smoothed, as the unfiltered DC voltage will still have pulses. The input voltage will also have to be stepped down so that it can be used by the components of the wall outlet. There are two methods of rectification that can be employed: half wave rectification and full wave rectification. We plan on using a rectifier that uses full wave rectification, as full wave rectification produces less ripple voltage than half wave, which could lead to incorrect functionality of the circuitry. Full wave rectification also requires less filtering in order to eliminate harmonics. The rectifier that we decided to use is the TI UC3610N, and specifications are shown in Figure 5.1.V.

Energy Measurement Circuitry

In order for us to calculate how much energy the user consumes with their lighting, our wall switch will have to employ some type of circuitry that will be able to measure the voltage and current that flows through the device. To accomplish this, we plan on employing energy measurement integrated circuits, or ICs, that will measure the current and voltage that is being consumed, and send this data to the microcontroller for processing.

Measurement Error	0.1%
Bandwidth	14 kHz
Input Range	2.2-2.6 V
Input Impedance	3.2 kΩ
Input Capacitance	10 pF
Operating Voltage	4.5-5.5 V
Operating Temperature Range	-40°C to 125°C
Storage Temperature Range	-65°C to 150°C

Figure 5.1.W - MCP3909 Specifications

There are some projects which used multiple components for this, such as combining an AC voltage meter and a current meter, then calculating the energy used based on both those measurements. However, we choose to use an energy

measurement IC instead, as it cuts down on necessary components. It is important that the measurement is extremely accurate, so that we can display accurate information to the users. The IC should also be compact so that our wall outlet will not have to be overly large in order to compensate. The energy measuring IC we planned on implementing is the MCP3909, whose specifications are located in Figure 5.1.W.

Wireless Module

To facilitate the wireless communication between the wall outlet and the user's smart device or computer, we will employ a module that supports both Wifi and Bluetooth communication. The reason we chose to employ this was so that if the user was in close proximity to their device, they could use Bluetooth, while if they were away from home, they could instead use Wifi. With Bluetooth, any device using this communication protocol can connect with any user, as long as they are within a certain proximity of each other. Any device in a piconet (Bluetooth network of multiple devices) can connect to up 7 other devices. The maximum range for a Bluetooth technology is 10 meters or 30 feet, but there is no limit so the manufacturers can implement or set their own range for the specific device. Bluetooth devices have a lot of benefits including: robustness, low power, and low cost. Bluetooth operates in an ISM (Industrial, Scientific, and Medical) band at 2.4 to 2.485 GHz. Also, Bluetooth use the adaptive frequency hopping, which is nothing more than detecting other devices nearby and avoiding their frequencies so their own signal won't be interfere. And because this technology uses low power consumption, the radios are powered down when inactive, which is a plus for this project because the main objective is to lower power.

Relay

One of the key features of our project is the ability to power on or off any of the products we develop connected wirelessly. They are three states that the wall switch will operate in: on, off, or standby. Many electronic devices and appliances will continue to draw power even if they are powered off but are still plugged in. This power that is consumed is called standby power or phantom energy. In average every household has a number of appliances that are constantly drawing phantom energy, eventually making the electric bill higher. Phantom energy can account for five to ten percent of an energy bill each month.

Many methods exist in order to verify that the power will be shut off completely in our wall outlet. We will have three different methods in which the power will be shut off too the device. The first method will be done through the application, where we will determine certain times of the day where the appliance connected to the outlet should be completely shut off. This automated control is regularly used in many designs turning on and off, lights, air conditioners, and other appliances. The next method will allow the user to shut off the device remotely

by use of a cell phone. The last method will be a minimum voltage control. In this method, when the measurements of voltage used fall below certain level, the actuator will shut of the device completely until is wirelessly activated. Many recent power strips and surge protectors employ technology that does this, to save on energy.

Rated Voltage	100-240 VAC
Operating Voltage	75-264 VAC
Impedance	72 kΩ ± 20%
Must Operate Voltage	75 VAC max
Must Release Voltage	20 VAC min
Rated Load Voltage	400-600 VAC
Load Voltage Range	360-660 VAC
Operate time	½ of load power source cycle + 1 ms max (DC input)
Output ON voltage drop	1.8 V (RMS) max
Leakage Current	10 mA max (at 400 VAC) 20 mA max (at 600 VAC)
Insulation Resistance	100 MΩ min
Dielectric Strength	4,000 VAC, 50/60 Hz for 1 min
Vibration Resistance	Destruction: 10-55-10 Hz, 0.75mm single amplitude (1.5mm double amplitude)
Shock Resistance	Destruction: 1,000 m/s^2
Ambient Temperature	Operating: -30°C to 80°C Storage: -30°C to 100°C
Ambient Humidity	Operating: 45% to 85%
Weight	120 g

Figure 5.1.X - Omron G3NA-6 Series Specifications and Characteristics

The relay is the type of electronic component that we will use in our design. There are two types of relays that are commonly used: the mechanical relay and

the solid state relay. The mechanical relay is a simpler and safer method to implement than other types of relays that are used, but it comes with the disadvantage of being very bulky and inconvenient for our small project. Other types of relays that we had up for consideration were: zero crossing silicon controlled rectifier (SCR), intelligent solid state contactor, SSC or solid state contactor, phase angle silicon controlled rectifier (SCR), mosfet solid state relay, IGBT(insulated Gate bipolar transistor), solid state relay and the opto-couple Triac. We decided to implement the solid state relay, as it would be the most suited for our project. Solid state relays are low cost, and since it does not contain any moving parts, it is ideal for a circuit. One risk of a solid state relay is that they tend to fail when they are shorted, meaning that when they fail a current still flows throughout the circuit, whereas mechanical relays tend to fail when "open", meaning no current is flowing. They also become expensive to build in high current ratings, but since our project will not be using very high currents this negative is not very impactful. One major benefit of using a solid state relay is the speed at which the switch can open and close, which is much faster than that of the mechanical relay, as there is an armature that must move in order for it to open and close. Solid state relays also use low current, produce very little noise when switching, are significantly smaller than other types of relays, and are less sensitive to the environment (humidity and magnetic fields). For our project, we decided to go with an Omron G3NA-6 series solid state relay, with specifications shown in Figure 5.1.X.

5.1.5.4 Communication with Software

The wall switch device's communication with the server and web and mobile applications is handled through the software in the microcontroller using the WiFi module. The procedure for the wall switch device to send and receive data to and from the server and mobile and web applications has 2 forms. These two forms are an upkeep form and an update form.

The upkeep form of the procedure occurs every cycle of the microcontroller software process. The purpose of this form is to maintain a constant stream of data between the device and the server to prevent the connection from being dropped by the ISP for being idle. The second purpose of the upkeep cycle is to maintain up to date power usage data. The final purpose of the upkeep form is to check for updates to the state from the mobile and web applications.

The state data sent by the upkeep form is always the same as the previous data that was sent to the server. This allows the software to save cycles by not updating all of the information being sent every time. Since the upkeep form is used far more often than the update form, the few cycles saved by splitting the two forms and excluding updating the data add up significantly. The data on the power usage through the device is sent as part of this form, in order to keep that information current and up to date.

The data received by the upkeep form is always checked against the data that was just sent to the server. The received data is then decrypted. If the received data is different, the software will go through its update process, which is dependent on and unique to the design of the software, as detailed in section 5.2.2. The update process always includes updating the data to be sent to the server during the next upkeep communication.

The update form of the procedure occurs only when the state of the device has changed since the last time data was sent to or received from the server. This case is specifically for when the device's state was changed manually. This designation allows the server to know that the device was changed manually. This second form also allows for information to be sent at a constant rate using the upkeep form and still be able to instantly send an update to the server when the state of a device is changed manually.

The data sent by the update form is always the same as the previous info sent to the server, except for the state which will always be different. The update form updates the current state variable in the information being sent to the server, and then sends that data to the server. The only new data being sent is the updated state of the wall switch. The reason the rest of the data besides the state does not need to be updated in this form is because regardless of whenever this form is triggered the upkeep form still occurs in the cycle, which will update that other data.

The data received by the update form is largely the same as the upkeep form, the difference being the update form only pays attention to the state information. A fast response time to any changes made using the mobile and web applications is an important part of our goals and objectives. As such, checking for updates more often improves this response time, and fits with our goals and objectives. After the data is decrypted it is checked against the current state data for the device. If the newly received state data is different, the wall switch goes through its update process, which is different depending on the software design used. The different software designs being considered for the wall switch are detailed in section 5.2.2.

5.1.6 Powerlock

For our powerlock, we plan on developing a lock that will employ an electric strike that will allow the user to wirelessly lock and unlock whatever door the lock in installed in. This product will also employ a Wifi & Bluetooth module to allow for wireless use, but we will not be including any energy measurement circuitry, as this will operate off of it's own independent power source.

Components/features of the Powerlock will include:

Wireless module

The WiFi and Bluetooth module will allow for the powerlock to transmit data to the application, and allow the powerlock to receive transmissions to either lock or unlock the door. It will also use low energy to transmit data.

Electric Strike

The electric strike is the component that will actually control whether or not the door is unlocked or locked. It has a ramped surface that can pivot upon given a command to do so, allowing the user to operate a door without a mechanical lock or key.

Power Source

In order to power the electric strike and the circuitry required for the powerlock, we will employ a small DC power source, as it will make it so that we do not have to worry about converting from AC to DC for the other aspects of the powerlock, such as the Wifi & Bluetooth Module.

Powerlock Hardware Block

Figure 5.1.Y below is a basic input/output of a tablet command of the powerlock system used for the doors. The door can accept a wireless signal using the app that goes through a WiFi signal. The signal then opens/closes a latch that will allow a 3.3 DC voltage to go through, opening or closing the electric striker. This can all be circumvented if the door is opened normally.

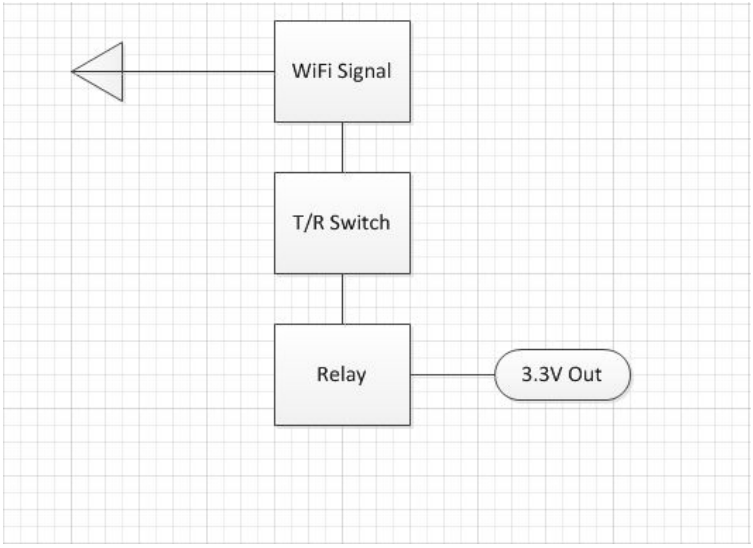


Figure 5.1.Y - Block Diagram

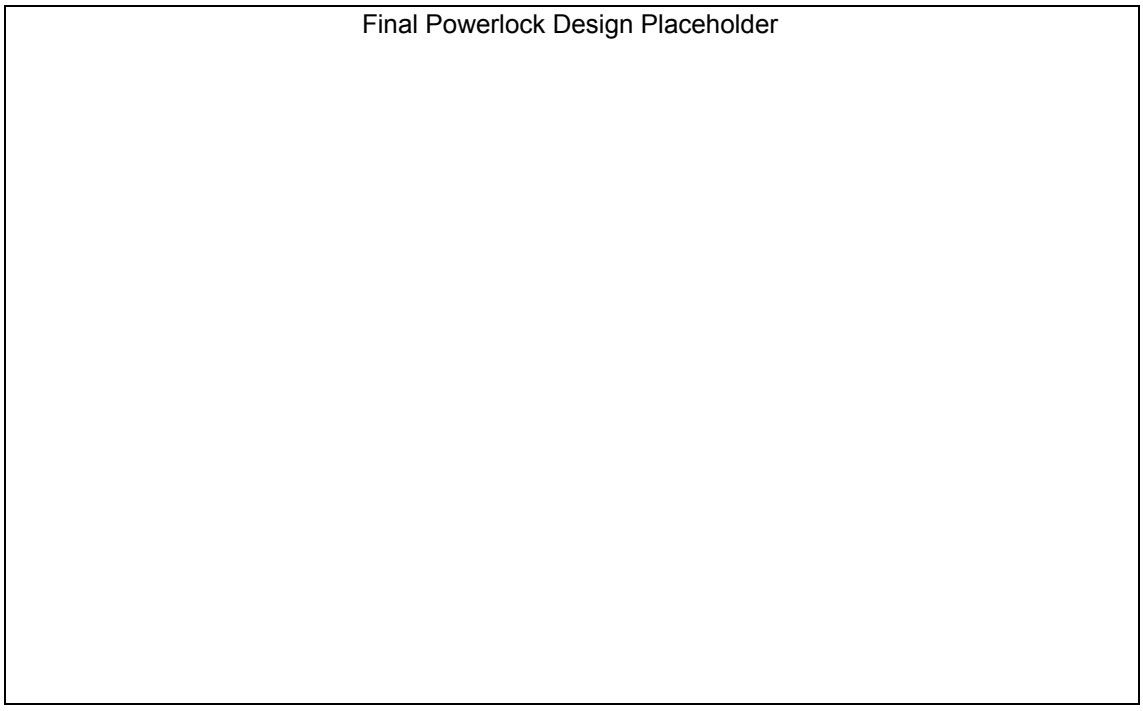


Figure 5.1.Z - Final Powerlock Design

5.1.6.1. Requirements

The powerlock will have the following requirements:

1. Operate with a DC power supply
2. Allow for easy installation in place of an existing lock on a door
3. Be controlled wirelessly via the application to either lock or unlock the user's door
4. Communicate with the application to update the status of the door (locked or unlocked)

5.1.6.2. Powerlock Specifications

Listed below in Figures 5.1.AA - 5.1.AD are the specifications for the powerlock.

Input Voltage	12 VDC
Max Current	2 A
Input Frequency	n/a
Power supply mode	Switching

Figure 5.1.AA - Electric Specifications

Wired	No
Wireless	Yes
Protocols	Wifi, Bluetooth
Band Frequency	2.4 GHz
MAC/PHY Specifications	802.11 bgn

Figure 5.1.AB - Communication Specifications

Dimensions	3 inches tall x 2.75 inches wide x 0.8 inches thick
Weight	6 ounces
Strike material	Stainless steel

Figure 5.1.AC - Physical Specifications

Temperature	32°F to 122°F
Humidity	5% to 85%
Elevation	0-6000 feet

Figure 5.1.AD - Operation Conditions

5.1.6.3. Components

Electric Strike

An electric strike is a low voltage access control device that can be used in place of traditional locks to provide added security and conveniences such as traffic control, specific and limited access as well as remote lock/unlock. Electric strikes generally have at most +/- 10% voltage tolerances and most electric strikes will require 12 DC volts to function, meaning to lock and/or unlock the hinge. Another common configuration for electric strikes requires 24 AC volts instead.
Electric strikes can come in two types of security configurations, “fail secure” and “fail safe”. A fail-secure electric strike, which is also commonly referred to as normally opened (NO switch) function type, is a configuration where by applying an electric current to the strike will cause it to unlock. They can be powered by alternating current (AC) or direct current (DC). Electric strikes that operate on AC can have a slight buzzing noise when a current is applied, where DC is virtually noiseless. In case of a power failure the strike would remain locked. A fail safe type also called normally closed (NC switch) is one in which applying an electric

current to the strike will cause it to lock, meaning that it needs power to keep it locked. Fail safe locks always use direct current DC. In case of a power failure the door could be opened by being pushed or pulled. A fail secure (normally opened) type electric strike will most likely be used for this type of project because it is more ideal for security purposes. The electric strike we decided to implement is the EDL-SL-YS131-NO, whose specifications are listed in Figure 5.1.AE.

Operating Mode	NO (Fail Secure)
Voltage	12 VDC
Current	200 mA
Operating Temperature	-10°C to 55°C
Holding Strength	500kg
Strike Structure	Stainless Steel
Dimensions	160 mm length x 25 mm width x 31 mm height

Figure 5.1.AE - EDL-SL-YS131-NO Specifications

Power Supply

For the powerlock, we decided to go with a DC voltage power supply over an AC power supply for a number of reasons. The biggest reason was that with a DC power supply, we would not have to worry about converting to DC so that all the components would be able to use power. Using a DC power supply over an AC power supply will also reduce heat and power dissipation, which leads to higher energy savings.
There are two types of power supplies that we can consider for this project: switching and linear. Switching power supplies convert the main power to a load by changing the voltage and current characteristics. The switching supply constantly switches from low dissipation to high dissipation, but does not linger while it is in high dissipation. Idealy, the switching device does not dissipate power, whereas a linear power supply is constantly dissipating power in the pass transistor. Another advantage of a switching power supply over a linear one is physical dimensions. Switching power supplies most of the time are smaller and lighter, which will help make our design more compact. Switching regulators are generally used when higher efficiency, smaller size and lighter weight is necessary for the design. Figure 5.1.AF shows a comparison between the two types. The power supply that we decided to go with is the ACUS-12V, whose specifications are shown in Figure 5.1.AG.

Specification	Linear	Switching
Line regulation	0.02-0.05%	0.05-0.1%
Load regulation	0.02-0.1%	0.1-1.0%
Output ripple	0.5-2 mV RMS	10-100 mV
Input voltage range	±10%	±20%
Efficiency	40-55%	60-95%
Power Density	0.5 W/cu in	2-10 W/cu in
Transient Recovery	50 us	300 us
Hold up time	2 ms	34 ms

Figreu 5.1.AF - Switching vs. Linear Specifications

Input Range	90-240 VAC
Output Voltage	12 VDC
Output Current	1 A
Power Consumption	12W

Figure 5.1.AG - ACUS-12V Specifications

5.1.6.4 Communication with Software

The powerlock's communication with the server and web and mobile applications is handled through the software in the microcontroller using the WiFi module. The procedure for the powerlock to send and receive data to and from the server and mobile and web applications has 2 forms. These two forms are an upkeep form and an update form.

The upkeep form of the procedure occurs every cycle of the microcontroller software process. The purpose of this form is to maintain a constant stream of data between the device and the server to prevent the connection from being dropped by the ISP for being idle. The second purpose of the upkeep cycle is to maintain up to date power usage data. The final purpose of the upkeep form is to check for updates to the state from the mobile and web applications.

The state data sent by the upkeep form is always the same as the previous data that was sent to the server. This allows the software to save cycles by not

updating all of the information being sent every time. Since the upkeep form is used far more often than the update form, the few cycles saved by splitting the two forms and excluding updating the data add up significantly. The data on the power usage through the device is sent as part of this form, in order to keep that information current and up to date.

The data received by the upkeep form is always checked against the data that was just sent to the server. The received data is then decrypted. If the received data is different, the software will go through its update process, which is dependent on and unique to the design of the software, as detailed in section 5.2.4. The update process always includes updating the data to be sent to the server during the next upkeep communication.

The update form of the procedure occurs only when the state of the device has changed since the last time data was sent to or received from the server. This case is specifically for when the device's state was changed manually. This designation allows the server to know that the device was changed manually. This second form also allows for information to be sent at a constant rate using the upkeep form and still be able to instantly send an update to the server when the state of a device is changed manually.

The data sent by the update form is always the same as the previous info sent to the server, except for the state which will always be different. The update form updates the current state variable in the information being sent to the server, and then sends that data to the server. The only new data being sent is the updated state of the powerlock. The reason the rest of the data besides the state does not need to be updated in this form is because regardless of whenever this form is triggered the upkeep form still occurs in the cycle, which will update that other data.

The data received by the update form is largely the same as the upkeep form, the difference being the update form only pays attention to the state information. A fast response time to any changes made using the mobile and web applications is an important part of our goals and objectives. As such, checking for updates more often improves this response time, and fits with our goals and objectives. After the data is decrypted it is checked against the current state data for the device. If the newly received state data is different, the powerlock goes through its update process, which is different depending on the software design used. The different software designs being considered for the powerlock are detailed in section 5.2.4.

5.1.7. HVAC Controller

A vital part of a standard home automation system is being able to control the air conditioning around the house. For the purpose of this project, a simulated air

conditioning system will be used and placed onto a plexiglass board for testing and presentation. The HVAC controller be built from scratch; an existing HVAC controller will be used and the insides will be customized to be compatible with the project's software and applications. A basic layout of the HVAC controller can be seen in Figure 5.1.AH.

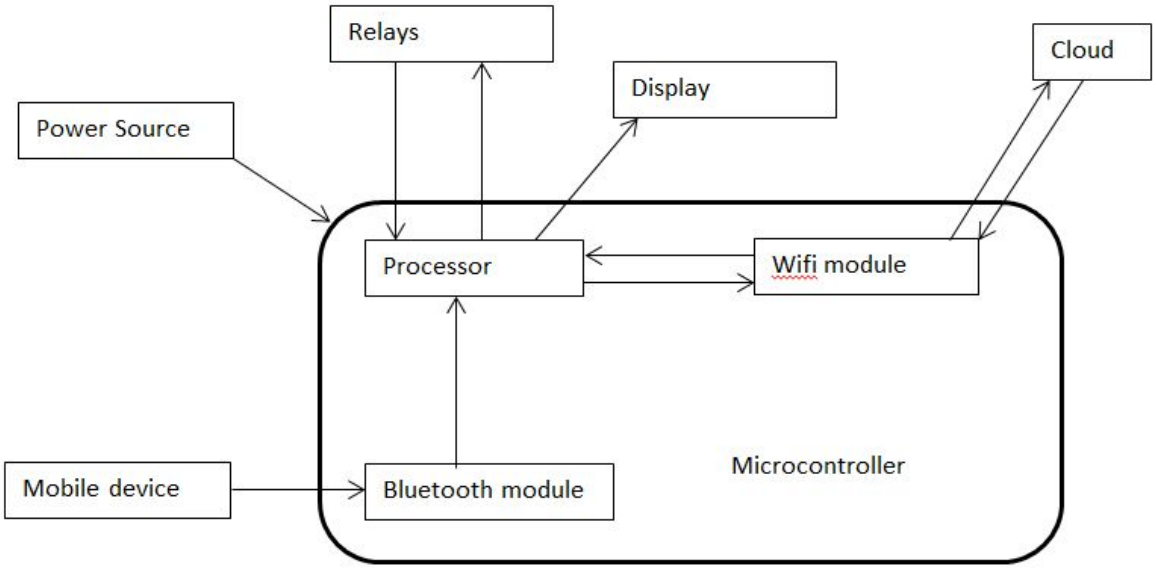


Figure 5.1.AH - HVAC Controller Block Diagram

The controller that will be modified is the Vent-Miser 91668 Programmable Energy Saving Vent. The vent is already programmed to synergize with the timer module. What we're interested in is the timer module. After removing the back of the timer module, the PCB and 3V motor can be easily removed. The PCB is a single layer board that is powered by two AAA batteries, as shown in Figure 5.1.AI. One of the advantages of this timer module is that a lot of space in the casing is empty and isn't used for anything. This allows for extra circuitry and components.

