KnightHome – Home Automation System

Jeffrey Benoit, D'Voran McIntosh, Roneal Valmonte, Zachary Zapasnik

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The objective for this project is to create a Home Automation System that allows consumers to not only control different appliances in their home wirelessly, but also allows them to view the energy consumption of these appliances through either a smart phone application or web application. This will be accomplished with a smart outlet, smart wall switch, powerlock, and HVAC controller. This paper will explain our motivations as well as the components we have chosen and the methodology behind our software programming.**

***Index Terms* — Access control, energy measurement, microcontrollers, relays, smart homes, thermostats.**

I. Introduction

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.)

II. Goals and Motivation

This project started with the idea of being able to control the electricity in your home from anywhere. The idea for 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. We wish to make it so that users would not have to worry about leaving appliances turned on in their home if they left. This also drove us to include our 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. 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 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.

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.

III. Hardware – Components

Because one of the goals of the home automation system is to have all the devices be cheap and affordable to the average consumer, it's important for the components inside the device to be universal and share common circuitry. Doing so will drop the price of production drastically as well as making the devices easier to configure and set up during the fabrication process. Below are the main components utilized. Common components such as resistors, capacitors, and diodes aren't mentioned but will be shown in the schematic of the devices.

A. MCU – Particle Photon

In order for the website or smartphone app to control the four devices, a signal needs to be sent to the device to turn on or off and to collect any ongoing data such as energy measurement or temperature. This requires a microcontroller that will be the center hub of all of the devices. This will allow all of the components surrounding the microcontroller to act accordingly, such as the voltage regulator in the powerlock or the temperature sensor on the HVAC controller. Below is a model of the microcontroller as well as a legend of the various parts around the MCU. Labels for each of the pins are also shown in Figure 1.

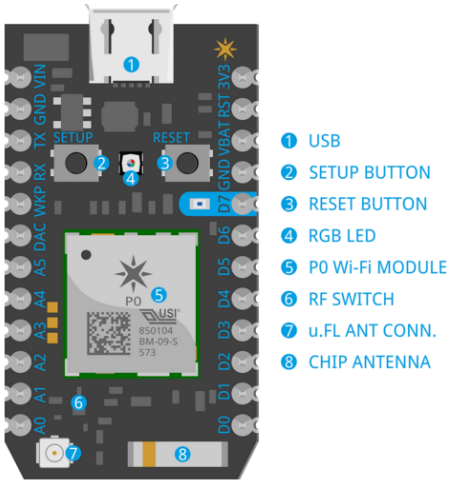


Figure 1. Particle Photon MCU

The microcontroller requires between 3.2V to 5V to turn on, which is supplied by a USB power adaptor that is connected to the microUSB port on the MCU. The GND pin is a ground pin used for grounding many components surrounding the MCU, such as the relay and current sensor. A0-A5 are analog I/O pins which the current sensor utilizes to input data into these pins, having the MCU read the voltage and currents, thus providing a power output, as given by the power equation P = VI. D0-D7 are used for connecting to the relay, turning on/off the devices when needed. [1]

B. Relay – Sunfounder 5V Relay Shield Module

Telling a device to turn on/off needs a relay module in between the MCU and the device. Because the relay needs 5V to operate, the MCU's Vin port would be able to provide the relay with the voltage it needs to operate. The relay runs using active-low, meaning a '0' sent to the relay will turn on the relay. When the relay is turned on, a red LED lights on either side of the relay, depending on which input the relay is turning on (because the wall outlet has two sockets, a two channel relay is needed, thus having two inputs IN1 and IN2).

