Wireless Home Control System

Reimagine your home ™

Grant Hernandez
Computer Engineer

Joseph Love
Electrical Engineer

Jimmy Campbell
Computer Engineer

University of Central Florida
August 3, 2015

Senior Design II Group #5
Sponsored in part by Boeing

Contents

List of Figures

List of Tables

1 Executive Summary

Wireless Home Control System (WHCS) is a solution for any homeowner to be able to remotely control core appliances of their home. WHCS allows the user to control lights, outlets, doors, and sensors around their home. The system's design philosophy emphasizes ease of use, affordability, and effectiveness. An Android phone application developed for WHCS allows users to monitor the state of the installed components and activate them remotely. A central base station equipped with a touch-enabled LCD is present, allowing the users to interact with the system without the need of a phone. Peripheral control modules may be installed into targeted appliances such as lights, outlets, and doors for WHCS to control.

The implementation of such a system required research in a myriad of fields to produce a full-fledged product. A well designed Android application is the key to creating a positive first impression of WHCS. Thus, care was taken to conform to the design philosophies of the Android ecosystem. The alternative interface offered for WHCS is the base station's display. Communication devices form the foundation for the wireless aspect of WHCS, thus an investigation into the advantages of different communication modules was required to realize the system. A network protocol has been developed and implemented in order to form a unified system from the independent modules. The activation of appliances around the home requires high voltage control, so methodologies for properly harnessing the power provided by homes were researched. To extend upon harnessing the home's power, our individual control modules and base station's logic level voltages (5V) depend upon the creation of an efficient way to step down the high voltage supplied from the home.

Wireless Home Control System is a solution targeting the masses and designed by few. Naturally such a system suffers from the constraints imposed upon the creators. Most prominent of all constraints are those stemming from economics. The development and production of WHCS must conform to the low budget available. Design decisions were made to minimize overall cost of the system to satisfy this constraint. WHCS has the potential for mass implementation if user reception is positive, therefore the design adheres to manufacturing constraints. The parts used in the system have been chosen so that they are widely available. Our boards and parts have been designed so that they are easy to replicate and manufacture. With a product such as WHCS health and safety is clearly an issue. The system is meant to be installed inside the home where the user will feel at ease with the system installed. Thus it is ethical for us to put effort into making the system safe to use. Things such as controlling the home's high voltage must be done in a safe manner.

2 Project Description

2.1 Motivation

The goal of this project was to improve the quality of life for people in their homes. Imagine sitting on the couch at home about to watch a movie, but all the lights are on and it's a little warm inside. It is irksome to have to get up and turn off every individual light. With the technology existing today it is perfectly feasible to be able to turn off the lights and

turn on a fan with a mobile phone. With the software available today it is even possible for this process to be initiated by voice. The problem that exists is these solutions lack mass implementation. By creating a wireless home control base station that a mobile phone could connect to these visions can be realized. The need to get up and physically interact with an appliance can be made a thing of the past.

We developed an easy to use system that allows people at their home to interact with their appliances without having to be in front of them. Our aim was for the solution to be reliable and low cost. The use case scenarios were intuitive so that even someone who was just visiting could utilize the system. A person using WHCS can turn on their lights or an outlet with the press of a button or with a voice command from their mobile phone. They can also turn on their coffee pot from their phone when they first wake up. If someone knocks on the door the person can unlock the door without having to get up. With the activation capabilities of WHCS there is an opportunity to utilize a foundation that can be expanded upon. We created the infrastructure for integrating different types of sensors into the home to provide users with information about things like temperature or air quality.

2.2 Overview

The diagram pictured in Figure 1 shows the highest level overview of WHCS. When the user wants to begin interacting with WHCS he has the option of choosing to use a mobile phone or the included LCD screen. Both options provide full capabilities for interacting with the system. The phone is connected to the system through a BlueTooth connection that is created by the user in the WHCS application. Using the phone is a more mobile and easy method for access because the LCD will be connected to the central component of WHCS, the base station. The base station is the brain of WHCS. It is the base station's job to take commands from the user and relay them to the endpoints, while also displaying the state of the system. The base station has a list of endpoints, also called control modules, that can be targeted by the system. This list is dynamic and allow for endpoints to be added or removed from the system during operation. Together the base station and the control modules form a network through a home and communicate wirelessly to one another through radio transceivers.

The control modules designed for WHCS allow for all the activation of appliances around the house. These endpoints are constantly listening for commands from the base station via a radio transceiver. Each control module is tailored for interacting with a certain device. There are control modules for toggling outlets, toggling lights, unlocking the front door, and also for monitoring sensors. The control modules are as similar as possible with a designated area that allows for assigning specific roles to the control modules.

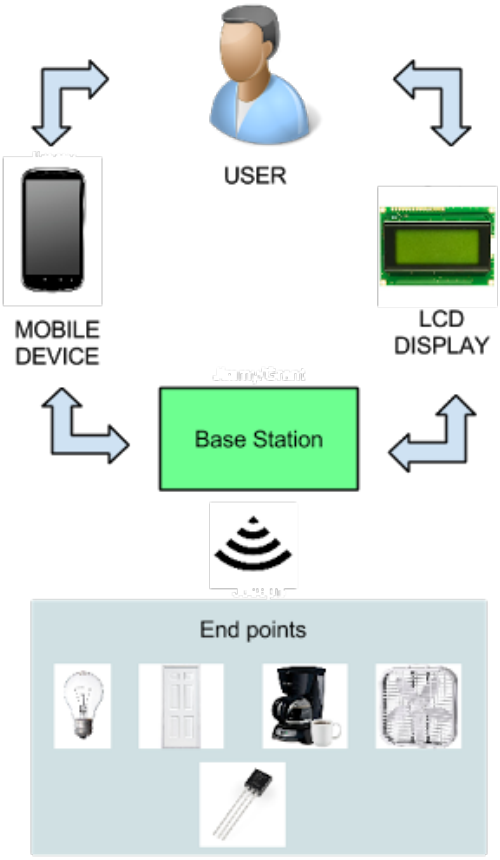


Figure 1: WHCS System Overview

2.3 Objectives

In order to enable homeowners to have the best experience with their new WHCS, we explain our core project objectives. These describe what the end-users are able to do with the system at a high level.

2.3.1 Voice Control

Voice control from a supported, BlueTooth enabled, Android device allows the user to remotely activate any part of the home that is integrated with WHCS. This would include activating lights, unlocking doors, turning off and on appliances (by controlling their respective outlets), querying sensors, and any other home specific applications.[1] All of these *actions* and *targets* are able to be used just from the user's voice. Voice actions are specific to each target, but they also consist of generic verbs such as **on**, **off**, **open**, **close**, and so on. The list of targets directly correspond to the number of control modules listed in the home and their type. This this explained in more detail in Section 4.

[1] WHCS is an extensible system. Control modules are built with a plugin-like interface, allowing for intrepid home owners to have a fully custom home. This combined with the control module's free breadboard area, new applications may be created.

2.3.2 Light Activation

Through activating lights and querying their status remotely, a homeowner no longer has to be present in the same room as the switch. By connecting lights to WHCS, they become integrated in to the home network and not be isolated in each room of the home. With just a spoken command or a tap on their smartphone, lights may be controlled. By automating the process of toggling light switches, WHCS has the ability to be smart about when they are ON or OFF, freeing the user from having to think about their state at all.

2.3.3 Outlet Activation

Lights aren't the only actionable thing in the home. There are a multitude of appliances throughout the home which could benefit from remote control. Some of these include coffee makers, toasters, or computers. If integrated with WHCS through outlet control, these appliances would be able to be a part of the home network. Imagine being able to start the morning coffee from the comfort of the bedroom. This would be possible with an appropriate coffee maker and WHCS outlet control. An added benefit from having outlets being automated is that there would be less draw from power leeching devices' power subsystems, which may be always-on.

2.3.4 Door Access

In addition to controlling home lights and various appliances, giving users remote control of their doors is a goal of WHCS. Through the use of an *electronic door strike*, we provide a specific control module the capability of locking and unlocking a door. This functionality is demonstrated in Demo 11.5. WHCS sees door access as important for a home automation system to support because remote access, like a garage door, is simple and easy. We want to make opening *any* door simple and easy.

Unlike controlling appliances and reading sensors, correctly managing the operation of a safety-critical door must be handled with great care. Any flaw in the implementation of the WHCS network would leave a user's home vulnerable to outside attack. Unfortunately due to time constraints, a security framework was not build. This is a common problem that also occurs in industry where companies do not put adequate enough resources towards securing their systems from the start. As a result, WHCS is an unauthenticated network, except for the BlueTooth link. WHCS is a prototype and should not be used in a real home.

2.3.5 Data Collection

In order to give users a broad overview of their home's state, WHCS supports the collection of data from *arbitrary* sensors. Data collected can include temperature, humidity, light level, sound levels, and so on. Each home may have sensors throughout collecting various data that the homeowner deems useful. The sensor integration with WHCS is transparent to the user. All they see is the list of sensors and the corresponding values. WHCS's pluggable control module's are tailored to each sensor or set of sensors, which would relay their data

back to the base station. This is illustrated in Figure 2. The base station supports queries from the LCD interface and simultaneously from a connected phone.

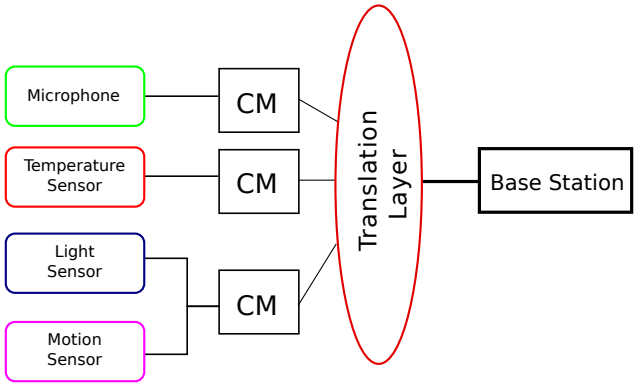


Figure 2: A illustration showing the translation of many different sensor nodes to WHCS' protocol

Beyond home sensors, all of the other controllable objects in the WHCS could have metadata being collected about them at all times. This extra metadata could include **connection status** and **power status**. See Section 6.1.4 for a more detailed description of the supported network packets and the type of fields they support. This extra metadata was not implemented due to time constraints.

All of this raw data being collected could be displayed to the user in the form of graphs and tables. It would also serve as the basis for a set of descriptive statistics for display to the user. No graphs or tables were implemented.

2.4 Requirements and Specifications

For WHCS to become the product that we wanted it to be we needed to adhere to a strict set of requirements and specifications. We identified certain requirements for subsystems involved in our system that we desired to fulfill in the development process. The following section is devoted to identifying requirements that we needed to meet to completely reach our goals for WHCS.

2.4.1 Mobile Application

The mobile application must communicate to other components of WHCS through BlueTooth. The process of connecting to the base station within the application must take no more than eight seconds from when the application is opened. Toggling the state of a control module within the system such as a light or outlet must also take no more than eight seconds from anywhere within the application. The user must be able to create sets of control modules to interact with simultaneously. The Android mobile application must follow Google's material design guidelines for Android development. The application must not request any privileges other than Bluetooth services. The application must support

full duplex communication with the base station. The application should be able to react to disconnections to the base station by polling the connection every thirty seconds and dropping the BlueTooth connection if no response is received.

2.4.2 Radio Module

The radio module must communicate in ISM allocated radio bands in order not to interfere with bands requiring radio licenses. The radio module used to communicate between subsystems must be able to communicate at ranges up to at least 50 meters. The radio chip must be able to act as a transmitter and a receiver and must be able to switch roles quickly. The radio module must be interoperable with common microcontrollers, so it should have a feasible method for interfacing with a microcontroller's pins. The radio chip must be capable of addressing, and further must be able to listen for messages from multiple addresses when in receiving mode.

2.4.3 Microcontroller

The microcontroller for the base station must have enough GPIO pins for an LCD, radio transceiver, and a BlueTooth module. The microcontroller must have an on-chip UART and SPI solution. All microcontrollers in the system must have on board flash memory to promote design simplicity. The microcontrollers must operate in logic level voltages ranging from 3.3V-5V. The microcontrollers must be capable of operating with a supply current of 50mA or less. The microcontrollers must have an option for setting the frequency to at least 8 MHz to boost throughput of the system. SMD footprints should be available for whichever microcontroller is chosen. The microcontrollers must be programmable while in circuit. The base station and control module microcontrollers must both have on-chip analog to digital conversion hardware. All microcontrollers must support interrupts to properly implement the network that is planned for WHCS.

2.4.4 BlueTooth Module

The BlueTooth must communicate with some form of RS232 to be interoperable with a microcontroller's UART. The BlueTooth chip must be able to exchange data with a baud rate of 9600, with higher baud rate options being a plus. The password and name of the BlueTooth module must be programmable for security measures. The BlueTooth module should operate at logic level voltages of 3.3V-5V. The BlueTooth module should be capable of communicating at ranges up to at least 50 meters.

2.4.5 Printed Circuit Board

The printed circuit board layout's outer dimensions should have a width of .008". The size of vias on the PCB should be set to .02". Isolation of ground pour polygons on the PCB layout should be set to .012". Logic level trace sizes should be .01", and any high level voltage traces should be set by referencing a trace width calculator. The preferred of resistor package for the PCB is 0805. SMD components should be used whenever possible

in the PCB design. The control module PCB should be no more than 4"x4". The base station PCB should be no more than 5"x6". High level voltages from the power supply should be isolated from the logic level voltages with an isolation area of at least .75". A two layer board design should be used for all of the PCBs in WHCS.

2.4.6 Power Specifications

The power supply developed for WHCS must rectify and downstep 120V AC provided by household mains. A transformer must be chosen that downsteps 120V AC to a level that is suitable for a switching buck regulator to regulate with high efficiency. A 5V and 3.3V line should be provided for each board in WHCS to meet the needs of individual logic chips. The 5V line must be obtained through a regulator with at least 80% efficiency. The step down to 3.3V from 5V does not need to be highly efficient. A relay must be chosen for switching high voltage to control high voltage components. The relay should have a load voltage capable of tolerating 120V AC. The input voltage required for activating the relay should be no more than 5V so it can be switched by a microcontroller. The relays forward current for activation must be no more than 40mA so that it can be supplied by the GPIO pin of a common microcontroller. At least 500mA should be suppliable to every board in WHCS. The electronic strike that is used in WHCS must consume no more than 700 mA while activated. The electronic strike must operate at 12V or less.

2.4.7 LCD

The LCD must have a touch screen interface to allow direct interaction with the system. Any interaction that can be performed on the system that is available from the mobile application must be performable from the LCD. The LCD must be operable with a supply voltage of 3.3V-5V. The LCD must be interoperable with a microcontroller with speeds of approximately 8 MHz. The LCD must be able to interface with a microcontroller through SPI or parallel data in. The LCD must be no more than 6"x6" and must be mountable directly to a PCB through the use of standoffs.

2.5 Research of Related Products

In order to have a successful product, the WHCS has done background research on products that have similar goals to WHCS. There is a lot to be learned from the "big players" in the industry. Through a survey of the below companies, we have a better idea of some of the features WHCS could offer and what the other companies aren't. This allows us to make new innovations and avoid mistakes that others have already made. Beyond that, we will know where WHCS falls in the spectrum of home automation by gathering as much data about other companies as possible.

2.5.1 Z-Wave

Z-Wave is a wireless technology that serves as a building foundation for home automation technologies. It is a certain type of home technology networking standard. The idea is

that any component for the home that carries the Z-Wave logo is interoperable with any other Z-Wave product. Certain vendors implement Z-Wave into their technology to give their products a wide appeal. Z-Wave is not a full home automation solution, it is merely the basis for wireless home automation products. Vendors like Schlage create components for the home that are compatible with Z-Wave networks. For example Schlage offers a door lock that can be controlled from a mobile phone due to its implementation of Z-Wave technology.

In relation to our product Z-Wave is similar to our implementation of a wireless network. Our Wireless Home Control System features radio transceivers that use a proprietary protocol to exchange information. Only components within our system are able to communicate within the network. The network also allows user's mobile phones to join in. So WHCS implements a tiny version of Z-Wave within the whole product. Z-Wave takes the networking to an extreme and offers it as a platform for vendors to create home automation equipment. For WHCS the network is a means to get our home automation equipment to communicate with each other instead of providing a foundation for others to build into.

Z-Wave has a great feature called "scenes." Scenes allow users to create groups of devices around the house and specify certain states for those devices that can be activated at the touch of a button. WHCS implements a feature similar to this. With the ability to control multiple devices around the home it is a common use case for the user to change the state of the system to certain common configurations. For example, a user might want to turn off all the lights in the house with a "sleep mode" every night. This is achievable through the hardware that WHCS creates for the house.

2.5.2 Belkin

Belkin has a brand of home automation products called WeMo. The WeMo brand consists of products that broadcast themselves via wifi and are controllable through mobile phone applications. There are multiple offerings such as heaters, outlet plug ins, light switches, and cameras. Overall their product line is quite similar to what we are aiming for. They have things that assist in home automation and are controllable by a mobile application. This is what want to accomplish with WHCS. They are different from the previously mentioned company Z-Wave because they actually provide smart appliances that go inside your home rather than just providing the foundation for a home automation system.

One notable difference between Belkin's line of WeMo products and our solution is how outlets are handled. WeMo offers outlet plugins that are put directly into the outlet and then expose an outlet facade for a user to insert into. Rather than completely turning off the outlet that is connected to the household main like WHCS does, WeMo shuts off the insert that has been put into the outlet as a middleman. This type of design is great for ease of installation but also clutters the outlet. When a WHCS outlet is installed it is not be noticeable from the outside and it will be a permanent addition to the household. Belkin also has a solution for toggling the state of lights with their WeMo light switches. These light switches are different from the outlets because they actually require the stock light switch in the home to be replaced with the WeMo product. Our version of light control leaves the stock light switch of the home intact.

2.5.3 Apple HomeKit

Apple HomeKit is an application programming interface that Apple develops for programmers to interact with any appliance that implements the Apple HomeKit Accessory Protocol. This is an approach to home automation that is not very popular. Apple seeks to make the mobile applications that access home automation systems not proprietary. With Apple HomeKit any Apple developer is capable of developing applications that interact with a multitude of appliances. HomeKit does not seek to develop one home automation solution, it wants to decouple the application development from the hardware. The company that we mentioned before called Belkin creates a mobile application to go along with the smart appliances that they sell. With Apple HomeKit the philosophy shifts away from development like this where the application comes along with the hardware.

With HomeKit multiple developers could develop independent applications all targeting the same hardware and users could choose which application they like best. Our WHCS solution for home automation features things that Apple HomeKit does except they will all be internal to us who are developing the system. There will need to be libraries that interact with the hardware but they will be proprietary and we will not expose them to developers. Together Z-Wave, Belkin, and Apple HomeKit complete the spectrum of home automation solutions. Z-Wave provides a network foundation for people to create appliances for, Belkin creates hardware and a mobile application to control it, and then Apple HomeKit creates tools to develop applications for controlling hardware.

2.5.4 Nest Labs

Unlike the previous companies, Nest Labs[19] is quite new, but has certainly claimed its space in the smart home market with its smart Nest Thermostat. Their other product, the Nest Protect, a smart smoke and CO_2 detector, integrates smoothly with the thermostat allowing for remote monitoring and control of the home. For their thermostat, the primary goal is to have a smart learning thermostat that aims to save energy in the home. By keeping temperatures at energy-saving levels when no one through the use of a motion sensor and learning algorithms, one of the most expensive home energy costs can be reduced.

In terms of administration, Nest has a online web interface and a mobile app that will display all of the networked devices, allowing for a highly connected experience. This would allow you to change your temperature from your warm bed or before you even get home.

Nest Labs was recently purchased by Google for $3.2 billion dollars[20], which certainly gives the company a powerful position in the market. One of Nest's goals is to have a *platform* for other companies and developers to create new products that *Work with Nest* ^{TM}[21]. This strategy is clever as now the success of the company will grow with every new developer who chooses to integrate their products with the Nest suite. Customers will see the multitude of devices that work with Nest and realize that they can "harness the future today." Quite a solid business model. Coming in at $249, the thermostat might be a tough sell for typical home owners who already have a working, yet "dumb", thermostat.

2.5.5 X10

Founded in 1970s, X10 is one of the oldest home automation companies still on the market. Home security and automation are the key goals of X10. To meet these goals, X10 offers a wide selection of products to link every aspect of the home or business. When these products arrived in the early days, they set the standard for home automation. The architecture is centered around wireless remotes, transcievers, and appliance modules. The transciever is a fixed plug-in that is listening for commands from the wireless remote. If it receives a command, it can send signals through the house power grid to an appliance module.

The applications for X10 products are limitless. One issue with X10 and its goals are that the entire system is proprietary and encourages lock-in. The frameworks for making new X10 devices are not open, which could limit any support. WHCS aims to do the opposite by providing a well documented home automation solution. This became an issue with the original company of X10, X10 WTI, went bankrupt in 2013. This caused a disruption in the X10 ecosystem, including dedicated services run by X10 WTI. Luckily another company, Authinx, picked up the domain[1] and is continuing to offer support and sales for X10 products. Give this recent change of hands, choosing X10 today may not be a sure choice for the future.

3 Realistic Design Constraints

3.1 Economic Constraints

Economic constraints were the biggest hindrance in the development of WHCS. The amount of money necessary to complete such a project added up quickly due to the necessity of obtaining hardware like printed circuit boards. This project was developed by college students so the amount of reserve money that was available is low. The average full-time student schedule can obstruct students from having time to earn extra money. To help alleviate the lack of funds our group applied for funding through Boeing. Boeing was kind enough to approve our application for WHCS. Unfortunately we did not get all the money that we asked for which means part of the project had to be paid for out of pocket. This would have been an issue had things went wrong during the development process that would have caused us to have to repurchase certain pieces of hardware. Fortunately this did not happen to us due to careful planning. In all of our design time decisions cost played the biggest factor. Our research and development budget was small compared to mass-produced products similar to WHCS. We recognized this constraint from the offset and planned accordingly.

3.2 Time Limitations

The amount of time that we had to create WHCS was a limiting factor for what we were trying to create. To begin with, the technologies used in this project were not something that we were already well-versed in before starting. There was a ramp-up period for figuring out what technologies would be necessary to implement a system with the capabilities we desired. This is not even including the research time necessary for finding exact chips that fit each of the requirements we needed. During the development lifecycle of the project we

were not just creating WHCS we were learning things necessary to design in general such as creating PCB layouts. Learning things like this, necessary to actually create parts of the project take a measurable chunk of time to do. To add on top of the ramp-up time that we had to endure, we missed valuable weeks due to the summer semester. Our second semester of senior design took take place in the summer semester which was 12 weeks instead of the normal 16 weeks of a semester. This meant that we lost out on almost an entire month due to the second half of development being in the summer. This whole month of lost time put a large strain on the group and it is something that we had accounted for early on.

3.3 Political Constraints

There were no relevant political constraints that we attributed to the development of WHCS. The development of WHCS was not aligned with the ideals of any political party. Our research results showed that there is not noticeable political involvement in products similar to ours.

3.4 Ethical, Environmental, and Sustainability Constraints

The discussion of ethical constraints for WHCS is directly related to environmental awareness and sustainability. The most ethically bearing decisions we had had to make during the development of our system involved power usage. There is no doubt that power usage has a vast effect on the environment. While developing WHCS our goal when faced with multiple options for implementation was to take the path which resulted in the least power consumed. For example, the type of electronic strike we chose was based on which one wasted less power during operation. We believed that making decisions in this area to help the environment and promote sustainability gave us the best ethical outlook. This is also the only ethical involvement we attributed to our project.

3.5 Manufacturability Constraints

WHCS has the potential to be implemented in many homes because consideration was put into manufacturability during design time. As we made decisions for our system we were constantly monitoring how it affected the overall ease of replication. We identified the base station PCB and control module PCBs to be the biggest contributors of our overall manufacturability. In order to maximize manufacturability we constrained ourselves to only using highly available parts for incorporation into our PCB designs. Following through with this commitment ensured that WHCS installations could be standardized and easy to replicate.

3.6 Safety and Security

The safety and security of WHCS is primary constraint of the project. Due to the integration with home, especially access control systems, WHCS must not negatively affect home security. Additionally, as WHCS is in control of large currents involving light control and outlet control, great care must be taken to design circuits and software to prevent fires

and misbehavior. If for instance, we decided to control a light that exceeded the rating of one of the control relays, then this could be a fire hazard. Also, in terms of door control, if the mechanism for controlling the door were to fall in to the hands of a burglar or fail completely this would present a critical safety and security issue.

For example, there is a trade-off that needs to be made for controlling a door with an electronic strike. Electronic strikes come in two main flavors: normally opened (NO) and normally closed (NC). NO favors security by *failing-secure*, meaning the lock will not be openable if in the event of a power loss. NC on the other hand will *fail-safe*, meaning the lock will be openable without power applied. This consideration should be based around local fire code and depending on the type of door and handle used. We explain our decision to go with a NO type electronic strike in Section 6.8.2.

In general, some principals for making sure that safety and security problems are taken in to consideration are

1. Analyze potential problem areas
2. Develop solutions for problem areas
3. Anticipate failures and handle them accordingly

Through premptive *analysis*, we may determine problem areas. We will *develop* solutions for these problems, consisting of a description of the problem, its potential impact, what area does it impact, and what we plan to do to address it. Finally, we will *anticipate* failures and build in the developed solutions directly in to our design. These solutions may be in the form of warning labels (Ex. DO NOT EXCEED 12V 10A) and software checks.

Due to project time constraints and priorities, the WHCS network is *not* considered secure from remote attack. No cryptography has been used to secure the networks. The only access control mechanism lies with the BlueTooth chip which requires a weak 4 digit pin.

3.7 Spectrum Considerations

For WHCS we provided over the air communication between the android device to the base station and between the base station and the control modules. Out of the frequency bands that the FCC has marked as unlicensed we decided to use the band that includes 2.4 GHz. Although there are other unlicensed bands that could have been used such as the 900MHz band and the 5GHz band, 2.4GHz provided the best solution. The higher the frequency the shorter the range yet the better the data rate and the smaller the device. The main reason why 2.4GHz was chosen is because it is a happy median of good range and acceptable data rates. Unfortunately because 2.4GHz is such a good frequency to operate at it also has a lot of interference from other devices that operate at this same frequency. However interference will happen from whatever frequency band that is chosen, and this issue wasn't enough of a problem for our design to deter us from using the 2.4GHz band. Both the NRF chip and the Bluetooth chip operate within this band.

4 System Design

This section contains an overview of the components necessary to create the Wireless Home Control system. Without going too deep into implementation details, an abstract view of WHCS is provided. Block diagrams are utilized to show the interaction between WHCS subsystems. The information in this section covers the central components necessary to realize the system and fulfill our objectives.

4.1 Base Station

The base station of WHCS is the central component. It is the only component that is involved in everything that is done with WHCS. Any command going to an endpoint passes through the base station and any information coming from an endpoint goes through the base station. The base station acts as the smart middleman between the user and the control modules that they are targeting. The base station is a small portable device that can plug into a home outlet for power. The user needs to be able to interface with the base station in order to control the system. To facilitate this interaction there is a mobile WHCS application that can be used to communicate to the base station, and there is also a touch enabled LCD. Both of these methods are available to control the base station. The base station is designed to be the shape of a rectangular prism and has an LCD flush with the surface of the top of the prism. This way the PCB is hidden away and does not have to be known to the user. This provides an aesthetically pleasing look to the base station and hides away its inner workings. Figure 3 shows what the base station would look like including the LCD, the inner PCB, and the plug for power in a home installation. The size of the base station is based off of the size required for the printed circuit board.

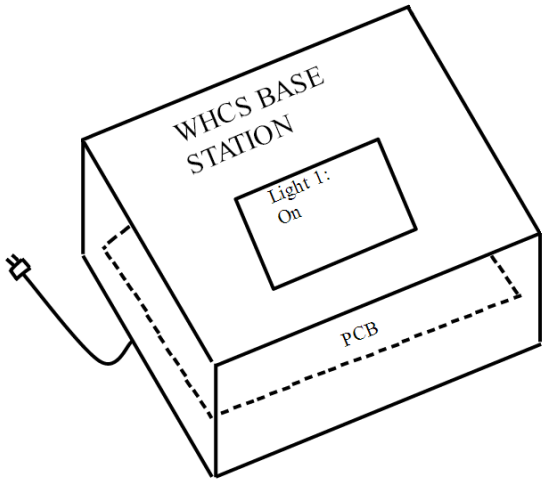


Figure 3: Base Station Exterior Design for WHCS

The base station requires a special printed circuit board that is different from the control modules. It is the only board in the system that has to support BlueTooth communication. This is how the base station communicates with the Android phone. It also needs to have space for the installation of the LCD screen. The LCD screen is able to be mounted on

the base station PCB so it can be pushed through the display box. Using this architecture the LCD is all the user has to see. Like the control modules the base station has a radio transceiver chip. This is the part of the base station that allows it to communicate to the control modules who have a similar radio transceiver. The LCD, BlueTooth chip, and radio transceiver connect to the microcontroller that is chosen for the base station. This whole system is powered by a 120V AC step down circuit. The base station also has room for a programmer pinout that way any errors in design can be fixed in circuit. The block diagram of the components for the base station is depicted in Figure 4.

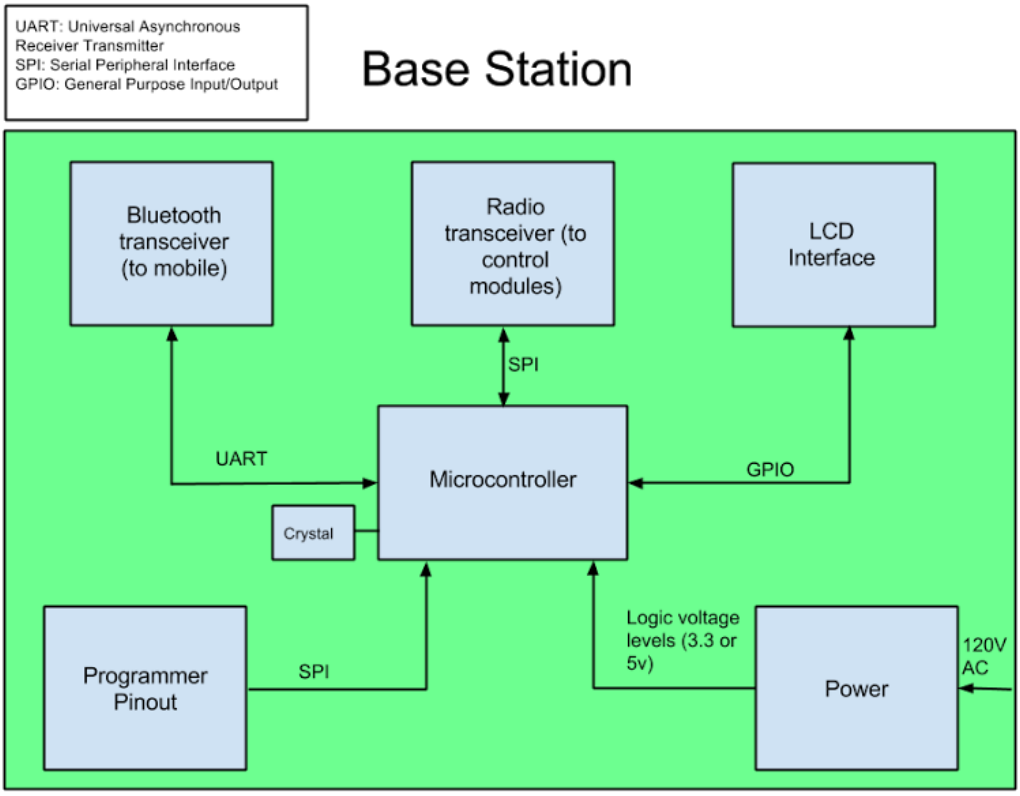


Figure 4: Base Station PCB Block Diagram

As shown in the block diagram the programmer pinout and the radio transceiver will both be connected using the SPI bus of the microcontroller. SPI allows for selecting which chip to utilize at any given time so therefore the bus sharing conflict will be able to be resolved.

4.2 Control Module

The control modules of WHCS are the boards that connect to the target endpoints around the home. Control modules can be of different types due to the fact there are multiple targets in the system. There are light/outlet modules, door modules, and sensor modules. All of these different types of control modules have similarities that can be taken advantage of in order to create a single control module design that can be adapted to the specific endpoint

it is targeting. Every control module has a power supply circuit, a microcontroller, and a radio transceiver at a minimum. The rest of the circuitry revolves around what the control module is meant to interact with. For example the circuit to activate the electronic strike on a door needs to switch the electronic strike power, while a temperature sensor circuit needs to do analog to digital conversion. There are two options that these possibilities brought about for the creation of control module boards. One option was to create a control module design that has room to solder the components necessary to interface with all of the supported endpoints of WHCS on one board. This meant the control module would have the foundation for adding the electronic strike circuit, the light/outlet circuit, and the temperature circuit all at once. All that would need to be done is to connect the circuit to the actual target of choice, and if desired the control module could be removed and hooked up to a different type of target. The second option was to create a printed circuit board design that has a section dedicated to the implementation of a single control module endpoint circuit. This way when the printed circuit board is assembled we could custom solder components based on what that control module was being used for. Either of these options would have worked. The option where all control module boards are able to support all three control module modes is the more involved option so that is the option that we considered for our design. Figure 5 shows the block diagram for a control module. The blocks labeled radio transceiver, power, and microcontroller are the only ones necessary for every single control module. The three blocks on the lower left are the circuits that interact with specific endpoints around the home. These are the parts of the control module that can either all be included in every control module, or can appear as lone circuits on every control module.