In order to have this module be compatible with our home automation system, the module needs to be wirelessly controlled by the smartphone app. The original timer module contains a basic LCD screen with buttons to program the vent to open and close at certain intervals. The LCD screen displays the current time, start time, and end time for the vent to open/close. This functionality will be kept and not tampered with. Adding our components to this PCB would require the need of the WiFi/Bluetooth module used in the other components of the home automation system. This is done by simply replacing the existing MCU with our custom made MCU, connecting the WL18xxMOD 8 Single-Band Combo Module to the Atmel SAM C ARM Cortex MCU and adding the LM134 3-Terminal Adjustable Current Source to monitor power. The electrical characteristics can be seen in Figure 5.1.AJ

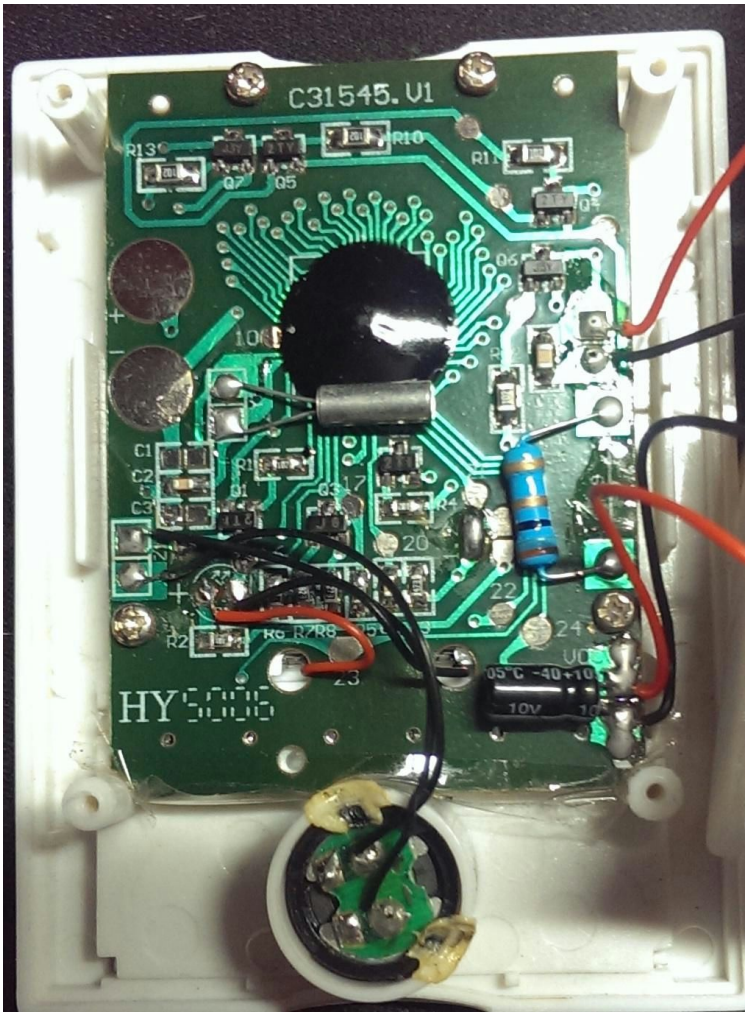


Figure 5.1.AI - HVAC Controller PCB

Placeholder for Datasheet and Electrical Characteristics

Figure 5.1.AJ - HVAC Controller Datasheet and Electrical Characteristics

5.1.7.1. Dimensions

Because of the big size of the casing, there is no need to make any custom casing and the original dimensions of the package. Combining the vent and timer module, the dimensions are 14.4 x 8 x 2.6 inches. This specific vent is meant for a standard bedroom on the wall near the ceiling in a position that the cool air could cover the room. Other Vent-Miser products show that there are variations in the size of the vent, such as 14.4 x 4.3 x 2.6 inches and 10.0 x 4.0 x 2.6 inches. For installing these kind of vents throughout an entire house, different size vents would need to be used. However, the same custom timer module can be used throughout.

5.1.7.2. Power Requirement

The HVAC controller does not plug into the wall for power, so the standard 120 V AC voltage is not needed. Instead, the controller uses 2 AA batteries for power. Each AA battery contains up to 2500 milliamp-hours which, in series, causes a 5000 milliamp-hours charge as well as 2.4 volts, 1.2 volts coming from each AA battery. With this in mind, this means the HVAC controller can operate for, theoretically, about 1000 hours. Converting this number to days, the HVAC controller can operate for about 41 days with 2 full AA batteries, which is roughly over a month.

5.1.7.3. Communication with Software

The HVAC controller's communication with the server and web and mobile applications is handled through the software in the microcontroller using the WiFi module. The procedure for the HVAC controller to send and receive data to and from the server and mobile and web applications has 2 forms. These two forms are an upkeep form and an update form.

The upkeep form of the procedure occurs every cycle of the microcontroller software process. The purpose of this form is to maintain a constant stream of data between the device and the server to prevent the connection from being dropped by the ISP for being idle. The second purpose of the upkeep cycle is to maintain up to date power usage data. The final purpose of the upkeep form is to check for updates to the state from the mobile and web applications.

The state data sent by the upkeep form is always the same as the previous data that was sent to the server. This allows the software to save cycles by not updating all of the information being sent every time. Since the upkeep form is used far more often than the update form, the few cycles saved by splitting the two forms and excluding updating the data add up significantly. The data on the power usage through the device is sent as part of this form, in order to keep that information current and up to date.

The data received by the upkeep form is always checked against the data that was just sent to the server. The received data is then decrypted. If the received data is different, the software will go through its update process, which is dependent on and unique to the design of the software, as detailed in section 5.2.3. The update process always includes updating the data to be sent to the server during the next upkeep communication.

The update form of the procedure occurs only when the state of the device has changed since the last time data was sent to or received from the server. This case is specifically for when the device's state was changed manually. This designation allows the server to know that the device was changed manually. This second form also allows for information to be sent at a constant rate using the upkeep form and still be able to instantly send an update to the server when the state of a device is changed manually.

The data sent by the update form is always the same as the previous info sent to the server, except for the state which will always be different. The update form updates the current state variable in the information being sent to the server, and then sends that data to the server. The only new data being sent is the updated state of the HVAC controller. The reason the rest of the data besides the state does not need to be updated in this form is because regardless of whenever this form is triggered the upkeep form still occurs in the cycle, which will update that other data.

The data received by the update form is largely the same as the upkeep form, the difference being the update form only pays attention to the state information. A fast response time to any changes made using the mobile and web applications is an important part of our goals and objectives. As such, checking for updates more often improves this response time, and fits with our goals and objectives. After the data is decrypted it is checked against the current state data for the device. If the newly received state data is different, the HVAC controller goes through its update process, which is different depending on the software design used. The different software designs being considered for the HVAC controller are detailed in section 5.2.3.

5.2. Microcontroller Software Design Plan

All of the software written for the microcontrollers is written in the C language using the Atmel Studio 7.0 integrated development platform with the ATSAMC21J18A firmware. In each section the design patterns to be tested for the indicated device will be detailed. The final design for each device will likely use a combination of all designs detailed within the sections for each device.

5.2.1. Wall Outlet

The design of the microcontroller software for the wall outlet device serves as the basis for all of the other devices. The basic designs of the wall outlet software can be seen as the foundation that the software for the rest of the devices is built on. 3 different design patterns will be tested for the wall outlet device.

5.2.1.1. First Design: Interrupt Based

The first design of the software for the microcontroller in the wall outlet will be based around interrupt service routines. When the software begins execution, it will begin by reading the current state of the outlet, which will be either on or off. The software then sets that state as a variable and enters the initial waiting phase. During this initial waiting phase, the device is idling until a Bluetooth connection is made and data detailing the WiFi connection is received. The microcontroller will use this data received from the Bluetooth module to configure the WiFi module and attempt a connection. The Bluetooth module will then send the results of the WiFi module's connection test to the device that made the Bluetooth connection. If the WiFi module failed to connect properly, the software will re-enter the waiting phase and wait for a new set of configuration settings to be sent for the WiFi module via the Bluetooth module. If the connection test is successful, the software will notify the device that made the Bluetooth connection, send the information detailing the device's current state to the database, and enter the main phase.

While in the main phase the software will periodically check and, if needed, update its state while waiting for an interrupt from either a WiFi or Bluetooth connection. Along with the current state of the device, data for the power usage of that outlet will be stored. Within the process of updating the current state of the wall outlet device, the software will send that updated state to the WiFi module to be sent to the database. At the end of the main phase upkeep data will be sent to the server. An interrupt will occur whenever either the Bluetooth or WiFi module receives data. If the Bluetooth module is the module that received data, the software will trigger an interrupt and follow the same process as the initial waiting phase. If the WiFi module is the module that received data, the software will trigger an interrupt and enter the update phase.

During the update phase the software will take the data received from the WiFi module and send signals to the circuit that will alter the state of the hardware accordingly (either from on to off, or from off to on). The software will then confirm the state of the circuit, update its own variable as to the new current state, and then send data containing the updated state to the database. The software will then re-enter the main phase, and wait for another interrupt to occur.

Figure 5.2.A shows a flowchart of the first microcontroller software for the wall outlet device as detailed above. This design pattern makes use of the microcontroller's low power mode during the main phase, and very infrequently sends data to the database, only doing so when the state of the wall outlet device is updated. Interrupts triggered by the receipt of data from the WiFi module would be given priority over those triggered by a Bluetooth connection so that all circuit state changes will occur and the database will be updated properly. By doing so, state changes also occur as soon as they are requested, reducing response time. Issues with this design pattern arise with the infrequency of updates to the database and the possibility of an interrupt from the WiFi module occurring during an update to the WiFi module's configuration during an interrupt from data received by the Bluetooth module.

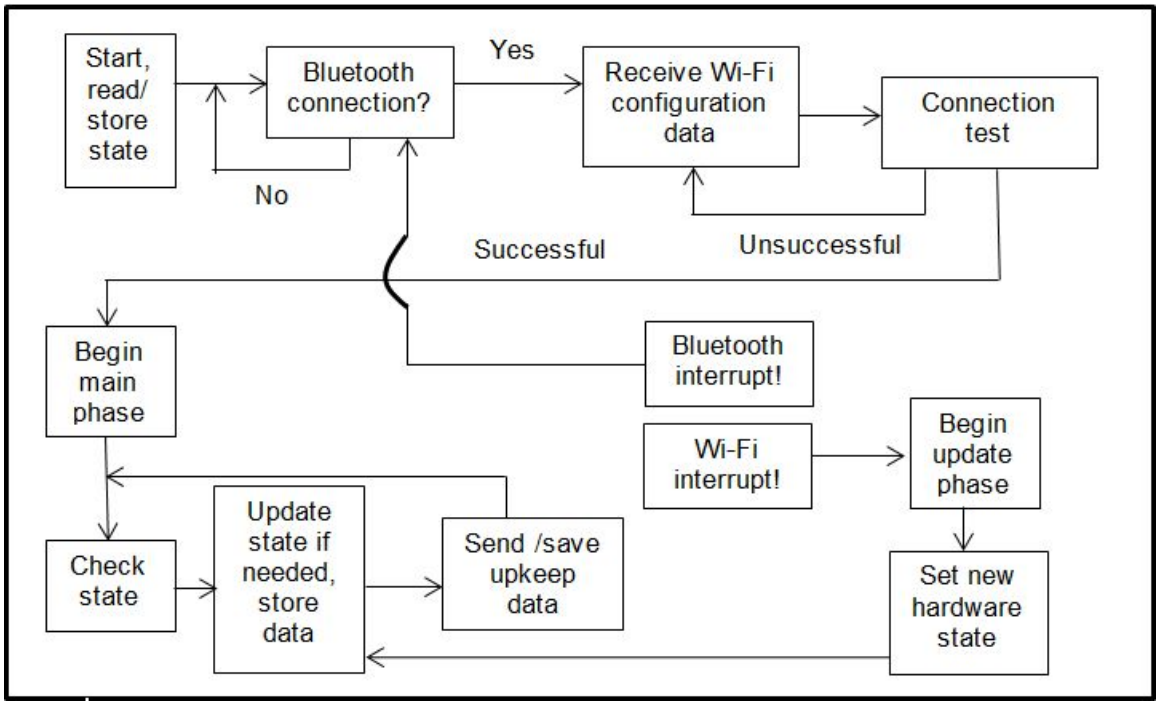


Figure 5.2.A - Flowchart for wall outlet microcontroller software design 1

5.2.1.2. Second Design: Procedurally Based

The second design pattern for the wall outlet device follows the same procedures as the first design when the software is first executed, checking the state and entering a waiting phase while waiting for data from the Bluetooth module. Where this second design differs is in its main phase. The main phase of design pattern two combines the main and update phase of design pattern one. Design pattern 2 does this by not using any interrupts for the main phase. Instead, design pattern 2 does every step of the process every cycle through the main phase.

The main phase begins by checking the current state of the circuit. If the current state of the circuit is different than the state in the software's state variable, then the software sends update data to the database and updates its local variable. The main phase then reads the current power output of the of the wall outlet, and stores that data. The stored power output data is then sent to the database. The main phase then checks if there is a connection from a Bluetooth device. If there is a connection from a Bluetooth device then the software receives the data and moves to the update WiFi module configuration function. This function will configure the WiFi module, and then test the connection to the internet. If the connection test is unsuccessful it will notify the device connected via Bluetooth, get new data from the Bluetooth device, and try the connection again, repeating until the connection test is successful. After checking for a Bluetooth connection and taking the appropriate course of action (either do nothing or update the WiFi module), the software then checks for data received from the WiFi module. If new data was received via the WiFi module it changes the state of the circuit accordingly. The main phase then sends upkeep data to the server and loops back to the start, and begins again by checking the current state of the circuit against its local state variable.

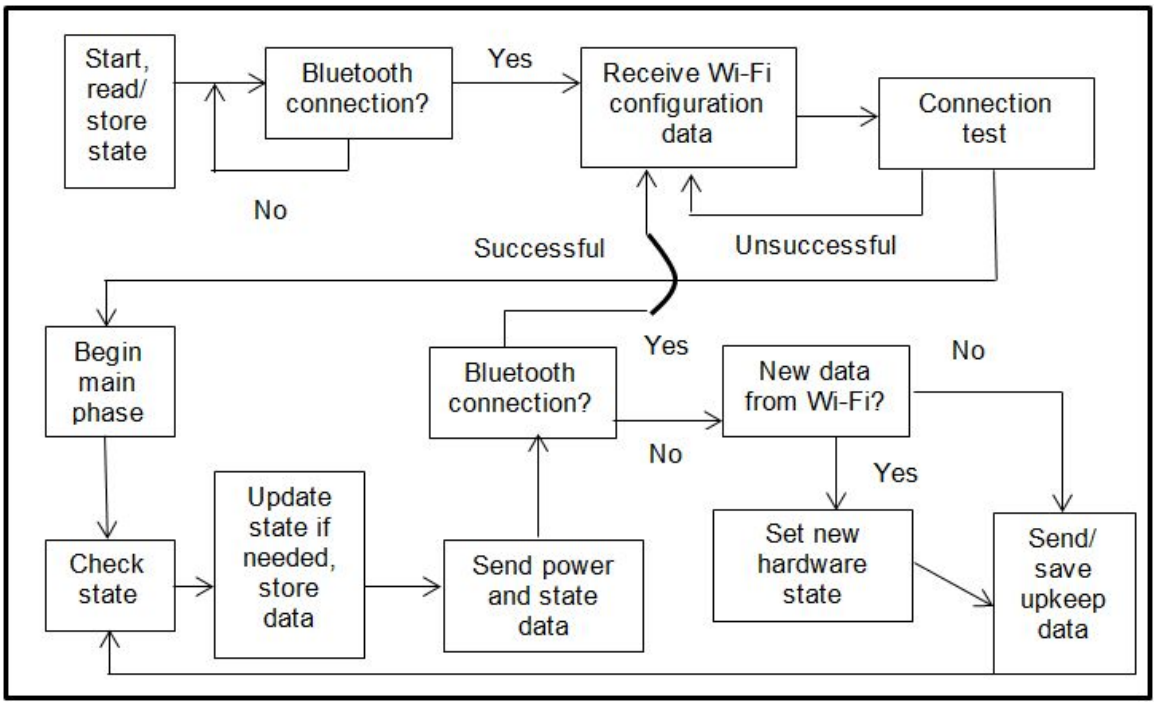


Figure 5.2.B - Flowchart for wall outlet microcontroller software design 2

Figure 5.2.B shows a flowchart for wall outlet microcontroller software design pattern two as it is detailed above. The benefits of design pattern two are that the database is constantly updated to ensure current data, and there's no possibility for interrupts to change data at inopportune times. The negatives of this pattern, however, are many. This design pattern constantly sends data back to the

database, which uses significantly more data than design pattern one. Design pattern 2 also allows for no idle or low power time, increasing the power consumption of the device. The main phase of design two also has many unnecessary steps, such as checking for a Bluetooth connection, that increase the cycle time of the process. Due to design two being entirely procedural, the software must reach a specific point in the main cycle before it can perform an action, causing a delay in response time.

5.2.1.3. Third Design: Multithreading Based

The third design for the wall outlet device incorporates aspects of both previous designs. The third design does not focus on phases like the previous two designs; instead it will incorporate multithreading to keep the database as up to date as possible, while still monitoring the device.

The initial waiting phase from the previous two designs returns for the third design, although in the third design it is incorporated into the startup of the device and not referenced later in the program. Upon startup, the device will wait for a Bluetooth connection. When a Bluetooth connection is made, the program will wait for WiFi configuration data to be input. When the configuration data is received the program will perform a connection test, if the connection test fails the program will wait for another connection. When a connection is made successfully the program will store the configuration settings, and read both the current power output of the device and the current state of the device (on or off), then store that data locally. The program then splits into two threads: the monitoring thread, and the updating thread. To prevent reading a variable while it is being written to, variables will be locked while being written to. If a thread attempts to read a locked variable, it will wait until the variable is unlocked to read the variable.

The first thread will be referred to as the monitoring thread, because it monitors the current state of the device and updates data accordingly. The monitoring thread begins by checking both the state and power output of the device, and updating both of those variables. While updating both of those variables, they will be locked to prevent the updating thread from reading them while they are being written to. The variables are then unlocked, allowing them to be read. The monitoring thread then moves to check if there is a Bluetooth connection. If a connection is found, the program locks the WiFi configuration variable, preventing the updating thread from sending data to the database. The program waits for WiFi configuration information from the Bluetooth connection. Upon receiving WiFi configuration information, a connection test will be performed. If the connection test is successful, the WiFi configuration data will be updated, the variable unlocked, and the thread will resume. Should the connection test fail or the Bluetooth connection be lost, the thread will unlock the WiFi configuration

variable without updating it, and then resume. Upon resuming, the thread loops back to the beginning and checks the current state of the device again.

The second thread will be referred to as the updating thread, because it both updates the database and updates the device. Upon startup, the updating thread reads the WiFi configuration variable, sets its own local WiFi configuration variable, and then moves to the main loop. Within the main loop, the thread reads the main WiFi configuration variable, and checks it against its own. If the two variables are different, the variable local to the updating thread will be updated to match the WiFi configuration variable. The WiFi configuration variable local to the updating thread is the data used for the connection to the database. The thread then reads the state and power output variables and sends them to the database using the WiFi configuration settings stored in the configuration variable local to the updating thread. Next the updating thread checks if data was received from the online application to change its state. If state change data was received, the thread sends a signal to the circuit to change its state. The thread then repeats back to the beginning of the main loop.

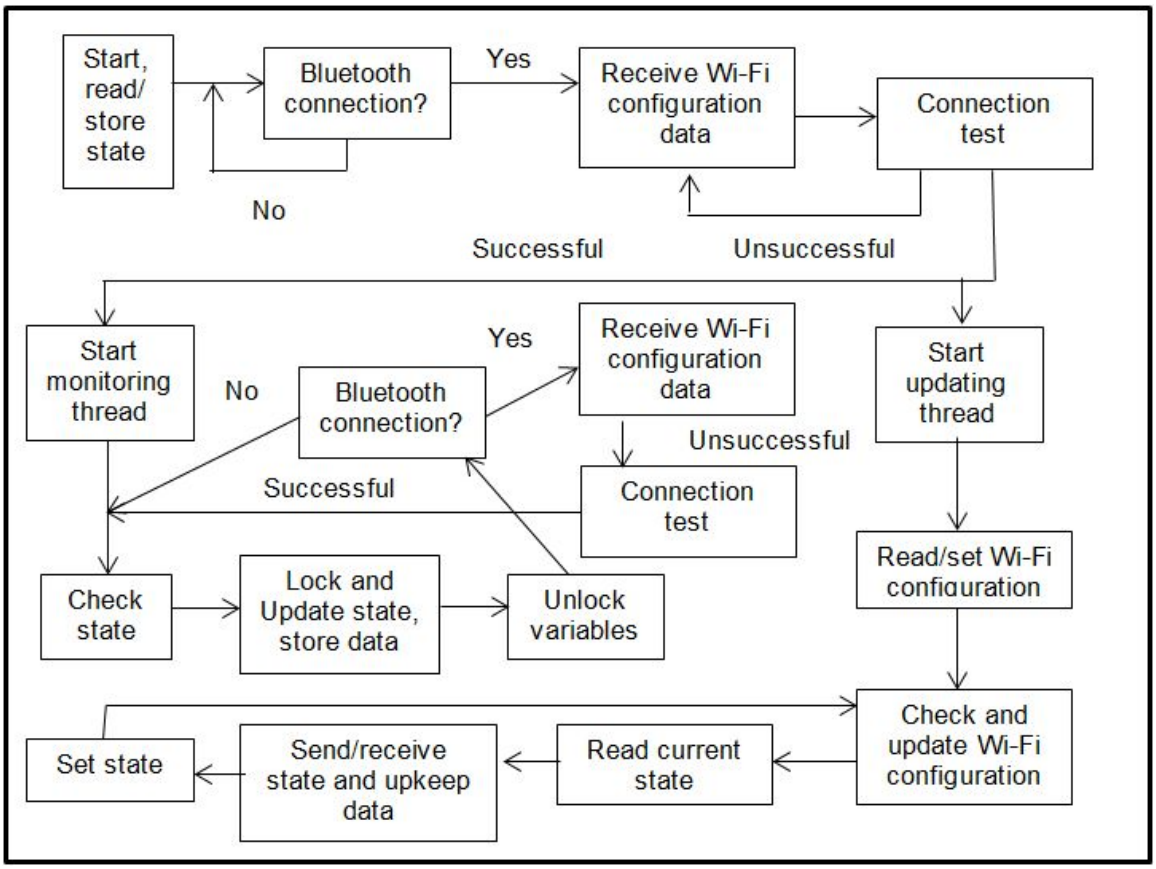


Figure 5.2.C - Flowchart for wall outlet microcontroller software design 3

Figure 5.2.C shows a flowchart for the third design of the microcontroller software for the wall outlet device. The main benefit of the third design is that it keeps the

database constantly up to date with the state and power usage of the wall outlet. The main drawback to this design is that the multithreading has high potential to cause unforeseen errors due to how often the variables are being written to and read from. The third design has very little down time overall, however, both threads will frequently stall due to the other locking a variable.

5.2.2. Wall Switch

The design of the software for the wall switch device is one step more complicated than the design for the wall outlet due to the possibility of the state of the circuit changing at any time from via the physical switch. Three designs will be tested for the wall switch device.

5.2.2.1. First Design: Interrupt Based

The first design for the wall switch software is based on the first design for the wall outlet software. When the software begins execution, it will begin by reading the current state of the wall switch, which will be either on or off, and the current power usage. The software then sets the state as a variable and enters the initial waiting phase. During this initial waiting phase, the device is idling until a Bluetooth connection is made and data detailing the WiFi connection is received. The microcontroller will use this data received from the Bluetooth module to configure the WiFi module and attempt a connection. The Bluetooth module will then send the results of the WiFi module's connection test to the device that made the Bluetooth connection. If the WiFi module failed to connect properly, the software will re-enter the waiting phase and wait for a new set of configuration settings to be sent for the WiFi module via the Bluetooth module. If the connection test is successful, the software will notify the device that made the Bluetooth connection, send the information detailing the device's current state to the database, and enter the main process.

While in the main process the software will continuously check the current state of the circuit against its own local state variable. If the state of the circuit does not match the local state variable in the software, the software will update its local variable. Within the process of updating the current state of the wall switch device, the software will send that updated state and the current power usage to the WiFi module to be sent to the database. At the end of the main process, upkeep data will be sent to the server. An interrupt will occur whenever either the Bluetooth or WiFi module receives data. If the Bluetooth module is the module that received data, the software will trigger an interrupt and follow the same process as the initial waiting phase. If the WiFi module is the module that received data, the software will trigger an interrupt and enter the update phase.

During the update phase the software will take the data received from the WiFi module and send signals to the circuit that will alter the state of the hardware

accordingly (either from on to off, or from off to on). The software will then check the state of the circuit, update its own variable as to the new current state, and then send data containing the updated state and current power usage to the database. The software will then re-enter the main phase, and wait for another interrupt to occur.

Figure 5.2.D shows a flowchart for the first software design of the wall switch device. This first design does not differ much from the first design of the wall outlet. The main difference between the two is that the hardware can easily change states outside of the influence of the software. This difference is most apparent during the update phase, where, instead of simply confirming that the desired change happened the software checks the state of the circuit again and updates the software's local state variable to the new state, which could be different than the state that the circuit was just updated to due to the interrupt. The priorities for the interrupts in this design are the same as the first design for the wall outlet, WiFi interrupts are higher priority than Bluetooth interrupts. The downside for this first software design is the interaction of the interrupts with the hardware can be complicated with the hardware able to easily change states at any time.