C. Power Sensor – ECS1030-L72 Split Core Current Transformer

A vital component to the energy management home automation system is easily the component that traces the current and voltage going into the MCU and outputs this data onto the website or smartphone. Because small components can not take in huge current and voltages coming from the wall (120V 30A), a bigger sensor would be needed. Because of this, KnightHome utilizes a non-invasive current sensor that is attached onto the wire going into the relay. The wires of the sensor go through a series of burden resistors for compatibility with the MCU and finally goes to the analog pin on the MCU. Because this non-invasive current sensor also measures voltage across the wire, the power can be measured and this data automatically pops up on the website or smartphone app. As shown in Figure 2, the power is proportional to the current and voltage in a linear fashion. [2]

D. LCD Screen – Lumex LCM-S01602DSF/A

On top of showing the temperature of the room on the website/app, there will also be a small LCD screen on the HVAC Controller that will display that same temperature. The digital I/O pins on the MCU are used to control the

segments on the LCD screen. The screen uses very minimal power (4.7V-5.3V) and thus can use the Vin pin on the MCU to power the device, just as the relay does. [3]

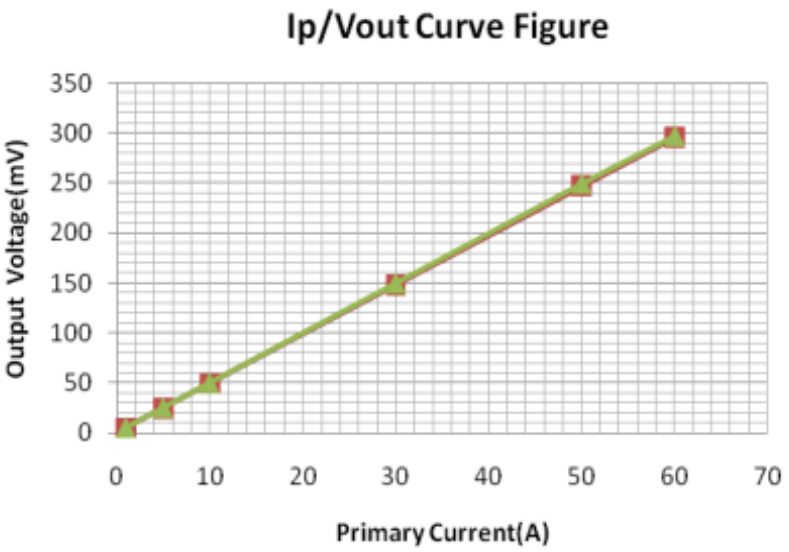


Figure 2. Current Sensor Output Graph

IV. Hardware – Devices

KnightHome uses four basic components to a home automation system that are all small, compact, and can be kept in inconspicuous areas around the house to keep them out of sight from dangers such as children and animals. The four devices are all Wi-Fi enabled and controlled through the KnightHome website or the KnightHome smartphone application. These devices can be grouped together according to what room it is if there are multiple KnightHome devices around the house.

A. Wall Outlet

Just as it is simple to plug your electronic devices into the wall, the KnightHome Smart Outlet allows anyone to plug whatever device they want into the wall outlet. Each plug on the wall outlet is individually controlled by the website/smartphone and the power monitoring can also be seen.

The wall outlet incorporates the Particle Photon MCU, 2-channel relay (since there are two plugs), and two current sensors (again, one for each plug). The current sensor is bridged to the MCU through a burden circuit that drops down the current and voltage to a level that can be read by the MCU and interpret this data easier. The circuit includes a voltage divider using 47kΩ resistors, a 10μF capacitor for filtering noise coming from the sensor, and a Burden resistor that was calculated to the 77.79 ohms, so a standard value of 75ohms is used for this circuit. The wall outlet is powered via microUSB by the microUSB port on the MCU that is connected to a standard USB wall adapter that is connected to the wall plugs. This means when the Smart Outlet is plugged in, the USB power adapter provides power to the MCU, thus turning on the Smart Outlet.

B. Wall Switch

Sometimes it isn't possible to have the website around or your smartphone around to operate the wall outlet. This is why the Smart Outlet is coupled with a wall switch that can override the website/smartphone's control and turn on/off devices. The wall switch acts like a typical light switch that is in most houses nowadays; the switch toggles between on and off. The switch is connected to a circuit that replicates the Smart Outlet.

C. Power Lock

Controlling the doorknob on the door wirelessly requires an electric strike. An electric strike is a mechanism between the door hinge and door knob that disallows the doorknob to work when the strike is off. A 12V input to the electric strike will turn it on, allowing the doorknob to be accessible. KnightHome utilizes an electric strike connected to a 1 channel relay that is connected to the MCU.