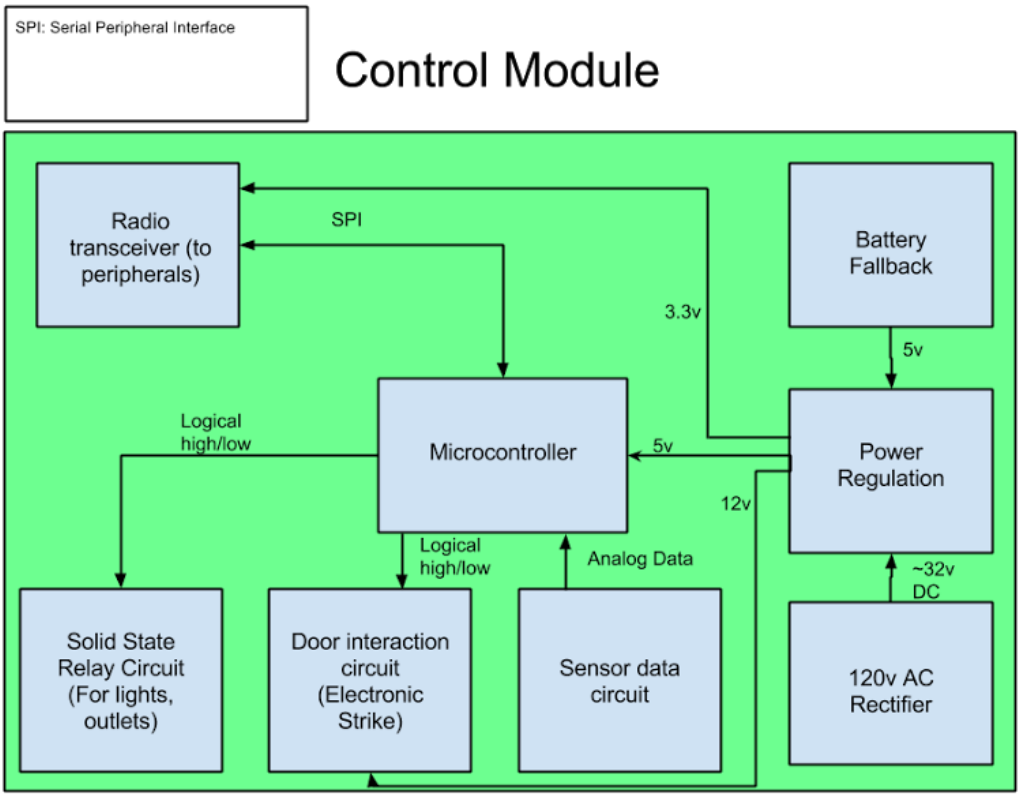


Figure 5: Control Module Block Diagram

4.3 BlueTooth Capable Phone

The Wireless Home Control System was based around the capability of interacting with the system from a mobile phone. A phone with BlueTooth communication capabilities was required for the system to operate at its full capabilities. With the selected BlueTooth phone platform, a software application was developed to allow users to interact with WHCS. It is through this application that users can monitor the state of their home and remotely interact with targeted appliances. The phone application features a low-complexity graphical user interface as well as the ability to use speech-recognition activation.

4.4 Software

WHCS contains a network consisting of its base station, its peripheral control modules, and the user's BlueTooth capable mobile phone. This created the need for three distinct software models. These three different software implementations follow a custom communication protocol in order to transmit data throughout the network. Commands frequently flow from the user's mobile phone all the way to a control module that they want to interact with. The data flow can also occur in the reverse direction. It is the goal of WHCS software to facilitate this interaction. Phone to base station communication is done through BlueTooth.

The phone is able to utilize BlueTooth libraries, while the base station microcontroller will interface with a UART to control a BlueTooth module. Communication between the base station and control modules relies on a ported driver written for the radio transceiver used in the system. The base station is the powerhouse in terms of software design. It is at this central hub that the collection of all active control modules is stored. The base station also constantly has to be ready to accept requests from any control module or the user. The main routine needs to have handlers ready for the LCD or the mobile phone because either choice will be usable to interact with the system. Figure 6 shows the block diagram for the entire system's software. This is meant to provide insight into the operation of the large components of the system

Apart from the networking aspect of WHCS the software also needed to control the interaction with the targets of the individual control modules. This is depicted in Figure 6 as the bottom-most block. This block had to be custom tailored based on the role of the control module. The routines running on the microcontrollers need to know the state of the appliance they are interacting with, and how to do any necessary activation that is requested by the user. For sensor type control modules the microcontroller needs to update at intervals in order to maintain correct information. For the modules that are activating outlets/lights or controlling the electronic strike the routine is as simple as toggling the state of a solid state relay connecting to a GPIO pin. It is the communication between subsystems that presented the biggest challenge for the microcontroller based boards.

The mobile application software has three main components, the user interface, bluetooth communication handler, and speech recognition. These blocks are shown at the top of Figure 6 and together they complete the mobile application. The user interface is the most involved piece of software. The interface has to constantly display the state of the system and provide intuitive use cases. All of the capabilities that we created for WHCS had to be reflected by the user-interface that we designed. The U.I. needed to be linked together with the BlueTooth communication handler because the user expects state changes in the system at the press of a button.

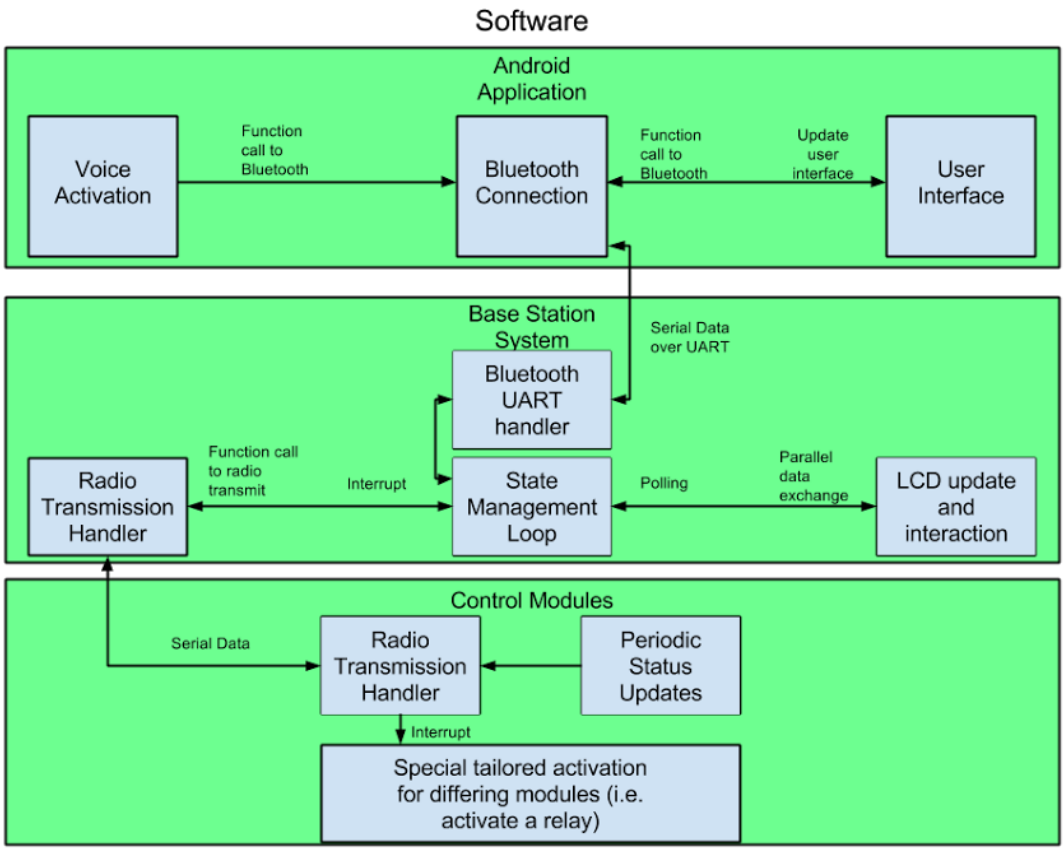


Figure 6: WHCS Software Block Diagram

5 Summary of Related Standards

In order to speed up development and promote compatibility the WHCS design adheres to multiple standards. Using standards allows us to rely on certain characteristics for the devices we use. Standards allow us to stand upon the design decisions made by organizations before us.

5.1 RS-232

RS-232 (Recommended Standard) is a serial communication standard that specifies hardware and software implementation for serial communication between two connected devices. Our microcontrollers will be using a UART for debugging and communication through BlueTooth. The UART module present in our microcontrollers is based off of RS-232. The microcontrollers use logic voltage levels which are are not the same as those specified in the standard. This version of RS-232 is often referred to as TTL-serial (Transistor to Transistor Logic). The UART also only uses the Rx and Tx lines specified in the standard. The software and algorithm used to communicate with the UART is the same as RS-232, so the deviation from the standard is hardware only.

5.2 BlueTooth

BlueTooth is the standard that we will rely upon for communication between the base station of WHCS and the mobile application. BlueTooth was originally standardized by IEEE as standard 802.15.1. The standard is now maintained by the BlueTooth special interest group. Devices that implement BlueTooth are able to connect to one another wirelessly to exchange data serially. The wireless signal's frequency is regulated to be between 2400 and 2483.5 MHz. The base station will contain a BlueTooth module which implements the BlueTooth standard. This module will allow communication with the mobile phone, who's hardware will support the BlueTooth standard. BlueTooth's serial communication characteristics make it well-suited for communicating with a microcontroller's UART. The UART to BlueTooth module connection is what we will be performing.

5.3 SPI

SPI (Serial Peripheral Interface) communication is a de facto standard so there is no regulatory body that develops or maintains the specification. SPI is used in microchips for communicating to multiple connected devices through one bus. SPI features a slave select line that allows the microchip (the master) to pick a target device (slave) to speak to. WHCS will use SPI for interfacing with the radio transceiver module from the microcontroller on the base station and control modules. SPI will also be used to program the microcontrollers we use. The slave select feature will allow us to have our radio transceiver still attached to the SPI bus while we are programming our microcontrollers.

5.4 FR-4

FR-4 (Flame Resistant) is a standard for flame resistant glass-reinforced epoxy laminate sheets. These provide the basis for printed circuit boards. The FR-4 standard was developed for complying with the regulations for flammable plastics set in standard UL94V-0. The core of PCBs are built around a laminate sheet that meets the FR-4 standard. We will be populating printed circuit boards for WHCS so we will rely on this standard for the proper operation and safety of our PCBs.

5.5 Android Development Guidelines

The Android developer's guide has a list of guidelines that are strongly recommended for Android applications. These are standardized guidelines, but they are not standards because they are not required. These guidelines are relevant to our project because we will be developing an Android application and we will be adhering to the specifications. The guidelines all have individual ID codes. Some noteable guidelines that we will be following during the development of the WHCS application are UX-B1, UX-N1, and FN-S1. Respectively these standards involve not redefining the functionality of on system icons, supporting back button operation in applications, and not leaving services such as BlueTooth open while an application is in the background.

5.6 ANSI/NEMA 1-15P, 5-15P, C84

WHCS will get its source power from household mains. The connectors for household mains, and the voltage levels provided are standardized by ANSI/NEMA. The plugs that will allow the base station to be connected to a United States structure's outlets are standardized by 1-15p (2 prong) and 5-15P (3 prong). We can count on the input voltage of WHCS being 120v AC because of standard C84 which standardizes the power supplied to household mains in the United States.

6 Hardware and Software Design

6.1 Radio Transceiver

For the radio transceiver of WHCS the chip that we decided to use was Nordic Semiconductor's NRF24L01+. This chip met all the requirements that we set for our radio transceiver. The NRF is also a very popular chip that is easy to find and rarely out of stock. This was a benefit to the manufacturability of WHCS because NRFs are cheap and easy to buy in bulk. Alternatively we could have chosen to use an XBee radio device which implements the zigbee 802.15.4 IEEE standard, however we did not see the need for this. XBee devices are also more expensive than the NRF chips that we have decided to utilize.

6.1.1 Operating Principles and Usability of NRF24L01+

The NRF24L01+ is a radio transceiver that operates in the ISM (Industrial, Scientific, and Medical) radio band. The range of channels for the NRF is 2.4GHz to 2.527 GHz, however because the designated ISM band that we used only ranges from 2.4GHz to 2.5GHz we were not be able to use all of the NRF's available channels. With the NRF we were capable of sending payloads with a maximum size of 32 bytes per transmission from module to module. We were able to change the data length from 1 to 32 bytes in order to find the optimal mix between reliability and speed. Every NRF chip has the ability to simultaneously store 1 transmission address and 6 receiving addresses. The first receiving address is utilized if the auto-acknowledgement feature is enabled, so effectively there are 5 receiving addresses. This capability gave us flexibility for implementing our network because we could make decisions such as having a dedicated address to each node in the network as well as an address for broadcasts of certain types. The addresses of the NRF are 5 bytes wide so we were be able to have many NRF modules within the network. A very useful feature of the NRF is the ability to enable auto-acknowledgement. When this feature is activated the receipt of a transmission from one NRF to the other is auto-acked without the need from any upper level software. This simplified the work necessary for creating our own network of NRF chips. We were able to confirm the receipt of data therefore increasing reliability. This auto-ack also allows the NRF to perform retries up to a given limit, so just in case there is noise during the transmission the NRF will repeatedly try to transmit again. The NRF also allows for low power mode and long range mode. For WHCS we are able to tweak whether or not to use long range mode or not depending on the performance of the system within its environment.

The NRF requires 3.3v of electricity to operate so all parts of WHCS require a 3.3v line. The datasheet lists the current consumption while in receive mode as 18mA. This is the most common mode for the NRF chips present in WHCS so they can be ready to receive commands at any time. The chip receives commands from a microcontroller through SPI (Serial Peripheral Interface). This is great because the NRF design philosophy fit perfectly with our microcontroller based base station and control modules. Beside the standard MOSI, MISO, and SCK for SPI, the NRF also has a csn pin for telling it to receive commands, ce pin for telling to transmit or receive at that moment, and an interrupt pin for notifying the microcontroller of important situations. The csn pin allows the SPI bus to be shared with other components such as the LCD being used for the base station. The interrupt wire pin can be monitored in order to listen for data received, data sent, and data failed to send notifications. In total the NRF takes up 6 pins whilst three of the pins are shareable with other SPI components.

6.1.2 Driver Use Case

The NRF chip that we decided to use for communication in WHCS needed to have a driver written for it. This was to help keep the way we interface with the NRF consistent and to provide clean code. All of the network code that we write for the base station and the control modules was relying on the integrity of the NRF driver that we wrote. The driver provides the foundation and if it is not reliable then none of the code we wrote for our system would have been reliable. The focus for the development of the NRF driver was elegance. We wanted everything the NRF driver offered to be simple yet accomplish everything necessary. We developed the use case diagram pictured in Figure 7 as a guideline for the development of the NRF driver. The NRF driver provides the functionality included in the use case diagram in an easy to use format. These are the most common uses of the NRF.

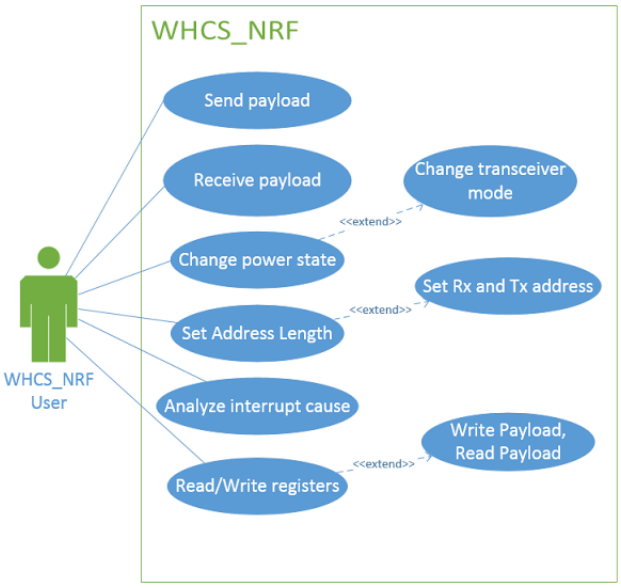


Figure 7: NRF Driver Use-case Diagram

The core use of the NRF driver is transmitting and receiving payloads. Every other use case is a supporting role for the final goal of transmission. The basis of the driver is reading and writing registers. Everything builds off of this capability, especially reading and writing the payload for transmission. Other use cases such as changing power mode, checking the status of the chip, and changing from a transmitter to a receiver are special forms of writing to a register. Thus reading and writing to registers is a use of the driver, however it is abstracted in a way that provides ease of use. A user of the NRF driver spends most of the time setting addresses, writing payloads, transmitting payloads, and analyzing interrupts. These use cases needed to be implemented perfectly to provide a strong foundation for the networking of WHCS.

Originally it was our plan to develop a NRF driver from the ground up to fully utilize the communication capabilities provided by the chip. We made good progress on this endeavor and were already able to access a large amount of the NRF's capabilities. At some point during the development process we found that there was a library called RF24 that was designed for use with NRF chips on Arduino boards. Since the microcontroller family that we used in WHCS was the same family that Arduino uses we thought that we would be able to port this library and compile it with AVR-GCC. We had to stub out and replace Arduino specific functions like digitalwrite() and timer calls. We also had to write an underlying SPI library for the RF24 register read and write functions to call upon. We were able to get the RF24 library fully ported and working with our base station and microcontroller. This saved us a large amount of time during the development process.

6.1.3 Driver Class Diagram

It was decided that the best design approach for implementing the NRF driver was as a class in C++. We used Atmel ATMega microcontrollers in WHCS so C++ was supported as a development language. Using C++ allowed us to create a class that can leverage object oriented programming techniques such as encapsulation. The class diagram for the driver is shown in Figure 8. Primitive functions such as ReadByte and WriteByte can be hidden from a user while PowerUp will be exposed as a public function. Using C++ also gave us the ability to use a constructor when using the WHCSNrf class, and in this constructor we can assign the only thing varying between uses of the NRF, the chip enable pin and the chip select not pin. Assigning the ce pin and the csn pin are the first step of using the NRF driver. Any communication between the microcontroller and the NRF relies on the proper assignment of these pins.

Figure 8: NRF Class Diagram

Usage of the NRF driver involves first constructing the class by telling the microcontroller which pins the NRF is connected to. Then the user does everything necessary to customize the way that data is transmitted and received. The driver exposes the common settings in an easy to access manner. Enabling things such as auto-acknowledgement and the number of retries for the transceiver is done with the call of a function with simple parameters. Setting the address for receiving and transmitting data is done in one line of code. The SetTxAddr function will be one of the most utilized function for an NRF involved in a network constantly sending payloads to other chips. A typical use will involve powering up the NRF with the PowerUp call, setting the transmission address, writing a payload, and then transmitting a payload. With this driver, the user does not need to know the registers involved with the NRF. The hardware interactions with the chip are all abstracted away.

In the final version of WHCS the driver that we used for the NRF was actually very close to this initial class design. The RF24 library used a C++ class and fully encapsulated the logic needed to operate the NRF. Our WHCSNrf class would have been a great solution to using the NRF but there was no need to continue implementing it once we found we were able to port the RF24 library.

6.1.4 Network Library

In order for WHCS to be networked, a set of networking functions have been constructed on top of the radio driver to form a network library or **NetLib**. Abstraction of the raw driver allowed for the concept of a **node** to be created. The network consists of a set of nodes capable of direct communication and reception of **logical packets**. These packets contain metadata describing their purpose and the data that they contain. For example, the base station to control module communication includes a "from" field to let the receiver know where the packet is from. By dividing up the required network features of WHCS in to

packets, we ended up with a library of datagrams. Individula datagrams no longer have to worry about low-level details such as when to transmit – they merely have to worry about who they are sending to and receiving from. This can be seen in Figure 9.

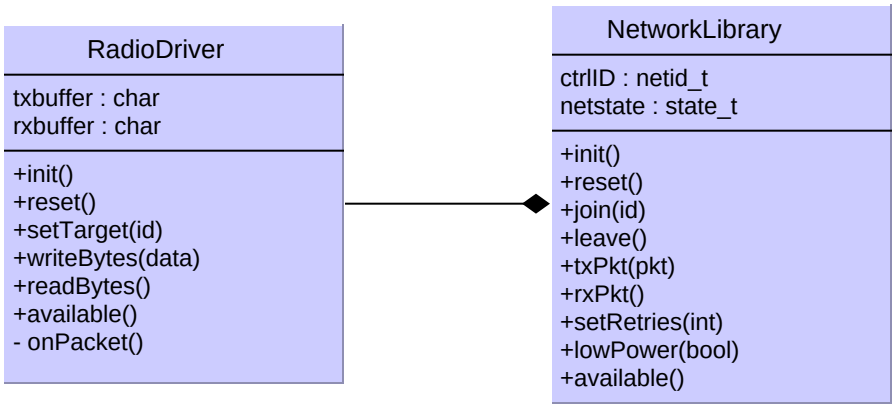


Figure 9: Network Library Class Diagram

Similar to network protocols today, WHCS implements a lightweight protocol stack similar to the OSI model. The physical layer is handled by the NRF chip itself, the logical link layer by the NRF driver, and all other layers by the NetLib. In WHCS's case, our version of an "Internet Protocol" is to assign each node with unique ID that can be targeted by other nodes for sending and receiving data. In terms of the transport layer, our application is similar to User Datagram Protocol, which is "best effort" in terms of reliability. We planned on adding additional code and sequencing to allow for this unreliable medium to withstand lost packets, but ran out of program space before being able to do so. The NRF already provides the ability to retransmit on a non-acked frame, but this just fails after a certain amount of attempts. This can handle transient errors, but not a complete loss of connectivity, which may occur if the base station lost power.

In order to keep the network topology simple, we opted to have a simple module to base network. This means modules talk directly to the base station and to no other modules. This is a simplification based on the underlying radio's range and our project requirements. We did not need a complicated mesh network as all of the nodes are within range of the base station during their lifetime. For larger or noisier households this architecture would break down, but for the purposes of an initial design, this configuration does the job. Additionally, the network topology is not fixed in hardware - it is merely additional code that needs to be written and implemented in to the overall design. More research wasn't be done along the direction of a full mesh network, as it wasn't necessary for good performance. The direct benefit of this simple architecture is that is easy to test and easy to program. The logic only expects transactions to occur between two nodes. That fact drives the construction of a specific and optimized network communication protocol that doesn't waste unnecessary bytes during transmission. The lower the byte count for each transmission, the more likely the packet will reach the destination in tact.

Modes of operation Before diving in to the specific functions required for implementing a robust NetLib, we discuss the high-level modality of the library. The state of a network link can be broken in to the following modes:

- Joining
- Communicating
- Idling
- Leaving

The network library is in the **joining** mode when it is first powered on or in the event of a disassociation. A node shall attempt to join on to a local network of nodes through a master node that is defined at the initial hardware configuration of the network. The joining process has its own state machine describing the required processes for association on the WHCS network. The **communicating** mode is active if the node is joined to a network and if there are data to be sent or received. Otherwise the node is in the **idling** state, which includes a reduction in power consumption. If for some reason the network needs to be reconfigured dynamically, the base station or a control module can begin the **leaving** mode. This frees any resources present for maintaining state on the active network join and prevent the leaving node from communicating on the network. A graphical overview of the NetLib modes and the transitions between modes is shown in Figure 10.

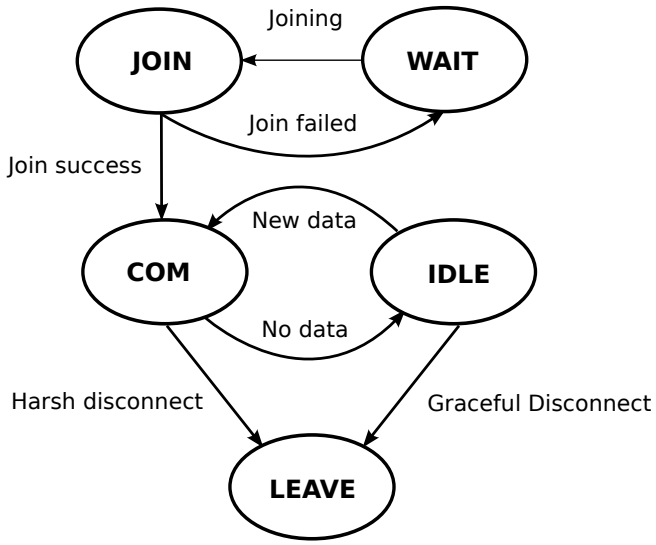


Figure 10: the overall state machine for the network library

Join mode detail The join mode is fired when a new node wants to associate to an existing WHCS network as maintained by a base station. The base station is the arbiter of all communication for the WHCS network and as such, handles initial join requests. It is the node's responsibility to know and maintain any necessary access tokens for a join request along with the required channel of the network. Having the required channel is required for the sender and receiver to communicate. The base station validates the joining node and grants or denies access to the WHCS network. This process is necessary to provide some assurance and tracking for the active nodes on the network.

Communicate mode detail Once a node has been successfully joined to the WHCS network, it is able to communicate directly with the base station as its network gateway. All communication flows through this node, which processes the data, and if required, generates a response. This response may be directed back at the transmitting node, such as an acknowledgement of the packet. Packets of control modules have the ability to change state in the base station through the use of "update packets." These packets provide some periodic or event information about the transmitting node. This includes arbitrary data and is parsed based off of the node type. For example, a temperature node has a specific data format that the base station knows how to unmarshal and store.

The overall flow of the communicate mode is to pump out packets to be sent, wait for any responses, and generate any actions as required by the packets. This loop is performed by any node with a radio. The base station is the only special case as it has to receive from multiple nodes simultaneously. Therefore some packet dispatching is required in order to make sure the appropriate packed reaches the target data structures.

Idle mode detail The idle mode is the simplest mode for the NetLib. This state is where the library has no data to send or receive. During this mode, the underlying radio is free to sleep in order to conserve power usage. Also interrupts are used to avoid polling the radio for new data, which would eat a lot of power. When a new packet is received or one needs to be sent, the radio wakes up and enters in to the communication mode.

Leaving mode detail The final mode of the network library is the leaving state. This is the state where the node decided or is ordered to leave the network. It disassociates from the WHCS network thereby causing its state and all of the resources associated with the join to be freed. Additional cleanup may be performed in order to get the library ready for a new join request. Any outstanding packet sends or received are dropped and any potential data is lost. A safer shutdown would assert that the radio must be in the idle mode. That way no data would be lost to the leave transition.

6.2 Microcontrollers

For the proper operation of WHCS, microcontrollers needed to be installed into all of the control modules as well as the base station. This meant research was needed to choose what the best microcontroller for each of the modules was. The base station is a bit more hefty than the control modules so the design considerations were different for the two. The first step was to figure out what family of microcontrollers to use.

6.2.1 Microcontroller Brand

Choosing the microcontroller brand would set the path for all the development that we do with the microcontroller. Every other choice would stem from this decision so we wanted to make a smart choice. We weighed out the documentation, support, ease of acquisition, ease of use, and community for the brands that we considered. Our initial choice was Texas Instruments because of the use of the MSP430 launchpad board in the UCF curriculum. The fact that using the MSP430 was required in EEL 4742 (Embedded Systems) meant

that everyone was already at least somewhat familiar with it. The familiarity factor was a plus for the MSP, and we all knew that T.I. has very good, albeit lengthy, documentation for their chips. A quick look at digikey showed that MSP430 microcontroller chips were well in stock so there would be no problem acquiring them if they were the chip that we wanted to use. The thing that we were unsure about with using Texas Instruments chips was the community surrounding the brand. For example if we encountered a problem rewriting a fuse on the microcontroller would we be able to find a forum of people who knew how to solve the problem, and things of that nature.

While we researched Texas Instruments based microcontrollers we also researched the Atmel brand. We knew Atmel is very popular especially because they produce the chip used on the Arduino development boards. We checked how the Atmel chips were documented by looking at the datasheet for the Atmega 88, and were pleased with how well things were laid out. Digikey had most of the common brands of Atmel chips in stock so acquisition would be no problem. We didn't feel the need to consider any other brands because we felt that between the two choices we looked into they could both suit our needs. Atmel offered chips and brand support similar to that of Texas Instruments, so for whatever kind of chip we required we could use Atmel or T.I. For us the choice came down to how prominent the communities for the two brands are. Ultimately we decided that the Atmel brand had a very pervasive community that was sure to aid us in our utilization of any chips produced by the company. While the MSP430 was familiar to all of us the Texas Instruments chips did not seem as broadly used across the open source populus. So our final decision was to use the Atmel brand of microcontrollers for WHCS.

6.2.2 Base Station Microcontroller

The base station is the center of operations for WHCS so it needs the most computing power. It has the most data structures to take care of, the most commands to process, and the longest compiled program. The microcontroller for the base station also needs to be able to connect to the LCD screen, the radio transceiver, and BlueTooth transceiver all at the same time. So taking all these things into consideration for choosing the microcontroller was important. We needed a large amount of pins to interface with all the peripherals, especially because we knew we wanted a parallel interface LCD. A large amount of flash memory was also a trait that we looked for since we knew the base station would be doing a lot, thus making its program long. The first big decision was whether to use an ARM based microcontroller for our base station since it was the brains of the system. ARM microcontrollers have much higher processing power, but also introduce complexity. Many ARM microcontrollers don't have on board flash memory so that is an added layer of design that is needed to get the unit working. After considering the processing necessary in the base station and the bandwidth of the network in WHCS we realized that an ARM based microcontroller would not be necessary. There was not enough data to be processed to warrant the use of an ARM microcontroller. Using an ARM unit would introduce design complexity for a solution that could be attained through easier means.

Eventually our research efforts landed us upon the Atmega series of Atmel microcontrollers. These microcontrollers are often thought of as the flagship Atmel chips in the DIY community and they did seem highly capable. The chips were cheap, had plenty of pins, plenty of on-chip peripheral support such as the UART, were easy to program, and were highly

available. After browsing the Atmega series of chips we narrowed down the chip that we would be using for the base station to the three chips listed in Table 1. The table shows that the lowest maximum operating frequency between the three is 16 MHz which is more than capable of powering the low bandwidth network of WHCS. 16 MHz is enough to handle communication, the LCD display and interrupt vectors, so all the chips are viable options in that feature. The 8 KB of flash memory was low compared to the 32 KB offered by the other two so we actually took that chip out of the running when we considered that the base station would have a large routine. When we were down to choosing between the Atmega328 and the Atmega32A we realized that the Atmega32A was actually the only option between the two chips that would work. The reason for this was the number of I/O pins. We had picked the LCD, radio transceiver, and BlueTooth chip that we would be using and the Atmega32A was the only chip with enough I/O pins to support all the peripherals. Our final decision was to use the Atmega32A as the microcontroller for the base station.

Atmega MCU	Flash Memory (KB)	Max I/O Pins	Max freq. (MHz)
Atmega88	8	23	20
Atmega328	32	23	20
Atmega32A	32	32	16

Table 1: comparison of ATMega chips

After we had decided to use the Atmega32A as the microcontroller for the base station there were still some things that needed researching to make it usable. The main issue was disabling the JTAG interface on port C to make the pins usable as general I/O. After researching the data sheet and the community forums we found that to be able to use port C pins as GPIO the JTAG fuse must be disabled. Otherwise setting these pins on port C to high or low will have no effect.

6.2.3 Control Module' Microcontrollers

The control modules are the parts of WHCS that do the interacting with the endpoint appliances. The microcontroller that we selected for the control modules would have to have the pin count and processing capabilities necessary to be able to interface with any of the appliances. This means that the microcontroller would need to have gpio pins for activating relays for the outlets and lights, and it would also need to be able to do analog to digital conversion for sensors. The control modules also receive commands from the base station and send status replies back, so the control module microcontroller would have to have an SPI interface for utilizing the radio transceiver that we chose for WHCS.

The list of constraints on the selection of the control modules' microcontroller was not long. The control modules were not going to be as demanding as the base station. We could choose a minimalistic chip for the control modules, but we knew it would help with the design and development process if we used a microcontroller similar to the one in the base station for the control modules. This led us back to the ATmega series. Referring back to Table 1 we see the three main ATmega series chips that we researched. All the chips had the SPI interface necessary to communicate with the radio transceiver. They all had enough pins to interface with the most demanding control module, and the all had analog

to digital conversion capabilities. Any one of the three chips would have been able to satisfy the needs of the control modules. The ATmega88 was the first chip out of the three that was removed from consideration for the base station. This was because the on-chip flash memory was quite low. We decided not to use it in the control modules for the same reason. The ATmega328 is only slightly more expensive than the ATmega328 but has 4 times the flash memory. The ATmega32A, while useful for the base station, would have been overkill for the control modules. The amount of pins on the ATmega32A would have taken up too much room on the board. Therefore we decided to use the ATmega328 for the control modules.

For the control module microcontroller we also put research into using an ATtiny microcontroller. The ATtiny series of Atmel microcontrollers are meant to be low power and low pin count. If we could use an eight pin or similarly sized microcontroller for the control module boards then it would save space and therefore money on the printed circuit board. One issue that turned us away from the ATtiny series was the lack of SPI and UART hardware modules on the low pin count chips. For the chips that had SPI and UART support the pin count was upwards of 20. This meant that there would be no large savings in terms of pin count from the ATmega328 to a viable ATtiny chip. Thus we saw no reason to use an ATtiny chip for the control modules rather than an ATmega chip.

6.2.4 Development Environment

To begin utilizing the microcontrollers that we chose we had to decide upon a development environment to use. We believed that having everyone in the group use the same environment would allow for easier collaboration and shorten development time. Research showed that there were two popular choices for writing programs for Atmel chips, Atmel Studio and WinAVR. Each of these two environments had pros and cons. Table 2 shows relevant comparisons between the two environments that led to the decision of what to use. The first thing that we considered when choosing a development environment was cost. We knew that certain IDEs, such as Visual Studio, require a license in order to use. We thought that since Atmel Studio was based off of Visual Studio that it probably required a similar license. After looking into this issue we found that Atmel Studio is free and WinAVR is also free. Knowing both are free we took under consideration that part of the group mainly uses Linux. Unfortunately neither of these development environments are usable on linux so we all agreed to use the Windows operating system to host whichever option we chose. Our next consideration was which IDE worked best along with the programmer that we decided to use for the microcontroller. We decided to use an AVR Pocket Programmer which uses a program called AVRDude for programming Atmel microcontrollers. The Atmel Studio development environment could be set up so the Pocket Programmer could be activated with the press of a button. This capability was a noticeable win over WinAVR which elongated the process. The feature that made Atmel Studio the clear winner of the two was the fact that it provided so much sample code. Things such as the utilizing the UART or the SPI interface for certain chips were about three clicks away while in Atmel Studio. While the two IDEs did have many similarities, we all agreed that Atmel Studio was the best choice for developing for our Atmel microcontrollers.