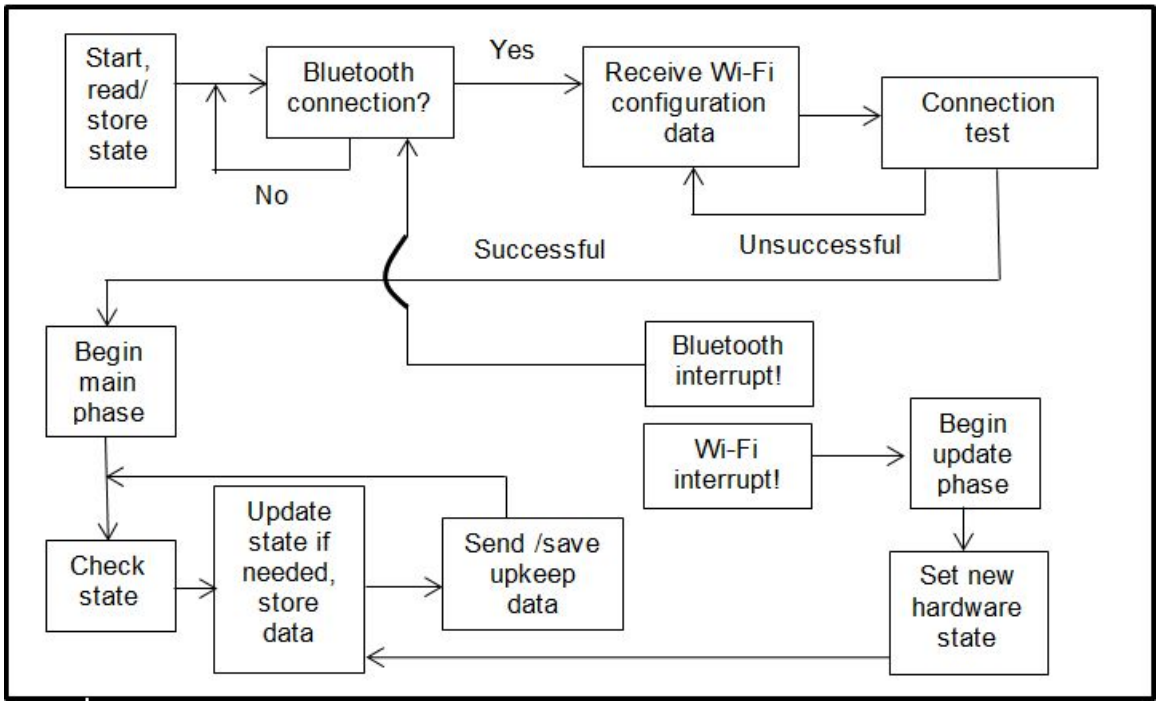


Figure 5.2.D - Flowchart for wall switch microcontroller software design 1

5.2.2.2. Second Design: Procedurally Based

The second design for the wall switch device software is similar to the second design for the wall outlet software. When the software is first executed it checks the current state of the circuit and updates its local state variable accordingly,

then enters the same initial waiting phase as design one, waiting for an interrupt triggered by the Bluetooth module. Once an interrupt from triggered by the Bluetooth module has resulted in a successful configuration of the WiFi module, the software enters the main phase.

The main phase for the second software design of the wall switch is procedural, and combines the update and main phases from designs one. The main phase begins by checking the current state of the circuit against the software's local state variable, and if they differ then the software updates its local variable. The software then reads the power usage of the circuit. The software then sends the local state variable and current power usage to the WiFi module to update the database with the most current state of the circuit. The main phase then checks if there is a connection from a Bluetooth device. If there is a connection from a Bluetooth device then the software receives the data and moves to the update WiFi module configuration function. This function will configure the WiFi module, and then test the connection to the internet. If the connection test is unsuccessful it will notify the device connected via Bluetooth, get new data from the Bluetooth device, and try the connection again, repeating until the connection test is successful. After checking for a Bluetooth connection and taking the appropriate course of action (either do nothing or update the WiFi module), the software then checks for data received from the WiFi module. If new data was received via the WiFi module it changes the state of the circuit accordingly. The main phase then sends upkeep data to the server and loops back to the start, beginning again by checking the current state of the circuit against its local state variable.

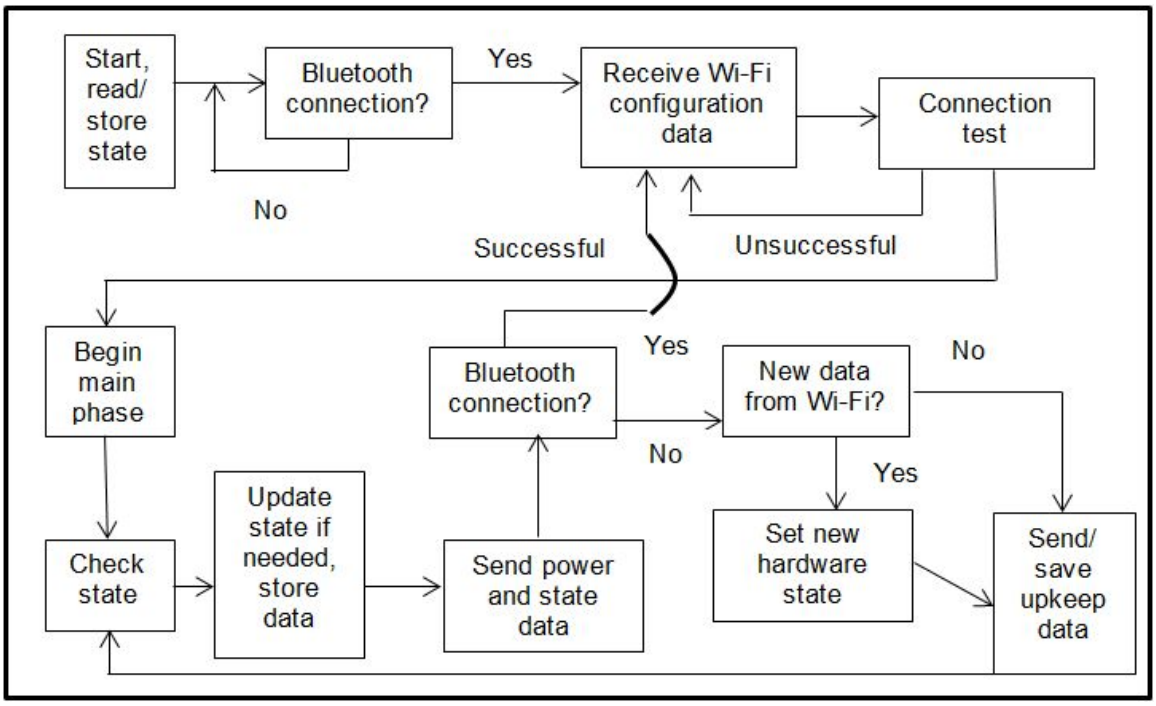


Figure 5.2.E - Flowchart of wall switch microcontroller software design 2

Figure 5.2.E shows a flowchart for the wall switch microcontroller software design two as it is detailed above. The procedural nature of this second design makes sure that only one action is occurring at a time with no possibility of an interrupt suddenly taking over. Design two's simplicity is its greatest benefit; there are no extra cases to consider where an interrupt occurs at an exact point during execution. The downsides come from the constant updating of the data, which uses a greater amount of data and cycles than design one. However, this constant updating is very important for the wall switch as the assurance of an up to date state in the database is of higher importance when the state of the circuit can easily change at any moment. The constant updating also causes a lot of unnecessary instructions, such as always checking for a Bluetooth connection instead of waiting for one to occur.

5.2.2.3. Third Design: Multithreading Based

The third software design for the wall switch device incorporates ideas from the previous two designs. The third design uses multithreading to keep the database constantly up to date while simultaneously staying up to date with the current state of the device. The third design ends up being nearly identical to the third design for the wall switch device. The main difference is that the wall switch device needs an extra check to ensure the physical switch has not been used to change the state of the device since the state was last checked before the process of updating the database.

The third design implements an initial waiting phase similar to the previous two designs. Upon startup, the device will wait for a Bluetooth connection. When a Bluetooth connection is made, the program will wait for WiFi configuration data to be input. When the configuration data is received the program will perform a connection test, if the connection test fails the program will wait for another connection. When a connection is made successfully the program will store the configuration settings, and read both the current power output of the device and the current state of the device (on or off), then store that data locally. The program then splits into two threads: the monitoring thread, and the updating thread. To prevent reading a variable while it is being written to, variables will be locked while being written to. If a thread attempts to read a locked variable, it will wait until the variable is unlocked to read the variable.

The first thread will be referred to as the monitoring thread, because it monitors the current state of the device and updates data accordingly. The monitoring thread begins by checking both the state and power output of the device, and updating both of those local variables. While updating both of those variables, they will be locked to prevent the updating thread from reading them while they are being written to. The variables are then unlocked, allowing them to be read. The monitoring thread then moves to check if there is a Bluetooth connection. If a connection is found, the program locks the WiFi configuration variable,

preventing the updating thread from sending data to the database. The program waits for WiFi configuration information from the Bluetooth connection. Upon receiving WiFi configuration information, a connection test will be performed. If the connection test is successful, the WiFi configuration data will be updated, the variable unlocked, and the thread will resume. Should the connection test fail or the Bluetooth connection be lost, the thread will unlock the WiFi configuration variable without updating it, and then resume. Upon resuming, the thread loops back to the beginning and checks the current state of the device again.

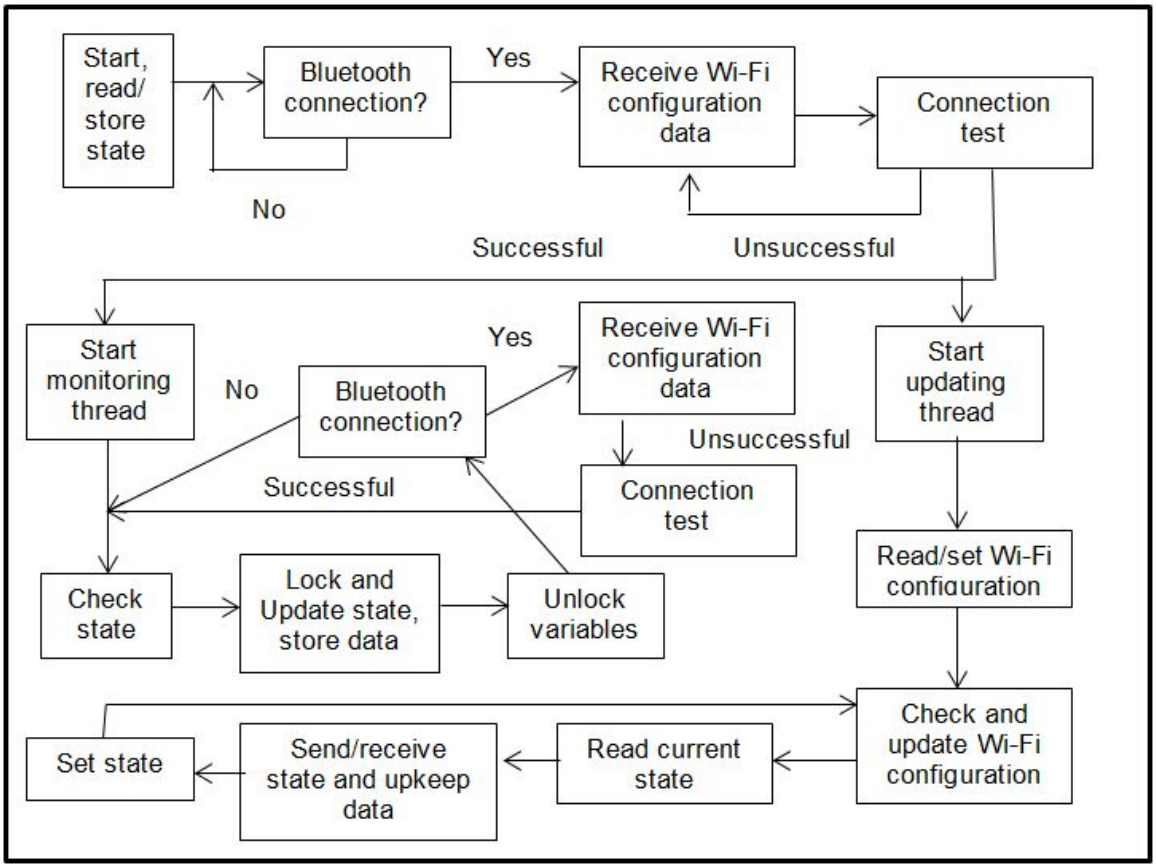


Figure 5.2.F - Flowchart of wall switch microcontroller software design 3

The second thread will be referred to as the updating thread, because it both updates the database and updates the device. Upon startup, the updating thread reads the WiFi configuration variable, sets its own local WiFi configuration variable, and then moves to the main loop. Within the main loop, the thread reads the main WiFi configuration variable, and checks it against its own. If the two variables are different, the variable local to the updating thread will be updated to match the WiFi configuration variable. The WiFi configuration variable local to the updating thread is the data used for the connection to the database. The thread then reads the state and power output variables. The thread then checks the current state of the device against the state variable it just read, and if they differ the software locks the state variable and updates it. If the two do not

differ this update is skipped. The thread then sends the state and power usage variables to the database using the WiFi configuration settings stored in the configuration variable local to the updating thread. Next the updating thread checks if data was received from the online application to change its state. If state change data was received, the thread sends a signal to the circuit to change its state, then locks and updates the state variable. The thread then repeats back to the beginning of the main loop.

Figure 5.2.F shows a flowchart for the third design of the microcontroller software for the wall outlet device. The main benefit of the third design is that it keeps the database constantly up to date with the state and power usage of the wall outlet. The main drawback to this design is that the multithreading has high potential to cause unforeseen errors due to how often the variables are being written to and read from. The third design has very little down time overall, however, both threads will frequently stall due to the other locking a variable.

5.2.3. HVAC controller

The HVAC controller is the most complex of the devices due to the greater amount of variables to consider as part of the state of the device and the increased number of outputs needed to update all of the parts of the device correctly. 2 designs will be tested for the software of the HVAC controller.

5.2.3.1. First Design: Interrupt Based

The first design for the HVAC controller utilizes interrupt service routines. There will be an interrupt for when any button is pressed on the device, an interrupt for a Bluetooth connection, and an interrupt for data received from the database via WiFi. Priority will be given first to the interrupt triggered by button presses, then to Bluetooth connection interrupts, then to WiFi data interrupts. A data structure will be created to hold all of the current settings.

Upon startup the device will look for a Bluetooth connection. When a Bluetooth connection is established, the software will read the current state and wait to receive WiFi configuration data. Once WiFi configuration data is received, an internet connection test will be attempted. If the connection test fails, the software will wait for new WiFi configuration data and perform another connection test with the new data. After a successful connection test, the WiFi connection data will be saved. The software will then initialize the display to show all sections on, and wait to receive initial settings from the Bluetooth device. Once initial settings are received, the settings data structure will be updated to those initial settings. The display will then be updated to show these new settings. Then a signal will be sent to the circuit to set the circuit to the initial settings. The software will then transition to the main phase.

In the main phase, the software will read the settings data structure and send the current settings to the database using the saved WiFi configuration. If the connection to the database is lost, the display will be updated to notify the user such an error has occurred. The main phase repeats this action while waiting for one of the three interrupt service routines to occur.

When a button is pressed on the device, it triggers the button press interrupt service routine. Which button was pressed is stored in a temporary variable, and then a switch statement determines the proper action. Whichever button was pressed, the settings data structure will be updated to reflect the proper change. The microcontroller will then send a signal to the circuit to set the state of the device to the settings in the settings data structure. The display function will then be called and the display will be updated to reflect the new changes in the settings data structure.

When a Bluetooth connection is found, it triggers the Bluetooth connection interrupt service routine. The software begins by waiting to receive new WiFi configuration data. Once the WiFi configuration data is received a connection test is performed. If the connection test fails, the software returns to waiting for new WiFi configuration data. When the connection test is successful, the software saves the new WiFi configuration data. If the Bluetooth connection is lost at any point, the interrupt service routine ends without updating the WiFi configuration data.

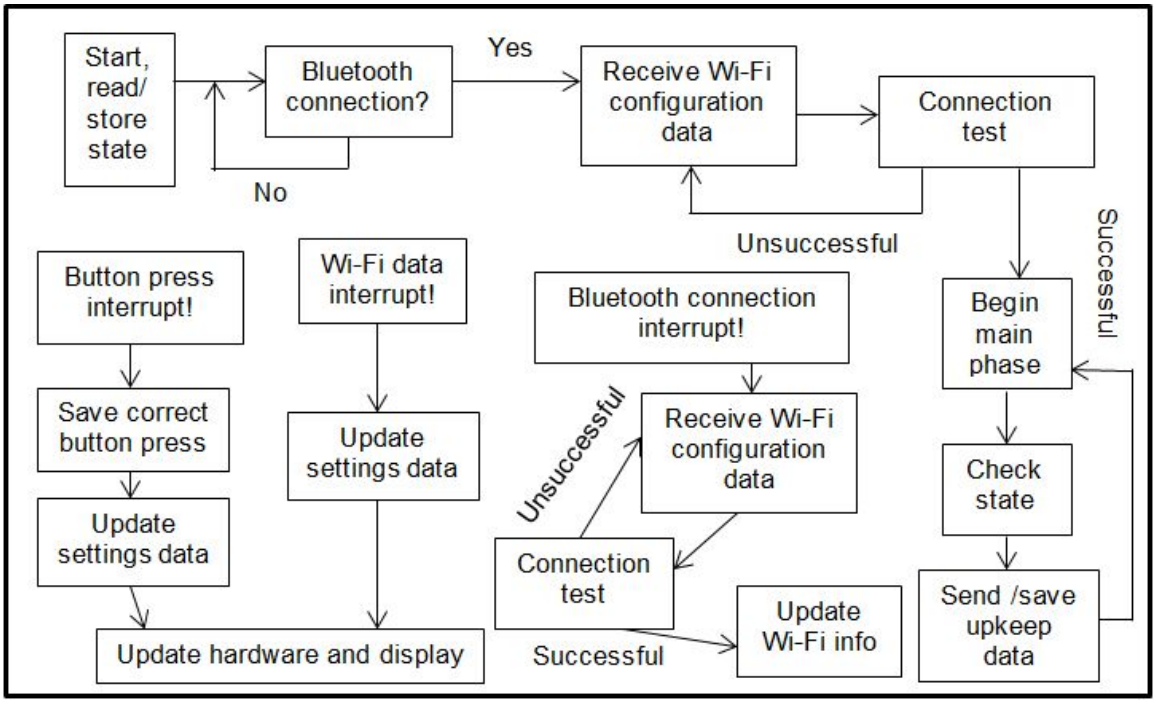


Figure 5.2.G Flowchart of HVAC Controller microcontroller software design 1

When update data is received from the database via the WiFi module, it triggers the WiFi data interrupt service routine. The software overwrites the settings data structure with the newly received settings data. The microcontroller then sends a signal to the circuit to update the state of the circuit to what is defined in the settings data structure. Then the display function is called and the display is updated to reflect the changes made to the settings data structure.

Figure 5.2.G shows a flowchart for the first design for the HVAC controller microcontroller software. The benefit of the interrupt service routine based design is that the device is never stuck performing an action that will turn up false, leaving the software ready for any possible action to occur. The downside to this is that the software cannot perform more than one action at a time, which can leave the software stuck in a high priority interrupt.

5.2.3.2. Second Design: Multithreading Based

The second design for the HVAC controller incorporates multithreading to process all actions simultaneously. The settings data structure and the WiFi configuration variable will both be locked by threads to prevent reading from and writing to them while they are being written to. If a thread attempts to read from or write to a locked variable, then the thread will wait until the variable is unlocked to interact with it. The settings data structure in particular requires being locked for longer than its initial write due to it being altered by multiple threads.

On startup the software reads the current state then waits for a Bluetooth connection to be found, following the same procedure as the first HVAC controller design. When a Bluetooth connection is established, the software will wait to receive WiFi configuration data. Once WiFi configuration data is received, an internet connection test will be attempted. If the connection test fails, the software will wait for new WiFi configuration data and perform another connection test with the new data. After a successful connection test, the WiFi connection data will be saved. The software will then initialize the display to show all sections on, and wait to receive initial settings from the Bluetooth device. Once initial settings are received, the settings data structure will be updated to those initial settings. The display will then be updated to show these new settings. Then a signal will be sent to the circuit to set the circuit to the initial settings. The software will then split into 3 threads: the WiFi thread, the Bluetooth thread, and the button thread.

The WiFi thread handles all communication with the database. The thread starts by reading the WiFi configuration variable. Then the thread reads the settings data structure, and sends the data to the database. The thread then checks for any received state change data. If state change data was received then the new settings data structure is copied to a local temporary version of the settings data structure, then the main settings data structure is locked, and then the main

settings data structure is updated to reflect the changes. The microcontroller then sends a signal to the circuit to set the circuit state to the state detailed in the temporary settings data structure. After the circuit is updated the display is updated to match the current state of the temporary settings data structure. Then the main settings data structure is unlocked. The thread then loops back to the start and begins reading the WiFi configuration data and settings data structure.

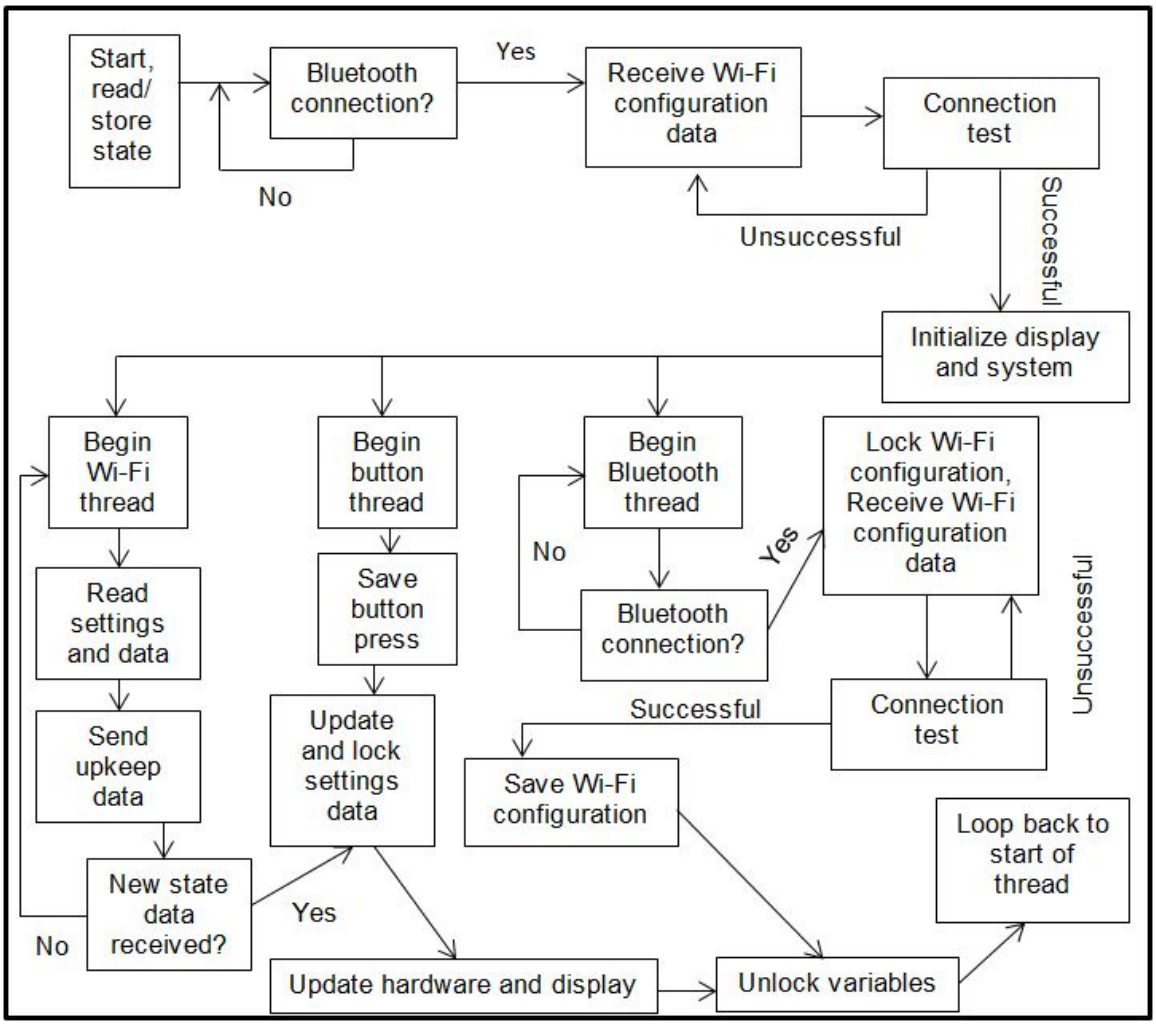


Figure 5.2.H Flowchart of HVAC Controller microcontroller software design 2

The Bluetooth thread handles all Bluetooth connection communications. The thread starts by checking if there is a Bluetooth connection. If a Bluetooth connection is found, then the software locks the WiFi configuration variable and waits to receive new WiFi configuration data from the connected Bluetooth device. Once the thread has received WiFi configuration data it will perform a connection test. If the connection test fails the thread will wait for new WiFi configuration data and will run the connection test again. When a connection test is successful, the thread updates the WiFi configuration data and unlocks it. The

thread then loops back to the start and waits for a Bluetooth connection to be found.

The button thread handles all interactions with the buttons on the device. When a button is pressed, which button was pressed is stored in a temporary variable. Then a switch statement uses the temporary variable to determine the correct action. For whichever button was pressed, the new settings data is copied to a temporary local version of the settings data structure. The main settings data structure is then locked and updated to match the new settings. Next the microcontroller reads the temporary settings data structure and sends a signal to the circuit to update its state to match those settings. The display is then updated to the current settings using the temporary settings data structure. The main settings data structure is then unlocked. The thread then loops back to the start and waits for a button to be pressed.