Because the electric strike uses 12 V and the MCU uses 5 volts, a voltage regulator is used between the 12V source and the MCU. The voltage regulator is a 7805 positive voltage regulator that inputs any voltage below 20V and outputs a fixed 5V. This allows both the MCU and electric strike to be on at the same time without any power issues. The powerlock is cased in a small box that hangs next to the door and uses 8 AA batteries (12V) that can be easily replaced.

C. HVAC

KnightHome offers a mini thermostat that tracks the temperature of the room and raises/lowers the temperature according to user settings on the website/smartphone. The HVAC Controller uses the Lumex LCM-S01602DSF/A LCD Screen to display the temperature while the website/smartphone controls the HVAC controller. The LCD Screen has several pins for the segments on the

display. These are connected to the D0-D7 pins on the MCU. In order to sense the temperature of the room, an M35CZ temperature sensor IC is connected to the A0 pin on the MCU to constantly track temperature and relay this information to the MCU. Again, a two channel relay is also connected in the same fashion as the Smart Outlet. However, this is used to control a fan and heater to simulate heating and cooling a room.

V. Software – Application

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. Figure 3 gives an application design overview of the entire application flow.

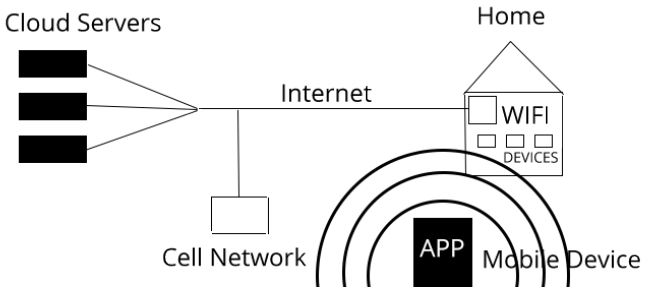


Figure 3. Software Overview

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.

A. Backend Software

The home automation application, both frontend and backend exist across multiple cloud servers. Using multiple cloud servers across multiple locations increases redundancy in the case a single virtual instance fails. 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 KnightHome, DigitalOcean is where the cloud is 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.

Each cloud server is 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.

The webserver is reachable from the internet by both the frontend clients as well as the automated devices. We are using 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 is 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 interface with only the backend API, not the frontend application being served by NGINX.

The backend of the application functions as an accessible programming interface (API) for the frontend clients and automated devices to communicate with. Since the client applications are utilizing JavaScript, a JavaScript Object Notation (JSON) API was created for clients to utilize. To create our RESTful JSON API, we used Node JS to create a backend service. Node JS was compiled and runs under a restricted user account on each cloud instance. This ensures that a vulnerability in our software does not compromise the entire server instance. Our backend service is deployed to the cloud servers using GIT, a version control system for software development.

Our API primarily makes use of ExpressJS, a NodeJS open source web framework/routing library, in order to create route modules to service the tasks necessary for our home automation devices to function.

In addition to the standard JSON API we've created, we also created a keep-alive socket API so that home automation devices and send and receive commands and

stats in real-time without constantly polling our JSON API. For toggling settings, setting changes get emitted to the device authenticated on the socket when they are changed. Vice versa, if the device contains a manual button (for instance, to override the app setting) then the device can also send the setting change and it will be saved to the database as a setting change. For stats, the device can send stat updates an any interval to be logged to our database to be analyzed. This is useful for power monitoring, one of the main focuses of our home automation system.