	Atmel Studio	WinAVR
Free	Yes	Yes
Linux Compatible	No	No
Integrated Compiler	Yes (GNU)	Yes (GNU)
AVRDude Integration	Yes	No
Sample Projects	Yes	No

Table 2: Comparison of Atmel Studio and WinAVR

6.2.5 Microcontroller Additions

To ensure that the microcontrollers we used on the base station and the control modules were operating at their full capabilities we added external crystals to their circuits. The external crystals boosted the microcontrollers operating frequency from their factory ratings of 1 MHz to the native frequency of the crystal. Because the lowest max operating frequency for the microcontrollers we chose is 16 MHz this is the frequency we looked for when choosing a crystal to use for the microcontrollers. We decided to use the NDK NX5032GA-16.000000MHZ-LN-CD-1 as the crystal for our microcontrollers. This crystal has a 16 MHz operating frequency and has the simple two connection design. For adding the crystal to our microcontroller we used the circuit pictured in Figure 11. The crystal connects to the XTAL pins on the microcontroller and the microcontroller gets its driving wave from the crystal. As the diagram shows there are two capacitors that must be utilized to get the circuit working. These capacitors are C_1 and C_2 and there is an equation for choosing their value.

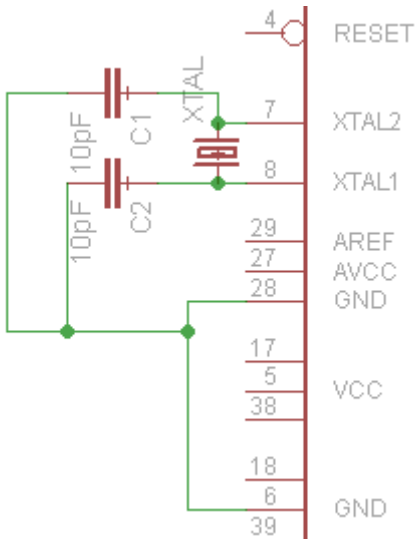


Figure 11: Microcontroller Crystal Schematic

The equation is for choosing the values for C_1 and C_2 is shown in Equation 1 where C_{stray} is dependent on board layout and is approximately 2-5 pF and C_L is a property of the crystal. For our calculations we used $C_{stray} = 3\text{pF}$. C_1 also is equal to C_2 so this simplifies the calculations. The value of C_L for the crystal we listed is shown as 8pF in the datasheet.

Using the given equation we found that the best value for the capacitors C_1 and C_2 necessary in the crystal circuit was 10pF.

$$C_L = (C_1 * C_2)/(C_1 + C_2) + C_{stray} \tag{1}$$

To make the external crystals work with the microcontrollers, certain fuses needed to be set through a programmer connected to the microcontroller. The datasheets for the microcontrollers have a fuse list that were used to make the proper alterations for incorporating an external crystal into the design. There is also a free online fuse calculator tool for AVR microcontrollers at [2] that can be used to quickly get the necessary fuses that must be changed for any sort of microcontroller alteration such as changing the operating frequency.

To get the microcontrollers running, the code we write needed to be programmed onto the chips flash memory. Atmega microcontrollers use SPI to transfer programs to the chip. There are different programmers that can be used to complete the task of uploading code to the microcontroller. We put research into which programmer to use in order to be the most cost effective with our design. The two programmers that our decision came down to are listed in Table 3. Atmel has a programmer called the AVRISP mkII. This programmer is supported by the designers of the chips so it was sure to work when ordered however it also came with a large price tag compared to other options. The online vendor SparkFun offers a product called AVR Pocket Programmer for $15 that accomplished all that we needed to get our programs onto our microcontrollers. The programmer relies on an open source program called AVRDUDE to get the hex file from the development computer to the chips flash memory. The program is free and the pocket programmer is much cheaper than the Atmel programmer which costs $34 if ordered from the Atmel site. The AVR Pocket Programmer was a clear winner when it came time to choose the programmer for our microcontrollers.

Programmer	Supplier	Price	Atmel Studio Compatible
AVRISP mkII	Atmel	$34	Yes
Pocket AVR Programmer	SparkFun	$14.95	Yes

Table 3: Comparison of Two Different AVR Programmers

6.3 BlueTooth Chip

The BlueTooth device for WHCS enables the base station to communicate with the mobile phone controller. Our guidelines for choosing a BlueTooth device included ease of use, reliability, size, cost, availability, and documentation. There were a multitude of BlueTooth devices to choose from. Special attention was paid to how well the BlueTooth device could connect to a microcontroller UART. Two BlueTooth devices, the HC-05 and the RN-41, showed the most promise. Our research of the two devices showed that they both had their advantages and either one could be implemented in our design. After careful consideration we chose to utilize the HC-05 in our design, however the RN-41 could still replace the HC-05 if necessary.

An important factor for considering the BlueTooth devices was if the internal settings of the BlueTooth devices could be changed and if possible how. Such internal settings include

things such as the device's BlueTooth name, baud rate, and passcode. These things need to be changed from their default settings or else many BlueTooth devices would have similar names, and they would all have default passcodes. We want to implement good security into WHCS so we need to be able to change the default passcode for the BlueTooth device. Also the baud rate is usually low in BlueTooth devices by default, which can be bumped up depending on the microcontroller being used. A BlueTooth device that could not be programmed easily was not an option.

6.3.1 RN-41

The RN-41 is a BlueTooth module designed by Microchip. This module is designed to be an all inclusive solution for embedded BlueTooth. It is clear that a lot of design went into this chip because it is very high quality and the data sheet is very thorough. Along with the high quality of the module comes the high cost. Of the two considered BlueTooth modules the RN-41 was much more expensive with a price of $21.74 from Microchip. The high price tag makes it a less appealing option out of the BlueTooth devices because they are effectively accomplishing the same thing. The module itself appears well designed visually and it has dimensions of 25.8x13.2mm so it is not obtrusive and could fit well onto a PCB. There are 24 pins on this device and the datasheet gives the dimensions down to the pin spacing allowing for easy PCB layout design. The RN-41 makes communicating with microcontroller UARTs easy by simplifying RS-232 down to the Rx and Tx wires. This means the only connections necessary for using the RN-41 with a microcontroller are power, ground, Rx, and Tx. The microcontroller's that we have decided to use include Rx and Tx pins that hook up directly to the RN-41. From the microcontroller's point of view the BlueTooth device does not even exist. The RN-41 acts as a transparent man-in-the middle and simply relays messages from a BlueTooth device to the microcontroller and vice versa. This is perfect for our design because the RN-41 could just be plug and play. With an advertised 100 meter transmission range the RN-41 meets the requirements we set for distance of BlueTooth reception. According to the datasheet the RN-41 has a maximum baud rate of 921K which means it goes above and beyond the transmission rates necessary for communicating between the mobile device and the base station.

The RN-41 has a manageable means of programming the internal settings. When the RN-41 is on, sending "$$$" over the UART lines puts the chip into command mode. From here there are a list of commands that can be passed in order to inquire or manipulate the state of the module. There are advantages and disadvantages to this approach. It is great that it is easy to program the RN-41 just by passing certain data while it is wired normally, however in the event that the sequence "$$$" was ever passed during operation it could throw off the whole system. This is not a terrible thing but it is worthy of consideration.

6.3.2 HC-05

The HC-05 is a BlueTooth module that shares many similarities with the RN-41. It is of comparable size to the RN-41 with dimensions of 27x13mm. HC-05 modules are also commonly sold along with a breakout board with male headers. This makes it an option to have the PCB include female headers and use them for installing the HC-05. Our intention for the base station containing the BlueTooth module is to have the PCB board hidden,

so using female headers for plugging in the HC-05 to the PCB is a viable option. The module is advertised as a low power class 2 BlueTooth device with power consumption for communication listed at 8 mA. This is lower power consumption than the RN-41. The max signal range is not listed in the data sheet, however we have tested this chip and have achieved a signal range of more than 50m which is more than enough for what we desired in our BlueTooth chip. Just like the RN-41 the HC-05 communicates to microcontrollers by simplifying RS-232 and only using the Rx and Tx pin. The maximum supported baud rate is 460800 which will allow for very fast data transfer and will exceed the needs of communication speeds in our system.

The HC-05 comes with default settings similar to most BlueTooth modules. The default baud rate is 9600 and the default passcode is 1234. In order to change this the module must be accessed in AT mode. AT mode is entered by utilizing pin 11 "key" on the HC-05. When this pin is held high, the module enters AT mode on startup and is ready to take commands. This means that whenever we want to program the BlueTooth module we will need to use a microcontroller with a UART connection to a terminal as well as a UART connection to the HC-05. This will require the implementation of a software Rx and Tx pin. This will only need to be done once because once the BlueTooth module has been programmed it retains that configuration. The requirement of holding the key pin high during startup of the module eliminates the danger of entering the programming mode during normal operation.

For WHCS we have decided to use the HC-05 as our BlueTooth module instead of the RN-41. Table 4 shows highlights of the features of each BlueTooth module that led to the decision. The main factors deciding this were the cheaper price of the HC-05 and the fact that they both accomplish the same thing. The table shows clearly that the HC-05 is a much more affordable option. The two chips were comparable in size, features, wiring, layout, and usability however at the price of $6.64 the HC-05 cost less than half of the RN-41. The RN-41 is the second best choice and can serve as a fallback if HC-05 chips went out of stock or an unforeseen circumstance occurred.

	Cost	**Range (meters)**	**Breakout?**	**Configurable**	**Size**
RN-41	$21.70	100	No	Yes	25.8x13.2mm
HC-05	$6.64	50+	Yes	Yes	27x13mm

Table 4: comparison of the BlueTooth chips

6.4 LCD

Being able to interface with WHCS like a normal wall thermostat is one of our project goals. Having a centralized display that can quickly display the most important information for homeowners is step up from traditional "dumb" thermostats. With a simple LCD combined with a touchscreen, users can have a way to control and query their home without having to find their phone.

To make this a reality, we chose the versatile, premade, and well supported Adafruit 2.8" TFT LCD[3] with a resistive touchscreen. Internally, the display is driven by the feature-rich ILI9341 chipset.[22] This chip is specifically created for a 240x320 pixel LCD with a focus on small, power-conscious mobile devices. Another reason we chose to buy this from Adafruit

and not integrate the chip directly is due to the complexity of the design. Also, with the abstracted product, there are plenty of usage examples[23]and Adafruit's excellent technical documentation.[24] This combined with Adafruit's libraries[25], assured that integrating the LCD was straight forward. The one issue with this solution was with the ILI9341 driver code: it was written to target the Arduino platform. Luckily, the Arduino platform is fairly close to bare AVR, minus the remapped pin numbers and some support libraries, so porting Adafruit's library was a feasible solution. It was also the simplest and most efficient solutions, therefore our base station uses a ported Adafruit graphics and ILI9341 controller library.

6.4.1 Capabilities

In terms of capabilities, we summarize the main features of the LCD module in Table 5

Specification	**Description**
Resolution	240x320
Colors	262K @ 18bits, 65K @ 16bits
Voltage Input	3.3 - 5V
Weight	40 grams
Dimensions (LCD itself)	2.8" diagonal
MCU Interface	Multiple. See Section 6.4.2
Touchscreen technology	Resistive (one finger)

Table 5: a brief summary of the pertinent features of the LCD module

These features are more than sufficient for our application as most of the drawing we did is non-realtime. Nearly all of the displayed information was text and UI elements, which don't change often. For an overview of our UI elements, see Section 6.4.5.

One of the beneficial features of the LCD is that it gives developers a choice of which interface to use. The broken out interfaces are 4-bit SPI and 8-bit parallel. For prototyping, we used the low pin count SPI interface, but for our final design, we used the 8-bit interface to avoid having to wait for lengthy SPI transfers. Also, our NRF radio is using the SPI bus, which needs priority over that bus. You may view a high level overview of the LCD module in Figure 12.

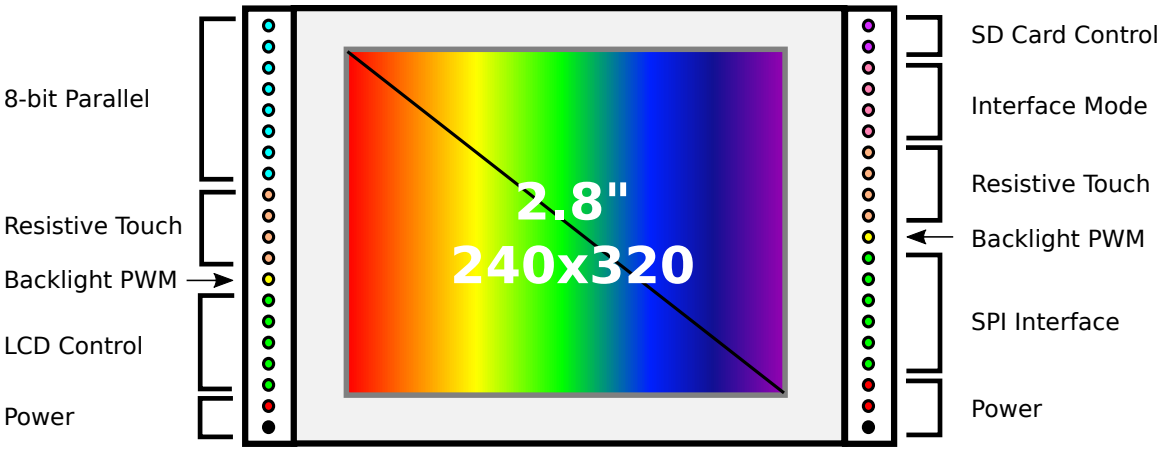


Figure 12: a high level outline of the LCD pin configuration and specifications

Another feature of Adafruit's module is the resistive touchscreen present on top of the LCD. With this, we don't just have to display information about WHCS, we can receive actions as well. Using this simple interface, we created a featureful UI library to communicate up-to-date information about the home while allowing for user control. The specific interface for the touchscreen requires 4 pins, 2 of which must be connected to the MCU's Analog to Digital Converter. By reading the resistance of the touchscreen, we were be able to calculate the position of a single finger. Using state tracking, we could generate separate TOUCH_DOWN and TOUCH_UP events for distribution to the appropriate UI element.

Finally, one extra component on the LCD module is an SD card slot. We are free to read and write directly to this card to store large images, such as logos for display on the LCD. We planned on storing at least our logo, but we didn't end up needing this feature as our logo is just text with colored lines;

6.4.2 ILI9341 Driver

In order to perform the required operations for drawing pixels to the LCD, we needed a robust driver to manage the state of the ILI9341 chip. This driver must implement primitives for choosing a position to draw and the ability to fill pixels.

Choosing where to draw The ILI9341 works like many chips in which it has a command set for controlling display parameters. One of these commands sets the window in which pixel data is drawn. Before a batch drawing operation, a window is selected by specifying the column and page addresses. These are set using the **Column Address Set (0x2A)** and the **Page Address Set (0x2B)** commands. From this point, the RAMWR command is sent followed by N different 16-bit pixel colors. This scheme of setting the window and filling the pixels is why the graphics transfers end up being so slow, which is discussed more in Section 6.4.2.

LCD State Management Beyond just drawing pixels, the LCD module has a wide variety of bonus features that could assist us in making a fully-featured interface. One of

these is vertical scrolling, which could enable us to cheaply draw familiar UI elements, such as list views. The most intensive state management comes from the early initialization of the LCD module straight from power on. This requires a long incantation of LCD commands with equally archaic parameters in order to reach the initialized zen. Luckily, there are multiple examples of slightly different LCD initialization sequences online for reference. In terms of power management, the LCD controller provides a command set to set the power mode. Although, the base station is on wall power, dimming the screen saves energy and looks professional.

LCD Performance Due to the real-time nature of microcontrollers, everything must be juggled as quickly as possible in order to have a good responsive design. A user event such as a touch event followed by a graphics redraw never takes prioirity over more important operations, such as sending and receving from the radios. This is a common theme throughout real time environments, so much so that highly specialized Real Time Operating Systems have been created to provide *assurance* that some operation/action will complete within a specified amount of time. We didn't have that luxury – instead we considered performance before hitting any performance walls.

The slowest part about managing the LCD is the synchronous transfer of commands and pixel data between the MCU and the graphics controller. Each ILI9341 command is 8-bits and before any pixel operation the CAS and PAS commands need to be set. These commands have two 16-bit arguments each. For settting just one arbitrary pixel 104 bits needs to be sent. This is a considerable overhead, so care must be taken to perform pixel operations in batch. Obviously, an algorithm that generates random pixels on the screen would have very low performance.

One other hit to potential performance is the lack of double buffering. The LCD chip only has enough RAM for one set of graphics memory, which means active drawing occurs on the viewing surface. In modern day graphics pipelines, double buffering and even *triple* buffering are common place as they eliminate tearing and free the developer from having to clear the screen. On modern day graphics cards, clearing the screen is an extremely cheap operation – on the ILI9341, it's just as expensive as every other drawing operation.

6.4.3 Touchscreen Driver

The touchscreen has code dedicated to polling the X+, X-, Y+, and Y- pins of the LCD. These pins give voltage readings that can be used to calculate the position of a single touch. They are read by using the built-in ATMega ADC. Due to there not being a dedicated touch controller, interrupts cannot be used to sense sharp changes (i.e a touch). This added to the processing time of the microcontroller's main loop as it has to constantly perform ADC operations and comparisons.

6.4.4 Graphics Driver

Unlike the previous ILI9341 driver, our graphics driver is in charge of taking the low-level primitives exported by the LCD driver and using them to draw useful screen elements. These include text, lines, rectangles, circles, and images. The driver essentially contain a

set of routines for drawing these objects given a set of parameters such as length/width for rectangles, radius and position for circles, and bitmap lookup-tables for the text. These functions and more are already created by Adafruit and we decided to go with their implementation.

Due to the unique hardware and software constraints (i.e limited clock speed and memory), we took take care to not exceed the capabilities of the hardware. This means floating point operations, which are required for shapes such as a circle must be optimized or used sparingly. A quick optimization for floating point operations, if necessary, would be to use a $sin()$ lookup table and only integer multiplications. The performance of the graphics driver was poor (50ms for a full screen draw), but given the slow clock rate, there was no significant improvements to be made to the library.

Algorithms Necessary The functionality of the graphics driver depends on some core algorithms for efficiently creating meaningful screen objects. One of these core algorithms is Bresenham's line algorithm.[26] This algorithm needed to be efficiently implemented as it serves as the base for nearly every derived graphics operation. For example, a transparent rectangle has four sides which results in four calls to this line drawing function.

Character Lookup Table In order to convey useful information, we display text to the user. This text is be stored in an efficient lookup table for quick drawing operations similar. On a limited embedded system like this, there is a limited amount of time that can be dedicated for font rendering. To save on time, used the existing 5x7 pixel font included in the Adafruit library.

Due to the very limited memory constraints of the base station microcontroller, this character lookup table is stored in PROGMEM. This embedded the bitfile in to flash, which then has to be read out using the AVR LDM instruction. This incurs a performance hit, but storing the characters in memory is impossible given only 2048 bytes of memory for the Atmega32-A.

6.4.5 UI Library

An LCD is nothing without a good library driving it. But even if all possible shapes and text can be drawn, without a good user interface library, the system is a painting. WHCS aims to have an LCD users *want* to interact with. This was accomplished with a feature rich UI library that enabled text, buttons, lists, and images to be drawn and organized simply. A sample WHCS home screen is mocked up and shows the required controls in Figure 13. In normal UI libraries, such as the Android View hierarchy, UI elements can be added on the fly. Unfortunately, unless the UI elements for WHCS are extremely low memory, storing dynamic UI elements may prove to be too costly for the microcontroller. This is primary due to `malloc()` (i.e dynamic memory) Until further testing is done, there is no way to say that this is possible or not beyond data structure size predictions.

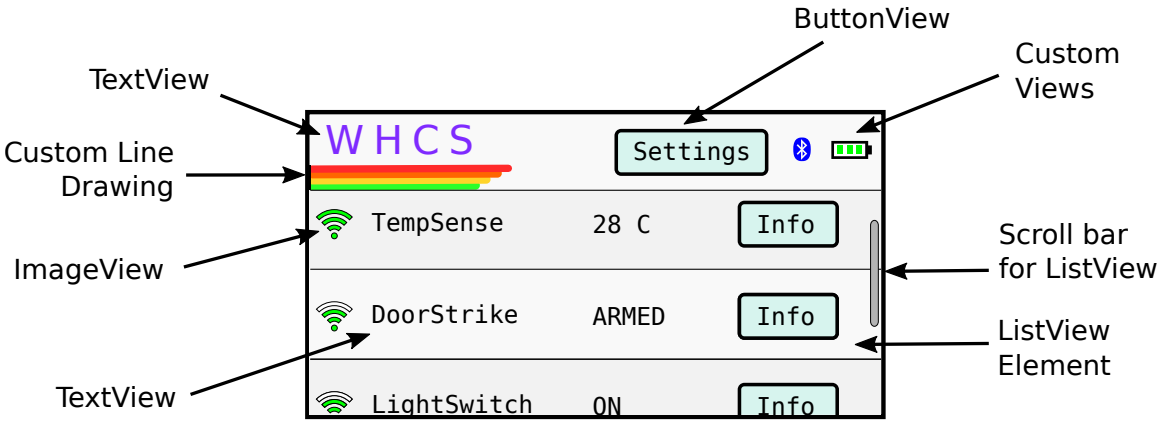


Figure 13: A mockup of WHCS' home screen with each control identified

View Abstraction When constructing a UI library, we use an Object Oriented class hierarchy to describe the controls. All drawable controls have the View class and a parent, which serve to contain all of the common properties that a control can have. This includes basic drawing and placement information, such as width, height, and X-Y position. This is illustrated in Figure 14.

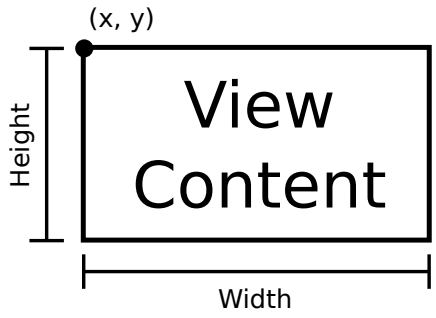


Figure 14: View abstraction properties

Making this abstraction allows for code reuse and simplifies the design of individual components. Specific components derive directly from the View class or each other to form more and more complex UI elements. Using Figure 13, created a UML representation of a potential view hierarchy, including properties and methods. This can be seen in Figure 15.

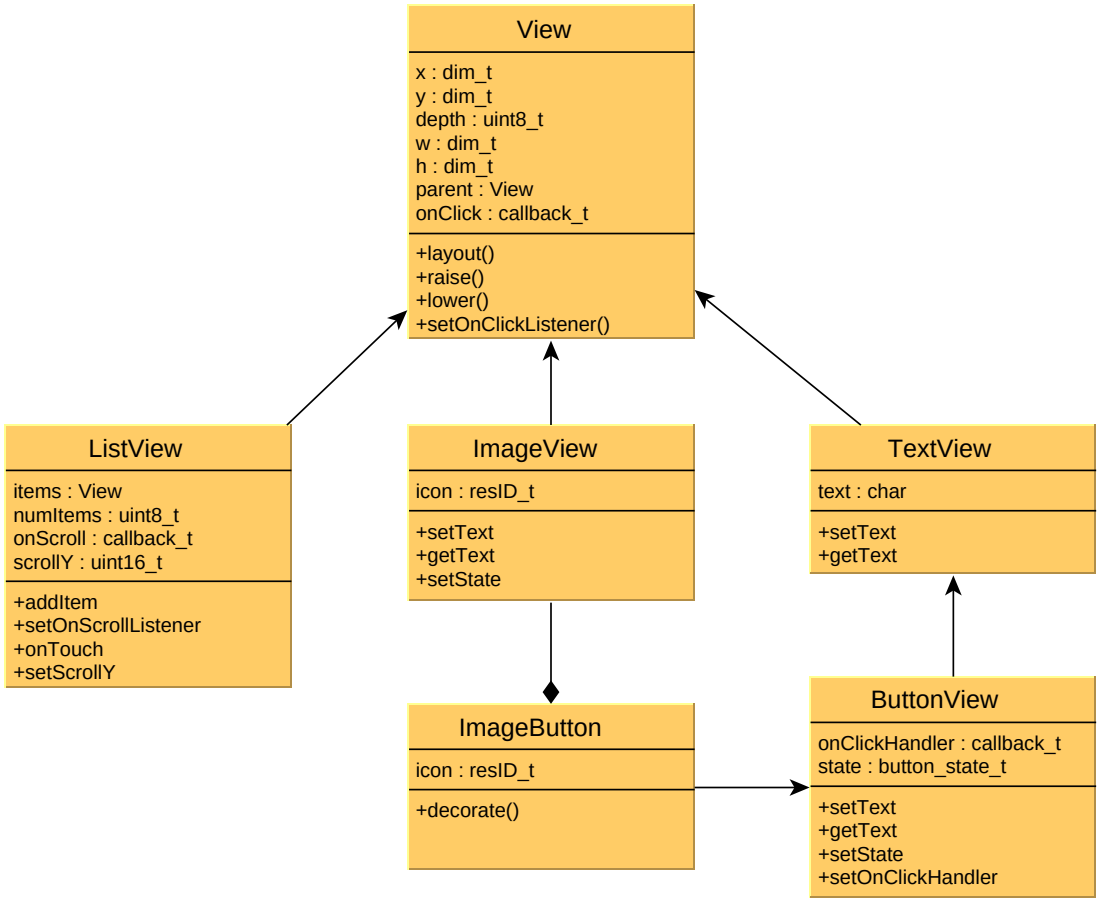


Figure 15: UML representation of the View hierarchy.

This hierarchy is modeled after Android's UI abstraction. This is done because reinventing the wheel for UI elements would have cost developer time. Android has more than proven that the hierarchy works in practice as well.

6.5 Android Application

For most WHCS users the mobile application will be the only physical interaction they have with the application. When we set out for development we made an easy to use application that should attract users enough to stick with our system. Operability and usability are emphasized in our design process. We made an appealing U.I. without complexity – after all we are targeting a simple solution to home automation.

6.5.1 Development Environment

Android is the mobile operating system that we chose to utilize for our BlueTooth enabled phone. The Android operating system is accessed through the Java language, which is a staple in the UCF curriculum therefore everyone in our group is versed in it. Developing on

Android is also a free endeavour where as developing on an iPhone requires enrolling in the Apple Developer Program. These programs actually cost a good amount of money that is unnecessary to spend. The Windows Phone platform is another option for the BlueTooth enabled phone, but they are very unpopular so we chose not to target this platform. With our target narrowed down to the Android operating system, we had to research what the best environment for developing our application would be. There were three options that we considered for managing the Android project each with their own perks: command line tools, Eclipse, and Android Studio.

The first development environment we considered for our Android project was creating our own project structure and using command line build tools. There are also debug tools available on the command line for Android projects. These tools would be necessary in order to do our testing on Android Virtual Machines running on our computers. This approach favors people who are command line or terminal oriented. Linux is popular within our group and the ability to do things from the terminal is appealing so this approach seemed like a good one. We realized that with the design we used for our project, it would become quite large and it might be difficult to handle without a dedicated IDE (Interactive Development Environment). This led us to looking into using Eclipse for developing our Android application. Eclipse seemed like a natural choice because it is what is recommended for using in the Java oriented UCF programming classes. The Android SDK provides an add-on for Eclipse that makes it a viable Android development environment. We were able to get this running and create sample Android applications. Inside of Eclipse the project structure for Android applications is laid out nicely. The debug tools are all organized at the top of the screen resulting in an easier development experience than debugging from the command line. The problem with using Eclipse as our IDE is that Eclipse is notorious for being slow and unwieldy.

Recently Google released a development environment named Android Studio that is made specifically for developing Android Applications. No one in our group had any prior experience using this IDE, however we realized that due to it being tailored specifically for Android it was probably better than anything else. This turned out to be correct, because it was much easier for us to create an Android project and navigate our code from within this IDE. We also decided to use Android Studio because it has built in Git support for source control. This meant that as we were writing our code we could easily submit our changes to a remote Git repository.

6.5.2 Use Case Diagram

The central use cases for WHCS are toggling the state of certain devices within the home and monitoring certain states. For example a user of WHCS will spend most of the time turning on lights, checking whether a light is on, or checking the temperature reading of a certain sensor. There are certain other use cases that were necessary in order for WHCS to be a functioning application, as well as to make it have a robust feel. Features like speech activation and creating endpoint groups are usability features that are not necessary in order to accomplish the central goals of WHCS but add to the applications value. Connecting to the base station the first time you use the application was a necessary use case that had to be incorporated into the application. Figure 16 shows the use case diagram for the WHCS application.

The design for the WHCS application involved making sure that performing the common use cases such as checking status and toggling endpoints were very fast. It was our intent for the user to be able to perform these tasks without having any knowledge of how the application works. Speech recognition is a supporting feature so it did not need to be a central focus like the area that will visualize the control modules. When the user wants to perform speech activation it involves pressing a button to prompt the speech recognizer, and then giving a command to WHCS. In order to make the speech activation feature more promising, the user has the ability to rename endpoints for activation. Creating endpoint groups is a feature that is not used frequently but adds a lot of value to the application. Users only have to create an endpoint group once for it to last in the application. Creating an endpoint group is a simple task involving assigning a group number to endpoints. That number is the endpoint group, then that endpoint group's state can be toggled.

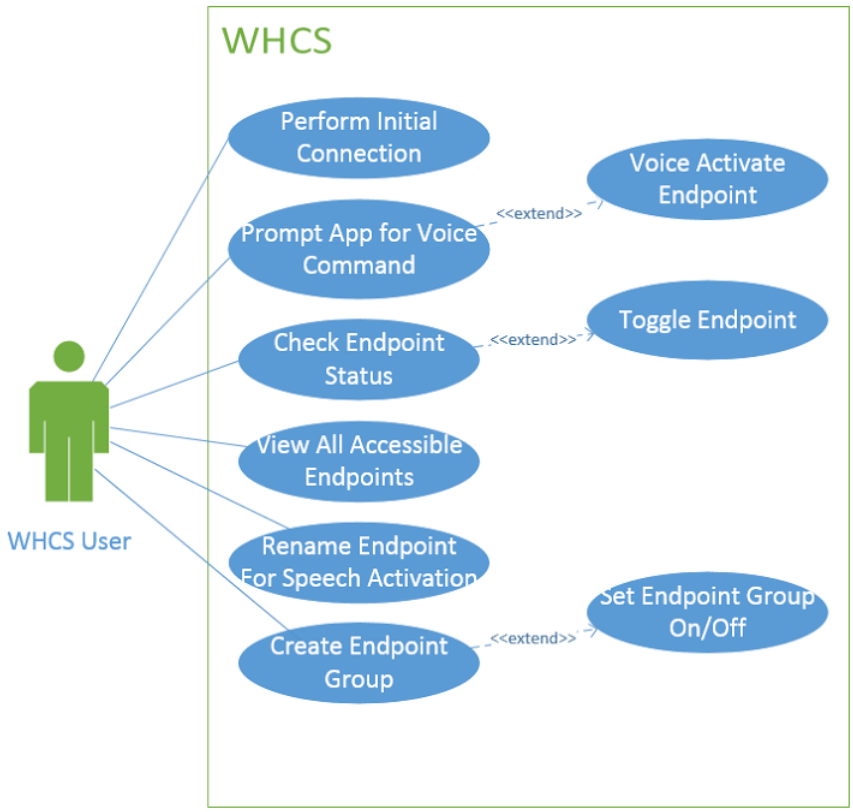


Figure 16: Android App Use-case diagram

6.5.3 Speech Recognition

The Android application for WHCS offers speech activation capabilities. These are on top of GUI activation capabilities. The speech activation sequence begins with the press of a button to start the speech recognition. The user is prompted with a microphone and can then give his command. The commands are formatted like "light one on." When the user gives commands using the speech method, a notification is given indicating the success of interpreting the speech into a known command. If the user's speech does not match a known command, the speech is shown back to the user to show what went wrong. We predicted

that the most frequent cause of this will be the Android phone mishearing the user. In the event that the speech matches a command, the application displays the command to the user and then perform it. The following flow chart in Figure 17 displays the sequence of events happening when a user performs speech activation.

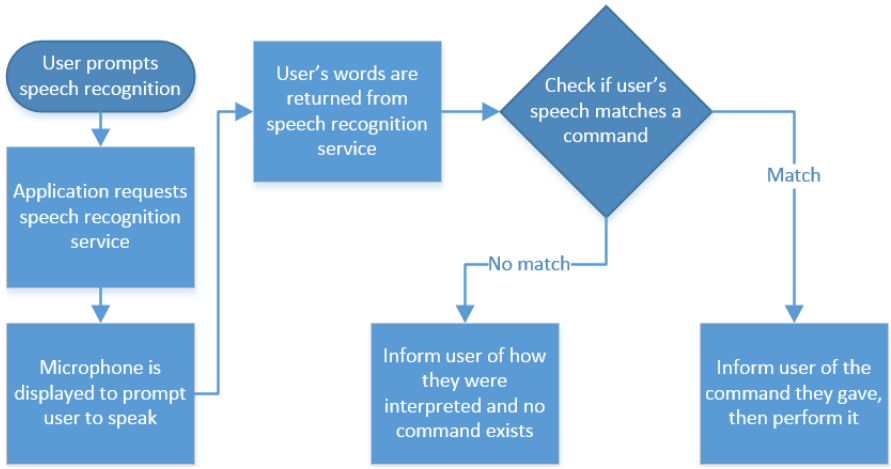


Figure 17: Android app speech activation chart

The goal of the speech activation feature is to be easy to use. In order to promote the usage of this feature we added the ability for users to rename the endpoints that the speech commands can target. For example the user can change "light 1" into "living room light." This way the user can say "living room light on" to the application in order to turn on the living room light. To do this, data structures need to be stored in the application which hold the preferred name of each type of endpoint. Endpoints can be distinguished by the type they are, their individual identifier number and their preferred name. The preferred name is stored when the application is closed so a permanent source of storage is needed to do this. The solution that we use for permanent storage is a SQLite database which is a common tool in Android applications.

In the code for our application we use the Android speech recognition API (Application Program Interface). Android has a speech recognition service that can be started by requesting it within an application. We request this service to be run by using an Android construct called an intent, specifically the recognizer intent. Once we request the service to be run, it gives us the text that it produced from listening to the user's speech. The code that performs this process ends up bloating up the application so we sought to develop a wrapper class in order to perform the request for the speech service and simply hand back the text. However because of the Android design philosophy, creating a wrapper class to start the speech recognition service was not easy enough to make it a worthwhile endeavor. Thus we concluded the best approach is to keep the calls to the Android speech recognition API within the class we use for our main activity.

6.5.4 BlueTooth Software Design

Bluetooth is the technology that allows WHCS users to interact with the base station from the mobile phone. This means that proper functioning BlueTooth software needed to be

written to ensure that users could interact with WHCS. From the user's standpoint the only knowledge of Bluetooth required is the ability to perform an initial connection to the base station. Once a user has connected to the base station once through the WHCS app we are able to cache the base station device and allow for automatic reconnection every time the application is launched. This is an important abstraction for the user because the user should not have to spend time handling BlueTooth connections every time they open the application. Figure 18 shows what the BlueTooth software will be doing whenever the user opens the Android application.

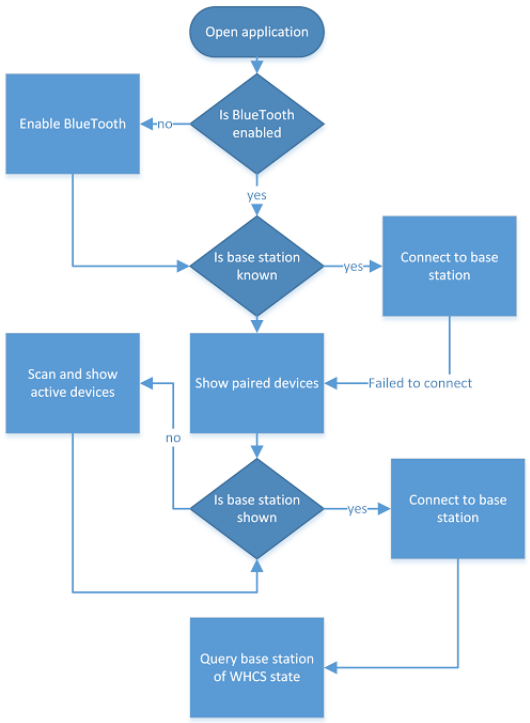


Figure 18: Android Bluetooth Startup Flowchart

In Figure 18 we see that the first check that is made is to ensure that Bluetooth is enabled. The Android operating system requires applications to ask the user whether they want to activate Bluetooth or not. It cannot just be turned on. If the WHCS application is opened and Bluetooth is off we prompt to the user to turn it on and if they refuse we exit the application. When it has confirmed that Bluetooth is on, the application can check to see if it knows the base station device. If the base station device is known then the application can skip asking the user what to connect to and can perform the connection automatically. This is what should be happening most of the time. If the base station is not stored in the applications data then the application will have to prompt the user to connect to a base station. When connecting to a device there are two possibilities for connection, paired devices and non-paired devices. The application first shows the user all devices that their phone has paired with previously, in case the application somehow forgot the base station. If the base station does not show up in the paired devices list, the user is able to search for active Bluetooth devices and select the base station. At the end of this start up cycle the WHCS application has an active Bluetooth connection with the base station that can be used for full duplex communication.

Our application leverages the API and design guideline for using Bluetooth from Android phones. The underlying driver for Bluetooth communication utilizes sockets similar to network sockets in other languages. Android offers a class named BluetoothDevice which contains all the address information necessary for opening a socket. When our application scans for devices or asks the user to pick an option from the list of paired devices this is to get the BluetoothDevice to open a socket from. Once we have obtained that BluetoothDevice we create a BluetoothSocket through one of its methods. Once a BluetoothSocket has been opened through calling connect, input and output streams become available that allow us to send and receive raw byte data. This is a primitive form of communication but it is also exactly what we wanted. All data that we send or receive from the base station over Bluetooth is in the form of a byte array. This form of primitive data transmission allowed us to implement certain communication protocols between the Android application and base station.

Once a BluetoothSocket has been opened on the Android device the application can begin communicating with the base station. We use a custom communication protocol between the Android device to ensure the base station can properly interact with the application. This protocol allows the Android application to give commands to the base station such as inquire about the state of the control modules or to toggle state within the system. Whenever the Android application wants to send a message to the base station the software creates a packet with a certain structure. The packet contains a byte for letting the base station know that a command is being given, the command itself, any variables for the command, and then a byte for finishing the command. The base station receives one byte at a time due to the serial nature of Bluetooth communication but it is able to parse the packets it receives in order to figure out what action the application is trying to perform. Figure 19 shows a visual representation of the communication between the aplication and the base station.

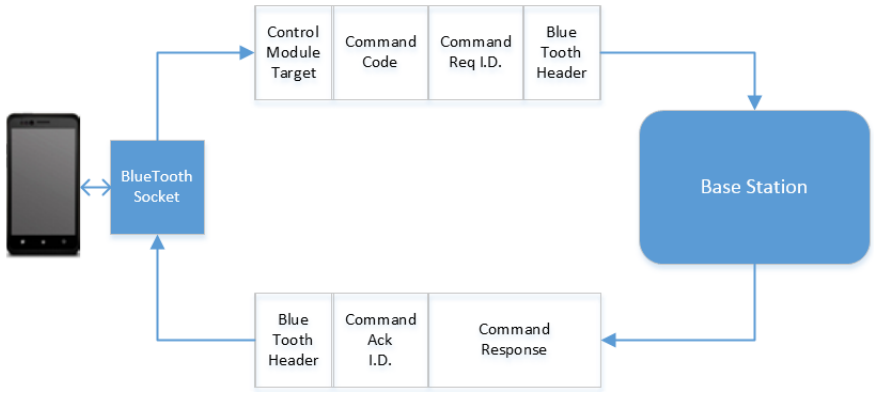


Figure 19: Visual of Communication Between Android Device and Base Station

6.5.5 GUI Philosophy

Our goal for the development of the user interface was to make something simple that users could navigate quickly and efficiently. There was no need for the UI to be deep or hold the user's attention. The only purpose of the GUI is to provide intuitive visuals for interacting with WHCS. When we developed the GUI we wanted to minimize the time it took for

the user to open the application and make a change within the system. For example the user should be able to open the application and turn a light on or off in the shortest time possible. This means opening up to a screen that lists all possible end points in the system that can be targeted by a command. The middle layout in Figure 20 shows the view that lists all of the accessible control modules in the system.

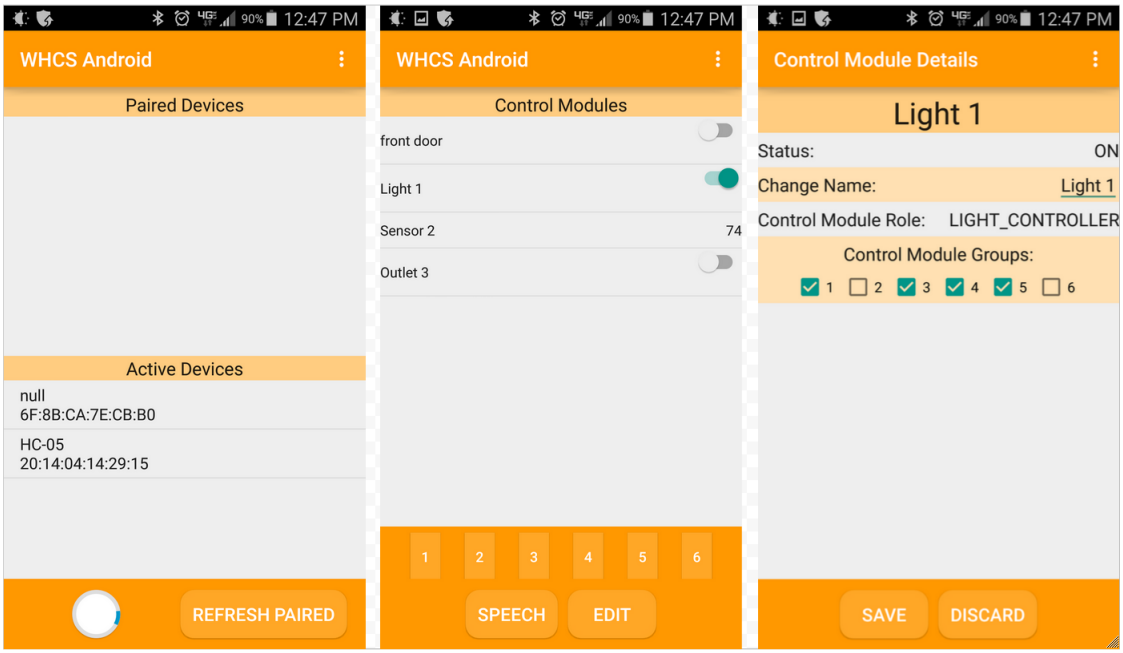


Figure 20: Android GUI Layout

As shown in Figure 20 there are layouts that provide support around the main list layout. The first layout in the left side of the figure is what the user would see when the base station is not known to the application yet. The user would need to select the base station from a list of paired devices or perform a BlueTooth scan for active devices. Once the user has selected a base station then the base station can be saved in the application and the user should be able to avoid seeing this screen again. The user would be viewing the main list layout at this point. From the main list layout the user can navigate to the individual control module viewer. This is achievable by clicking on the name of a control module. The individual detail viewer allows the user to toggle the state of the control module, change the speech recognition name of the control module, and assign the control module to a group. The detail viewer also lists the current state of the control module.

In Android different aspects of a GUI can be created in two different ways, fragments and activities. Typically fragments are used when two different screens serve very similar purposes or are trying to accomplish a shared goal. Fragments are typically used when exchanges are meant to be done very fast between screens. In the case of our application we use separate activities for each of the screens. This is a logical approach because the layouts in our application are independent of one another. The base station connection layout serves as the root activity and any other screen is an activity that is placed upon it. For example when the application opens up the first time it tries to open the connection activity and if it successfully connects it places the list activity on the top of the GUI stack.

When the detail viewer activity is called it stacks on top of the list activity and when the user is done with it, it is removed off of the stack.

To make our list activity look clean and function effectively we created a custom adapter. In Android, adapters are the classes that allow objects to be transformed into data that a listview can turn into list items to be displayed to the user. The name of the adapter is cmAdapter. The cmAdapter that we created has an array of control modules as one of its fields, as well as a function named getRow that it inherits from its base class Adapter. The cmAdapter knows how to get the data from a control module object necessary to populate the main list. The main list activity constantly calls the getRow method that is present in our adapter to fill the list. This creates a nice object oriented design for listing all of our control modules. If we want to display different data for control modules, we can simply alter the getRow method that is implemented in cmAdapter. Figure 21 shows the class diagram for the cmAdapter. The class is simple but provides important functionality for the Android application.

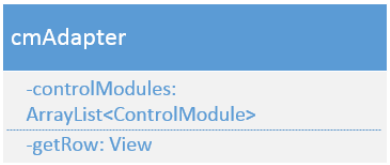


Figure 21: `cmAdapter` Class Diagram

6.5.6 BlueTooth Listener Class

When the WHCS application is communicating with the base station it is easy for the base station to be interrupted and start parsing communication using the UART interrupt vectors on the microcontroller. We wanted our application to possess the same event driven capability so we created the BlueToothListener class. This class handles listening for any incoming BlueTooth communication aimed at the phone. The class must be initialized by telling it what BlueTooth device it should be listening for. Once this happens it can create a thread and constantly check to see if the BluetoothSocket's input stream contains any data from the target device. If the inputstream contains data then we know that the target device has transmitted to the application. The BlueToothListener class raises an event whenever receipt of data has been confirmed. This allows the application to conform to event-driven Android philosophy. We can design around the BlueToothListener class and subscribe to the event it raises whenever data has been received. This is one of the core classes for communicating with the base station. Figure 22 shows the class diagram for the BlueToothListener class as well as the classes associated with it.

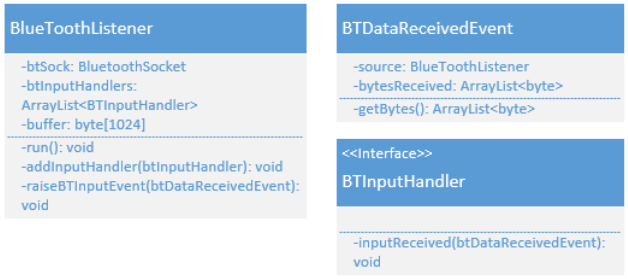


Figure 22: `BlueToothListener` Class Along With Supporting Data Structures

The BlueToothListener allows the application to directly hook up a parser for the incoming data. We can create a custom class that parses incoming byte arrays and transforms them into an understandable format for the application. This class would implement the interface for handling the data received event and could dictate what happens when certain data sequences are received. For example when the application asks the base station what control modules are currently known and active the base station would respond raising the data received event. The parser would begin working on the data received because it would have been subscribed to the event. The parser would realize that the data received is an indication of the state of the system and would have a case for handling what to do when this type of information is received. This would be how the communication protocol for receipt is implemented on the Android application.

6.6 Power Hardware

Throughout this section we will be discussing all that deals with the power. For reasons that will be discussed in greater detail in Section 6.6.9 we decided that each control module and the base station would have a power board that would be separate from the PCB of the control modules and base station. The reason for each power board is to convert 120VAC to DC lines of 3.3V 5V and 12V. We will also needed to be able to switch on and off 120VAC for the outlet and light switching control modules as well switch on and off the 12V used to operate the strike. The designs for each board will be mostly the same with slight variations depending on the application. This section will also not go into the specifics of why certain designs were chosen over other designs but will provide a big picture view of how our design works.

6.6.1 Design Summary

To start off we will explain the baseline design of our board. Figure 23 is used to explain what is constant for every board design. This basic design provides power to the microcontroller using AC outlet power. First the AC power is transformed with a transformer down from 120VAC to 14VAC this 14VAC is rectified using diodes into 19.8 VDC. At this point the line goes through a DC to DC converter that will transform the 19.8 V to 5V. This 5V line leaves the power board and enters the PCB containing the microcontroller and turns it on.

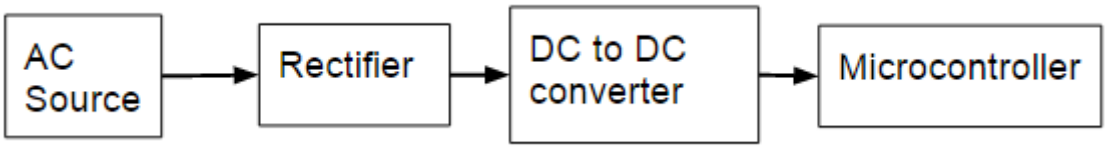


Figure 23: Baseline design for power board

Off this baseline design we will start to explain how the control modules and base station power boards differ in design. The first board that I'd like to go into is the design for the light control and outlet control. The basic function of these control modules is to switch on and off power to either lights or outlets. Basically in addition to the line taken from the 120VAC to power the microcontroller we have a line to power the actual application (the light and/or outlet). This is shown in Figure 24 shown below. This extra line will run along our power board and is only interrupted by a 5V operated relay. This relay will be connected to the microcontroller so that from the microcontroller WHCS will be able to open and close the 120VAC line. Initially we considered placing the relay on the power board, but in the end we decided it made most sense to place the relay on the control module. This is the easiest way to implement switching into our design. The only difference between the light and outlet modules is the load that is placed at the end of it.

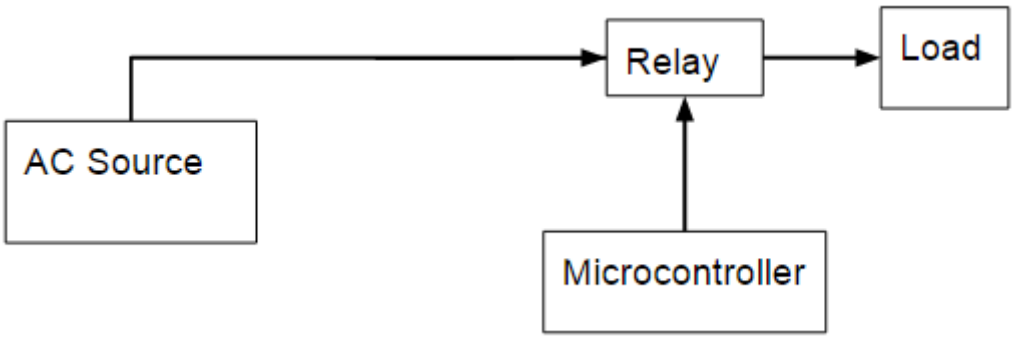


Figure 24: Additions to the baseline design for the implementation of light and outlet control module boards

The next board design to be addressed is the design for the door access module. The design of this module is actually very similar to the design of the light and outlet control modules . Door access is fairly simple, it consists of providing or not providing power to the electronic strike in order to allow the user to lock and unlock the door. The strike runs on 12V of DC power and draws 450mA. Figure 25 shows the basic design of the door access control module. Similarly to the light and outlet control modules it too has a relay that is used in order to provide the switching. The only change is that electric strikes operate on 12VDC instead of AC power. In order to provide the 12V an additional DC to DC converter (in addition to the DC to DC converter used for the microcontroller) is used after the rectifier. The 12V line in combination with the relay from the microcontroller is all this module needs to perform its task of providing power to the door access module. We also considered adding a back up battery to this design. The reason for this is that we wanted the microcontroller

to be powered even if a power failure occurred. The backup battery would not provide power to the 12V line meaning the strike would remain in locked mode. However having the microcontroller powered would allow it continue completing tasks such as checking the state of the system. In the end we decided not to implement a backup battery.

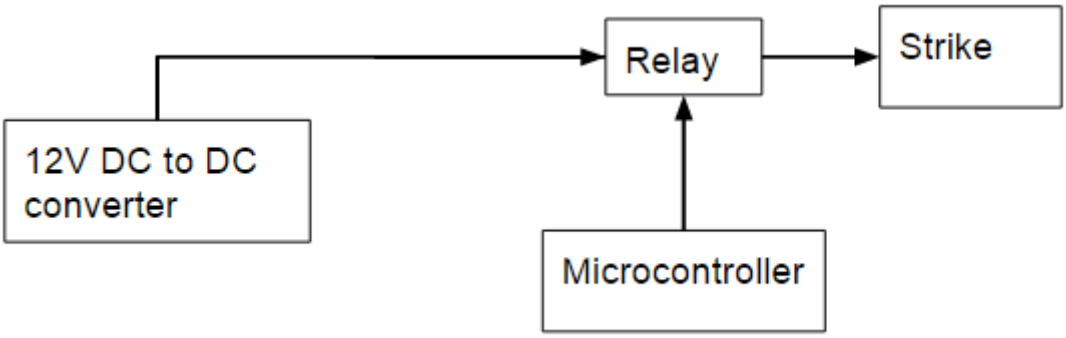


Figure 25: Additions to the baseline design for the implementation of door access module board

Another control module that must be discussed is the temperature sensor control module. There is no flowchart used to explain the module of the temperature sensors. This module really only needed the basic design that provides the microcontroller with 5V. if we had implemented a backup battery the information about the temperature of the home could still be gathered from the sensors even if there is a power failure.

Lastly and arguably the most important is the design of the board for the base station. The base station in addition to the microcontroller will make use of NRF and Bluetooth which require 3.3V lines. The 3.3V line will stem out from the the 5V line as shown in Figure 26. In order to make this step down we used a switching regulator. Like the temperature sensor module and the door access modules, we also considered equipping the base station with a backup battery in case of power failure. This would have allowed the base station to be fully operational even if the power goes out.

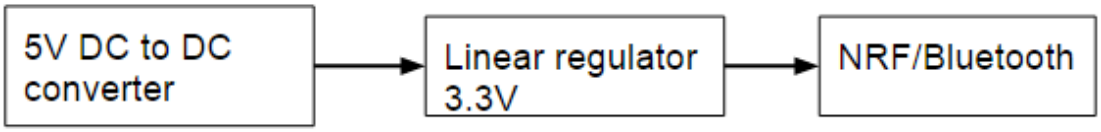


Figure 26: Additions to the baseline design for the implementation of the base station board

The only reason we didn't consider equipping the light and outlet modules with a backup battery is because a backup battery for these modules would serve no purpose. The backup batteries allow the microcontrollers of the control modules to remain operational, thus allowing the microcontrollers to complete small tasks such as checking the state of the system. For light and outlet applications however there is no need for this. In the case of a

power outage it can be assumed that the state of the lights and outlets if off. There is no need to check this with a microcontroller and therefore a back up battery would be useless.

6.6.2 Power Consumption

In this section we will discuss the amount of power consumed by the boards. First of all it is important to make note of the fact that saving power is not of incredible importance to us. While it is important not to be incredibly wasteful to the point that it becomes a problem, power was not something that we decided we wanted to be competitive on. Had we wanted to be more competitive with power, we would have taken a lot more into account and made different design decisions. For example with the microcontrollers we would have looked more carefully into the amount of current that they drew to help us weigh our decisions. MSP430 boards for example would have been attractive because of the low amounts of current that they draw. With that said we made all of our decisions based on performance in other aspects. The comparisons that we made and the details that were of importance to us can be shown in the sections were we decided on each component that we selected.

In Table 6 below we have all the information on the amount of current drawn from the different elements in our design. Note that the information from the datasheet about the current drawn for the microcontrollers is under a clock speed of 16MHz. We choose to run everything at 16 MHz because is was the maximum speed for the Atmega32A. Although the Atmega328P can run at a higher clock speed of 20 MHz, we decided tomake everything consistent to run at 16MHz. The reason why we give ranges for the currents and voltages of the microcontrollers is because there are a number of different current voltage relationships that can achieve the same clock speed. This isn't to say however that any combination will work. If a lower current is selected a higher voltage must be selected in order to maintain the same clock speed of 16MHz.

	Operating Voltages	**Current Drawn**
Atmega32A	4.5-5.5V	12-16mA
Atmega328P	4-5.5V	Active: 7-11mA Savings: 1.75-2.75mA
Electric Strike	12V	450mA
Adafruit TFT LCD	3.3V	150mA
Bluetooth HC 05	1.8-3.6V	35mA
nRF24L01+	1.9-3.6V	13.5mA
TMP 35 36 37	2.7-5.5V	.05mA

Table 6: Currents and voltages of devices used in WHCS

Note that the microcontroller of the control modules shows both an active and a power saving mode. When the module is in operation it runs in active mode, however since our microcontroller has the capability switch into a less consuming power mode, we will utilize this mode in order to save power. The microcontroller for the base station also has a low power setting for idle use, yet we do not run in this mode for the base station because it is impractical. Some of the datasheets gave us specific information on how much current would be drawn while for others we were simply able to estimate the limit. For example the LCD (most of it's current is drawn from the backlight) only gave a description of the LDO

that would be 3.3V at 150mA. Therefore we were able to assume that the current would not surpass 150mA.

As you can see the components in WHCS do not draw a lot of power. This is because the components that constitute the system are all intrinsically low power devices. Based on the fact that Power = Voltage*Current and that not all these devices are used at a time. We can see that the devices themselves never really draw that much power.

This data is used as a reference later when the amount of power drawn plays a part in design decisions. Again this section was not made to try and explain why certain decisions were made in order to try and conserve power. Our design is not heavily concerned with testing the limits of low power applications. Rather this section was made to present the data of the power usage that comes from the devices that have already been selected to be used in our design.

6.6.3 DC-to-DC Converters vs. Linear Voltage Regulators

In our control modules and base stations we are required to provide lines of three different voltages 12V, 5V and 3.3V. This could have been accomplished with a voltage regulator or a DC to DC converter. Voltage regulators are variable resistors that dissipate energy as heat to get the desired voltage levels. The advantage of voltage regulators is that they are cheap. The issue is that they are highly inefficient. Linear voltage regulators are good for low power applications where not too much power will be wasted due to inefficiency. Whereas DC to DC converters are more appropriate for large step downs in voltage.

The basic tradeoff between the two technologies is cost vs efficiency. Dc to DC converters are capable of achieving efficiency levels as high as 95% but cost close to $10. The efficiency of linear voltage regulators depends on the difference between the input and the output voltage. Yet the cost of a linear voltage regulator is usually below $2. Power wasted in a linear voltage regulator can be determined by using Equation 2.

$$P_{wasted} = (V_i - V_o) * i_l \tag{2}$$

Where P_{wasted} is the power wasted, V_i is the input voltage, V_o the output voltage, and i_l the load current.

The basic question becomes how large of a stepdown are you expecting to have from your regulators. In our design we decided to have a step down from 19.8 volts down to 5V and/or 12V, as well as a step down from 5V down to 3.3V. Based on these step down values it made most sense for us to use a DC to DC converter to step down 19.8V to 5V and/or 12V, and to step down from 5V to 3.3 volts.

6.6.4 Backup Battery Configuration

Initially we considered equipping each control module along with the base station in WHCS would be equipped with a backup battery. In the case of power failure WHCS would still be able to carry out the basic function of checking the state of the system. Actual operation of some of the control modules would be impossible without the use of AC power. Yet

checking statuses such as temperature and the position of door locks would still be fully operational. This was meant only to serve as a short term solution to power failure.

Although we decided not to use backup battery into our final design, here are some of the designs we considered for the backup battery. The circuit below in Figure 27 was made in Multisim. [4]

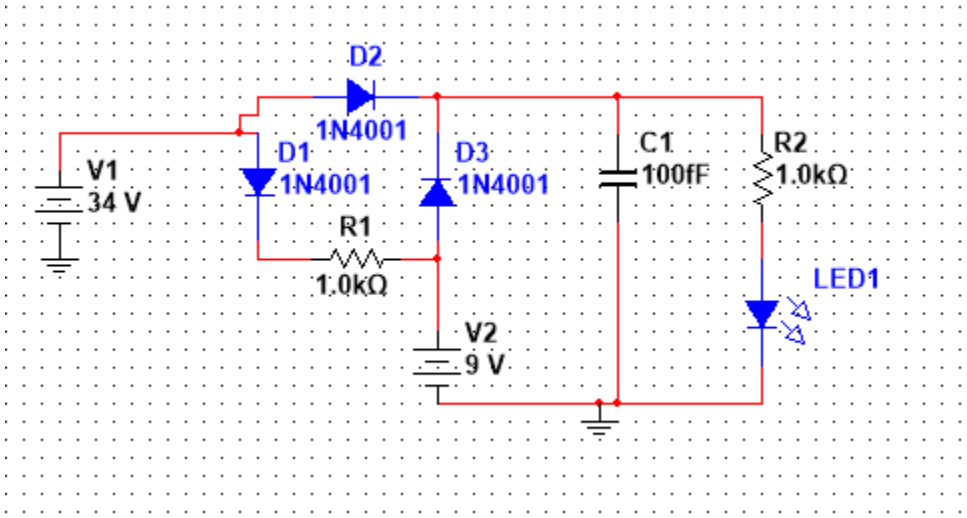


Figure 27: First possible design for the backup battery

This design is simple. The circuit being open indicates when there is a power failure, while a closed circuit signifies normal operation. The circuit shows two sources, one (the one farthest to the left) that is of a higher voltage and acts as the primary source and a secondary lower backup voltage source. Since the secondary source is of lower voltage it will not discharge until the voltage of the first source drops below a certain level. This is how tradeoff occurs. The secondary source has no potential until the first sources potential drops below a certain level. In Figure 27 we see that during normal operation (closed switch) that this design will recharge the batteries, that is if the batteries are rechargeable. This feature can easily be taken out by removing the diode and the resistor that feed into the secondary battery.

In this design it is very important that the secondary source be of a lower voltage than the primary source. If this is not the case the batteries will discharge even when the primary source is functioning properly. Initially we believed that this would be a handicap for our project and that the design could not be used. Yet we realized that there was a way to ensure that the primary voltage was higher than the secondary voltage. If we were to place the backup battery before the DC to DC converter, and allow the DC to DC converter to do a large portion of the voltage step down. Say we had the transformer only transform down to 24VAC (roughly equivalent to 34 VDC) then had the DC to DC converter bring the voltage down to 5VDC. Our primary voltage would be 34V and our secondary backup source could be anything lower than 34V that is accepted by the DC to DC converter.

Designing in such a fashion has it's advantages and it's disadvantages. The benefit of placing this design right before the DC to DC converter is that no matter what the voltage is coming into the converter (as long as it is within the range or acceptable voltages for the

converter) the output will always hold the same voltage level. The main disadvantage is that the required voltage rage needed by the voltage regulator may cause us to need higher voltage batteries than we would have needed otherwise. Say for example we need a 5V line, the DC to DC converter takes voltages from 9V-40V and converts it to 5V. For our back up battery we are now required to use a 9V battery whereas in another design we may have been able to get away with a 6V battery source.

Because of the high efficiency level of the DC to DC converters using a higher voltage battery than may have been required is not a problem of wasting power rather a problem of design cost. 9V batteries cost around $6 for a pack of two while AA batteries cost about $14 for a pack of 24. The cost difference a 9V battery and a 6V source with AA batteries is $0.33. The 9V battery is more expensive but it actually isn't that much more expensive. The cost/volt of a 9V battery is better than that of the AA batteries. Unfortunately later we realized another problem. The average AA or 9V battery could not supply the current requirements we needed. This was largely why we decided not to implement the backup battery.

The design above works because we have decided to use a DC to DC converter with a decently sized voltage step down.If however this were not the case and the primary and secondary designs were much closer in voltage levels than an alternative design would have to be used. There are a few other design possibilities that were explored.

Another design that was explored was to use a relay in order to drive the switch. In Figure 28 current flows from the primary source to ground which activates the relay allowing the primary source to power the load. If there were a power failure in the primary source the relay would switch to circuit with the backup battery and thus the secondary source would take over. The main issue with this circuit is that it requires a line that goes directly to ground, thus wasting energy. The effect of this can be lessened by choosing a large resistor value.

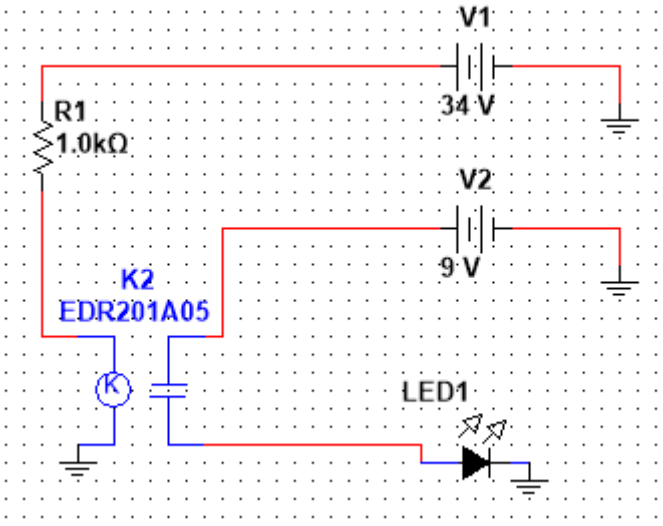


Figure 28: Possible design for back up battery[16]

In the end trouble finding the appropriate battery is what stopped us from building a backup battery for WHCS. We would have liked to have found a small battery that was at least

9V and could supply our current requirements.

6.6.5 Transformer Choice

One of the basic requirements for our project is to have a transformer to convert the AC power to DC power. There were however a lot of different transformers to choose from. With size and performance having a large impact on which transformer we chose. From previous sections we decided that our transformer would convert 120VAC to 14VAC. The first thing we considered was how much current our circuit would load draw. We first consider the amount of current drawn so that we could determine the VA (volt amperes) rating. The VA rating is an actual specification of the transformers that helped us make our decision. Volt amperes is different from watts in that it is the "apparent power" instead of the "real power". In order to find the VA rating we needed the currents and voltages used in our system.

The devices that our system powers needed to be taken into account in order to make sure we are making the correct approximation for how much current is being drawn. Initially we made approximations of the amount of current drawn based on datasheets and component characteristics. When we were in the prototyping stage we were able to run test to see if the actual amount of current drawn matched the amount that was expected.Yet in the design stage we just drew our information from the datasheets. If we look to Section 6.6.2 there is a table that makes a nice summary of the current and voltage requirements of each device.

There are quite a few devices to consider that draw current. There's the microcontroller for the base station, the microcontrollers for the control modules, the electric strike, the temperature sensor, the LCD screen, the bluetooth module, and the nRF chip. Now not all of these devices are used at the same time. Therefore we don't really need to take into account everything at the same time. We made our conclusions of current based on the max current drawn at a time. The operation that would draw the most current would come from using the electric strike from the base station. When we add the currents involved with operating the strike the current would be 665mA. The VA can be calculated by I*Vrms. Were "I" pertains to the current directly after the transformer. The voltage we already know to be 24 Vrms. Using the current from the device endpoints as an approximation of what the current would be directly after the transformer we would calculated the VA rating to be 16. When selecting our transformer we needed to pick a value that is above 16, we decided on 20VA.

After determining the VA rating there were still other factors that we need to take into account. Another thing we considered was the type of transformer. Did we want a regular transformer or a center tapped transformer. This was decided by the type of rectifier we used. We decided on a full wave bridge rectifier and therefore used a regular transformer as opposed to a center tapped transformer. There are also a whole list of mounting methods for transformers: Chassis, DIN Rail, PCB, Snap in, Surface Mount, Through Hole, Wall Mount. As will be discussed in Section 9.3.3 our design is a full installation design meaning that the power board along with the control module and base station boards are placed into the framing of the home. Now that means that the transformer could have been directly connected to the framework of the house, that would be a very sturdy way to do it but would also require extra steps for installation. Having a transformer that is part of our

power board kept things simpler as far as installation goes. In Section 6.6.9 we discuss why our power board is a through hole power board. Therefore our transformer is also a through hole transformer. Size had some influence but was of less influence than the other constraints involved with choosing the part for our design.

6.6.6 Rectifiers, Diodes, Capacitors

The outline of the components needed for our design are fairly simple. It's already been discussed in other sections that we needed to use a rectifier, capacitors, diodes, relays, and a linear regulator in our design. The point of this section is to go into the specifics of what type and what components will be used, as well as what size of components were used. In this section we explain all the possible options that could have been selected for our design along with why what was chosen was chosen.

Rectifier First let's discuss the rectifier used after the transformer. Not to explain how rectifiers work, rather to justify the choice of rectifier made for our design. Now there are two types of rectifiers; full wave and half wave rectifiers. For our design we used a full wave rectifier. Full wave rectifiers are far more efficient because they utilize the full cycle, plus they have less ripple with peaks occurring at twice the frequency of half wave rectifiers. Of the full wave rectifiers there are center tapped rectifiers and full-wave bridge rectifiers. Center tapped rectifiers use two less diodes and a transformer tapped at the center. Using less diodes is an advantage because of the reduction in voltage drop (meaning less power is wasted). The problem with the center tapped rectifier is that it requires a transformer of twice the size for the same voltage step down. In the end full-wave bridge rectifiers are the best choice for power supplies because transformer size is of high importance.

Capacitor Having decided what type of rectifier we want we still needed to determine the specifics on the components used for the rectifier. The two components under discussion were the diodes and the capacitors. The capacitor determined the voltage ripple. To calculate the ripple in full wave bridge rectifiers we used the following equation Vr = I/(2*f*C) for an approximation of the expected ripple (Millman-Halkias, pp 112–114). Where I is the current in the circuit, f is the frequency, and C is the capacitor value chosen. In the United States f=60 Hz for wiring used in the home. For the sake of our design we will assume "I" to be at approximately .5A. Knowing the values of I and f we controlled the amount of ripple by adjusting C. The real question became what is the acceptable amount of ripple in our circuit. Since the circuit goes through a DC to DC converter after this smoothing capacitor, there is actually no real need for the capacitor to smooth out the ripple since the DC to DC converter will ensure that a steady voltage is made at the output. Yet regardless we made our design such that 1V is the allowable amount of ripple. With an acceptable ripple of 1V the corresponding capacitor value of 4150 microfarad. With a capacitor of this size we ensured that the ripple would be less than 1V. Having a capacitor of this size is really quite unnecessary though, we would've done fine with using a smaller capacitor. Yet using a 4700 microfarads capacitor does the trick.

There are many types of capacitors. Yet the capacitor that we used was an electrolytic capacitor. These capacitors are highly used in rectifier circuits due to the fact that they

have a small size for their level of capacitance.[6] The capacitor that we will be using in our board will be a through hole capacitor. The main thing that we had to take into consideration was if the capacitor would be able to handle the voltage used for the power board. This stage in the circuit is right after the 14VAC has been rectified into DC. Meaning we will get a DC voltage equivalent to the 14/.707 or 19.8VDC. We had to make sure that the capacitor used was at least able to handle this level of voltage. Capacitors with this sort of requirement can easily be found on sites like mouser or digikey. The peak voltage depends on the input voltage and the type of diodes used in our circuit.

Diodes Now let's discuss the diodes used for the power board. There are many different types of diodes. For the rectifier their is no need for any type of special diode. The diode that we used for our rectifier was the 1N4001 diode. This diode is a simple rectifier diode. It is not as fast as other diodes such as a schottky diode that could have been used; however, there is no real need for a higher performing diode. Because of the simplicity and economy of this diode it is highly used, even though it is less efficient than other diodes. This diode is rated to be able to handle 1A of current and 50 V which is sufficient for our design. This diode however is not the only diode that we used. There is also a diode in the DC to DC converter. This diode however does have a much stricter requirement for diode speeds. In this case we were required to use a schottky diode. [7]

Relay Next let us discuss the relay. The requirements of the relays used in our power boards is that they must be able to handle switching an AC circuit with a DC driving circuit of 5V. This was the main requirement of the relay. Now when evaluating relays you run into one key design choice, solid state relay or electromechanical relay. Let's discuss the advantages and disadvantages of each. Electromechanical relays run with actual inductors that pull the switch by induction when voltage is applied across the terminals. Because the electromechanical relay uses these parts they tend to be much larger than the solid state relays. Solid state relays produce very little interference and they consume very little power. Solid state relays also have no arc meaning that they are very good for situations that require large amounts of isolation. A disadvantage of solid state relays is that they are more likely to overheat, and could require heat sinks. Electromechanical relays are better for applications where there is a potential of large current or voltage surges. Both could be used in our design, however for our design we decided to use a solid state relay rather than an electromechanical one. [8]

Linear Regulators There are two types of linear regulators standard regulators and LDO regulators. LDO stands for Low Drop Out. The difference between the two regulators is in the dropout voltage. Regulators with a higher dropout voltage have a larger required difference in size between the input and output voltages. The reason why LDO's are often talked about as more efficient is because they allow a closer input and output voltage, thus making them more efficient. If however you are using a drop out voltage large enough to use a standard linear regulator, using an LDO won't have much of an impact on efficiency. We would have needed a linear regulator in order to drop 5V down to 3.3V for use with our bluetooth module as well as with the nRF chip. In the end we ended up using a switching regulator instead of a linear regulator, but had we used linear regulators we would have considered the research above. [9]

Converting 5.5V to 3.3V is a voltage drop of 2.2V. For this application we could use either a standard regulator or a linear regulator as long as the dropout voltage is less than 2.2V. Other things that have to be taken into consideration are the maximum and minimum input voltage. For our design the maximum can't be above 5.5V and the minimum shouldn't be below 4.5V. The output current is also very important. For our design the current drawn from the nRF chip along with the bluetooth module is lower than 50mA. The linear regulator would have to be a through hole regulator since that is what is used in the power board. An acceptable part would be the texas instruments UA7M33. It gives an output of 3.3V takes a max voltage of 25V. Our input will always be around 5V. The output current of the device can be as large as 500mA, which is about 10 times what we need.

6.6.7 Isolation

There are a few different interpretations for what is meant by isolation when we are discussing electronics. Isolation can sometimes be used to describe the ability to couple two wires together without the use of wire connections. But for the sake of this paper we are discussing isolation as a means of safety. A proper definition of the word isolation would be the electrical or physical separation from an electrical circuit. Isolation is important not only to stop two wires from interfering with one another, but also to prevent dangerous human encounters with circuits. There are a few isolation design considerations we will make in our power board in order to keep safe operation. The first things are actually accounted for by using a design software to construct our boards. The built in Design Rule Checking (DRC) will ensure that our wires are not too closely placed together, thus ensuring we are isolating our wires correctly. Next we should discuss the isolation of our transformer. We must discuss isolation transformer and autotransformers. The main difference between an autotransformer and a isolation transformer is that in an isolation transformer there are two separate winding, while with an autotransformer the primary and secondary coils share their winding. Autotransformers may be smaller but they have less isolation. Isolation transformers keep the input ground and the output ground separate. Isolation transformers protect voltage spikes from crossing the primary coil to the secondary coil. For our design we will be using an isolation transformer in order to provide more isolation and therefore have a safer design.[10]

6.6.8 Simulation

In this section we show the preliminary testing of the systems that are part of the power circuit. This is accomplished using Multisim. Multisim is a highly capable simulation software that is freely available to engineering students at any of the computers in the engineering department at UCF. Simulation is an important step to take in project design because it is a very low risk method for checking if circuits work in the same manner that they are expected to work in. Although multisim may not be able to account for everything it prevents us from making silly mistakes at an early stage in our design process. Mistakes made while prototyping virtually are much better than mistakes discovered in actual physical prototyping or in our worse in our final project. Simulation may be an extra step, yet we expect it to save us time in the long run by allowing us to avoid mistakes that can be easily overlooked.

One of the things that is of interest in our design is the ripple. In multisim we can rebuild the circuit and test to see what ripple values we are returning. Using multisim allows us to reconfirm our assumptions made in our calculations. The circuit found in the Figure 29 below shows the expected ripple value with a drawn current of approximately 500mA. If we compare this to our calculations we find that the resulted ripple is a little larger than what we expected. What we expected was a ripple of approximately 1 volt, and what we got was a ripple about 1.6 volts. Now this could mean that there is an error in our calculation of the expected ripple or it could just be due to error. Regardless if we now have two sources that tell us that the ripple is below 2V, and since this voltage is still going to go through a DC to DC regulator this level of ripple is still acceptable.

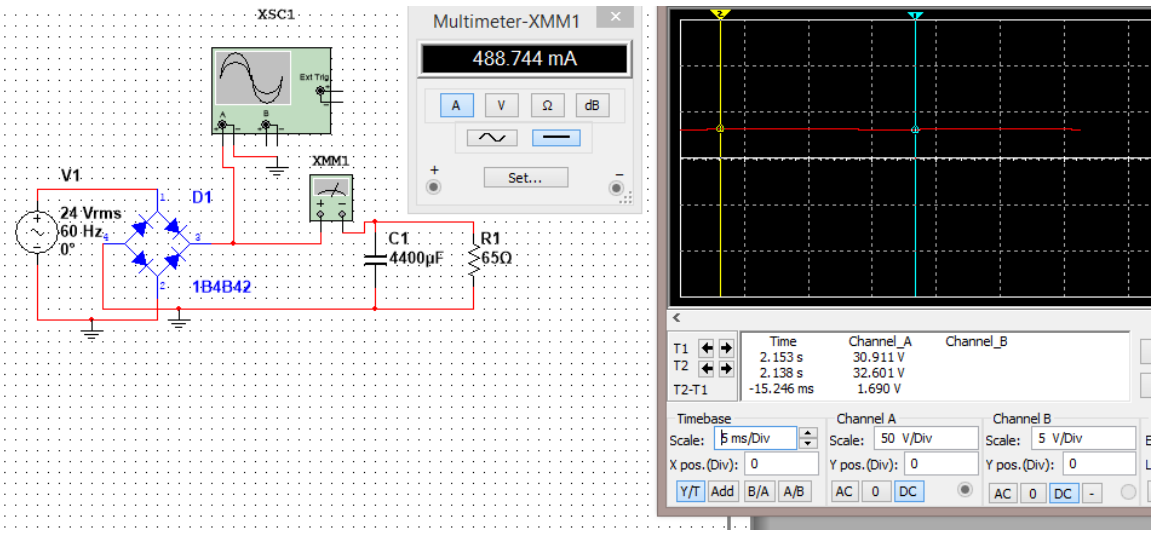


Figure 29: Multisim showing the expected ripple of our design

There are many other things that could also be tested in Multisim. One of the things that would have been nice to have tested would have been the integrity of the 5 V line after having gone through the buck regulator. Unfortunately Multisim is a little limited in part numbers and they didn't have the buck regulator we wanted to test the LM2576 regulator. In fact we had a hard time finding any switching regulator in multisim.

Regardless testing things first in multisim will give an extra way to test the design. In addition to avoiding mistakes made in the actual prototyping, it allows us to troubleshoot. If the prototype were to have not worked as it ought to, yet the simulation was working correctly, then we could test the current and voltage levels for the nodes in multisim and the nodes in our design and see how they compare.

6.6.9 Power Through Hole Board

Sending in PCB's to be built can be expensive. Also the size of the boards have a large impact on the cost. Due to the fact that components found in power designs are large and will therefore be costly to place on a PCB we initially wanted to self build a separate board for power. Not only is this an advantage because it is cheaper but it allows us keep high voltage AC isolated from the DC used for our control modules and the base station.

Additionally we initially thought each of our power boards were going to be slightly different from one another so making our own boards would have made it easy to customize each board for their specific needs.

After deciding that we were going to build a separate power board we had to decide how to build the board. The three candidates were: perf boards, vero boards, and etched PCB boards. Out of the three the team decided that an etched PCB would be the cleanest and would look more professional than an a perfboard or stripboard. Next we decided that a through hole board would be superior to a surface mounted board; mostly due to the fact that power boards can contain heavy components (for example transformers) and through hole boards are more mechanically sound than surface mounted boards. Later we realized that we could do a board that was both surface mount and through hole so we decided to make it through hole for large components and surface mount for everything else.

If we were to have built our own boards we would have used Rogers RO4003. The primary reason we decided to use Rogers was because they provide free samples for learning purposes through their University Sample Orders Program. Since this is in fact a non commercial academic project we were able to order a sufficient amount of laminate material to complete our project. RO4003 is a double sided laminate typically used for high frequency applications. Building this board from scratch will give added experience to the members of the group, which is the ultimate goal of this project. In the end we decided not to build our own boards for a number of reasons. One it was well within our budget to have the boards made and it was also a lot simpler of a process.

6.6.10 Schematic Breakdown

This section breaks down the full schematic of our power board (full schematic is available in Appendix A). Although there is a difference on what is populated for each board, below is a design that has all the components of all the boards. For our boards we only attach the components needed per application. As you can see from Figure 30, our power board has a simple textbook AC to DC transformation.

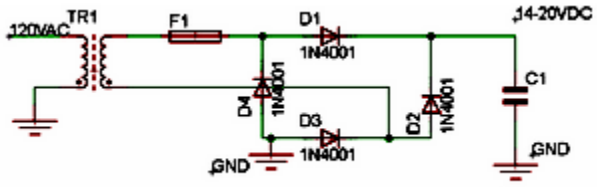


Figure 30: Power board AC transformation

Our base station and control modules require a mix of 5V, 3.3V, and 12V (for the electronic strike). You can see the regulators and extra components required to create these voltages in Figure 31. These switching regulator were designed with texas instruments WEBENCH.

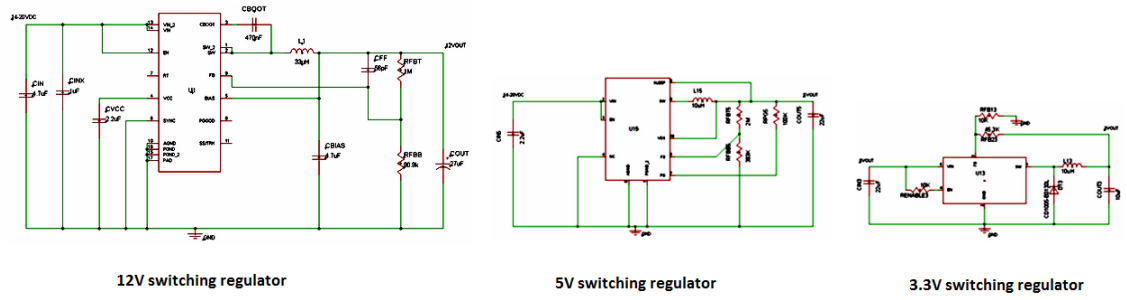


Figure 31: Power board switching regulators

6.6.11 Board Layout

After designing the PCB to the layout specification, the front and back images below are the result after sending the Gerbers to OSH park.

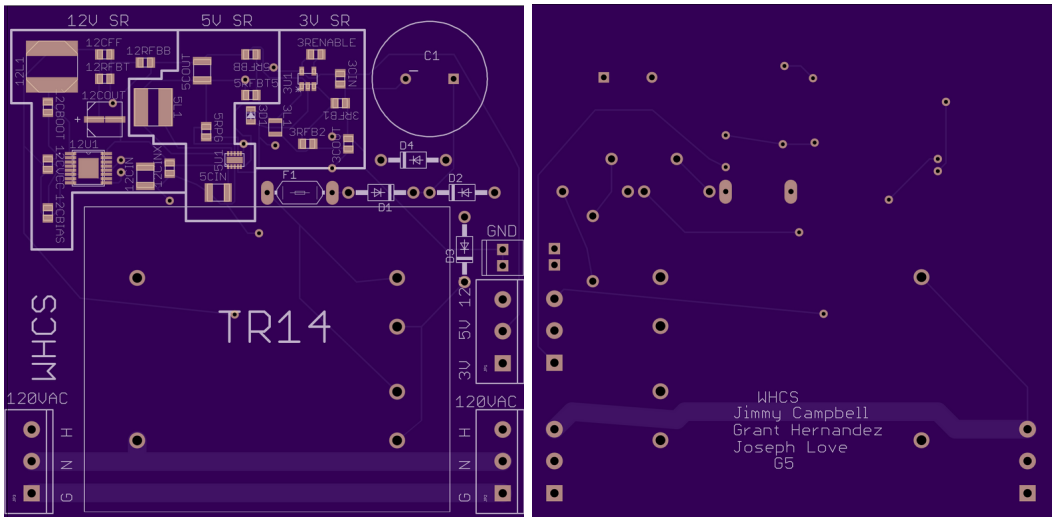


Figure 32: Power board OSH park PCB layout

6.7 Base Station

The heart of WHCS resides with the base station (BS). When users think of WHCS, they will think of the base station as it is the most visible part of the system. The base station is responsible for managing, collecting, and displaying information from all of the control modules. If the BS were to fail, WHCS would cease to function.

When a new control module is introduced in to the system, it must first pair with the BS, which will authenticate it on to the network. Once it joins, the control module can be abstracted as an "API" meaning, the specific hardware details do not have to be considered. This abstraction is excellent in keeping the system clean from one-off cases and allow for a Domain Specific API to be formed.

6.7.1 Software Flow

The main software flow for the base station is the most complicated in WHCS. It has to manage three separate devices simulatenously and be able to service each one in a timely manner. The LCD, NRF radio, and Bluetooth module are all being controlled and commanded by one ATMega32-A chip. There isn't much room for busy waiting or any expensive operations as everything has to be running as fast as possible. Given this, the BS is the least point of failure for the WHCS.

In Figure 33, we have constructed the general architecure of the main loop for the BS, including some early initialization. Most of the implementation details are left out as they are very specific to the final drivers.

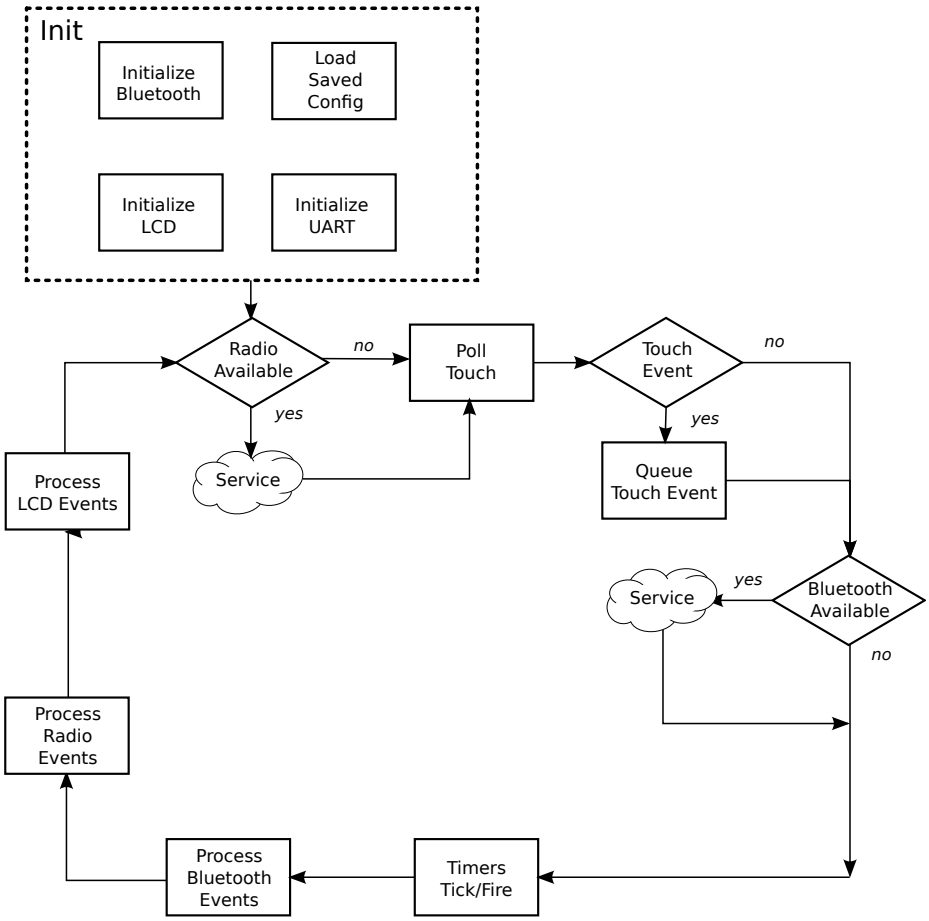


Figure 33: the high level software flow for the base station

The large amount of tasks that are required of the BS become clear when viewing Figure 33.

The base station from reset first loads any saved settings from the EEPROM (saved control modules, behavior settings, LCD settings, etc.), then it brings up all of the main modules (radio, LCD, BlueTooth, and UART). If any of these initialization steps were to fail, the BS would indicate through an LED or from a UART debug message. An initialization failure shouldn't be handled as it's a critical failure of the system along with its assumptions. We

believe that by assuring that the initialization sequences for all modules is well defined, we will be able to diagnose hardware issues quickly and without strenuous debugging. One of the issues we found while prototyping is that it's difficult to determine if the issue with module initialization is with the wiring or code. Having a golden model for code that is know to work on our target setup would allow us to have a good working state.

Each module has a unique sequence of "commands" with parameters that are required to configure the device. The NRF radio has to have its power, channel, payload length, and other parameters set before usage. Once these basic options, any further configuration is done at run time. This includes switching from listening to transmitting mode and enabling or disabling the automatic acknowledgement feature. The built-in Atmel UART only needs to know the baud rate at which data will be sent and received. There are also other options relating to other bits used (parody, stop, etc.). In our case, we are using the default 8 bit data, 1 stop bit, which is simple enough for our needs. The LCD happens to have the most extensive initialization sequence due to required screen configuration, gamma settings, pixel order and other more archaic options.

Once all of the modules are brought up correctly, the BS begins the main loop. Radio events, BlueTooth events, and LCD state are all be processed as required. For both the BlueTooth and radio, new packets are checked for and serviced as needed. From these packets, internal state is updated and any response packets are generated and sent. Internally, WHCS has an internal event queue with different event types. These events are be processed and any responses are generated, if any. These responses include a confirmation packet over the NRF radio, or a status update through BlueTooth. The base station may also initialize actions despite not receiving events.

In terms of the NRF radio, the module we have chosen exports an interrupt pin which goes low on the reception of a new packet. This feature is great for the WHCS architecture as the radio requires a significant amount of power while actively receiving SPI commands and transmitting. By avoiding the polling of the NRF, the main loops for both the control module and base station have more cycles to process more important and intensive tasks. Essentially, the NRF is kept in the listening mode at all times, waiting for a packet from a control module to be received.

The BlueTooth module won't require as much time to service as the other modules due to the low data rate. It is connected directly to the hardware UART of the base station, instead of the normal serial debugger. This is connected to some free GPIO ports and software serial is be used. This is unfortunate as "bit banging" serial isn't efficient and takes up many more cycles than just using the hardware UART. Packets over BlueTooth are be sent asynchronously from the main loop due to the hardware UART. If the micro loops fast enough, then it is as if there is no delay in transmission. Due to the real time nature of the NRF radio and LCD, WHCS takes steps to avoid blocking too long on a single activity. Blocking too much lowers the overall performance and responsiveness of the system. At worst, touch events could be lost and packet buffers could overflow. An effort has been made to profile the main loop's average time using hardware timers. This was an invaluable statistic when developing the base station code.

One of the most important parts of WHCS is its usage of timers. A global list of timers is constantly maintained and updated to schedule events in the future, instead of having to handle them immediately. The granularity of the timers depends on the average execution

time of the main loop along with how many timers are processed at once. If the main loop is slow on average, then timers have to wait for extended periods to be serviced (starvation).

All of the above tasks are be executing in the same way as single core CPU would: in pseudo-parallel. The faster the whole system runs, the better the appearance of everything executing at once.

6.7.2 Control Module Abstraction

For WHCS to function smoothly and scale well, a neat and abstracted interface must be defined to accept *any* type of control module. New control module types are easily added to the system without affecting older types and there is be a set of generic data structures for managing and storing information on modules. These structures care carefully defined to wrap more specific control module packets in all of the shared metadata. Think of it like a hierarchy where all of the common attributes and actions shared by control modules have packets that can be sent to any module. Whereas the more specific packets (get temperature, engage door, etc.) is be wrapped up in the generic ones (essentially a derived object from the generic control module.) This can be visualized in Figure 34. The details of a network structure that enables this clean interface is further described in Section 6.1.4.

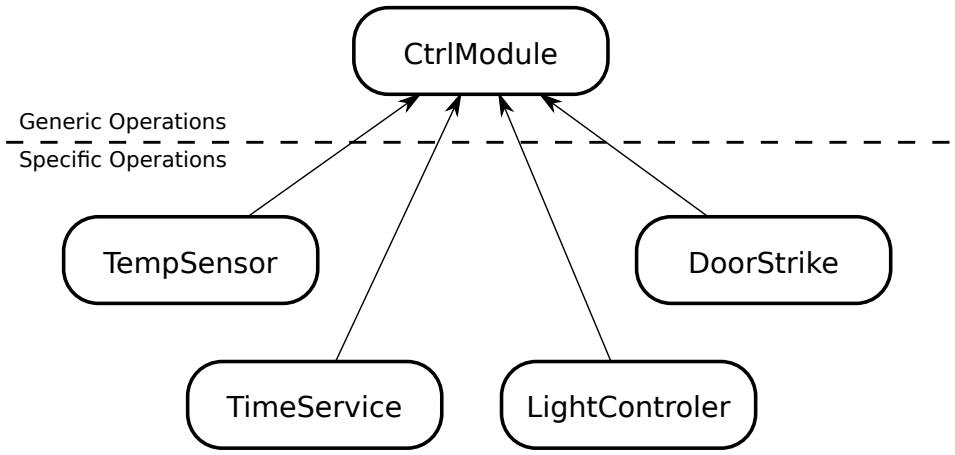


Figure 34: showing the control module hierarchy for WHCS

Beyond sending packets, the base station must accurately record and update state for all of the control modules. Depending on the control module, additional state is needed to be stored and functions were written to query that state. Each control module has its own state machine that controls its function in relation to the base station.

6.7.3 Subsystems

The base station has the hardest job in the entire WHCS architecture. It has to juggle a lot of data with limited memory and processing speed. Packets are processed and queued to keep the pipeline flowing smoothly. The following sections break down the individual subsystems and how they work in concert to make the base station a well oiled microcontroller.

NRF24L01+ The NRF radio is directly connected to the Atmega32-A microcontroller which controls its state. This module is be constantly listening for new packets from the control modules and sending responses in turn. Due to this link being the most critical for WHCS, it had the most attention to detail when constructing the layout and software design. Also, besides the expensive drawing operations for the LCD, this takes up the most CPU time to process packets and perform data transfers. This is partly due to the slow SPI interface that is used to transfer the data. the NRF24LO1+ breakout board does not offer any other options for transferring data to and from a microcontroller. The overall system speed is limited by the maximum transfer speed of this radio (2 MBPS). When the considerations for the WHCS topology were brought forward, it was obvious that the organization relies solely on the speed of the base station. We considered this fact throughout the system design - for example, the LCD could have *also* been controlled over SPI, but we made a trade-off for pin count in order to use the parallel interface. This freed up the SPI bus for programming (infrequent) and the NRF. Essentially, the NRF has full control over the SPI bus and receives the full attention of the microcontroller.

Assuming the NRF is not limited by the transfer rate of the SPI bus, the additional considerations can now be focused on. The microcontroller needed a way to quickly assess the state of the NRF chip. This state query could include the status of the radio's transfer queues - meaning if there is a packet to be received or if a packet has been successfully sent. This is something that is checked very frequently, as the radio usage will increase linearly with the amount of control modules that are a part of WHCS. If any algorithms or tight loops in the microcontroller were to **not** have linear performance, then the system speed would suffer as more control modules were brought in to the network. Luckily most operations were $O(n)$. We focused on WHCS' scalability for more complex households than a small demo setup. Of course, once the network becomes too crowded, the somewhat weak Atmega32-A will no longer be able to sustain the flood of packets that are required to maintain WHCS. At this point, more base stations would need to be operating simultaneously in order to handle the load. Our implementation of WHCS briefly considered this fact, but due to the prototype nature of this endeavour, we left these more complicated details to another, more scalable revision of WHCS.

In WHCS' architecture, the main loop of the base station will not be interacting directly with the radio. This is due to the abstraction we plan on building around the low-level radio driver. All driver specific functions will be wrapped in to a façade pattern, network library. This would allow WHCS to swap out the underlying network hardware for another, similar radio. This would encourage code reuse and prevent massive, expensive rewrites of the net code. The high-level interface that the BS will know about will be sufficient and feature rich enough to carry out all of the actions required for WHCS to come to fruition.

HC-05 The HC-05 BlueTooth module is quite simple in its operations. Data is sent over a two line serial bus and if there is an active connection to a bluetooth enabled device, it is able to easily receive the data and handle it. In this case the device on the other end is expected to be a phone, but not limited to one. As long as the device on the other end of the BlueTooth link follows the WHCS BlueTooth application protocol, then WHCS is able to receive commands from arbitrary devices. Assuming that the only thing that will connected to WHCS is a phone, then a suitable protocol for querying and changing WHCS' state has been derived. This set of functions is important for more than just the BlueTooth

link as it could scale to many different consumers and producers of commands. A simple diagram showing a very high level interaction of a BT device with the base station is for reference in Figure 35. Once again this underlying protocol is abstracted away from the underlying hardware.

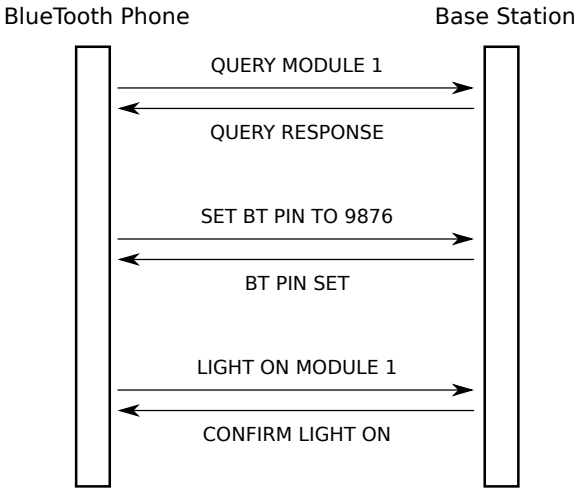


Figure 35: an example sequence diagram that could occur between a connected bluetooth phone and the base station

In regards to the application level protocol for WHCS, there is a well defined, easy way, for the BlueTooth library to gather information from WHCS' state. This can be handled on the top-level flow of the base station by gluing together two different libraries without them knowing about each other. This is a good approach because it decouples the two modules from each other, making their individual implementations separate. Two tightly coupled modules may start to take on the appearance of a "ball of mud." A connected phone will be able to accomplish any task that manually interacting with the LCD could handle. This includes controlling the function of individual modules and querying their current state. The BT connection try to avoid generating too many packets over the NRF radio in response to user events. Instead it merely looks up the cached state from the base station's memory. This is faster and the round trip time is lower.

LCD In what could be considered the "face of WHCS", the LCD module, which is situated directly over the Atmega32-A, has the tough job of accurately and quickly conveying any desired information about the state of WHCS' control modules. This is no simple task as not only does it have to display, but with an attached touchpanel, it reacts to user touches. What functionality is exported to the LCD is only limited by the underlying processor speed and the UI library. The high level design of the WHCS LCD only had to worry about what the end goals are for its usage. An example of this abstraction may be viewed in Figure 36.

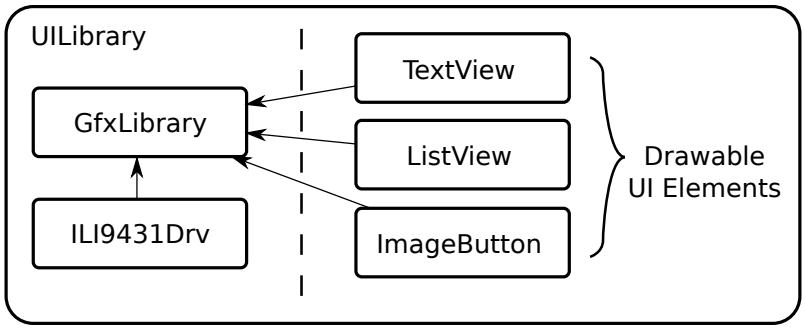


Figure 36: the level of abstractions for the LCD subsystem

As the state of the WHCS network changes, the base station has to fire off redraw events in order to keep the LCD up-to-date. These redraws sync the internal state of WHCS with the user viewable interface. The physical connections to the LCD consist of data signaling and an 8-bit wide parallel data bus. There is an optional reset pin that WHCS uses for emergency resets and debugging. The high level interface with the LCD occurs directly with the high-level UI library and if necessary the underlying graphics library. The base station never uses the direct driver interface, if it can avoid it, as this is subject to change in a new revision. In addition, a subtle feature that WHCS choose to implement was dynamic power saving through screen dimming. Although we assumed the base station would have wall power at all times, there may come a time where the system may migrate over to a lower wattage current source, such as power stealing from an HVAC unit. In this case, the system would most certainly have to be power efficient. Despite not needing to worry about power, this function would be simple to implement as only one microcontroller pin is required to control the screen brightness.

Touchpanel In order to provide a way for an end user to be able to control WHCS from the LCD unit, there is a requirement to poll for touch events. This subsystem can be considered a part of the LCD, but the driver is independent from the graphics and ILI9431 drivers. These events are dispatched to the appropriate UI element based on the X and Y position of the touch event. There is also an optional Z "position" which represents the pressure of the touch event. This is used to gather more fine grained information about the touch itself. One of the unfortunate properties of the touchpanel is that it must be actively polled for new touches. This requires that the ADC be constantly providing conversions, which raises the dynamic power of the MCU. This isn't a major concern as the base station is expected to have power from the wall most of the time.

The extent of the touchpanel interaction occurs from a `getTouch()` method. This method returns the latest touch event, if any. The base station has full control over where this touch event is dispatched to. Depending on the LCD scene (i.e main menu, boot screen), this event will be handled in different ways. Further details are discussed in Section 6.4.5 for the UI library.

Timers Accurate timing is one of the tasks microcontrollers excel at. The WHCS base station is using timers to schedule periodic events, wake up from sleep modes, to provide automated send and resend delays, and much more. A simple Timer class serves as the

mechanism and state for a single Timer object. These objects are be kept track of in a global list and ticked every loop. When a timer has expired (now() - start ¿ timeToWait), then it will be fired. This firing may cause an action to occur, which is programmable to the specific need of the Timer client. This action could be something like scheduling a network health check or to increase the step of an LCD graphics animation. A UML representation of the WHCS timer class is shown in Figure 37.

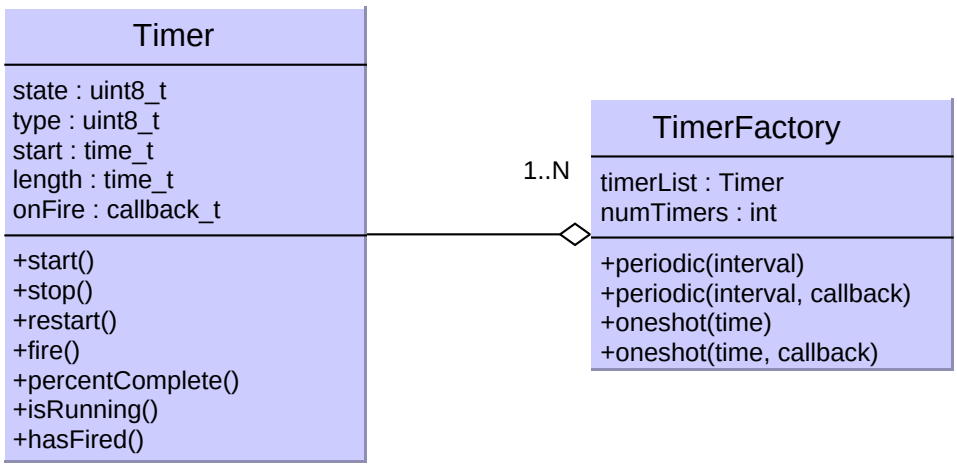


Figure 37: A UML representation of the Timer class

6.7.4 Schematic Breakdown

To tie the whole design of the base station together, the schematic, created in KiCad is broken down below. The full schematic is available for viewing in Appendix A.

In Figure 38 we see a focused view of the Atmega32-A microcontroller with an attached 16MHz crystal and power passives. The crystal has two capacitors that are dependent on the target crystal. These are required to get the correct oscillation for the external crystal. Also the AREF, VCC, and AVCC lines of the MCU have decoupling capacitors. These are used to make sure that the base station performs well under a large current spike. When designing the board, these capacitors are be placed as close as possible to the MCU to avoid a long high-current path through the ground plane. The capacitor on the analog reference pin (AREF) is used to stabilize the reference to make ADC conversions more accurate.

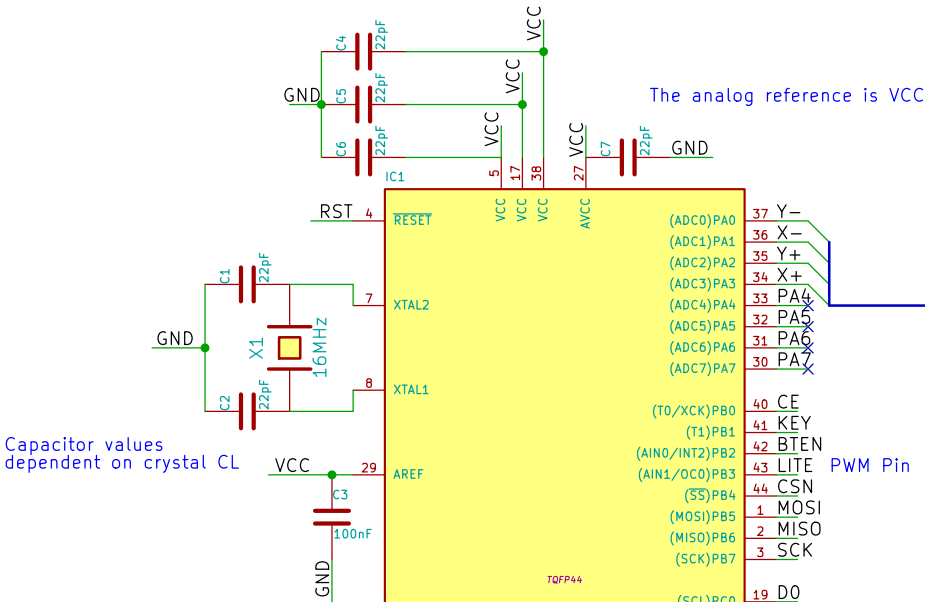


Figure 38: Base Station crystal and decoupling capacitors

In Figure 39 we see the buses used to connect the LCD to the MCU. This is the most pin heavy component and care must be taken not to mix up any of the signal paths. All of the data control signals are connected to PORTD of the MCU, the touchpanel signals to PORTA (ADC), and the 8-bit parallel data bus completely uses PORTC. There are a few one off signals such as LITE which is a PWM input to control the LCD's backlight brightness.

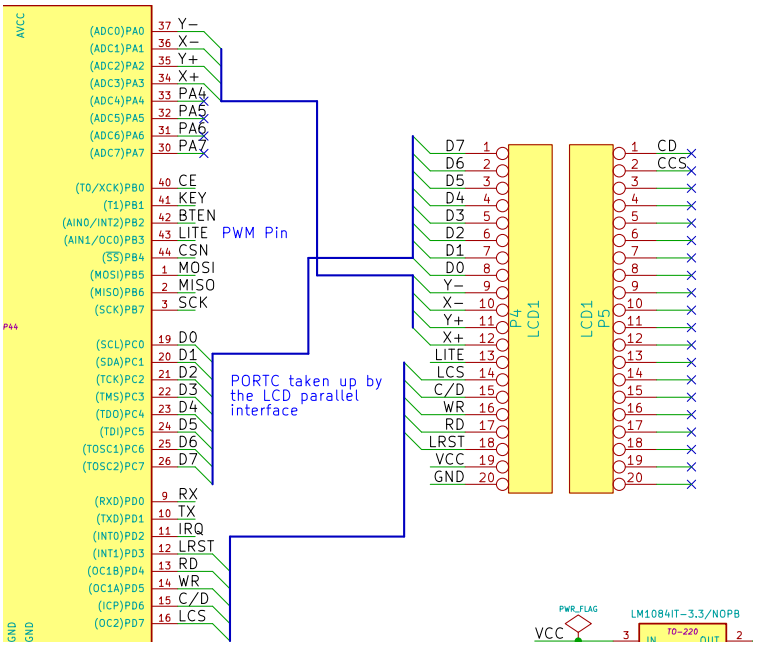


Figure 39: Base Station LCD header to MCU

In Figure 40 we see the 3.3V 3 terminal regulator converting the 5V VCC line down. Additionally, we see the external RESET pull-up resistor and a manual reset push button. The left corner has the pinout for the NRF breakout board we are using.

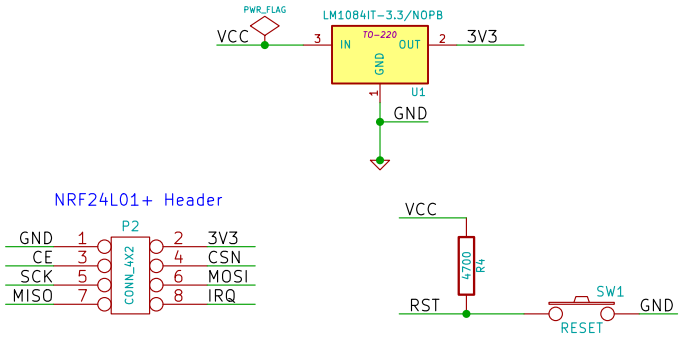


Figure 40: Base Station power schematic and NRF header

In Figure 41 we see the header for the HC-05. We examine this further because of the unique electrical characteristics of the HC-05 module. The module only accepts 3.3V power and logic. Our MCU is running at 5V, which means we need a 5V to 3.3V logic shifter. To simply implement this, we used a resistor voltage divider which provides the required logic level for the TX pin. The RX pin does not need a shifter because 3.3V is still above the V_{IH} minimum for the MCU.

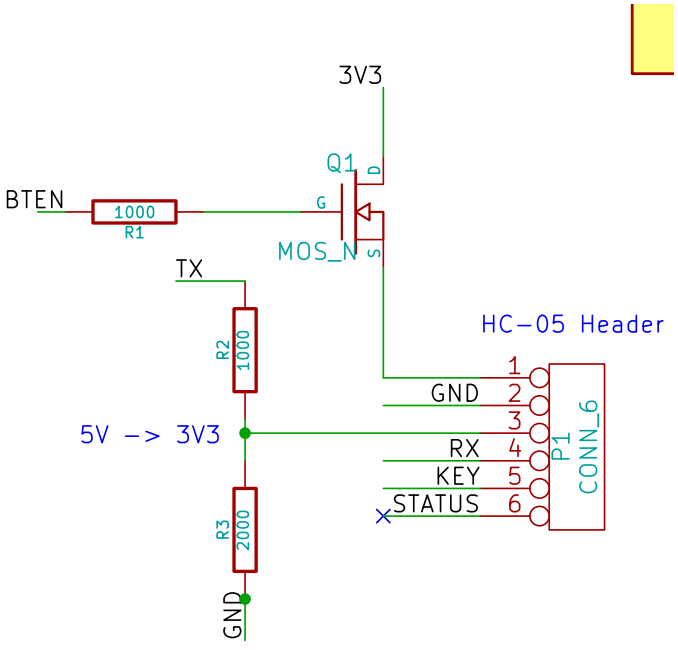


Figure 41: Base Station HC-05 header

Finally, in Figure 42 we see the standard ICSP header that most AVR line microcontrollers use. This pin array serves as a quick and easy way to connect an external programmer to our base station while in the field. In this schematic revision, this header can provide power directly to the 3.3V regulator and MCU.

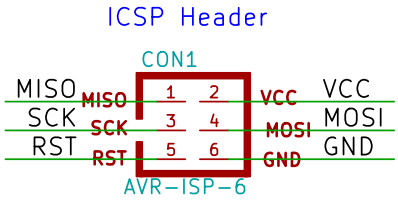


Figure 42: Base Station ISP header

6.7.5 Board Layout

After designing the PCB to the layout specification, the front and back images below are the result after sending the Gerbers to OSH park.

Figure 43: Base station OSH park PCB layout

6.8 Control Module

What can be thought of as the "arms of WHCS", the control modules serve as the main devices that seed the network with data. This data is specific to the control module that is emitting it. The base station is more complex than an individual control module because it needs to be[2] . The control modules should be as lightweight as possible to save cost and keep power usage down. If the control modules were too complex, then the entire cost of WHCS would increase proportionally to the number of control modules.

[2]Seriously, it really needs to be. Take our word for it.

6.8.1 Software Flow

The general flow for the control modules is much simpler than the base station just due to the requirements of the system. There isn't as much that needs to be done on each loop iteration. The only main module that the control module needs to work with is the NRF radio. This can be seen in Figure 44. Due to the capabilities for the NRF radio to provide an interrupt signal on the reception of a packet, the control module actually has the ability to sleep when not doing anything. Control modules should be as mobile as possible, which limits their overall functionality and processing power. Without these limits, any battery attached would quickly be drained.

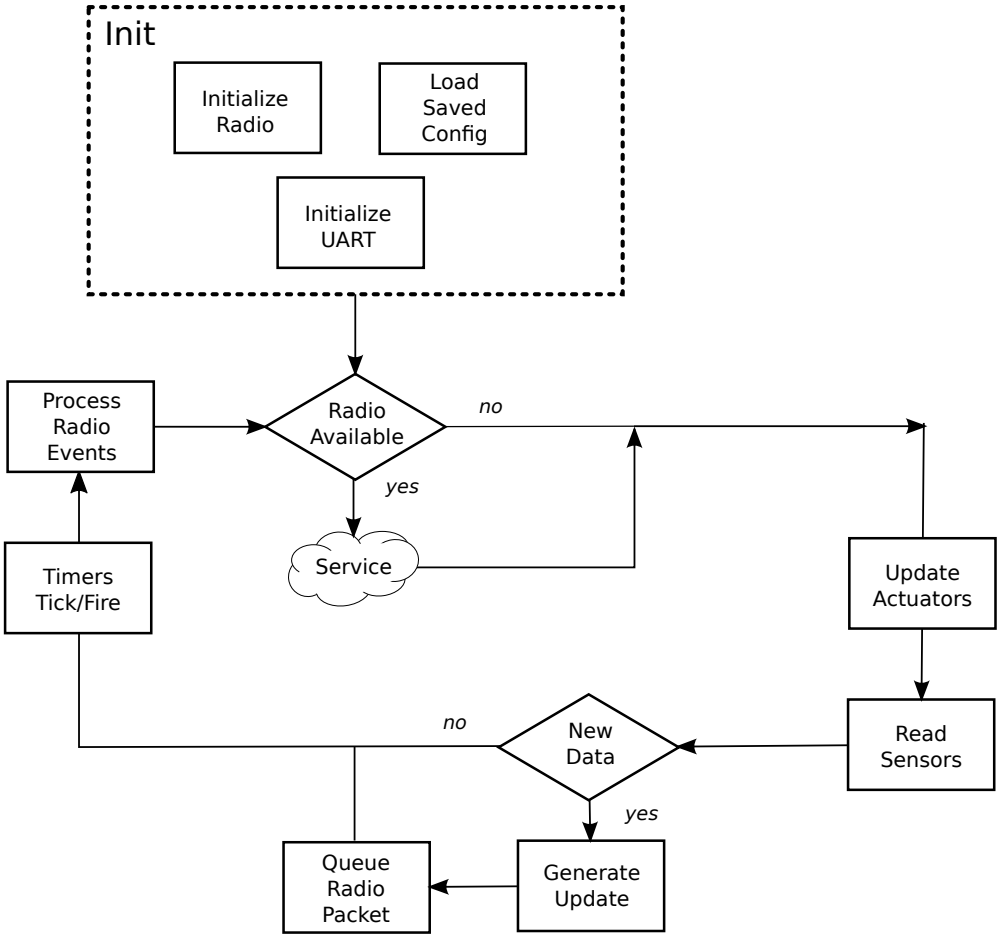


Figure 44: the high level software flow for a generic control module

Beyond scheduling packets and receving responses, the sensor or actuator that the control module is responsible needs to be serviced. The rate at which the sensor is polled is depending on the required data rate. For instance, if this control module is controlling the door, then it needs to be actively listening for a packet to open or close it. This is quite different from the requirements of a control module that is gathering temperature data. Like the base station, the control module will also have a global list of timers that will govern the timing of events and actions to be taken. The timer list can be associated to any thing that is required to run in the future or periodically.

Events spawned from receiving packets from the base station will change the internal state of the control modules. This state is going to be transitioned between using a specific set of functions that act as an API to the base station. This API will extend across the network layer through a network protocol that will enable the control and querying of the control module's state from remote locations. It is important that this is done right in and in a sustainable manner in order to have clean API for usage across the entire network. With out a clean and well thought API, each control module will have duplicate functionality to handle common tasks. This has already been in further detail in the base station and network library sections.

The mobile nature of WHCS' control modules means that they need to be aware of their power state. We want our modules to consume as little power as possible in order to sustain an isolated power outage. Also, some modules may opt to be battery only if their power requirements are low enough. For instance, a temperature module may be a low enough power consumption that it could last on a battery long enough to be feasible. This requires further field testing and research in order to prove right or wrong. Regardless of the end goals for WHCS control modules, the state of the battery for each control module should be known to the base station for analytics and tracking.

6.8.2 Electronic Strike

For WHCS we knew that we wanted access control to be part of our design. Basically a way for the user to unlock and lock the doors from their smart device. We explored a number of different options for what kind of locking and unlocking mechanism we would use. First we considered servo motors. The advantage of using servo motors is that they allow for very precise control. The design would be fairly simple the rotating motor would slide a deadbolt that could lock and unlock the door. Figure 45 shows a simple graphic of how such a system would operate.

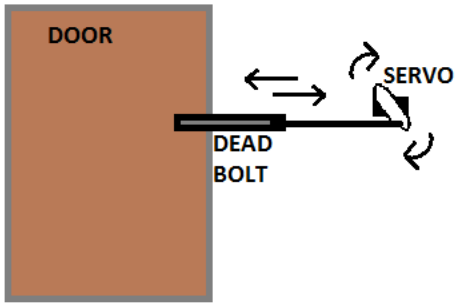


Figure 45: servo motor access control design

The fact that the servo motors allow us to control exactly how much rotate is an advantage for two reasons. One once the servo is done rotating we know exactly what position the lock is in, and two we know for sure that the lock will stay in that position. The other design consideration was using a solenoid to move a deadbolt lock. When activated the solenoid would push or pull the deadbolt into the locked or unlocked position. While this

would work it's design is a little more complicated than the servo motors. Unfortunately the solenoid design would require some sort of locking mechanism once the deadbolt is fully extended. Plus it would be hard to measure whether or not the door really did get locked or not.

While both of these methods would have worked, in the end we decided that these designs were too mechanically involved and we wanted to focus our efforts on electrical and computer engineering designs. The alternative was to use a premanufactured electric strike. In the following sections we will discuss what design considerations were taken when selecting our strike.

Normally Open or Normally Closed The first thing we considered was whether we wanted a normally open lock or a normally closed lock. Normally opened means that the door requires power to be unlocked and is otherwise locked without power, while normally closed means that the door needs power in order to be locked and is only unlocked when the power is shut off. It was pretty easy for our group to decide that normally open was the better design choice because it would allow the door to be locked most of the time without wasting power. The only issue we saw initially was that it might be a potential safety hazard to have doors locked while there is a power failure. In the case of an emergency this could be a huge problem. We did however find an easy way to have a mechanical alternative (this will be discussed in the next Section 6.8.2) so that the safety hazard was no longer present.

Strike vs Deadbolt There are two main types of electric locks to choose from, electric strikes and deadbolts. While some may argue that deadbolts are more secure, electric strikes have the advantage that they can be used with a regular door knob. This is an advantage because it allows us to include a door knob with a mechanical lock. That way if there is ever a power failure the mechanical lock can still be used. This gets rid of the safety hazard that could arise if for example a fire where to occur. It also allows for a backup system in case you were to lose your phone or if there were some sort of failure in the electronics that give the command to unlock the door. While perhaps there is a higher level of security that could result from using a deadbolt, the advantage that comes from using an electric strike outweighs the benefit of an electric deadbolt. Figure 46 shows the two methods for opening the door when an electric strike and a door knob with a lock is used.

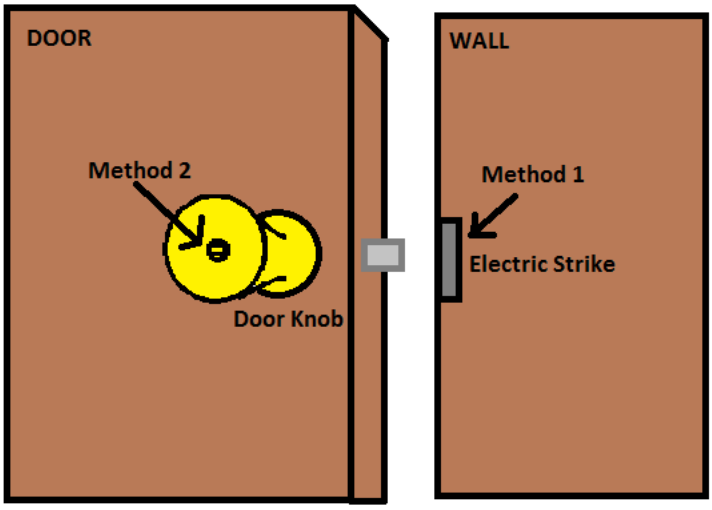


Figure 46: electric strike entry methods

6.8.3 Sensor Data Collection

The control modules of WHCS are not only used for switching things on and off. They also provide a foundation for sensing data around the house. For example any sensor that can turn physical data into an analog or digital signal can be connected to a control module and then be a part of WHCS. Examples of these would be temperature sensors and humidity sensors. For our project's prototype we used a temperature sensor to demonstrate WHCS's data collection. With the way the circuit was designed only the sensor chip would need to be removed and any other sensor would be able to be hooked up.

The temperature sensor that we decided to use for WHCS was the TMP36. The TMP36 is widely available and comes in through hole and smd packages. The temperature sensor is simple in design as it has only three pins that require connection. The schematic shown in Figure 47 shows how the temperature sensor is connected to the ATmega328 on the control modules. The VOUT pin of the TMP36 outputs a voltage signal that varies based on the temperature surrounding the component. The voltage range is between 2.7 to 5.5 volts which is suppliable through or logic level voltage lines. This model of temperature sensor is capable of sensing temperatures within the range of -40 to 125 degrees Celsius. This covers all the temperatures that one would encounter in a home and more. Connecting to the analog sensor required the use of one of the ATmega328's ADC (Analog to Digital Conversion) pins for converting the analog signal to a digital signal. There were plenty of ADC pins available on the microcontroller and even when one pin on the ADC port was connected for conversion the other pins could still be used as GPIO pins. The schematic shows the output of the temperature sensor going to pin ACD0 but any ADC pin would be able to accomplish converting the signal to digital.

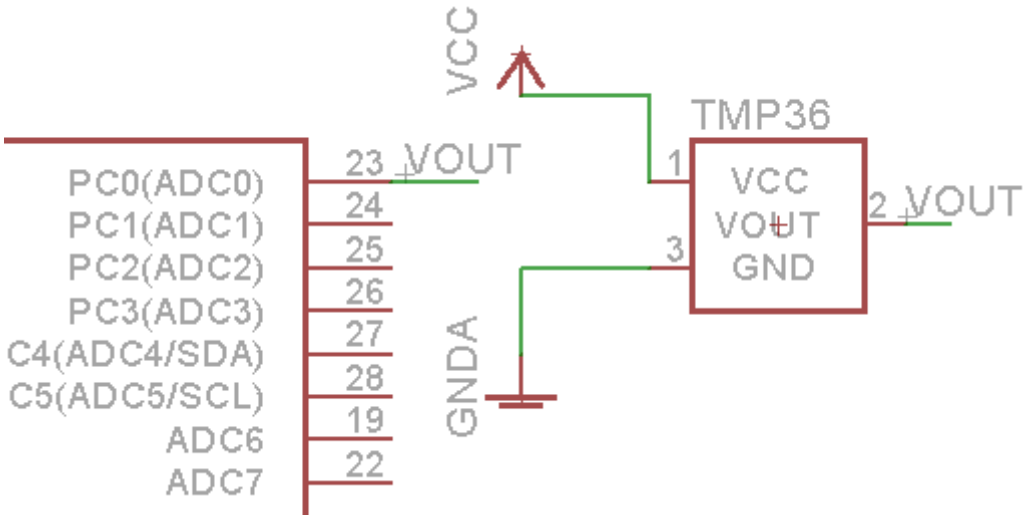


Figure 47: Temperature Sensor Connection Schematic

Utilizing the ADC pins required working with special registers inside the microcontroller. The ADMUX register allows for selecting which pin out of the 8 ADC pins should be connected to the internal hardware for doing the conversion. This register also holds bits necessary for setting up the reference values used in the conversion. The reference voltage in the conversion was important because the digital value that was obtained was a result of where that voltage level fell in between ground and the reference voltage that was supplied. For our design we used the power supply of the microcontroller as the reference for the ADC which meant setting bits 7 and 6 of the microcontroller to 0. The ADCSRA register was also very important because this is where we set the prescaler that divided the clock rate of the chip into the frequency that we wanted to use for ADC. We are ran our microcontrollers at 16 MHz in WHCS which is too fast for good accuracy in the ADC. We set the prescaler in the ADCSRA register to 128 in order to achieve an ADC sampling rate that provided reliable accuracy.

6.8.4 Light and Outlet Control

The control modules of WHCS are capable of controlling lights and outlets around the home. To be able to support this functionality the control modules needed to be able to switch 120V AC circuits. A simple circuit with a transistor was not capable of switching such high voltages using the logic voltage levels of microcontrollers. In order to switch such high voltages we used relays. There are two different forms of relays, mechanical and solid state. For our purposes we investigated both to see which would be the best option. We originally considered mechanical relays in our design. For mechanical relays an actual arm inside the device is being physically moved due to the current going through the device. A side effect of this mechanical motion is that the current and voltage requirements to drive the relay are relatively high. Our prototyping of the system originally used a mechanical relay but the relay required 9V input to activate the large 120V load voltage. Mechanical relays are cheap because they are made out of simple components, and low cost is always a big factor. The biggest down side to the mechanical relay is that using one would limit us to not being able to directly activate the relay from our microcontroller. The required

current and voltage to activate a mechanical relay is too high for any of our microcontrollers to supply.

We wanted to be able to control the relay that is switching 120V AC directly from the control module microcontroller's GPIO pins. We concluded that this would not be achievable through the use of mechanical relays. A mechanical relay would require extra transistors to switch another voltage source to the relay. Solid state relays solved the problem of supply voltage and current which is posed by mechanical relays. Solid state relays do not move any physical components to switch a circuit, instead they operate through semiconductors. As a result they require much less input to complete the circuit for the load voltage. One tradeoff for solid state relays is that they are generally more expensive than their mechanical counterparts. In this situation the price jump was not large enough dissuade us from utilizing SSRs (Solid State Relays) instead of mechanical relays. If we decided to use mechanical instead of solid state we would not have gotten the behavior that we desired in our circuit so the decision is easy to make.

Once we narrowed down our solution to controlling lights and outlets through the use of a SSR, we searched for a chip that had the electrical characteristics we were searching for. The GPIO pins of the ATmega328 that we are using for the control modules are capable of outputting a maximum of 40mA DC current. We needed a solid state relay that had a tolerance for 120V AC as a load voltage, could be supplied by 5V, and needed less than 40mA for activation/forward current. We found a solid state relay made by Sharp Microelectronics with the part number S116T02F that met all the requirements that we set. We confirmed that the chip is in stock and suppliable by digikey. The chip is available at an affordable price of $5.10. Figure 48 Shows a schematic using this solid state relay. The schematic for the control module mimics the one shown in this figure. The activation input is directly connected to the microcontroller's GPIO pin in order to toggle on and off the state of the relay. This particular relay has an activation voltage of 1.2V DC which means that when the relay is on this is the voltage across the diode shown in the schematic. Thus the forward current for this schematic can be calculated through the equation (5V-1.2V)/R1. The SSR in the figure requires at least 15mA for activation. The microcontroller's pins are adequate for supplying this low current. When the microcontroller's activation pin, pin PB1 in the figure, is set to high, the 120V AC is free to flow through the triac of the SSR and power the component in place of the load resistor. In WHCS the load resistor RL in the figure is replaced with an outlet or a light. Whenever the microcontroller pin goes high, the light or outlet will receive the power it normally would from the household main.

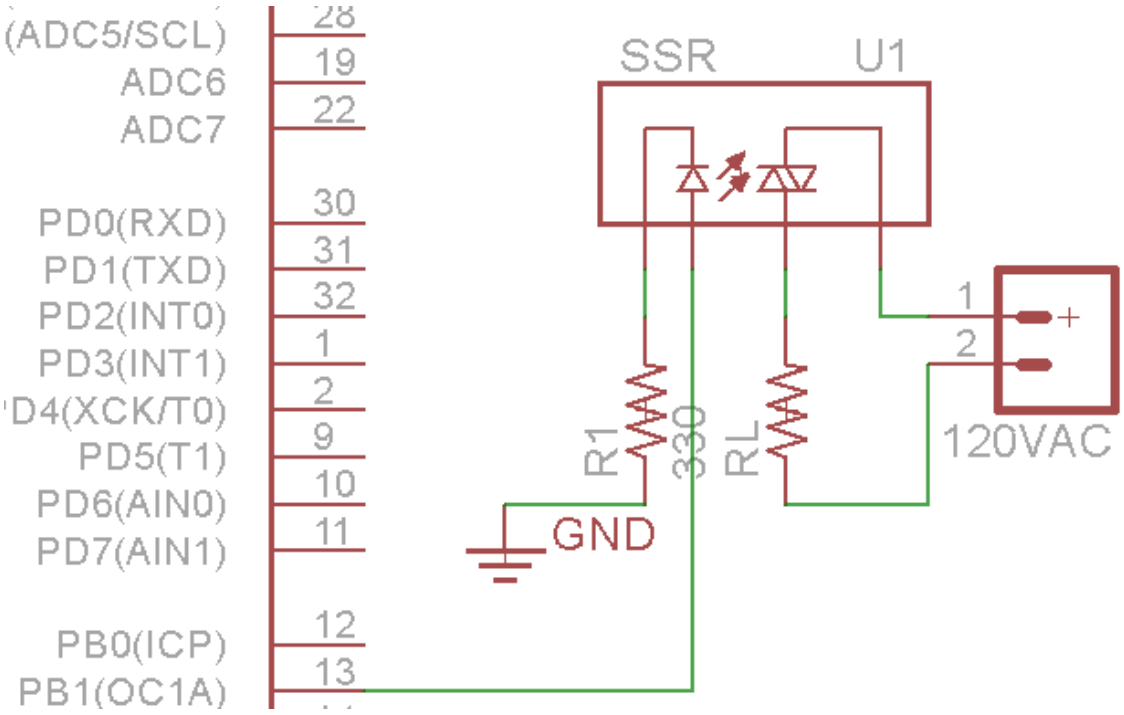


Figure 48: Wiring Schematic for Solid State Relay

6.8.5 Schematic Breakdown

The control modules for WHCS had to be able to support communication via a radio transceiver as well as interaction with their target endpoints. Figure 49 shows the schematic for the control modules that was implemented in WHCS. The full schematic is available for viewing in Appendix A. The main component of the schematic is the ATmega328. Everything in the schematic is connected to the microcontroller in some way. In the schematic three different VCC lines are shown. This is because the control modules have to access to a 3.3V line, a 5V line, and a 12V line. The power board supplies these power lines to the control module. The 5V and 3.3V lines are necessary because they provide power to the logic chips like the microcontroller and the radio transceiver. The 12V line is necessary solely for the electronic strike that we have chosen.

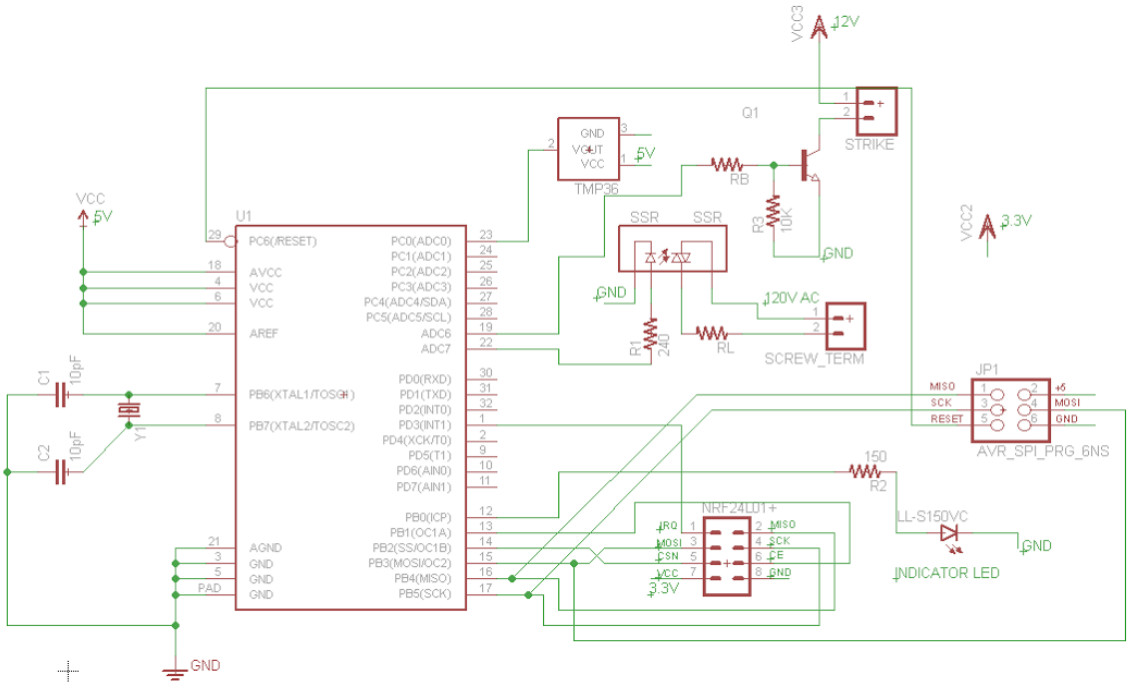


Figure 49: WHCS Control Module Schematic

All the control module endpoint targets are featured in this schematic. In the upper right of the schematic you can see the TMP36 temperature sensor, the electronic strike representation, and the solid state relay for switching AC. We are able to create all the control modules with the same design and just put whichever parts we want onto each individual control module based on which one of the three endpoint types it will be targeting. One option we were thinking of during design for modularity is instead of putting the footprints for the circuits necessary for interacting with endpoint targets directly on the control module PCB, we could put a section on each control module that acted similarly to a breadboard. This way we could shrink the size of the control module PCB because the circuit for each target would not be on every control module. A smaller PCB meant less money spent ordering it to be made. The issue with the breadboard imitation design is it does not look as good and it allows for mistakes to be made when soldering the circuit to the PCB. When we weighed the two options we decided that just having every circuit on each control module and only soldering the parts on to the circuit we were utilizing was the best option.

The control module schematic is quite similar to the base station schematic. It is missing the Bluetooth module and replaces that with all the different circuits for interacting with appliances around the house. The microcontroller is also smaller. The crystal that is shown in the figure is the same on the base station and the control module. Our goal during design was to make the boards for the control module and the base station to be as similar as possible. There was no reason to make them very different and it saved research and development time for us to be able to recycle footprints and things of that nature. One thing we realized when we were designing the control module and referencing other PCB designs is the importance of an indicator LED. We originally did not have any indicator LEDs in our design. We ended up having an indicator LED in every PCB we created if there was a pin available. The indicator LED for the control module is shown in Figure 50

along with the resistor necessary for forward biasing it. The LED indicator is valuable for showing that the board is getting power which can save a large amount of time during testing. The LED can also be programmed to blink while the control module is in certain states. The usefulness of the indicator LED should not be underestimated.

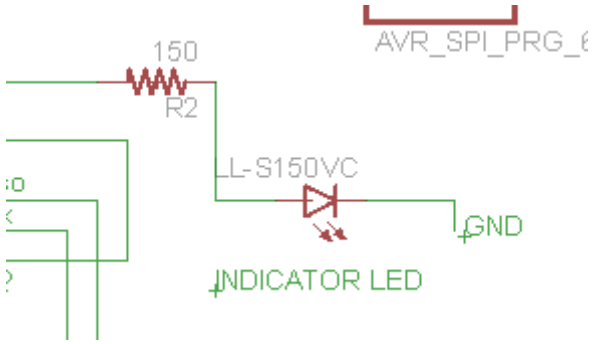


Figure 50: Control Module Indicator LED

The areas where we connect the electronic strike and the AC voltage to the control module are screw terminals. This is the best way we found to connect these components. The electronic strike is not able to be mounted directly to the control module because of it's large size, so wires will have to go from the screw terminals to the leads on the strike. Using the screw terminals for the AC voltage allowed us to shorten the traces carrying that voltage on our PCB because we could put the terminal wherever we wish. Similarly to the base station the NRF radio transceiver is connected to the board via female pin headers. The NRF comes with a breakout option so this is the easiest way to get it on our board from the prototyping phase.

6.8.6 Board Layout

After designing the PCB to the layout specification, the front and back images below are the result after sending the Gerbers to OSH park.

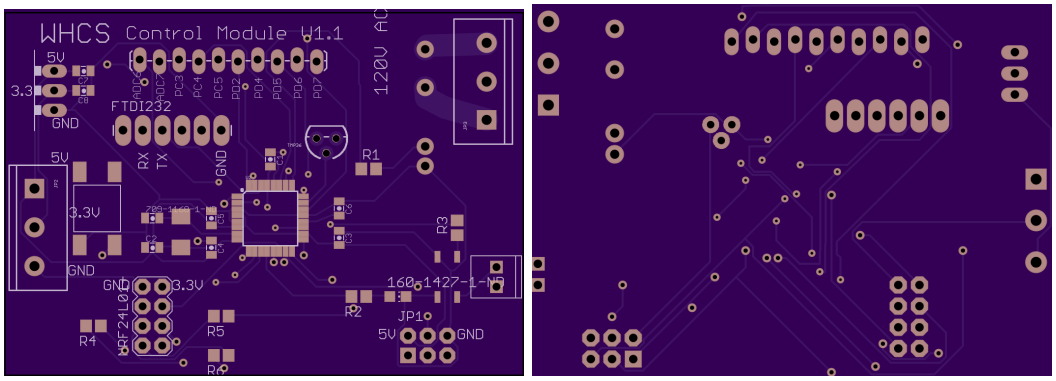


Figure 51: Control module OSH park PCB layout

7 Printed Circuit Board

7.1 Software Considerations

Before designing any of our Printed Circuit Boards, we decided to analyze which software would allow us to do the job the quickest and easiest. Nearly all of the team was familiar with EAGLE as it's one of the most talked about board design software due to its EAGLE Lite version. Instead of going with the most common solution, we decided to compare EAGLE CAD to another open source solution: KiCad.

7.1.1 EAGLE

EAGLE PCB is commercial software for schematic capture and board layout. It supports a wide variety of features that would help us make our board. The only issue is that the normal software costs money. Luckily, they offer a free evaluation version that can only be used for non-commercial purposes.

This freeware version of EAGLE has strict limitations in the size of the board that can be designed and how many signal layers there may be. The size of any board is limited to 4 x 3.2 inches[17] and there may only be a top and bottom copper layer. These limitations would be a show stopper for a moderately complex board, but considering our project requirements, it would be suitable. If we are to consider future board designs for WHCS, we may want a more flexible solution.

7.1.2 KiCad

As an alternative to EAGLE PCB, KiCad performs admirably well. It has all of the primary features of EAGLE and yet, is completely free and open source. The benefit of this is that the whole suite of tools is cross platform, allowing group members to easily work together despite different operating systems.

One issue with KiCad is the lack of a built in Autorouter. KiCad provides an external router, FreeRouting[18], but it has experienced recent legal trouble due to one of the developers previous employers.

Another nifty feature that KiCad has is its 3D board view. This feature is great for getting a sense of your board layout in relation to the selected footprints.

8 Prototyping

8.1 Point-To-Point Transmission

The most essential part of WHCS is the ability for the modules and the base station to be able to communicate wirelessly. In our research and prototyping phase we made sure that this feat would be achievable. To ensure that we were able to communicate using

our radio transceivers we set up a prototype for point-to-point transmission. The setup involved the use of two breadboards each populated with a microcontroller and a radio transceiver. One microcontroller out of the two was operating as the symbolic base station. The HC-05 BlueTooth module was connected to the Atmega328 microcontroller and through this module we were able to administer tests with the prototype setup. The microcontroller acting as the base station had a routine that enabled reading and writing to the NRF24L01's registers. We were able to ensure that the state of the radio transceiver is the state that we needed to communicate. The other microcontroller was connected to the other radio transceiver as well as a TTL-serial module for connecting to a computer's terminal. The control module microcontroller also had LED's connected to two of the GPIO pins. The setup that we created is shown in Figure 52 This setup allowed for us to make sure that the radio transceivers were able to send packets to one another and that the packets could be read into the microcontrollers.

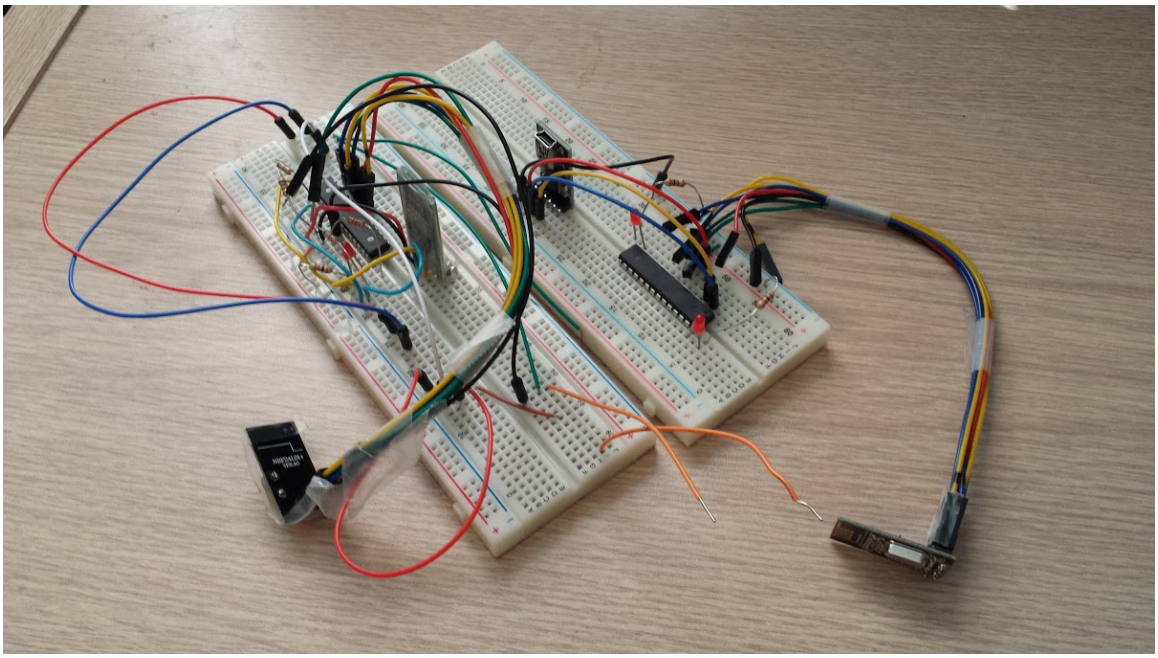


Figure 52: Point to Point Transmission Prototyping Setup

When we finished connecting and hooking up this prototype setup we were able to run the routine on the microcontroller that acted as the base station to take input from the BlueTooth module. Using an Android phone we communicated with the microcontroller and manipulated registers of the NRF24L01 to ensure that the microcontroller was communicating with the device using the correct timing protocol. Once we ensured we were correctly interfacing with the NRF24L01 from the base stations side we connected the NRF24L01 to the control module chip. The control module chip was connected to a computer terminal via the TTL-serial chip located in the top center of Fig. 8.1.1. We made sure that we were able to interface with the radio transceiver for this microcontroller in the same way as the first. Once both radio transceivers were confirmed to be connected and interacting correctly we ordered the base station to transmit data to the control module. The command was initiating from the Android application. The data was transmitted to the control module and was successfully received. As a result, one of the LEDs attached to the control module

breadboard toggled to an on state.

This prototype was able to give us a good gauge of how feasible our approach to designing a wireless home automation solution was. We were successfully able to get an Android device to communicate to a stripped down base station and then to a stripped down control module. The actions that were done with this prototype will be the core of WHCS. Every action done in the system centers around the ability to communicate between microcontrollers and to and from the mobile phone. When the control module microcontroller is doing more than just toggling an LED is when WHCS will be impressive.

8.2 Rogers Board Etching Prototyping

In our design we wanted to make the most use out of prototyping as possible. the worst mistakes are always the mistakes that are discovered to be mistakes in the last stage of a design. The first way to go about prototyping is to test things at a very low level. Checking if individual subsystems work the way that you would expect them to work, this level can definitely be done with a breadboard. Once the individual subsystems have been tested the next step is to check how these systems integrate, this too can be done with a bread board. At this point you already have a complete proof of concept. The final stage in prototyping is to do everything possible to make a design that is as close to what the final product will be as possible. Initially we thought we were etching our own powerboards. We figured that testing the control modules and base station with an etched PCB design would be the best way to know if our drawings from our gerber files lined up the way we wanted them too. As a prior step you can also compare dimensions of the parts of the design with the gerber file drawing by simply printing out the gerber file on to copy paper and placing elements on top. Etching would have added add an even further advantage in that it will allow us to see if the electrical connections in our drawing worked out the way we would expect them too. If we had proved the concept to the level that it has been etched on to practice boards we would have known that our design would work as it was intended to work. We did however decide not to make etched boards when prototyping our designs. We were pretty confident in our designs and thought it would be best to get our boards fabricated as soon as possible.

Etching is actually a fairly simple process. If we had etched we would have used toner paper to transfer the gerber file from the computer onto the board. Next we would have utilized TRF paper to further protect the design. Finally we would have used an etchant to eat away the copper that is not part of our design. There are many different options for etchants that could have been chosen in order to complete this step. The most easily accessible though is made by combining vinegar, hydrogen peroxide, and salt together. After using the etching solution it is important to dispose of it properly (it can't just be poured down the sink). Since the solution is very acidic balancing it with a strong base such as baking soda or soap will prepare the solution for a better disposal. We would have used rogers corporation boards because they provide free laminate samples to students. Once the board was etched we would have simply soldered on the components to the board.

8.3 WHCS Proto-Panel

Presentation plays a huge part on how the public feels about a product. This is why it was so important that the display of the project was well put together. Not only is it important to have a nice display for the purpose of it being a proper representation of our design, but also so that it is aesthetically appealing. If WHCS were to be launched into industry, marketing would play a huge part in it's success. People's first impressions are always driven by what they see. If what they see causes them to believe that the product is of high quality, they are less likely to be highly skeptical of how the product performs. In this section we will be going into detail about how we made our display in order to showcase the functions of WHCS. This section will not go into the practical side of how the display is coming together; rather it will lay the framework for what goals we want to accomplish with the display.

Our design is not a plug and play design, therefore installation will require more than simply plugging in the system. We had to find some way to duplicate what the installation of a house would look like. In a home what we're provided with is interior wiring. When we are actually presenting our idea what we'll be presented with is an outlet that we can use to draw power from. Therefore the first thing we did was convert this outlet back into wiring. To do this we took a basic power cord that had a hot, neutral, and ground wire and spliced it with 14 gauge wire that we got from home depot that is meant to be used in a home. We spliced this wire into may different branches and used it to power the control modules and the base station.

We wanted to make the display as accurate of a representation of what would be found in a home as possible. Therefore we tried to follow as many codes and standards for home construction as possible. Since we ourselves do not have a homeowners electrical permit we are not equipped to actually install the system in a real home. Yet for demonstrative purposes we did fine to follow codes and present them in our display. In Section 9.3.3 we go into further detail of what was done to follow these codes and standards.

Our display consists of the frame, the wiring, and drywall. Each made to follow standards. We also decided to make it such that each board was displayed in our proto-panel. To do this we replaced part of the drywall with plexiglass and displayed the temperature sensor control module and the power board.

8.3.1 Materials

The first thing we needed was a plug. It was important that we used a three prong plug because wiring in the home uses three wires. In addition to the wiring, the interior walls of homes consist of drywall, insulation, and a wooden frame. The wooden frame of homes is made out of 2 by 4 wood. These pieces of 2 by 4 wood are usually put together with either screws or nails. Drywall will also be needed to provide the presentation side of our wall. For drywall all we needed was enough drywall to cover the entire wall along with some screws in order to attach the drywall. Insulation wasn't necessary since we weren't worried about temperature or sound insulation for our project. Also insulation in a regular home wouldn't interfere with the installation of our project so really the insulation is irrelevant.

In addition to these basic materials made to construct the walls we needed a few other

things. First we needed the actual control module and base station boards. If this design were a final product these boards along with the power board would be housed in a case that would be attached to the wooden framework. However for our project we simply mounted the boards directly to the frame of our display. For the light control module we needed to buy a wall mounted lamp with a three wire connection. For the outlet we needed to buy an outlet. For the outlet and light control modules the relays were placed along the hot wires in order to switch them on and back off. To display the outlet control module we needed to have something plugged into our display board outlet (we used a coffee maker). Additionally we needed a door knob, the strike, a lamp shade, and some plexiglass to display a power board and the temperature sensor control module.

8.3.2 Dimensions

The dimension of our project was pretty arbitrary, it just had to be large enough to fit each control module along with the base station. The first thing that we decided was whether or not we wanted to use a full size door for our design. Although the idea was tempting, because it would really give the user the feel of a home experience, we decided against it mostly because of weight. If the fame had to be of that size all the wood used in the frame along with the weight of an actual door would have made the project very difficult to move around. Also it would have been extremely bulky. Getting an entire door (actually even larger than an entire door because of the other attached components) through a door can be quite a struggle, add weight to the mess and you're asking for difficulties.

The door we used was a homemade door. Fortunately because we designed the frame that will be used for the wall, we simply made the gap in-between the studs the same size as the door we wished to make. We decided to use a 2 by 12 piece of wood in order to make the door. As a rough estimate we decided that 2 feet by 2 feet would be a large enough area of space to display each individual module. The total square footage of the five control modules would be 20 square feet. We decided that a 4 by 5 feet display wall would showcase our design quite nicely. Yet ended up going with a 4 by 6 feet display because it was easier to view and play with.

8.3.3 Sketch

This section shows the original mock up of our 4 by 5 display (even though we ended up using a 4 by 6 display). The most important interactive parts of our design are the door access control module and the base station. We made sure that these two modules were at an acceptable height where they could be interacted with. In the mock up we decided to place three of the modules on the bottom portion and two modules on the top portion. The two on top were at a more accessible height therefore the door access and the base station were placed there for our mock up. The light outlet and temperature sensor were placed on the bottom half. To make things look symmetrical the lamp was placed in the middle while the outlet and the temperature sensor were placed on the side. The design we came up with in the mock up is shown below in Figure 53.

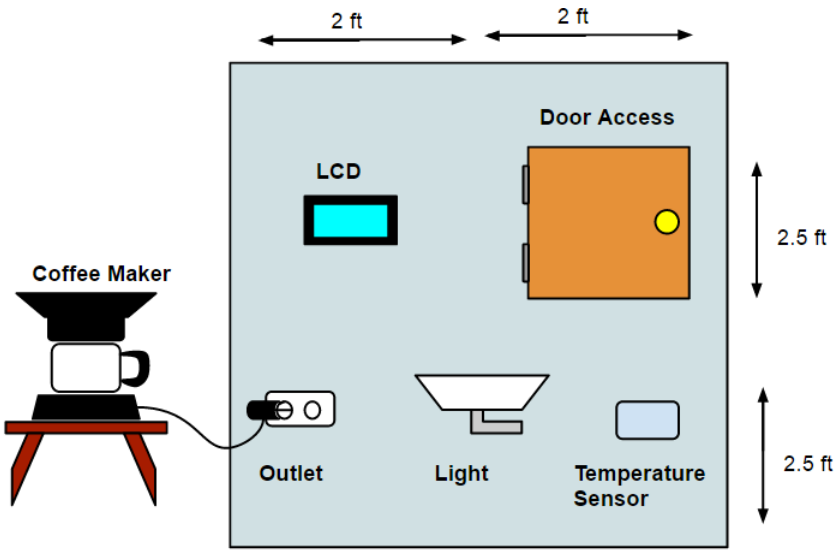


Figure 53: Mock up design of WHCS display board

What we actually ended up building is shown in Figure 54 below.

Figure 54: Built WHCS display board

9 Manufacturing

9.1 PCB House

After carefully designing our schematics, we first need to decide on a PCB manufacturer to use for our printed circuit boards. There are a few popular and affordable options available for hobbyists and small run jobs. We weigh the options available in the following sections.

9.1.1 Seeed Studio

This PCB house offers many services including a low cost PCB service. The pricing model is based off of the number of layers, thickness, and size class. The size class of 10cm x 10cm would have a base cost of around $21.00. This is quite fair and would meet our needs for manufacturing. Seeed Studio is based in China, which is something to take in to considering if we are ordering boards and need them quickly. Also, even ordering initial boards would have a long enough turn around time that we wouldn't be able to test and construct our design iteratively. Seeed's cost is very competitive, but we would be losing time to save cost.

9.1.2 OSH Park

OSH park is quite a popular option due to their low price point for small boards. They are known by their distinctive purple boards. WHCS has decided to choose OSH Park due to the fair price for a 2 layer board run. A single run would yield 3 copies for $5 a square inch. Unfortunately, this pricing model will make any significantly large boards too expensive. For the control modules, it is possible to create a very small design that still meets the WHCS requirements. WHCS has no special PCB requirements beyond being able to create surface mount pads for the control module and base station boards. The base station PCB will be limited by the size of the LCD, but this is only around 2 by 3 inches.

Overall, OSH park is the best choice for WHCS' budget and technical requirements. Its fast shipping time will ensure that we have plenty of time to test our board and make any additional runs if the worst happens. We ended up going with OSH park for all of our boards. We had two orders of three control modules, one order of base stations, and only one order of power boards to save on cost.

9.2 Parts

This section discusses how parts came into play of the manufacturing of our PCB. With choosing parts there was much to consider. In addition to whether or not the part performs as we needed them to, and whether or not these parts were in stock we also had to take a look at how each part would be implemented into our design.

9.2.1 Footprint (SMD vs Through-Hole)

The footprints of our board relied mostly on the parts that were chosen to be part of our design. The parts needed for a surface mounted board and the parts needed for a through hole boards are different. Therefore in order to continue make our boards we had to make a design choice. In WHCS we considered both through hole boards and surface mounted boards. Through hole board technology is the older of the two technologies and is currently much less popular than surface mounting. One of the of the advantages of surface mounting is that it takes up less space allowing more real estate for parts for a given board. Because surface mounting does not involve drilling it is simpler and faster to construct. Although there are some advantages in through hole boards for most applications surface mounted technology wins. Therefore in our design we will be using surface mounted technology. With the exception of the power board which will indeed take advantage of through hole for our heavier components. The footprints in our control module.

9.3 Construction

In this section discuss the assembly of our project. This involves decisions that were made for the PCB board construction along with the construction of the display that shows how WHCS operates.

9.3.1 Soldering

Once we received our etched boards we had to consider methods to solder on the components. There are various methods that can be used to accomplish soldering. Note that any of the methodologies described could have been used to accomplish the task. The first of the methods to be described is the use of hand soldering. This is by far the simplest and the easiest conceptually for our group to implement. Every member of our group has some experience soldering by hand. This method is cheap as it only really requires a soldering iron, which most if not all members in our group own. The main take back with this method is that while it is easy by methodology it is difficult in the fact that it can be time consuming. Also since everything is done by hand this method tends to be a little messy and may tend to look less professional. This is even more so the case if we make use of flux, which can tend to make the board look dirty. We initially wanted to use this method as a last resort.

9.3.2 Reflow Oven

We also considered refow. Reflow methods are different from hand soldering in that it separates the placement of the components from the soldering. It tends to be less time consuming and does a more consistent professional looking job. There are various method that our group could use to reflow the solder for our PCBs. The UCF amateur radio club has access to a professional reflow oven that we could have taken advantage of. Additionally, there are online methods describing how to use a toaster oven to achieve the same affect.

In the end we actually did a combination of both reflow and hand soldering. We achieved the reflow by using solder paste and a heat gun. Hand soldering SMD parts was actually

easier than expected with the use of magnifying glass.

9.3.3 Proto-Panel

In this section we will focus on some of the details of the construction of our design, the general overview of how the display board was put together is given in Section 8.3 This section will not explain how the proto-panel came together, rather it will explain the non obvious construction details. We discuss the design considerations that had to be accounted for. This section takes into consideration the norms that go into home construction. It also goes into the specifics of safety precautions as well as regular sizes and spacing used in a home.

First let's talk about the wiring. The amount of current that can be drawn from an outlet is 15 amps if not 20 amps, therefore we had no problems in drawing enough current from the outlet to power our board. 15 amps is more than enough to satisfy our needs even when the current is divided into five different applications. Something we had to consider was the gauge of the wire used. The gauge of the wire dependents on the amount of current it can safely handle. As discussed previously 1 amp is larger than anything we ever see from our circuit. Therefore we designed our wire gauge for 1 amp. However as stated before most homes are designed to be able to draw 15 if not 20 amps. This level of amperable is equivalent with wire of gauge 14 and 12. A common brand of wiring used for these tyes of applications is Romex. Just for the sake of being consistent with what is used in the home we used 14 gauge wire. To splice the wire we considered either hand soldering it or we using a wire nut. Both are an acceptable methods for joining the wires, and both are used in homes. Wire nuts are considered to be an easier/ faster method for doing the job. While soldering is seen as the higher performing link. We will made our links by soldering them because it is slightly more professional than using a wire nut but really this is simply a matter of preference.

When wiring something it is often a good idea to attach your wire at more locations than simply the location where the connection is made. This way if for whatever reason the wire is pulled the stress will not go completely to the connection. Before splicing the wire and connecting the wires to the different control modules and base station, it was a smart safety precaution to run the unspliced wire through the wood framing and attaching it. After splitting the wire it was a smart idea to continue the practice of attaching the wire at more places than the connections. While doing our wiring we made sure to match the understood color scheme. In our design we have a single phase hot and neutral and ground. In the US the ground wire matches with green, the black wire matches with the hot wire, and the white wire matches with neutral. [14] Using these color codes made it easier to keep the project neat and organized. It made it easier to avoid making mistakes in wiring our circuits.

Now let's discuss the boards themselves and how they were placed into the framework of our wall. originally we wanted to place the boards in some sort of casing that would easily be attached to the framework of the home. Our thinking was the less the number of things needed for installation the better. The easiest way to do this would have been to attach them to or place them inside of the electrical boxes. Having only one thing to install per control module or base station would have made real installation of our device more realistic

for actual use in the future. In our design we wanted to make use of electrical boxes since they are used in home electrical wiring. The boxes can be either metal or plastic, yet plastic boxes are a little easier to work with as they the holes are easier to punch out. In the end we just mounted the boards to the frame, yet if we had more time we would have made some sort of casing similar to what was described. [15]

There are some specific considerations that were made with the different control modules. First off for the light it was important that we used an actual wall mounted light, as this is the type of light fixture that we would be controlling in an actual home. It was important that it was not simply a plug in light, because this would defeat the purpose of having light fixture control module. It would have been the same thing as the outlet module. The wall mounted light we used came with three wires; a hot, a neutral, and a ground wire. From these three wires we were able to install the fixture in the same way as what would be expected in a real home. The hot wire is the wire that we interrupted with the relay in order to switch the light on and off.

The outlet we used was a GFIC. GFIC stands for ground fault interrupter. Using this outlet provided an extra safety precaution. What a GFIC outlet does is constantly compare the output current from the neutral wire to the input current from the hot wire. If there's a difference in current, within the range of a few milliamps, the outlet will shut off in 20-30 milliseconds. In the case that someone were to be electrocuted by this outlet, the current that goes into their body would cause a current leakage that would cause the GFIC to have a current difference and thus shut off. GFIC outlet are normally required for kitchen and bathrooms. Since none of the members in our group have extensive experience in working with AC power it was best that we took every safety precaution available. We do not expect that homes that actually implement our design will use GFIC outlets, it is simply an extra safety precaution that our group decided to take. [16]

For the door control module we decided to make our own door with a 2" by 12" piece of wood. Since we did not buy the door but are custom made it ourselves we needed to cut the holes ourselves. The first order of business is cutting the door to length and leaving a frame of the right size. To cut the hole of the latch we will needed to use a $\frac{7}{8}$" spade bit. The hole for the door knob will had to be made with 2-$\frac{1}{8}$" diameter hole saw. A 1" wide chisel was used to cut out the recess of the latch. After all that cutting we were able to install the door knob.

10 Testing

10.1 Power Supply

When in the process of building something it is important to check to see if what you designed for is similar to the results you are receiving. That is why it was important to run test occasionally to see if the results matched. In this section we will be going through the steps that we took in order to verify the accuracy of our designed power supply. It was of great importance that the power supply worked as expected otherwise the entire project would fail to work properly. Our power supply consists of various lines each operating at different voltage levels. It was important that we checked each level to see if they were as

we anticipate them to be.

10.1.1 Line Integrity

For our boards we have a 12V line, a 12V line, a 5V line, a 3.3V line, and another line at 120VAC. Each line had to be tested and measured. This testing was done as early a the prototyping stage of assembling our design. The earlier boards are tested the better. We intended to recheck our values at each progression (from one type of prototyping to another) by doing this we could ensure that nothing abnormal happened as we continue to make changes. The first level of prototyping that could have occurred was at the breadboard level. At this stage we would have had nothing more than the parts we wish to put on the board along with the breadboard. First we would hook up the AC power to the transformer, on the other end of the transformer we could check to see if the 120VAC was converted as expected into 14VAC. Although this could be done using a basic multimeter, using an oscilloscope is nice because it allows us to see the entire waveform. Next we were also able to test the circuit after the rectifying diodes. Once the voltage was measured we saw voltages of about 20V since the AC voltage level does not translate exactly into the DC values. At this point we could see what our ripple was, either by having the oscilloscope give the max and min values of the circuit. Finally we were able to test if the DC buck regulators were working as expected. At the breadboard level we were able to use the probes to pinch the circuit, yet once we were testing our fabricated boards we needed to hold the probes onto wire of the boards.

10.1.2 Battery Backup

If we had implemented it the backup battery would have been another power source that would have needed to be checked. Usually for most buck regulators there is a minimum voltage level that they will accept in order to convert that voltage to the level that they are designed to yield on the output. The only buck regulator that we would have needed to use with the backup battery would have been the 5V regulator. We would have wanted to test the back up battery to make sure that the battery that we used was a high enough voltage to be steped down by the buck regulator. Although this information can be found in the datasheet, it would have still been a good idea for us to test different input voltage levels to see what the cutoff would be for our specific regulator. With that information we could have made sure to put in a backup battery of a high enough voltage. However since a back up battery was never placed in our design we never had to worry about testing it.

10.2 Base Station

The base station is the most complex item to test and verify functionality. There are a lot of modules working together to make the base station work with the WHCS network. This is why testing these certain aspects of the system will help ensure good performance and prove that the individual subsystems are functioning properly.

10.2.1 LCD Control

Controlling the WHCS network from the LCD is a key part of the design. The base station should be able to accept touch inputs from the resistive touch screen on top of the LCD, dispatch that event to an underlying UI element, and cause some action to occur. If the button to command the control module that is responsible for the electronic door strike is pressed, this information must flow quickly and correctly through the base station's code. A simpler test would be the toggling of a remote LED by hitting a button on the LCD. This test would prove many things correct simultaneously.

10.2.2 LCD and NRF Simultaneous

One consideration that was discussed earlier was the fact that the base station's microcontroller has a limited speed. The NRF, LCD, and BlueTooth modules will need to be serviced as quickly as possible by the base station MCU. If there were to be an error in the code that handles the LCD drawing, such as a slow graphics drawing loop, then the performance of all the other modules would suffer. In order to stress test the system, a test case where many packets are being sent and received over the NRF radio, while an expensive graphics drawing operation is occurring. If either module takes too much execution such that the other suffers, then the code structure will have to be rethought and designed in order to avoid this starvation. A solution to this problem if it arises would be to have points in either process where the action can be paused or canceled to be resumed at a later time. That way the MCU can perform the most important task at any one time. This would be partially handled by the NRF radio generating interrupts when a new packet has been delievered, but the packet still needs to be fetched and processed.

10.2.3 UART and Software Serial

The UART is a quintessential part of a microcontroller development system because it allows for easy debugging and testing. The UART can be used to log information to a screen to show the internal state of the microcontroller and therefore the system it is controlling. The UART can also be used to give simple commands to the microcontroller that would otherwise be given through other components attached to the microcontroller. The base station for WHCS will have a BlueTooth module connected directly to the Rx and Tx lines so the lines will be blocked for a simple UART chip. However the BlueTooth module itself can be used for debugging. In WHCS an Android device will be used to host the application to interact with the system. The Android play store has an application available for download called BlueTerm. With this application any Android phone will be able to easily connect to the BlueTooth module that is connected to the base station. An Android phone will be a viable option for printing out debug information and performing tests that would benefit from the ability to manually input certain commands directly through the UART.

We realized that there may be certain times during testing and debugging that the BlueTooth module would not serve as a good method for printing out information to a terminal. For example the BlueTooth module won't be usable for testing when we are programming the BlueTooth module as mentioned in Section 6.3.2 or when we are testing things that require interaction with the WHCS Android application. For these cases we will have an

alternate UART chip, the FT232RL FTDI, designated for debugging and testing. This will be a simple UART module that connects to two pins on the base station's microcontroller and then to a computer's USB port through a mini-USB to USB connector. Since the base station's Rx and Tx lines will be occupied by the BlueTooth module, this serial module will have to be connected to two GPIO pins of the microcontroller and we will have to utilize software serial. There are already public libraries and routines available for implementing software serial both in half-duplex and full-duplex operating modes. The forum AVRFreaks.com has plenty of information on the topic and we will base our routines for software serial debugging and testing off of the examples listed on their site. The only things that we should have to specify are the two pins that will operate as the transmitter and the receiver on the microcontroller. It is possible that we will only need half-duplex operation to confirm all of our test-cases and debugging.

10.3 Control Module

The control modules are significantly easier to test than the base station because of the usage of a single major module - the NRF radio. Some control modules will have specific roles that require specialized testing, but the common functionality is the most important to be tested. All other verification will build off of the results of this initial testing.

10.3.1 UART Chip Testing/Debugging

Unlike the base station the control modules do not have a Bluetooth module attached to the UART that can be used for debugging. The absence of the Bluetooth module frees the Rx and Tx lines of the control module microcontroller's UART. This means that the FT232RL FTDI chip that is used for debugging and testing the base station when the Bluetooth module is unavailable is a suitable option for the control modules. The ATmega328 registers for operating the UART are fully operational for debugging and testing and we can even leave the serial module in circuit for UART debugging access at any time. This is because there is no other need to use the Rx and Tx pins of the control module microcontrollers. Unlike the base station the control modules suffer no limitations from the implementation of software serial routines. The control modules are able to operate in full-duplex mode. In full-duplex mode we are able to give commands to the control modules that would otherwise have to be received from the base station through the radio transceiver. This allows for ease of development and testing.

10.3.2 Command Execution

The control modules frequently receive commands from the base station. There are different types of control modules that execute different commands. Whenever the control module receives data from the base station that signifies that a command should be executed the control module should carry out certain operations to fulfill the request. Tests need to be carried out when the control modules are setup so we know that the control modules are all capable of completely executing the command sets that are available to them. The tests need to be decoupled from the communication pipeline of the base station in order to ensure that no factors outside of the control modules scope are interfering with the

test. Command execution testing is executed through the microcontroller's UART port. Commands are given just as they would be given via the base station, the only difference will be the method through which the control module receives the command.

For each type of control module the full set of actions available to it is listed. From the list for each control module the commands to perform those actions are given through the UART. The tester documents the success of each individual command given. The command execution tests are passed once the control modules for every independent role can perform their tasks completely and without error. The known control modules roles and actions available to them are listed in Table 7. All of the actions listed in the "Commands Available" column must be performed correctly in order to ensure the proper operation of the control modules for WHCS.

Control Module Role	Commands Available
Light/Outlet Module	Toggle On, Toggle Off, Check State
Door Strike Module	Lock Strike, Unlock Strike, Check State
Sensor Module	Read value

Table 7: a tabularization of control module roles and the available commands

10.4 Door Access

One of the most critical functions for WHCS is the control over a door. WHCS must verify that the door can be correctly engaged and disengaged wirelessly from commands by the base station LCD or Android phone. Both paths of execution need to be tested and verified for correct functionality in these areas to ensure that no homeowner will be left stranded with out a way in to their home.

10.5 Android To Base Station Communication

Once all of the independent components of the base station have been tested and shown to be working correctly it will be time to see if the base station is able to communicate with the Android device. This test will require that the base station's microcontroller is hooked up to the HC-05 BlueTooth module and that the Android device has BlueTooth enabled. Confirmation that the base station can communicate to the mobile phone is essential for the proper operating of our system.

10.5.1 BlueTerm

The simplest method we have for full-duplex communication between the base station and an Android phone is BlueTerm. BlueTerm is a free terminal application on Android. It allows for scanning for BlueTooth devices, connecting to other devices, and setting up the framing of packets. Using BlueTerm we are able to connect to the HC-05 on the base station and send serial data to the microcontroller. The microcontroller is able to receive the data from BlueTerm and also reply with data by using the HC-05. By writing a simple echo routine on the base station microcontroller that receives data from the UART and then

echoing it back out of the UART we can test whether or not the BlueTooth communication is working. With this simple test setup we are able to quickly test the functionality of our circuits once we create our printed circuit boards. We just need to open up BlueTerm, connect to the HC-05 module, and then type any letter into BlueTerm while awaiting the letter to be echoed back. If the letter is echoed back then we know that the connection is good and we are able to send data from Android to the base station. If the letter is not received then there is an issue in the base station. The error could be coming from the UART routine, the circuit connection, or the BlueTooth module, but most likely if this test fails it will be because of the base stations circuit connections.

10.5.2 BlueToothListener

BlueTerm is a very useful application that we can rely on to ensure that the link between the Android device and the base station is functional. BlueTerm will not suffice to ensure that our application can communicate to the base station. In our application we use raw BlueTooth sockets for data exchange with the base station. The main class that handles communicating to the base station is the BlueToothListener which is mentioned in the Android application section of this document. Testing this class is essential to ensure that the communication link is working correctly for our application. BlueToothListener has access to the socket that wraps the buffers for communicating with the base station microcontroller so writing to and reading from this socket needs to be performed. When we have tested and confirmed that we are able to use the socket in the BlueToothListener class for full-duplex communication then we know that our application is fully capable of sending whatever data we wish to exchange.

The BlueToothListener class is decoupled from other classes that handle what is done with the data received. This design allows for modularity in testing. We can make a test class that subscribes to the BlueToothListener class's data received event and then asserts that the data is what we expected. Then when we want to use the BlueToothListener for the actual application and not just testing we can simply unsubscribe the test asserting class from the data received event.

10.5.3 LED activation test

Activating an LED is a standard test that we used during prototyping and testing for WHCS. For Android to base station communication testing, activating an LED ensured that the base station was able to perform commands based on the data exchanged between the Android device and the microcontroller. The LED activation test can be performed through BlueTerm or through the raw BlueTooth socket that is present in the BlueToothListener class. This models situations where the user of WHCS wants to alter the state of the system from the Android phone. Toggling the state of an LED is the simplest form of physical state change. When we were able to turn our LED on and off we knew that the Android to base station communication link was fully operational and we were be able to control the system from the mobile device. Table 8 shows the way the test was implemented. The microcontroller receives bytes from the mobile device. Based on the byte that we send the microcontroller will perform the required action. The table shows what data results in the LED on state as well as the LED off state.

Data sent to microcontroller	LED state
'A' (0x41)	ON
Anything but 'A'	OFF

Table 8: LED Activation Test Commands

11 Demos

11.1 Voice Controlled Light Activation

The first demo that we would like to perform with the functioning prototype of WHCS will be voice controlled light activation. Voice control is one of the big features of WHCS because it adds a lot of interest to any project. Light activation through the system will be one of the most common use cases. Voice controlled light activation combines an exciting feature with one of the most common use cases so it will be a frequent demo. To perform this demo the Android application will have to be paired with the base station already and ready to communicate to the system. We can allow any person that wishes to participate to utilize the application and access the voice command feature. The person performing the voice activation will be able to say something similar to on or off and toggle the state of a light in the system. This demo can be extended to include outlets as well. We will have a coffee pot hooked up to an outlet being controlled by WHCS. After activating the light the performer will be able to turn off the light and then start interacting with the outlet. The outlet can be turned on, thus turning on the coffee pot. This is a preferable way to start the day so it should be a relatable demo for an audience. From this point we can show that this feature is also accessible through the GUI of the Android application. The voice chat feature is only a replication of the GUI capabilities and this is an important point to make.

An extension that would help this demo would be letting participants create their own commands for interacting with WHCS. This is a feature that we will be incorporating into the Android application. This demonstrates how WHCS is designed to be customizable for each household. A participant of the team will be able to add a setting that merely saying mouse turns off all the lights and outlets connected to the system. Then the performer would say mouse into the voice control system and the result should be the lights and outlets of the display being turned off. The point of this demo is to show the ease of use and the power of the voice control in the Android application.

11.2 LCD Light Activation

In order to demonstrate how the base station LCD works separately from the Android phone, we show that all of the WHCS functions can be controlled from the touch panel. The simplest test is to press a button on the LCD and have a controlled light toggle off and on. This demonstrates that WHCS is functional on many levels: the LCD works and can receive input, the base station can send packets to a specific control module, and that specific control module can correctly execute a received command. Members present during the demo were able to see for themselves how the LCD operates. They could explore the menus and try to control some modules themselves.

11.3 Sensor Query

Sensors are a passive part of WHCS that enhance the overall appeal of the system but do not demo as well as turning things on or off. Our prototype of WHCS will have a functioning temperature sensor control module that is updating based on the temperature of the environment. In our demos we want to make sure to show the operation of this sensor. WHCS allows sensors like temperature sensors to be mobile around the home. The sensor can be plugged in anywhere and receive power and update for the temperature of that environment. When we are demoing WHCS we want to show this mobility by showing the sensor being moved around and updating accordingly. The demo will show how the sensors information is viewable from the LCD as well as the Android application. If the sensors data is not recent enough we can prompt the sensor to update its most recent reading and this should be performed during the demo. This demo could be made more powerful for viewers if the system reacted to changes in the sensors reading. At this point in time we have not considered methods for WHCS to react to sensor data. For our purposes the sensors will only serve as information providers for users.

11.4 Fault Recovery (Loss of Power)

A demo that would have a lot of impact would be to pull the power on WHCS. To prove that the backup battery failover systems work, physically yanking the power cord from the wall would show that the system is able to quickly recover from an unexpected power outage. Technically this would be like watching a uninterruptible power supply activate, but on a smaller scale and with hardware that is visible and easier to understand. Also, the battery indicator on the LCD could reflect the detected power event and spread that event to all listening control modules and connected phones. That way the user could be remotely notified in the event of a power outage. Of course due to the limited range of the BlueTooth connection the owner would most likely know when the power goes out as they are probably in the home.

Due to design changes, this demo was never done, nor implemented. It required a backup battery infrastructure, which was removed from the design.

11.5 Remote Door Access

The electric strike of WHCS will provide us with a door access demo. This will be a great demo because it is very appealing to be able to unlock your door wirelessly and it will show off the scalability of our system. The electric strike will be physically mounted to the proto-panel that we will be using with our demo. While the command to unlock the strike has not been given it will be impossible to open the imitation door that is attached to the electric strike. The point of this demo will be to show that from the Android application it will be possible to activate the electric strike and therefore allow the door to be opened. The electric strike will also be available to be unlocked from the LCD. During this demo we will explain that the electric strike that we use for WHCS does not consume power while it is in the locked state. It is only drawing current when it is toggled to the open state and this happens for a maximum of thirty seconds. This will also be a great time to mention how the utilization of an electric strike does not impede normal home access. As long as

the doorknob on the door has a key lock on it it can remain in the locked position at all times. if the electric strike ever fails and is unaccessible from the application then the door knob can simply be unlocked and bypass the electric strike.

12 Project Management

The large effort required to create a functional, working product, wouldn't have been possible without careful planning. The project management of WHCS is paramount to our success as a team. We need to be able to estimate our costs, discover parts for our design, and make sure we keep on track of our timeline. Planning these aspects of our project before hand creates a well defined roadmap that all group members can follow. Without such planning, everyone would have their own perception of time and responsibility and the project would flounder.

12.1 Budget

To help discover the costs of WHCS before diving straight in to the project, we planned a comprehensive budget. This budget contained names, quantities, and estimated costs of all the major parts of our project. This itemized breakdown of WHCS allowed us to budget our funds amongst the group members and also to apply for external funding. Through the use of our detailed budget, we were able to land a funding offer of $434.42 from Boeing. This covered all WHCS expenses except for a few items bought by the members themselves. Beyond this funding offer, the group members stated that they were to provide up to $300 each.

The costs listed in Table 9 are derived from costs researched online. These are the best estimate at the time of creation for each item. Some items are special in that the costs are dependent on other factors, such as size. This items have a note specifying the method used to come to the final price. This is important to note for auditors and for our future purchasing.

Item	Cost	Qt.	Item Total	Note
Base Station Bluetooth Module	$7.95	1	$7.95	
Base Station RF Chip	$0.97	1	$0.97	
Base Station PCB	$90.00	1	$90.00	OSH Park[11] $5 per square inch. Assumed 4x4 board, and $10 shipping
Base Station LCD	$30.00	1	$30.00	Comes assembled
Control Module RF Chip	$0.97	4	$3.88	Per control module
Control Module PCB	$33.00	4	$132.00	[12]
Control module MCU	$4.00	4	$16.00	Per control module
Pocket AVR Programmer	$15.00	1	$15.00	[13]
Solid-State Relay	$17.00	4	$68.00	Alternative to mechanical relay
Bluetooth Capable Phone	$0.00	1	$0.00	Members possess a Bluetooth capable phone
Casing for Modules	$2.00	5	$10.00	
Battery Holder	$2.50	5	$12.50	
DC-DC Step Down (LDO)	$1.00	5	$5.00	1 per board (including control modules)
AC Rectifier	$5.00	1	$5.00	Self-built and assembled (diodes, capacitors)
GFIC Outlet	$17.00	1	$17.00	With safety features
Electronic Strike	$50.00	1	$50.00	For access control
Temperature Sensor	$1.50	1	$1.50	
Light Fixture	$4.50	1	$4.50	Bulb and socket
Tranformer	$15.00	4	$60.00	120v AC to AC voltage of logical DC
Shipping Costs	$8	12	$96.00	Boards , parts, prototyping equipment
Project Miscellaneous	N/A	N/A	$50.00	Wire, Resistors, Capacitors, Inductors, LED's prototyping equipment, etc
TOTAL	**$675.30**			

Table 9: a comprehensive break down of WHCS' budget

Naturally making a completely correct budget that covers every possible expense would be difficult and time consuming. There will be items and purchases made that are below, above, or not even on the list. With sufficiently complex projects this is to be expected, but will not be a problem. The individual money offered to support the group will cover any additional, unplanned expenses.

In order to keep track of what was reimbursed versus what was purchased by group members, all receipts will be kept for record. Any purchases made will be documented and a running sum of used funds will be maintained. This will keep the purchases within the range of the budget and prevent large, unexpected costs from surprising group members.

12.2 Parts Acquisition

In order to fulfill WHCS's Bill of Materials, we will need to identify manufacturers, retailers, and any other online and offline resources for purchasing. Initial research gave the beginning insight in to where to look for specific products. This should be considered early on in order to consolidate purchases, which will simplify shipping and ultimately save on cost. Some items may be special enough to warrant their purchase from a "one-off" retailer. The electronic strike falls in to this category due to its specialty nature.

For all of our bulk electronic parts, we will be purchasing them from Digikey or Mouser, depending on the price. Digikey is familar to us as engineers and their interface for purchasing is straight forward. Additionally they ship from the United States, which will ensure that we get our parts in a timely manner. Shipping parts from China may be cheaper overall, but the cost is higher in the time lost to waiting for their arrival. To avoid this debt, purchases from overseas will be made a early as possible. That way we will not be blocked in our design by waiting for parts.

12.3 Milestones

The WHCS project has a limited amount of time to complete its end goal of having a working system. When thought of all at once, this seems like an insurmountable task. To simplify this large problem, we have broken up the task in to smaller, more achievable, milestones. This milestones give us a target window for completion and also the interdependencies between milestones. For instance, if the PCB board for the base station is not finished, the next step of sending it off for manufactoring will be delayed with every day. This will push every dependent milestone back causing a project delay. The gantt chat shown in Figure 55 is used to keep our project goals in check with the reality that we a final deadline.

	Name	Start	Finish
1	**Senior Design 1**	**02/03/2015**	**04/25/2015**
2	Research Bluetooth Communications	02/03/2015	02/13/2015
3	Research 120V AC Rectification	02/03/2015	02/13/2015
4	Research Communication between 2 RF Chips	02/03/2015	02/13/2015
5	Prototype Bluetooth Communications	02/13/2015	02/23/2015
6	Prototype 120V AC Rectification	02/13/2015	02/23/2015
7	Prototype Communication Between 2 RF Chips	02/13/2015	02/23/2015
8	Research Outlet Control Module	02/23/2015	03/05/2015
9	Research Android App Development	02/23/2015	03/05/2015
10	Research Door Control Module	02/23/2015	03/05/2015
11	Prototype Outlet Control Module	03/05/2015	03/15/2015
12	Prototype Android App	03/05/2015	03/15/2015
13	Prototype Door Control Module	03/05/2015	03/15/2015
14	Research Light Control Module	03/15/2015	03/25/2015
15	Research LCD Interface with Base Station	03/15/2015	03/25/2015
16	Research Battery Powered Fallback	03/15/2015	03/25/2015
17	Research Temperature Sensor Control Module	03/15/2015	03/25/2015
18	Prototype Light Control Module	03/25/2015	04/04/2015
19	Prototype LCD Interface with Base Station	03/25/2015	04/04/2015
20	Prototype Battery Powered fallback	03/25/2015	04/04/2015
21	Prototype Temperature Sensor Control Module	03/25/2015	04/04/2015
22	Research Board Design Software	04/04/2015	04/15/2015
23	Final Documentation Senior Design 1 Complete	04/15/2015	04/25/2015
24	**Senior Design 2**	**05/18/2015**	**08/07/2015**
25	Design Boards	05/18/2015	05/27/2015
26	Bill of Materials Complete	05/18/2015	05/27/2015
27	Send in Order for PCB's	05/27/2015	06/05/2015
28	Prototype Base Station	06/05/2015	06/14/2015
29	Prototype User Acsess	06/14/2015	06/23/2015
30	Prototype Endpoints (lights/outlets/temperature/door)	06/23/2015	07/02/2015
31	Prototype Total Integration	07/02/2015	07/11/2015
32	Design Display	07/11/2015	07/20/2015
33	Build Display	07/20/2015	07/29/2015
34	Add Ordered Boards to the Display	07/29/2015	08/07/2015

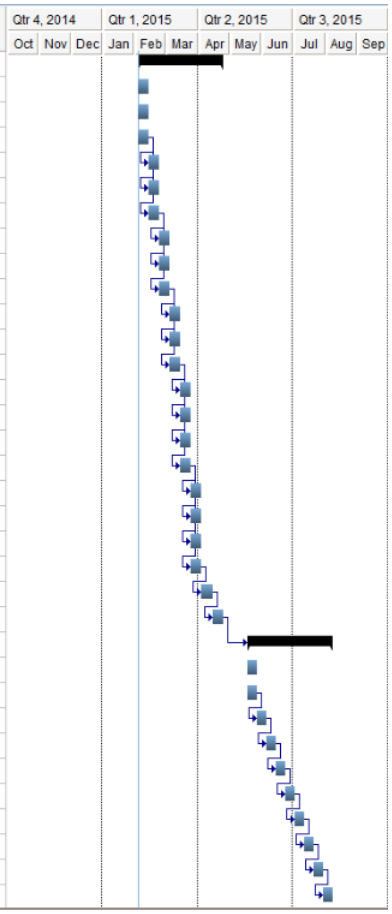
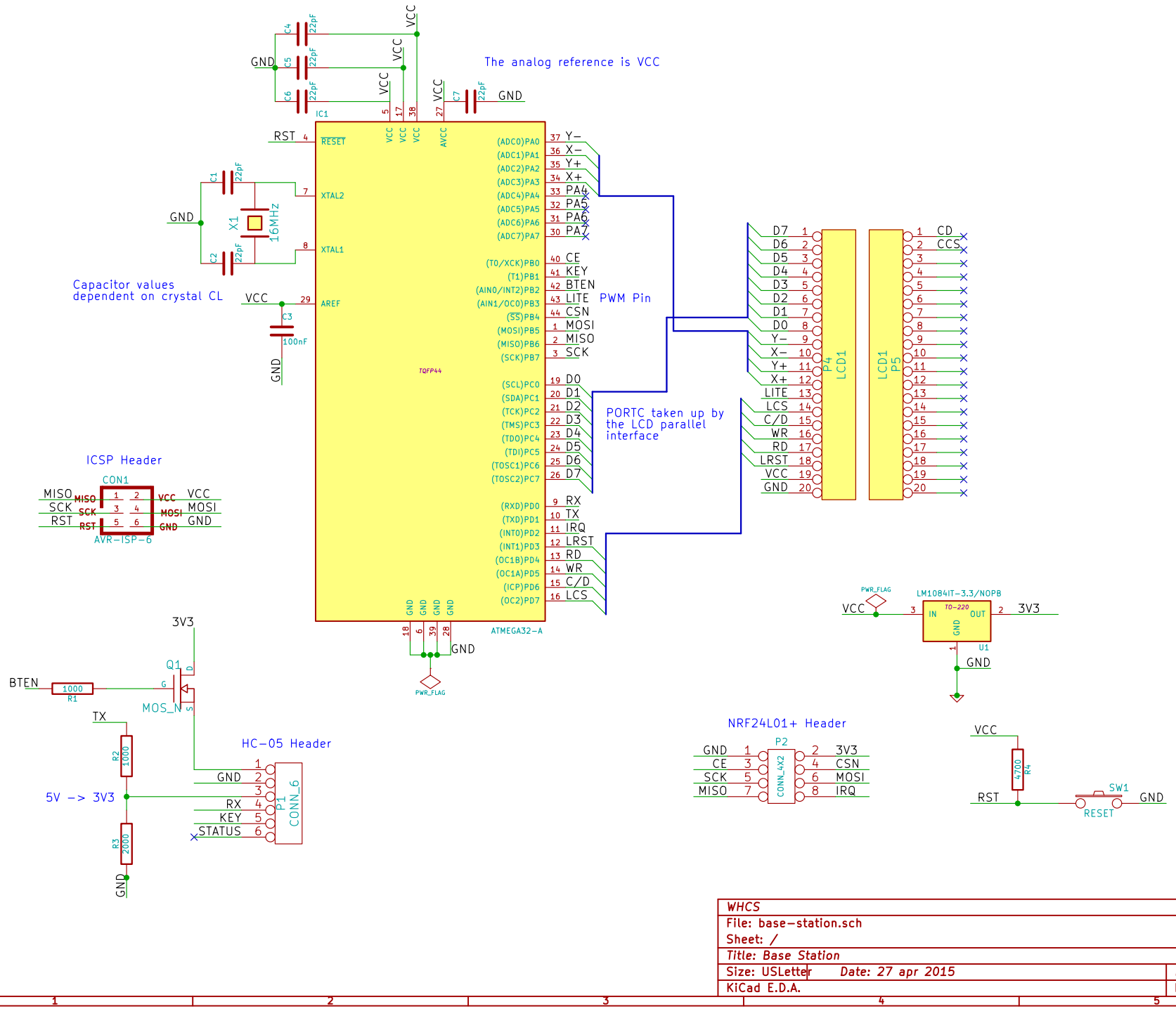
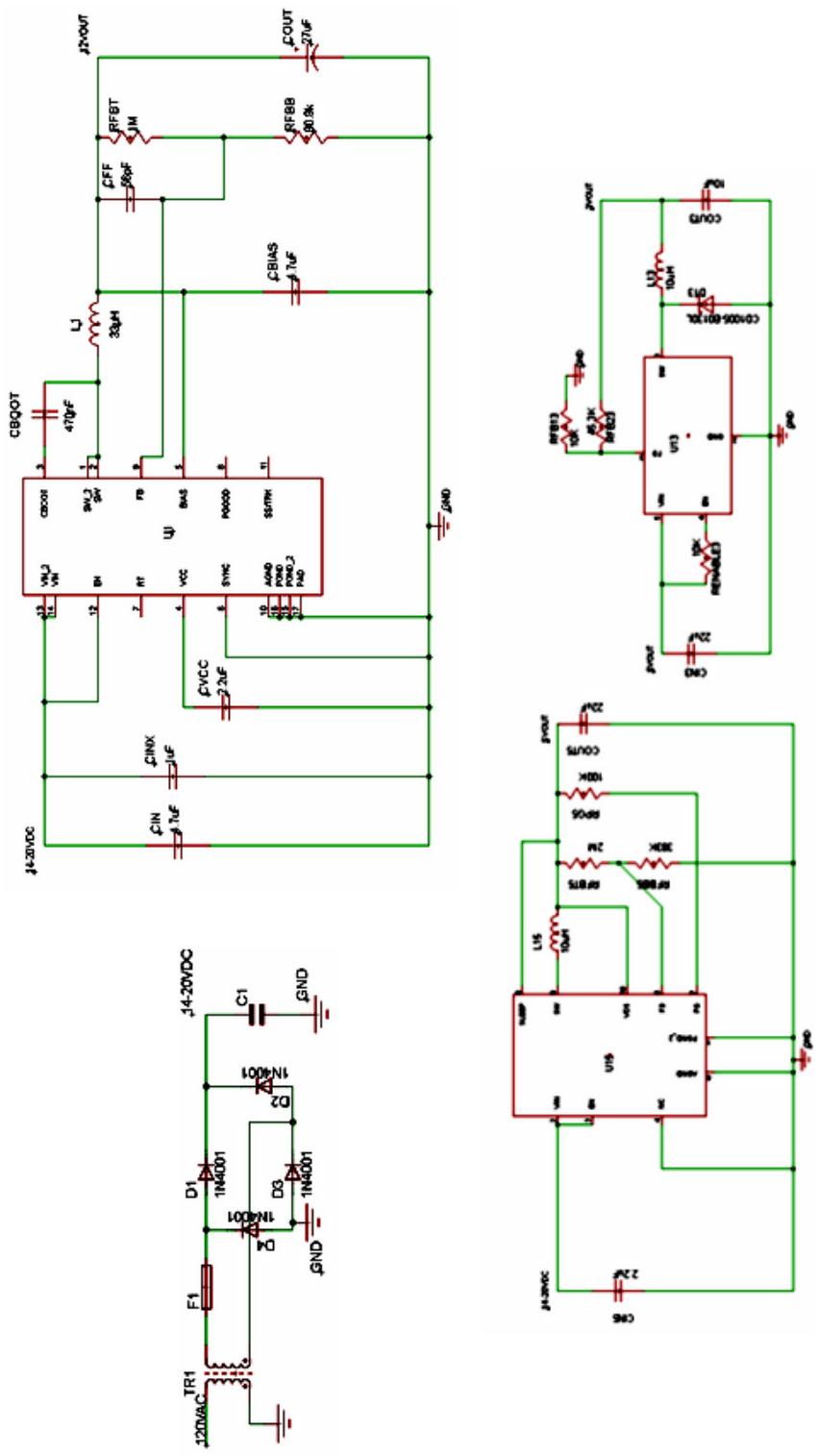


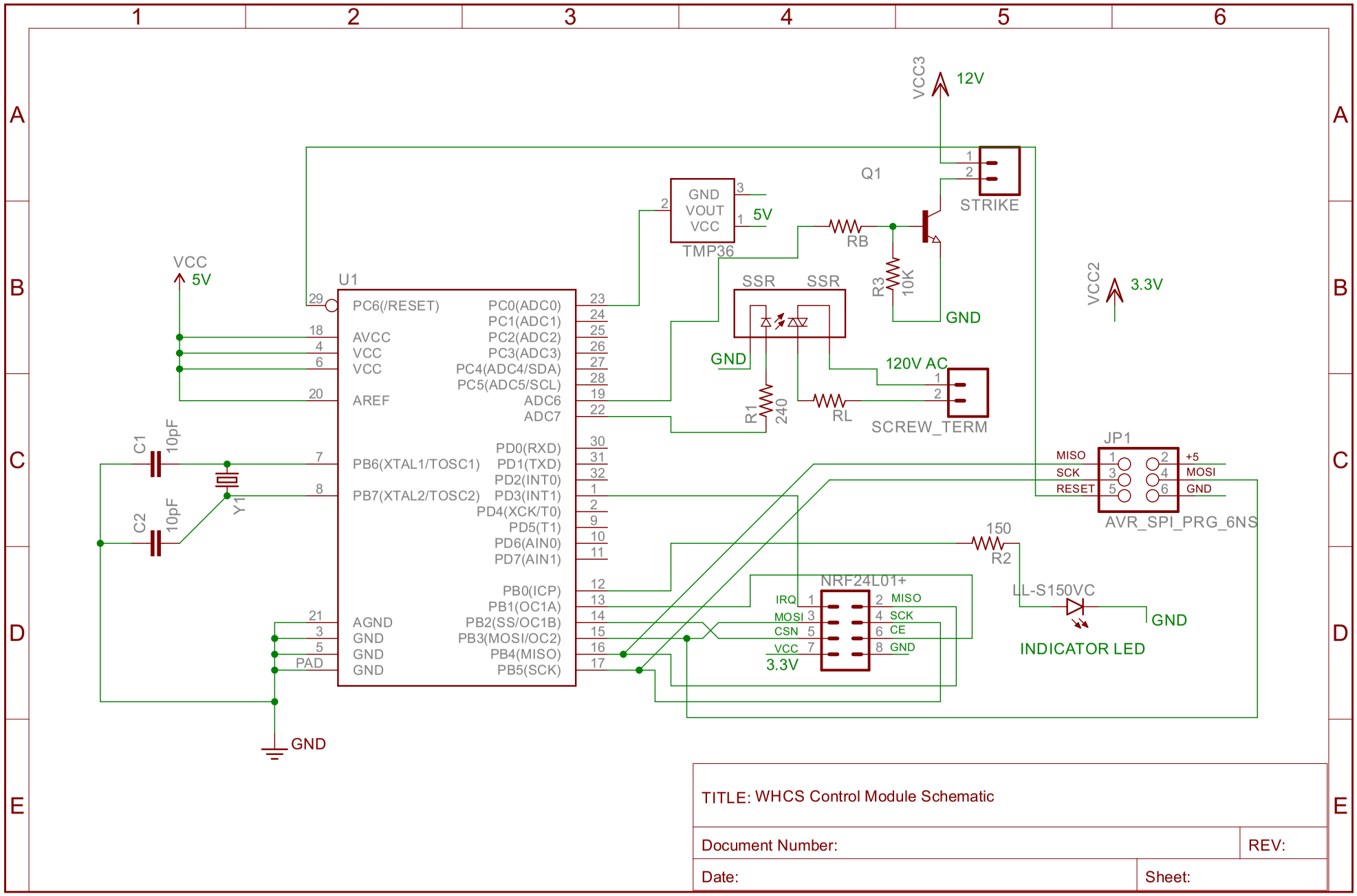
Figure 55: The master gantt chart for the WHCS project

13 User Manual

A Appendix - Complete Schematics


The analog reference is VCC
Capacitor values dependent on crystal CL
PWM Pin
PORTC taken up by the LCD parallel interface
ICSP Header
HC-05 Header
NRF24L01+ Header
5V -> 3V3
ATMEGA32-A
TQFP44
AVR-ISP-6
CONN_6
CONN_4X2
LM1084IT-3.3/NOPB
WHCS
File: base-station.sch
Sheet: /
Title: Base Station
Size: USLetter
Date: 27 apr 2015
Rev: 1
KiCad E.D.A.
Id: 1/1


J2VOUT
COUT
27uF
RFBT
1M
RFBB
80.6k
CFF
68pF
CBIAS
0.1uF
LJ
33µH
CBOOT
470pF
GND
CVCC
2.2uF
CINX
1uF
CIN
1uF
J4-20VDC
C1
1N4001
D2
D1
D3
D4
F1
TR1
120VAC
L15
10µH
L13
10µH


1
2
3
4
5
6
A
B
C
D
E
VCC3
12V
Q1
STRIKE
GND
VOUT
VCC
TMP36
5V
RB
R3
10K
SSR
SSR
GND
VCC
5V
U1
PC6(/RESET)
AVCC
VCC
VCC
AREF
PB6(XTAL1/TOSC1)
PB7(XTAL2/TOSC2)
AGND
GND
GND
PAD
GND
PC0(ADC0)
PC1(ADC1)
PC2(ADC2)
PC3(ADC3)
PC4(ADC4/SDA)
PC5(ADC5/SCL)
ADC6
ADC7
PD0(RXD)
PD1(TXD)
PD2(INT0)
PD3(INT1)
PD4(XCK/T0)
PD5(T1)
PD6(AIN0)
PD7(AIN1)
PB0(ICP)
PB1(OC1A)
PB2(SS/OC1B)
PB3(MOSI/OC2)
PB4(MISO)
PB5(SCK)
C1
10pF
C2
10pF
Y1
R1
240
RL
120V AC
SCREW_TERM
VCC2
3.3V
JP1
MISO
SCK
RESET
+5
MOSI
GND
AVR_SPI_PRG_6NS
150
R2
LL-S150VC
INDICATOR LED
GND
NRF24L01+
IRQ
MOSI
CSN
VCC
3.3V
MISO
SCK
CE
GND
TITLE: WHCS Control Module Schematic
Document Number:
REV:
Date:
Sheet:

B Appendix - Copyright Notices

During the creation of this document, we avoided using any copyrighted works.

C Appendix - References

[1] X10 Website www.x10.com

[2] Fuse Calculated http://www.engbedded.com/fusecalc/

[3] TFT 2.8" LCD https://www.adafruit.com/products/1770

[4] 5V Battery Backup http://www.instructables.com/id/Simple-5v-battery-backup-circuit/

[5] 12V Battery Backup http://electronics.stackexchange.com/questions/96632/12v-battery-backup-supply-for-gprs-tracker

[6] Capacitor Tut http://www.electronics-tutorials.ws/capacitor/cap_2.html

[7] Diode Ref Sheet http://www.diodes.com/datasheets/ds28002.pdf

[8] Solid State Relay http://electronicdesign.com/components/electromechanical-relays-versus-solid-state-each-has-its-place

[9] LDO Dropout http://focus.ti.com/download/trng/docs/seminar/Topic%209%20-%20Understanding%20LDO%20dropout.pdf

[10] ISOCompare http://www.tortech.com.au/isocompare

[11] OSH Park https://oshpark.com/pricing

[12] 4PCB Service http://www.4pcb.com/33-each-pcbs/

[13] Sparkfunhttp://sparkfun.com

[14] Circuit Tutorial http://www.allaboutcircuits.com/vol_5/chpt_2/2.html

[15] Electrical http://homerenovations.about.com/od/electrical/a/artelecbox.htm

[16] GFI Outlet http://diy.stackexchange.com/questions/15684/what-is-a-gfi-outlet-used-for-and-where-should-i-install-them

[17] EAGLE Cadhttp://www.cadsoftusa.com/download-eagle/freeware/

[18] FreeRouter KiCadhttp://www.freerouting.net/

[19] Nest Website http://nest.com

[20] Forbes Article Nest Acquisition http://www.forbes.com/sites/greatspeculations/2014/01/17/googles-strategy-behind-the-3-2-billion-acquisition-of-nest-labs/

[21] Works with Nest https://nest.com/works-with-nest/

[22] ILI9341 http://www.newhavendisplay.com/app_notes/ILI9341.pdf

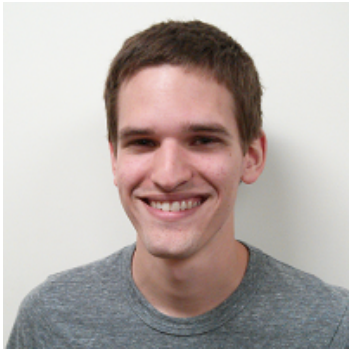
[23] ILI9341 Adafruit Lib https://github.com/adafruit/Adafruit_ILI9341/tree/master/examples

[24] Adafruit 2.8" TFT https://learn.adafruit.com/adafruit-2-dot-8-color-tft-touchscreen-breakout-v2

[25] ILI9341 Fruit https://github.com/adafruit/Adafruit_ILI9341

[26] Bresenham Line Algo https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

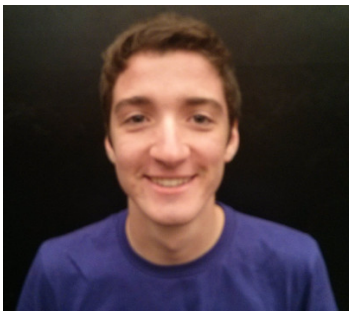
D Appendix - WHCS Team



Grant Hernandez is a senior at the University of Central Florida. He will be graduating with a Bachelor of Science in Computer Engineering this summer. In his spare time, Grant writes lots of code, reverse engineers binaries, plays in cyber Capture the Flag competitions, dabbles in computer graphics, and tinkers with embedded systems. He will be attending the University of Florida in fall 2015 to begin his Ph.D in Computer Engineering with a security research lab.

Jimmy Campbell is a senior at the University of Central Florida. He will be receiving a Bachelor of Science in Computer Engineering in August 2015. His interests include embedded programming, mobile development, and web back-end development. Jimmy will

be taking a full time position with Microsoft as a Software Development Engineer after graduating.



Joseph Love is currently a senior at the University of Central Florida and will receive his Bachelor of Science in Electrical Engineering in August of 2015. He is currently working with Direct Beam Incorporated and plans to pursue his masters in Electrical Engineering during the Fall of 2015 at UCF with a focus in Electromagnetics.