Figure 5.2.H shows a flowchart for the HVAC controller microcontroller software design two. The main advantage of this design is that by using multithreading the software is able to respond to any action immediately. The greatest drawback is that due to the need to share the settings data structure, if one of the threads gets locked up, the software comes to a halt. Like the first design this one is able to frequently and constantly update the database, ensuring all data shown in the online application is current.

5.2.4. Powerlock

The powerlock is the simplest of the devices in this project from the viewpoint of the microcontroller software. There is no power usage monitoring tied to the powerlock as only the state of the device (locked or unlocked). Three designs will be tested for the powerlock.

5.2.4.1. First Design: Procedurally Based

The first design for the powerlock software is entirely procedural. There are only two phases to this design: the initial waiting phase, and the main phase. The initial waiting phase is purely for the startup of the software, and once the software enters the main phase it will never return to the initial waiting phase unless the device is disconnected from electrical power or restarted.

On startup, the software begins with the initial waiting period, reading the current state and checking for a Bluetooth connection. Once a Bluetooth connection is established, the software waits for WiFi connection configuration data to be sent. Once the WiFi configuration data is received a connection test is performed. If the connection test fails, the software returns to waiting for new WiFi configuration data. When the connection test is successful, the software reads

the current state of the device and stores it in a local variable. The software then moves to enter the main phase.

In the main phase the software checks the state of the device, and updates its local variable to the current state. The software then sends the current state variable to the database using the WiFi configuration data. Next the software checks if any change of state data was received from the database. Then the software checks for a Bluetooth connection. If a Bluetooth connection is found, the software waits for WiFi configuration data to be received. When WiFi configuration data is received the software follows the same procedure as the initial waiting phase, performing a connection test and only updating the data if the test is successful. If change of state data was received the microcontroller sends a signal to the circuit to change the state of the device. The software then returns to the start of the main phase, checking the current state of the device. At the end of the main phase upkeep data is sent to the server and the process loops back to the beginning of the main phase.

Figure 5.2.I shows a flowchart for the first design of the powerlock microcontroller software. The simplicity the first design is its greatest advantage. The software is quick to write, easy to understand, and thus easy to debug. The main disadvantage of this design is that it has a comparatively slow response time for changes done by the online application.

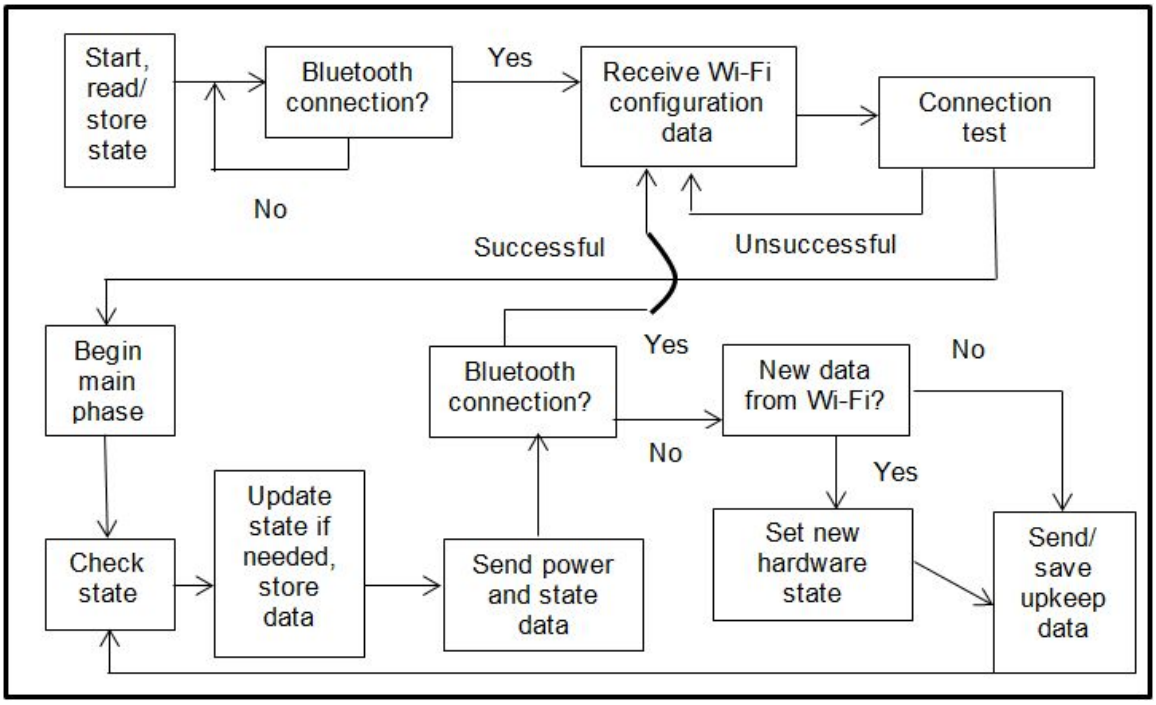


Figure 5.2.I - Flowchart for powerlock microcontroller software design 1

5.2.4.2. Second Design: Interrupt Based

The second design for the powerlock uses interrupt service routines to signal any change in the device. This design allows for minimal interaction with the online application, and cuts down on the number of unnecessary checks from the first design. The second design has 3 phases: Bluetooth connection phase, main phase, and WiFi update phase. Interrupt priority is given to the Bluetooth connection phase interrupt over the WiFi update phase interrupt.

Upon startup the software reads and stores the current state, then waits for a Bluetooth connection. When a Bluetooth connection is found, the software enters the Bluetooth connection phase. The Bluetooth connection phase is similar to the initial waiting phase from the first design. In the Bluetooth connection phase the software waits to receive WiFi configuration data. When WiFi configuration data is received, an internet connection test is performed. If the connection test fails, the software waits to receive new WiFi configuration data and repeats the test. Once a connection test is successful, the software stores the WiFi configuration data. The software then checks the current state of the device, stores that state in a local variable, and moves to the main phase.

The main phase is where the software will spend a majority of its running time. At the start of the main phase the software checks the state of the device and compares it to the state it has stored in its own variable. If the state of the device is different from the state stored in the variable, then the main phase updates its current state variable. The main phase then sends the updated state data to the database. If the state of the device and the state in the variable are not different, or the main phase has just sent updated state information to the database, the main phase sends upkeep data to the server and loops back to the start, checking the state of the device again. Should a Bluetooth connection be established at any point, an interrupt service routine will trigger, and the software will enter the Bluetooth connection phase detailed previously. When state change data is received via the WiFi module, an interrupt service routine will trigger, and the software will enter the WiFi update phase. When either interrupt service routine has finished, the main phase will be restarted.

The WiFi update phase occurs when state change data is received from the online application. During the WiFi update phase, the software sends a signal to the circuit to change the state of the device. The WiFi update phase then checks the state of the device and updates the current state variable.

Figure 5.2.J shows a flowchart for the second design of the powerlock microcontroller software. This design holds the advantage of only acting and updating data when necessary. That key advantage causes this designs main disadvantage, which is infrequent updates to the database that lead to the possibility of outdated data. Due to the structure of this design, if an error with the

WiFi connection should occur, there would be no immediately apparent way to notice it.

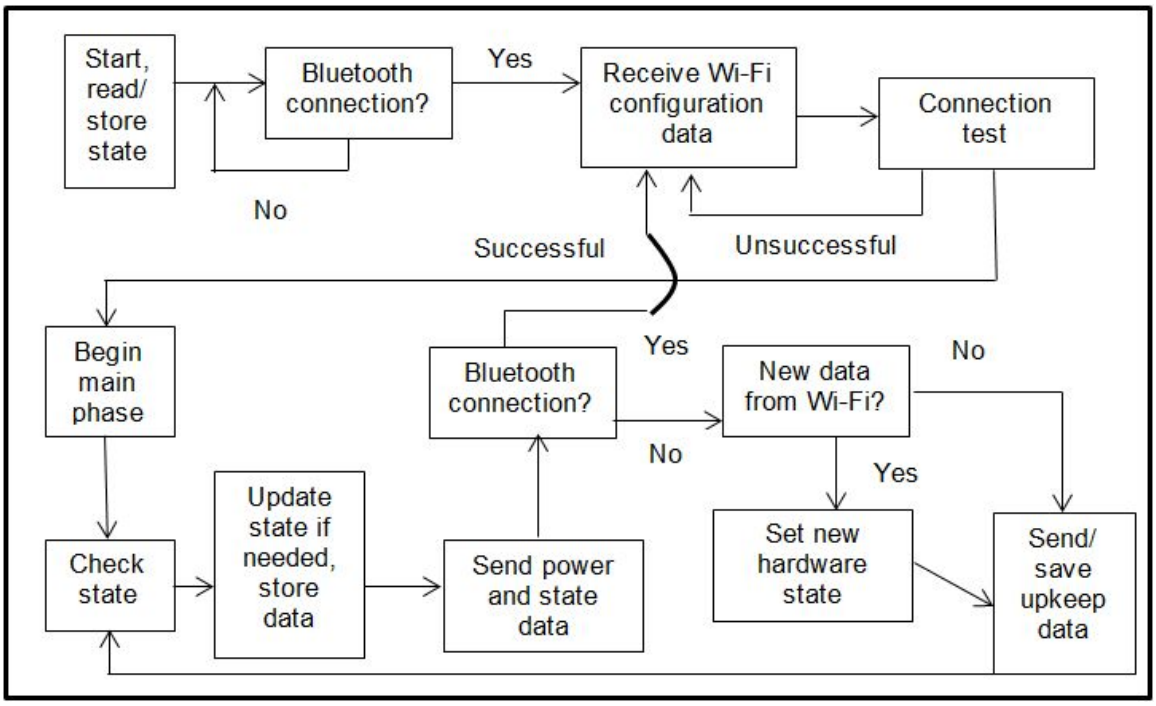


Figure 5.2.J - Flowchart for powerlock microcontroller software design 2

5.2.4.3. Third Design: Multithreading Based

The third design for the powerlock microcontroller software incorporates multithreading in order to make frequent communication the database. Threads will lock common variables when they are being written to to ensure that they cannot be read from while being written to. If a thread attempts to read from a locked variable, the thread will wait until the variable is unlocked to read from it.

Upon startup the software will enter an initial waiting phase, reading the current state and waiting for a Bluetooth connection to occur. When a Bluetooth connection is found the software waits to receive WiFi configuration data from the connected Bluetooth device. Once WiFi configuration data is received, an internet connection test will be attempted. Should the connection test fail, the software will wait for new WiFi configuration data and repeat the connection test. When the connection test is successful, the WiFi configuration data is stored. The software then splits into two threads: the Bluetooth thread, and the WiFi thread.

In the Bluetooth thread the software checks for a Bluetooth connection. If a Bluetooth connection is found the thread locks the WiFi configuration data variable and waits for new WiFi configuration data to be received. When new

WiFi configuration data is received an internet connection test is attempted. If the connection test is successful the WiFi configuration data variable is updated and unlocked. If the test is unsuccessful, the thread waits to receive new WiFi configuration data, and then runs a connection test using the new data. The thread then loops back to the beginning, checking for a Bluetooth connection.

In the WiFi thread, the software reads the WiFi configuration data variable. The thread checks the current state of the device. The thread then sends that current state to the database using the WiFi configuration data. Then the thread checks for any received state change data. If state change data was received, the microcontroller sends a signal to the circuit to change the state of the device.

Figure 5.2.K shows a flowchart for the third software design for the powerlock microcontroller. This design allows for constant communication with the database that is not halted by checking for a Bluetooth connection. The database is always up to date using this design, and should an error occur, the extended lack of communication would be a quickly noticeable sign.

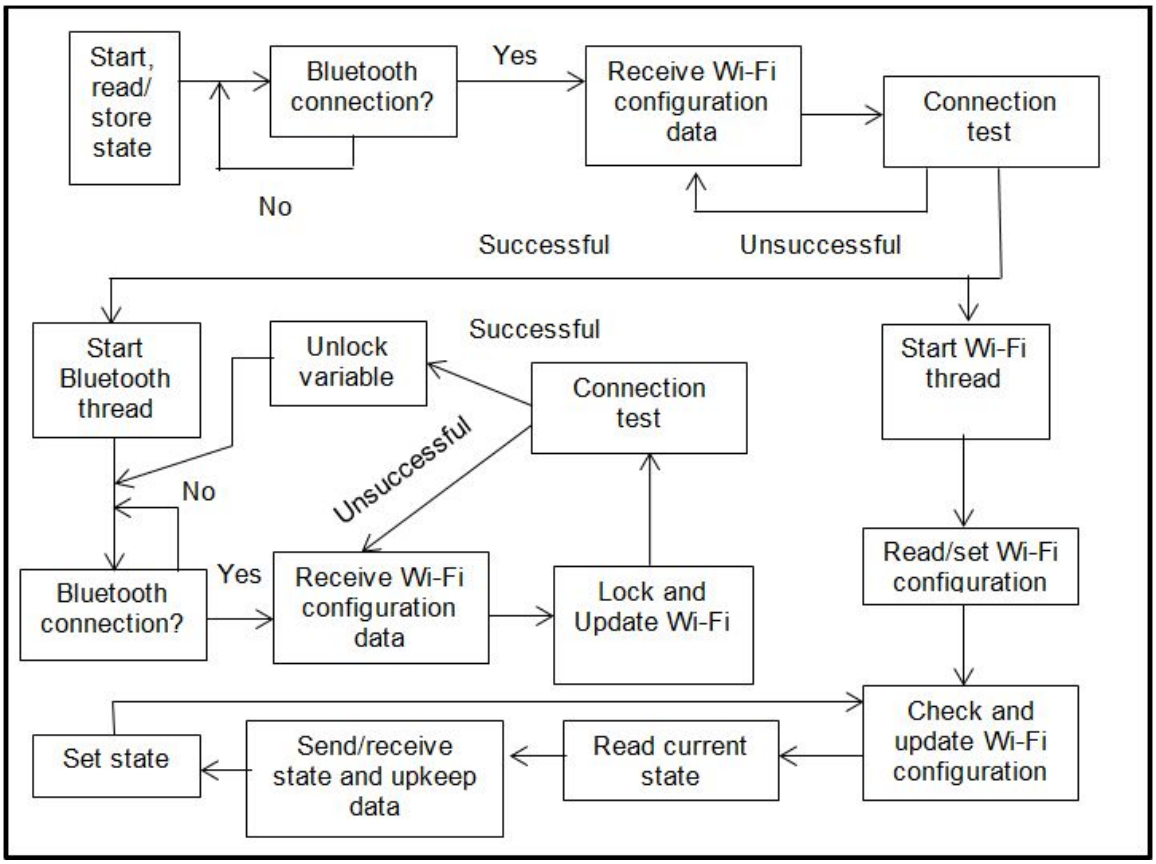


Figure 5.2.K - Flowchart for powerlock microcontroller software design 3

5.3. Application Software Design Plan

In order to provide the best user experience possible, automated devices should be able to be controlled anywhere in the world, regardless of where you are. Sometimes a light is left online or a door unlocked. There is a piece of mind knowing that with a quick glance at an app you can see the status of electronics in your home, and if something is amiss be able to modify their status remotely. The following figure gives an application design overview of the entire application flow.

To achieve a great user experience, the application is divided into two separate applications, the backend application running on cloud servers and the frontend application running on mobile devices (as well as a cloud hosted web-based control panel usable on desktops and unsupported mobile devices that still have browsing capabilities). By the term mobile device, we are referring to a wireless device containing either cellular or WIFI connection as well as Bluetooth connectivity. The backend application will communicate with both the frontend application as well as home automation devices. The frontend application will connect with the backend application over API to perform actions on home automation devices. The extent of this division is shown in the application overview in figure 5.3.A.

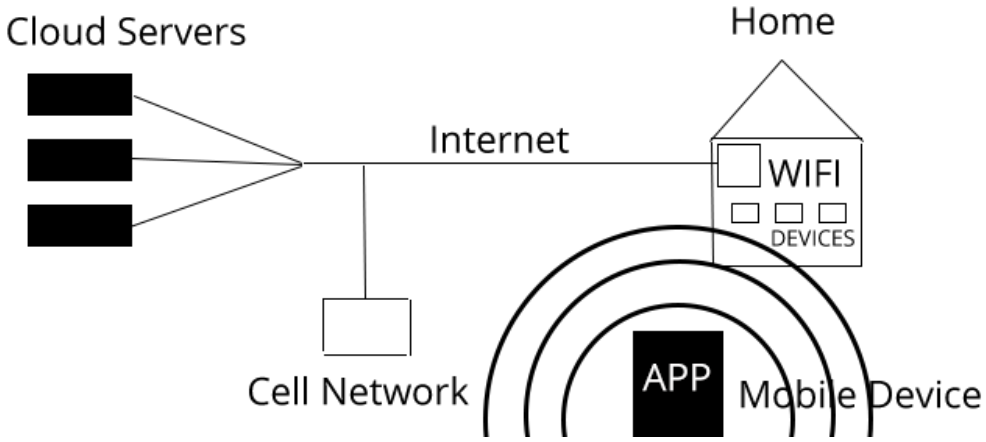


Figure 5.3.A - Application Overview

5.3.1. Cloud Servers

The home automation application, both frontend and backend will exist across multiple cloud servers. Using multiple cloud servers across multiple locations increases redundancy in the case a single virtual instance fails. As noted in our constraints, it is important that the service remain online at a near 100% availability since every home automation system depends upon a reliable API to function as intended.

For this home automation system, DigitalOcean is where the cloud will be hosted. It's cost effective since each "droplet" (the term DigitalOcean uses for each virtualized instance being ran) costs at minimum $5/month. Since the point of this project is a proof of concept, the amount of resources necessary to run the service will not be much at all. At the base price of $5 each droplet gives 512MB memory, 1 core of processing power, 20GB SSD storage, and 1TB of data transfer. This is more than enough for what is initially needed to run this service. As the service grows the plans can be increased vertically, and more instances can be spun up to scale horizontally.

In figure 5.3.B, the general layout of each virtualized cloud instance is given. Each cloud server will be configured with a minimal installation of Debian 8 (Jessie), 64-bit. Since we have a limited amount of memory, a minimal installation is preferred. Under a minimal installation, only system essentials are installed. This means that most services need to be installed post-installation, including the web server software, database software, and runtimes required to run our application.

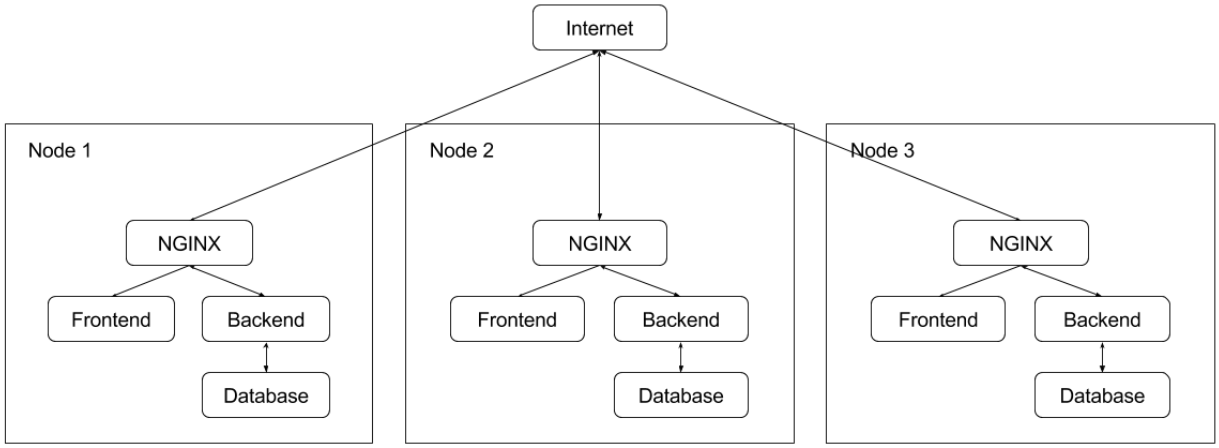


Figure 5.3.B - Cloud Servers Overview

The webserver will be reachable from the internet by both the frontend clients as well as the automated devices. We are choosing to use NGINX as the web server for this project. NGINX is a light, barebones, and fast webserver that is widely replacing Apache in many infrastructures. NGINX will be configured to reverse proxy traffic to the backend service as well as serve the frontend cloud hosted web application. Native apps for iOS and Android will interface with only the backend API, not the frontend application being served by NGINX.

5.3.2. Backend

The backend of the application will be run solely on cloud servers and functions as an accessible programming interface (API) for the frontend clients and

automated devices to communicate with. The backend is separated between API and database operations, as shown in figure 5.3.C.

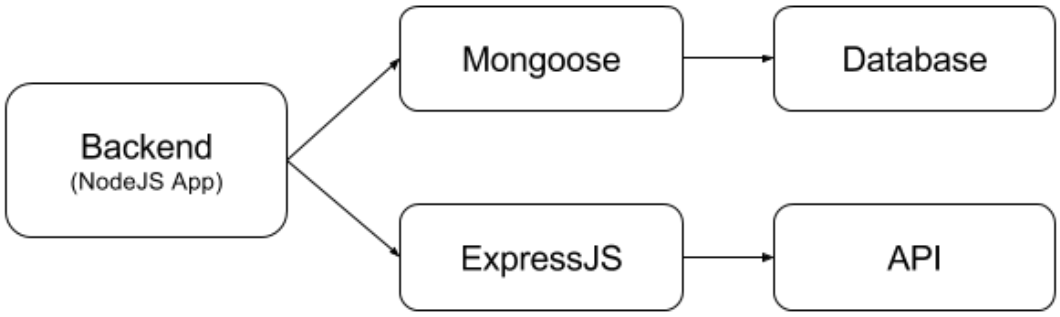


Figure 5.3.C - Backend overview

5.3.2.1. API Design

In order to produce a modern API that can be ingested easily by the client side applications, the API will be architected in a Representational State Transfer (REST) way.

A REST API has the following characteristics:

Uniform Interface – The server responds in a practical and predictable way when accessing isolated resources on the API.

Client-Server – In order to more easily share data across clients in real-time, clients do not store long term data (only current stateful data). Since the server doesn't render content for the client and only provides data, it allows the server to more easily scale

Stateless – The server never has to store any state data apart from authentication credentials in a session. Clients hold their own states and fetch information from the server to update their states as needed.

Cacheable – The API can be cached when frequently accessed in order to conserve processing power and bandwidth.

Layered System – The API may run a layered system where multiple servers handle requests. Responses from these different servers should not be discernable by the client.

Since the client applications are utilizing JavaScript, JavaScript Object Notation (JSON) formatting will be used for API output since it is a native format to the runtime.

Simple Example of JSON format:

```
{
      “key” : “string value”,
      “number” : 1,
      “float” : 1.1,
      “key” : [ “array”, “values” ],
      “nested_object” : {
            “key” : “value”
      }
}
```

As you can see from the example above, the formatting of JSON is native in JavaScript and it allows for good performance when encoding and decoding data in this format within JavaScript. Almost all languages have JSON libraries as well, since JSON is now the most widely used API format. With a fully RESTful JSON API, the client side frontend application will be able to run separately from the backend with ease.

To create our RESTful JSON API, we’ll be using Node JS to create a backend service.

Node JS will be compiled and ran under a restricted user account on each cloud instance. This will ensure that a vulnerability in our software does not compromise the entire server instance. Our backend service will be deployed to the cloud servers using GIT, a version control system for software development.

In the GIT code repository for our backend NodeJS application, dependencies for the codebase will be listed in an Node Package Manager (NPM) “package.json” file, which when deployed will automatically install all dependencies to run the application.

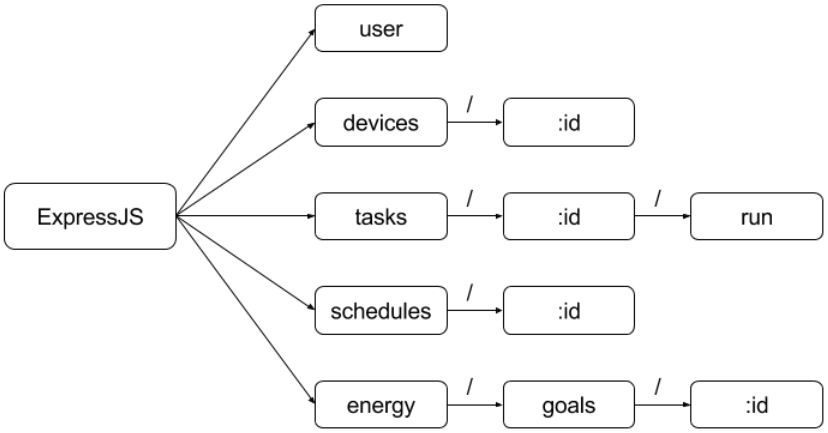


Figure 5.3.D - ExpressJS Route Path Progression

As shown in figure 5.3.D, the codebase will primarily consist of the Express JS library, which is a web application routing framework. This will allow us to create a RESTful API by generating multi-stage routes for each resource we wish to modify. By nesting paths, each of the resources are scoped to their own modules. Each resource on the API will consist of specific data models that are being retrieved, modified, and deleted.