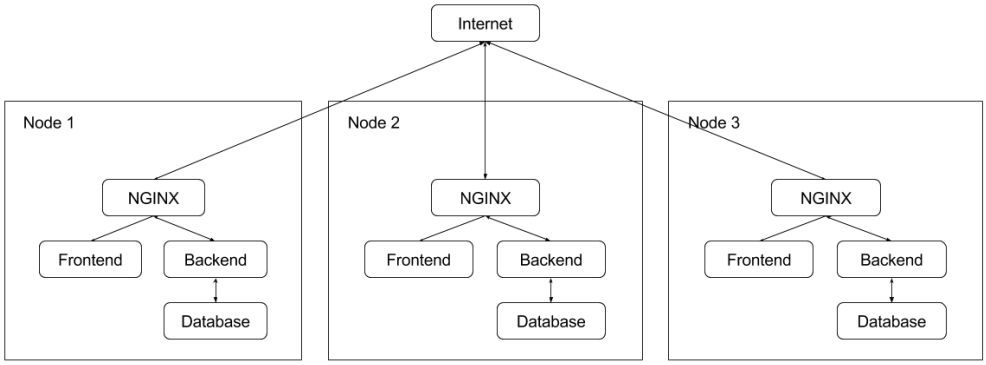


Figure 4. Cloud Infrastructure Overview

Speaking of databases, database design is a crucial aspect of any web application. For our application, an open source document storage database called MongoDB is 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 (useful for graphing and analyzing data for power use).

In order to ensure a stable cloud hosted database, we needed to ensure our data storage was 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 chose 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 succumb to corruption the data will remain available for use by the applications. Each of these instances communicate with each other over a TLS encrypted socket using key-based authentication. The entirety of our cloud-based backend infrastructure is depicted in Figure 4.

B. Frontend Software

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

AngularJS is 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 are 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).

Since we are 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 menubar (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 rest of the application

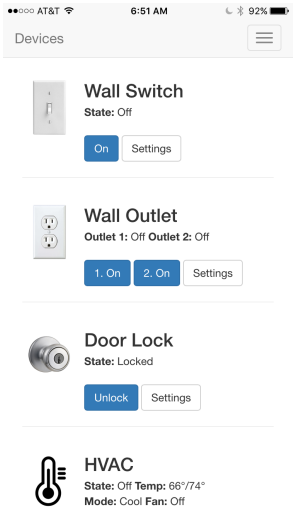


Figure 5. App Device Management

is composed of individual pages devoted to specific tasks: configuring devices, configuring tasks, managing schedules, and energy usage monitoring.

For the devices page, new devices can be added and existing devices can have their common settings toggled at a glance. Each device also reports its current state to the screen, making it easy to see what devices are active. The device managerment page is shown in Figure 5.

Managing individual devices can be tedious if you want multiple to work together. We've added a grouping feature that allows you to pair devices into a task. This allows you to preconfigure settings you'd like for every device in the task, making it easy to set devices in one swoop.

Sometimes you might want to also automate these tasks on a schedule, so we've also integrated a scheduler that allows you to schedule one-time or repeating tasks.

Energy monitoring is another great feature of our application. It shows you to total energy consumption of all devices, predicts your energy usage for the month, and also allows you to show each individual devices energy usage in a graph. Graphing energy use allows us to better depict peak energy usage times during your average day, which might help you to better reduce your energy footprint during those events.

VI. Software – Microcontroller

Software for the microcontroller is written in C using Particle's web IDE and their standard application library. Each of the 4 devices in KnightHome is programmed with the same structure in mind, but the code for each type of device is unique to it.

A. START UP

All 4 of the KnightHome devices follow the same procedure upon startup. The device will first connect to the supplied Wi-Fi network, no other action will occur until a Wi-Fi connection is established. Once connected the device initializes all variables and settings to the off position except for the enable setting, which is set to on. Next the device sends authentication and identification information to the server through a TCP socket. Once the device is properly identified and the authentication is accepted, the device receives settings data as a response from the server and begins its main function. All main functions begin with checking the connection status; the procedures for handling disconnecting and reconnecting are covered later in this section, these procedures aren't mentioned in the device sections as they are ideally never triggered.

B. Wall Outlet

The wall outlet is not the simplest of the 4 devices in terms of the program running it, however, it serves well as a skeleton for the code for the other devices to be molded to.

The main phase of the wall outlet first checks for any messages sent from the server. If a new message is detected the device reads the message(s) from the server as character arrays, parses the array, and takes the appropriate action. For the wall outlet these messages include receipts of authentication, server pings, and state enables for each of the two wall sockets on the outlet. When the device receives an enable signal from the server it sends an appropriate output to the pin corresponding to which wall socket was indicated by the message, changing the state of the relay connected to that pin. Once all of the messages have been read from the socket the socket the socket is flushed to prevent old messages from remaining in the socket.

