

# Wireless Home Control System

*Reimagine your home™*

Grant Hernandez  
*Computer Engineer*

Joseph Love  
*Electrical Engineer*

Jimmy Campbell  
*Computer Engineer*

University of Central Florida



Senior Design Group #5

# Contents

<b>1</b>	<b>Executive Summary</b>	<b>1</b>
<b>2</b>	<b>Project Description</b>	<b>2</b>
2.1	Motivation . . . . .	2
2.2	Overview . . . . .	2
2.3	Objectives . . . . .	3
2.3.1	Voice Control . . . . .	3
2.3.2	Light Activation . . . . .	4
2.3.3	Outlet Activation . . . . .	4
2.3.4	Door Access . . . . .	4
2.3.5	Data Collection . . . . .	5
2.4	Requirements and Specifications . . . . .	5
2.4.1	Mobile Application . . . . .	6
2.4.2	Radio Module . . . . .	6
2.4.3	Microcontroller . . . . .	6
2.4.4	Bluetooth Module . . . . .	6
2.4.5	Printed Circuit Board . . . . .	7
2.4.6	Power Specifications . . . . .	7
2.4.7	LCD . . . . .	7
2.5	Research of Related Products . . . . .	8
2.5.1	Z-Wave . . . . .	8
2.5.2	Belkin . . . . .	8
2.5.3	Apple HomeKit . . . . .	9
2.5.4	Nest Labs . . . . .	9
2.5.5	X10 . . . . .	10
<b>3</b>	<b>Realistic Design Constraints</b>	<b>10</b>
3.1	Economic Constraints . . . . .	10
3.2	Time Limitations . . . . .	10
3.3	Political Constraints . . . . .	11
3.4	Ethical, Environmental, and Sustainability Constraints . . . . .	11
3.5	Manufacturability Constraints . . . . .	11
3.6	Safety and Security . . . . .	11
3.7	Spectrum Considerations . . . . .	12
<b>4</b>	<b>System Design</b>	<b>12</b>
4.1	Base Station . . . . .	12
4.2	Control Module . . . . .	14
4.3	Bluetooth Capable Phone . . . . .	15
4.4	Software . . . . .	16
<b>5</b>	<b>Summary of Related Standards</b>	<b>17</b>
5.1	RS-232 . . . . .	17

5.2	BlueTooth . . . . .	18
5.3	SPI . . . . .	18
5.4	FR-4 . . . . .	18
5.5	Android Development Guidelines . . . . .	18
5.6	ANSI/NEMA 1-15P, 5-15P, C84 . . . . .	19
<b>6</b>	<b>Hardware and Software Design</b>	<b>19</b>
6.1	Radio Transceiver . . . . .	19
6.1.1	Operating Principles and Usability of NRF24L01+ . . . . .	19
6.1.2	Driver Use Case . . . . .	20
6.1.3	Driver Class Diagram . . . . .	21
6.1.4	Network Library . . . . .	22
6.1.4.1	Modes of operation . . . . .	23
6.1.4.2	Join mode detail . . . . .	24
6.1.4.3	Communicate mode detail . . . . .	24
6.1.4.4	Idle mode detail . . . . .	24
6.1.4.5	Leaving mode detail . . . . .	25
6.2	Microcontrollers . . . . .	25
6.2.1	Microcontroller Brand . . . . .	25
6.2.2	Base Station Microcontroller . . . . .	26
6.2.3	Control Module' Microcontrollers . . . . .	27
6.2.4	Development Environment . . . . .	28
6.2.5	Microcontroller Additions . . . . .	29
6.3	BlueTooth Chip . . . . .	30
6.3.1	RN-41 . . . . .	31
6.3.2	HC-05 . . . . .	31
6.4	LCD . . . . .	32
6.4.1	Capabilities . . . . .	33
6.4.2	ILI9341 Driver . . . . .	34
6.4.2.1	Choosing where to draw . . . . .	34
6.4.2.2	LCD State Management . . . . .	35
6.4.2.3	LCD Performance . . . . .	35
6.4.3	Touchscreen Driver . . . . .	36
6.4.4	Graphics Driver . . . . .	36
6.4.4.1	Algorithms Necessary . . . . .	36
6.4.4.2	Character Lookup Table . . . . .	37
6.4.5	UI Library . . . . .	37
6.4.5.1	ButtonView . . . . .	37
6.4.5.2	TextView . . . . .	37
6.4.5.3	ListView . . . . .	37
6.5	Android Application . . . . .	37
6.5.1	Development Environment . . . . .	37
6.5.2	Use Case Diagram . . . . .	38
6.5.3	Speech Recognition . . . . .	39
6.5.4	BlueTooth Software Design . . . . .	40
6.5.5	GUI Philosophy . . . . .	42
6.5.6	BlueTooth Listener Class . . . . .	44
6.6	Power Hardware . . . . .	45

6.6.1	Design Summary . . . . .	46
6.6.2	Power Consumption . . . . .	48
6.6.3	DC-to-DC Converters vs. Linear Voltage Regulators . . . . .	49
6.6.4	Backup Battery Configuration . . . . .	50
6.6.5	Transformer Choice . . . . .	52
6.6.6	Rectifiers, Diodes, Capacitors . . . . .	53
6.6.6.1	Rectifier . . . . .	53
6.6.6.2	Capacitor . . . . .	54
6.6.6.3	Diodes . . . . .	54
6.6.6.4	Relay . . . . .	55
6.6.6.5	Linear Regulators . . . . .	55
6.6.7	Isolation . . . . .	56
6.6.8	Simulation Testing . . . . .	56
6.6.9	Power Through Hole Board . . . . .	56
6.6.10	Schematic . . . . .	56
6.7	Base Station . . . . .	56
6.7.1	Software Flow . . . . .	57
6.7.2	Control Module Abstraction . . . . .	59
6.7.3	Subsystems . . . . .	60
6.7.3.1	NRF24L01+: Interaction . . . . .	60
6.7.3.2	HC-05: Interaction . . . . .	61
6.7.3.3	LCD: Interaction . . . . .	62
6.7.3.4	Touchpanel: Interaction . . . . .	63
6.7.3.5	Timers: Interaction . . . . .	63
6.7.3.6	Networking State Machine . . . . .	63
6.7.4	Schematic Breakdown . . . . .	63
6.8	Control Module . . . . .	63
6.8.1	Software Flowchart . . . . .	63
6.8.2	High-Voltage Control . . . . .	65
6.8.3	Electronic Strike . . . . .	65
6.8.3.1	Normally Open or Normally Closed . . . . .	66
6.8.3.2	Strike vs Deadbolt . . . . .	66
6.8.4	Sensor Data Collection . . . . .	67
6.8.5	Light and Outlet Control . . . . .	68
6.8.6	Schematic Breakdown . . . . .	69
<b>7</b>	<b>Printed Circuit Board</b> . . . . .	<b>71</b>
7.1	Software Considerations . . . . .	71
7.1.1	EAGLE . . . . .	71
7.1.2	KiCad . . . . .	72
<b>8</b>	<b>Prototyping</b> . . . . .	<b>72</b>
8.1	Point-To-Point Transmission . . . . .	72
8.2	Rogers Board Etching Prototyping . . . . .	74
8.3	WHCS Proto-Panel . . . . .	74
8.3.1	Materials . . . . .	74
8.3.2	Dimensions . . . . .	75
8.3.3	Sketch . . . . .	75

<b>9</b>	<b>Manufacturing</b>	<b>76</b>
9.1	PCB House . . . . .	76
9.1.1	OSH Park . . . . .	76
9.1.2	Sseed Studio . . . . .	76
9.1.3	4PCB . . . . .	76
9.2	Parts . . . . .	76
9.2.1	Footprint (SMD vs Through-Hole) . . . . .	76
9.3	Construction . . . . .	77
9.3.1	Soldering . . . . .	77
9.3.2	Reflow Oven . . . . .	77
9.3.3	Proto-Panel . . . . .	77
<b>10</b>	<b>Testing</b>	<b>77</b>
10.1	Power Supply . . . . .	77
10.1.1	5v Line Integrity . . . . .	77
10.1.2	120v Line Integrity . . . . .	77
10.1.3	3.3v Line Integrity . . . . .	77
10.1.4	Battery Backup . . . . .	77
10.2	Base Station . . . . .	77
10.2.1	LCD Control . . . . .	77
10.2.2	LCD and NRF Simultaneous . . . . .	77
10.2.3	UART and Software Serial . . . . .	77
10.3	Control Module . . . . .	78
10.3.1	Voltage Level Correct . . . . .	78
10.3.2	UART Chip Testing/Debugging . . . . .	78
10.3.3	Command Execution . . . . .	79
10.4	Door Access . . . . .	79
10.5	Android To Base Station Communication . . . . .	79
10.5.1	BlueTerm . . . . .	79
10.5.2	BlueToothListener . . . . .	80
10.5.3	LED activation test . . . . .	80
10.6	Base Station to Control Module Communication . . . . .	81
10.6.1	LED Toggle . . . . .	81
10.6.2	UART to Hyperterm . . . . .	81
<b>11</b>	<b>Demos</b>	<b>81</b>
11.1	Voice Controlled Light Activation . . . . .	81
11.2	LCD Light Activation . . . . .	82
11.3	Sensor Query . . . . .	82
11.4	Fault Recovery (Loss of Power) . . . . .	82
11.5	Remote Door Access . . . . .	82
<b>12</b>	<b>Project Management</b>	<b>83</b>
12.1	Budget . . . . .	83
12.2	Parts Acquisition . . . . .	83
12.3	Milestones . . . . .	83
<b>13</b>	<b>Appendix</b>	<b>83</b>

13.1	Appendix A - Figures . . . . .	83
13.2	Appendix B - Tables . . . . .	83
13.3	Appendix C - Complete Schematics . . . . .	83
13.4	Appendix D - Copyright Notices . . . . .	83

## List of Figures






















1	WHCS System Overview . . . . .	3
2	A illustration showing the translation of many different sensor nodes to WHCS' protocol . . . . .	5
3	Base Station Exterior Design for WHCS . . . . .	13
4	Base Station PCB Block Diagram . . . . .	14
5	Control Module Block Diagram . . . . .	15
6	WHCS Software Block Diagram . . . . .	17
7	NRF Driver Use-case Diagram . . . . .	20
8	NRF Class Diagram . . . . .	21
9	the overall state machine for the network library . . . . .	23
10	Microcontroller Crystal Schematic . . . . .	29
11	a high level outline of the LCD pin configuration and specifications . . . . .	34
12	Android App Use-case diagram . . . . .	39
13	Android app speech activation chart . . . . .	40
14	Android Bluetooth Startup Flowchart . . . . .	41
15	Visual of Communication Between Android Device and Base Station . . . . .	42
16	Android GUI Layout . . . . .	43
17	cmAdapter Class Diagram . . . . .	44
18	BluetoothListener Class Along With Supporting Data Structures . . . . .	45
19	Baseline design for power board . . . . .	46
20	Additions to the baseline design for the implementation of light and outlet control module boards . . . . .	46
21	Additions to the baseline design for the implementation of door access module board . . . . .	47
22	Additions to the baseline design for the implementation of the base station board . . . . .	48
23	chosen backup-battery configuration <sup>1</sup> . . . . .	50
24	another backup battery configuration <sup>2</sup> . . . . .	52
25	the high level software flow for the base station . . . . .	57
26	showing the control module hierarchy for WHCS . . . . .	59
27	an example sequence diagram that could occur between a connected blue-tooth phone and the base station . . . . .	61
28	the level of abstractions for the LCD subsystem . . . . .	62
29	the high level software flow for a generic control module . . . . .	64
30	servo motor access control design . . . . .	65
31	electric strike entry methods . . . . .	66
32	Temperature Sensor Connection Schematic . . . . .	67
33	Wiring Schematic for Solid State Relay . . . . .	69
34	WHCS Control Module Schematic . . . . .	70
35	Control Module Indicator LED . . . . .	71

36	Point to Point Transmission Prototyping Setup . . . . .	73
37	General design of WHCS display board . . . . .	76

## List of Tables

1	comparison of ATmega chips . . . . .	27
2	Comparison of Atmel Studio and WinAVR . . . . .	29
3	Comparison of Two Different AVR Programmers . . . . .	30
4	comparison of the BlueTooth chips . . . . .	32
5	a brief summary of the pertinent features of the LCD module . . . . .	33
6	Currents and voltages of devices used in WHCS . . . . .	49
7	a tabularization of control module roles and the available commands . . . . .	79
8	LED Activation Test Commands . . . . .	81

## Todo list

	Fix footnotes in captions and list of figure hyperlinks . . . . .	i
	3 pages . . . . .	3
	make a section for scheduling or remove . . . . .	4
	find some better examples . . . . .	4
	cite article hyping up how much power you lose to leeches . . . . .	4
	0.5 pages . . . . .	9
	0.5 pages . . . . .	10
	0.5 pages . . . . .	11
	3 pages . . . . .	22
	Figure: Join mode state machine . . . . .	24
	Figure: Communicate mode state machine . . . . .	24
	Figure: Idle mode state machine . . . . .	25
	Figure: Leaving mode state machine . . . . .	25
	whole section . . . . .	32
	1 page . . . . .	33
	2 page . . . . .	34
	Figure: class diagram . . . . .	35
	1 page . . . . .	36
	Figure: diagram showing the touchscreen coordinates . . . . .	36
	2 page . . . . .	37
	1 more page if possible . . . . .	71
	1 page . . . . .	76
	this isn't complete from Joseph . . . . .	76
	1 page . . . . .	77
	1 page . . . . .	82
	1 page . . . . .	82
	Must have an “old style” reference section . . . . .	84

## 1 Executive Summary

The Wireless Home Control System is meant to be a solution for any homeowner to be able to remotely control core appliances of their home. The WHCS vision is to allow 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 will allow users to monitor the state of the installed components and activate them remotely. A central base station equipped with a touch-enabled LCD will be present to allow the users to interact with the system without the need of a phone. Peripheral control modules will be installed into the targeted appliances such as lights, outlets, and doors for WHCS to control. Together the phone, base station, and peripheral control modules will constitute WHCS.

The implementation of such a system requires research in a myriad of fields to produce a full-fledged product. Android software development is the core for creating the users first impression with WHCS. Research must be conducted to conform to the design philosophies of the Android ecosystem. The alternative interface offered for WHCS will be the base station's display, as a result the best solution for a touch capable LCD will be examined. Communication devices will form the foundation for the wireless aspect of WHCS, thus an investigation into the advantages of different communication modules is required to realize the system. A network protocol will need to be planned out 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 are a requirement. To extend upon harnessing the home's power, our individual control modules and base station's logic level voltage depend upon the creation of an efficient way to step down the high voltage supplied to the home. Efficient forms of rectification and power provision will need to be researched for WHCS to be self-sufficient. Important research and critical design decisions will go into the development of the printed circuit boards that will provide the foundation for all the hardware of WHCS.

Wireless Home Control System is a solution targeting the masses and designed by few. Naturally such a system will suffer from the constraints imposed upon the creators. Most prominent of all constraints are those stemming from economics. The development and production of WHCS will have to conform to the low budget available. Design decisions will be made to minimize overall cost of the system to satisfy this constraint. WHCS has the potential for mass implementation if user reception is positive so the design will adhere to manufacturing constraints. The parts used in the system will be chosen so that they are widely available. Our boards and parts will be 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 should 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 will have to be done in a safe manner. Security is another strong constraint attributed to products targeting the home. The system will be designed to maximize security for our users.



## 2 Project Description

### 2.1 Motivation

The goal of this project is 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 want to develop 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 is for the solution to be reliable and low cost. The use case scenarios should be intuitive so that even someone who was just visiting could utilize the system. A person using WHCS should be able to turn on their lights or an outlet with the press of a button or with a voice command from their mobile phone. They should also be able to turn on their coffee pot from their phone when they first wake up. If someone knocks on the door the person should be able to 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 will have 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 option will provide full capabilities for interacting with the system. The phone will be attached to the system through a BlueTooth connection that is created by the user in the WHCS application. Using the phone will be 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 will be the brains of WHCS. It will be 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 will have a list of endpoints, also called control modules, that can be targeted by the system. This list will be dynamic and allow for endpoints to be added or removed from the system during operation. Together the base station and the control modules will form a network through a home and will communicate wirelessly to one another through radio transceivers.

The control modules designed for WHCS will allow for all the activation of appliances around the house. These endpoints will be listening for commands from the base station via a radio transceiver. Each control module will be tailored for interacting with a certain device. There will be control modules for toggling outlets, toggling lights, unlocking the front door, and

also for monitoring sensors. The control modules will be as similar as possible with a designated area that allows for assigning specific roles to the control modules.

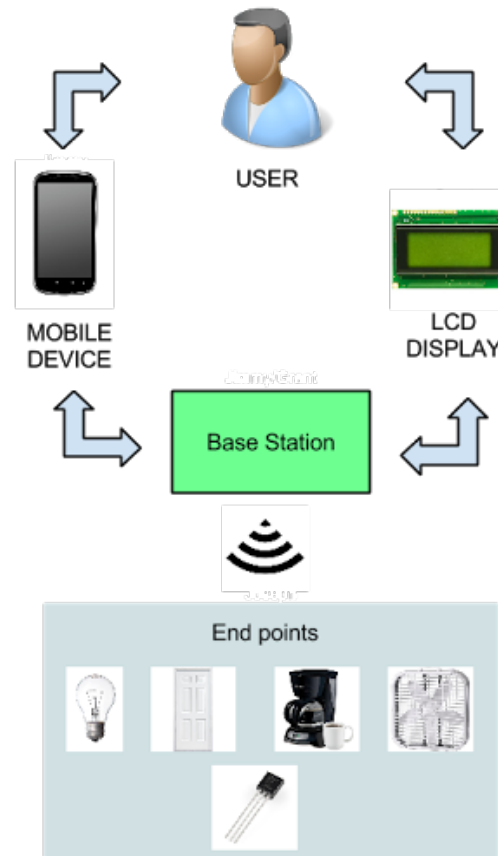


Figure 1: WHCS System Overview

## 2.3 Objectives

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

3 pages

### 2.3.1 Voice Control

Voice control from a supported, BlueTooth enabled, Android device will allow 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.<sup>3</sup> All of these *actions* and *targets* will be able to be used just from the user's voice. Voice actions will be specific to each target, but they will consist of verbs such as **turn**, **query**, **check**, **open**, **close**, and so on. The list of targets will directly correspond to the number of control

modules listed in the home and their type. This will be explained in more detail in [Section 4](#).

### 2.3.2 Light Activation

Through activating lights and querying their status remotely, a homeowner will no longer have to be present in the same room as the switch. By connecting lights to WHCS, they will 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 will be controlled. By automating the process of toggling light switches, WHCS will have the ability to be smart about when they are ON or OFF, freeing the user from having to think about their state at all. In addition, lights, just like all of the other connected control modules in the home, will be able to be actuated on a specific schedule. This schedule can be designed by the homeowner to meet their daily lives or in a special circumstance, such as travel.

make a section  
for scheduling  
or remove

### 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.

find some bet-  
ter examples

cite article  
hyping up how  
much power  
you lose to  
leeches

### 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 would provide a specific control module the capability of locking and unlocking a door. This functionality would be 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. This is why any control module whose purpose is to control doors will support additional security features. These additional features will include mechanisms to prevent replay attacks (which garage doors are vulnerable to) and also prevent outside attackers from engaging the door opening mechanism just by sniffing traffic. For additional details on the security considerations of WHCS, please refer to [Section 3.6](#).

---

<sup>3</sup>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.

### 2.3.5 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 would be transparent to the user. All they would see would be the list of sensors and the corresponding values. WHCS's pluggable control module's would be 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 would support queries from the LCD interface and simultaneously from a connected phone.

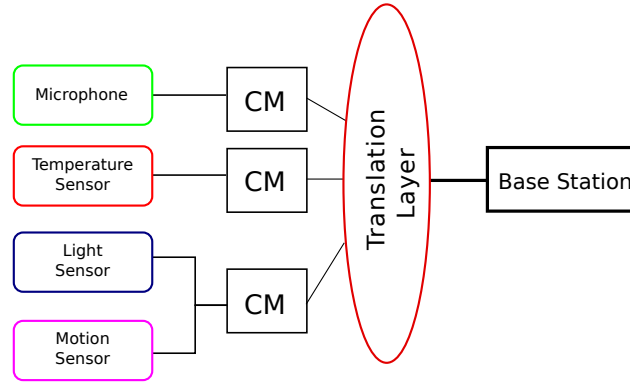


Figure 2: A illustration showing the translation of many different sensor nodes to WHCS' protocol

Beyond home sensors, all of the other controllable objects in the WHCS would have meta-data being collected about them at all times. This extra metadata would 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.

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.

## 2.4 Requirements and Specifications

For WHCS to become the product that we want it to be it must adhere to a strict set of requirements and specifications. We have identified certain requirements for subsystems involved in our system that we must fulfill in the development process. The following section is devoted to identifying requirements that we must 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 sub-systems 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**

### **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 will feature radio transceivers that use a proprietary protocol to exchange information. Only components within our system will be able to communicate within the network. The network will also allow 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 will also implement a feature similar to this. With the ability to control multiple devices around the home it will be 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 will be achievable through the hardware that WHCS will create 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 will be doing, 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 will 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 will leave 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 will feature 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](#) 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<sub>2</sub> 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](#), 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™*. This strategy is clever as now the success of the company will grow with every new developer who

0.5 pages



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

0.5 pages

## 3 Realistic Design Constraints

### 3.1 Economic Constraints

Economic constraints are the biggest hindrance in the development of WHCS. The amount of money necessary to complete such a project adds up quickly due to the necessity of obtaining hardware like printed circuit boards. This project is being developed by college students so the amount of reserve money that is 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 will have to be paid for out of pocket. This will especially be an issue if things go wrong during the development process that cause us to have to repurchase certain pieces of hardware. In all of our design time decisions cost plays the biggest factor. Our research and development budget is small compared to mass-produced products similar to WHCS. We have recognized this constraint from the offset and we plan accordingly.

### 3.2 Time Limitations

The amount of time that we have to create WHCS is a limiting factor for what we are trying to create. To begin with, the technologies used in this project are 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 are not just creating WHCS we are 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 have to endure, we will be missing valuable weeks due to the summer semester. Our second semester of senior design will take place in the summer semester which is 12 weeks instead of the normal 16 weeks of a semester. This means that we lose out on almost an entire month due to the second half of development being in the summer. This whole month of lost time will put a large strain on the group and it is something that we have accounted for early on.

### 3.3 Political Constraints

There are no relevant political constraints that we have attributed to the development of WHCS. The development of WHCS is 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 have had to make during the development of our system involve 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 is to take the path which results 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 believe that making decisions in this area to help the environment and promote sustainability give us the best ethical outlook. This is also the only ethical involvement we have attributed to our project.

### 3.5 Manufacturability Constraints

WHCS will have the potential to be implemented in many homes as long as consideration is put into manufacturability during design time. As we make decisions for our system we are constantly monitoring how it affects the overall ease of replication. We have identified the base station PCB and control module PCBs to be the biggest contributors of our overall manufacturability. In order to maximize manufacturability we have constrained ourselves to only using highly available parts for incorporation into our PCB designs. Following through with this commitment will ensure that WHCS installations will be standardized and easy to replicate.

### 3.6 Safety and Security

0.5 pages

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 6.8.3](#).

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 preemptive *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.

### 3.7 Spectrum Considerations

For WHCS we will need to provide 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 provides 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 will be involved in everything that is done with WHCS. Any command going to an endpoint will pass through the base station and any information coming from an endpoint will go

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 and will have an option for battery backup. The user will need to be able to interface with the base station in order to control the system. To facilitate this interaction there will be a mobile WHCS application that can be used to communicate to the base station, and there will also be a touch enabled LCD. Both of these methods will be available to control the base station. The base station will be in the shape of a rectangular prism and will have the LCD flush with the surface of the top of the prism. This way the PCB can be hidden away and does not have to be known to the user. This will provide an aesthetically pleasing look to the base station and hide away its inner workings. Figure 3 shows what the base station will look like including the LCD, the inner PCB, and the plug for power. The size of the base station will be based off of the size required for the printed circuit board.

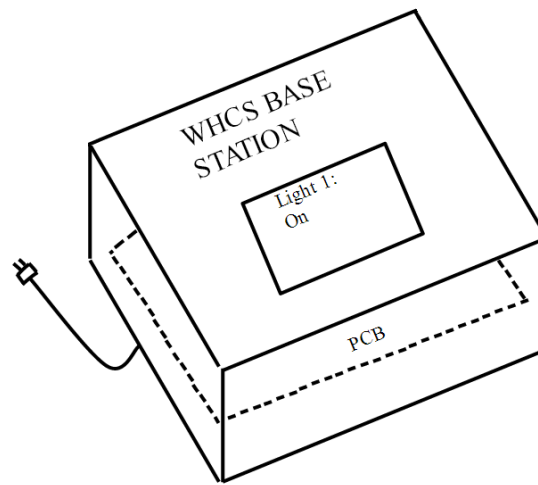


Figure 3: Base Station Exterior Design for WHCS

The base station will require a special printed circuit board that is different from the control modules. It will be the only board in the system that has to support Bluetooth communication. This is how the base station will communicate with the Android phone. It will also need to have space for the installation of the LCD screen. The LCD screen will be able to be mounted on the base station PCB so it can be pushed through the display box. Using this architecture the LCD will be all the user has to see. Like the control modules the base station will have a radio transceiver chip. This will be the part of the base station that allows it to communicate to the control modules who will have a similar radio transceiver. The LCD, Bluetooth chip, and radio transceiver will have to connect to the microcontroller that is chosen for the base station. This whole system will be powered by a 120V AC step down circuit and a battery backup. The base station will also have 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.

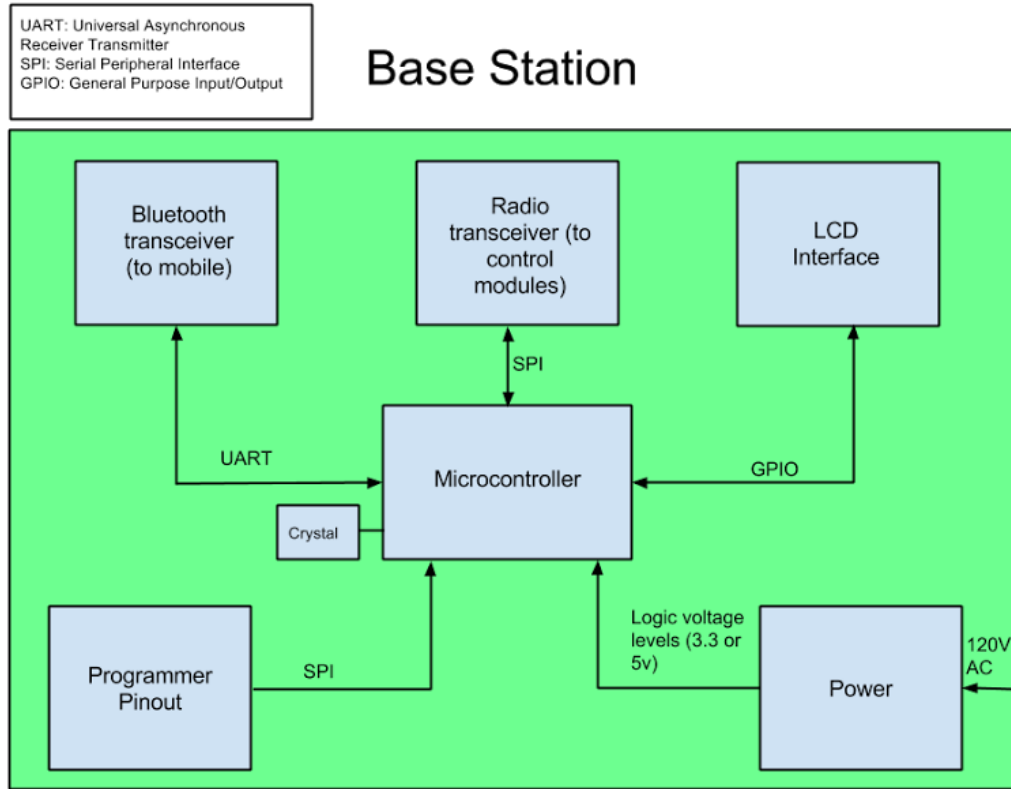


Figure 4: Base Station PCB Block Diagram

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 will 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 will be 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 will need to have a power supply circuit, a microcontroller, and a radio transceiver at a minimum. The rest of the circuitry will revolve around what the control module is meant to interact with. For example the circuit to activate the electronic strike on a door will need to switch the electronic strike power, while a temperature sensor circuit will need to do analog to digital conversion. There are two options that these possibilities bring about for the creation of control module boards. One option is 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 means 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 is 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 can custom solder components based on what that control module is being used for. Either of these options work. 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 consider 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.

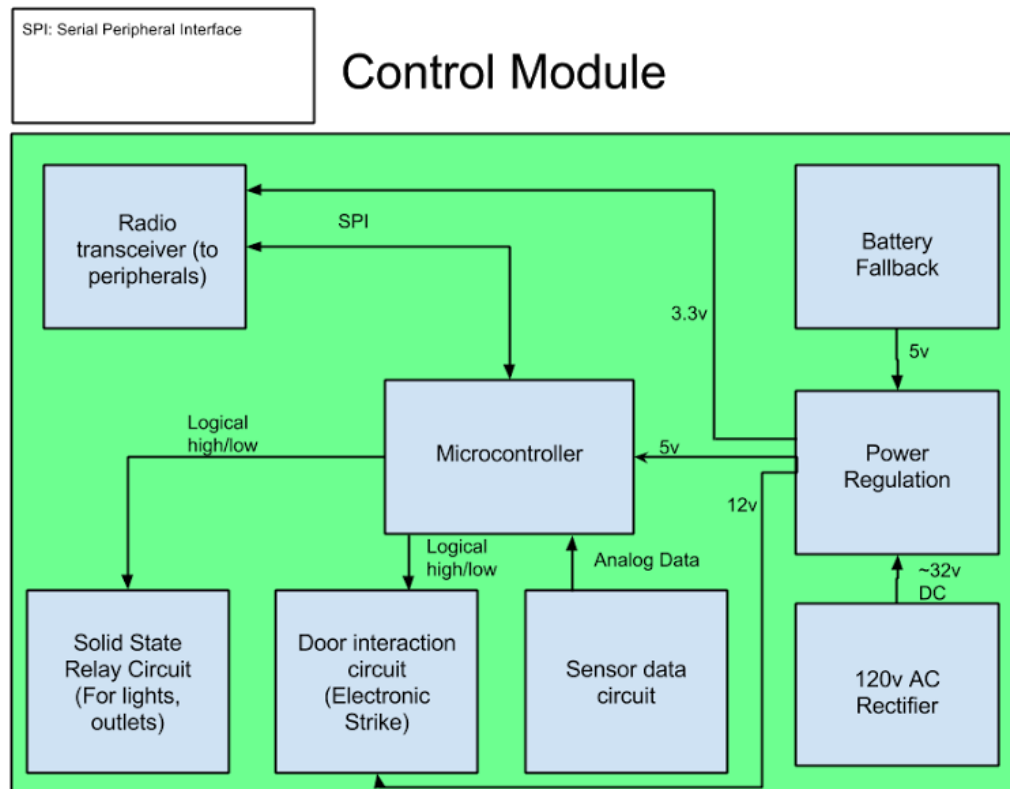


Figure 5: Control Module Block Diagram

### 4.3 Bluetooth Capable Phone

The Wireless Home Control System is based around the capability of interacting with the system from a mobile phone. A phone with Bluetooth communication capabilities is required for the system to operate at its full capabilities. With the selected Bluetooth

phone platform, a software application can be developed to allow users to interact with WHCS. It will be through this application that users can monitor the state of their home and remotely interact with targeted appliances. The phone application will feature a low-complexity graphical user interface as well as the ability to use speech-recognition activation.

## 4.4 Software

WHCS will contain a network consisting of its base station, its peripheral control modules, and the user's Bluetooth capable mobile phone. This creates the need for three distinct software models. These three different software implementations will follow a custom communication protocol in order to transmit data throughout the network. Commands will 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 will be done through Bluetooth. The phone will be 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 will relay on a custom driver written for the radio transceiver used in the system. The base station will be the powerhouse in terms of software design. It is at this central hub that the collection of all active control modules will be stored. The base station will also constantly have to be ready to accept requests from any control module or the user. The main routine will need 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 will also need 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 will have to be custom tailored based on the role of the control module. The routines running on the microcontrollers will 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 will need 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 will be as simple as toggling the state of a solid state relay connecting to a GPIO pin. It will be the communication between subsystems that presents the biggest challenge for the microcontroller based boards.

The mobile application software will have 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 will be the most involved piece of software. The interface will have to constantly display the state of the system and provide intuitive use cases. All of the capabilities that we create for WHCS will have to be reflected by the user-interface that we design. The U.I. will need to be linked together with the Bluetooth communication handler because at the user will be expecting state changes in the system at the press of a button.

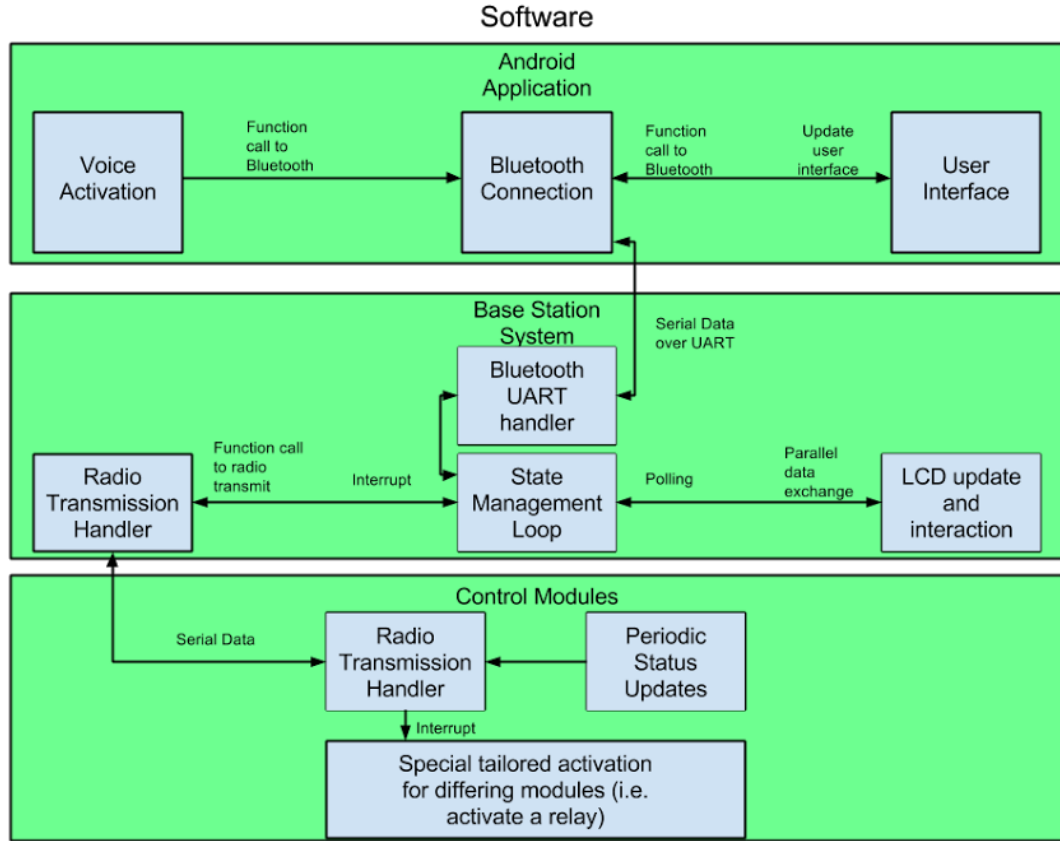


Figure 6: WHCS Software Block Diagram

## 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 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 notable 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 is Nordic Semiconductor's NRF24L01+. This chip meets 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 is 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 are using only ranges from 2.4GHz to 2.5GHz we will not be able to use all of the NRF's available channels. With the NRF we are capable of sending payloads with a maximum size of 32 bytes per transmission from module to module. We will be 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 gives us flexibility for implementing our network because we 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 will 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 simplifies the work necessary for creating our own network of NRF chips. We will be 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 will be 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 will require a 3.3v line. The datasheet lists the current consumption while in receive mode as 18mA. This will be 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 fits 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 it to transmit or receive at that moment, and an interrupt pin for notifying the microcontroller of important situations. The csn pin will allow 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 will take up 6 pins whilst three of the pins will be shareable with other SPI components.

### 6.1.2 Driver Use Case

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

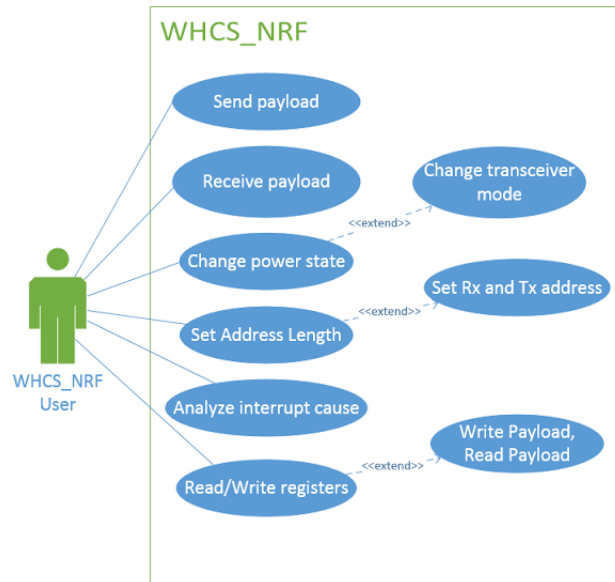


Figure 7: NRF Driver Use-case Diagram

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 will be reading and writing registers. Everything will build 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 will be special forms of writing to a register. Thus reading and writing to registers is a use of the driver, however it will be abstracted in a way that provides ease of use. A user of the NRF driver will spend most of the time setting addresses, writing payloads, transmitting payloads, and analyzing interrupts. These use cases need to be implemented perfectly to provide a strong foundation for the networking of WHCS.

### 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 will be using Atmel ATmega microcontrollers in WHCS so C++ is supported as a development language. Using C++ allows 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 gives 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 will rely on the proper assignment of these pins.

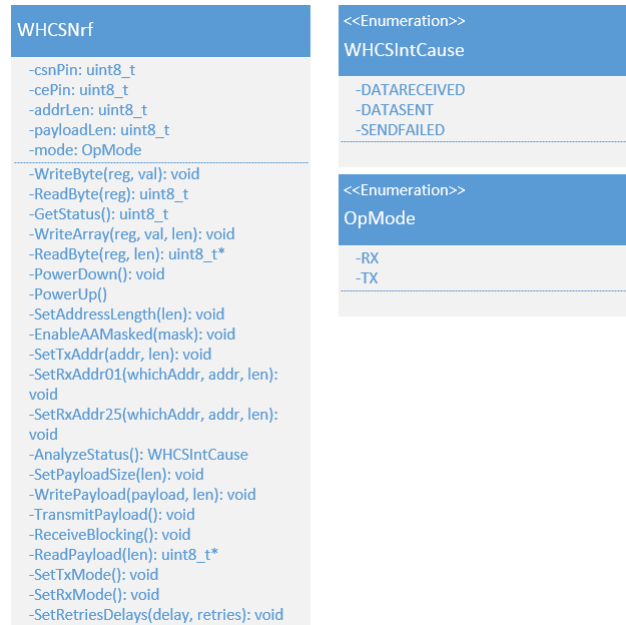


Figure 8: NRF Class Diagram

Usage of the NRF driver will involve first constructing the class by telling the microcontroller which pins the NRF is connected to. Then the user will be doing 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 can be done with the call of a function with simple parameters. Setting the address for receiving and transmitting data can be 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.

#### 6.1.4 Network Library

3 pages

In order for WHCS to be networked, a set of networking functions will have to be constructed on top of the radio driver to form a network library or **NetLib**. Abstraction of the raw driver will allow for the concept of a **node** to be created. The network will consist of a set of nodes capable of direct communication and reception of **logical packets**. These packets will contain metadata describing their purpose and the data that they contain. By dividing up the required network features of WHCS in to packets, we will have a library of datagrams. These 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.

Similar to network protocols today, WHCS will implement 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" would be 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 would be similar to User Datagram Protocol, which is "best effort" in terms of reliability. We plan on adding additional code and sequencing to allow for this unreliable medium to withstand lost packets. The NRF already provides the ability to retransmit on a non-acked frame, but this will just fail after a certain amount of attempts. This would handle transient errors, but not a complete loss of connectivity, which would occur if the base station lost power.

In order to keep the network topology simple, we opt to have a simple module to base network. This means modules will 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 do not need a complicated mesh network as all of the nodes will be 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 will do 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 could be done along the direction of a full mesh network, but only if necessary for good performance. The direct benefit of this simple architecture is that is easy to test and easy to program. The logic will only expect transactions to occur between two nodes. That fact will drive 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.

**6.1.4.1 Modes of operation** Before diving in to the specific functions required for implementing a robust NetLib, we will 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 will be in the **joining** mode when it is first powered on or in the event of a disassociation. The node will 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 will have its own state machine describing the required processes for association on the WHCS network. The **communicating** mode will be active if the node is joined to a network and if there are data to be sent or received. Otherwise the node will be in the **idling** state, which will include 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 will free 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 9.

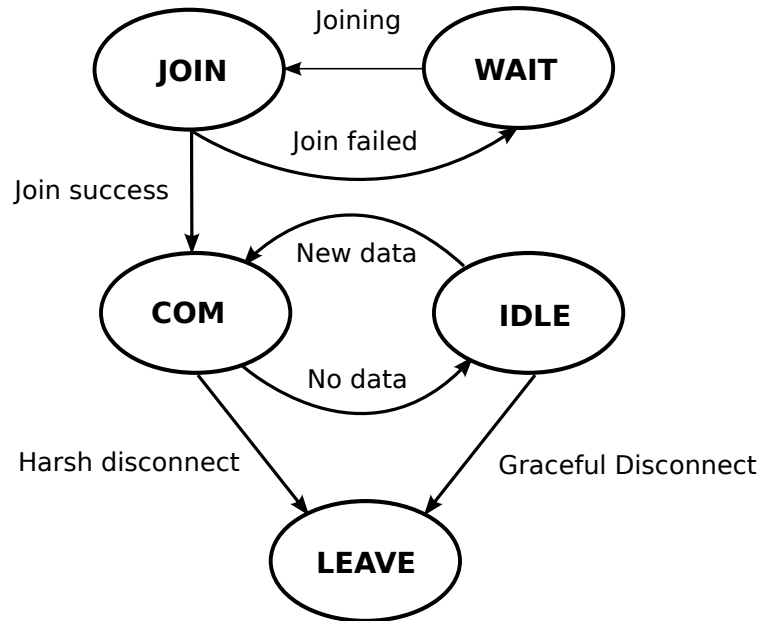
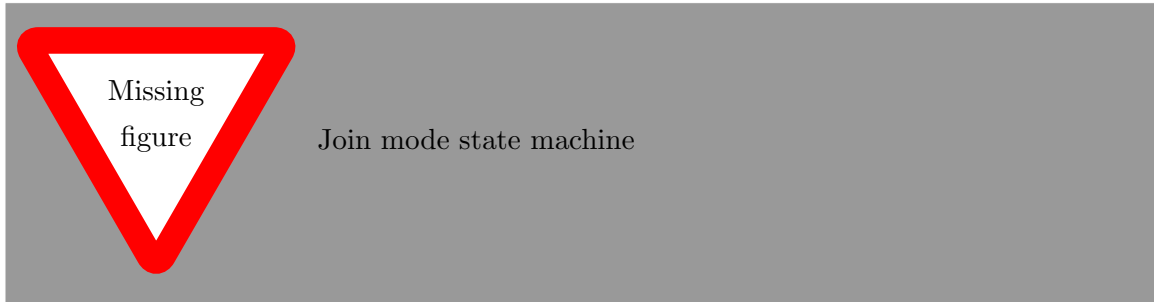


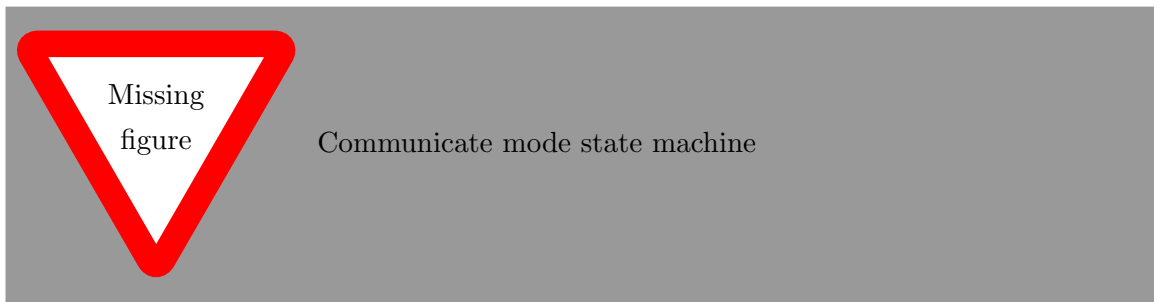
Figure 9: the overall state machine for the network library

**6.1.4.2 Join mode detail** The join mode will be fired when a new node wants to associate to an existing WHCS network as maintained by a base station. The base station will be the arbiter of all communication for the WHCS network and as such will also handle 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. An optional (as per the setup configuration) will be communicated during a join request. The base station will validate the joining node and grant or deny access to the WHCS network. This process is necessary to provide some assurance and tracking for the active nodes on the network.



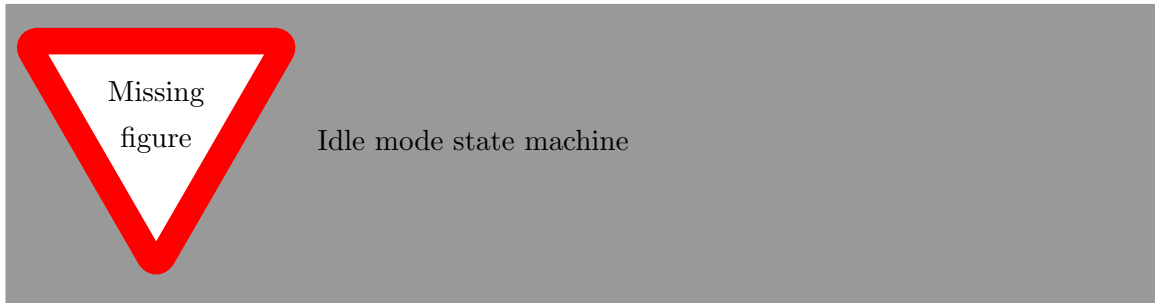
**6.1.4.3 Communicate mode detail** Once a node has been successfully joined to the WHCS network, it will be able to communicate directly with the base station as its network gateway. All communication will flow through this node, which will process the data, and if required, generate a response. This 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 will provide some periodic or event information about the transmitting node. This could include arbitrary data and is parsed based off of the node type. For example, a temperature node will have a specific data format that the base station will know how to unmarshal and store.

The overall flow of the communicate mode will be to pump out packets to be sent, wait for any responses, and generate any actions as required by the packets. This loop will be performed by any node with a radio. The base station is the only special case as it has to receive from multiple nodes simultaneously.

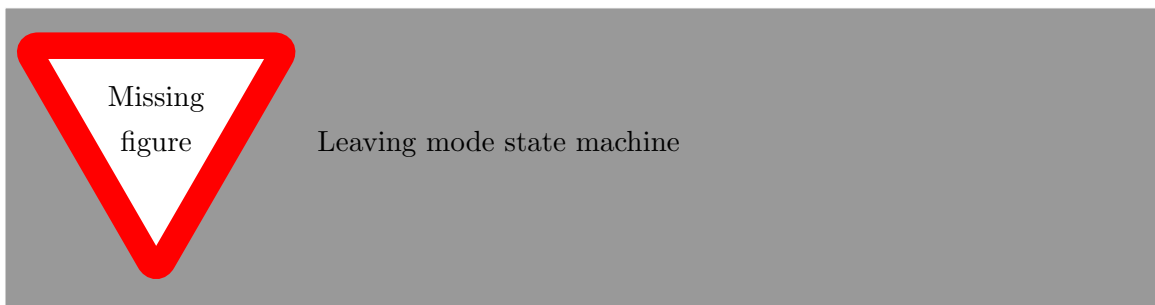


**6.1.4.4 Idle mode detail** The idle mode is the simplest mode for the NetLib. This state is where the library has no data to send or receive. During this mode, the underlying

radio is free to sleep in order to conserve power usage. Also interrupts would be 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 would be woken up and enter in to the communication mode.



**6.1.4.5 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 will disassociate from the WHCS network thereby causing its state and all of the resources associated with the join to be freed. Additional cleanup will be performed in order to get the library ready for a new join request. Any outstanding packet sends or received will be canceled harshly and any potential data will be lost. A safer shutdown would assert that the radio must be in the idle mode. That way no data will be lost to the leave transition.



## 6.2 Microcontrollers

For the proper operation of WHCS microcontrollers will need 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 are 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 they 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 use on the base station and the control modules are operating at their full capabilities we will be adding external crystals to their circuits. The external crystals will boost 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 have decided to use the NDK NX5032GA-16.000000MHZ-LN-CD-1 as the crystal for our microcontrollers. This microcontroller has a 16 MHz operating frequency and has the simple two connection design. For adding the crystal to our microcontroller we will be using the circuit pictured in [Figure 10](#). 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.

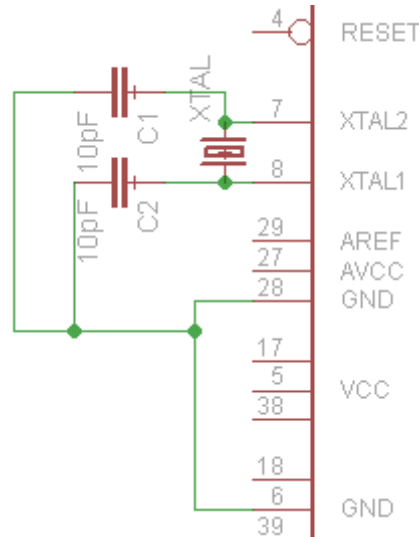


Figure 10: Microcontroller Crystal Schematic

The equation 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.

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

To make the external crystals work with the microcontrollers, certain fuses must be set through a programmer connected to the microcontroller. The datasheets for the microcontrollers have a fuse list that can be 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 <http://www.engbedded.com/fusecalc/> 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 right needs 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 is sure to work when ordered however it also comes with a large price tag compared to other options. The online vendor SparkFun offers a product called AVR Pocket Programmer for \$15 that accomplishes all that we need 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 will enable 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 Option	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 would be step up from traditional “dumb” thermostats. With a simple LCD combined with a touchscreen, users would have a way to control and query their home without having to find their phone.

To make this a reality, we have chosen the versatile, premade, and well supported Adafruit 2.8” TFT LCD<sup>4</sup> with a resistive touchscreen. Internally, the display is driven by the feature-

whole section

rich [ILI9341 chipset](#). 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](#) and Adafruit's [excellent technical documentation](#). This combined with Adafruit's [libraries](#), assure that integrating the LCD will be straight forward. The one issue with this solution is with the ILI9341 driver code: it was written to target the Arduino platform. Now, the Arduino platform is fairly close to bare AVR, minus the remapped pin numbers and some support libraries, so porting Adafruit's library would be a feasible solution. Our plan is to use the existing driver along with the chip datasheet to create a library specific to our needs. That way we get full control over the port placement and the experience of creating an LCD driver.

### 6.4.1 Capabilities

In terms of capabilities, we have summarized the main features of the LCD module in [Table 5](#)

1 page

Specification	Description
Resolution	240x360
Colors	262K @ 18bits, 65K @ 16bits
Voltage Input	3.3 - 5V
Weight	40 grams
Dimensions (LCD itself)	2.8" diagonal
MCU Interface	Multiple. See <a href="#">Section 6.4.2</a>
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 will be doing will be non-realtime. Nearly all of the displayed information will be text and UI element, 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 will be using the 8-bit interface to avoid having to wait for lengthy SPI transfers. Also, our NRF radio will be using the SPI bus, which should have priority over that channel. You may view a high level overview of the LCD module in [Figure 11](#).

<sup>4</sup><https://www.adafruit.com/products/1770>



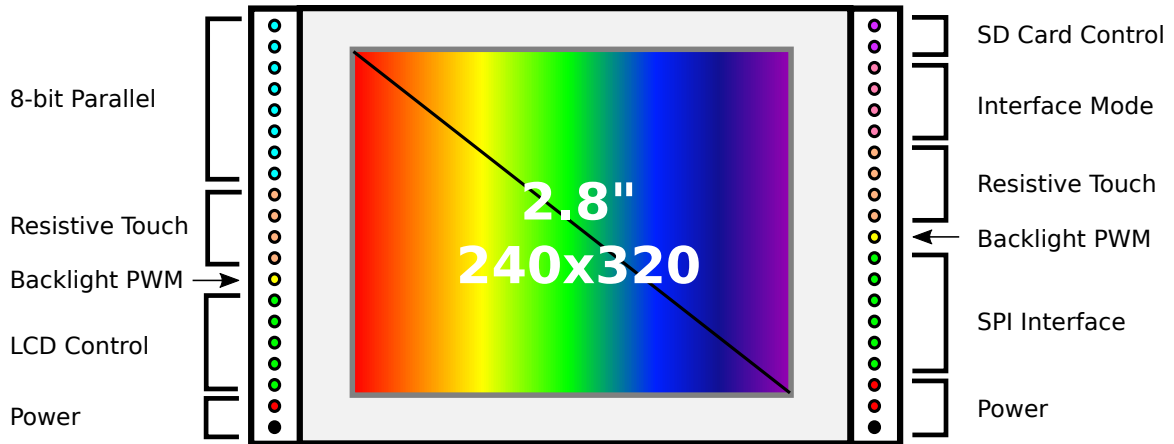


Figure 11: a high level outline of the LCD pin configuration and specifications

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 commands as well. Using this simple interface, we plan on creating 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 would be able to calculate the position of a single finger.

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 plan on storing at least our logo, but we may also store fancy UI badges such as arrows ( $\leftarrow$ ,  $\rightarrow$ ), X's ( $\otimes$ ), checkboxes ( $\checkmark$ ), and anything else we can think of.

## 6.4.2 ILI9341 Driver

2 page

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

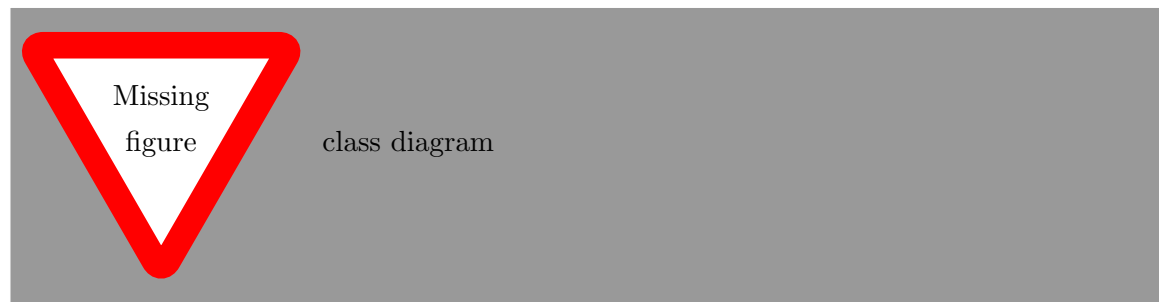
**6.4.2.1 Choosing where to draw** The ILI9341 works like many chips in which it has a command set for controlling display parameters. One of these commands will set the window in which pixel data is drawn. Before a batch drawing operation, a window is selected by specifying the column and page addresses. These are set using the **Column Address Set (0x2A)** and the **Page Address Set (0x2B)** commands. From this point, the RAMWR command is send 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 6.4.2.3](#).

**6.4.2.2 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 will be on wall power, being able to dim the screen and save energy would be a useful feature.

**6.4.2.3 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 should never take priority over more important operations, such as sending and receiving 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 don't have that luxury – instead we must consider performance before we hit any performance walls.

The slowest part about managing the LCD are 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. For setting just one arbitrary pixel 104 bits would need 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.

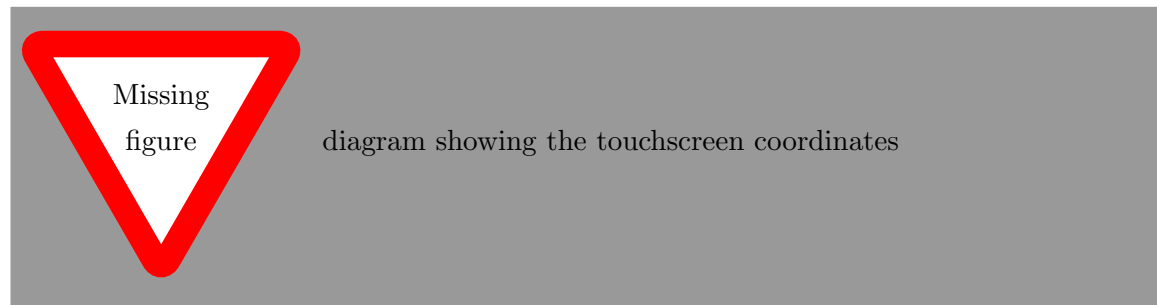
One other hit to potential performance is the lack of double buffering. The LCD chip only has enough RAM for one set of graphics memory, which means active drawing occurs on the viewing surface. In modern day graphics pipelines, double buffering and even *triple* buffering are common place as they eliminate 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 will need code dedicated to polling the X+, X-, Y+, and Y- pins of the LCD. These will give voltage readings that can be used to calculate the position of a single touch. They can be read by using the built-in ATmega ADC. Due to their not being a dedicated touch controller, interrupts cannot be used to sense sharp changes. This will add to the processing time of the microcontroller's main loop as it will constantly have to perform ADC operations and comparisons.

1 page



### 6.4.4 Graphics Driver

Unlike the previous ILI9341 driver, our graphics driver will be 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 will 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, but for the experience and control of designing the graphics routines, we choose to implement our own.

Due to the unique hardware and software constraints (i.e limited clock speed and memory), we must take care to not exceed the capabilities of the hardware. This means floating point operations, which may be required for shapes such as a circle must be optimized or not used at all. A quick optimization for floating point operations would be to use a *sin()* lookup table and only integer multiplications. These performance issues will be addressed as needed and only if necessary. By avoiding unnecessary optimizations, we will save valuable time for building out the library.

**6.4.4.1 Algorithms Necessary** The functionality of graphics driver depends on some core algorithms for efficiently creating meaningful screen objects. One of these core algorithms is [Bresenham's line algorithm](#). This algorithm needs to be efficiently implemented as it will be 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.

If there were any need for UI elements to be at different depths, then each UI element would have to have its

**6.4.4.2 Character Lookup Table** In order to convey useful information, we will need to display text to the user. This text will 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, we will use an existing font available online. Adafruit also bundles a font in with their graphics library that is implemented as a function instead of a block of memory.

## 6.4.5 UI Library

### 6.4.5.1 ButtonView

**6.4.5.2 TextView** Properties: length (uint16\_t), width (uint16\_t), depth (uint8\_t), text (16 bytes),

### 6.4.5.3 ListView

## 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 wanted to make an easy to use application that would attract users to stick with our system. Operability and usability were emphasized in our design process. We wanted 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 had in mind 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 are 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. Connecting to the base station the first time you use the application is a necessary use case that must be incorporated into the application. [Figure 12](#) shows the use case diagram for the WHCS application.

The design for the WHCS application involves making sure that performing the common use cases such as checking status and toggling endpoints are very fast. The user should be able to perform these tasks without having any knowledge of how the application works. Speech recognition will be a supporting feature so it does 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 will involve pressing a button to prompt the speech recognizer, and then giving a command to the WHCS. In order to make the speech activation feature more promising, the user will have the ability to rename endpoints for activation. Creating endpoint groups will be a feature that is not used frequently but adds a lot of value to the application. Users will only have to create an endpoint group once for it to last in the application. Creating an endpoint group will be a simple task involving assigning a group number to endpoints. That number will be the endpoint group, then that endpoint groups state can be toggled.

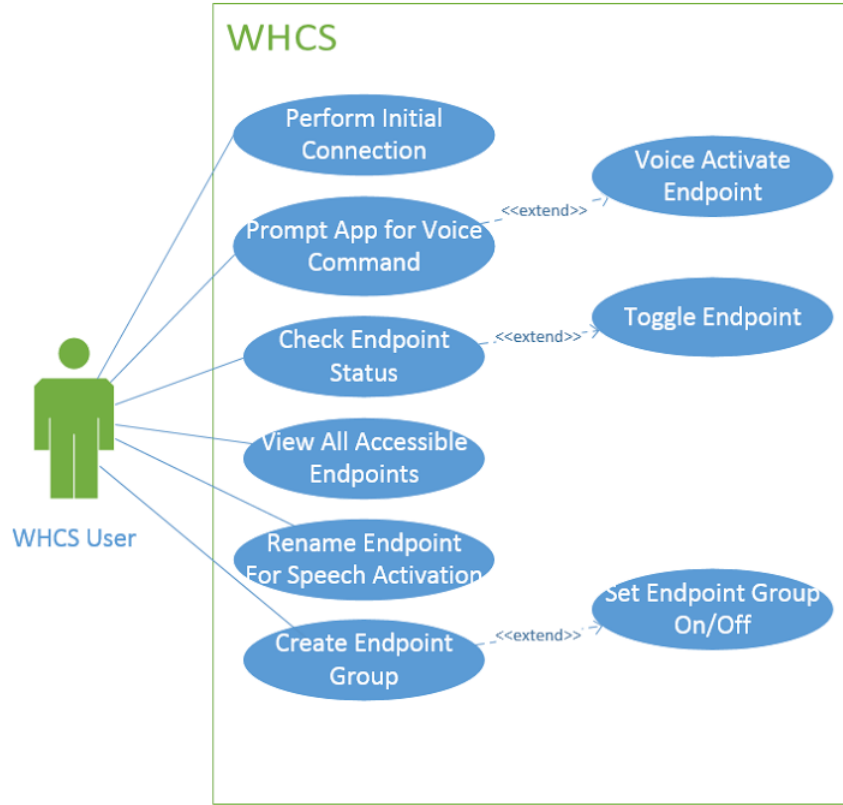


Figure 12: Android App Use-case diagram

### 6.5.3 Speech Recognition

The Android application for WHCS will offer speech activation capabilities. These will be on top of GUI activation capabilities. The speech activation sequence begins with the press of a button to start the speech recognition. The user will be prompted with a microphone and can then give his command. The commands will be formatted like “light one on.” When the user gives commands using the speech method, a notification will be 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 will be shown back to the user to show what went wrong. We are predicting 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 will display the command to the user and then perform it. The following flow chart in [Figure 13](#) displays the sequence of events happening when a user performs speech activation.

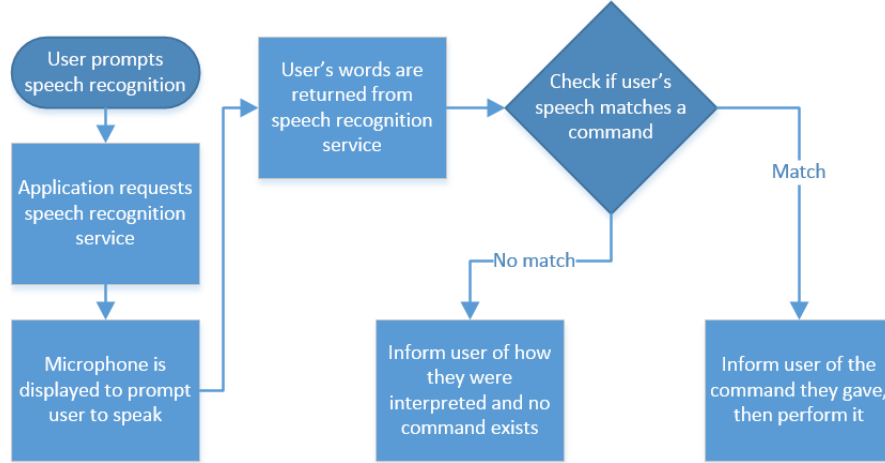


Figure 13: Android app speech activation chart

The goal of the speech activation feature is to be easy to use. In order to promote the usage of this feature we will add the ability for users to rename the endpoints that the speech commands will target. For example the user could change “light 1” into “living room light.” This way the user could say “living room light on” to the application in order to turn on the living room light. To do this data structures will 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 should be stored when the application is closed so a permanent source of storage is needed to do this. The file system can be used or possibly a SQLite database.

In the code for our application we will be using 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 will request this service to be run by using an Android construct called an intent, specifically the recognizer intent. Once the 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 endeavour. 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 will be the technology that allows WHCS users to interact with the base station from the mobile phone. This means that proper functioning BlueTooth software must be written to ensure that users can interact with WHCS. From the user’s standpoint the only knowledge of BlueTooth required will be 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 will be 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 14 shows what the BlueTooth software will be doing whenever the user opens the Android application.

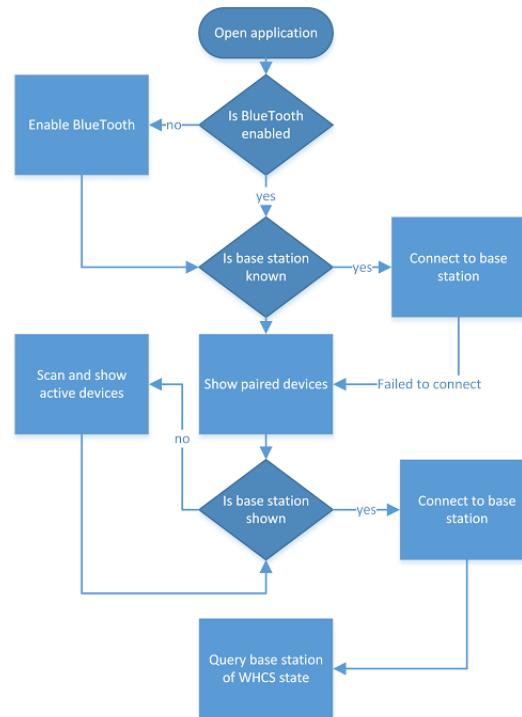


Figure 14: Android BlueTooth Startup Flowchart

In Figure 14 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 will prompt to the user to turn it on and if they refuse we will 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 will first show 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 will be able to search for active BlueTooth devices and select the base station. At the end of this start up cycle the WHCS application will have an active BlueTooth connection with the base station that can be used for full duplex communication.

Our application will be leveraging 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 ap-



plication scans for devices or asks the user to pick an option from the list of paired devices this will be to get the BluetoothDevice to open a socket from. Once we have obtained that BluetoothDevice we can create a BluetoothSocket through one of its methods. Once a BluetoothSocket has been opened through calling connect, an input and output stream 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 want. All data that we send or receive from the base station over BlueTooth will be in the form of a byte array. This form of primitive data transmission allows us to implement certain communication protocols between the Android base station.

Once a BluetoothSocket has been opened on the Android device the application can begin communicating with the base station. We will use a communication protocol between the Android device to ensure the base station can properly interact with the application. This protocol will allow 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 will create a packet with a certain structure. The packet will contain 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 will receive one byte at a time due to the serial nature of BlueTooth communication but it will be able to parse the packets it receives in order to figure out what action the application is trying to perform. [Figure 15](#) shows a visual representation of the communication between the application and the base station.

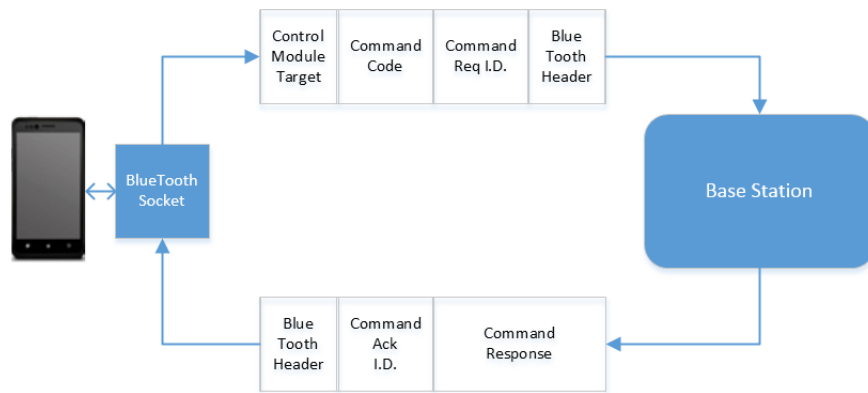


Figure 15: Visual of Communication Between Android Device and Base Station

### 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 is 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 top right layout in [Figure 16](#) shows the view that

would list all of the accessible control modules in the system.

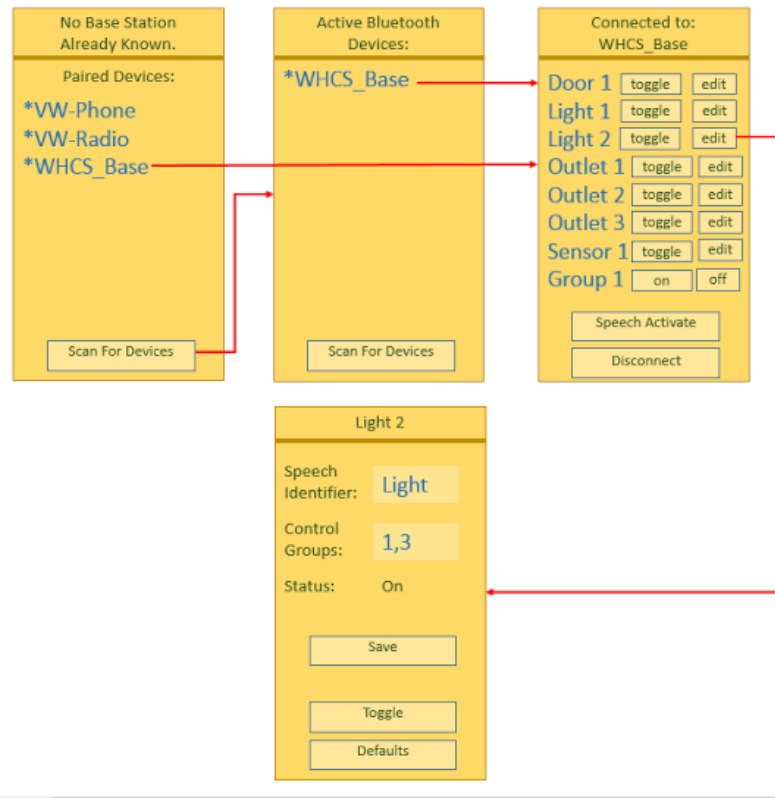


Figure 16: Android GUI Layout

As shown in Figure 16 there are layouts that provide support around the main list layout. The first two layouts in the upper left and upper middle are the 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 these screens 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 will be achievable by clicking on the name of a control module or by clicking the edit button. The individual detail viewer will allow 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 will also list 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 will be using separate activities for each of the screens. This is a logical approach because the layouts in our application are independent of one another. The main list layout will serve as the root activity and any other screen will be an activity that is placed upon it. For example when the application opens up the first time it will try to open the list activity but

will notice that it is not connected to the base station. This will cause the paired devices activity to stack on top of the list activity. When the base station is selected the paired devices activity can return the result of the selection to the list activity and the list activity can function normally. When the detail viewer activity is called it stacks on top of the list activity and when the user is done with it, it will be removed off of the stack.

To make our list activity look clean and function effectively we will create 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 will be `cmAdapter`. The `cmAdapter` that we create will have 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` will know how to get the data from a control module object necessary to populate the main list. The main list activity will constantly call the `getRow` method that will be 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 17](#) shows the class diagram for the `cmAdapter`. The class is simple but provides important functionality for the Android application.

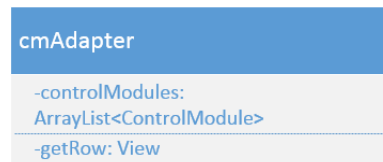


Figure 17: `cmAdapter` Class Diagram

### 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 want 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 18](#) shows the class diagram for the `BlueToothListener` class as well as the classes associated with it.

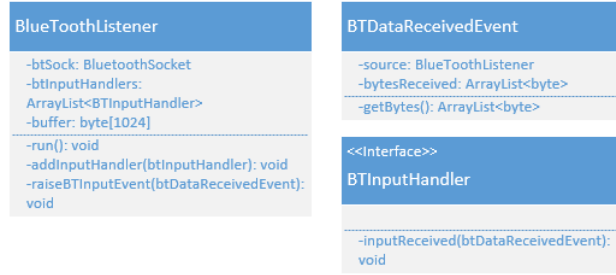


Figure 18: `BluetoothListener` Class Along With Supporting Data Structures

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 need 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.

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 need 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 19 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 24VAC this 24VAC is rectified using diodes into 33.6 VDC. At this point the line goes through a DC to DC converter that will transform the 33.6 V to 5V. This 5V line leaves the power board and enters the PCB containing the microcontroller and turns it on.

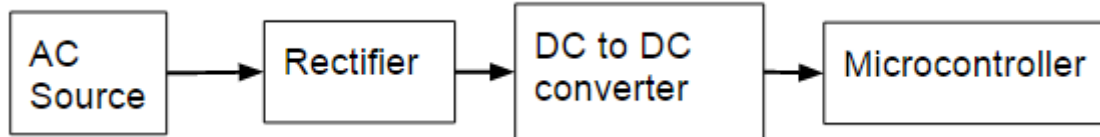


Figure 19: Baseline design for power board

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 will have a line to power the actual application (the light and/or outlet). This is shown in Figure 20 shown below. This extra line will run along our power board and will only be 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. 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.

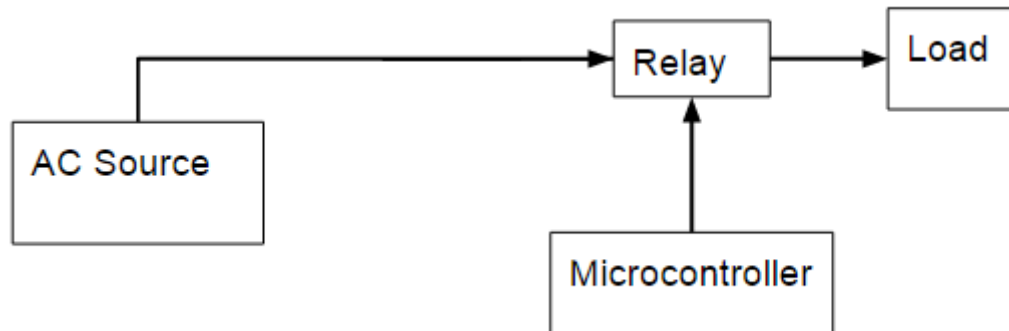


Figure 20: Additions to the baseline design for the implementation of light and outlet control module boards

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 21](#) 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. Another detail about this board that is not shown in the figure is that it has a backup battery. The reason for this is that we still want the microcontroller to be powered even if there is a power failure from the 120VAC home power. The backup battery will not provide power to the 12V line meaning the strike will remain in locked mode. However having the microcontroller powered will allow it continue completing tasks such as checking the state of the system.

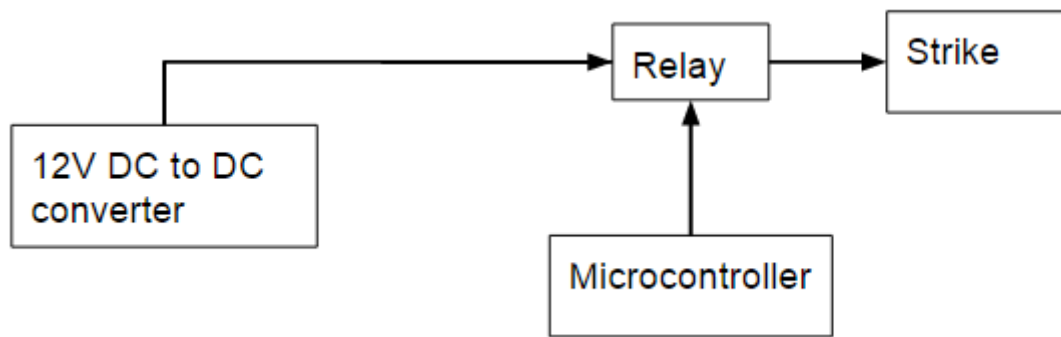


Figure 21: Additions to the baseline design for the implementation of door access module board

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 needs the basic design of the black line that provides the microcontroller with 5V. Yet it too like the green and blue line is equipped with a back up battery. This way the information about the temperature of the home can 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 22](#). In order to make this step down we will be using a linear regulator. Like the temperature sensor module and the door access modules, the base station will also be equipped with a backup battery in case of power failure. This will allow the base station to be fully operational even if the power goes out.

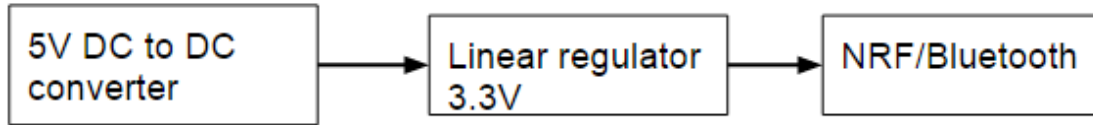


Figure 22: Additions to the baseline design for the implementation of the base station board

The only reason that the light and outlet modules are not equipped 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 is 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 where we decided on each component that we selected.

In [Table 6](#) below we have all the information on the amount of current that is to be 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 it was the maximum speed for the Atmega32A. Although the Atmega328P can run at a higher clock speed of 20 MHz, we decided to make 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
<b>Atmega32A</b>	4.5-5.5V	12-16mA
<b>Atmega328P</b>	4-5.5V	Active: 7-11mA Savings: 1.75-2.75mA
<b>Electric Strike</b>	12V	450mA
<b>Adafruit TFT LCD</b>	3.3V	150mA
<b>Bluetooth HC 05</b>	1.8-3.6V	35mA
<b>nRF24L01+</b>	1.9-3.6V	13.5mA
<b>TMP 35 36 37</b>	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 will run in active mode, however since our microcontroller has the capability to run 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 will not be running 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 will be 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. Not because of smart design choices, but mostly because the things that our project must execute does not require a lot of power. Based on the fact that  $\text{Power} = \text{Voltage} * \text{Current}$  and that not all these devices will be used at a time. We can see that the devices themselves will never really draw that much power.

Now that this data has been presented it can be 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 can be 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.

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

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 (mostly because of the backup battery) we decided to have a step down from 34 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 makes most sense for us to use a DC to DC converter to step down 34V to 5V and/or 12V, and to use a linear voltage regulator in order to step down from 5V to 3.3 volts.

#### 6.6.4 Backup Battery Configuration

Each control module along with the base station in WHCS will be equipped with a backup battery. In the case of power failure WHCS will still be able to carry out the basic function of checking the state of the system. Actual operation of some of the control modules will be impossible without the use of AC power. Yet checking statuses such as temperature and the position of door locks will still be fully operational. This is meant only to serve as a short term solution to power failure.

Here we will consider different designs to make a backup battery. We'll start with the circuit that was actually selected to be used in our design. The circuit below was made in Easy Circuit. The idea for the design came from the following link

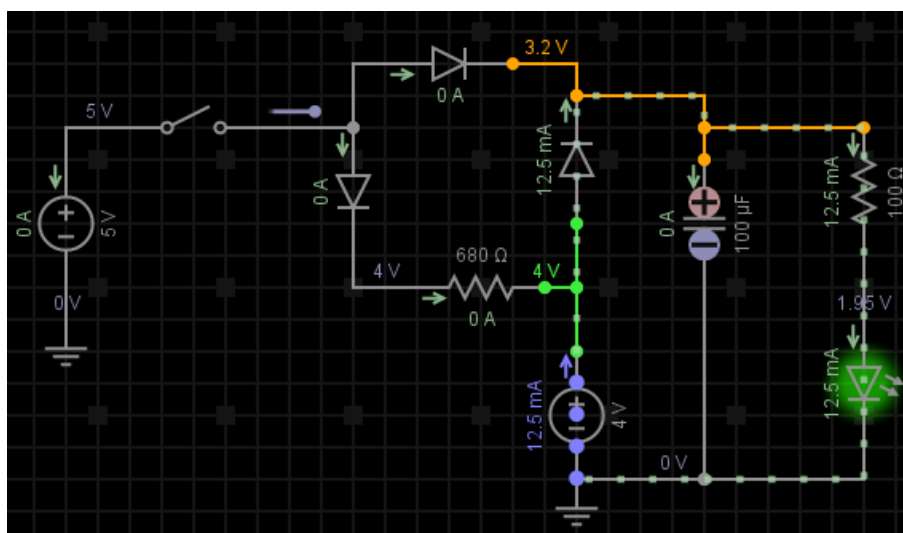


Figure 23: chosen backup-battery configuration<sup>5</sup>

<sup>5</sup>Design idea taken from <http://www.instructables.com/id/Simple-5v-battery-backup-circuit/>. Circuit remade.

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 23](#) 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 a new design would have to 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 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 range 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.

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 24](#) 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. Yet in the end we decided to go with the first design because it provides the simplest solution.

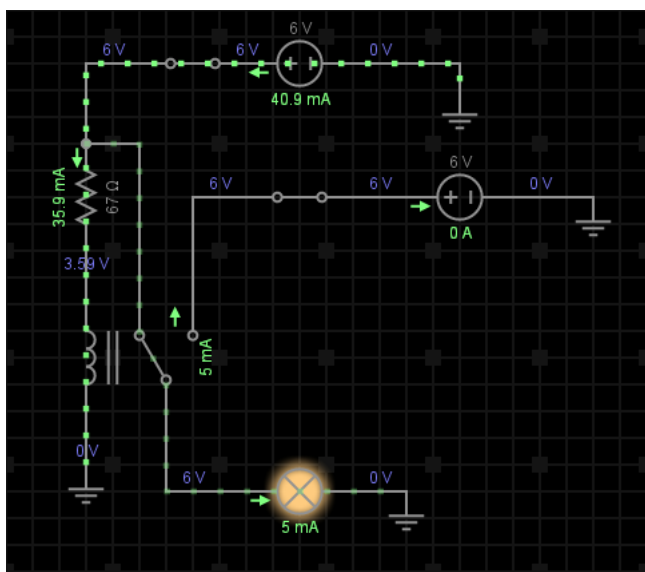


Figure 24: another backup battery configuration<sup>6</sup>

### 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 are however a lot of different transformers to choose from. With size and performance having a large impact on which transformer to choose. From previous sections we decided that our transformer will convert 120VAC to 24VAC. The first thing to consider is how much current will our load draw. We first consider the amount of current drawn so that we can determine the VA (volt amperes) rating. The VA rating is an actual specification of the transformers that will help us make our decision. Volt amperes is different from watts in that it is the “apparent power” instead of the “real power”. In order find the VA rating we will need the currents and voltages used in our system.

The devices that our system powers will all need to be taken into account in order to make sure we are making the correct approximation for how much current is to be drawn. For now we can make approximations of the amount of current drawn based on datasheets and component characteristics. When we are in the prototyping stage we can run test to see if the actual amount of current drawn matches the amount that is expected. Yet for now we’ll just draw 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

<sup>6</sup>This schematic was constructed using Easy Circuit, and the idea for the schematic came from a [Stack-Exchange post](#).

of these devices will be used at the same time. Therefore we don't really need to take into account everything at the same time. We can make 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. If we were to add the currents involved with operating the strike the current would be 665mA. The VA can be calculated by  $I \times V_{rms}$ . Where "I" pertains to the current directly after the transformer. The voltage we already know to be 24 Vrms. If we were to use the current from the device endpoints as an approximation of what the current would be directly after the transformer we would calculate the VA rating to be 16. When selecting our transformer we'll need to pick a value that is above this value, adding some tolerance is a smart idea.

After determining the VA rating there are still other factors that we need to take into account. Another thing to consider is the type of transformer. Do we want a regular transformer or a center tapped transformer. This is decided by the type of rectifier we use. We decided on a full wave bridge rectifier and will therefore be using 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 will be placed into the framing of the home. Now that means that the transformer could be 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 will keep things simpler as far as installation goes. In [Section 6.6.9](#) we discuss why our power board is going to be a through hole power board. Therefore if our board is to be a through hole board, the transformer will also be a through hole transformer. Size has some influence but is 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 will need 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 will be 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.

**6.6.6.1 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 will be using full wave rectifiers. 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.

**6.6.6.2 Capacitor** Having decided what type of rectifier we want we still need to determine the specifics on the components used for the rectifier. The two components under discussion are the diodes and the capacitors. The capacitor will determine the voltage ripple. To calculate the ripple in full wave bridge rectifiers we use the following equation  $V_r = I / (2 * f * C)$  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 can control the amount of ripple by adjusting  $C$ . The real question now becomes what is the acceptable amount of ripple in our circuit. Since the circuit will go 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 will make 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 would definitely ensure that the ripple would be less than 1V. Having a capacitor of this size is really quite unnecessary though, we’ll do fine to get away with using a smaller capacitor. Yet using two 2200 microfarads would also do the trick.

There are many types of capacitors. Yet the capacitor that we are most likely to use is a 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.<sup>7</sup> The capacitor that we will be using in our board will be a through hole capacitor. Using two 2200 microfarad capacitors we’ll have a capacitance of 4400 microfarads. The main thing that we must take into consideration is if the capacitor will be able to handle the voltage used for the power board. This stage in the circuit is right after the 24VAC has been rectified into DC. Meaning we will get a DC voltage equivalent to the  $24 / .707$  or 33.9VDC. We must make sure that the capacitor used is 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 one such part would be the Vishay 021 capacitor rated for 40V with a value of 2200 microfarads. The peak voltage will depend on the input voltage and the type of diodes used in our circuit.

**6.6.6.3 Diodes** Now let’s discuss the diodes used for the power board. There are many different types of diodes. For the rectifier there is no need for any type of special diode. The diode that we had in mind to use 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 will be using. There will also be a diode in the DC to DC converter should we choose to design this ourselves. This diode however does have a much stricter requirement for diode speeds. In

---

<sup>7</sup>[http://www.electronics-tutorials.ws/capacitor/cap\\_2.html](http://www.electronics-tutorials.ws/capacitor/cap_2.html)

this case we will be required to use a schottky diode. <sup>8</sup>

**6.6.6.4 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 is 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. <sup>9</sup>

**6.6.6.5 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 smaller a closer input and output voltage, thus making them more efficient. If however you are using a dropout voltage large enough to use a standard linear regulator, using an LDO won't have much of an impact on efficiency. We will need 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. <sup>10</sup>

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.

---

<sup>8</sup><http://www.diodes.com/datasheets/ds28002.pdf>

<sup>9</sup><http://electronicdesign.com/components/electromechanical-relays-versus-solid-state-each-has-its-place>

<sup>10</sup><http://focus.ti.com/download/trng/docs/seminar/Topic%20-%20Understanding%20LDO%20dropout.pdf>

### **6.6.7 Isolation**

### **6.6.8 Simulation Testing**

### **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 decided 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 each of our power boards are slightly different from one another so making our own boards will make it easy to customize each board for their specific needs.

After deciding that we needed 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.

In order to build our board we will be using 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.

### **6.6.10 Schematic**

## **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 will be 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 25](#), 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.

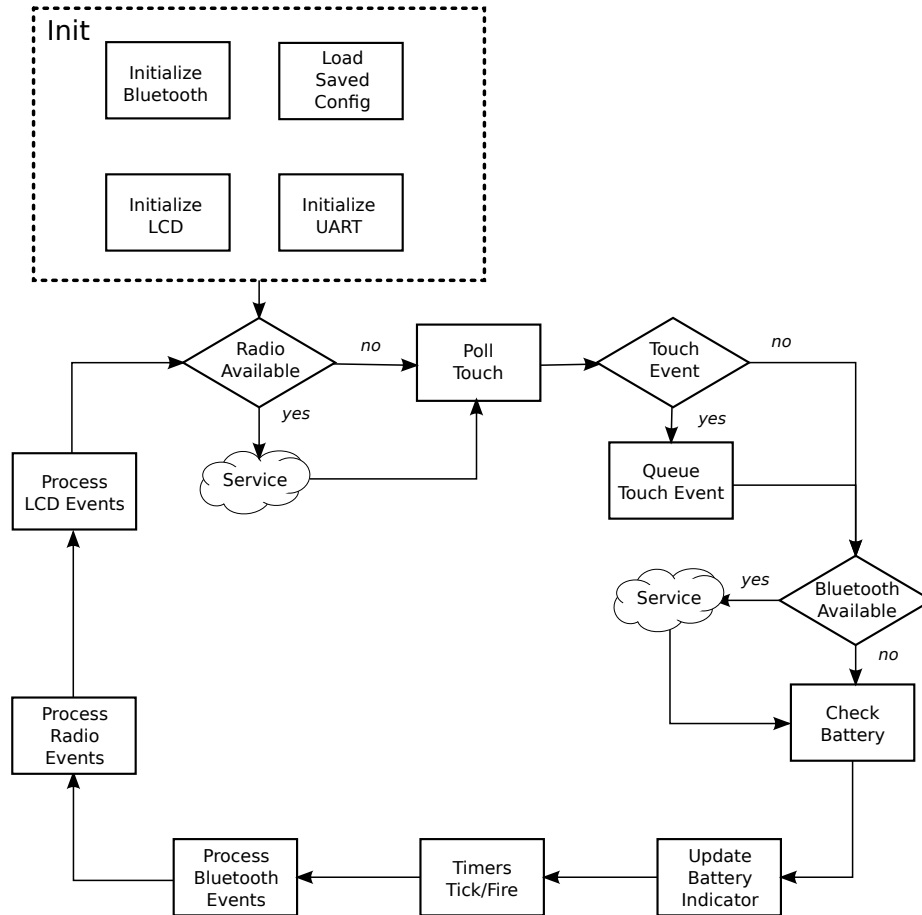


Figure 25: the high level software flow for the base station

The large amount of tasks that are required of the BS become clear when viewing [Figure 25](#).

The base station from reset would first load any saved settings from the EEPROM (saved control modules, behavior settings, LCD settings, etc.), then it would bring 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 known 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 would include 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 (parity, 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 would begin the main loop. Radio events, Bluetooth events, and LCD state would all be processed as required. For both the Bluetooth and radio, new packets would be checked for and serviced as needed. From these packets, internal state would be updated and any response packets would be generated and sent. Internally, WHCS will have an internal event queue with different event types. These events will be processed and any responses will be generated, if any. These responses could 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. These could be triggered from timers firing.

In terms of the NRF radio, the module we have chosen exports an interrupt pin which will fire 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 will have more cycles to process more important and intensive tasks. Essentially, the NRF will be kept in the listening mode at all times, waiting for a packet from a control module to be received. This is in contrast to the control modules which will try to avoid the transmitting and listening states to save power. The power down mode is great for saving watts, but has very poor performance for quickly responding to new packets.

The Bluetooth module won't require as much time to service as the other modules due to the low data rate. It will be connected directly to the hardware UART of the base station, instead of the normal serial debugger. This will have to be connected to some free GPIO ports and software serial will be used. This is unfortunate as "bit banging" serial isn't efficient and will take up many more cycles than just using the hardware UART. If needed, a packet over Bluetooth could be sent asynchronously from the main loop due to the hardware UART. If the micro loops fast enough, then it would be as if there was no delay in transmission. Due to the real time nature of the NRF radio and LCD, WHCS will take steps to avoid blocking too long on a single activity. Blocking too much will lower the overall performance and responsiveness of the system. At worst, touch events could be lost and packet buffers could overflow. Until the system's code is ready, this worry cannot be confirmed, but an

effort will be made to profile the main loop's average time using hardware timers.

One of the most important parts of WHCS is its usage of timers. A global list of timers will be constantly maintained and updated to schedule events in the future, instead of having to handle them immediately. The granularity of the timers will depend 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 will have to wait for extended periods to be serviced (starvation). Depending on the criticality of the timer and the event associated with it, there may need to be a mechanism for assigning a priority for a timer. This will have to be determined during actual system programming as it is heavily dependent on the task required.

All of the above tasks will 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 should be easily added to the system without affecting older types and there should be a set of generic data structures for managing and storing information on modules. These structures must be 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.) would be wrapped up in the generic ones (essentially a derived object from the generic control module.) This can be visualized in [Figure 26](#). The details of a network structure that would enable this clean interface is further described in [Section 6.1.4](#).

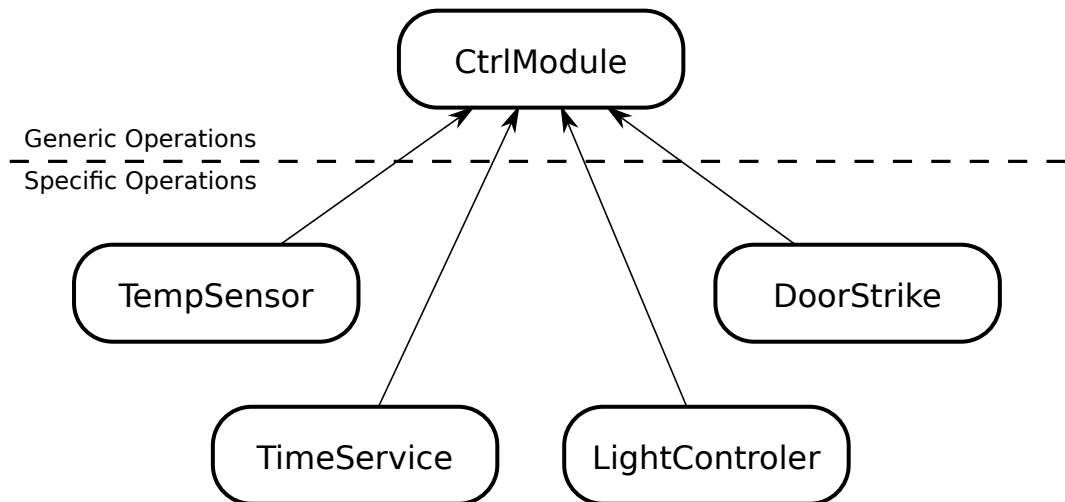


Figure 26: showing the control module hierarchy for WHCS

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 will need to be stored and functions will need to be written to query that state. Each control module would have its own state machine that would control 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 data with limited memory and processing speed. Packets need to be processed and queued to keep the pipeline flowing smoothly. The following sections will break down the individual subsystems and how they work in concert to make the base station a well oiled microcontroller.

**6.7.3.1 NRF24L01+: Interaction** The NRF radio will be directly connected to the Atmega32-A microcontroller which will control its state. This module will 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 needs to have the most attention to detail when constructing the layout and software design. Also, besides the expensive drawing operations for the LCD, this may take up the most CPU time to process packets and perform data transfers. This is partly due to the slow SPI interface that will be used to transfer the data. the NRF24L01+ breakout board does not offer any other options for transferring data to and from a microcontroller. The overall system speed will be limited by the maximum transfer speed of this radio. When the considerations for the WHCS topology are brought forward, it is obvious that the organization relies solely on the speed of the base station. We have 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 frees up the SPI bus for programming (infrequent) and the NRF. Essentially, the NRF has full control over the SPI bus and will receive 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 needs 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 will need to be checked very frequently as the radio usage will increase linearly with the amount of control modules that are apart 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. We are focusing 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 will need to be operating simultaneously in order to handle the load. Our implementation of WHCS briefly considers this fact, but due to the prototype nature of this endeavour, we have 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.

**6.7.3.2 HC-05: Interaction** 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 will be 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 will be 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 will need to be derived. This set of functions is important for more than just the BlueTooth link - if written correctly, 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 27](#). Once again this underlying protocol will be made to be abstracted away from the underlying hardware. For example, if the HC-05 were to fail to meet WHCS' strict requirements, then we would have to switch it for the next best unit - the RN-41. If we write the underlying driver to be “top level” layer that the base station will interact with, then large amount of code and possibly architecture will have to be swapped out to meet the needs of another hardware device.

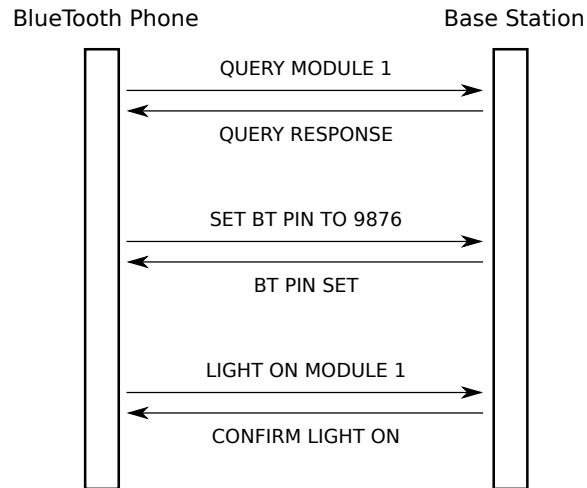


Figure 27: an example sequence diagram that could occur between a connected bluetooth phone and the base station

In regards to the application level protocol for WHCS, there will need to be 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 will decouple 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 would include controlling the function of individual modules and querying their current state. The BT connection would try to avoid generating too many packets over the NRF radio in response to user events. Instead it would merely lookup the cached state from the base station's memory. This would be faster and the round trip time would be quick. Also, if a bluetooth packet did require a packet to be generated over the NRF radio, this interaction would have to asynchronously tracked until the request was fulfilled. This would complicate the system, but would allow for more advanced queries and commands.

**6.7.3.3 LCD: Interaction** In what could be considered the “face of WHCS”, the LCD module, which will be situated directly over the Atmega32-A, will have 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 has to react 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 will only have to worry about what the end goals are for its usage. An example of this abstraction may be viewed in [Figure 28](#).

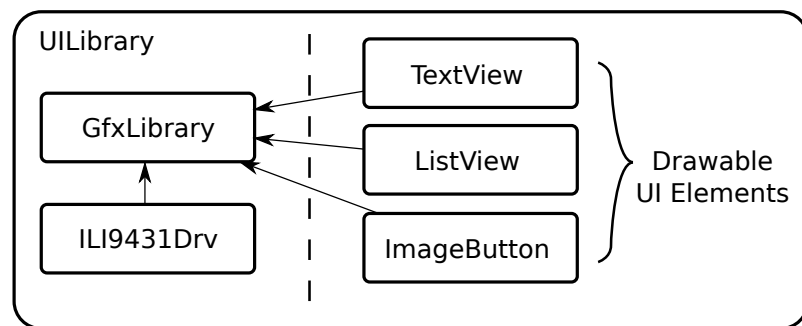


Figure 28: the level of abstractions for the LCD subsystem

As the state of the WHCS network changes, the base station will have to fire off redraw events in order to keep the LCD up-to-date. These redraws will sync the internal state of WHCS with the user viewable interface. The physical connections to the LCD will consist of data signaling and an 8-bit wide parallel data bus. There is an optional reset pin that WHCS will opt to use for emergency resets and debugging. The high level interface with the LCD will occur directly with the high-level UI library and if necessary the underlying graphics library. The base station should never use the direct driver interface as this is subject to change in an emergency (if the LCD fails to meet WHCS' requirements.) In addition, a subtle feature that WHCS may choose to implement would be dynamic power saving through screen dimming. Although we assume the base station will 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.

**6.7.3.4 Touchpanel: Interaction** 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 will be 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 could be 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 will occur from a `getTouch()` method. This method will return the latest touch event, if any. The base station will have 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.

#### **6.7.3.5 Timers: Interaction**

#### **6.7.3.6 Networking State Machine**

### **6.7.4 Schematic Breakdown**

## **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 more complex than an individual control module because it needs to be. 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.

### **6.8.1 Software Flowchart**

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 29](#). 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.

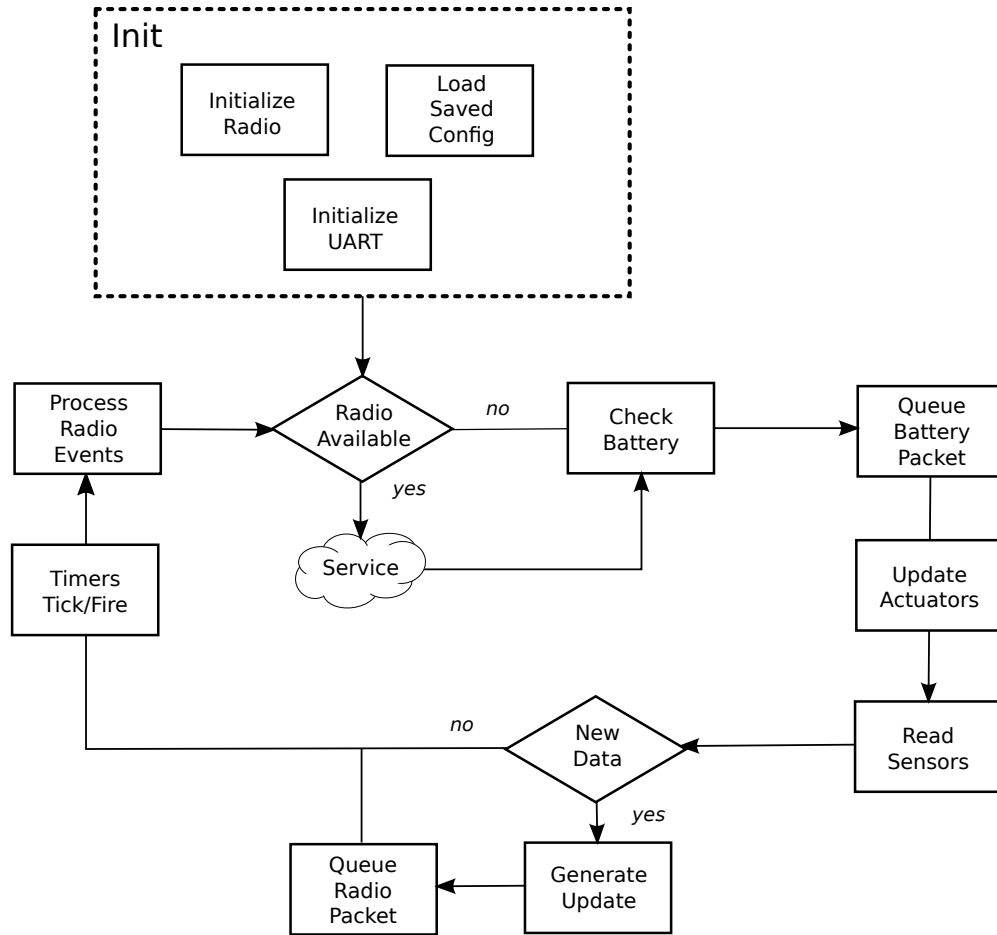


Figure 29: the high level software flow for a generic control module

Beyond scheduling packets and receiving 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. Without 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 High-Voltage Control

### 6.8.3 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 could 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 30](#) shows a simple graphic of how such a system would operate.

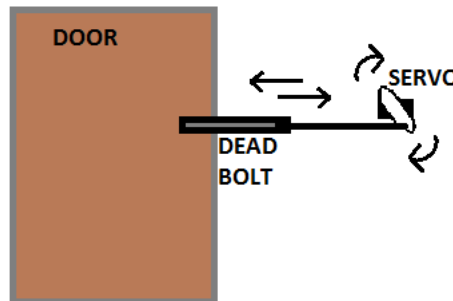


Figure 30: servo motor access control design

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 premanufacture electric strike. In the



following sections we will discuss what design considerations were taken when selecting our strike.

**6.8.3.1 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 6.8.3.2](#)) so that the safety hazard was no longer present.

**6.8.3.2 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 it 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 31](#) shows the two methods for opening the door when an electric strike and a door knob with a lock is used.

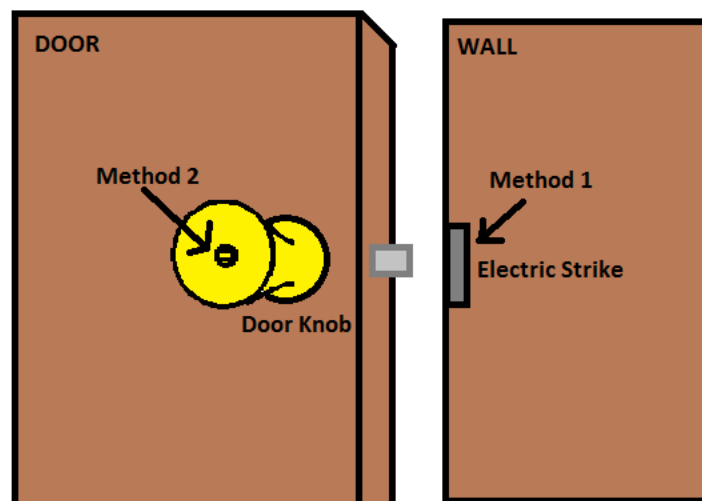


Figure 31: electric strike entry methods

#### 6.8.4 Sensor Data Collection

The control modules of WHCS will not only be used for switching things on and off. They will 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 could 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 we will specifically be using a temperature sensor to demonstrate WHCS data collection. With the way the circuit will be 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 have decided to use for WHCS is 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 32](#) shows how the temperature sensor would be 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 will be 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 will require the use of one of the ATmega328's ADC (Analog to Digital Conversion) pins for converting the analog signal to a digital signal. There are plenty of ADC pins available on the microcontroller and even when one pin on the ADC port is connected for conversion the other pins can still be used as GPIO pins. The schematic shows the output of the temperature sensor going to pin ACD0 but any ADC pin will be able to accomplish converting the signal to digital.

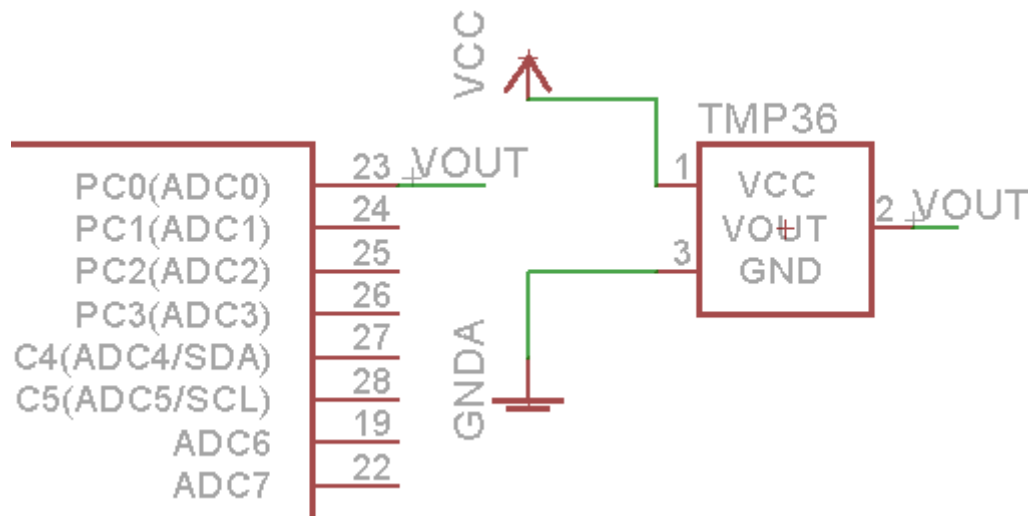


Figure 32: Temperature Sensor Connection Schematic

Utilizing the ADC pins will require 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 is important because the digital value that is obtained is a result of where

that voltage level falls in between ground and the reference voltage that is supplied. For our design we will be using the power supply of the microcontroller as the reference for the ADC which means setting bits 7 and 6 of the microcontroller to 0. The ADCSRA register is also very important because this is where we set the prescaler that will divide the clock rate of the chip into the frequency that we want to use for ADC. We are running our microcontrollers at 16 MHz in WHCS which is too fast for good accuracy in the ADC. We will set the prescaler in the ADCSRA register to 128 in order to achieve an ADC sampling rate that provides reliable accuracy.

### 6.8.5 Light and Outlet Control

The control modules of WHCS will be capable of controlling lights and outlets around the home. To be able to support this functionality the control modules need to be able to switch 120V AC circuits. A simple circuit with a transistor is not capable of switching such high voltages using the logic voltage levels of microcontrollers. In order to switch such high voltages relays can be used. 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 want 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 solve 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 is 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 get 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 S108T02F 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 33 Shows a schematic using this solid state relay. The schematic for the control module would mimic the one shown in this figure. The activation input would be 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  $R_L$  in the figure will be 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.

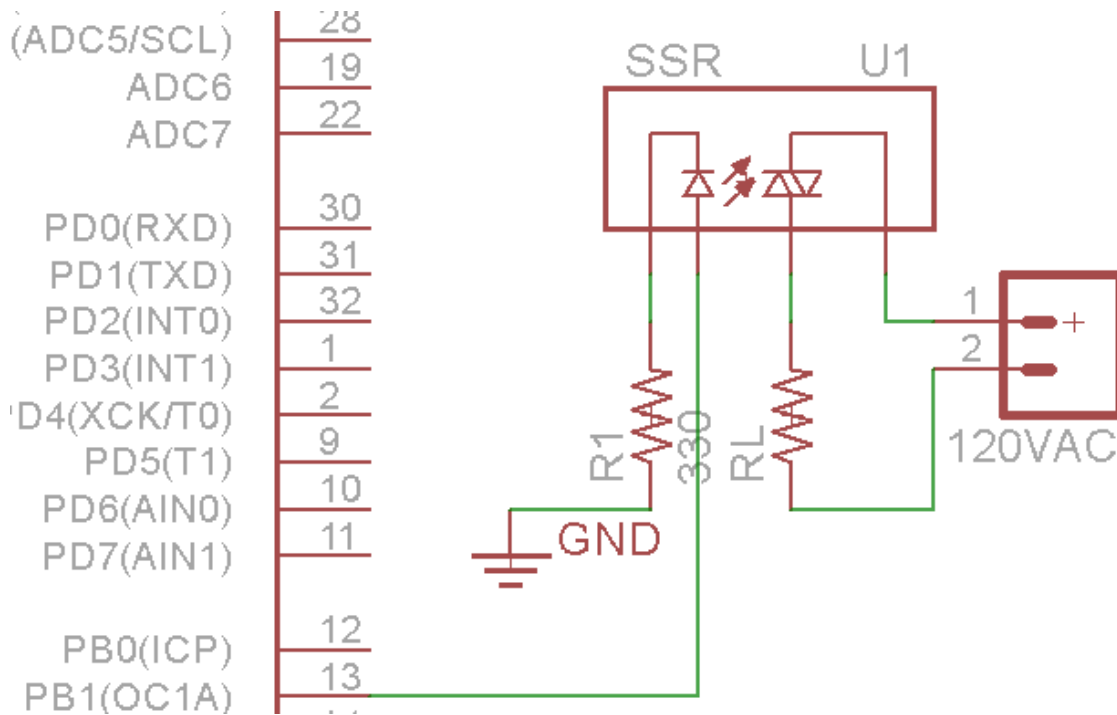


Figure 33: Wiring Schematic for Solid State Relay

### 6.8.6 Schematic Breakdown

The control modules for WHCS have to be able to support communication via a radio transceiver as well as interaction with their target endpoints. Figure 34 shows the schematic for the control modules that will be implemented in WHCS. 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 will have to access to a 3.3V line, a 5V line, and a 12V line. The power board will supply 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

Figure 34: 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 will be 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 means 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 will be 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 is no reason to make them very different and it saves 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. Now we will have an indicator LED in every PCB we create if there is a pin available. The indicator LED for the control module is shown in Figure 35 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.

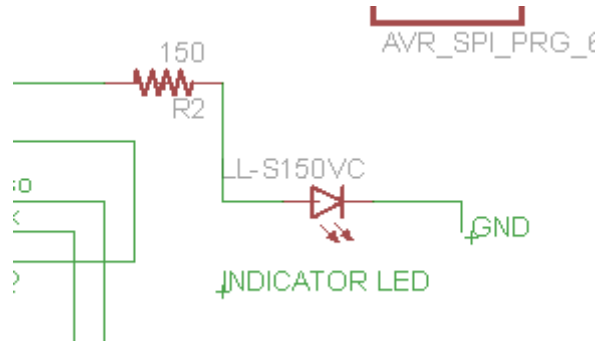


Figure 35: Control Module Indicator LED

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

## 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.

1 more page if possible

#### 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<sup>11</sup> 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<sup>12</sup>, 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 36 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.

---

<sup>11</sup><http://www.cadsoftusa.com/download-eagle/freeware/>

<sup>12</sup><http://www.freerouting.net/>



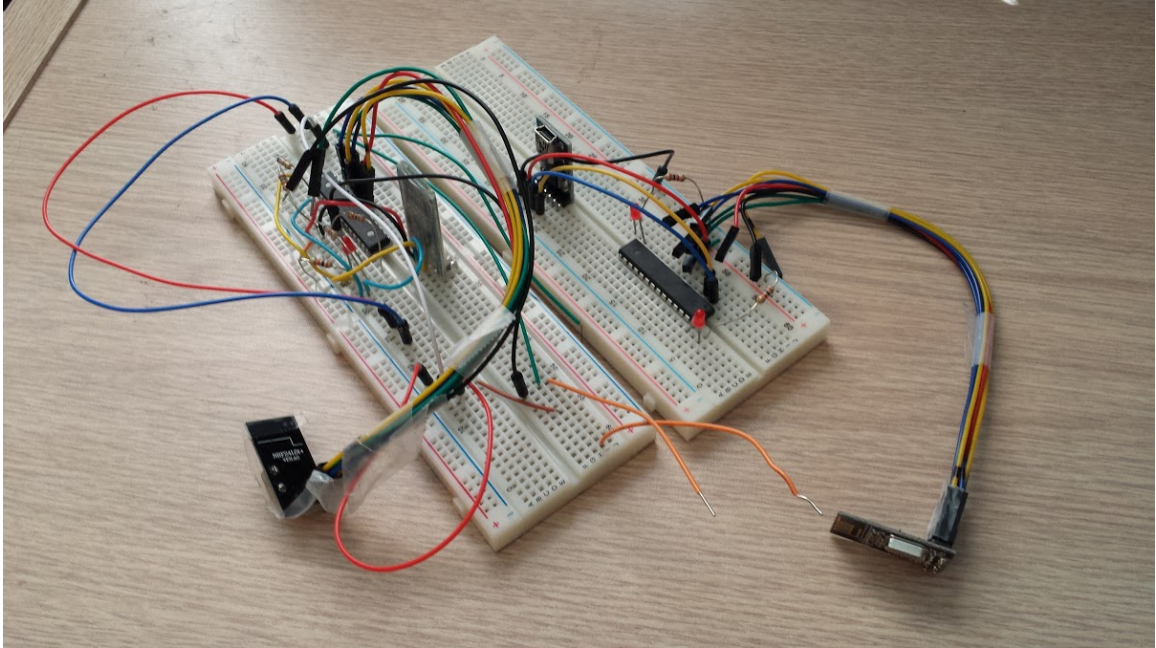


Figure 36: Point to Point Transmission Prototyping Setup

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 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

### 8.3 WHCS Proto-Panel

Presentation plays a huge part on how the public feels about a product. This is why it is so important that the display of the project is 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 into details about what plans will be made 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 we won't be able to simply plug in the system. We'll have to actually find some way to duplicate what the installation of a house would look like. In a home what we'd be 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 step we'd need to take is to convert this outlet back into wiring. We could do this by simply using a basic cord. This cord will provide us with a hot and a neutral wire and a path to ground. These wires will then be used to power the control modules and the base station. We need to somehow splice this wire into multiple wires.

We want to make the display as accurate of a representation of what would be found in a home as possible. Therefore we will try 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'll do fine to follow codes and present them in our display. In [Section 9.3.3](#) we'll go into further detail of what was done to follow these codes and standards.

Our display will consist of the frame, the wiring, and drywall. Each made to follow standards. Now our design will consist of 5 different wirings; four control modules and one base station.

#### 8.3.1 Materials

The first thing we need is a plug. It is important that we use 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'll need is enough drywall to cover the entire wall along with some drywall screws in order to attach the drywall. Insulation will be unnecessary since we aren't worried about temperature or sound insulation for our project. Also insulation in a regular home won'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'll need a few other things. First we'll need the actual control module and base station boards. these boards along with the power board will be housed in a case that will be attached to the wooden framework. For the light control module we will need a wall mounted lamp with a three wire connection. For the outlet we will obviously need to buy an outlet. For the outlet and light control modules the relays will be placed along the hot wires in order to switch them on and back off. To display the outlet control module we will need to have something plugged into our display board outlet (for example a fan or a coffee maker). The other three modules will need the LCD screen, the door knob, the strike, the temperature sensor.

### 8.3.2 Dimensions

The dimension of our project is pretty arbitrary, that is as long as it is large enough to fix each control module along with the base station. The first thing that really had to be 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 frame had to be of that size all the wood used in the frame along with the weight of an actual door would make the project very difficult to move around. Also we'd have to consider sheer bulkiness of it. 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 will be using will be a homemade door. Fortunately because we are also designing the frame that will be used for the wall, we can simply make the gap in between the studs the same size as the door we wish to make. We decided that 18 inches by 18 inches is a suitable size to make the door. After deciding this we figured that 2 feet by 2 feet would be a more than large enough area of space to display each module. The total square footage of the five control modules would be 20 square feet. We decided that a 4 by 5 display wall would showcase our design quite nicely.

### 8.3.3 Sketch

In the 4 by 5 display we have to decide where everything will go. The most important interactive parts of our design are the access control module and the base station. We must make sure that these two modules are at an acceptable height where they can be interacted with. We decided to place three of the modules on the bottom portion and two modules on the top portion. The two on top at a more accessible height therefore the door access and the base station will be placed here. The light outlet and temperature sensor will be placed on the bottom half. To make things look symmetrical the lamp will be placed in the middle while the outlet and the temperature sensor will be placed on the side. The display will look something like [Figure 37](#).

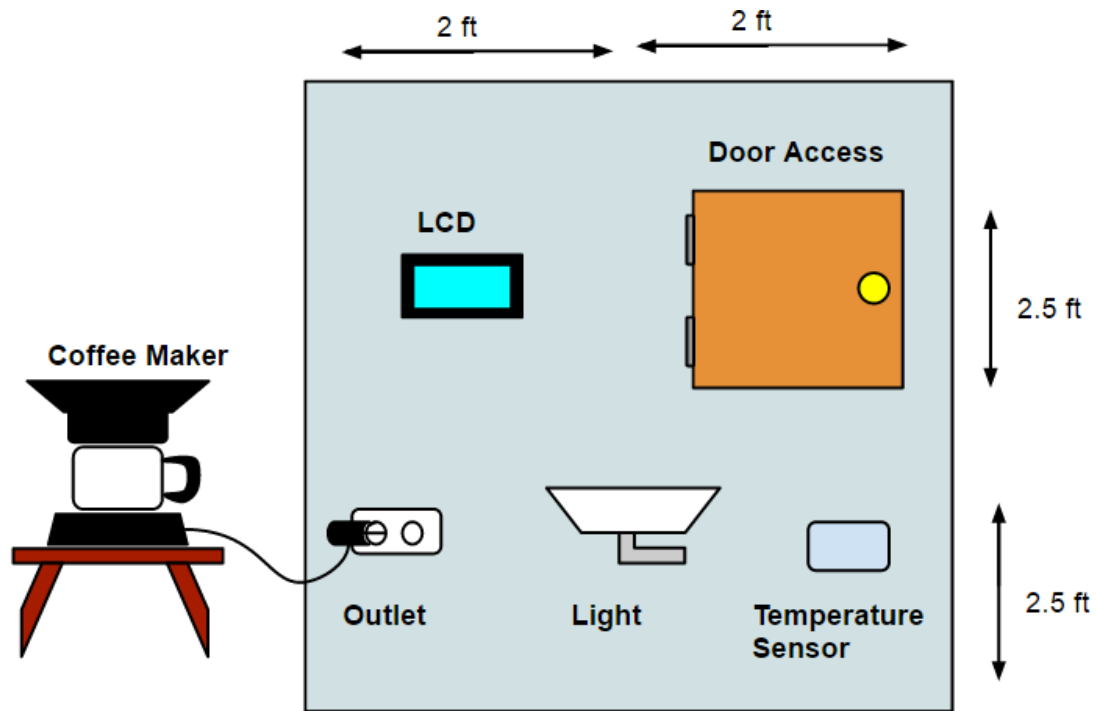


Figure 37: General design of WHCS display board

## 9 Manufacturing

*This section is in the process of being written.*

### 9.1 PCB House

1 page

#### 9.1.1 OSH Park

#### 9.1.2 Seeed Studio

#### 9.1.3 4PCB

### 9.2 Parts

#### 9.2.1 Footprint (SMD vs Through-Hole)

In WHCS we had to consider two construction methods for our base station and control module boards; 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

this isn't complete from Joseph

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.

## **9.3 Construction**

### **9.3.1 Soldering**

### **9.3.2 Reflow Oven**

### **9.3.3 Proto-Panel**

## **10 Testing**

*This section is in the process of being written.*

## **10.1 Power Supply**

### **10.1.1 5v Line Integrity**

### **10.1.2 120v Line Integrity**

### **10.1.3 3.3v Line Integrity**

### **10.1.4 Battery Backup**

## **10.2 Base Station**

### **10.2.1 LCD Control**

### **10.2.2 LCD and NRF Simultaneous**

### **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**

### **10.3.1 Voltage Level Correct**

### **10.3.2 UART Chip Testing/Debugging**

Unlike the base station the control modules will 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 will be used for debugging and testing the base station when the BlueTooth module is unavailable will be a suitable option for the control modules. The ATmega328 registers for operating the UART will be 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 will be no other need to use the Rx and Tx pins of the control module microcontrollers. Unlike the base station the control modules will suffer no limitations from the implementation of software serial routines. The control modules will be able to operate in full-duplex mode. In full-duplex mode we will be able to give commands to the control modules that would otherwise have to be received from the base station through the radio transceiver. This will allow for ease of development and testing.

### 10.3.3 Command Execution

The control modules will frequently be receiving commands from the base station. There will be 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 will 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 will 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 will be executed through the microcontroller's UART port. Commands will be 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 will be listed. From the list for each control module the commands to perform those actions will be given through the UART. The tester will document the success of each individual command given. The command execution tests will pass 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

*This section is in the process of being written.*

## 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 will be able to connect to the HC-05 on the base station and send serial data to the microcontroller. The microcontroller will be 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 will be 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 will be 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 will be using raw Bluetooth sockets for data exchange with the base station. The main class that will handle communicating to the base station is the BluetoothListener which is mentioned in the Android application section of this document. Testing this class will be essential to ensure that the communication link is working correctly for our application. BluetoothListener will have access to the socket that wraps the buffers for communicating with the base station microcontroller so writing to and reading from this socket will need 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 will allow 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 will be a standard test that we use during prototyping and testing for WHCS. For Android to base station communication testing, activating an LED ensures that the base station is 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 will model 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 are able to turn our LED on and off we know that the

Android to base station communication link is fully operational and we will be able to control the system from the mobile device. Table 8 shows the way the test will be implemented. The microcontroller will receive bytes from the mobile device. Based on the byte that we send the microcontroller should 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

## 10.6 Base Station to Control Module Communication

### 10.6.1 LED Toggle

### 10.6.2 UART to Hyperterm

## 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

1 page

## 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)

1 page

## 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 inaccessible from the application then the door knob can simply be unlocked and bypass the electric strike.

## 12 Project Management

*This section is in the process of being written.*

### 12.1 Budget

### 12.2 Parts Acquisition

### 12.3 Milestones

## 13 Appendix

*This section is in the process of being written.*

### 13.1 Appendix A - Figures

### 13.2 Appendix B - Tables

### 13.3 Appendix C - Complete Schematics

### 13.4 Appendix D - Copyright Notices

Must have an "old style" reference section