Below is a list of the API data models that will be integrated for our API. Each data model lists supported HTTP verbs as well as the resource endpoint it can be accessible from. In the endpoint, brackets are optional and parameters are prefixed with a colon.

API Data Resource Models:

1. User – HTTP GET/POST/PUT/DELETE /user

   The user data model will consist of personal information pertaining to the registered user. This means information like email address, name, password, and other personally identifiable information that is needed for the home automation system.

   In addition to private, personal information, the data model for users will also hold an array of sub-users. More than one person usually exists in a household. Rather than having everyone share an account, we can have household owners invite other people to join their household.

   To create a user, an HTTP POST request is sent to the endpoint. The body of the request would contain an email address and password, which is what will be used for signing in users. The password will be encrypted using bcrypt on the server side for secure password storage. An API access token will be returned upon successful login, which will be used to negotiate future requests. The client is then responsible for finishing the user account setup by letting them add devices to their account.

   To login as a user, an HTTP Post request is sent to the endpoint. The body of the request would contain an email address and password. Upon successful login, an API access token will be returned upon successful login, which will be used to negotiate future requests.

   The user model can be a retrieved using an HTTP GET request. The password is not returned in the data model response for security purposes.

To update a user, a simple HTTP PUT request can be sent to this endpoint containing updated fields for the data model.

To delete a user account (that is to say deregister from the service), a simple HTTP DELETE request can be sent to this endpoint.

2. Device – HTTP GET/POST/PUT/DELETE /devices[/:id]

   The device data model will consist of automated device information and settings.

   For each device, the data model will contain a unique hardware identifier. This identifier will be used to negotiate commands to each automated device as well as update state information from these hardware devices (like if a light is on or off).

   Each device can have a variety of state settings that can be toggled, so the API is responsible for handling each of the different cases it might encounter when swapping the states of many different devices. For the light switch as well as the wall outlet, the manual switch has a state as well as the device attached to the switch. For the door lock, the door is only a temporary unlock, so state is not a permanent change. For the HVAC, the temperature, fan, and cooling/heating settings are all different states that must be taken into consideration.

   A paginated list of devices can be retrieved with an HTTP GET of /devices.

   To add a new device, a POST request is sent to /devices containing the hardware identifier of the device being added. The API will then communicate with the automated device in order to finish setup.

   To update a device, a PUT request is sent to /devices/:id, where “:id” is the database ID of the device. When a stateful value of the device changes, the API will then communicate with the automated device in order to update the state on the device (to perform an action like turning on a light).

   To delete a device, a DELETE request is sent to /devices/:id, where “:id” is the database ID of the device.

3. Task – HTTP GET/POST/PUT/DELETE /tasks[/:id[/run]]

   There is a need for grouping together devices into particular tasks since each device can only control items directly plugged into them. Because of this, a tasks data model is used to group devices into user configurable tasks.

Each task model contains an array of device objects. Each of these device objects contain the database ID of each device being controlled by the task, as well as states being changed for each device. This allows users to define specific stateful values for each and every device in the list, since each device may have different states that can be controlled (for example, someone could potentially mix a door lock with a light switch, and then lock the door and turn on an outdoor light with the task).

A paginated list of tasks can be retrieved with an HTTP GET of /tasks. An array of task models will be returned by this endpoint, and each task model will contain a nested populated device object containing detailed information about the devices in each task as well as the task and device states.

To add a new task, a POST request is sent to /tasks containing an array of devices objects formatted similarly to the above device object description. The user interface will need to populate a list of devices to to choose from using the devices endpoint documented above.

To update a task, a PUT request is sent to /tasks/:id, where ":id" is the database ID of the task. The body of the request should contain the full array of device objects being modified as well as the status of the task.

To run a task, a POST request is sent to /tasks/:id/run, where ":id" is the database ID of the task.

To delete a task, a DELETE request is sent to /tasks/:id, where ":id" is the database ID of the task.

4. Schedule – HTTP GET/POST/PUT/DELETE /schedules[/:id]

   Hardware timers are still in use by many households still to this day. They usually hold a daily or weekly routine on which to turn a device on or off. In the case of our home automation system, the schedule is created to run a task (the data model mentioned above). These schedules will repeat on whatever interval desired by the user.

   A paginated list of schedules can be retrieved by sending a GET request to /schedules. An array of schedule data models will be returned. Each schedule model gives information about when it will next run as well as the current state of the task and devices in the task. The task and devices for each schedule will be populated into nested objects of each schedule data model in the array.

To add a new schedule, a POST request is sent to /schedules. The body of the request must contain the interval on which to repeat the task as well as the task database ID that is to be executed.

To update a schedule, a PUT request is sent to /schedules/:id, where ":id" is the database ID of the schedule. The body of the request should contain the updated schedule model.

To delete a schedule, a DELETE request is sent to /schedules/:id, where ":id" is the database ID of the schedule.

5. Energy – HTTP GET/POST/PUT/DELETE /energy[/goals[/:id]]

   Energy management is an important part of reducing energy footprint of households. Our home automation system automatically monitors the power usage being reported by automated devices in a home and allows users to set goals, graph power usage, and set alerts for when they break a defined limit.

   A list of energy management graphs and limits can be retrieved by sending a GET request to /energy. The graphs are returned in an array that can be rendered into a graph client side.

   To update energy management limits and alerts settings, a PUT request is sent to /energy with the changes set in the request body.

   Goals are a bit more complex in the API. Every goal has a set value of energy usage used over a specified time period. These goals will be computed based on the energy usage per day, and feedback will be able to be generated based on goal progress. For example, if you set a goal of 2000 kWh for a month and are using 150 kWh a day, the goal can provide you feedback that you will not meet the goal at the current rate of usage, and then try offering feedback on how to conserve electricity for the remainder of the time frame left.

   By allowing users to setup multiple goals dynamically, they can set long term goals for themselves to gradually reduce their environmental footprint.

   To list energy management goals, a GET request is sent to /energy/goals. An array of goal objects is then returned. Each goal object contains an array of graph data for each goal progress thus far, percentage elapsed until goal completion, and other settings like whether to alert if a goal may not be reached.

To update a goal, a PUT request is sent to /energy/goals/:id, where ":id" is the database ID of the goal. The body of the request should contain the updated energy goal model.

To delete a goal, a DELETE request is sent to /energy/goals/:id, where ":id" is the database ID of the goal.

5.3.2.2. Interfacing with Microcontrollers

Each microcontroller will interface with both the client-side application and the server-side backend application.

For the server-side backend interface, the microcontrollers will connect to the API using a persistent keepalive socket over a secure TLS encrypted connection. The communication between the API and automated devices is shown in figure 5.3.E.

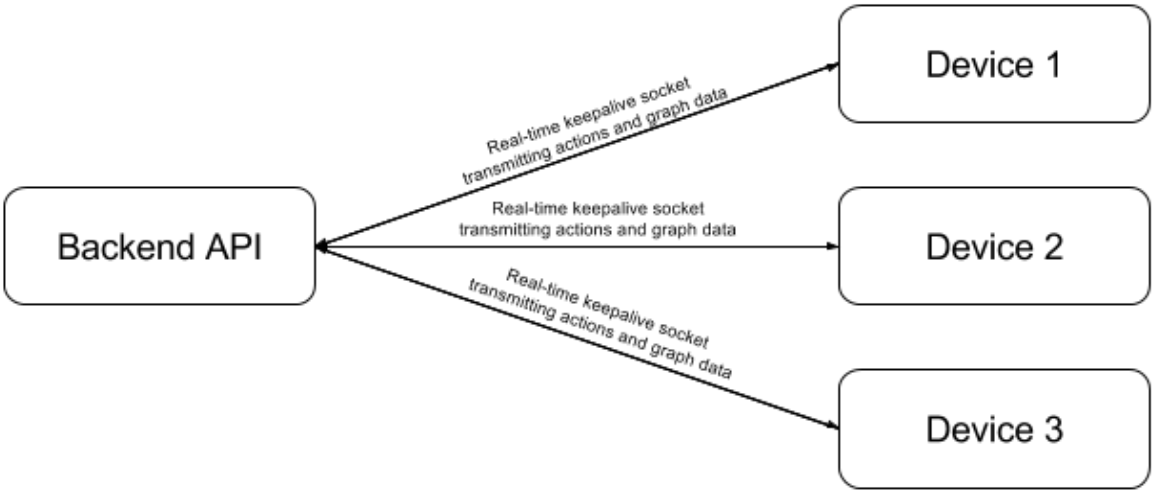


Figure 5.3.E - Visual Depiction of API <-> Device Communication

The format of data sent through this socket is as follows:

```
[device_id] [action] [state_name] [state_value]\n
```

where:

device_id – the hardware device ID
action – the action type to perform, possible values are set and log
state_name – the name of the stateful data being set or logged
state_value – the value of the stateful data being set or logged
\n – new line character (used as a boundary between messages)

For the API to update states on microcontrollers, it would set action to “set” and then send the state name and value that needs to be changed.

For the automated device to report usage (like power usage), it would set the action to “log” and then send the state name and value that needs to be logged.

While the REST-ful API is serving JSON, because the data sent to and from the microcontrollers is very basic there is not a need for extravagant data types here. We can conserve bandwidth and make parsing the data easier for microcontrollers by simplifying the communication between them and the API.

The microcontroller is responsible for establishing and keeping maintained a stable connection to the API. In order for the microcontroller to establish a connection to the API over a socket, it needs to authenticate using a “set” action request. This would require the device to share the secret value that was given during device registration. If the secret value is incorrect, the device will be disconnected from the socket with an invalid secret error. If the secret value provided is correct, the device will stay connected.

Once connected, the automated device will report usage information on an interval to the server. This allows us to aggregate time series graphs for energy usage as well as maintain goals and limits set by the user in the application. In addition, internet service providers (ISPs) like to close idle connections. If data isn’t constantly going through the connection, many ISPs will close the connection to free up another port on their routers for someone else to use. In order to keep the connection alive, data must be reported somewhat frequently (like once every minute).

Some automated devices also change state outside of the application. For example, the light switch may be turned on at any time manually. In order to retain the state of automated devices within apps, microcontrollers will log these state changes back to the API. As a result, the API will be able to maintain each device's state reliably in the database.

5.3.2.3. Database Design

Database design is a crucial aspect of any web application. For our application, an open source document storage database called MongoDB will be used. MongoDB allows us to store object models directly into the database, which allows us to avoid transforming database structure into object models within the application. MongoDB is also highly scalable and useful for its quick processing of writes (through use of journaling) and its aggregation abilities for processing real-time data streams.

In order to ensure a stable cloud hosted database, we need to ensure our data storage is redundant. MongoDB allows you to configure servers into replica sets, which automatically assign a single instance as a “primary” server and sync data to “secondary” servers. In the event of a server outage, MongoDB automatically chooses a new primary and everything continues functioning normally.

For our cloud infrastructure, we will be choosing to run 3 instances of MongoDB in a replica set. This will ensure that even if one or two of the database instances are down or corrupts the data will remain available for use by the applications. Each of these instances will communicate with each other over a TLS encrypted socket using key-based authentication. The operation of MongoDB both in normal operation as well as failover is depicted in figure 5.3.F.

One important caveat to using MongoDB is that it does not enforce a document schema structure. In order to ensure consistent object representation, object model classes must be designed to maintain the structure for each collection of data we wish to store in the database.

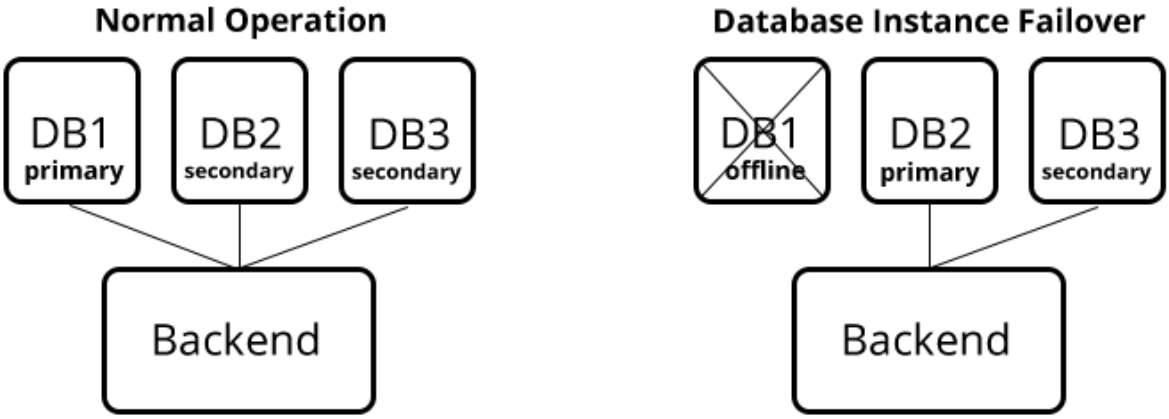


Figure 5.3.F - MongoDB Replica Sets

Rather than spend a lot of time dealing with object representation in our application by creating object model classes manually, we will be using Mongoose JS, which is a MongoDB Object Document Model (ODM) module for JavaScript. Mongoose will allow us to define schemas for our data objects, which is not something that is built into MongoDB.

In addition to schemas, Mongoose will allow our application to resolve references between collections (the equivalent of a SQL JOIN call, but implemented in the application level instead of the database level) through “population.” This means that if “populate” is called on a field `user` storing a user’s ID, then Mongoose would grab that user from the database and fill in its user key with the user’s object. This is extremely useful since MongoDB doesn’t have the ability to perform collection JOINs (since with MongoDB the less relational your data, the better).

Mongoose will also play a large role in validation of user input from the frontend applications. Mongoose has a convenient asynchronous validation callback API, which allows you to ensure that data being stored in the database is of both the correct type of data as well as an appropriate value. With both standard SQL and MongoDB databases, data validation and data typing is the responsibility of the application. Mongoose makes it much easier by allowing us to define validation requirements on a schema level (for example, a data value in the schema must be a number less than 12, not a string).

With the features of Mongoose defined, the object schema models being planned for use can now be explained.

Database Schema Models:

1. User Model

   The user model will contain user information and global settings pertaining to the user. More specifically, it will primarily contain a unique id that can be used for referencing the user object in other models, the user's email address, and the user's password.

   Within the database, we will be indexing the user model by id as well as email address. This will allow O(log n) lookups (since indexes are a binary tree in MongoDB) on email addresses and ids. User ids will be resolved within references of other models and email addresses will be used for logging in.

   In addition, because we will be supporting sub-users (that is users created under the main user for use by family members to prevent account login information sharing), there will be an array of sub-users in this user data model. The array will contain user ids of sub-users, and each added sub-user will have its own user model loaded via Mongoose's population API. Each sub-user account contains a flag to lock down these nested accounts to only access their parent user's account.

2. Device Model

   The device model contains a device's name, unique id, and any setting toggles that the device may have. Since the home automation system will support multiple types of devices, there will have to be a device type field in our database model. This field value will be one of a list of enumerated values from a configured validation array. The enumerated values will correspond to the multiple different devices we offer (wall outlet, wall switch, power lock, hvac).

Within the database, we will be indexing the device model by id as well as user id. The user id field will be used to compile a list of devices per user, and the device id will be used for updating device information as well as via reference within other models (like the task, schedule, and goal models).

The backend API will be responsible for determining what values are available for configuration on a per-device-type basis. This will be made possible by defining device type on each per-field configuration within the model schema. The API can use this information by looking at the schema's definition for the value being set and comparing device type. For example, the API should distinguish that the temperature cannot be set on a wall outlet and return an error response in the event someone tries to set that value. The API should also hide all schema values that are not applicable to the device type selected. This will allow the frontend to render forms to change only applicable values.

3. Task Model

   The task model contains an array of device actions that are grouped into a user's task. The task will also contain a name field (used as a personal identifier of the task's purpose by the user), a task id (used for executing and updating tasks), and user id (used to compile a list of tasks per user). The tasks model will be indexed by task id and user id.

   The array of device actions is a list of all devices and the setting changes being performed on each device. For example, if a task has a device set to turn a light on, the device action would keep an object of that setting's new value as well as a reference id to the device the action is being performed on (the light device). For the array of device actions, Mongoose population will be used to replace the reference ids with the device model. Since the device id references are resolved in a O(log n) operation, it's a relatively efficient query in MongoDB despite the amount of excess queries generated by the population API.

4. Schedule Model

   The fields for the model consist of the schedule id (to perform updates on this resource), user id (the user who owns this scheduled task), task id (that's being run) nextRunAt, and the interval.

   The schedule model allows users to set an interval by which tasks automatically run. In order to do this in the most efficient way possible, the schema will be designed to allow the application to stateless-ly execute timers by pulling only tasks from the database which are ready to be executed. To achieve this there will be a nextRunAt field in the model which will store the

next time the schedule is to be ran. The application will then query the database every minute for scheduled tasks where nextRunAt is less than the current date and time. Upon execution of a scheduled task, the nextRunAt is computed based on the interval configured.

The interval will either be an exact date and time (for a one-time scheduled task) or a schedule based on Unix cron syntax. Cron syntax will only be used internally, and a UI will be created in the frontend for ease of use for users (who most likely do not know cron syntax).

The task id will be populated with the task model (and sub-populated with the devices within the task model, as described under the task model description). This will allow the user to preview the schedule actions being performed when adding and editing scheduled tasks.

5. Energy Management Goal Model

   The fields for the goal model consist of the goal id (to perform updates on the goal as well as retrieve goal graphs), user id (the user who owns the goal), device id (the device whose data is being calculated), and the goal information such as a goal name and threshold to beat (like kWh).

   The device id and device stats are both populated on retrieval by Mongoose. The device stats will be an aggregate population, which is not supported by Mongoose at this time. A custom function to aggregate stats will be created for this model.

6. Device Stats Model

   The fields for the device stats model consist of the user id (the user who the stats belong to), device id (the device the stats belong to), and the date the stats correspond to. The model also includes a day total aggregation stat as well as an array of smaller time increment stats for the device.

   In order to easily and efficiently aggregate data into graphs from devices, there will be a new stats document per device per day. Each stats model will have a stats array. Each index of this array represents an hour of the day (0-23) and each hour index will store another array representing each minute of the hour (0-59).

   The stat being recorded (like power usage) will update the device stat document corresponding to the current day. It will increment the stat according to the current hour and minute. It will also increment the day total aggregation stat.

The API will be responsible for aggregating all stats data from this collection within the date-range requested by the frontend into a format that the frontend can best process. For graphs ranging a timespan greater than a day, the daily total values within the stats model can be used to generate an API response. For a shorter timespan (less than a day), the API should return the arrays of the device's usage during the entire day.

When data is pulled from the database for graphic purposes, the frontend will then be responsible for transferring this data into a graph-able data array once retrieving it from via the API.

As is evident from the explained database models, there is quite a bit of relation between some of these models. Particularly users and devices are heavily depended upon. A full relation graph between models is provided in figure 5.3.G.

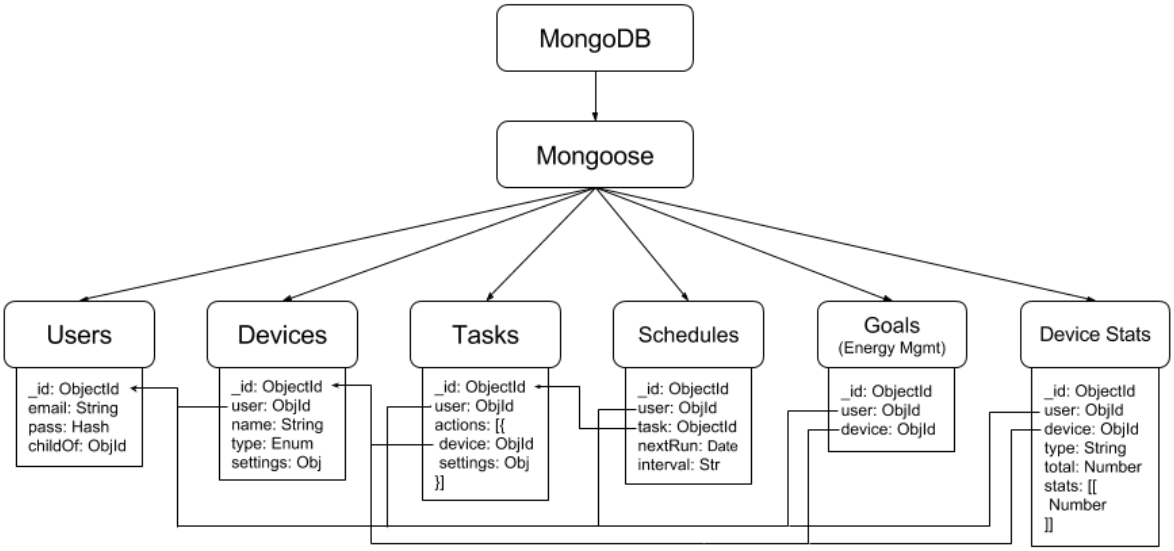


Figure 5.3.G - Database relation graph

5.3.3. Frontend

The frontend will be comprised of the AngularJS Model View Controller (MVC) framework coupled with the Bootstrap theming framework.

In order to serve a variety of devices, the frontend will be served as a website as well as a mobile application for iOS and Android devices. The frontend for both the website and mobile applications will rely on the same HTML/CSS/JS application being developed, but slight differences will occur based on what device a user is using (mobile device or. website). PhoneGap will be used to package the application into a native mobile app, and its APIs will be used in order to extend the functionality of the mobile apps.

5.3.3.1. AngularJS Setup

AngularJS will be used to render dynamic views for the website and mobile applications. In AngularJS, the template is composed of views controlled by controllers and directives in JavaScript. These templates will be split into smaller sub-views using Angular UI-Router, an open source module for AngularJS that allows you to dynamically load view templates and scope controllers to them by routing. This allows AngularJS to be a single-page application that loads new pages with JavaScript. This type of client rendering for new pages is much more efficient than server-side rendering of pages, saving on bandwidth and more importantly: page rendering times (giving users a quick, responsive app experience). An overview of the function of our AngularJS application with UI-Router is given in figure 5.3.H.

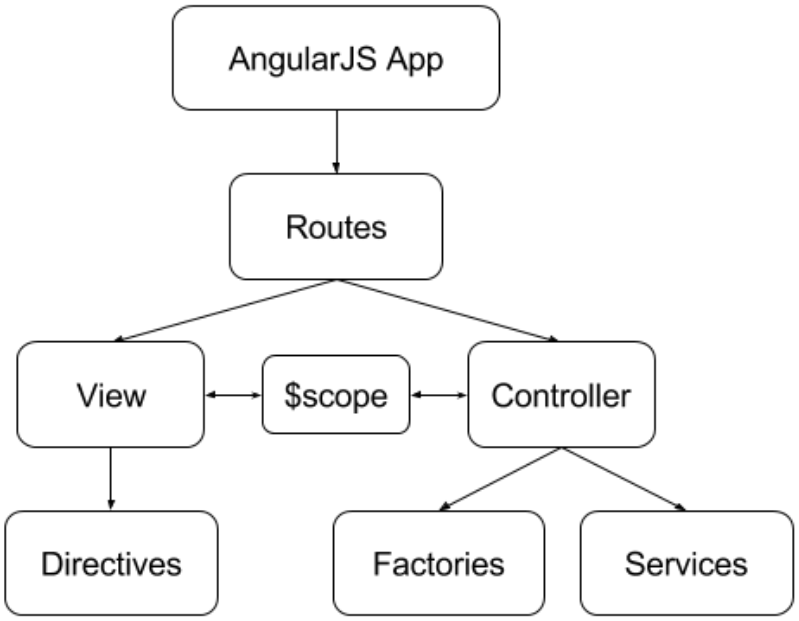


Figure 5.3.H - AngularJS Overview

The Angular UI-Router will be configured to have a specific path dedicated to each page being accessed. For each of the pages, a different controller will be used to bind data pulled from the API to the views (like populating a table of devices, buttons to edit specific devices, and populating forms with pre-filled information). Since data is two-way data bound in AngularJS (through the $scope), submitting forms is also handled within route controllers. This allows users to correct form errors without leaving the page or having to press the back button on their browsers.

Directives are modularized functions for manipulating the view's HTML with AngularJS. It allows you to create wrappers for modules not supported in

AngularJS, like graphing libraries. For our project, we'll be using Chart.js, an open source graphing library, in order to produce graphs for the energy usage stats aggregated for each of the devices being tracked.

AngularJS also has services and factories, which are able to be included as a dependency to all controllers. For this application, we will have one of each: a user service and an API factory.

The user service will hold the state of the user throughout their usage of the control panel. It is responsible for ensuring the user is logged in, has access to the page being requested, and holds all the user's stateful information (like email, name, etc.).