The wall outlet then reads input from the current sensors attached to the sockets. The input from these sensors is sampled many times as an analog input which is then averaged and converted into volts. The voltage measure is between 0 and 3.3 V due to the limitations of the microcontroller, so that voltage is converted back to its original voltage before the stepdown. That voltage is the input received from the current sensor, and as such is then converted into a measure of current. The measured current is converted to wattage under the basis that the wall outlet device is connected to a standard home wiring of 120V. This wattage is sent to the server as the measure of the present power consumption of the outlet.

The main process then loops indefinitely. Should the entire device lose power it will start again from the start up process described earlier.

C. Wall Switch

The wall switch expands on the wall outlet code by adding the ability to toggle the device's state locally. As such this device adds a variable that keeps track of the current local state of the device by reading an input from the relay controlling the electrical flow. The order of the main function is important to make sure that a local input is not immediately overwritten by the state of the server before said change can be sent to the server.

The main function of the wall switch begins the same as the wall outlet by checking for new messages from the server and taking the appropriate action should a new message be present. For the wall switch, only authentication, ping, and enable messages are received. The process for parsing these messages and the actions

taken are the same as for the wall outlet. Once all of the messages have been read the socket is flushed to ensure the removal of old messages.

After checking for messages from the server, the device then reads the input from the relay to determine the local state of the relay. This state is then sent to the server as a message.

The same power monitoring procedure as the wall outlet is used for the wall switch. The current is calculated based on the voltage read from the input of the current sensor, and said current is converted into watts. The watts measurement is sent to the server in a message as the devices present power usage. It is important that the power management data is sent after the state data because the server response to the power monitor data can include enable messages which would overwrite the local change if received before the server was set to the proper state.

D. Power lock

The powerlock is the simplest of the 4 devices from a code perspective. The power lock has its own power supply, and since it does not draw power from the wall it does not have the same power monitoring functionality as the previously mentioned devices. As it is meant for security it does require its own specific procedures. This device also has a small change to the startup phase: the LED on the MCU is initialized as on. This is done as a power indicator of sorts since the power lock has its own power supply, if the device has power the LED will be on.

The main function of the power lock begins the same as the wall outlet by checking for new messages from the server and taking the appropriate action should a new message be present. For the power lock, only authentication, ping, and enable messages are received. The process for parsing these messages and the actions taken are the same as for the wall outlet. Once all of the messages have been read the socket is flushed to ensure the removal of old messages.

The remainder of the main phase is procedures to ensure security. If Wi-Fi connection is lost the device is set to be locked to prevent a loss internet from leading to a breach of security. When the device is connected or reconnecting but has not yet received a message from the server it is set to be locked. This time period is very short and the device should never be in this state when this point in the main phase is reached.

E. HVAC Controller

The HVAC controller is the most complicated of the 4 devices. Not only are there more states to keep track of and more messages that can be received, the microcontroller also controls an LCD screen display.

The main function of the HVAC controller begins the same as the wall outlet by checking for new messages from the server and taking the appropriate action should a new message be present. For the HVAC controller messages for ping, authentication, device enable, AC enable, Heat enable, Fan state, and temperature are received. The process for parsing these messages is the same as for the wall outlet except that all state data is stored in an array called Settings which keeps track of the current state of the device rather than immediate action being taken. Each enable controls a relay attached to the HVAC system, the fan state has 3 possible states for controlling the procedures of the HVAC controller, and the temperature messages set the desired temperature of the device. Once all of the messages have been read the socket is flushed to ensure the removal of old messages.

AC enable and heat enable settings both control both the heat and AC relays. When a message comes in to turn on the AC, the AC relay is turned on and the heat relay is turned off, making sure that only the AC is running. The opposite occurs when a message is received to turn the heat on. These two settings can never both be set to on, and setting one of them to on sets the other to off. Both settings can be set to off at the same time, meaning you want neither heat nor AC running, which is why they exist separately. It should be noted that the device enable setting does not set these states to off, it causes the device to not read them when determining which action it should take, which allows the device to quickly return to its previous state when it is enabled again.

After checking for messages the device reads the temperature sensor. The temperature sensor input is received as analog input through an analog pin. The analog input arrives as an integer, which is then multiplied by 0.0008 to convert it into a voltage between 0 and 3.3V. This voltage is multiplied by 100 and then 50 is subtracted, resulting in the current temperature in ºC. Our devices are intended for use in the United States, so the temperature is then converted to ºF before being stored in the Settings array. The current temperature is then sent to the server.

The main phase then moves to act on the data in the Settings array. The device first reads the device enable setting, if this setting is off it sends an off signal through all relays. If the device enable is set to on the device continues through this phase. The device then reads the heat enable and AC enable states from the Settings array and sends an appropriate signal to the corresponding relays. The fan state setting is read next, and if it is set to either on or off the appropriate signal is sent to the corresponding relay. If the fan state is set to auto the device checks the heat enable and AC enable states and then the desired temperature against the current

temperature to determine whether the fan should be blowing or not.

After taking proper action the device displays the current contents of the Settings array on the LCD screen as the state of the device. The device first displays either heat, cool, or off in the top left, showing whether the heat or AC is on or if both are off. In the top right the word "set: " is displayed with the desired temperature in ºF displayed afterwards. On the bottom row on the far left the word "fan: " is displayed with the current fan state (on, off, or auto) displayed afterwards. In the bottom right the current temperature as read by the temperature sensor is displayed.

F. Handling Disconnects

Should the device ever become disconnected from the socket, a reconnect needs to occur. At the start of every main process there is a check for both the Wi-Fi connection status and the status of the connection to the socket. Should Wi-Fi ever be disconnected the device will attempt to reconnect using the previously entered Wi-Fi credentials until either new credentials are entered or a connection is reestablished. If the device ever becomes disconnected from the socket it will reconnect to the socket. A ping message is received from the server every 30 seconds, so should a device go 31 seconds without receiving a message from the server it will assume it has lost its connection to the socket and close the connection then reconnect. After the successful reconnection the authentication and identification information is re-sent to the socket so the device can be properly connected to its owner and it receives the current state from the server as a response.

VII. Conclusion

This project was an immense learning experience for all the members involved. We gained valuable experience as engineers, not only through the research and prototyping we had to perform, but as well as experience in working as a group for an extended period of time. This experience will be invaluable as we go off into our careers.

One major thing that was learned through this project is the importance of beginning early and working on a consistent schedule throughout the project life. Our work on the hardware aspect of the project was not on a regular schedule at the beginning of the semester, and as a result we fell behind and had to double up on our meetings in order to catch back up to a good position. Thankfully, we caught back up and are on schedule to complete all aspects of our project in time for our demonstration.

Acknowledgement

Our group would like to take this moment to thank the help of the following professors for guidance and instruction during our project. We kindly thank: Dr. Samuel Richie, Dr. Lei Wei, Dr. Chungyong Chan, and David Douglas. We would have have not been able to accomplish this without their support.

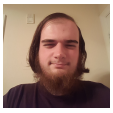
References

[1] Particle. (n.d.). Retrieved April 06, 2016, from https://docs.particle.io/datasheets/photon-datasheet/

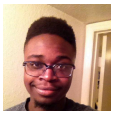
[2] Split Core Current Transformer ECS1030-L72. (n.d.). Retrieved April 06, 2016, from http://cdn.sparkfun.com/datasheets/Sensors/Current/ECS1030-L72-SPEC.pdf

[3] LCM-S01602DSF/A Datasheet. (n.d.). Retrieved April 06, 2016, from http://www.mouser.com/ds/2/244/LCM-S01602DSF A-108827.pdf

Meet The Engineers



Jeffrey Benoit, a senior student of the computer engineering department at the University of Central Florida, plans to pursue a job in the I.T. field after graduation.



D'Voran McIntosh, a senior student of the computer engineering department at the University of Central Florida, plans to pursue a career in the computer engineering field, as well as returning to pursue a master's degree in business administration.

Roneal Valmonte, a senior student of the electrical engineering department at the University of Central Florida, plans to pursue a Master's Degree in Electrical Engineering with a focus in Control Systems and Wireless Communication.

Zachary Zapasnik, a senior student of the computer engineering department at the University of Central Florida, is pursing a career in the software engineering field.