The API factory is a bit more complicated. AngularJS has helper methods for grabbing data from JSON APIs. However, these helper methods are barebones and do not support authentication over APIs without appending each request with login credentials. For better code reusability, the API factory will essentially be an API caller class structure. It will define methods get, post, delete, and put. These methods will get the user's API credentials from the user service and automatically fill in the credentials with all API requests. Since AngularJS uses an ES6 promise-like API for its API methods, the outgoing API requests are simply wrapped and returned (allowing our API factory to act as a negotiator rather than a middle-man for API requests).

5.3.3.2. Page Design

Since we will be using Bootstrap as a theme base, much of the CSS styling is already finished. This leaves the templates of pages to be designed. Apart from the login and registration page, the majority of the control panel will be part of a wrapper view with a sidebar (that has a purpose of providing navigation) as well as a header (that has a purpose of displaying the product logo, any alerts the user has received, and login/logout). The figure in 5.3.I is a basic mockup for the website wrapper design.

The UI View is the remaining portion of the wrapper to be explained. The UI View is controlled by Angular UI Router, and is populated based on the current route that is loaded. Angular UI Router loads the view HTML and the view controller and then runs them. Each of these views correspond to individual page views.
The container will also scale appropriately for mobile devices (since the same HTML/CSS/JS will be used in the mobile apps as well). Because space is very limited for mobile devices, the sidebar will hide and become a toggle when the screen region is small. The toggle functionality is depicted in figure 5.3.J.

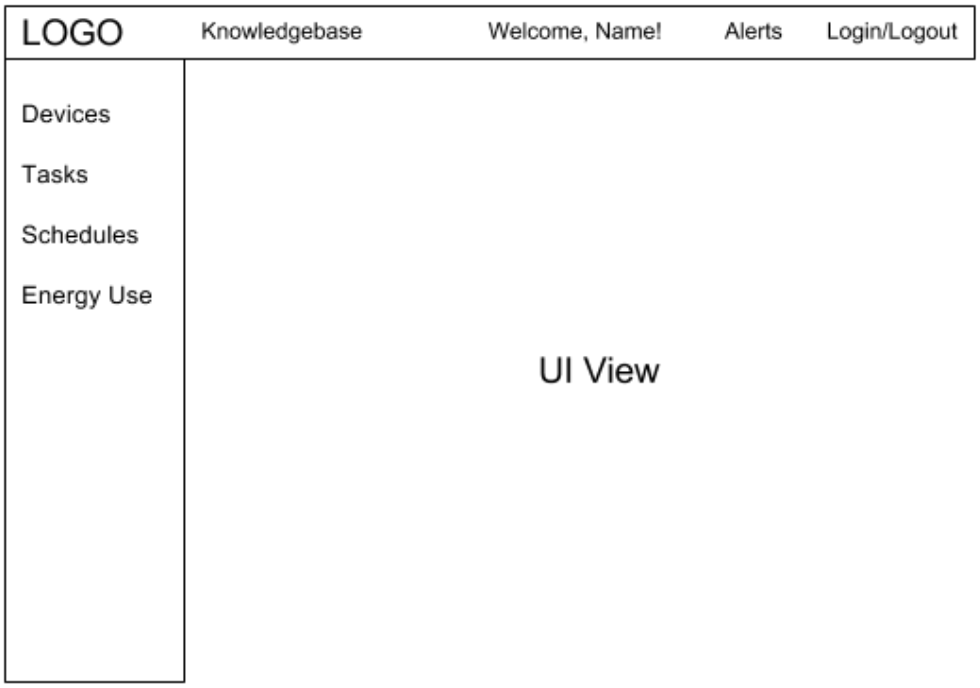


Figure 5.3.I - Page Design Wrapper

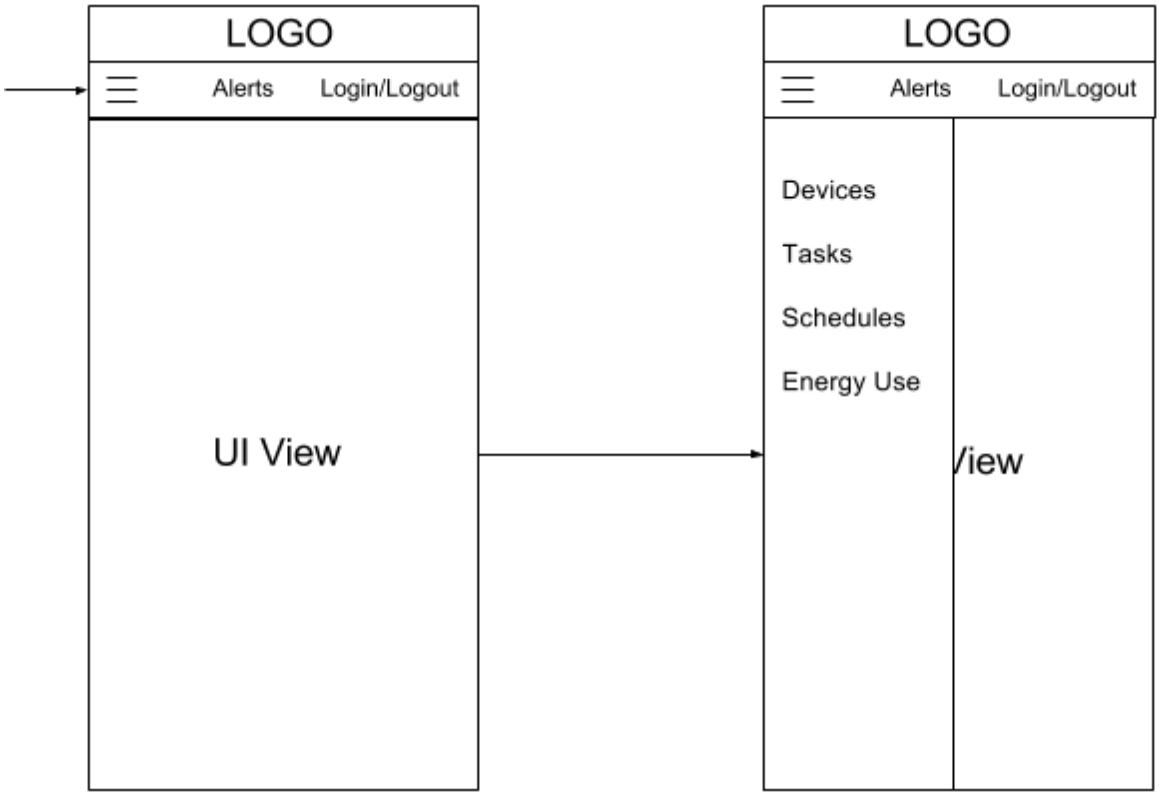


Figure 5.3.J - Scaling the Navigation Menu for Mobile Devices

1. Account Settings

   The account settings page will feature the ability to update name, email, and password. It will also allow users to delete their account and add or remove child users to and from their account.

   The design of this page will be fairly straightforward. The page will be organized vertically such that there is a form panel for updating settings, a delete account panel, and a child users panel.

   The delete account panel will pop up a window requesting confirmation to delete the user's account. This will issue a DELETE API request to the API, which will then set the account to be "soft deleted" (that is to say it can be recovered manually should this have been a mistake).

   The child users panel will feature a simple table populated by the child users array served from the users API endpoint. There will be line-by-line delete icons as well as a plus icon atop the table to add new child users. The delete icon will prompt for confirmation before allowing child users to be deleted.

2. Devices

   The devices page will list all of the devices retrieved from the devices API endpoint within a table. Each device in the table will have an edit and delete button listed line-by-line. At the top of the table will be a button to add a new device.

   The edit icon will popup a configuration window within the page (called a modal). The modal will be a form allowing users to edit the device information as well as configure all of the device's settings.

   The delete icon will popup a confirmation window before deleting the device.

   In addition to edit and delete icons for each device, the device's primary setting will be configurable within the list too. For example, On/Off for a light or setting temperature for the HVAC. This will give users an at-a-glance ability to change their devices.

   The ability to add a new device is more complicated. Devices will only be able to be setup from the mobile application. This is a limitation due to needing Bluetooth to configure the WiFi settings on the device as well as negotiate API configuration to the device. On non-mobile apps, an error message will be returned letting users know that they must use the mobile app. When on a mobile app, there will be a setup guide to help the users link the device via

Bluetooth to their phone and subsequently configure WiFi and finish setting up the device. This setup is detailed in §5.3.3.3.

3. Tasks

   Tasks are a preconfigured setup for a grouped-together devices. They can be used to turn everything off when you leave home, or turn off lights and turn down the air conditioning when you go to bed. There's virtually endless possibilities for tasks.

   The tasks page will list all of the tasks retrieved from the tasks API endpoint within a table. Each task entry will have line-by-line run, edit, and delete buttons, and at the top of the table will be a button to add a new task.

   The run button will run the saved task.

   The delete icon will popup a confirmation window before deleting the task.

   The add a new task button will popup a modal and ask for a name for the new task. After submission, the users' page will redirect to the edit page for that task.

   The edit icon will also direct people to an edit page for the task. Instead of using modals for editing tasks, there is too much configuration needed so a modal would not be a large enough space to work with.

   Because of the amount of configuration required for tasks, there will be a devices list on the editing page. Each task contains an array of device actions. These actions are pre filled into the edit page by the tasks API endpoint. The devices list will allow users to add, edit, and remove devices from the task.

   The add button will prompt the user to select a device from a list of unused devices for that task. AngularJS will load the user's' available devices from the devices API endpoint and filter the returned list to exclude those already assigned to the task. After adding a device, the user will see the device appear in the list and they will be able to configure the device's settings to be set when the task is run.

   The devices in the list can be edited or removed. When removing a device, the delete icon will popup a confirmation window before deleting the device. The edit icon will popup a similar modal window used for editing device settings from the devices page, with the exception of the ability to edit things like the device name. Device settings changed here will be applied to the

device action's settings object, not the device's settings object, so changes are unique only to the task being performed.

4. Schedules

Schedules allow users to automate the running of tasks. This could mean on an interval (like Monday through Friday at 6pm) or a specific, one-time date and time like (Dec. 15th, 2015 at 6pm).

A list of schedules will be imported from the schedules API endpoint into a table. Users will be able to add, edit, and remove scheduled tasks. Tasks will also have the ability to be toggled on or off (in the case where you don't want to remove a schedule and only need it to be temporarily disabled).

When removing scheduled tasks, the user will be prompted to confirm the action to ensure they don't accidentally click the button.

Adding and editing schedules tasks will rely on the same HTML to create a modal. The modal will have a spot to name the schedule, select the task to run for the schedule, and then allows the user to select either a static date or an interval.

To select a static date, the open source JavaScript library DateTimePicker will be used within an AngularJS directive to allow users an easy graphical interface to select a static date.

If the user opts to select an interval, a custom interval selection directive will be used since the API endpoint requires cron syntax for the interval. The cron syntax is explained in figure 5.3.K.

The syntax also supports dashes (-) for ranges, commas (,) for selecting multiple, and asterisks (*) for all. For the implementation being designed, we will only be selecting with commas or asterisks or neither.

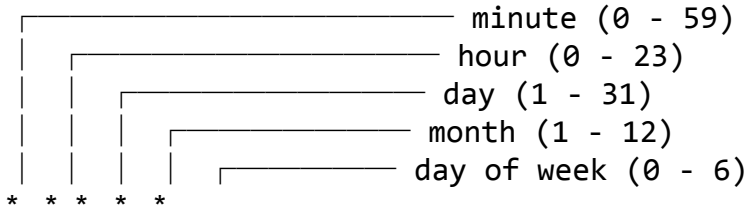


Figure 5.3.K - The cron syntax format

There will be 5 input multiple select form elements, each corresponding to the minute, hour, day, month, and day of week. For each input, the ranges specified in the syntax from figure 5.3.K will be loaded for selection (but

instead of using numbers for month and day of week the numbers will be translated into their verbose terms (month names and day names). The design of the cron interval selection form is provided in figure 5.3.L.

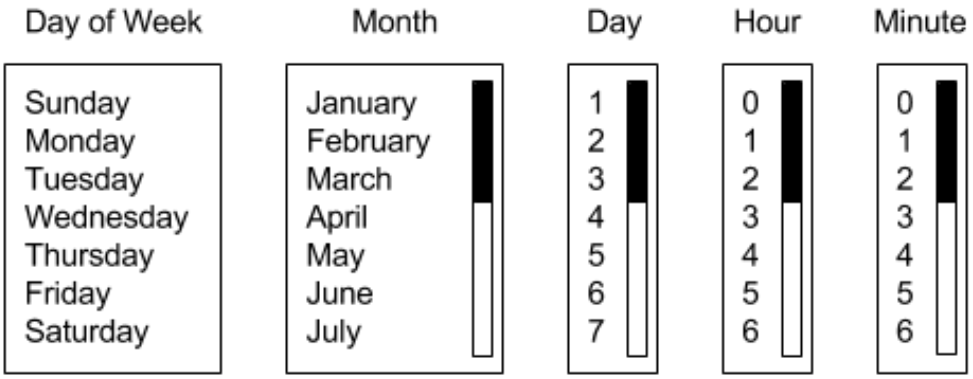


Figure 5.3.L - Cron Interval Selector

The AngularJS directive being created will watch each form input for a change, and then compile the list of selected elements into the cron syntax. If all of the values in the input are selected, the directive will set that input's value in the cron syntax to an asterisk. Otherwise, the directive will make a comma-separated list of the selected values in the input and place it in the cron syntax output.

5. Energy Management

The energy management page will allow users to monitor their energy usage and make goals to keep them on track to reduce their energy footprint.

The layout of the energy management page will be just a bunch of graph panels. The default loaded graphs will include total energy consumption within varying time periods (day, month, year) and will show the goals (with their graphs) of any configured goals. You will be able to select individual devices to get stats for single devices rather than grouped totals.

We will be utilizing the open source graphing library Chart.js to create line graphs for energy consumption. There is an open source Chart.js AngularJS directive called Angular Chart which we will be including to speed up development time.

The graphs will be loaded with data from the energy API endpoint. Since Chart.js requires each graph to have data formatted into arrays of purely integers, the data will be processed client side into a label and data value format where label is the X axis value and data value is the Y axis value.

```
{
  labels: [“January”, “February”],
  datasets: [{
    data: [100, 105]
  }]
}
```

The goals will be able to be managed right from the energy management page. For existing goals, there will be a delete icon in the upper right of the graph panel for the goal. There will be a confirmation prompt if a user attempts to remove a goal, acting as a safeguard to accidental deletion. To add a new goal, there will be a button in the upper right of the page.

When adding new goals, a modal will pop up prompting for information like a goal name, the device/task/total being tracked, as well as the goal limit (the value that you are attempting to achieve). After adding a goal, the graph panel for the goal will appear on the page.

If a goal gets broken or the system detects you will not be able to meet a goal, it generates an alert. Alerts will show at the top of the control panel as entries for the dropdown alert icon. When there is an unread alert, there will be an unread message count next to the alert icon. An email will also be dispatched to the user to notify them they will not be able to meet their goal.

5.3.3.3. Microcontroller Setup

The microcontroller setup will only occur when setting up new devices. If a device needs to be reconfigured, it will need to be removed and then re-added. Additionally, setup can only be performed on mobile devices due to the requirement of Bluetooth being needed to program the device for use.

For mobile devices, we will be using PhoneGap to merge our HTML/CSS/JS AngularJS application into a native mobile app for Android and iOS. PhoneGap has an official plugin for Bluetooth, that, once included in the configuration, injects a JavaScript Bluetooth API into the scope of the AngularJS application. This allows the AngularJS application full control over the Bluetooth module within the mobile device being used to configure the microcontroller.

The setup process begins by pressing a button on the automation device to pair the Bluetooth module on the automation device with the Bluetooth on the mobile device. Once the button is pressed, the mobile device can find and connect to the automation device.

Once connected, the application will negotiate with the device to ready it for programming the WiFi module. The user will then type in their WiFi connection

settings that the automated device needs to access the Internet. The automated device will receive these connection settings and then test the connection to ensure proper connectivity.

Once the automated device confirms that the cloud service is reachable over the Internet, it will exchange its device ID and device type back with the server and obtain an API token for the API server. The API token will allow to to establish a TLS keep-alive socket with the API server to both transmit usage data to the API to be used for graphing purposes as well as receive commands from the API (like turning on and off the light).

After the device completes setup, it disconnects from Bluetooth and the mobile device displays a message of success with configuring the networking. The mobile app will then force the user to test out the newly added device to ensure that it is functioning normally.

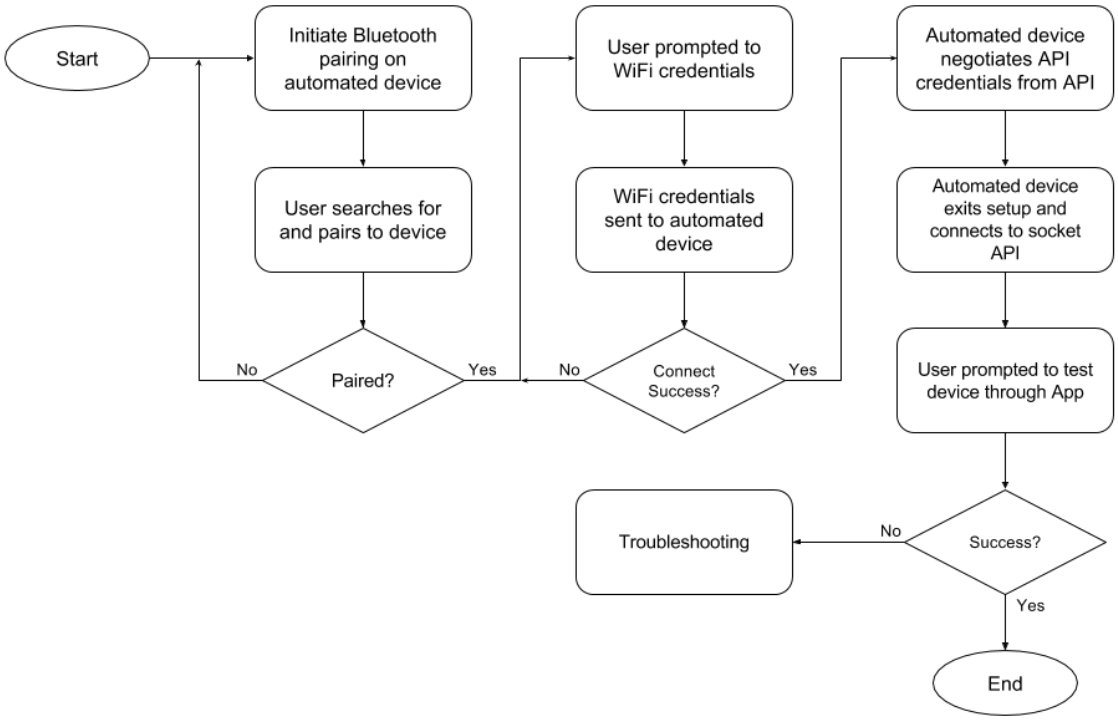


Figure 5.3.M - Microcontroller Setup Flow

The mobile app will ask the user to set the temperature for HVAC, unlock the door for a powerlock, and toggle on and off a switch or plug, depending on which device that is being configured.

After the application sends a test action to the device, the mobile application will prompt the user to ask if the device performed the test as expected. If the user agrees that the test functioned properly, the mobile application exits device

microcontroller setup and heads back to the devices page. If the user denies that the test functioned properly, the mobile application tries to run through troubleshooting steps to resolve the issue with the user.

The entire setup process is depicted by a flowchart with figure 5.3.M, which will give a more visual understanding of the setup process needed to configure a microcontroller.

6. Testing

The following section covers all of the testing necessary to ensure every component of the home automation system was functional and performed as expected. There are two sections of testing, one for hardware and one for software. The following sections will cover the plan for testing, environments in which testing was done, and the results of testing, including data collected and observations made during testing. All testing was done under supervision for safety purposes.

6.1. Hardware Testing

In order to test the hardware components, each individual module needs to be tested first before assembling everything together. Each module needs to perform according to the datasheet as well as according to the project specifications and requirements. Because this is only hardware testing, interfacing with software was not necessary and thus the final code for the components are not needed. Testing Plan will discuss the testing for the individual components of the PCB. Testing Environment will explain the environment in which the components were testing as well as the ideal conditions that are needed for the home automation system to work in complete functionality.

6.1.1. Testing Plan

For the electrical components (WiFi Bluetooth module, Power Sensor IC, Relay), because of the nature of these components, it is not possible to individually test each component, as they can not operate without acting on another module. For example, the power sensor IC can not transmit a signal to nothing; the power sensor needs to be connected to a WiFi/Bluetooth module to transmit the signal to the central and also needs to be connected to a microcontroller in order to know when to turn on and off. However, a microcontroller is easy to test because all MCU's can act on their own and can take in a code input and output a signal. Testing the MCU requires a quality assurance check of the timers and registers for basic function of the MCU. For example, a code for basic input of a keyboard input to the MCU was used to make sure the code was properly input into the

MCU and a proper output from the MCU was asserted. Afterwards, a simple routine code was created to test the interrupt service routines of the MCU. Results of the test can be seen in Figure 6.1.A.

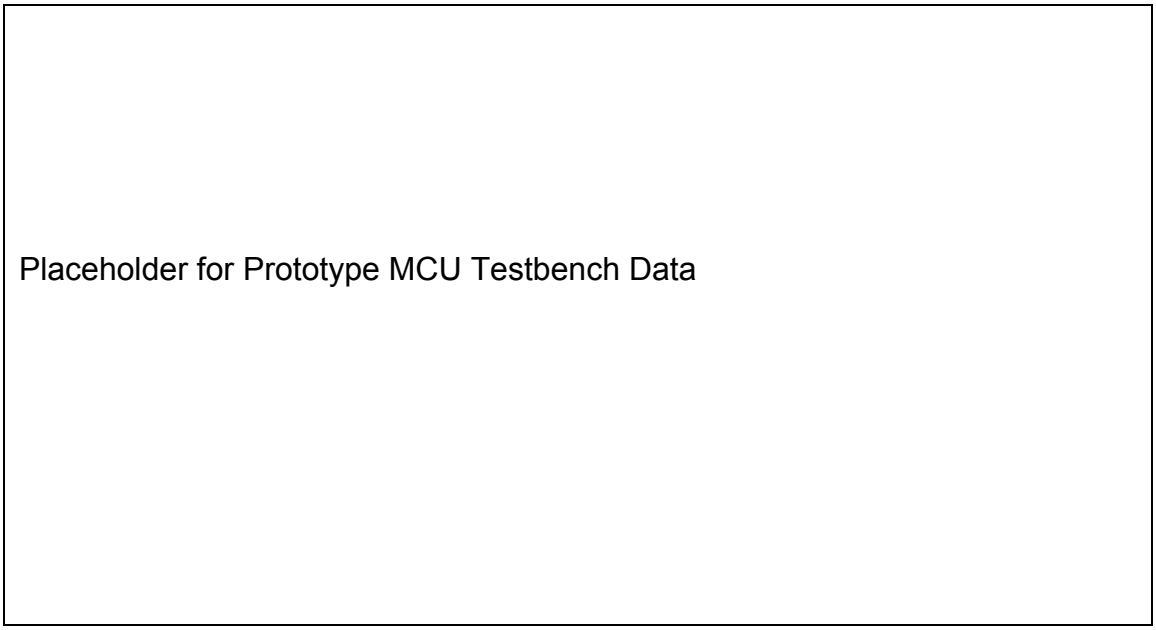


Figure 6.1.A - MCU Testbench Data

When all preliminary tests were passed and completed, the other electrical components needed to be tested, one at a time. For the WiFi/Bluetooth module, a simple connection to the MCU and testing via smartphone is needed to assure the module works. This is done by connecting the WiFi/Bluetooth module to the MCU. The MCU contains a code that detects the wireless module and has a connection to the WiFi access point that is the central hub. A smartphone can also connect to this central hub. This central hub access point can receive a signal from the smartphone and transmit the signal to the WiFi/Bluetooth module, which then goes to the MCU for further instruction. In order to make sure this routine works, a computer, connected to the access point, asserted an input to acknowledge the MCU to the access point. The central hubs receives this input and forwards the signal to the WiFi/Bluetooth module, which is sent to the MCU. If everything works as planned, the MCU should output a signal via sending a confirmation message back to the computer via the reverse process. Because this is simply testing for the MCU and WiFi/Bluetooth module and both of these modules are being used for every component of the home automation system, the above procedure only need to be done once. The same results will produce for the same MCU inside other components, such as the wall outlet or powerlock. Figure 6.1.B shows data for these tests.

The above process can be repeated for the power sensor IC. While the MCU is operating, the power sensor is constantly receiving the power output and collecting the data over several intervals. This data is periodically sent to the

central and forwarded to the main website/smartphone app. For testing purposes, simply testing the power sensor IC only requires the sensor to display the current power onto the smartphone. If the correct power is shown, the power sensor IC is functional. The relay sensor can be easily tested by connecting the MCU to the relay. Inside the MCU, a sample code to operate the relay is used and an asserted output will show if the relay is functional.

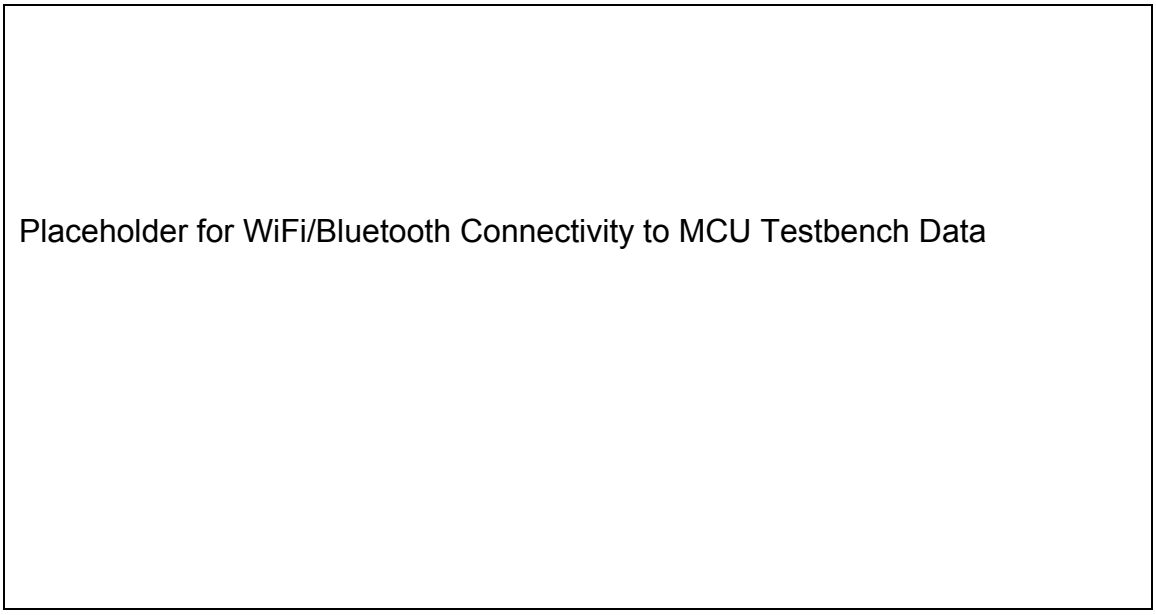


Figure 6.1.B - MCU WiFi/Bluetooth Connectivity Testbench Data

After all of the individual electrical modules are tested, it is time to put everything together and test if the overall setup functions properly. This is explained further under the next sections, where each component of the home automation system is implemented and tested.

6.1.2. Testing Environment

The home automation system needs to be able to operate under everyday conditions inside a typical room. A normal bedroom was used to test the final product of the home automation system. Because this particular bedroom is located in Florida, the constant change in weather needs to be considered. The components need to be able to operate under temperatures between 60℉ to 100℉. The casing that covers the PCB needs to be able to withstand hot temperatures over a long period of time without any hindrance to the performance.

Testing the individual components as described in the testing plan requires a lab that contains a breadboard for basic wiring and a multimeter for testing voltages across certain pins. There also needs to be computers that have the necessary software to create and run code for the MCU. These softwares include Code

Composer Studio and any C programming IDLE. The Senior Design laboratory and TI Innovation Lab were used for these purposes.

6.1.3. Wall Outlet Testing

One of the most important components of the home automation to test would be the wall outlet. Since a lot of the other components rely on the wall outlet, making sure the wall outlet functions as expected is the primary goal. This is why the wall outlet will be the first to be tested.

6.1.3.1. Prototype Construction

A very basic prototype of the wall outlet was created in the TI Innovation Lab using MCU's and relay sensors that are not part of the final product. This is due to availability of the components at the time and a need to explore a more variety of components. For this demonstration a TI Launchpad MCU was used, connected to a WL18xxMOD 8 Single-Band Combo Module and a simple relay was connected to the MCU. For testing purposes the power sensor was not needed and will be excluded from this section for now. Figure 6.1.C shows a functional block diagram of the prototype wall outlet. Figure 6.1.D and Figure 6.1.C shows a functional block diagram and picture of the prototype wall outlet used for testing purposes, respectively.

6.1.3.2. Electrical Input/Output

After building the prototype an electrical test was used to test if this wall outlet was able to accept an input of 120V AC into the plug, convert this AC voltage into DC, and output the correct voltage by measuring the output using a multimeter. If done correctly the MCU will be turned on without any complications or issues. Figure 6.1.E shows a table of outputs measured using the multimeter provided in the TI Innovation Lab. The outside temperature is also recorded for datasheet purposes, although temperature did not have any affect on these tests.

6.1.3.3. Wireless Communication

When the MCU passed the above test, a communications test is needed to make sure the MCU is correctly wired to the WiFi/Bluetooth module and that a signal can be transmitted and received to this wireless module. For testing purposes, the wall outlet prototype will be connected to the computer and a test for a good, established connection will be utilized. Figure 6.1.F shows the setup used to test this functionality. Following an input from the user on the computer, the wall outlet should respond by successfully activating the MCU and the relay turning on a light bulb. Multiple runs of this test will be run, making sure the user input is able to turn on and off the light bulb plugged into the wall outlet. Figure 6.1.F shows the runs completed as well as a success/fail for each run.

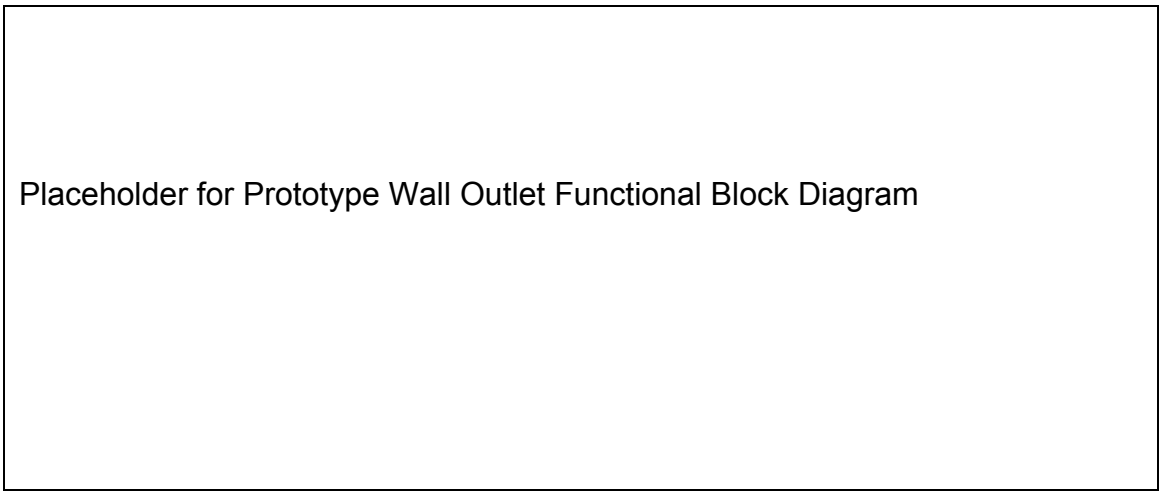


Figure 6.1.C - Wall Outlet Functional Block Diagram

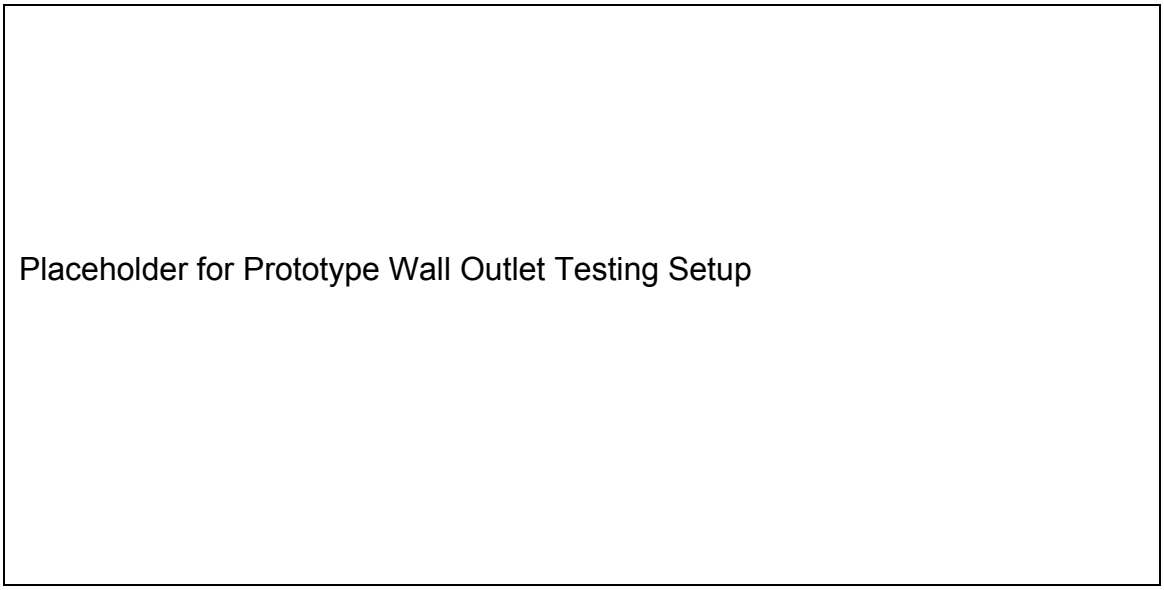


Figure 6.1.D - Wall Outlet Testing Setup

Test Number	Outside Temperature (in F)	Output Voltage (in V)
1		
2		
3		
4		
5		

Figure 6.1.E - Wall Outlet Temperature and Voltage Table

Test Number	Light Bulb Status (ON/OFF)	Expected Light Bulb Status
1		
2		
3		
4		
5		

Figure 6.1.F - Wall Outlet Wireless Communications Test for Proper Execution

6.1.4. Wall Switch Testing

Because the wall outlet is wirelessly connected to the wall switch using WiFi/Bluetooth, the tests for the wall switch will be very similar to those of the wall outlet. Figure 6.1.G and Figure 6.1.H shows a functional block diagram of the prototype wall switch as well as a picture of the basic setup used for testing the wall switch, respectively. Assuming the wall outlet works as intended, testing the wall switch simply requires the button to be pressed/flipped. This should transmit to the wall outlet and activate the MCU and relay to either turn on or off the light bulb. Again, a success/fail run was implemented for the wall switch. Results can be seen on Figure 6.1.I

Placeholder for Prototype Wall Switch Functional Block Diagram

Figure 6.1.G - Wall Switch Prototype Functional Block Diagram

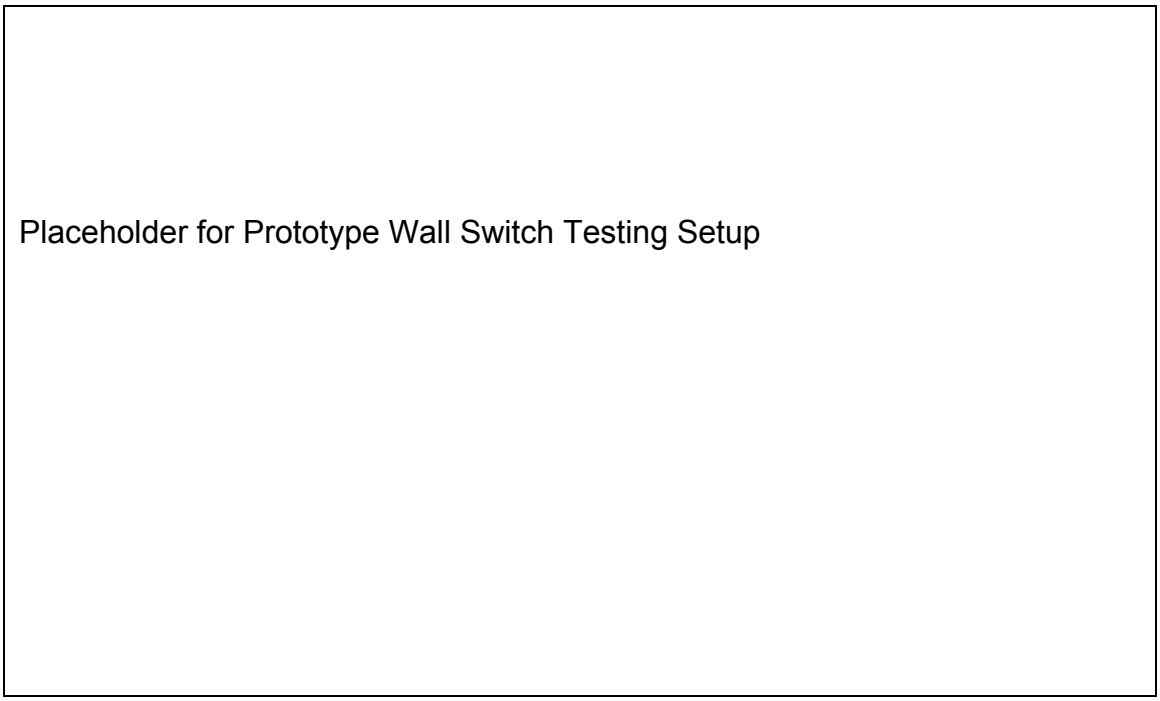


Figure 6.1.H - Wall Switch Prototype Testing Setup

Test Number	Light Bulb Status (ON/OFF)	Expected Light Bulb Status
1		
2		
3		
4		
5		

Figure 6.1.I - Wall Switch Table for Button Testing

6.1.5. Powerlock Testing

Testing the powerlock is different from the other components in the sense that the powerlock utilizes a mechanical part, which is the electric strike needed to open and close the powerlock. Because of this, there will be a delay between user input and execution of instruction. This is covered further under the "Wireless Communication" section in "Powerlock Testing". Figures 6.1.J and 6.1.K show the functional block diagram of the prototype powerlock and the testing setup, respectively.

Placeholder for Prototype Powerlock Functional Block Diagram

Figure 6.1.J - Powerlock Prototype Functional Block Diagram

Placeholder for Prototype Powerlock Testing Setup

Figure 6.1.K - Powerlock Prototype Testing Setup

6.1.5.1. Primary Testing

Testing the prototype powerlock requires ensuring that the powerlock is able to accept a user input from the computer to the powerlock as a basis. This will be done with the same test as used for the wall outlet. The MCU will be directly connected to the computer via micro-USB cable. A user keystroke should send a message to the powerlock's MCU. This allows the MCU to activate the powerlock, either opening or closing depending on its current state. This test is simply to see if the MCU was properly connected to the powerlock. Figure 6.1.L shows the results of this test.

Test Number	Powerlock Status (Open/Close)	Intended Status
1		
2		
3		
4		
5		

Figure 6.1.L - Powerlock Testing for Initial Setup

6.1.5.2. Wireless Communication

The next step is to have the MCU interact with the WiFi/Bluetooth module. Because the power lock mechanism is directly connected to the MCU, there is no need for a relay. Another factor for the wireless communication aspect of testing is the time that it takes between user input and execution. The desired time for execution should be a very small and insignificant. For example, a time of at least 1 second would be considered non ideal and very slow for what is desired. A time between 0.25 seconds to 0.5 seconds means a very good connection and time, but not exactly ideal. Figure 6.1.M shows the results of this testing.

Test Number	Powerlock Status (Open/Close)	Time to Complete (in sec)
1		
2		
3		
4		
5		

Figure 6.1.M - Powerlock Wireless Communication Test For Response Time

6.1.6. HVAC Controller Testing

Perhaps one of the most challenging components of the home automation system to test will be the HVAC controller. This is due to the component being built from scratch and being compatible with the proprietary system of the home

automation system. There are three aspects that will be tested in the HVAC controller include prototype, user input, and wireless communication testing. Figures 6.1.N and 6.1.O show the functional block diagram of the prototype HVAC controller and the testing setup, respectively. Because the prototype does not contain any inputs other than power and physical input, the received wireless signal of the final result HVAC controller is not shown. However, the final design functional block diagram of the HVAC controller will have these inputs.

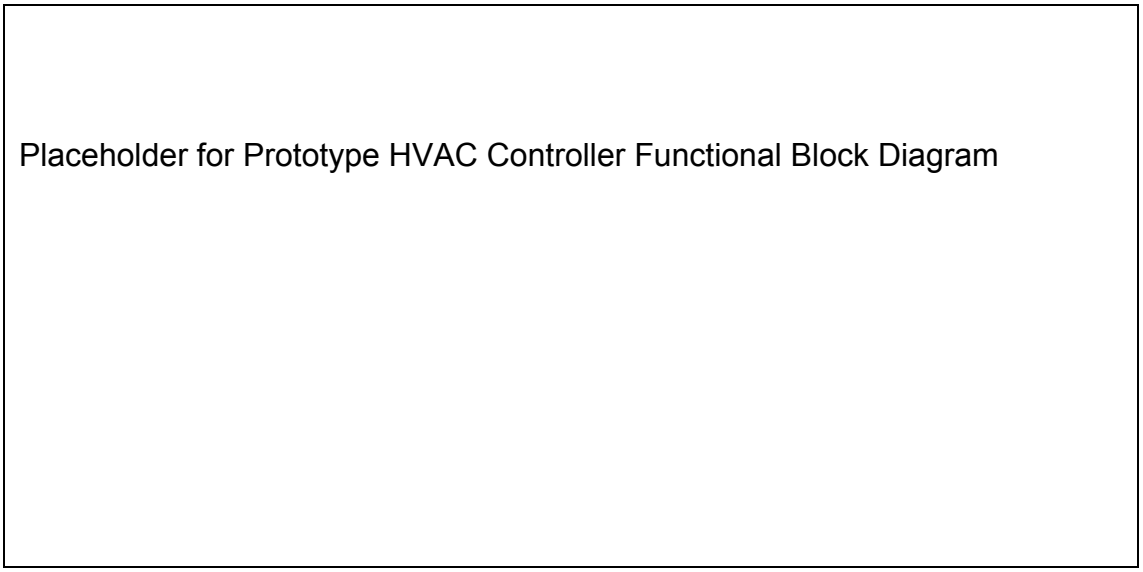


Figure 6.1.N - HVAC Controller Prototype Functional Block Diagram

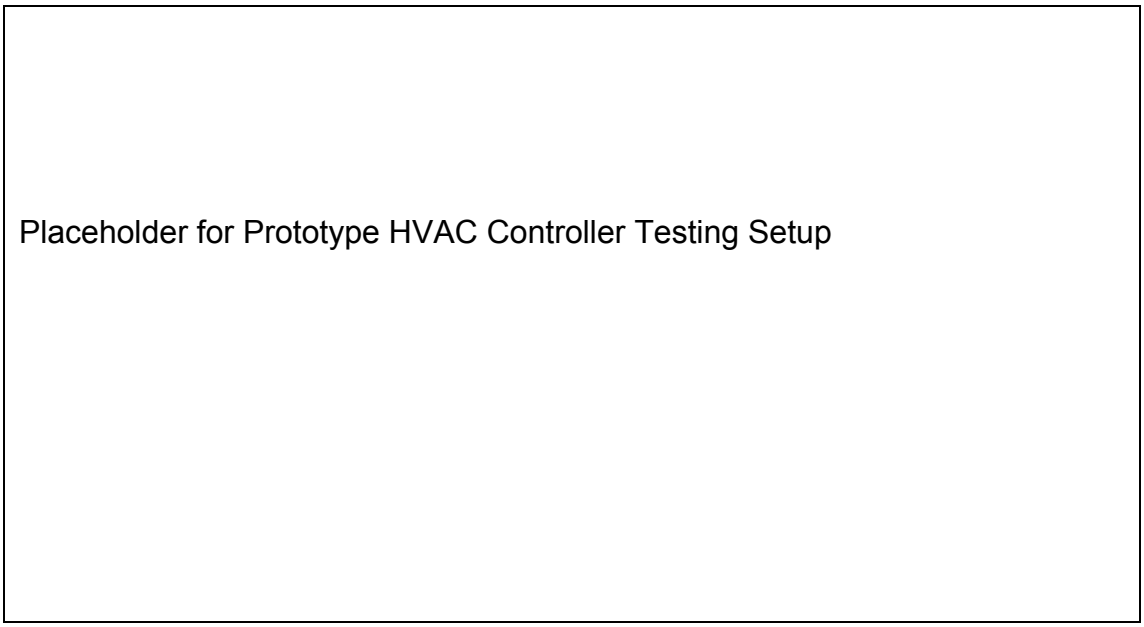


Figure 6.1.O - HVAC Controller Prototype Testing Setup

6.1.6.1. Prototype Testing

This section covers the basic testing of the prototype HVAC controller. Because this component is still in its early stages of development, there are no casing or special LCD screens to account for. These tests will only test if the completed prototype is able to lower, raise, and set temperatures using the buttons on the device. Completing these tests would fulfill the duty of a basic HVAC controller. Specific functions unique to the home automation system will be covered under the “User Input Testing” section of “HVAC Controller Testing”.The prototype testing will only use the physical buttons on the HVAC controller itself. Testing for a user input via computer will covered under “User Input Testing”.

The tests will involve what the user pressed and the result of what happened. The goals of the tests are as follows:

- Up Arrow correctly increases temperature/ time of schedule
- Down Arrow correctly decreases temperature/ time of schedule
- Power Button turns on/turns off the device
- Center Button correctly confirms set temperature

The table will include the desired result after pressing the button as well as what actually occurred. For example, if the desired result was to increase the temperature from 44 degrees Fahrenheit to 45 degrees Fahrenheit and the observed result was a decrease in temperature, then there was an error in either the wiring of the buttons or the coding. Figures 6.1.P, 6.1.Q, 6.1.R, and 6.1.S shows the results of the testing.

Test for Up Button		
Test Number	Observed Result	Desired Result
1		
2		
3		
4		
5		

Figure 6.1.P - HVAC Controller Prototype Test for Up Button

Test for Down Button		
Test Number	Observed Result	Desired Result
1		
2		
3		
4		
5		

Figure 6.1.Q - HVAC Controller Prototype Test for Down Button

Test for Power Button		
Test Number	Observed Result	Desired Result
1		
2		
3		
4		
5		

Figure 6.1.R - HVAC Controller Prototype Test for Power Button

Test for Center Button		
Test Number	Observed Result	Desired Result
1		
2		
3		
4		
5		

Figure 6.1.S - HVAC Controller Prototype Test for Center Button

6.1.6.2. User Input Testing

The main aspect of the HVAC controller is to be able to wirelessly control the HVAC controller from a smartphone. However, before a WiFi/Bluetooth connection can be made, the HVAC controller needs to be able to accept input from another source. The inputs will be similar to the physical buttons on the device itself and a replica screen would be shown on either the computer or smartphone.

This section covers using a computer connected to the HVAC controller in order to attain an established connection and confirm that the MCU inside the HVAC controller can accept input from an outside source. When the user inputs from the computer a specific command, the command should reflect that of pressing the physical buttons on the controller itself. The test covers all of the buttons shown in the “Prototype Testing” section under “HVAC Controller Testing”. Figures 6.1.T, 6.1.U, 6.1.V, and 6.1.W shows the results of testing.

Test for Up Button		
Test Number	Observed Result	Desired Result
1		
2		
3		
4		
5		

Figure 6.1.T - HVAC Controller User Input Test for Up Button

Test for Down Button		
Test Number	Observed Result	Desired Result
1		
2		
3		
4		
5		

Figure 6.1.U - HVAC Controller User Input Test for Down Button

Test for Power Button		
Test Number	Observed Result	Desired Result
1		
2		
3		
4		
5		

Figure 6.1.V - HVAC Controller User Input Test for Power Button

Test for Center Button		
Test Number	Observed Result	Desired Result
1		
2		
3		
4		
5		

Figure 6.1.W - HVAC Controller User Input Test for Center Button

6.1.6.3. Wireless Communication

The final step of testing the HVAC controller is testing the WiFi/Bluetooth portion. As stated in the previous “Wireless Communication” sections, the time of execution is important in determining the efficiency and responsiveness of the WiFi/Bluetooth module inside the HVAC controller. However, it is more understandable to have a slightly longer response time than that of the powerlock. This is because of the HVAC controller processing the data received as well as being able to execute the task in such a short time. Because of this, acceptable response times would be around 0.5 seconds to 1 second. Tables 6.1.X and 6.Y shows the results of testing. A significant number of tests is needed to account for every possible option included in the smartphone app. Each action was recorded on top of response time and if the instruction was executed properly. Each response time also includes an expected response time and if the time recorded compared to expected time is close or vastly different from each other, as shown by a percent error.

Test Number	Command Executed	Did It Execute Properly?
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Figure 6.1.X - HVAC Controller Wireless Communication Test For Proper Execution Of Commands

Test Number	Response Time	Expected Time	% error
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Figure 6.1.Y - HVAC Controller Wireless Communication Test For Response Time

6.2. Software Testing

The testing of our software will be primarily focused on automated tests, and testing by hand will only be used to ensure the application frontend and hardware/software communication functions properly.

6.2.1. Automated Tests

Since the majority of the software components rely on the API, it is important that the API is functioning properly and without errors.

Automated testing is something relatively new with development workflows, and has come as a result of increasingly complex applications being built regularly and by large teams of developers. When there's multiple people working on a project at the same time, mistakes can happen and functionality can break.

In order to prevent group members from writing code that can break functionality that is already working, automated tests can be created to ensure proper support for all new code changes being added to the project's API software.

There are a many ways people choose to run tests, but mostly people either utilize a test server with populated data to test code changes against or test data is generated per-method (matching expected input for the method being tested) to ensure proper functionality.

Because of time constraints on this project, writing per-method tests can be expensive in terms of time it takes to write accurate and complete per-method tests. We will be going with the test server method instead, which should free up more time to work on the rest of the software development cycle.

We will be using a test server (running locally in a virtual machine) to test code changes on. It will be a server mimicking the configuration of the standard cloud server for our application, with the exception of it being configured to run in a single instance mode instead of distributed across multiple nodes (for example, standalone MongoDB instead of running it in a replica set). It will be preconfigured with a test user account in the database to allow automated tests to run on their own.

For the automated tests themselves, we will be utilizing Postman, a freeware tool by Postman Labs. It has a low-cost paid upgrade that allows you to create and run automated tests with their open source companion command line tool called Newman. Newman can be added to the package.json of the API software in order to easily run tests with the "npm test" command.

In Postman, collections of tests can be created. For the API, this will consist each of the major modules of the API: users, devices, tasks, schedules, and energy.

Each collection will contain multiple API calls to the test API server running locally. These API calls will try out every test case necessary to ensure that the resource being tested covers all potential problems that could happen, ranging from invalid information being returned to invalid input data being submitted into a form by the user.

User Test Cases

- User Registration – PUT /user
  - Valid/Invalid Email Address (ensure email address is only allowed chars)
  - Valid/Invalid Password (enforce minimum password length)
  - Valid/Invalid User Parent (for child users, optional)
    - User Parent must also be verified in the GET /user response after creating a child user account
  - Valid Success Response (returns API credentials for API authentication)
  - Valid Error Response (returns error response object)
- User Login – POST /user
  - Valid/Invalid Email and Password (checked in combination)
  - Valid Success Response (returns API credentials for API authentication)
  - Valid Error Response (returns error response object)
- User Info – GET /user
  - Invalid API credentials
  - Valid Success Response (returns user object model)
  - Valid Error Response (returns error response object)

Device Test Cases

- Add New Device – POST /devices
  - Invalid API credentials
  - Valid/Invalid Device ID (unique device identifier)
  - Valid/Invalid Device Type (enumerated value of supported device types)
  - Valid Success Response (returns API credentials for the device)
  - Valid Error Response (returns error response object)
- List Devices – GET /devices
  - Invalid API credentials
  - Valid Success Response (returns list of device object models)
  - Valid Error Response (returns error response object)
- Delete Device – DELETE /devices/:id
  - Invalid API credentials
  - Valid/Invalid Device ID
  - Valid Success Response (empty page, status code 204)
  - Valid Error Response (returns error response object)

- Modify Device – PUT /devices/:id
  - Invalid API credentials
  - Valid/Invalid Device ID
  - Valid/Invalid Setting Changes (like setting an integer to a string or modifying an inapplicable setting for the device type)
  - Valid Success Response (returns modified device object model)
  - Valid Error Response (returns error response object)

Task Test Cases

- Add New Task – POST /tasks
  - Invalid API credentials
  - Valid/Invalid Task Name
  - Valid Success Response (returns task object model)
  - Valid Error Response (returns error response object)
- List Tasks – GET /tasks
  - Invalid API credentials
  - Valid Success Response (returns list of task object models)
  - Valid Error Response (returns error response object)
- Delete Task – DELETE /tasks/:id
  - Invalid API credentials
  - Valid/Invalid Task ID
  - Valid Success Response (empty page, status code 204)
  - Valid Error Response (returns error response object)
- Modify Task – PUT /tasks/:id
  - Invalid API credentials
  - Valid/Invalid Task ID
  - Valid/Invalid Setting Changes (like setting an integer to a string or modifying an inapplicable setting for the device type)
  - Valid/Invalid Devices (ensuring no duplicate devices are added to a task)
  - Valid Success Response (returns modified task object model)
  - Valid Error Response (returns error response object)

Schedule Test Cases

- Add New Schedule – POST /schedules
  - Invalid API credentials
  - Valid/Invalid Schedule Name
  - Valid/Invalid Schedule Interval
  - Valid/Invalid Task ID
  - Valid Success Response (returns schedule object model)
  - Valid Error Response (returns error response object)
- List Schedules – GET /schedules
  - Invalid API credentials
  - Valid Success Response (returns list of schedule object models)

  - Valid Error Response (returns error response object)
- Delete Schedule – DELETE /schedule/:id
  - Invalid API credentials
  - Valid/Invalid Schedule ID
  - Valid Success Response (empty page, status code 204)
  - Valid Error Response (returns error response object)
- Modify Schedule – PUT /schedule/:id
  - Invalid API credentials
  - Valid/Invalid Schedule ID
  - Valid/Invalid Schedule Name
  - Valid/Invalid Schedule Interval
  - Valid/Invalid Task ID
  - Valid Success Response (returns modified schedule object model)
  - Valid Error Response (returns error response object)

Energy Management Test Cases

- List Energy Graphs – GET /energy
  - Invalid API credentials
  - Valid Success Response (returns list of energy usage stats models)
  - Valid Error Response (returns error response object)
- List Goals – GET /energy/goals
  - Invalid API credentials
  - Valid Success Response (returns list of energy goal models)
  - Valid Error Response (returns error response object)
- Add New Goal – POST /energy/goals
  - Invalid API credentials
  - Valid/Invalid Device (one of device, task, default graph type)
  - Valid/Invalid Task (one of device, task, default graph type)
  - Valid/Invalid Graph Type (one of device, task, default graph type)
  - Valid/Invalid Goal Name
  - Valid/Invalid Goal Threshold
  - Valid Success Response (returns goal object model)
  - Valid Error Response (returns error response object)
- Delete Goal – DELETE /energy/goals/:id
  - Invalid API credentials
  - Valid/Invalid Schedule ID
  - Valid Success Response (empty page, status code 204)
  - Valid Error Response (returns error response object)
- Modify Goal – PUT /energy/goals/:id
  - Invalid API credentials
  - Valid/Invalid Device (one of device, task, default graph type)
  - Valid/Invalid Task (one of device, task, default graph type)
  - Valid/Invalid Graph Type (one of device, task, default graph type)
  - Valid/Invalid Goal Name

- Valid/Invalid Goal Threshold
- Valid Success Response (returns modified goal object model)
- Valid Error Response (returns error response object)

6.2.2. Software with Devices Testing

Since the automated tests can only check if the API is functioning, each device must be tested manually to ensure it is functioning properly.

For each device, it will be connected manually to the mobile app to be tested. Once linked, the device will be tested by sending commands to it through the API within the app on the devices page. Since the API reuses the same classes internally for sending data to each of the automated devices, only the devices page needs to be tested here (it is pointless to also test whether tasks and scheduled tasks also run).

For wall outlets and light switches, this involves turning them on and off. For HVAC, the app must successfully turn the system on and off, set its fan and auto modes, set the temperature, and swap between heating and cooling. For the powerlock, the app must be able to unlock a door. If all of the devices perform as desired, then the entire system is operating as expected.

7. Administrative Content

The way we chose to break down the workload was to have two members working on the electrical aspect of the project, and two members working on the software development aspect. This made it so that each part of the project had two members working on it, as having only one member assigned to a part would have been a daunting task. Since our group was comprised of three computer engineers and only one electrical engineer, one of the computer engineering majors who had more of an interest in hardware was assigned to work on the electrical aspect. This allowed all members to expand their knowledge in fields they may be interested in working in the future.

Labor on the project is divided amongst the four team members according to the nature of the task. Zachary Zapasnik is in charge of writing the web and mobile applications, creating and managing the database, and testing all of the applications. Jeffrey Benoit is in charge of all microcontroller programming and its communication to the web and mobile applications, as well as testing the microcontroller itself. D'Voran McIntosh and Roneal Valmonte are in charge of all tasks related directly to the hardware. D'Voran is in charge of purchasing and parts research, which Roneal is in charge of circuit and device design. Both D'Voran and Roneal together are in charge of hardware construction and circuit testing. The entire responsibilities list is provided in figures 7.A and 7.B.

Main Component	Subcomponents	Members Responsible
Wall outlet		
	PCB	Roneal Valmonte/D'Voran McIntosh
	Microcontroller	Jeffery Benoit then Roneal Valmonte/D'Voran McIntosh
	Relays	Roneal Valmonte/D'Voran McIntosh
	WiFi and Bluetooth Module	Roneal Valmonte/D'Voran McIntosh
	Rectifier	Roneal Valmonte/D'Voran McIntosh
	Energy Measurement Integrated Circuit	Roneal Valmonte/D'Voran McIntosh
Wall switch		
	PCB	Roneal Valmonte/D'Voran McIntosh
	Microcontroller	Jeffery Benoit then Roneal Valmonte/D'Voran McIntosh
	Relays	Roneal Valmonte/D'Voran McIntosh
	WiFi and Bluetooth Module	Roneal Valmonte/D'Voran McIntosh
	Rectifier	Roneal Valmonte/D'Voran McIntosh
	Energy Measurement Integrated Circuit	Roneal Valmonte/D'Voran McIntosh

Figure 7.A - Group Member Responsibilities

Main Component	Subcomponents	Members Responsible
Powerlock		
	Electric Strike	Roneal Valmonte/D'Voran McIntosh
	Relay	Roneal Valmonte/D'Voran McIntosh
	WiFi and Bluetooth Module	Roneal Valmonte/D'Voran McIntosh
	Power supply	Roneal Valmonte/D'Voran McIntosh
	Microcontroller	Jeffery Benoit then Roneal Valmonte/D'Voran McIntosh
Application		
	Application	Zachary Zapasnik

Figure 7.B - Group Member Responsibilities

7.1. Milestones

Below is a timeline that discusses our projected milestones. We estimated the time we believed each part would take based on projects that we had completed in the past. These predicted times are tentative, and our final timeline may have to be adjusted depending on how we actually proceed with the project.

7.2 Proposal

Below is a copy of our proposal to Leidos.

Project Description

Home automation has been seen as a futuristic idea, mostly inspired by movies and cartoons over the years. Only recently has it become a possibility to actually create such automation through the rise of the Internet of Things (IoT). The IoT is the idea that all devices can be networked to work together seamlessly, each having its own purpose and tasks.

Event	Expected Time to Complete Event	Expected Start Dates and Expected Completion Dates
Form Group/Pick Desired Project	1 week	August 24rd – August 31th
Research Existing Products/Projects	2 months	September 3rd – November 10th
Decide on Project Aspects	1 month	September 3rd – October 2nd
Research Necessary Technologies	2 months	September 3rd – November 10th
Research Necessary Components	2 months	September 3rd – November 10th
Decide on Components	3 months	September 3rd – December 1st
Leidos Proposal		Due September 24th
Order Samples	2 months	September 3rd – November 19th
Order Parts	1 month	December 1st – January 11th
Design Wall Outlet	1 month	December 1st – January 11th
Design HVAC Controller	1 month	December 1st – January 11th
Design Powerlock	1 month	December 1st – January 11th
Design Wall Switch	1 month	December 1st – January 11th
Design Circuit Board	1 month	December 1st – January 11th
Test Circuit Board	2 months	December 1st – February 1st
Application Development	4 months	December 1st – April 1st
Application Testing	4 months	December 1st – April 1st
Microcontroller Coding	2 months	December 1st – February 1st
Microcontroller Testing	3 months	December 1st – March 10th
Senior Design 1 Paper		Due December 10th
Construct First Prototypes	1 month	January 11th – February 12th
Test First Prototypes	1 month	January 11th – February 12th
Construct Second Prototypes	1 month	February 12th – March 11th
Test Second Prototypes	1 month	February 12th – March 11th
Construct Third Prototypes	1 month	March 11th – April 15th
Test Third Prototypes	1 month	March 11th – April 15th
Final Presentation	1 week	April 28th – May 4th

Figure 7.1.A - Timeline

For this project, IoT technology is being embraced to make a simplified home automation system. Independently designed hardware devices will be hooked up to the Internet over in-home WiFi networks. Over WiFi, these devices will connect to a cloud-hosted infrastructure allowing households to register accounts on a platform by which devices can be managed and configured through web and mobile apps in addition to a provided in-home tablet.

With the cloud-hosted infrastructure, these devices can be turned on and off individually or as assigned virtual groups (like to perform a specific task like turning an entire room's lights off). Due to limited time and resources, the primary use case of these devices will be to control lights and fans in a home. This project could be extended to control air conditioning, door locks, appliances, and other devices.

Project Goals

The main goal of this project is to create a cheap and efficient home automation system through multiple wireless devices designed for specific tasks (such as turning a switch on and off).

These devices will be used to allow homeowners to control lights and fans from any room in their house, in addition to remotely via a web application or mobile device.
In addition to added control, these devices will also monitor, record, and graph power usage. This will allow homeowners to better account for their energy use.

To further limit the amount of energy used, motion detection and audio recording devices can be used to automatically turn off the power to devices in empty rooms.

Project Specifications

There are some constraints for this project. It must be assumed that the user will have a wireless network and Internet in their home. Additionally, the user must be capable of installing wall switches, which involves basic household wiring.
As for competing products, commercial in-home technologies exist like Vera, Wink, Control4, and many more. These competing products offer a similar setup, although some are not cloud based services like the one proposed for this project.

7.3 Budget

Below is a copy of our projected budget. In order to come up with these figures, we used information we gained from our research of components, as well as past

experience with project budgets. This budget is also tentative, and may not reflect the actual budget of our final product.

Item	Quantity	Expected Cost
Microcontroller	4	$50
WiFi and Bluetooth Modules	4	$100
Energy Measurement Integrated Circuit	3	$10
Electric Strike	1	$50
Relays	3	$10
Wiring	n/a	$20
Housing for Wall Outlet	1	$15
Housing for Wall Switch	1	$15
Housing for Powerlock	1	$15
Housing for HVAC Controller	1	$15
Rectifiers	4	$30
Shipping and Handling	n/a	$100
Soldering Materials	n/a	$50
Web Hosting	n/a	$60
Printing/Binding for Senior Design I	1	$20
Total Estimate		$560

Figure 7.3.A - Projected Budget

8. Appendices

8.1. Appendix A: Works Cited

AngularJS Developer Guide. (n.d.). Retrieved November 23, 2015, from https://docs.angularjs.org/guide

Angular UI-Router Wiki. (n.d.). Retrieved November 23, 2015, from https://github.com/angular-ui/ui-router/wiki

Atmel Corporation. (2015, September). SAM C20E Datasheet Preliminary. Retrieved from http://www.atmel.com/images/Atmel-42364-SAMC20_Datasheet.pdf.

Bootstrap Documentation. (n.d.). Retrieved November 15, 2015, from http://getbootstrap.com/

ExpressJS Documentation. (n.d.). Retrieved November 15, 2015, from http://expressjs.com/en/

Maxim Integrated. (2012, January). 78M6613 Single-Phase AC Power Measurement IC Datasheet. Retrived from http://www.datasheets.maximintegrated.com

MongoDB Documentation. (2015, December 3). Retrieved December 1, 2015, from https://docs.mongodb.org/manual/MongoDB-manual-v3.0.pdf

MongooseJS Documentation. (n.d.). Retrieved December 1, 2015, from http://mongoosejs.com/docs/guide.html

PhoneGap Bluetooth Plugin. (n.d.). Retrieved November 27, 2015, from https://github.com/bcsphere/bluetooth

Postman Documentation. (n.d.) Retrieved December 3, 2015, from http://www.getpostman.com/docs/

Texas Instruments. (2015, December). WL18xxMOD WiLink 8 Single-Band Combo Module. Retrieved from http://ti.com/lit/swrs152l/swrs152l.pdf

Smart Outlet Handbook. (n.d.). Retrieved November 23, 2015, from http://postscapes.com/smart-outlets

Smart Air Vents. (2014, September 11). Retrieved December 7, 2015, from https://community.smartthings.com/t/smart-air-vents/4913

mbed TLS Core Features. (n.d.). Retrieved December 1, 2015, from https://tls.mbed.org/core-features

NEMA DC 3 - 2013. (2014, January 27). Retrieved December 1, 2015, from https://www.nema.org/Standards/Pages/Residential-Controls-Electrical-Wall-Mounted-Room-Thermostats.aspx

NEMA DC 3 Annex A - 2013. (2014, January 27). Retrieved December 1, 2015, from https://www.nema.org/Standards/Pages/Energy-Efficiency-Requirements-for-Programmable-Thermostats.aspx

NEMA 5-15 grounded plug. (2013, July 31). Retrieved December 1, 2015, from http://www.nema.org/Standards/Pages/Wiring-Devices-Dimensional-Specifications.aspx

NEMA Enclosure Types. (n.d.) Retrieved December 1, 2015, form https://www.nema.org/Products/Pages/Enclosures.aspx

Atmel SAM C specifications (n.d.) Retrieved December 1, 2015, from http://www.atmel.com/products/microcontrollers/arm/sam-c.aspx

WeMoÂ® Insight Switch. (n.d.). Retrieved November 20, 2015, from http://www.belkin.com/us/p/P-F7C029/

Smart Outlet | ConnectSense. (n.d.). Retrieved November 20, 2015, from https://www.connectsense.com/smart-outlet

WeMo® Light Switch. (n.d.). Retrieved November 20, 2015, from http://www.belkin.com/us/p/P-F7C030/

GE12722 Z-Wave Wireless Lighting Control On/Off Switch. (n.d.). Retrieved November 20, 2015, from http://www.amazon.com/GE12722-Z-Wave-Wireless-Lighting-Control/dp/B0035YRCR2

The Key Evolved. (n.d.). Retrieved November 20, 2015, from http://www.kwikset.com/kevo/default.aspx#.Vmc01rgrJhE

ACUS-12V - AC 90-240V input power adapter for FRM220, FIB1 and FMC series converters. (n.d.). Retrieved November 20, 2015, from http://datainterfaces.com/ACUS-12V.aspx

August Smart Lock | August. (n.d.). Retrieved November 20, 2015, from http://august.com/products/august-smart-lock/

Honeywell Home. (n.d.). Retrieved November 20, 2015, from http://lyric.honeywell.com/thermostat/

MCP3909 Datasheet. (n.d.). Retrieved November 20, 2015, from http://ww1.microchip.com/downloads/en/DeviceDoc/22025C.pdf

Narrow-Type Electric Strike Lock for Home Office Wood Metal Door NO Mode Fail Secure DC 12V Access Control. (n.d.). Retrieved November 20, 2015, from http://www.amazon.com/Narrow-Type-Electric-Strike-Office-Control/dp/B0105WEFBW

Relay Construction. (n.d.). Retrieved November 20, 2015, from http://www.allaboutcircuits.com/textbook/digital/chpt-5/relay-construction/

UC3610N Datasheet. (n.d.). Retrieved November 20, 2015, from http://pdf.datasheetcatalog.com/datasheet2/8/0i5ziwr5ja76xi7e93t3p17w2ayy.pdf

Ecobee3, HomeKit-Enabled. (n.d.). Retrieved November 20, 2015, from https://shop.ecobee.com/products/ecobee3-homekit

8.2. Appendix B: Permissions

Atmel

Jessen, Ralph <Ralph.Jessen@atmel.com>
to me

Roneal,

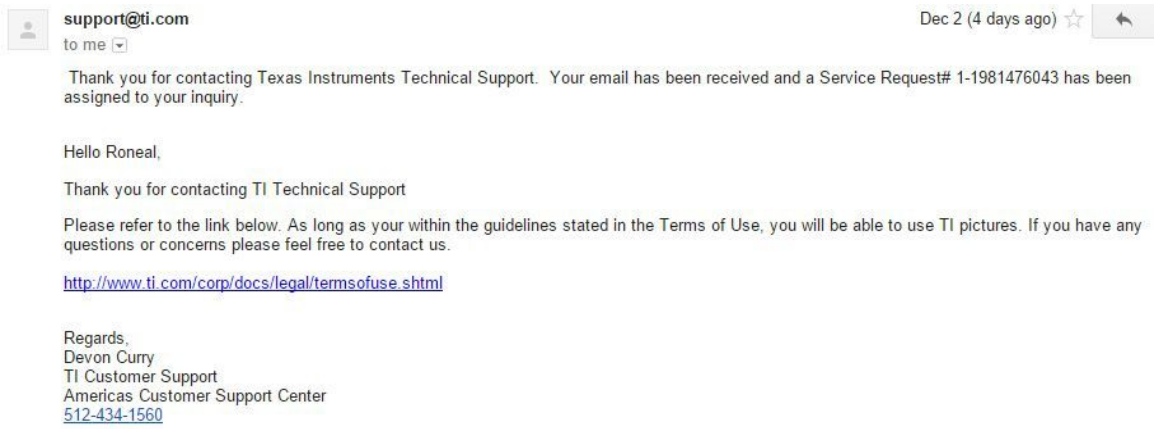
We approve this request provided that they are credited.

Thank you for choosing Atmel Corporation

Ralph Jessen
Account Manager-Global Key Accounts / Atmel Corporation
Tel: (+001) (408) 436.4120 / Mobile: (+001) (408) 489.4496 / Fax: (+001) (408) 436.4200
Ralph.Jessen@atmel.com / www.atmel.com

Atmel | Enabling Unlimited Possibilities®

Texas Instruments



support@ti.com Dec 2 (4 days ago)

to me

Thank you for contacting Texas Instruments Technical Support. Your email has been received and a Service Request# 1-1981476043 has been assigned to your inquiry.

Hello Roneal,

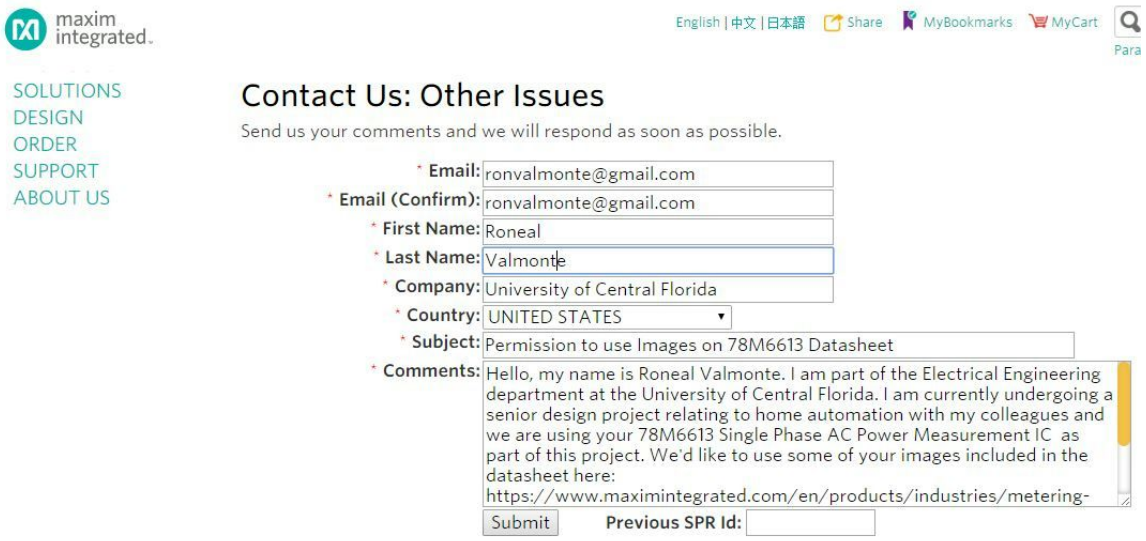
Thank you for contacting TI Technical Support

Please refer to the link below. As long as your within the guidelines stated in the Terms of Use, you will be able to use TI pictures. If you have any questions or concerns please feel free to contact us.

http://www.ti.com/corp/docs/legal/termsofuse.shtml

Regards,
Devon Curry
TI Customer Support
Americas Customer Support Center
512-434-1560

Maxim Integrated (Pending)



maxim integrated.

English | 中文 | 日本語 Share MyBookmarks MyCart Para

SOLUTIONS
DESIGN
ORDER
SUPPORT
ABOUT US

Contact Us: Other Issues

Send us your comments and we will respond as soon as possible.

* Email: ronvalmonte@gmail.com
* Email (Confirm): ronvalmonte@gmail.com
* First Name: Roneal
* Last Name: Valmonte
* Company: University of Central Florida
* Country: UNITED STATES
* Subject: Permission to use Images on 78M6613 Datasheet
* Comments: Hello, my name is Roneal Valmonte. I am part of the Electrical Engineering department at the University of Central Florida. I am currently undergoing a senior design project relating to home automation with my colleagues and we are using your 78M6613 Single Phase AC Power Measurement IC as part of this project. We'd like to use some of your images included in the datasheet here: https://www.maximintegrated.com/en/products/industries/metering-

Submit Previous SPR Id: