Table of Contents

1.0 Executive Summary

2.0 Project Description
   2.1 Project Motivation and Goals
   2.2 Objectives
      2.2.1 - Lighting Subsystem Objectives
      2.2.2 - HVAC Subsystem Objectives
      2.2.3 - Television Subsystem Objectives
      2.2.4 - Main Controller Subsystem Objectives
      2.2.5 - Radio Frequency Link
      2.2.6 - User Interface
      2.2.7 - Android Application
   2.3 Project Requirements and Specifications
   2.4 Division of Labor

3.0 Research related to Project Definition
   3.1 Existing Similar Projects and Products
      3.1.1 - Historical Perspective
      3.1.2 - Existing Products
      3.1.3 - Existing Projects
   3.2 Possible Architectures and Diagrams
      3.2.1 - Main Control System
      3.2.2 - Lighting Subsystem
      3.2.3 - Embedded Microcontroller
      3.2.4 - Google App Engine
      3.2.5 - Remote server and a MySQL instance
      3.2.6 - Web page with local data storage
      3.2.7 - Radio Frequency and Antenna Design

4.0 Project Hardware and Software Design Details
   4.1 Project Block Diagrams
      4.1.1 – Android platform
      4.1.2 - Google App Engine
      4.1.3 - Stellaris
      4.1.4 - MSP430
      4.1.5 - Packaging
   4.2 Lighting Subsystem
      4.2.1 - Command Set
      4.2.2 - Physical Lamps
      4.2.3 - MCU Power
      4.2.4 - Microcontroller
      4.2.5 - Digital Potentiometers
      4.2.6 - RF Transceiver
4.2.8 - LED Arrays
4.2.9 - Logic and Data Structures
4.2.10 - Schematics and PCB Layouts

4.3 HVAC Subsystem
4.3.1 - Command Set
4.3.2 - Main PCB
4.3.3 - Microcontroller Unit
4.3.4 - Temperature Sensor
4.3.5 - RF Transceiver
4.3.6 - System Status Indicator
4.3.7 - Logic and Data Structures
4.3.8 - Schematics and PCB Layouts

4.4 Television Subsystem
4.4.1 - Command Set
4.4.2 - Parts List
4.4.3 - CC1000 RF Transceiver
4.4.4 - IR Transmitter
4.4.5 - Timing and Signal
4.4.6 - Samsung Decoder
4.4.7 - Schematics and PCB Layouts

4.5 Main Controller
4.5.1 - Raspberry Pi
4.5.2 - Beagle Board
4.5.3 - Stellaris
4.5.4 - Micro Internet Protocol (uIP)
4.5.5 - Lightweight Internet Protocol (lwIP)
4.5.6 - Dynamic Domain Name Service (DDNS)
4.5.7 - API call format
4.5.8 - RF transceiver
4.5.9 - USB/Ethernet ports
4.5.9 - Schematics and PCB Layouts

4.6 Google App Engine
4.6.1 - Distributed Data Storage Scheme
4.6.2 - Database Layout
4.6.3 - Java Persistence API (JPA) Inheritance Strategy
4.6.4 - Choice of language, SDK, and IDE
4.6.5 - User Interaction
4.6.6 - Software System Architecture
4.6.7 - Login Screen Layout
4.6.8 - Menu Layout
4.6.9 - Input Appliance Information
4.6.10 - Current Settings Window
4.6.11 - Electricity Provider Information Layout
4.6.12 - Usage Calculation Interface Layout
4.6.13 - Dimmer Control Interface
4.6.14 - HVAC Control Interface
4.6.15 - TV Control Interface
4.6.16 - Credits Page
4.6.17 - Tabbing Subsystems

5.0 Design Summary of Hardware and Software
5.1 Overview
5.2 Lighting Subsystem
   5.2.1 - Microcontroller Unit
   5.2.2 - RF Transceiver
   5.2.3 - Digital Potentiometers
   5.2.4 - LED Arrays
   5.2.5 - Lighting Overview

5.3 HVAC Subsystem
   5.3.1 - Microcontroller Unit
   5.3.2 - Temperature Sensor
   5.3.3 - RF Transceiver
   5.3.4 - LED Cluster
   5.3.5 - HVAC Overview

5.4 Television Subsystem
   5.4.1 - Microcontroller Unit
   5.4.2 - RF Transceiver
   5.4.3 - IR Transmitter
   5.4.4 - Clock timing and Signal
   5.4.5 - Television Subsystem Overview

5.5 Main Controller
   5.5.1 - Stellaris
   5.5.2 - Lightweight Internet Protocol
   5.5.3 - API Call Format
   5.5.4 - RF Transceiver
   5.5.5 - USB/Ethernet Ports
   5.5.6 - Stellaris Main Base Station Overview

5.6 Google App Engine
   5.6.1 - Database Layout
   5.6.2 - User interaction
   5.6.3 - Software Architecture
   5.6.4 - Login Screen
   5.6.5 - Menu Layout
   5.6.6 - Usage Calculation
   5.6.7 - Dimmer Control
   5.6.8 - HVAC Control
   5.6.9 - TV Control

5.7 Other Possibilities
   5.7.1 - Security
   5.7.2 - Home Phone
   5.7.3 - Appliances

6.0 Project Prototype Construction and Coding

6.1 Parts Acquisition and BOM
   6.1.1 - Parts Acquisition
   6.1.2 - Bill of Materials

6.2 Proof of Concept
   6.2.1 - RF Communication link
   6.2.2 - Lightweight Internet Protocol (lwIP)
6.2.3 - Dynamic Domain Name Service (DDNS)
6.2.5 - Database Management
6.2.6 - Dynamic Android Layout Content
6.2.7 - Trigger a Command!
6.2.8 - Bag of Words

6.3 Final Coding Agreement
   6.3.1 - Jimmy’s assignments
   6.3.2 - Heston’s assignments
   6.3.3 - Kyle’s assignments
   6.3.4 - John’s assignments
   6.3.5 - Other thoughts

7.0 Project Prototype Testing
7.1 Lighting Subsystem
   7.1.1 - Description of the test Environment
   7.1.2 - Test Procedure
7.2 HVAC Subsystem
   7.2.1 - Description of the test Environment
   7.2.2 - Test Procedure
7.3 Television Subsystem
   7.3.1 - Description of the test environment
   7.3.2 - Test Procedure
7.4 Google App Engine
   7.4.1 - Description of the Test Environment
   7.4.2 - Stopping Criteria
   7.4.3 - Description of Individual Test Cases
7.5 Main Controller
   7.5.1 - Description of the Test Environment
   7.5.2 - Lighting Test Procedure
   7.5.3 - HVAC Test Procedure
   7.5.4 - Television Test Procedure

8.0 Administrative Content
8.1 Timeline and Agile Model with Sprints
   8.1.1 - Sprint 1
   8.1.2 - Sprint 2
   8.1.3 - Sprint 3
   8.1.4 - Sprint 4
   8.1.5 - Sprint 5
   8.1.6 - Sprint 6
   8.1.7 - Sprint 7
   8.1.8 - Sprint 8
   8.1.9 - Sprint 9
   8.1.10 - Sprint 10
   8.1.11 - Sprint 11
   8.1.12 - Sprint 12
   8.1.13 - Sprint 13
8.2 Budget and Finances
   8.2.1 - Budget management
1 - Executive Summary

Home automation has become more popular in the recent years. Our busy lifestyles of the 21st century have created both conveniences and problems in the household environment that dictate the necessity of these systems. A major
drawback of home automation systems has been the cost. Our senior design project consists of building a home automation system that will control various aspects of the modern house. The objectives of the project are to design a home automation system that is functional and easy to use from the user's perspective, make the system more accessible in terms of price in the consumer market, and have the consumer be able to track the energy consumption of the devices used in their home. One of today's biggest concerns for all aspects of society is energy consumption. However, the traditional methods to cut back on energy consumption have also created headaches and major inconveniences for the end user. One of our main objectives is to change this traditional approach and create a home automation system that saves energy, but does not inconvenience the user. In fact, another one of our objectives is to design our home automation system in such a way that it saves the user time.

Technical objectives of the project as a whole include integrating software and hardware knowledge gained from previous coursework into a practical and working system and to have the system to consume a low amount of power. In order to meet the objective of having the system run on a low amount of power consumption, parts known for lower power consumption that are relevant to our design, will be used. An example of a part for our design is Texas Instruments MSP430 microcontroller. These microcontroller units are incredibly energy efficient and can retain memory with a very small amount of power in the microvolt range. In order to cover the tracking of the energy consumption, the amount of time that the devices are used for will be recorded and stored in a database. The software will allow the consumer to calculate the electric usage, at any time via a web page. In addition, the user will be able to schedule automatic controls for the home automation system. One of these automatic events shall be turning lights on and off at certain times of the day. This feature could be used both as a convenience and a crime deterrent. Another scheduled event that shall be implemented is the automatic control of a home's heating, ventilation, and air conditioning system (HVAC). The user will be able to set different temperatures for different times of the day, as well as set a limit on how much electricity shall be used and adjust the system accordingly.

2 - Project Description

2.1 - Project Motivation and Goals
Senior design is a course created to put engineering students' skills to the test. Our plan to create an automated home system that can be controlled remotely
from inside and outside the home is a challenging and exciting project to undertake. Home automation has been something of a dream for the past few decades but is slowly becoming realizable. The problem today is cost. Most home automation systems can burn quite the hole in your pocket. Similar products such as Homeseer and HAI Omnitouch range from the price of $500.00 to $700.00. We seek to use cheaper and more efficient means to implement the functions of these products.

Another product the group has looked at is the Control4 products. They actually have every module we plan to build and more. Their products have motivated us to want to create something a little cheaper than this product. One of their lighting modules is $80 alone and that does not even include the base station. They charge around $300 for the main controller, which we plan to spend less than $50 on. Our goal is to make a more affordable version of their produce for ourselves.

Another goal the group wants to accomplish is voice control. The group has done the research about android.speech and we believe it can be implemented quite easily thanks to Google's application programming interface. We are really taking advantage of the high powered processor in the phone for our project since everything else is a low powered microcontroller. The group will use a bag of words style decoder because we believe that codewords will be easier to match than entire sentences. This is the main reason we have decided not to make this application cross platform yet.

This project will require the knowledge of many, if not all, of the aspects of our engineering education. For example, the television communication module will use a Texas Instruments MSP430 ultra-low power infrared remote control transmitter which requires knowledge of embedded systems, circuitry and programming as well as a few other skills gained throughout our college careers. Testing our abilities is part of our motivation to choose a project such as home automation.

It is our belief that a home automation project will challenge us in many aspects of the engineering realm. Being an engineer requires vast knowledge of science and mathematics. We are motivated to do a home automation system because it has the potential to allow us to experiment with many parts of the engineering discipline such as electronics, digital systems, circuit building, simulation, wireless communication, and so on and so forth. Cost effectiveness and efficiency have also been ingrained in the minds of engineers since the beginning of time as well as the ideas of minima and maxima.

Realistic goals and achievements must be set for all projects of this sort such as attempting to attain maximum and minimums. For example we want to achieve minimum power usage while still maintaining maximum efficiency of system operation. Low cost is also a typical goal for senior design projects. Parts and
implementation must have a low price tag for the possibility of widespread usage of a product such as home automation.

2.2 - Objectives
The overall objective of the 2012: A Home Odyssey home automation system is to implement a low cost, compact and energy efficient home solution for potential users households. Being engineers encourages us to make it better, smaller, cheaper and faster to overcome past technologies and to improve the future. By the end of our college careers we plan to have a working home automation system. This system shall consist of four engineered printed circuit boards designed using our knowledge gained throughout our engineering college careers. The printed circuit boards shall consist of three different subsystems and a main base station for relay communication between said subsystems and a user. The home automation system will allow a user to control certain parts and rooms of his or her home from a remote location using a smartphone application or a computer. The user can control home lighting, a television and a heating, ventilation and air conditioning control system. These systems are fed information from a main central base station which is in turn controlled by a user through the Google App Engine. This system shall also read the energy consumed by each subsystem through the Android application.

2.2.1 - Lighting Subsystem Objectives
To ensure efficiency of the lighting subsystem we chose a Texas Instruments MSP430G2553 Value Line microcontroller for its cheap price and ultra low energy consumption. Another factor that allows the low use of power is the decision to use a light emitting diode array instead of incandescent light bulbs. Incandescent lights require high wattage and energy where certain problems may arise such as danger from high voltage and high energy consumption. The choice of light emitting diodes uses low power and is an efficient way to convey the functionality of the lighting subsystem. These light emitting diode arrays shall represent different sections of a house and separate appliances in these sections. One of the objectives of each array shall be to be able to dim the lights between 0% and 100%. We decided that the lighting subsystem shall be completed first because it is contained in itself. We can test this system alone and make sure each light emitting diode is working properly.

2.2.2 - HVAC Subsystem Objectives
To ensure efficiency of the lighting subsystem we chose a Texas Instruments MSP430G2553 Value Line microcontroller for its cheap price and ultra low energy consumption. In addition, the data structures necessary to operate this system shall only require 2 kilobytes of data which is only one half the total memory of the microcontroller. To simulate the heating, ventilation and air conditioning subsystem we shall use an array of multicolored light emitting diodes to indicate to the user the status of the system. The correct combinations of particular light emitting diodes for the system shall indicate which status is being portrayed such as cooling, heating, emergency heat and fan only. This
system will represent a real time home air conditioning system. This shall be the second subsystem completed and tested after the lighting subsystem is in place and functional.

2.2.3 - Television Subsystem Objectives
Using an MSP430F2131 Texas Instruments microcontroller ensures energy efficiency for the infrared remote. To conserve energy we shall use the ultra low power microcontroller unit which in idle mode only consumes 0.1 μA. This unit shall also use minimal amounts of energy when in full use. Since infrared television remote controls are wireless we also decided to go wireless by using a 3.3V lithium coin cell battery. While energy is an important factor, so is speed. We want our channels changed fast so the RC5 packet transmission protocol shall only take 0.89 ms to transmit a request from the infrared remote to the television. The goal for capabilities shall be to turn the volume up and down, change the channel up and down, turn the power on and off, and finally to select a specific channel by entering a channel number. The television subsystem shall be the third task we choose to tackle due to the interfacing with a television and extensive coding.

2.2.4 - Main Controller Subsystem Objectives
The chip we chose to act as a base station for the entire system is the Stellaris LM3S8962 microcontroller. This microcontroller unit is a bit more powerful than the aforementioned MSP430 microcontrollers but still falls within our desired range of low energy consumption. The Stellaris unit is also a cheap alternative to some other possible base stations. The main base station shall communicate to the other subsystems via radio frequency links. This station shall be the hub between the subsystems and the user input. This controller shall be hardwired to the internet for non-wireless communication while a universal serial bus based interface shall be used to flash the chip for configuration.

2.2.5 - Radio Frequency Link
The final decision to use the CC1000 radio frequency transceiver above the XBee and the Zigbee was quite thought out. The challenge of implementing our own wireless communication is one we welcome. Although this task has been daunting for others in the past we believe we have the skills and motivation to take it on. The plan is to use a low level sub 1 gigahertz protocol that shall transmit data between the subsystems and the Stellaris main controller. The radio frequency transceiver, when sending data, shall send a broadcast to all subsystems with specific information which shall then be decoded by the correct subsystem and implemented by the MSP430 units. Another alternative wireless communication system considered was bluetooth technology.

2.2.6 - User Interface
To tie everything together, a user interface must be implemented. The choice of using the Google App Engine allows our group to use a web based application and online data storage. Through this application we plan to allow the user to
control each subsystem and calculate energy costs via an Android smartphone. The application shall hold the names of the subsystems, the area of the house which is being accessed, the specific appliance and the electric company information. The user interface will be a sleek, aesthetic design to attract users but still be a simple system to navigate on the fly.

2.2.7 - Android Application
Our Android application will be built dynamically from what is stored in the Google App Engine. We will need to send a request to see what modules the user has and the group has decided to use a centralized database for this. We believe that it will be a better idea to do this over statically programming the device because it would be a pain to program a new device. We have also decided to implement voice control which the Android application will need to process. Since the phone actually has the most powerful processor in our project, it was the best place for this to be done. The group has looked into android.speech and we feel confident in its ability to give us the required data. Another decision the group made was to implement a bag of words keyword driven command system. The user would only need to say certain keywords to perform the desired commands rather than a complete sentence. We are not going to focus too much on design and graphics because we are engineers and not graphic designers.

2.3 - Project Requirements and Specifications
The central controller in the system will be a Texas Instruments Stellaris ARM Cortex M3 microcontroller running a lightweight web server that will be sent commands over the Google App Engine interface running on any android device. The user’s commands will be sent in real-time to the subsystem controllers with a delay of not more than 10ms. The controller will be powered via a 3.3V rectifier drawing power from a common household 110V 60 hertz alternating current power. The central controller will communicate with the subsystems via an radio frequency link.

Each subsystem will be controlled by a Texas Instruments MSP430 microcontroller. Each of these subsystems will be powered via individual 3.3V direct current adapters that will be connected to a 110V 60 hertz alternating current source.

The lighting subsystem will use a MSP430 microcontroller that will serve two basic functions: one to transmit commands from the central controller to a dimmer relay and the other to transmit back to the controller the total power consumption of the lighting system. The subsystem will use TI CC1000 wireless transceivers operating at a sub 1 gigahertz frequency to communicate with the central controller. The user will be able to select any channel (room) via the web interface on the dimmer board and set lighting levels via a virtual slider. The data will be collected by the MSP430 to measure the power output and sent to the main controller where it will be stored and manipulated.
The heating, ventilation and air conditioning subsystem will use another MSP430 microcontroller connected to a TMP100 temperature sensor via the I2C interface. In addition, the MSP430 will also be connected to a series of 24V light emitting diodes to simulate a typical heating, ventilation and air conditioning thermostat output. To simulate the heating, ventilation and air conditioning subsystem we shall use an array of multicolored light emitting diodes to indicate to the user the status of the system. The correct combinations of particular light emitting diodes for the system shall indicate which status is being portrayed such as cooling, heating, emergency heat and fan only. The user will be able to set their desired temperature and the software will decide whether to turn on the heating or cooling light emitting diodes.

The television subsystem will use a third MSP430 microcontroller connect to a Texas Instruments TIR1000 Infrared Controller to send infrared commands to the television set. These commands will be channel up/down, volume up/down, sleep timer settings, power on/off and channel number selection. The power input on the television set that will send power consumption data to the subsystem controller and then to the central controller.

A local database will be built in conjunction with the web page taking in user commands. The database will store daily calculated energy usage for each individual controlled device around the house. Time, in which the appliances will run for, will be stored into the database along with the wattage of common appliances. For every minute that an appliance will run, the MSP430 controlling said appliance will send out a signal to the Stellaris. Whenever the Stellaris receives a signal, the variable keeping track of the time that the appliance is running will increment by 1. At the end of each month, a calculation will be implemented to calculate estimated energy costs according to local rates. The database management system that will be used for this system is SQLite.

Another proposed web server solution we're debating is the google app engine. The google app engine would provide a simple and sleek platform to build a java based web app on. It would have a web based user interface where we would use their big table database to store and display statistics plus show and control the current home environment. We could also tie everything to their gmail account for security reasons by using the app engine’s oAuth. The smart phone could also access this either through the web or on a custom app. The Stellaris will need lightweight internal protocol to properly host a web server, and will also have some security of its own. The Stellaris itself will not host the website (except for configuration), but will only host pages where you can control the subsystems. Another solution we’re looking at using with the Stellaris is Dynamic domain name service in order to directly connect with it.

API calls will have a similar format:

Stellaris_URL+”/subsystem?userid&password&unit&command”
ie: “/lighting?jedison89@gmail.com&**********&bedroom1&lights_on”

Above is a sample URL we plan to use, it will first tell the stellaris which subsystem to use, then it will check to see if this user has permission to access the unit, and finally it will command a MSP430 if allowed.

2.4 - Division of Labor

Each member in our senior design group has their own individual background in electrical engineering or computer engineering. Our group consists of two computer engineers and three electrical engineers. In addition, each member has their own specific interests and expertise in their major that needs to be effectively utilized for the project. The project will be divided into multiple parts, each effectively utilizing the strengths and expertise of each individual member as shown in figure 2-1.

Kyle is one of the two electrical engineers in the group. His expertise lies within interfacing analog to digital and digital to analog circuits, analog and digital filter design, semiconductor circuit analysis, and radio frequency communication. His major responsibility will be interfacing the main control unit with the subsystems over an RF link as well as design the pcb layouts for the main control unit and the individual subsystems. The individual subsystems he will be responsible for are the HVAC and lighting subsystems.

Kyle D’Arcangelis
- Microcontroller programming
- HVAC subsystem temperature sensing and control
- Lighting subsystem design
- RF Links

John is one of the two computer engineers. His expertise lies within high level software backend programming and RF communications. His major responsibilities will be interfacing the google app engine with the main control unit and also design the software portions of the RF links.

John Edison:
- Software portion of RF Link
- Internet Connectivity
- Backend software interface
- Main Controller mcu programming

Heston is the other electrical engineer in our group. His expertise lies within semiconductor circuit analysis, analog filter design, and operational amplifier design and implementation. His major responsibilities will be designing the lighting subsystem led clusters and designing the television subsystem with the infrared command set. He will also be responsible for laying out printed circuit boards.
Heston Posner
- Television Subsystem
- Light emitting diode clusters
- PCB layouts

Jimmy is also another computer engineer in our group. His background lies in frontend user interface design and high level api programming. His responsibilities will be designing the frontend users interface and data structures. He will also be responsible for programming the high level data structures for the microcontroller units in the individual subsystems.

Jimmy Wo Wong
- High level programming
- Frontend user interface
- Data structures and logic analysis

Figure 2-1: Division of Labor Flowchart
3 - Research Related to Project Definition

3.1 - Existing Similar Projects and Products

3.1.1 - Historical Perspective
Home automation systems are nothing new in 2012. The basic technology behind these systems have existed since the early 80’s, but have been only available to the super wealthy until the past decade. One reason that this technology has been beyond the reach of the average consumer is the expense of the microcontroller units which are the heart of these systems. The major driving force behind this increased availability of microcontroller units is the product lifecycle. In reality, a microprocessor is never really sent to the graveyard in one step, but rather transferred into the embedded world where state-of-the art performance is not the topmost priority, but price is. Even today if you browse through the intel website, microprocessors such as the original Pentium and even earlier models are available in physical form or in a hardware description language to purchase for embedded applications.

3.1.2 - Existing Products
If one does an internet search of “home automation system” a plethora of results will populate the screen. Browsing through product catalogs from different automation companies gives the potential customer a whole host of automated lighting systems, audio/video controllers, and HVAC control systems. Even though the selection of products and services seem very overwhelming at first, if one looks at the basic architecture of these systems they are all very similar in form and functionality. Each system has, in some form or another, a central
control unit that communicates with various end devices and subsystems as shown in figure 3-1. These devices can range from a simple light dimmer, to a fully fledged home entertainment system.

![Figure 3-1: Typical Commercially Available Home Automation System](image)

The major item that differentiates these systems is the underlying protocol used to intercommunicate with all of the connected devices. The major protocols in use today are INSTEON, X10, and ZigBee. Each of these protocols have their own set of advantages and disadvantages depending on the desired application. X10 primarily runs over power lines and is the most basic of the three, thus is usually the least expensive. However, it is limited to simple dimmer controls with a maximum of only 8 devices. INSTEON takes X10 even further and allows up to a maximum of 16,777,216 devices and can be used over radio frequency. The latest protocol, Zigbee, takes home automation to an entirely different level. Operating in the lower part of the 2.4 gigahertz band, ZigBee, as shown in figure 3-2, is a general purpose wireless protocol that allows up to 250 kilobits per second of bandwidth to be available at any given time. While by today's standards 250 kilobits per second may seem like a small data pipe in our world of on-demand high definition video, for a home automation system with its relatively small data structures, this is plenty of bandwidth.
3.1.3 - Existing Projects

Home automation systems have been very popular electrical/computer engineering senior design projects due to their relative simplicity and integration of hardware and software. One project in particular from the Stevens Institute of Technology titled “SHARP”, as shown in figure 3-3, has very similar objectives to ours. The main objectives, to remotely control the basic systems of a house and monitor these system, and the basic layout, a central controller sending commands to subsystems, is almost exactly the same layout our system will use. The differences, however, are in the hardware implementation. This particular project uses the Zigbee protocol for sending commands to the various subsystems and relays to control the power sources. Our implementation does not need the 250 kilobits per second maximum theoretical throughput of Zigbee due to the simple command set. In addition, all of our devices will be “soft-on”, so the need for relays is eliminated.
3.2 - Possible Architectures and Diagrams

3.2.1 - Main Control System
There exists many different architectures for the main control system for a home automation system. Two possible architectures we discussed are using a typical desktop computer as the main controller or an embedded system connected to the Google App Engine cloud. Each of these architectures have their own unique host of advantages and disadvantages.

Using an operating system on a desktop computer as the main controller allows for greater future expandability as well as ease of use. An operating system is an event driven program or set of programs that manages system resources efficiently, provides a friendly user interface, and creates an environment for other programs to run in, as shown in figure 3-4. To add a new component to the system would not require an extensive modification of the entire main controller. Any modifications, whether hardware or software, simply require editing lines of
code instead of redesigning a printed circuit board layout. In addition, using an operating system allows for greater standardization of the software-front end and back-end, as well as system resource protection. In an embedded situation if a segment of code overruns into another, there is no memory protection to keep the entire system from crashing. Operating systems also provide for better multitasking, such as in the case of receiving commands from and radio frequency transceiver and sending data to a serial bus simultaneously.

The downsides to using a desktop computer are greater power requirements and the reliability problems associated with using an operating system. Operating systems work well for general purpose computing needs, such as word processing and internet browsing, but can use too many unnecessary hardware and software resources. Operating systems are generally laden with bugs and resource conflicts which is not ideal for creating a reliable home automation system. In addition, interfacing hardware at a low level requires the use of specific device drivers, which use even more system resources than an embedded application.

![Operating System Resource Layers](image)

To the contrary, using an embedded system with a microcontroller unit as the main controller offers some unique advantages over a desktop system running an operating system. The largest advantage over using a desktop system are
the smaller power requirements. Without the need for disk drives, graphics cards, network interface cards, and other external hardware, the power consumption significantly decreases when using an embedded system. Another advantage is the lack of overhead and reliability issues associated with using an operating system. For the simple, small data structures used in a home automation system, an embedded system is all that is really necessary for a functioning system. In addition, embedded solutions are generally much cheaper than desktop solutions simply because you are only using the hardware that is absolutely necessary for the system to function.

While embedded systems offer some advantages over a desktop system, it does not come without its own set of disadvantages. A major disadvantage of using an embedded system is that if any modifications need to be performed on the system, a partial or entire revamp of the system may be necessary. Entire integrated circuits may have to be either modified or replaced and programming must be redone at a very low level. Embedded systems also lack the rigorous structure of an operating system environment, allowing for a greater chance of scheduling conflicts.

3.2.2 - Lighting Subsystem
Two possible designs for the lighting subsystem is to use either low voltage LED bulbs or high voltage incandescent bulbs. Each bulb type has its own host of advantages and disadvantages. Incandescent bulbs offer greater brightness per unit bulb, but draw more power and are more difficult to dim via the digital I/O from a microcontroller unit. In order to translate the low direct current voltage signal from a microcontroller unit to the high alternating current line level voltage, a series of triacs and transistors are needed to convert the pulses to voltages. A entire rectification system is needed to compare the line level ac voltage to the regulated dc voltage in order to generate the correct pulses to drive the triacs, as shown in figure 3-5. While this offers greater flexibility in terms of lamp availability, it is not ideal with regards to safety and reliability. Incandescent bulbs also produce much greater heat when compared to LEDs which produce almost negligible heat.
As shown in figure 3-6, Using LED bulbs greatly simplifies the lighting subsystem. There is no need for a complicated triac system to separate the high and low voltage. Instead, a simple digital to analog converter is used to translate the digital signal from the microcontroller into corresponding voltages. All that is needed to ensure the light emitting diodes do not draw too much current from the microcontroller is a simple buffer using an operational amplifier. In addition, light emitting diodes do not generate anywhere near the amount of heat of conventional incandescent lamps and last much longer. This saves a tremendous amount in HVAC and replacement bulbs costs.
3.2.3 - Embedded Microcontroller
While there exists many different microcontrollers on the market today, only a few suite the needs of a home automation system. We considered three different microcontrollers to control each of our three subsystems: the Motorola 68HC11, Peripheral Interface Controller (PIC), and the Texas Instruments MSP430 line of microcontroller units.

The Motorola 68HC11, as shown in figure 3-7, was a popular choice in embedded system up until the late 1990’s. It is a CISC based microcontroller than is programmed via the Motorola 6800 assembly instruction set. This particular microcontroller can be found almost any older embedded devices such a key card locks and digital thermostats. They are very inexpensive to purchase even today and the development tools are very basic, but straightforward.
The downside of using the Motorola 68HC11 line of microcontroller units is the 8-bit general purpose registers and the lack of a C-based programming interface. Any programming must be done in the 6800 assembly language no matter how large or small of a task. This can create some very unique challenges when trying to interface radio frequency transceivers and serial buses. In addition, due to its 8-bit general purpose registers, data structures must either be kept very small or read and writing from memory to registers become necessary, hogging the already severely limited resources of the Motorola 68HC11.

The Peripheral Interface Controller (PIC) mcu has been another popular choice in the world of embedded computing. They are very low cost, consume very little power, and are widely available. The user knowledge base is expansive and the development tools are either low cost or free. In addition, the user can program in either assembly or C, giving much greater flexibility when performance or ease of use are top priorities. Even if the user chooses to program in assembly, the PIC processor is a RISC based architecture making the instruction set very small and easy to learn. The PIC processor also natively supports a very wide range of serial buses such as I2C and SPI.
Unfortunately, the PIC processor does not come without some limitations. The first major limitation is that it only has one accumulator register, making multitasking next to impossible. In addition while the instruction set is small, it requires many instructions to perform the same task as in a CISC based architecture like the Motorola 68HC11. Expanded memory access can also be confusing to the end user due to its bank-switching memory architecture.

The last microcontroller we discussed is the Texas Instruments MSP430 Microcontroller Unit. This architecture, as shown in figure 3-8, also has a very wide knowledge base and the development tools are low cost and easy to use. The MSP430 architecture is RISC based and can be programmed in assembly or C. In addition, the Value Line series is capable of running on extremely small amounts of power, as low as 0.1 microamperes for memory retention and 200 microamperes for MIPS. The TI MSP430 also supports a wide range of serial interfaces making it easier to connect to rf transceivers and digital-to-analog converters.

![Figure 3-8: TI MSP430 Architecture](image)

A few drawbacks of using the MSP430 architecture is the limited system resources. Multitasking on the MSP430 can become quite a challenge when data structures and programs start to increase in size and complexity. The interrupt based nature of the MSP430 is the reason behind this. In addition, memory size is also severely limited which creates problems when data becomes very large.
3.2.4 - Google App Engine
An option to consider for the front-end software user interface is to implement a web-based application via the Google App Engine. An advantage of using the Google App Engine is the availability of a database, Google’s BigTable database, which is applied to the server space allocated to the application. A remote server and a relational database instance are no longer needed because of this. Another advantage is the fact that 10 free applications can be made by a single Google account. This is advantageous due to the need of prototyping the design. Google App Engine also supports applications for the Android platform. This would be a necessity in this design due to the need of portability.

A drawback of the Google App Engine is the memory limit. For any free application on the Google App Engine, 1 GB of space, for the Google Datastore, and 200 indices are allocated to the application. This, of course, can be remedied by charging the consumer for memory expansion requests.

When it comes to data storage with the Google App Engine, there are two data distribution options: High Replication Datastore and Master/Slave Datastore. The main differences between the two data distribution options, in terms of performance, are write latency and average read/write error rates.

According to Alfred Fuller of the Google App Engine team, the Master/Slave Datastore scheme consists of one data center acting as the sole destination for communication with the application. When the Master data center receives data, it asynchronously replicates data to subsequent “Slave” data centers. An advantage of the Master/Slave Datastore scheme is a low read and write latency. The major disadvantage of the Master/Slave Datastore scheme is the handling of critical errors and a noticeable downtime for planned maintenance. When the Master/Slave Datastore scheme has a global failure, the Datastore and the entire application will be unusable. This results in partial data loss.

The High Replication Datastore scheme consists of data being synchronously replicated across multiple data centers. This results in a higher availability of data reading, but at the cost of a higher write latency. The main advantages of the High Replication Datastore scheme are no partial data loss when it comes to global failures and a shorter downtime for planned maintenance. When a global failure occurs, since data has been repeatedly replicated synchronously throughout multiple data centers, no partial data loss occurs.

The following represents the basic functionality differences between the Master/Slave Datastore scheme and the High Replication Datastore scheme:
The following table represents a detailed summary of the differences in performance between the High Replication Datastore scheme and the Master/Slave Datastore scheme:

<table>
<thead>
<tr>
<th></th>
<th>Master / Slave</th>
<th>High Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Released</td>
<td>April 2008</td>
<td>January 2011</td>
</tr>
<tr>
<td>Replication</td>
<td>Asynchronous</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Replicas</td>
<td>2</td>
<td>&gt;2</td>
</tr>
<tr>
<td>Master</td>
<td>Single</td>
<td>None</td>
</tr>
</tbody>
</table>

(Courtesy of Alfred Fuller and Matt Wilder of the Google App Engine Team)

Figure 3-9

<table>
<thead>
<tr>
<th>Average Latency</th>
<th>Master / Slave</th>
<th>High Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>15 ms</td>
<td>15 ms</td>
</tr>
<tr>
<td>Write</td>
<td>20 ms</td>
<td>45 ms</td>
</tr>
<tr>
<td>Average Error Rate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>.1%+</td>
<td>.001%</td>
</tr>
<tr>
<td>Write</td>
<td>.1%+</td>
<td>.001%</td>
</tr>
</tbody>
</table>

99.9% = Three 9’s
= 8.7 hours/year

99.999% = Five 9’s
= 5 minutes/year

(Courtesy of Alfred Fuller and Matt Wilder of the Google App Engine Team)

Figure 3-10
On the topic of data storage, there are many choices pertaining to the layout of the usage of Google’s BigTable database. The Java Persistence API (JPA) provides a data storage layout that is similar to the layout of a traditional relational database management system. For the Google App Engine, the data storage implementation does not reflect a traditional database management system. Thus, some features of JPA cannot be used when dealing with the Google App Engine.

According to Google Developers, a feature of JPA that is limited in use with the Google App Engine is class inheritance. In relation to the Google App Engine data storage feature, JPA inheritance is vital in creating entities that are subtypes of other entities. For the controlling of the subsystems, a generic entity for a subsystem needs to be created. Along with the generic entity, specific subsystem entities need to be created for subsystem specific attributes. The JPA inheritance layouts that are applicable to the Google App Engine are the “TABLE_PER_CLASS” and “MAPPED_SUPERCLASSES” layouts.

The “TABLE_PER_CLASS” layout consists of tables being made for the supertype and its subtypes. The supertype will contain its own attributes while the subtypes will contain their own attributes in addition to the attributes of the supertype. The following diagram represents an example of the “TABLE_PER_CLASS” layout being implemented:

![Diagram of TABLE_PER_CLASS layout](image)

The “MAPPED_SUPERCLASSES” layout is similar to the “TABLE_PER_CLASS” layout. The only difference is that the “MAPPED_SUPERCLASSES” layout does not consist of a table for the supertype. This is useful if the subsystems all have unique attributes to be stored into the BigTable database.
3.2.5 - Remote server and a MySQL instance
Another option for the implementation of the front-end software application of the design is to implement a MySQL database instance on a remote server. An Android platform based application would be created for the user interface. The FUNAMBOL JSON Connector would be used for the connection between the application and the server itself. The JSON connector would allow the web-based syncing of data between a mobile device and the server. The following diagram illustrates the JSON Connector’s role in server communication:

![Diagram illustrating JSON Connector's role in server communication](image)

The advantage of using the JSON Connector is the fact that it is compatible with any programming language, according to FUNAMBOL forge. A disadvantage of implementing this option into the design is the fact that the use of a remote server would add onto the costs.

3.2.6 - Web page with local data storage
Another option for the implementation of the front-end software application of the design is to implement a web page for the user interface and the storage of data to a file of a local instance. The web page would take in user input and write necessary values, the usage and electric company information, into a file as a local data storage instance.
The advantage of this design is the ease of writing and reading from the data source. The disadvantage of this design is the lack of security. Simple editing of the file would compromise the stored data.

### 3.2.7 - Radio Frequency and Antenna Design

Since our system is going to communicate over radio frequency, the correct antenna design is necessary to avoid noise and interference which in turn could potentially lose or corrupt valuable data. In addition, we need to choose the most ideal wireless spectrum as allowed per the Federal Communication Commission for our project.

The first antenna we have considered using for communication in our home automation system is a simple monopole antenna oriented in a vertical fashion. This type of antenna has some unique advantages over others that could potentially be desirable for the objectives of our system. A monopole antenna uses a simple coaxial conductor for the antenna and its base for the ground. This would be ideal if we use a metal chassis for the various subsystems.

The second antenna we have considered using for communication in our home automation system is a dipole antenna, as shown in figure 3-13. Dipole antennas also have their own unique advantages and disadvantages. They tend to have excellent uniform dispersion, but the connecting signal cable can be more susceptible to interference.

![Dipole Antenna Diagram](image)

Figure 3-13: Dipole Antenna
In addition to selecting the correct antenna for our communication system, we also need to choose the best frequency spectrum. Since we are more than likely going to use Chipcon (now a part of Texas Instruments) radio frequency transceivers, we shall either use 2.4 gigahertz or sub 1 gigahertz spectrums. Each spectrum has its own unique advantages and disadvantages for our project specifications.

The 2.4 gigahertz spectrum is widely used today for many forms of communication. 802.11 and zigbee wireless networking as well as cordless telephones all share this same spectrum for various forms of communication. Some advantages of 2.4 gigahertz is more allocated bandwidth per the Federal Communications Commission as well as shorter operating distances which can be considered a more secure form of communication. Some disadvantages of using 2.4 gigahertz is that it is a very crowded spectrum with many devices operating on the same or very close frequencies to one another. Examples of such devices are cordless telephone and microwave ovens, both of which are extremely common household items. These devices could cause temporary or permanent interference with our home automation system.

Using a sub 1 gigahertz protocol also has its own unique set of advantages and disadvantages for our home automation system. A primary advantage of using sub 1 gigahertz is the simple nature of electromagnetic waves. Lower frequencies tend to propagate further than higher frequencies, which can be desirable depending on the specific application. Another advantage of using lower frequencies is that the FCC usually allows higher output power at lower frequencies than they do for higher frequencies. However, one disadvantage of using the sub 1 gigahertz spectrum is that it is more susceptible to interference from terrestrial radio and television signals. In addition, the greater propagation of lower frequencies can cause security concerns if the output power is not regulated.

Another option for wireless communication was bluetooth technology. Bluetooth supports more than just radio frequency communication including link and application layer communication. These communications can transmit data and voice commands. The full duplex signals work on an unlicensed band operating at 2.4 to 2.485 gigahertz. Bluetooth technology uses a frequency hopping scheme and a detection algorithm which allows devices to detect other devices in the same spectrum and avoid interference. Bluetooth holds some advantages and disadvantages. Bluetooth communication avoids interference from other wireless devices and does not use a line of sight protocol to send signals. Connectivity is automatically requested when bluetooth devices are in close vicinity. The disadvantages for bluetooth somewhat outweighed the advantages in the group’s decision to go with radio frequency technology. Compared to an radio frequency link, bluetooth is much more susceptible to interruption. This openness can leave holes for intruders and possible hacks. Bluetooth has a slower data rate transfer and more direct communication capabilities compared
to the broadcast the group wishes to implement. Finally, when using the internet through bluetooth the connection can be very slow.
4 - Project Hardware and Software Design Details

4.1 - Project Block Diagrams
This diagram represents the high level architecture of our project. We plan to have an Android phone communicate to a MSP430 powered controller subsystem. We plan to build three controller subsystems. A lighting subsystem which can control LED lights. An air conditioner subsystem which controls a HVAC unit. Finally we will have a television subsystem which controls an entertainment system. We have chosen an architecture like this because we believe that this is our simplest route for our required specifications. The Android phone will have the ability to communicate commands and receive data in the callback of those commands. This will be an event driven system where the user on an Android phone is the driver.

Figure 4-1: High Level Architecture

4.1.1 - Android Platform
We have chosen to start development of our mobile applications to be in android exclusively. We were going to make a cross platform application, however most of our background has been in Java so we feel most comfortable releasing version 1 in Android only. Android and Google App Engine are both Google, so there are lots of pre-built code we can reference as we are programming. We
also want to include a voice command interface. This would allow the user to speak to the phone and tell it a command to execute.

4.1.2 - Google App Engine
The Google App Engine was chosen because of its scalability and quality of service. Because this is running on a Google server we will have top reliability and performance given our code is written properly. A nice feature of the Google App Engine is the ability to scale how much bandwidth you need and use. You will only get charged for the bandwidth you use and there is an initial amount of use that is completely free.

4.1.3 - Stellaris
We chose to use a Stellaris ARM based processor from TI because we were taught how to use during a seminar. We feel that with lightweight IP paired with dynamic DNS it should be able to host a small web server to send commands to the various MSP430 controlled subsystems. Light weight IP allows for simple API calls to be sent to it which can then be relayed to a subsystem.

4.1.4 - MSP430
The MSP430, we believe this chip is more than powerful enough to receive an RF signal and send out a signal. We have also been trained on how to program the MSP430 value line in a seminar. We have also chosen to use the most powerful value line processor because we will know if we need a different chip depending on if it can handle the task or not.

4.1.5 - Packaging
The last thing we wanted to find was an enclosure to house all of our subsystems in. We felt that it is important to have different sizes and shapes because our 4 subsystems are very different. We feel that it is necessary to enclose our product in a case because we are dealing with electricity and sometimes high power applications like in the lighting subsystem.
As shown in figure 4-2 above, this would be an ideal candidate for our lighting subsystem. This case would have a built in dongle for us to plug in our board to and provide power to whatever the application is. We have actually found this exact case and it is only around five dollars each. Our major concern for this is wireless because we also need to make sure the the case does not weaken to signal too much.

**4.2 - Lighting Subsystem**

The lighting subsystem will consist of three main components: a microcontroller unit, RF transceiver, and a digitally controlled potentiometer. Our objective is to send one of several commands from the main controller over a RF link, process these commands, then finally control the basic functions of a lighting system. From the google app engine based interface, the user will be prompted with several labeled sliders and will be able to select how much light he/she wants in a particular room.
4.2.1 - Command Set
1. Full on
2. Full off
3. Percentage of full on (dimmer).

The format of the commands shall be: 
Bank+Percentage+Motion, where bank is room the user wishes to control and percentage is how much light the user wants. %100 shall be full on and %0 shall be full off.

4.2.2 - Physical Lamps
The physical lamps shall consist of light emitting diodes mounted on pcbs in 4x4 clusters. There shall be 3 clusters representing three different rooms to the end user. We considered dimming incandescent lamps running on 110v 60hz ac line voltage, but the hardware needed to convert the signal from the microcontroller unit to higher voltages is complicated and creates a potential safety hazard. In addition, LEDs are much more energy efficient and are the future of lighting technology.

4.2.3 - MCU Power
The microcontroller unit shall receive its commands from a RF transceiver mounted on the same PCB as the microcontroller. The microcontroller unit will receive these commands and store them temporarily in registers. These signals will be be routed to three digital potentiometers running on an I2C bus. Each potentiometer will control the voltage on a particular bank of leds with each bank representing a room to the user.

The controller PCB shall contain the following parts:
- 3x TI TPL0102-100 Digital potentiometer
- 1x TI MSP430G2553 microcontroller
- 1x CC1000 RF transceiver

4.2.4 - Microcontroller
We chose the Texas Instruments MSP430G2553 Value Line microcontroller for the lighting subsystem for a multitude of reasons. First and foremost it is an ultra-low power mcu which allows us to run this system on extremely low voltages, making the subsystem very energy efficient when compared to other mcus. Another reason why we chose the MSP430 is its c-based IDE which allows to us code faster and much more efficiently than an assembly based mcu. The MSP430 will read and transmit data to and from the radio frequency transceiver over a serial bus. This data will contain commands sent from the main controller and status signals from the MSP430 to the main controller. Once this data is written into the mcu’s registers, it shall be processed and sent to the correct pins for the serial bus that will be connected to the digital potentiometers.

4.2.5 - Digital Potentiometers
In order to dim the led clusters, potentiometers will be necessary to vary the constant output voltage of the microcontroller unit to the LED clusters. Instead of traditional analog potentiometers, we will use digital potentiometers that receive commands digitally from a microcontroller. We chose Texas Instruments TPL0102-100 digital potentiometers for several reasons. First, it operates over an I2C bus which is easily implemented in the MSP430 microcontroller. In addition, the TI TPL0102-100 digital pots consume a very small amount of energy when paired with a MSP430 mcu. Each potentiometer will be assigned its own unique I2C address via jumper configuration.

4.2.6 - RF Transceiver
Since our design calls for wireless interoperability between the different subsystems, an rf transceiver will be necessary to transmit and receive data. We shall use a TI CC1000 radio frequency transceiver that operates on a SPI bus. A low level sub 1 gigahertz protocol that will be independently designed will be used to transmit the data.

4.2.8 - LED Arrays
The led arrays will consist of three 4 inch x 4 inch PCBs that will house 16 leds in a square array. There will be an operational amplifier placed before each of these clusters to ensure an adequate buffer between the digital potentiometers and the leds. In addition, there will be resistors placed at each row of LEDs to ensure proper biasing. Pin headers will be used for connectivity to the serial bus from the main pcb.

4.2.9 - Logic and Data Structures
The MSP430 will receive data from the RF transceiver over a 4-pin SPI bus, connected to pins 1.1-1.4 on the MSP430. Since the RF link will be shared with the other subsystems, CDMA multiplexing will be used. The MSP430 will be constantly monitoring these pins for data coded specifically for the lighting subsystem. Any other data destined for other subsystems will be rejected. Once the data destined for the lighting subsystem is received by the MSP430, it will be decoded and compared to the current system state. If the current system state is equal to the requested state, no changes will be made. However, if the current system state is different from what is requested, the request will then be routed to the correct potentiometer on the I2C bus connected to pins 1.5 and 1.6 on the MSP430.

The current state for each cluster of LEDs will be stored in RAM in the form of string arrays and loaded into registers when a request is made.
4.2.9 - Schematics and PCB Layouts

The lighting subsystem control board, as shown in figures 4-3 through 4-7, shall consist of a single pcb with all of the required hardware. The components consist of dual inline package and surface mount parts. All external connections shall be with male pin headers. RF components will be isolated from power components and shall be shielded to avoid interference.

Figure 4-3: Overall Lighting Subsystem Schematic
Figure 4-5: RF Transceiver PCB Layout

Figure 4-6: LED Cluster Schematic
The HVAC subsystem will consist of a temperature sensor, microcontroller unit, radio frequency transceiver, and an LED indicator cluster. The objective for this subsystem is for the user to set a desired temperature via the Google App Engine interface and have the system automatically set the correct fan and heating element parameters and simulate a working thermostat via an indicator cluster.
4.3.1 - Command Set
- Desired temperature > Ambient = Heating system activated
- Desired temperature < Ambient = Cooling system activated

The format of the commands shall be:
room+operation

The operating modes shall be:
- Cool
- Heat
- Emergency Heat
- Fan only

4.3.2 - Main PCB
The mcu shall receive its commands from the main controller over sub 1ghz RF. Once the commands are received, the mcu will compare the current state of the hvac system to the desired state from the user and configure the system correspondingly. The mcu and temperature sensor will be housed on the same pcb with the following components:

1x MSP430G2553 Microcontroller
1x TI TMP100 Temperature Sensor
1x CC1000 RF Transciever
1x LED System Status Indicator Array

4.3.3 - Microcontroller Unit
We chose another TI MSP430G2553 Value Line Microcontroller for the HVAC subsystem. Since the entire subsystem requires the use of only 10 pins, this particular mcu is ideal for the HVAC subsystem. In addition, the data structures necessary to operate this subsystem only require at most 2KB of instructions and data, which is one half the total memory on the MSP430G2553 chip.

4.3.4 - Temperature Sensor
We chose a Texas Instruments TMP100 for sensing the ambient temperature. Based on the temperature detected by the sensor, the system will decide what operating mode to select. This particular sensor operates on an I2c bus making it easy to interface with the MSP430.

4.3.5 - RF Transceiver
A TI CC1000 RF transceiver will be used to transmit and receive data. A low level sub 1ghz protocol that will be independently designed will be used to transmit the data.

4.3.6 - System Status Indicator
To simulate a working HVAC system, an array of multi-colored LEDs will be used to indicate to the user the status of the system. The colors shall indicate as follows:

Blue - Cool
Red - Heat
Orange - Emergency Heat
Green - Fan

The correct combinations of leds for a particular system status shall be as follows:

Blue and Green - Cool
Red and Green - Heat
Orange, Red, and Green - Emergency Heat
Green - Fan Only

Each lead will be connected to pins 2.1-2.4 on the MSP430 in ascending order as listed above. The MSP430 will simply use a 1 on a pin to indicate on and 0 to indicate off.

4.3.7 - Logic and Data Structures
The current system states will be stored as Boolean values in memory on the MSP430. TRUE will indicate that a particular mode is on and FALSE will indicate that a particular mode is off. The requests from the user will be encoded in CDMA with a particular codeset assigned to the HVAC subsystem. If the requests are destined for the HVAC subsystem, the subsystem will decode the data and store it in registers. After the requests are stored, the current system state is loaded into registers and compared with the request. If the request is the same as the current system state, no changes will be made. However, if the request is different from the current system state, the register containing the status information will be changed, setting the correct values on the output pins connected to the LEDs.

4.3.8 - Schematics and PCB Layouts
The HVAC subsystem control board, as shown in figure 4-9 through 4-11, shall consist of a single pcb with all of the required hardware. The components consist of dual inline package and surface mount parts. All external connections shall be with male pin headers. RF components will be isolated from power components and shall be shielded to avoid interference. All LEDS shall be placed at the edge of the board to be mounted into a metal chassis.
Figure 4-10: HVAC PCB
4.4 - Television Control Subsystem

The television control subsystem shall consists of three main parts, an MSP430 ultra-low power TV IR remote control transmitter, a rf module and a Samsung LCD high definition television. Our objective is to send commands from the main controller over an rf link, process these commands, then control a few basic functions of the television. From the google app engine based interface, the user will be prompted with several labeled sliders and will be able to select different command options for the television.

4.4.1 - Command Set
1. TV On
2. TV Off
3. Channel Up/Down
4. Volume Up/Down
5. Channel Number Entry

The control subsystem will use an RC5 protocol consisting of mark-to-space and space-to-mark transitions. The packet transmitted by the RC5 protocol consists of 14 bits which are two start bits, one control bit, five address bits and six command bits. When a packet is transmitted the start bits are always one. The control bit of the packet toggles between 1 and 0 whenever a new key is pressed. The five address bits will be constant pertaining to the address of the IR receiver of the television while the command bits represent 64 commands that can be transmitted. The IR remote will be completely interrupt driven and it will use only 5 I/O ports. For efficiency the remote will be in Low-power mode 4 (LPM4) mode most of the time which is a state where the chip consumes on 0.1 μA, the CPU, ACLK and DCO dc-generator are disabled and the crystal oscillator is stopped.

4.4.2 - Parts List
- 1x TI MSP430F2131 microcontroller
- 1x CC1000 RF transceiver
- 1x TI TMP100 IR transmitter
- 1x Samsung LCD HDTV
- 1x IR LED
- 1x MMBT2222 Transistor
- 2x Capacitors (0.1μF and 200μF)
- 1x 3V CR2032 Lithium coin cell battery
- 3x Resistors (5Ω  402Ω  and 47kΩ)
- 3x 1N4148 Diodes

4.4.3 - CC1000 RF Transceiver
A TI CC1000 RF transceiver will be used to transmit and receive data. A low level sub 1ghz protocol that will be independently designed will be used to transmit the data.

4.4.4 - IR Transmitter
The IR transmitter we will use is a COM-09349 infrared emitter. This device works around a 940-950 nm wavelength and works well with the IR remote design. The LED has a forward voltage of 1.5 VDC and 50 mA max forward current. The transmitter will be will be sourced with a capacitor because the application requires short bursts of high current which that battery itself cannot provide. This capacitor is charged through the 402Ω resistor and discharged through the 5Ω resistor when the IR LED is in the on mode.

4.4.5 - Timing and Signal
The timing for an RC5 packet transmission is a 0.89ms half bit time. A mark-to-space transition protocol will be implemented in which a 38 Khz pulse width modulated signal will be generated as a mark. The digitally controlled oscillator within the microprocessor uses a 1 Mhz operating frequency. The production of a 38 kHz PWM output we use the timer of the MSP430 which will be sourced by the digitally controlled oscillator. The value of 24 sets the CCR0 to a period of 38 kHz while the CCR1 determines the duty cycle. A value of 7 produces a duty cycle of 29% which will be long enough for the IR LED to adjust.

4.4.6 - Samsung Decoder
The IR remote control transmitter requires an infrared wave decoder. Instead of building and implementing a custom controller, the use of an actual television seemed the smarter choice. The television that shall be used is a Samsung HDTV model number LN26A450C1DXZA. The IR protocol determines the bits 1 and 0 by pulse distance. The carrier frequency is 38 kHz in which the frame period is 108 ms. The carrier wave consists of 32 bits, 16 bits of address and 16 bits of data. A 1 bit is on for 650 μs and off for 1500 μs while a 0 bit is on for 650 μs and off for 650 μs. To design a remote to be compatible with the Samsung television there must be compromises with the conflicting protocols. The pulse distance encoding of the bits uses a pulse which is about 650 μs long. A 1 bit takes 2.25 ms to transmit while a 0 bit takes 1.15 ms. The decoder on the television shall detect the start bit. Once the start bit is detected a loop is executed to detect a 0 bit. Once it detects this bit it waits for a high signal until the signal is low again. The first time the loops are executed the variable bit of the timer is zero. The variable bit increments 16 times therefore saving 16 bits of the address of the message. These bits are then converted into hexadecimal numbers.
4.4.7 - Schematics and PCB Layouts

Figure() below is a visual representation of the RC5 data packet transmission protocol. The S1 start bit takes a total of 1.8ms to transmit. The CCR0 and CCR1 determine the period of the clock cycle and the duty cycle respectively. Start bits are labeled S1 and S0, the control bit is labeled C, Bits A4 through A0 are the address bits and the command line is represented by the bits C% through C0.

Figure 4-12: RC5 Data Packet Transmission
Courtesy Texas Instruments
An example of the IR remote control is shown below in figure(). This is a representation circuit diagram of an infrared remote television controller with six switches representing the buttons on the remote. The IR LED transmits the pulse width modulated signal as dictated by the MSP430 to the television demodulator. The MMBT2222 transistor shall be the creator of the signal while the 200 microfarad capacitor supplies power to the IR LED and transistor.
The Eagle Cadsoft schematic in figure() depicts the IR remote control connected to the CC1000 RF transceiver. The RF transceiver uses information from the user to simulate button presses to control the communication between the MSP430 and itself. These messages are then relayed through the infrared LED to the Samsung HDTV decoder. From the Eagle schematic, the printed circuit board is created. The board in figure() shows the nets between each component.
Figure 4-15: IR Remote PCB
The software flowchart diagram in figure() shows the modes and execution of the IR remote through the MSP430. At turn on the MSP430 prepares for an interrupt while waiting in LPM4 consuming only 0.1 μA. Once an interrupt is received it clears the flags and exits low power mode and goes into active mode. After the interrupt is received the microcontroller scans for the message and checks if the data is relevant. If the data is not relevant to its specific task it returns to low power mode, otherwise it decodes the message. Once the message is decoded it sends said message to the television decoder through the infrared light emitting diode.

Figure 4-16: Remote Software Flowchart
4.5 - Main Controller
For the main controller of our system we need it to act as the gateway between the Google App Engine or Android phone and the MSP430 subsystems. For this application we have looked at many options but have chosen to use a Texas Instruments Stellaris because we already have the chip for testing and feel it has more than enough power for our application. We have also decided to try and stick with Texas Instruments chips because we believe they make great technology.

4.5.1 - Raspberry Pi
We first looked at using the popular Raspberry Pi prototyping board. Being recently released, it is one of the more powerful boards out there. Its price tag lies way under fifty dollars as well which made it even more appealing. We decided that since Raspberry Pi was more intended to run embedded linux applications, which is way more power than required for our project, so we decided to do more research.

4.5.2 - Beagle Board
We had then looked at using a Texas Instruments prototyping board called the Beagle Board. This board has a high powered Texas Instruments OMAP processor. Like the Sellaris it also uses an ARM architecture which is great for higher powered embedded applications. After doing more research into this board we decided that since we could not just use it from Texas Instruments and its expensive price tag of one hundred fifty dollars that there had to be a better option.

4.5.3 - Stellaris
Finally we did our research and found the Stellaris. We believe that we can achieve the requirements more efficiently with our own custom board. If we lay out and design everything according to our specifications the design will most likely be better since it was intended to do this specific application. We also feel by using our own custom layout we will gain more useful knowledge.

4.5.4 - Micro Internet Protocol (uIP)
At first we thought that using the most sleek applications would be best because performance was our major enemy. Micro IP was a serious consideration for our implementation of the TCP/IP stack however we feel that this protocol would not provide the type of performance and reliability we require. It lacks many built in features of the IP stack that we felt would be necessary to properly build our project with good throughput and reliability.

4.5.5 - Lightweight Internet Protocol (lwIP)
Lightweight IP has been chosen for our implementation of the TCP/IP stack. Since our project does not require very much data to be transferred we decided to go with this rather than requiring a full IP stack implementation. After doing the
research we also like the simplicity of it because it meshes well with our goal of not over complicating things.

4.5.6 - Dynamic Domain Name Service (DDNS)
One major problem in our TCP/IP stack implementation resided in the application layer. Normally a DHCP client would assign an IP Address and automatically connect it to the internet but this address would only be within the network. However since we plan to use the Google App Engine we need to provide some sort of landing space for communication. Since we do not have a static name space we have chosen to use Dynamic DNS to update our name space.

4.5.7 - API call format
We will be using Hypertext Transfer Protocol (HTTP) to send our API calls. They will all have a similar format by sending data in the GET method.

```
<table>
<thead>
<tr>
<th>Stellaris_URL</th>
<th>/subsystem?</th>
<th>userid&amp;</th>
<th>password&amp;</th>
<th>unit&amp;</th>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.stellaris.com">www.stellaris.com</a></td>
<td>/lighting?</td>
<td><a href="mailto:jedison89@gmail.com">jedison89@gmail.com</a>&amp;</td>
<td>**********&amp;</td>
<td>bedroom1&amp;</td>
<td>lights_on</td>
</tr>
</tbody>
</table>
```

HTTP GET: www.stellaris.com/lighting?jedison89@gmail.com&**********&bedroom1&lights_on

Figure 4-17: API call format

Sample API Call:
This simple implementation will take calls from either the Google App Engine or Android device and will perform the specified task. Because our commands are so simple we can send all of our data in the metadata. We will literally be sending bytes of data to the Stellaris that encode a function. We have chosen to include security on our system by implementing a username / password check each time a user pings the system.

4.5.8 - RF transceiver
The Stellaris needs to communicate messages to the MSP430 subsystems. We have chosen to use a sub 1Ghz module to send data over. We have chosen an Anaren transceiver which has been proven to have great integration with the MSP430. They provide their own proprietary stack called AIR Booster Stack which we will be using to configure the RF communication. We have chosen to use a sub 1Ghz implementation because it goes through walls better and we do not need a high bit rate since we are using a state machine with low data transfer. We also think that competitors on the 2.4Ghz bands such as ZigBee and their Zig Stack provide a product way too advanced and complicated for our project’s requirements.
4.5.9 - USB/Ethernet ports
We have chosen to include a USB and ethernet ports on our board for non wireless I/O communication. The USB will be used to flash the Stellaris and configure everything. The Ethernet will be directly connected to a router so that it has internet. We thought about not included the USB port but finally decided that it would be easier to program the chip if we had one.

4.5.9 - Schematics and PCB Layouts

Figure 4-18: Main Controller Overall Schematic
Figure 4-19: Main Controller USB Debug Interface Schematic
Figure 4-20: Main Controller Stellaris ARM Cortex M3 Schematic

Figure 4-21: Main Controller Stellaris ARM Cortex M3 Schematic (continued)
Figure 4-22: Main Controller Overall PCB
4.6 - Google App Engine
The Google App Engine will be used to provide a simple and sleek platform to build a Java based web application. In the Google App Engine web application, an interface will allow a user to control the lighting, HVAC, and television subsystems and view the calculated energy consumption of the subsystems from the beginning of that particular month to the current date. The Google App Engine was chosen since it allows for connected Android support and an ease of data storage and retrieval via the Datastore application.

4.6.1 - Distributed Data Storage Scheme
As mentioned before, the Datastore feature of the Google App Engine will be utilized for the data storage of our web based application. According to Google Code, the Datastore feature is a schemaless object datastore that includes a query engine and transaction capabilities.

When choosing a Datastore option, there were two to choose from: High Replication Datastore and Master/Slave Datastore. The High Replication Datastore was chosen over the Master/Slave Datastore. The High Replication Datastore was chosen because of the higher reliability, due to the lower error rates when compared to the Master/Slave data storage option. Although the High Replication Datastore option has higher latency when it comes to writing data (Around 2.25x according to Alfred Fuller of the Google App Engine Team), when compared to the Master/Slave option, latency of data reading is of higher priority. This is evident since a large amount of data will need to be read when the user chooses to calculate the current calculated energy costs of using the subsystems. According to Alfred Fuller and Matt Wilder, the latency of reading data between High Replication and Master/Slave are nearly identical. Because of this, the lower error rates of the High Replication Datastore option outweighed the lower latency of the writing operation of the Master/Slave Datastore option.

4.6.2 - Database Layout
For the database layout, 4 tables will be needed. The four tables are as follows: Electricity Provider, Appliance, Home, and TV Control. The Electricity Provider table will consist of the Electricity Provider company name and the residential rate (per kWh). The Home table will hold the first name and last name of the person who owns the Google account, which will be used to log into the application. The Appliance table will be used to hold information regarding the HVAC and Lighting subsystems. The general information on the appliances and the usage date, start time, and end time will be stored into the table. The Television Control table will act as a subclass of the Appliance table. The unique attributes of the Television Control table include the TV brand and the TV code.

The following diagram represents the tables that will be used for storing the subsystems’ energy usage and appliance information:
The Appliance_Name variable was included in order to represent the different rooms using the same type of appliance. An example would be the dimmer system included in the living room and a bedroom. The appliance names would be respectively named “living_room_dimmer” and “bedroom_dimmer” in order to differentiate between the two separate dimmer systems during the querying of the database.

Each of the above tables shall be coded into separate classes. Whenever a new entry needs to be added into the database, a new instance of the respective class will be initiated. In order to keep the integrity of consistent entries within each table of the database, a key variable will be used to indicate the respective table that the class should be inserted into. Classes with the same key variable will be grouped together in the database (Known as “Entity groups”). To insert the newly instantiated class into the database, the appropriate Datastore methods shall be used to insert the data.

4.6.3 - Java Persistence API (JPA) Inheritance Strategy
Inheritance needs to be implemented to store the television control subsystem’s specific attributes. Due to the fact that Google App Engine’s BigTable database
is not treated as a traditional database, there are two inheritance choices to choose from: TABLE_PER_CLASS and MAPPED_SUPERCLASSES. The inheritance strategy that will be used is the TABLE_PER_CLASS strategy. This was chosen over the MAPPED_SUPERCLASSES strategy due to the fact that the other subsystems not having distinct attributes. Because of this, a table with common attributes for a common appliance needed to be created.

### 4.6.4 - Choice of language, SDK, and IDE
The Google App Engine supports two programming languages: Java and Python. We chose to program the Google App Engine web application in Java. Java was chosen due to the plethora of exposure to the language from previous coursework. Since Java will be used, the Google App Engine Java SDK will be downloaded from the Google Developers website. The IDE that will be used for the programming of the Google App Engine will be Eclipse - version 3.7 (Indigo). Eclipse was chosen due to the availability of the necessary plugins for development.

The Google plugin will be used to allow a necessary interface for development and to provide the necessary libraries to be imported. The Android plugin will also be used to provide the necessary libraries for mobile device development. The Android platform that will be used in development is version 2.2. This version of the Android platform was chosen because the libraries associated with the platform are sufficient enough for the planned user interface to exist and function properly, the developers associated with the Google App Engine development have had the most experience with Android 2.2, and the fact, according to ANDROID developers, that around 93 percent of existing Android devices are able to run an Android 2.2 application on their mobile device.

XML will be used as the programming language for the layout of the Android application due to the fact that the default language for Android application layout is XML.

### 4.6.5 - User Interaction
For interaction between users and the application, the users will just be the residents in the household.

- Login → Input electricity provider information
  - Input electricity provider company name
  - Input residential rates (in kWh)
- Login → Input wattage for the automated devices
- Login → Control dimmer system
- Login → Control HVAC system
- Login → Control TV
- Login → Calculate daily usage
- Login → Calculate monthly usage
The following is a use case diagram outlining the user interaction with the Android application:

![Use Case Diagram]

All of the <<extends>> relationships outlined in the diagram represent the subclass functionalities to the pointed superclass function. From the functionality layout above, the sub functions will be chosen through a separate menu.

4.6.6 - Software System Architecture
The software system architecture is of a mixed client-server and layered architecture. The “clients” of the architecture are owners of Android devices. The “server” will be Google’s server space, allocated to the application itself. The Android devices will interact with Google’s server space by sending queries to the BigTable database. The web application directs output, received from the database, to the android devices that sent the correlated queries.

The layered architecture exists as a strict two-layered system that illustrates the interaction between the server application and the Google BigTable database.
The web application interacts with the BigTable database by writing, retrieving, updating, and deleting entries via GQL queries.

The following diagram represents the chosen overall system architecture:

Figure 4-25: Overall Architecture

4.6.7 - Login Screen Layout

The login screen will enable the user to input their Google account username and password in order to login to the web application. The login screen will be the first screen to appear to the user. In order to achieve this functionality, the specification will be coded in the premade Android project file, “AndroidManifest.xml.” If the username or password or both the username and password are invalid inputs, then a message will prompt the user to input values for a valid Google account. The layout will consist of the following Android layout objects (From top to bottom - Vertical Layout):

<p>| Username Textview | This textview will simply output the string, “Google Account Username.” This will be placed at the top of the screen. This will be placed in the layout in order to describe the purpose of the below textfield. |</p>
<table>
<thead>
<tr>
<th>Username Textfield</th>
</tr>
</thead>
<tbody>
<tr>
<td>This textfield will take in the user’s Google account username. The input type will be the default setting (Characters and numbers).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Password Textview</th>
</tr>
</thead>
<tbody>
<tr>
<td>This textview will simply output the string, “Password.” This will be placed right below the textfield for the username. This will be placed in the layout in order to describe the purpose of the below textfield.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Password Textfield</th>
</tr>
</thead>
<tbody>
<tr>
<td>This textfield will take in the user’s Google account password. The input type will be the default setting (Characters and numbers). The inputted text will also be masked with bullet points for the sake of privacy.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Login Button</th>
</tr>
</thead>
<tbody>
<tr>
<td>This button will be pressed when the user has inputted their username and password. If one or both of the textfields are empty, then an error message will prompt the user to input values for the respective textfield(s).</td>
</tr>
</tbody>
</table>

The following screenshot represents the full layout of the login screen (From generated XML code in the Graphical Layout tab of Eclipse):

![Login Screen](image.png)
4.6.8 - Menu Layout
After a successful user login, the application will transition to the menu. The menu will consist of the main functionalities of the software and the credits. The menu will be an instance of the “simple_list_item” object. The menu will consist of the following options to choose from:

- Input Electric Company Values
- Calculate Usage
- Control Subsystem
- Credits

An object, named “MenuClass,” will be created to store the menu item name and the class name that is associated with the menu item. In the .java file for programming the menu, an array of “MenuClass” will be created to store the menu item names and classes associated with them. A function, named “onListItemClick” will be created to handle the selection of the menu. Within “onListItemClick,” a new instance of a “Class” object will be created to store the class that is associated with the chosen menu option. A new instance of the “Intent” object will be created to store the action of opening up the initiated “Class” object. Finally, the intent will be used to run the action via the “startActivity” function.

The following classes will be imported to create the functioning menu:

- android.app.ListActivity
- android.content.Intent
- android.os.Bundle
- android.view.View
- android.widget.arrayAdapter
- android.widget.ListView

The following screenshot represents the full layout of the menu (From generated XML code in the Graphical Layout tab of Eclipse):
4.6.9 - Input Appliance Information

In order to perform the necessary calculations of the electricity usage of the subsystems, the wattages of the appliances must be known before making the calculations. The wattages of the appliances shall be inputted by the user. The user shall be able to input the wattages of every controlled appliance. Also, the television code must be entered by the user in order for said subsystem to be fully operational.

To fulfill this need, an “Appliance Inputs” interface has been created. This will be one of the options, on the menu screen, that can be chosen by the user. Through the interface, the user shall be able to enter in the television code and the wattages of the appliances. On top of the above mentioned functionalities, the user will also be able to view the current television code and appliance wattage settings. The wattages for the various lighting fixtures of the dimming system and the components of the HVAC system shall be inputted through the use of an Android spinner object. The spinner will allow the user to choose a specific appliance for the subsystems. Once the appliance has been chosen, the user shall enter in the wattage in the respective textfield. After the wattage has been inputted by the user, the respective “ENTER” button shall be pushed.
To avoid missing entries in the database, if there are no inputs for the wattages nor the television code, then the software will prevent the user from controlling any of the subsystems. The decision to implement this function was decided due to the possibility of a scenario where the user will be unable to calculate a monthly or daily cost due to the lack of wattages in the database. Also, if the television code was never inputted by the user, then the television subsystem would not be operational.

The layout shall consist of the following Android layout objects:

<table>
<thead>
<tr>
<th>Title textview</th>
<th>Displays the string, “Appliance Inputs,” as the title of the interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Television code textview</td>
<td>Displays the string, “Television code,” to prompt the user to enter in the television code into the following textfield</td>
</tr>
<tr>
<td>Television code textfield</td>
<td>Allows the user to enter in a numeric input for the television code. This television code will be used to control the television in the TV control subsystem</td>
</tr>
<tr>
<td>Television code ENTER button</td>
<td>When pressed, the inputted television code shall be stored into the BigTable database</td>
</tr>
<tr>
<td>Wattages textview</td>
<td>Displays the string, “Wattages,” to indicate the functionality of the following Android objects</td>
</tr>
<tr>
<td>Dimmer textview</td>
<td>Displays the string, “Dimmer,” to prompt the user to input wattages for the light fixture(s) affiliated with the dimming system</td>
</tr>
<tr>
<td>Dimmer spinner</td>
<td>Stores the names of the individual lighting fixtures. The user shall be able to choose a certain lighting fixture for wattage input. An “ArrayAdapter” instance, within the .java file, will be used to store the strings of the individual lighting fixtures to be inserted into the spinner</td>
</tr>
<tr>
<td>Dimmer textfield</td>
<td>Allows the user to enter in a numeric input for the wattage of the chosen lighting fixture.</td>
</tr>
<tr>
<td>Dimmer ENTER button</td>
<td>When pressed, the inputted dimmer lighting fixture wattage shall be stored into the BigTable database</td>
</tr>
<tr>
<td>HVAC textview</td>
<td>Displays the string, “HVAC,” to indicate the functionality of the following spinner and textfield objects</td>
</tr>
<tr>
<td>HVAC spinner</td>
<td>Stores the names of the individual HVAC components. The user shall be able to choose a certain HVAC component for wattage input. An “ArrayAdapter” instance, within the .java file, will be used to store the strings of the individual HVAC components to be inserted into the spinner</td>
</tr>
<tr>
<td>HVAC textfield</td>
<td>Allows the user to enter in a numeric input for the wattage of the chosen HVAC component</td>
</tr>
<tr>
<td>HVAC ENTER button</td>
<td>When pressed, the inputted HVAC component wattage shall be stored into the BigTable database</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TV wattage textview</td>
<td>Displays the string, “TV,” to prompt the user to input the wattage for the television in the following textfield</td>
</tr>
<tr>
<td>TV wattage textfield</td>
<td>Allows the user to enter in a numeric input for the wattage of the television</td>
</tr>
<tr>
<td>TV wattage ENTER button</td>
<td>When pressed, the inputted television wattage shall be stored into the BigTable database</td>
</tr>
<tr>
<td>Current Settings button</td>
<td>When pressed, the user will be transitioned to a window showing the current inputted television code and wattages of the various appliances of the subsytems</td>
</tr>
</tbody>
</table>

The following screenshot represents the full layout of the Appliance Inputs interface (From generated XML code in the Graphical Layout tab of Eclipse):

![Figure 4-28: Appliance Input Interface](image)

4.6.10 - Current Settings Window
The current settings window will allow the user to view the current set wattages of the various appliances for the subsystems. It will also display the current television code, used for television subsystem control. In order to not overwhelm the user with the plethora of wattages, the interface will display wattages for one lighting fixture and one HVAC component at a time, from the user’s choice. The user will be able to choose the component through a spinner object on the interface. The interface will always display the television code and the television wattage due to the fact that only one television set will be controlled.

The current settings shall be accessed through the “Appliance Inputs” interface, accessed through the main menu. If user inputs for the wattages and television code do not exist in the database, then a message will appear, via the press of the “Current Settings” button through the “Appliance Inputs” interface, prompting the user to input valid values before checking the current settings.

The layout shall consist of the following Android layout objects:

<table>
<thead>
<tr>
<th>Title textview</th>
<th>Displays the string, “Current Settings,” as the title of the interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Television code textview</td>
<td>Display the string, “Television code: ,” followed by the current television code inputted by the user</td>
</tr>
<tr>
<td>Wattages textview</td>
<td>Displays the string, “Wattages,” to indicate that the following textviews will display the wattages for the appliances of the subsystems</td>
</tr>
<tr>
<td>Dimmer textview</td>
<td>Displays the string, “Dimmer,” to indicate the purpose of the following textviews</td>
</tr>
<tr>
<td>Dimmer lighting fixture textview</td>
<td>Displays the string, “Fixture: ,” followed by the lighting fixture chosen by the user</td>
</tr>
<tr>
<td>Dimmer lighting fixture wattage textview</td>
<td>Displays the string, “Wattage: ,” followed by the wattage of the chosen lighting fixture</td>
</tr>
<tr>
<td>HVAC textview</td>
<td>Displays the string, “HVAC,” to indicate the purpose of the following textviews</td>
</tr>
<tr>
<td>HVAC component textview</td>
<td>Displays the string, “Component: ,” followed by the HVAC component chosen by the user</td>
</tr>
<tr>
<td>HVAC component wattage textview</td>
<td>Displays the string, “Wattage: ,” followed by the wattage of the chosen HVAC component</td>
</tr>
<tr>
<td><strong>Lighting fixture spinner textview</strong></td>
<td>Displays the string, “Lighting Fixture,” to indicate the purpose of the following spinner</td>
</tr>
<tr>
<td><strong>Lighting fixture spinner</strong></td>
<td>Used for the choice of the lighting fixture for the dimmer subsystem. An ArrayAdapter instance will be initiated in the .java file to populate the spinner with the names of every lighting fixture for the dimmer system.</td>
</tr>
<tr>
<td><strong>HVAC component spinner textview</strong></td>
<td>Displays the string, “HVAC Component,” to indicate the purpose of the following spinner</td>
</tr>
<tr>
<td><strong>HVAC component spinner</strong></td>
<td>Used for the choice of the HVAC component for the HVAC subsystem. An ArrayAdapter instance will be initiated in the .java file to populate the spinner with the names of the components of the HVAC system.</td>
</tr>
<tr>
<td><strong>ENTER button</strong></td>
<td>When pressed, the chosen appliances, from the 2 spinners, will be inputted and concatenated within a GQL query, as a string. The query will then be ran and the resulting wattages will be extracted from the returned table entries. The wattages will then be concatenated with their respective textviews and then outputted to the screen.</td>
</tr>
</tbody>
</table>

The following screenshot represents the full layout of the Current Settings interface (From generated XML code in the Graphical Layout tab of Eclipse):
4.6.11 - Electricity Provider Information Layout

In order for the calculation of electricity usage of the controlled appliances to take place, the residential rate of the chosen electrical company must be inputted by the user. If the “Input Electric Company Values” option is chosen from the menu, then the user will be transitioned to an interface asking for the electricity provider’s company name and residential rate. The residential rate will be used to calculate the daily and monthly electricity costs. The layout will consist of the following Android layout objects (From top to bottom - Vertical Layout):

<table>
<thead>
<tr>
<th>Title textview</th>
<th>Displays the string, “Electricity Provider Information,” as the title of the interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company information textview</td>
<td>Displays the most recently entered electric company’s information. If there is no stored information, then “N/A” will be used as a placeholder</td>
</tr>
<tr>
<td>Company name textview</td>
<td>Displays the string, “Company Name,” as an indicator of the value to be inputted into the following textfield</td>
</tr>
<tr>
<td>Company name textfield</td>
<td>Takes in an input of an alphanumeric type. This will serve as the electricity provider’s company name.</td>
</tr>
</tbody>
</table>
Residential rate  | Displays the string, “Residential Rate,” as an indicator of the value to be inputted into the following textview
---|---
Residential rate | Takes in an input of a monetary type ($X.XX). This will serve as the residential rate of the electricity provider.
textfield | 
ENTER button | This button will be used to prompt the software to save the inputs into the BigTable database. The company information textview will be refreshed and display the inputted information as a confirmation of data storage.

The following screenshot represents the Electricity Provider Information interface (From generated XML code in the Graphical Layout tab of Eclipse):

![Electricity Provider Menu](image)

**Figure 4-30: Electricity Provider Menu**

**4.6.12 - Usage Calculation Interface Layout**
The usage calculation interface will allow the calculation of the daily usage, according to the selected date, and the monthly usage, according to the selected month. If the “Calculate Usage” menu item is selected, then the user will be transitioned to the “Calculate Usages” interface.

For both calculations, the software will access Google’s BigTable database on the server space, allocated to the application, to retrieve entries pertaining to the usage of the controlled appliances. For a daily usage, the date selected will be
used as a parameter, the “WHERE” clause, when building the query for the database. The usages will be calculated for each appliance. Because of this, 3 separate GQL queries will be constructed, but with the same parameters (other than the table name).

Three queues, for each subsystem, will be used to store the returned entries of the queries. One queue will be dequeued at a time. For each entry in the queues, the runtime of the appliance will be calculated and the result, in seconds, will be added to a respective “int” variable. The “int” type variable was chosen to store the number of seconds due to the fact that the type is able to hold the number of seconds in a 31-day period, the maximum amount of time to be used in the calculation of the monthly usage. This will be done until the queues are empty. The queue data structure was chosen due to the simplicity, coding wise, of having to just dequeue from the queue when performing the calculation and the fact that the queue is a dynamic data structure. A dynamic data structure is needed since the number of entries returned from a query will always be unknown before running the software. After the total daily usage, in seconds, has been calculated, the number will be converted into hours. The residential rate, per kWh, and the wattage of the appliances will also be retrieved from the database. If the date selected yields no returned entries, then a message will prompt the user to enter in another date. The following formula will be used to calculate the daily usage of each subsystem:

\[ \text{Cost} = (\text{Rate}_{\text{per kWh}}) \times \left( \frac{(\text{Wattage}) \times (\text{Usage in hours})}{1000} \right) \]

Once the daily costs of the three appliances have been calculated, the calculated daily costs will be added to calculate the daily cost of the three subsystems. The total daily cost will finally be outputted to the user in the following format:

Month-Day-Year = $X.XX

The calculation of the monthly cost will be of a similar process. Instead of having a particular date as the parameter for the queries, a range of dates will be used. As before, three queues will be used to store the returned entries from the database. The three “int” variables will store the total usage in a month, in seconds, for each subsystem. The following pseudo code represents the “WHERE” clause parameter for the queries:

WHERE Appliance_Name = <insert appliance name>
    AND Usage_Date >= DATETIME(<first day of the month>)
    AND Usage_Date <= DATETIME(<last day of the month>)

As before, there will be 3 separate queries for each subsystem usage retrieval from the BigTable database. The formula for the cost will be reused to calculate the monthly electricity usage cost.
The layout of the calculating usage cost interface will require the following Android layout objects:

<table>
<thead>
<tr>
<th>Title textview</th>
<th>Displays the string, “Electricity Provider Information,” as the title of the interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result textview</td>
<td>Displays the resulting daily cost or the monthly cost, depending on the chosen calculation</td>
</tr>
<tr>
<td>Daily usage textview</td>
<td>Displays the string, “Daily usage,” as an indicator of the type of value to be inputted into the following “Date Picker” object</td>
</tr>
<tr>
<td>Date Picker</td>
<td>Allows the user to choose a particular date for the daily usage calculation. The user will be able to choose the month, date, and year of the target date</td>
</tr>
<tr>
<td>Daily usage ENTER button</td>
<td>If this button is clicked, then the different fields in the “Date Picker” object will be taken in, via an onClickListener, and be placed into “int” variables. The resulting “int” variables will be concatenated into a string with a partially completed GQL query and be ran as a parameter for the full GQL query.</td>
</tr>
<tr>
<td>Monthly usage textview</td>
<td>Displays the string, “Monthly Usage,” as an indicator of the type of value to be chosen from the following “Spinner” object.</td>
</tr>
<tr>
<td>Month choice spinner</td>
<td>Allows the user to choose one of the listed 12 months. In order to place the 12 months into the spinner, a new .xml file will be created to store the month strings. In the .java file associated with the .xml layout, an ArrayAdapter&lt;CharSequence&gt; will be used to populate the spinner itself.</td>
</tr>
<tr>
<td>Monthly usage ENTER button</td>
<td>If this button is clicked, then the month chosen in the “Spinner” object will be taken in, via an onClickListener, and be concatenated into the GQL query, which will then be ran to retrieve database entries pertaining to the usage according to the chosen month.</td>
</tr>
</tbody>
</table>

The following diagrams represent the layout of the “Calculate Usages” interface, with the resulting daily usage choice and the resulting monthly choice respectively (From generated XML code in the Graphical Layout tab of Eclipse):
4.6.13 - Dimmer Control Interface
The dimmer control interface will enable the user to control the dimmer subsystem. The dimmer is controlled in three different rooms: living room,
kitchen, and bedroom. The dimmer will control 4 lighting fixtures in the living room: 2 lamps, the fan lights, and the ceiling lights. The dimmer will control 3 lighting fixtures in the bedroom: desk light, fan lights, and the corner lamp. The dimmer will control 2 lighting fixtures in the kitchen: the ceiling lights and the dining lights.

The layout will enable the user to choose anywhere from 1 lighting fixture to every lighting fixture to control the lighting level. The process of changing the lighting level of a lighting fixture consists of a user choosing the lighting fixtures and then changing the lighting level via a scroll bar.

The layout of the calculating usage cost interface will require the following Android layout objects:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title textview</strong></td>
<td>Displays the string, “Dimmer Control,” as the title of the interface</td>
</tr>
<tr>
<td><strong>Living room control textview</strong></td>
<td>Displays the string, “Living Room,” as an indicator of the subsystem that the below checkboxes will control</td>
</tr>
<tr>
<td><strong>Lamp checkbox #1</strong></td>
<td>Checkbox for the selection of a lamp in the living room</td>
</tr>
<tr>
<td><strong>Lamp checkbox #2</strong></td>
<td>Checkbox for the selection of the second lamp in the living room</td>
</tr>
<tr>
<td><strong>Fan lights checkbox</strong></td>
<td>Checkbox for the selection of the fan lights in the living room</td>
</tr>
<tr>
<td><strong>Ceiling lights checkbox</strong></td>
<td>Checkbox for the selection of the ceiling lights in the living room</td>
</tr>
<tr>
<td><strong>Bedroom control textview</strong></td>
<td>Displays the string, “Bedroom,” as an indicator of the subsystem that the below checkboxes will control</td>
</tr>
<tr>
<td><strong>Desk lamp checkbox</strong></td>
<td>Checkbox for the selection of the desk lamp in the bedroom</td>
</tr>
<tr>
<td><strong>Fan lights checkbox</strong></td>
<td>Checkbox for the selection of the fan lights in the bedroom</td>
</tr>
<tr>
<td><strong>Corner lamp checkbox</strong></td>
<td>Checkbox for the selection of the corner lamp in the bedroom</td>
</tr>
<tr>
<td><strong>Kitchen control textview</strong></td>
<td>Displays the string, “Kitchen,” as an indicator of the subsystem that the below checkboxes will control</td>
</tr>
<tr>
<td>Kitchen ceiling lights checkbox</td>
<td>Checkbox for the selection of the ceiling lights in the kitchen</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Dining lights checkbox</td>
<td>Checkbox for the selection of the dining table lights in the kitchen</td>
</tr>
<tr>
<td>Dimming seekbar</td>
<td>This operable seekbar will allow the user to change the brightness of the lights. The values of the seekbar towards the left indicate a lower brightness level whereas the values of the seekbar towards the right indicate a higher brightness level.</td>
</tr>
</tbody>
</table>

The following diagrams represent the layout of the “Dimmer Control” interface (From generated XML code in the Graphical Layout tab of Eclipse):

![Dimmer Control Menu](image)

**Figure 4-32: Dimmer Control Menu**

### 4.6.14 - HVAC Control Interface

The HVAC control interface will allow the user to operate the following functions for the controlled HVAC system: Fan, heating and emergency heating systems, and the cooling system. The fan system can be turned on, off, or auto at any time. The heating and cooling systems can only be activated with the correct desired temperature. If the user would like to operate the heating system, then a temperature, greater than the actual temperature, needs to be inputted. If the
user would like to operate the cooling system, then a temperature, less than the actual temperature, needs to be inputted.

The layout of the HVAC control interface will require the following Android layout objects:

<table>
<thead>
<tr>
<th>Title textview</th>
<th>Displays the string, “HVAC” as the title of the interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired temperature textview</td>
<td>Displays the user inputted desired temperature</td>
</tr>
<tr>
<td>Actual temperature textview</td>
<td>Displays the current temperature</td>
</tr>
<tr>
<td>Fan textview</td>
<td>Displays the string, “Fan,” as an indicator of the functionality of the following “Spinner” object</td>
</tr>
<tr>
<td>Fan spinner</td>
<td>Enables the user to choose whether to set the fan system to “AUTO,” “OFF,” or “ON.”</td>
</tr>
<tr>
<td>System textview</td>
<td>Displays the string, “System,” as an indicator of the functionality of the following “Spinner” object</td>
</tr>
<tr>
<td>System spinner</td>
<td>Enables the user to choose whether to choose to turn on the heating system, cooling system, or emergency heating system</td>
</tr>
<tr>
<td>User input temperature textfield</td>
<td>Displays the string, “Enter Desired Temperature.” Prompts the user to press on the following textfield to enter in the desired temperature</td>
</tr>
<tr>
<td>Desired temperature textfield</td>
<td>Allows the user to input a desired temperature</td>
</tr>
<tr>
<td>ACTIVATE button</td>
<td>If the button is pressed, then the desired temperature, fan option, and system option will be taken in as input. If the desired temperature is less than the actual temperature and the heating system option was chosen, then the software will prompt the user to input an appropriate value that is greater than the current temperature. If the desired temperature is greater than the actual temperature and the cooling system is chosen, then the software will prompt the user to input an appropriate value that is less than the actual temperature. The inputted desired temperature will be outputted to the desired temperature textview.</td>
</tr>
</tbody>
</table>

The following diagrams represent the layout of the “HVAC Control” interface (From generated XML code in the Graphical Layout tab of Eclipse):
4.6.15 - TV Control Interface
The TV control interface will allow the user to control their television set. The TV control interface will enable the user to perform the following operations: Set the appropriate television code, turn the TV on/off, lower/increase the volume, change the channel to the next channel or previous channel, and go to the user inputted channel.

The layout of the TV control interface will require the following Android layout objects:

<table>
<thead>
<tr>
<th>Title textview</th>
<th>Displays the string, “TV Control” as the title of the interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWER button</td>
<td>Enables the user to turn the television on and off</td>
</tr>
<tr>
<td>Volume control textview</td>
<td>Displays the string, “Volume Control,” as an indicator of the functionality of the 2 following buttons</td>
</tr>
<tr>
<td>Volume up button</td>
<td>Enables the user to increase the volume by one increment (“+” symbol)</td>
</tr>
<tr>
<td>Volume down button</td>
<td>Enables the user to decrease the volume by one increment (&quot;-&quot; symbol)</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Channel control textview</td>
<td>Displays the string, “Channel Control,” as an indicator of the functionality of the 2 following buttons</td>
</tr>
<tr>
<td>Channel up button</td>
<td>Enables the user to change the channel to the next channel</td>
</tr>
<tr>
<td>Channel down button</td>
<td>Enables the user to change the channel to the previous channel</td>
</tr>
<tr>
<td>Channel selection textview</td>
<td>Displays the string, “Channel Selection,” as an indicator of the functionality of the following textfield</td>
</tr>
<tr>
<td>Channel selection textfield</td>
<td>Enables the user to input up to a 3-digit numerical value, which will indicate the channel to jump to</td>
</tr>
<tr>
<td>ENTER button</td>
<td>If this button is pressed, then the input taken in from the channel selection textfield will be used as the channel for the television to jump to</td>
</tr>
</tbody>
</table>

The following diagrams represent the layout of the “TV Control” interface (From generated XML code in the Graphical Layout tab of Eclipse):
4.6.16 - Credits Page
The credits page shall simply credit the creators of “2012: A Home Odyssey.” The credits page shall be accessed through the main menu. The credits page shall simply consist of one textview containing the titles of the creators. The following diagrams represent the layout of the “TV Control" interface (From generated XML code in the Graphical Layout tab of Eclipse):
4.6.17 - Tabbing Subsystems

In order to conveniently have all the subsystems to be available to be controlled by the user, a TabHost Android instance shall be instantiated in a separate Android .xml layout file. The tabbing instance was chosen due to the convenience of a simple press on a tab to transition to a different layout. It was also chosen due to the simplicity of attaching an existing .xml layout file to a specific tab.

The TabHost instance will be instantiated in the tab.xml Android layout file. The TabHost’s functionality will be included in the .java file. The .java file will initiate the layout assignments and the respective onClickListeners. Three onClickListeners will be used to allow the ability to press on the tab via the user interface to access the specific layout.

The layout of the tabbing interface will require the following layout:
<table>
<thead>
<tr>
<th>TAB #1</th>
<th>Displays the dimmer interface Android .xml layout. The dimmer.xml layout file will be tagged to the leftmost tab. A lightbulb icon will be used to showcase the functionality of this tab. The icon will be attached to the tab layout via the .xml Android layout file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAB #2</td>
<td>Displays the HVAC interface Android .xml layout. The hvac.xml layout file will be tagged to the second to the leftmost tab. A fan icon will be used to showcase the functionality of this tab. The icon will be attached to the tab layout via the .xml Android layout file.</td>
</tr>
<tr>
<td>TAB #3</td>
<td>Displays the television control interface Android .xml layout. The tv.xml layout file will be tagged to the rightmost tab. A stock television set icon will be used to showcase the functionality of this tab. The icon will be attached to the tab layout via the .xml Android layout file.</td>
</tr>
</tbody>
</table>
5 - Design Summary of Hardware and Software

5.1 Overview
2012: A Home Odyssey shall be a home automation system controlled by one or multiple users from an android smartphone or personal computer. The backbone of the architecture shall be the use of the google app engine to send and receive commands, communicated between users and subsystems. The system is divided into three separate subsystems each doing an individual home automated task. The lighting subsystem shall control an array of LED lights, the television subsystem shall contain an IR remote control much like a television remote control and an HVAC control system to measure and change the ambient temperature of the home.

The user shall input commands into his or her Android smartphone or personal computer which then leads to either the google app engine of the wireless router. The use of the google app engine is to use the 3G and 4G networks to communicate from the Android smartphone to a wireless router. If the phone is connected wirelessly to the router itself it shall not use the google app engine. From the wireless router, through an ethernet LAN, the information is sent to a central base station. The base station then will communicate through radio frequency signals to each individual subsystem.

Each subsystem shall revolve around the use of a Texas Instruments MSP430. The central base station shall contain a Texas Instruments Stellaris ARM based processor to host a small web server and to communicate remotely to each of the subsystems. The wireless router will communicate with either the google app engine or directly to the user’s Android smartphone through WiFi. The central Stellaris controller shall use RF transceivers to communicate to the subsystems.

Labview is a circuit simulation tool which we are using to simulate each of the subsystems to test for any defects or imperfections that may occur. The schematics as well as the printed circuit board layouts are created using Eagle software. We used Eagle software because it was presented to the engineering students by Eagle software and we were given free versions of the program. The programming languages used to control the hardware and software are Java and XML.

As engineers, a main objective is to reduce size and cost. To achieve this goal we chose ultra-low power MCUs and small circuit board ranges to ensure low consumption. The non-invasive size of components shall make it an attractive addition of a user’s home. The cost of our product is much lower than other competitors home automation systems due to the parts chosen and low consumption of power.
Each design has been carefully researched and planned by each member on his individual tasks. The designs are then overviewed by each group member and the group as a whole to validate the logic and thought behind the ideas conveyed.

5.2 Lighting Subsystem
The lighting subsystem contains three main components: a microcontroller unit, a radio frequency transceiver and digitally controlled potentiometers. This system will communicate through the stellaris base station and the user input to control an LED array. The LED shall be able to turn on, turn off and dim to a certain percentage.

5.2.1 - Microcontroller Unit
The MCU chosen for the lighting subsystem is the TI MSP430G2553 Value Line microcontroller. This MCU shall use ultra low power consumption to process the commands received from the user. The MSP430 shall directly speak to the RF transceiver and two digital potentiometers. The main task of the MCU shall be to decode the information from the RF transceiver, in doing so, send the correct information to the potentiometers. The microcontroller tells the potentiometer what percentage to dim to, to turn on or to turn off.

5.2.2 - RF Transceiver
The RF transceiver shall receive many different frequency signals. The receiver chosen for this task is the TI CC1000. The job of the transceiver shall be to determine which signal is specifically meant for the lighting subsystem depending on the frequency. Once the correct frequency is received from the Stellaris base station, it will then be sent to the MSP430 to be decoded and implemented.

5.2.3 - Digital Potentiometers
The TI TPL0102 - 100 digital potentiometers are used to dim the LED cluster to the desired percentage of light. The potentiometers are used to vary the output voltage to the LED array to dictate the amount of light they shall give off. The potentiometers communicate with the MSP430 to determine the correct output voltage and send that voltage to the LED array.

5.2.4 - LED Arrays
The LED arrays shall be arranged on four inch by four PCBs which contain 16 light emitting diodes in a square formation. These arrays are controlled through the potentiometer which is in turn controlled by the user through multiple levels of logic.

5.2.5 - Lighting Overview
The goal of the lighting subsystem design is to use low power, cost effective, energy efficient and safe parts and implementations to represent a potential
lighting control system in the home. Using the MSP430G2553 allows low power consumption. The use of LEDs instead of incandescent bulbs reduces the risk of electric shock due to the use of high voltage and reduces the power usage. Finally, the RF transceiver removes the need of a wired home system reducing clutter and components.

5.3 HVAC Subsystem

The HVAC system contains four parts: a temperature sensor, microcontroller unit, RF transceiver and an LED indicator cluster. The goal of the HVAC subsystem is to control the ambient temperature inside a home through an Android smartphone user interface with commands such as cooling, heating, emergency heat and fan only.

5.3.1 - Microcontroller Unit

The MCU chosen for the HVAC subsystem is the TI MSP430G2553 Value Line microcontroller. This MCU shall use ultra low power consumption to process the commands received from the user. The MSP430 shall directly speak to the RF transceiver, the temperature sensor and the LED array. The main task of the MCU shall be to decode the information from the RF transceiver, in doing so, send the correct information to the status LEDs. The microcontroller tells the LED array which indicator to light.

5.3.2 - Temperature Sensor

The temperature sensor shall communicate with the MCU determining what the ambient temperature is currently. This task helps the user decide to turn on cooling, heating, emergency heat or fan.

5.3.3 - RF Transceiver

The RF transceiver shall receive many different frequency signals. The receiver chosen for this task is the TI CC1000. The job of the transceiver shall be to determine which signal is specifically meant for the HVAC subsystem depending on the frequency. Once the correct frequency is received from the Stellaris base station, it will then be sent to the MSP430 to be decoded and implemented.

5.3.4 - LED Cluster

The LED cluster will act as a system status indicator. The array shall be used to display which mode is currently being used by the user. Four LEDs, each connected to resistors in series, have color schemes indicated the current mode. The colors of the LEDs shall be blue, green, red and orange. If blue and green are on, the system is in cool mode. If red and green are on, the system is in heat mode. If orange, red and green are all on, the system is currently in emergency heat mode. Finally, if only the green indicator LED is on, the system is in fan only mode.
5.3.5 - HVAC Overview
The HVAC subsystem for our particular use will not physically access an air conditioning unit but shall display which mode the said AC unit is on. The MSP430G2553 is used to confirm low power consumption and low cost. A LED status indicator is used to ease the testing process. Finally, the radio frequency transceiver removes the need of a wired home system reducing clutter and components.

5.4 Television Subsystem
The television control subsystem consists of three main components: an MSP430F2131 ultra-low power TV IR remote control transmitter, a rf transceiver and a Samsung HDTV. The objective of the television subsystem is to take commands from a user’s Android smartphone, process these commands, and use the IR remote controller to convey these commands to the television.

5.4.1 - Microcontroller Unit
The task of the MSP430 itself shall be to decode the given commands through the RF transceiver. These decoded commands shall be sent through a 40 kHz infrared signal through the IR LED to the Samsung HDTV. Such commands shall be to turn the TV on/off, change the channel up/down, volume up/down and enter a certain channel number.

5.4.2 - RF Transceiver
The RF transceiver shall receive many different frequency signals. The receiver chosen for this task is the TI CC1000. The job of the transceiver shall be to determine which signal is specifically meant for the television subsystem depending on the frequency. Once the correct frequency is received from the Stellaris base station, it will then be sent to the MSP430 to be decoded and implemented.

5.4.3 - IR Transmitter
The IR transmitter is a simple IR LED. The LED transmits infrared waves around a 940-950 nanometer wavelength. The signals sent by this transmitter shall correlate with the specific coding to the Samsung HDTV model number LN26A450C1DXZA.

5.4.4 - Clock timing and Signal
The protocol that shall be used for packet transmission is RC5. This protocol uses 40 kHz pulse width modulation. Using a mark-to-space transitional protocol, a mark shall denote the beginning of a transmission and a space shall denote no transmission.

5.4.5 - Television Subsystem Overview
The television subsystem uses an ultra low power MSP430 which implements a low power mode when not in use where it only consumes 0.1 μA. This mode helps us to achieve our efficiency goals which include low power consumption and low cost products. The user shall use an interface on an Android smartphone to control the IR remote controller. The user sends a command through either a wireless router or a 3G/4G network to the google app engine. These commands are processed then sent to the Stellaris base station for delivery. The base station shall send out a particular radio frequency meant for the television subsystem. The television subsystem then receives this signal through the RF transceiver. The RF transceiver sends these codes to the MSP430 which then decodes the information. This information shall consist of a certain command. These commands are TV on, TV off, channel up, channel down, volume up, volume down and channel number entry.

5.5 - Main Controller
The main controller is designed to handle the traffic between the user, the network, the Google App Engine, the Android phone, and the MSP430 subsystems. These tasks require a powerful controller that can handle the traffic with low power usage and delay. The controller we chose is the Texas Instruments Stellaris because it has these capabilities and is a cheap alternative to some other possibilities.

5.5.1 - Stellaris
The Stellaris shall be placed on its own PCB separate from the other subsystems. The Stellaris shall be the central base station and act as a communication hub between all of the subsystems of the 2012: A Home Odyssey. The Stellaris shall be paired with an RF transceiver to send signals to and from the user and the MSP430 based subsystems.

5.5.2 - Lightweight Internet Protocol
The LwIP has been chosen for communication between the main controller and the subsystems. The Lightweight Internet Protocol uses a TCP/IP stack. The TCP/IP stack is key for this application because it establishes a connection between the controllers throughout the system and between the Google App Engine.
5.5.3 - API Call Format

<table>
<thead>
<tr>
<th>Stellaris_URL</th>
<th>/subsystem?</th>
<th>userid&amp;</th>
<th>password&amp;</th>
<th>unit&amp;</th>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.stellaris.com">www.stellaris.com</a></td>
<td>/lighting?</td>
<td><a href="mailto:jedison89@gmail.com">jedison89@gmail.com</a>&amp;</td>
<td>**********&amp;</td>
<td>bedroom1&amp;</td>
<td>lights_on</td>
</tr>
</tbody>
</table>

HTTP GET: www.stellaris.com/lighting?jedison89@gmail.com&**********&bedroom1&lights_on

Figure 5-1

5.5.4 - RF Transceiver
The RF transceiver shall receive many different frequency signals. The receiver chosen for this task is the TI CC1000. The job of the transceiver shall be to determine which signal is specifically meant for the Stellaris main controller system depending on the frequency. Once the correct frequency is received from the Android smartphone and Google App Engine, it will then be sent to the MSP430 to be decoded and implemented.

5.5.5 - USB/Ethernet Ports
The USB and ethernet ports shall be used for non wireless I/O communication. The USB is used to flash the Stellaris controller to configure the main controller to the system as a whole. The main controller is the only component of the system that is directly connected via ethernet to the router.

5.5.6 - Stellaris Main Base Station Overview
The Stellaris acts as a central main base station. It is the communication hub that connects the television, lighting and HVAC subsystems to the Google App Engine, the Android user’s phone and the internet. The key benefits of using the TI Stellaris board is low power, low cost, efficient size and connectivity. The Stellaris also implements well being a Texas Instruments part.

5.6 Google App Engine
The Google App Engine shall be used to build a Java based web application for Android smartphones. The Google App Engine shall be used to control the lighting, HVAC and television subsystems as well as calculating the energy consumption. This engine was chosen due to its ease of data storage and Android connectivity.

5.6.1 - Database Layout
Four tables shall represent the electricity provider, appliance, home and TV control. The electricity provider shall contain the name of the company and the residential kilowatt per hour rate. The appliance table shall hold information
regarding the HVAC and lighting subsystems. This information stores the usage start times, end times and which appliance is being used. The home table holds the Google account information used to log into the application. Finally the TV control shall hold the TV brand and TV code.

5.6.2 - User interaction
The interaction between the users and the Google App Engine always begins with the login information. Once the user is logged into his or her Google account, the user then inputs the electricity provider and residential rates in kWh. From here the user may choose to access the lighting, HVAC or television subsystem. The user may also choose to calculate daily usage and monthly usage.

5.6.3 - Software Architecture
The software system architecture is a mix between client-server and layered architecture. Clients are the users of the Android devices and the server shall be Google’s server space. The devices shall interact with Google’s server by sending queries to the BigTable database. The layered architecture is a strict two-layered system that dictates the interaction between the server application and the Google BigTable database.

5.6.4 - Login Screen
The login screen shall prompt the user to input his or her Google account username and password. The Username text view shall be a simple string stating “Google Account Username.” Under that will be a textfield for the user to input the password. Finally, a login button shall be pressed once the user has entered his or her username and password.

5.6.5 - Menu Layout
After a successful login, the main menu layout shows a list to choose from including Input Electric Company Values, Calculate Usage, Control Subsystem and Credits. The menu is created as a “simple_list_item” table.

5.6.6 - Usage Calculation
This interface calculates the daily and monthly energy usage according to the selected date by the user. The title textview shall display a string that says “Electricity Provider Information” and the result shall display the resulting daily or monthly cost depending on what the user chooses to calculate. The user chooses whether to view the daily usage or monthly usage and uses the spinners on the interface to choose a specific day, month and year.

5.6.7 - Dimmer Control
The dimmer control shall display three different rooms for the user to control depicting a living room, kitchen and a bedroom. The dimmer controls four lighting fixtures in the living room: two lamps, a fan light and a ceiling light. The dimmer
controls three lighting fixtures in the bedroom: a desk light, a corner light and a fan light. The dimmer also controls two lighting fixtures in the kitchen: ceiling lights and dining lights. The interface displays each room individually with its corresponding light fixtures set in a list beneath it to choose between dimming controls. Once the user chooses one of these checkboxes, he or she shall use a seekbar to control the percentage of dimness of the chosen fixture.

5.6.8 - HVAC Control
The Android application shall allow the user to control the HVAC subsystem. The user is able to control four main parts of the HVAC system: heating, cooling, emergency heating and fan systems. The user can choose which system he or she wishes to control. If the user chooses the fan, he or she may turn it on or off. If the user chooses heating or cooling, he or she may then choose the desired temperature. The interface displays the desired temperature, the actual temperature, a drop down menu for the fan and a drop down menu for the systems. The user shall choose an input and then press the ACTIVATE button which shall send the requests to the HVAC subsystem.

5.6.9 - TV Control
The TV control interface allows the user to control a television from the Android application. The main controls that can be used are power (tv on/off), Channel Control (channel up or down), Volume Control (volume up or down) and Channel Selection. The user shall press the power button to turn on or turn off the television. The volume and channel controls display a button with a “+” sign to increase the volume or channel number and a “-” sign to decrease the volume of the channel number. The Channel selection section displays a text field in which the user is able to input up to three digits which indicate the desired channel. Once the channel number is entered the ENTER button shall then be pressed to jump to the desired channel.

5.7 Other Possibilities
There are numerous systems a household contains other than the three main subsystems this home automation system contains. The chosen subsystems, television control system, heating, ventilation and air conditioning control system and the lighting system are three subsystems that just about every house contains.

5.7.1 - Security
Security systems such as the system shown in figure() can be an important part of a household. Security systems usually contain multiple settings and parts. Most security systems are centered around an alarm system set in place to prevent breaking in and entering. In this case most, if not all, doors and windows are fitted with sensors to detect if an entry is being accessed or not. Security systems can range from somewhat complex systems to simple systems. Most
simple systems include an alarm system, a smoke detection system and security lights which are all centered around a base keypad. This keypad usually includes numbers 1 through 9 for an alarm code, as well as buttons for emergency calls to the fire department, alarm response teams and the police. Smoke alarms detect harmful rising smoke that may be caused by a fire in the home. Once this alarm sounds it automatically calls the fire station and warns people in the home of a possible fire hazard. Security lights are usually placed in front of houses to detect an approaching person and possible burglar. They are designed to sense movement which triggers the light to shine. More complex security systems may contain security cameras, indoor movement sensors and driveway alarms. Security cameras can either be monitored by a security guard for large businesses or the exceedingly paranoid or recorded onto obsolete VHS tapes. Indoor movement sensors can be placed in parts of the house that view the optimal amount of space to detect intruders while the alarm system is activated. Finally, driveway alarms sense incoming people with either movement or pressure sensors. Since security systems aren’t as popular as other systems, can be very complex including multiple subsystems and can contain expensive components the 2012: A Home Odyssey doesn’t contain an automated system for security.

![Figure() Example Security System](image)

5.7.2 - Home Phone
Home phone systems in the past have been a main system of most homes. Phone systems contain multiple phones throughout the house interconnected to the same phone number. Phone systems aren’t very customizable other than a few features such as speakerphone, voicemail and remote dialing. Due to the fact that home phones are becoming obsolete with the increase of cellular telephones and that this home automation system is actually controlled from a user’s cellular device, the group decided against a home phone subsystem.

5.7.3 - Appliances
Household appliances which are plugged into power outlets are vastly spread throughout most homes. The possibility of controlling any of these appliances is well in reach although there are such a great number of different appliances and models. For example, a coffee machine can be programmed to turn on or off using a sleep timer. A computer, washing machine or dryer can be turned on before arriving home or from a remote location if desired but many of these appliances contain high power and mechanical systems which the group wishes to avoid. These systems can be controlled via a power outlet by implementing a system to supply power or not to supply power. Homes could contain zero to a large amount of these systems but the group decided against this notion due to the high number of possibilities and complexities.
6 - Project Prototype Construction and Coding

6.1 - Parts Acquisition and Bill of Materials:
Obtaining the correct parts is a very important part of building our home automation system. In addition, a Bill of Materials will be necessary to ensure all parts for each system and subsystem are purchased and accounted for.

6.1.1 - Parts Acquisition
The purchasing of the majority of components necessary to build our home automation system will done in an electronic form. Since one of our primary objectives is to reduce the total cost of the system as much as possible, we shall acquire parts at the lowest cost possible. We will be utilizing the services of two of the biggest electronic component distributors in the United States: Newark (Element 14 division) and Digikey Incorporated. Each supplier has their own extensive stockroom of parts, most overlapping which each other. Newark has even offered an exclusive UCF student discount which helps reduce the cost even more. The parts will be ordered in the order noted in the priority list below:

- 2x Texas Instruments MSP430G2552 Microcontroller
- 1x Texas Instruments MSP430F2131 Microcontroller
- 1x Texas Instruments LM3S8962 Microcontroller
- 4x Texas Instruments CC1000 RF Transceiver
- 1x Android Smartphone
- 1x TMP100 Temperature Sensor
- 1x LED System Status Indicator Array
- 1x Samsung LCD HDTV
- 1x MMBT2222 Transistor
- 2x Capacitors (0.1μF and 200μF)
- 1x 3V CR2032 Lithium coin cell battery
- 3x Resistors (5Ω, 402Ω, and 47kΩ)
- 3x 1N4148 Diodes

The Texas Instruments parts shall be purchased through www.ti.com including the MMBT2222 transistor and three 1N4148 diodes. The Android Smartphone that shall be used is currently owned by a group member. The Samsung LCD HDTV is currently owned by a group member and shall be used for the testing phase.
### 6.1.2 - Bill of Materials

<table>
<thead>
<tr>
<th>Component</th>
<th>Model</th>
<th>Quantity</th>
<th>Price</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI Microcontroller</td>
<td>MSP430G2552</td>
<td>2</td>
<td>$0.60</td>
<td>$1.20</td>
</tr>
<tr>
<td>TI Microcontroller</td>
<td>MSP430F2131</td>
<td>1</td>
<td>$1.35</td>
<td>$1.35</td>
</tr>
<tr>
<td>TI Microcontroller</td>
<td>LM3S8962</td>
<td>1</td>
<td>$8.35</td>
<td>$8.35</td>
</tr>
<tr>
<td>TI RF Transceiver</td>
<td>CC1000</td>
<td>4</td>
<td>$5.70</td>
<td>$22.80</td>
</tr>
<tr>
<td>Android Smartphone</td>
<td>Motorola Droid</td>
<td>1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Digital Potentiometer</td>
<td>TPL0102-100</td>
<td>3</td>
<td>$1.60</td>
<td>$4.80</td>
</tr>
<tr>
<td>Temperature Sensor</td>
<td>TMP 100</td>
<td>1</td>
<td>$0.75</td>
<td>$0.75</td>
</tr>
<tr>
<td>LED Array</td>
<td>X-Bright White LED</td>
<td>1</td>
<td>$0.16</td>
<td>$2.56</td>
</tr>
<tr>
<td>Samsung LCD HDTV</td>
<td>LN26A450C1DXZA</td>
<td>1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Transistor</td>
<td>MMBT2222</td>
<td>1</td>
<td>$0.25</td>
<td>$0.25</td>
</tr>
<tr>
<td>Capacitor</td>
<td>0.1μF and 200μF</td>
<td>2</td>
<td>$0.25</td>
<td>$0.50</td>
</tr>
<tr>
<td>Lithium Coin Cell Battery</td>
<td>3V CR2032</td>
<td>1</td>
<td>$1.25</td>
<td>$1.25</td>
</tr>
<tr>
<td>Resistor</td>
<td>5Ω 40Ω 47kΩ</td>
<td>3</td>
<td>$0.25</td>
<td>$0.75</td>
</tr>
<tr>
<td>Diode</td>
<td>1N4148</td>
<td>3</td>
<td>$0.15</td>
<td>$0.45</td>
</tr>
</tbody>
</table>

Table 6-1

---

### 6.2 - Proof of Concept:

Our project plans to follow an agile programming development model. By specifically using the SCRUM model the group will have sprints and meetings on a regular basis to discuss progress and what concerns and problems we may have with the current sprint. First we must start with building basic building blocks for our project before we can assemble the big picture.
Below we have broken down the components into a proof of concept phase. This will be the time that the group will test the actual parts on evaluation boards before we purchase the printed circuit boards. We have spent much time researching compatible parts that would be easiest and least expensive for us to prototype with.

6.2.1 - RF Communication link
One big hurdle we must pass first is configuring a radio frequency communication link between two MSP430 microcontrollers. To do so we have purchased two MSP430 launchpads as well as a CC110L Air Module BoosterPack. The Air Module is a plug in and play component that we can simply place on top of the launchpad’s exposed pins. We will send simple commands to LED lights to prove that we have an ample channel to transmit data over. If tests go well we will use this antenna for our project.

6.2.2 - Lightweight Internet Protocol (lwIP)
Our main controller will be one of the middle links in between the Android device and the MSP430 subsystems. This board will have a functional internet protocol stack via the lightweight internet protocol provided to us by Texas Instruments. First we must be able to host some sort of hypertext markup language page on the Stellaris that can be accessed via an internet protocol address. Our Android
device will be used to see if we can see this page or not. To do so we have acquired a Stellaris EKS-LM3S8962 Evaluation Kit to run simulations with. It has more than the required components to build this, even an organic light emitting diode screen built into the board. If the Android device can ping the Stellaris, we have a valid wifi connection for communication.

6.2.3 - Dynamic Domain Name Service (DDNS)
The Stellaris will need to communicate with the Google App Engine but we need to place the Stellaris on the internet somehow. Dynamic domain name service will provide us with a semi-static landing space for our application to run on. The beauty of dynamic domain name service is that it will update your dynamic internet protocol address periodically so that you do not need your own static internet protocol address. If we can view our simple hypertext markup language page from our test in section 6.2.2 on the internet now, we have confirmed a valid landing spot for our application to communicate through.

6.2.4 - HTTP Communication
The Google App Engine provides us a great service at no cost to us. It will give us a database to store the necessary data via their big table database. We will want to make sure that the android can not only locate the Google App Engine’s pages but also save data to the database and keep preset configurations. First we will need create a hypertext markup language page for the Android device to ping so validate communication. Next we will need to configure a servlet on the Google App Engine to test functionality. Android has Apache installed on the device which we will use to send hypertext transport protocol requests to the Google App Engine that encode for some type of advanced programming interface communication.

6.2.5 - Database Management
As stated in the prior section we will need to manage a database for the users settings. This will be an extension to the prior component because we will still be sending a hypertext transport protocol request to the Google App Engine but it will be doing something when sent. We will want to send data to a Big Table and test all the required functionalities. We will need to create some sort of advanced programming interface to command Insert, Update, Select, and Delete structured query statements so we can manage the records in the database. We will also need to manage how we layout our landing space because if not our server will look like a mess and performance may be hurt by this.

We have proposed using a model view controller (MVC) style layout, however we will have very few actual rendered views. Most of the views will be server side landing spaces for our application to transmit requests to. Our models will be where we place all of our structured query language statements in their appropriate classes and methods. The controller will be our actual java server page where the user will send the hypertext transport protocol requests to.
6.2.6 - Dynamic Android Layout Content
Since our layout will not be hardcoded into the application itself, we will need to create some sort of data representation to build the android application’s layout content dynamically. Our group has worked with web applications before and feel that this will be the best route to go because it will enable easy and limitless expandability to our application. To test this functionality we will need to send a HTTP_GET request to the Google App Engine and retrieve some data that is used to build the content of the layout on the application.

Our tests will need to be lengthy since we have multiple screens that will all be using a similar advanced programming interface. We hope to knock out some of our graphical user interface during this time as well because it is directly associated with the layout of the application. Below is an example of how we could test the lighting subsystem with dynamic content. We will just need to load which rooms you have the lighting subsystems located in so that you can direct your advanced programming interface calls to that unit.

![Figure 6-2: Example of dynamic layout for lighting subsystem](image)

6.2.7 - Trigger a Command!
Our final component we will need to complete for our proof of concept is a link from the Android device to the Google App Engine and then relayed to the Stellaris. Our command will be to simply make the buzzer go off on the Stellaris evaluation board.
After we have completed this relay, our proof of concept is complete. We have configured a RF communication link for in the home and a connection between the system and the stellaris over the internet.

6.2.8 - Bag of Words
Our wow factor we wish to incorporate into our project is a speech to command feature. Android has long had speech to text software which works pretty reliably. The API library we will be using is android.speech which is a built-in speech to text processor. We will need to tune its functionality to work properly. There are parameters such as confidence scores and the actual string itself. We feel that since we do not have proper NLP (Natural Language Processing) skills to help aid the confidence we will need to parse the string. We have chosen to use a bag of words style format that will listen for key words. You can add all of the filler and fluff into your command as you would like because the we will only be looking for certain words to complete commands.
If we can get the Android device to filter through the commands said by the user by the use of keywords we believe it will be easier to use. We realize that it would be almost impossible to remember the exact format of every command so if we have a similar style across the board it will be easy to use. We think this is a great addition to our system because it will add that extra wow factor.
6.3 - Final Coding Agreement:

Our final coding agreement has been worked out and seems to be pretty evenly distributed based on strengths and amount of work. Most of our coding will be done by the computer engineers since they will not be working as much with building the physical hardware. The electrical engineers will be coding in either C or assembly.

Below we have broken down the assignments into sections with project leaders. Since we do not expect one person to do a complete subsystem on their own they will simply be the leader instead of given sole responsibility of it.

6.3.1 - Jimmy’s assignments
Jimmy will be in charge of the android application. He will build the design and look of the application. He will become familiar with the rest client used to send hypertext transport requests to the Google App Engine. He will also help with the database management of the Google App Engine. Jimmy will also be working with John to create the client side advanced programming interface structure for our project.

6.3.2 - Heston’s assignments
Heston is in charge of the television subsystem and has been focusing on that unit mostly. We believe that this subsystem will be one of the harder subsystems to complete because we have to get infrared codes sent to a television that are stored in the Google App Engine. He will also help Kyle with the heating ventilation and air conditioning subsystem.

6.3.3 - Kyle’s assignments
Kyle is in charge of the lighting as well as the heating ventilation and air conditioning subsystems. He will be designing and programming the MSP430’s to do the desired tasks. He will also be helping with the radio frequency communication modules between the Stellaris and MSP430’s.

6.3.4 - John’s assignments
John is in charge of the Google App Engine as well as the Stellaris main controller. He will be doing mostly backend work and not end user components. He will be mostly in charge of putting the pieces together rather than a physical subsystem. He will also be in charge of programming the RF communication link with Kyle.

6.3.5 - Other thoughts
Since all of us have been trained to be able to design this project we will be active in every step. We feel that it is vital for our group to work together so that we can all get the most out of this experience.

7 - Project Prototype Testing

7.1 - Lighting Subsystem
The main objective in testing the lighting subsystem is to see how the system responds to commands given to it in the form of software loops running solely on the subsystem’s mcu and then finally from the main controller.

7.1.1 - Description of the test Environment
For the initial testing of the lighting subsystem it will operate as an independent entity in order to ensure the basic functionality works correctly. The initial testing shall be done in a closed laboratory setting with the ambient temperature being no less than 32 degrees fahrenheit and no greater than 110 degrees fahrenheit.

7.1.2 - Test Procedure
The microcontroller unit shall be pre-loaded with software that will go through a controlled loop to test all functionality of the system. To put the microcontroller unit into loop mode, a jumper will be assigned. The loop will execute as follows:

- Turn all banks on for 5 seconds
- Turn all banks off for 5 seconds
- Turn bank 1 full on for 2 seconds
- Turn bank 1 full off for 2 seconds
- Dim bank 1 from full off, to full on, back to full off within 5 seconds
- Turn bank 2 full on for 2 seconds
- Turn bank 2 full off for 2 seconds
- Dim bank 2 from full off, to full on, back to full off within 5 seconds
- Turn bank 3 full on for 2 seconds
- Turn bank 3 full off for 2 seconds
- Dim bank 3 from full off, to full on, back to full off within 5 seconds

If any of the above operations do not occur, take longer time than is specified, or occur out of sequence, the test has failed. If the above operations do occur, take the specified amount of time, and occur in sequence, the test has passed. There are no exceptions.

Once the above test has been performed and is confirmed working, the microcontroller unit with be pulled out of loop mode and into operational mode. To test the subsystem in operational mode, the frontend software interface shall be used. The tester will check to ensure each bank of leds are working correctly by following the specified commands.
- Enter into lighting portal on user interface
- Press the “All Rooms On” button, all rooms should light at full on.
- Press the “All Rooms Off” button, all rooms should be turned off.
- Slide the “Room 1” slider up and down several times. The leds should increase in brightness when sliding up and decrease in brightness when sliding down in a linear and smooth fashion.
- Slide the “Room 2” slider up and down several times. The leds should increase in brightness when sliding up and decrease in brightness when sliding down in a linear and smooth fashion.
- Slide the “Room 3” slider up and down several times. The leds should increase in brightness when sliding up and decrease in brightness when sliding down in a linear and smooth fashion.

If any single part of this test fails, the whole test fails. If all parts of the test pass, the whole system passes.

7.2 - HVAC Subsystem
The main objective when testing the lighting subsystem is to ensure all operating parameters and variables work according to the specifications. The system should switch into the appropriate mode based upon the user’s desired temperature.

7.2.1 - Description of the test Environment
For the initial testing of the HVAC subsystem it will operate as an independent entity in order to ensure the basic functionality works correctly. The initial testing shall be done in a closed laboratory setting with the ambient temperature being no less than 32 degrees fahrenheit and no greater than 115 degrees fahrenheit.

7.2.2 - Test Procedure
The microcontroller unit shall be pre-loaded with software that will go through a controlled loop to test all functionality of the system. To put the microcontroller unit into loop mode, a jumper will be assigned. The loop will execute as follows:

- Automatically set desired temperature to 70 degrees fahrenheit.
- If the ambient temperature is greater than 70 degrees fahrenheit, the blue (cooling mode activated) and green (fan on) LED’s should light.
- If the ambient temperature is less than 70 degrees fahrenheit, the red (heating mode activated) and green (fan on) LED’s should light.
- If the ambient temperature is greater than 10 degrees less than ambient, the orange (Emergency heat mode activate) and green (fan on) LED’s should be activated.
- After 10 seconds the system will automatically reset and run through the loop until the jumper is removed.

If the system behaves exactly as above, the test **passes**. If it behaves any different or takes longer than specified, the test procedure has failed.

Once the above test procedure has been completed, the system shall be put into operational mode via the configuration jumper. This will allow the hvac subsystem to be utilized by the main control system. The tester will be able to run the following commands from the user interface to ensure the subsystem is working correctly:

- Enter into the HVAC portal on the user interface.
- Set desired temperature to ambient.
- Using a small heat source of no more than 115 degrees fahrenheit, place the heat source no closer than .25 inch away from temperature sensor. The cooling mode should activate along with the appropriate LEDs.
- To test the heating mode, set the desired temperature for no more than 10 degrees less than ambient. The heating mode should activate along with the appropriate LEDs.
- To test the emergency heating mode, set the desired temperature for more than 10 degrees less than ambient. The emergency heating mode should activate along with the appropriate LEDs.

The system should behave exactly as described above to **pass** the test procedure. If any component of the above test procedure fails, the whole system **fails** the entire test procedure.

### 7.3 - Television Subsystem

The main objective of the television subsystem is to implement an IR remote control. The desired feedback will rely on the user input and response of the Samsung HDTV.

#### 7.3.1 - Description of the test environment

For the initial test setting, the television subsystem will operate as a single entity including only the remote itself and the television to ensure basic functionality between the transmitter and the decoder of the television. The initial testing shall be done in a closed laboratory setting with the ambient temperature being no less than 32 degrees fahrenheit and no greater than 110 degrees fahrenheit.

#### 7.3.2 - Test Procedure
The microcontroller unit shall be pre-loaded with software that will go through a controlled loop to test all functionality of the system. To put the microcontroller unit into loop mode, a jumper will be assigned. The loop will execute as follows:

- Turn on the television.
- Set volume to 0%.
- Increase volume to 50%.
- Loop through 10 random channel numbers.
- Increase channel by 5 channels.
- Decrease channel by 10 channels.
- Turn off the television.

If the test behaves exactly as above, the test passes. If the test behaves any differently or takes longer than anticipated, the test has failed.

Once the above test procedure has been completed, the system shall be put into operational mode via the configuration jumper. This will allow the television subsystem to be utilized by the main control system. The tester will be able to run the following commands from the user interface to ensure the subsystem is working correctly:

- Enter into the Television remote control portal on the user interface.
- Set the television on and prompt the user with a set of possible actions.
- The user shall then test the functionality of the volume by increasing and decreasing at random.
- The user shall then test the functionality of the channel selection by selecting channel numbers at random.
- The user shall then test the functionality of the channel up and down function by increasing or decreasing the channel value at random.
- The final test by the user shall be setting the television to off mode.

The system should behave exactly as described above to pass the test procedure. If any component of the above test procedure fails, the whole system fails the entire test procedure.

### 7.4 - Google App Engine

The main objective of the testing of the Google App Engine is to see if all of the front-end software components are fully functional. The goals are to have all of the functions of the software to be in working order and to be completely free of aesthetic bugs.

#### 7.4.1 - Description of the Test Environment
For the actual running of the software, there are two options: an Android device simulator, created through the Android plugin for the Eclipse IDE, and an actual Android mobile device. Android mobile devices will be used for testing due to the popularity of the usage of Android devices and the fact that the final product’s environment will be Android mobile devices. The testers will be composed of the developers and actual users. The developers will concentrate on testing the major components of the software to ensure non-buggy functionality. The actual users will also test the functionality and give feedback for the aesthetics of the design. The developers will perform testing before actual users are able to test the software.

For the developers, one interface will be tested at a time. The ordering of the interface testing is as follows: inputting values, calculating usage, controlling the dimmer system, controlling the HVAC system, and controlling the television. For the actual users, they will perform the same tasks as the developers. In addition to the testing of the interfaces, users will give feedback on aesthetics of various parts of the software. Parts of the software include the interface separation via tabs, the scrollbar unit for the dimmer, icons, and the login screen. Chosen users will have a variety of Android mobile devices from various popular companies (htc, Motorola, Samsung, etc.). The specifics of the test cases are outlined in section 7.3.3.

7.4.2 - Stopping Criteria
To ensure quality testing, stopping points need to be stated for cases of stopping the testing phase to correct major errors and an acceptance of a prototype. Stopping the testing phase prematurely will only apply to developers.

The developers will continue to test until there are no critical errors associated with the functionality of the software. During the testing phase, if a critical error were to occur, such as not being able to save the electricity provider’s information, then the testing of the software will temporarily cease to run. The error will then be attempted to be fixed, on the spot, by having a developer go through the code associated with the function that is currently being tested. If the debugging results in the error not being to be easily fixed, then the testing phase for developers will come to a stop. At that point, the developers will go through the entire source code to pinpoint the cause(s) of the error. If the error were to be fixed on the spot, then testing of the function will continue. This testing strategy has been chosen to ensure proper working functionality of the software before exposing the software to actual users. Also, the resulting strategy will lead to minor errors being fixed without the time wasted going through the entire source code.

When the developer testing phase is complete, actual users, with Android mobile devices, will be given the task to test the functionalities of the software and give feedback for the aesthetics. Further testing of the functionalities by the users will further ensure that the software contains no noticeable bugs. Also, the common
tasks that the users perform while operating the software will be noted. Examples of common tasks would be specific values entered into the textfields pertaining to the provider information, the chosen actions to be performed on the television, and the chosen lights to be operated on. The noted common tasks performed by users will be applied by having the developers concentrate on what functionalities to keep. The users will be asked to give feedback on the layout of the functionalities. Any constructive criticism will be noted and if a consensus on a certain part of the layout has been reached, then the suggestion will be applied to the final product.

When the developer testing phase and the actual user testing phase have been completed, the design criteria of the final product will be achieved. By then, the product will have no noticeable software bugs and the product will be aesthetically pleasing to the common user.

7.4.3 - Description of Individual Test Cases
To ensure an orderly testing process, specific test cases have been created. The following list shows the individual test cases that will be performed by the developers and actual users:

7.4.3.1 - Login testing
- Textfields for the Google App Engine URL, username, and password will be inputted to see if logging into the respective account is possible. The expected outcome will consist of a welcome message. A valid gmail account from one of the developers will be used to test the application. The objective of this test is to see if logging into a valid account is possible.
- Textfields for the Google App Engine URL, username, and password will be inputted with invalid inputs. The expected outcome will consist of an error message. The objective of this test is to see if the invalid input exception runs successfully.

7.4.3.2 - Input Values Testing
- Textfields for the electricity provider’s company name and residential rate will be inputted with sample values in correct form. After the information has been entered in, the button for entering the information will be pressed. The expected outcome should have the textfield at the top of the screen displaying the newly added values. The objective of this test is to see if information on the electricity provider can be saved.
- Textfields for the electricity provider’s residential rate will be inputted with an invalid value (Non-monetary value). The expected outcome should have a message prompting the user to input a correctly formatted monetary value. The objective of this test is to see if the software can catch invalid inputs for this section.

7.4.3.3 - Dimmer testing
- A single light from the living room will be chosen. The scrollbar controlling the dimming of the lights will be controlled to the point where varying levels of brightness are chosen. The expected outcome should have the targeted LED, representing the chosen light from the living room, dim and light up according to the degree of brightness. The objective of this test is to see if a single LED can be manipulated via the application.

- Two lights from the living room will be chosen. As before, the scrollbar controlling the dimming of the lights will be moved to varying levels of brightness. The expected outcome should have the two targeted LEDs, representing the chosen lights from the living room, dim and light up according to the degree of brightness. The objective of this test is to see if multiple LEDs in a single room can be manipulated at the same time via the application.

- Two lights from the living room and two lights from a bedroom will be chosen. The scrollbar controlling the dimming of the lights will be moved to varying levels of brightness. The expected outcome should have the targeted LEDs, representing the two lights from the living room and two lights from the chosen bedroom, display varying levels of brightness according to the set brightness. The objective of this test is to see if multiple lights from different rooms can be manipulated via the application.

- Variations of the previous test case will be conducted. This will be done until every single room has been tested. The objective of this test is to see if multiple LEDs from different rooms, for all combinations of rooms, can be controlled via the application.

- Every light from every room will be chosen. As before, the scrollbar will be controlled to varying brightness levels. The expected outcome should have every LED, pertaining to the dimming system, display varying levels of brightness according to the set brightness. The objectives of this test are to make sure that every light is functional and to see if the hardware can handle the lighting of every LED.

- No lights will be chosen. The scrollbar will be controlled to varying brightness levels. The expected outcome should have no LEDs lighting up at all for any of the selected brightness levels. The objective of this test is to see if the default case, having no lights selected, is fully operational.

7.4.3.4 - HVAC testing
- The textview for the current temperature will be observed as the temperature is being changed for the hardware testing. The expected outcome should consist of the textview being constantly updated as the current temperature is being changed. The objective of this test is to see if the on demand update function for the current temperature is fully operational.

- The textfield for the desired temperature will be inputted with a value that is 5 degrees lower than the current temperature. The button, for updating the desired temperature, will be pressed. The expected outcome should have the LED, denoting the cooling mode, light up. The LED should turn
off when the desired temperature is met. The objective of this test is to see if the cooling mode, without the fan being turned on, for the HVAC system is fully operational.

- The textfield for the desired temperature will be inputted with a value that is 10 degrees below the current temperature. The fan will be set to “ON.” The expected outcome should have the LED, denoting the cooling mode, turn on and the LED, denoting the fan, turn on at the start. When the desired temperature is met, the LED for the cooling mode should turn off. The LED for the fan operation should still be on when the desired temperature is met. The objectives of this test are to reinforce the functionality of the cooling mode with another test case and to see if the continuous fan operation is fully operational.

- The textfield for the desired temperature will be inputted with a value that is 5 degrees below the current temperature. The fan will be set to “AUTO.” The expected outcome should have the LED, denoting the cooling mode, and the LED, denoting the fan operation, turn on at the start. When the desired temperature is met, both the cooling mode LED and the fan operation LED will turn off. The objectives of this test is to further reinforce the functionality of the cooling mode with another test case and to see if the automatic fan operation is fully operational.

- The textfield for the desired temperature will be inputted with a value that is 5 degrees above the current temperature. The fan will be set to “OFF.” The expected outcome should have the LED, denoting the heating mode, turn on at the start. When the desired temperature has been met, the LED should turn off. The objectives of this test are to see if the heating mode of the HVAC system is fully operational.

- The textfield for the desired temperature will be inputted with a value that is 10 degrees above the current temperature. The fan will be set to “ON.” The expected outcome should have LED, denoting the heating mode, and the LED, denoting the fan operation, turn on at the start. When the desired temperature has been met, the heating mode LED should turn off and the fan operation LED should still be on. The objectives of this test are to see if the heating mode of the HVAC system and the continuous fan operation is fully operational.

- The textfield for the desired temperature will be inputted with a value that is 5 degrees above the current temperature. The fan will be set to “AUTO.” The expected outcome should have the LED, denoting the heating mode, and the LED, denoting the fan operation, turn on at the start. When the desired temperature has been met, both LEDs should turn off. The objectives of this test are to see if the heating mode and the automatic fan operation are fully operational.

- The textfield for the desired temperature will be inputted with a value that is 10 degrees above the current temperature. The fan will be set to “AUTO.” The emergency heat will be set to “ON.” The expected outcome should have the LED, denoting the emergency heat mode, and the LED, denoting the fan operation, turn on at the start. When the desired
temperature has been met, both LEDs should turn off. The objective of this test case is to test the functionality of the emergency heat system.

7.4.3.5 - TV control testing

- The “POWER” button will be pressed once. The expected outcome should have the television turn on. If the television turns on, then the power button will be pressed again. The television should turn off. The objective of this test is to see if the application is able to communicate through the Stellaris to turn television on and off.

- The “CHANNEL UP” button will be pressed once. The expected outcome should have the television go to the next channel. If this occurs, then the “CHANNEL UP” button will be pressed 10 more times. After every button press, the channel should go to the previous channel, in respect to the current channel. The objective of this test is to validate the functionality of the channel up feature of the television control interface.

- The “CHANNEL DOWN” button will be pressed once. The expected outcome should have the television go to the previous channel. If this occurs, then the “CHANNEL DOWN” button will be pressed 10 more times. After every button press, the channel should go to the previous channel, in respect to the current channel. The objective of this test is to validate the functionality of the channel down feature of the television control interface.

- The “VOLUME UP” button will be pressed once. The expected outcome should have the television increase its volume by one interval. If this occurs, then the “VOLUME UP” button will be pressed 10 more times. After every button press, the volume should increment by one interval in respect to the current volume. If the volume changes to the correct amount, then the button will be pressed repeatedly until the max volume has been reached. The objective of this test is to validate the functionality of the volume increase feature of the television control.

- The “VOLUME DOWN” button will be pressed once. The expected outcome should have the television decrease its volume by one interval. If this occurs, then the “VOLUME DOWN” button will be pressed 10 more times. After every button press, the volume should decrement by one interval in respect to the current volume. If the volume changes to the correct amount, then the button will be pressed repeatedly until the television has been muted. The objective of this test is to validate the functionality of the volume decrease feature of the television control.

- The following 2-digit channels will be inputted via the channel number entry interface: 08, 15, 23, 37, 49, 50, 64, 75, 86, and 98. The expected outcome should have the television switch to the inputted channels. The objective of this test is to see if the channel number entry interface is fully operational.

7.4.3.6 - Calculations testing
The daily usage option will be chosen. After running the previous tests throughout the day, the current date will be entered into the daily electricity consumption option. The expected outcome should have the calculated electricity usage cost for the current date. In order to test the validity of the calculated amount, the amount will be manually calculated by the tester. In order to have the appropriate values, the Datastore Viewer of the web application will be accessed. Once the Datastore Viewer has been accessed, a GQL query will be ran to receive the residential wattage of the electricity provider. After doing so, GQL queries will be ran to receive the appropriate usage times of each appliance on the given date. The following pseudocode illustrates the logic behind the GQL queries to be inserted:

```
SELECT *
FROM <Appliance or Television Control>
WHERE Appliance_Name = <insert appliance name>
    AND Usage_Date = <insert date here>
ORDER BY <Ascending order of usage start time>
```

The GQL queries will output usages of the selected appliance. For each BigTable database entry, the total usage time will be yielded by calculating the amount of time between the starting time of usage and the ending time of usage. The calculated time of usage, the wattage of the appliance, and the residential rate will be inserted into the following equation to calculate the daily cost (Wattage is in watts):

\[
\text{Daily cost} = (\text{Rate\_per\_kWh}) \times \left[ \left( \text{Wattage} \times \left( \frac{\text{Minutes\_used}}{60} \right) \right) / 1000 \right]
\]

This will be done for all entries. The daily cost of every entry will be added at the end of the individual calculations. If the manually calculated daily cost matches the daily cost yielded by the application, then the daily cost calculation is operational.

The monthly usage option will be chosen. This will be done after doing the previous tests in order to have entries in the database ready to be used. The procedure to validate the output will be the same as the previous test, but the GQL queries will yield entries for the entire current month instead of a particular day.

To simulate the calculation of monthly usage, the Datastore Viewer will be used to insert entries of usages of various appliances on various days of the current month of testing. The expected output should have a monetary cost denoting the monthly cost of electricity usage according to the electricity supplier’s residential rate. The following pseudocode outlines the logic of the query for usages for the entire month:
SELECT *
FROM <Appliance or Television Control>
WHERE Appliance_Name = <insert appliance name>
    AND Usage_Date >= DATETIME(<first day of the month>)
    AND Usage_Date <= DATETIME(<last day of the month>)
ORDER BY <Ascending order of usage start time>

As before, the calculations, with the same formula, will be done for all entries of the resulting query. The resulting daily costs of the various dates will be added together at the end to yield the monthly cost. If the manually calculated monthly cost matches the monthly cost yielded by the application, then the monthly cost calculation is operational.

7.4.3.7 - Appliance Inputs and Current Settings Testing
In order to properly test the Appliance Inputs interface, the current settings interface must be used in conjunction with the Appliance Inputs interface due to the fact that the current settings interface displays the inputs gathered from the Appliance Inputs interface. Along with the current settings interface, the BigTable database dashboard, accessed through a valid gmail account hosting the application, will be utilized to view the insertion of individual input of an appliance. If an input has been received correctly, then the entry should be available on the BigTable database dashboard.

The following list contains the test cases to test the appliance inputs interface:

- To test the television code input, a numerical television code, chosen from the programmable television code datasheet, shall be inputted into the textfield. After pressing the ENTER button, the entry should be present in the BigTable database dashboard. Error checking cases shall consist of inputs that are of a non-numerical type (Pure character and alphanumeric). If an error message appears to prompt the user to input a valid value, then the function is operational.

- To test the wattages of the dimmer and HVAC subsystems, each appliance from both subsystems will have their wattages inputted on at a time. As before, with each input, the BigTable database dashboard shall be checked with every input. If the entry appears in the dashboard, then the input is fully operational.

- To test the input for the wattage of the television, a wattage shall simply be inputted into the textfield. As before, the ENTER button shall be pressed and the BigTable database dashboard shall be reviewed for confirmation of insertion of the data.

- After every wattage and the television code have been inputted the “Current Settings” button shall be pressed. If the said interface has been successfully transitioned to, then the layout call within the .java file is fully operational.
• To test the output of data into the respective textviews, individual lighting fixtures and HVAC components shall be chosen. If the output, in the respective textviews, match the values in the respective entries in the BigTable database dashboard, then the current settings interface is fully operational. The correct values will also validate the functionality of the GQL retrieval queries, instantiated in the respective .java file(s).

7.4.3.8 - Credits Page Testing
The purpose of the credits page is to credit the developers of the 2012: A Home Odyssey project. To test the output of the credit page, the Credits option shall simply be selected. If the interface successfully transitions to the credits page, then the option is fully operational. This test will also further validate the functionality of the main menu. This shall validate the ordering given in the AndroidManifest.xml file.

7.5 Main Controller
The testing procedure for the Stellaris centered main controller base station is to confirm that the main controller is communicating and dictating the other devices correctly. The main controller receives data from the router via the Google App Engine via the user with an Android smartphone.

7.5.1 - Description of the Test Environment
The testing of the Stellaris based main controller will operate between each system individually through direct information. Each test shall be performed between the main controller and each individual subsystem using the Google App Engine and Android user interface.

7.5.2 - Lighting Test Procedure
The first test shall only be between the main controller and the lighting subsystem. Only once the lighting subsystem passed its individual test procedure this test can take place. The initial testing shall be done in a closed laboratory setting with the ambient temperature being no less than 32 degrees fahrenheit and no greater than 110 degrees fahrenheit. To test the main controller with the lighting subsystem we shall use API calls from the Google App Engine or Android device in line with the lighting subsystem test as follows:

- Turn on Living Room Lamp 1
- Turn on Living Room Lamp 2
- Turn on Living Room Fan Lights
- Turn on Living Room Ceiling Lights
- Test dimness levels 25% 50% and 75% on each appliance
- Turn off all Living Room appliances
- Turn on Bedroom Desk Lamp
- Turn on Bedroom Fan Lights
- Turn on Bedroom Corner Lamp
- Test dimness levels 25% 50% and 75% on each appliance
- Turn off all Bedroom appliances
- Turn on Kitchen Ceiling Lights
- Turn on Kitchen Dining Lights
- Test dimness levels 25% 50% and 75% on each appliance
- Turn off all Kitchen appliances

If the test behaves exactly as above, the test passes. If the test behaves any differently or takes longer than anticipated, the test has failed.

7.5.3 - HVAC Test Procedure
The second test shall only be between the main controller and the HVAC subsystem. Only once the HVAC subsystem passed its individual test procedure this test can take place. The initial testing shall be done in a closed laboratory setting with the ambient temperature being no less than 32 degrees fahrenheit and no greater than 115 degrees fahrenheit. To test the main controller with the HVAC subsystem we shall use API calls to and from the Google App Engine via a user operated Android smartphone device in line with the HVAC subsystem test as follows:

- Set Fan to AUTO
- Set System to Cooling System
- Set Desired temperature lower than the current Actual temperature
- Wait for a few moments while the Actual temperature adjusts to the Desired temperature
- Once the Actual temperature is the same as the Desired temperature, set the system to Heating System
- Set the Desired temperature higher than the current Actual temperature
- Wait for a few moments while the Actual temperature adjusts to the Desired temperature
- Set Fan to OFF

If the test behaves exactly as above, the test passes. If the test behaves any differently or takes longer than anticipated, the test has failed.

7.5.4 - Television Test Procedure
The third test shall only be between the main controller and the television subsystem. Only once the television subsystem passed its individual test procedure this test can take place. The initial testing shall be done in a closed laboratory setting with the ambient temperature being no less than 32 degrees fahrenheit and no greater than 115 degrees fahrenheit. To test the main controller with the television subsystem we shall use API calls to and from the Google App Engine via a user operated Android smartphone device in line with the television subsystem test as follows:
-Press the TV Control Power button to switch the television into on mode
-Press the “-” Volume Control button until the volume reaches 0%
-Press the “+” Volume Control button until the volume reaches 50%
-Using the Channel Selection textfield, choose five plausible channels at random pressing enter at each iteration
-Press the “+” Channel Control button five times to increase the the channel value
-Press the “-” Channel Control button five times to decrease the the channel value
-Press the TV Control Power button to switch the television into off mode

If the test behaves exactly as above, the test passes. If the test behaves any differently or takes longer than anticipated, the test has failed.
8.0 Administrative Content

8.1 - Timeline and Agile Model with Sprints
Project management is a key aspect of any successful project. We have decided to go with an agile method because we are most familiar with it. We believe that if we break things down into small components it will be more manageable to complete. Our timeline has also taken into consideration the amount of time it will take to get the printed circuit boards in the mail. We plan to design the hardware first, code the software and run simulations, and then finally put it on our printed circuit boards later.

Time goals are based on the SCRUM project management model with one week sprints and weekly meetings where we will discuss what progress we have made for that week. We will also extend any sprint if necessary because of some sort of technical issue. We feel that good communication is absolutely vital to our project's success. We feel that by having weekly meetings issues will be brought to our attention quicker than if we let them have set meetings on a consistent basis. Below are the sketched and proposed timelines for our project.

8.1.1 - Sprint 1
During sprint one we plan to build and design the lighting subsystem. Figure 1 shows our three step process we hope to do in this weekly sprint. We believe that this is the easiest subsystem to design since its only function is to turn on one light switch. We have considered using a triad for high power operation as well as a low powered light emitting diode module. This will also be the time when we start designing and laying out the radio frequency module. The rest of the MSP430’s will use the same radio frequency module for their communication link. After one week we will hold a meeting to discuss our progress and problems. This sprint might take more than one week because it will be our first time doing something like this.

As seen below, we will start by researching the requirements for the lighting subsystem. The first step will require the most amount of time since it will be the most detailed part of sprint 1. Next we will select parts that are compatible with the design. Finally we will layout everything on a printed circuit board for this sprint.
8.1.2 - Sprint 2
During sprint two we plan to design our heating ventilation and air conditioning subsystem. Figure 2 shows our three step process we hope to do in this weekly sprint. We believe that this is the next baby step we can take in our agile development method because it just incorporates digital logic into the equation. We will need to control nine pins in our heating ventilation and air conditioning subsystem to replicate the module in our apartment. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by researching the requirements for our heating ventilation and air conditioning subsystem. The first step will require the most amount of time since it will be the most detailed part of sprint 2. Next we will select parts that are compatible with the design. Finally we will layout everything on a printed circuit board for this sprint.

8.1.3 - Sprint 3
During sprint three we plan to create our television subsystem. Figure 3 shows our three step process we hope to do in this weekly sprint. This will be our most complex MSP430 application because it needs to send specific codes via an infrared bulb. We will have to select the television we plan to use during our
demo and get the codes for that model. We are also considering having this being the only battery powered module. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by researching the requirements for our television subsystem. The first step will require the most amount of time since it will be the most detailed part of sprint 3. Next we will select parts that are compatible with the design. Finally we will layout everything on a printed circuit board for this sprint.

### Sprint 3

![Sprint 3 Timeline](image)

#### 8.1.4 - Sprint 4

During sprint four we plan to design our main controller. Figure 4 shows our three step process we hope to do in this weekly sprint. This is the most heavy duty board we have to build. It will need to have ethernet, universal serial bus, and radio frequency communication all hosted and controlled by the Stellaris. We feel that this might take more than one week because of the amount of research and work this subsystem will require us to do. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by researching the requirements for our main controller subsystem. The first step will require the most amount of time since it will be the most detailed part of sprint 4. Next we will select parts that are compatible with the design. Finally we will layout everything on a printed circuit board for this sprint.
8.1.5 - Sprint 5
During sprint five we plan to prototype our RF communication module. Figure 5 shows our three step process we hope to do in this weekly sprint. We will need to select the correct parts and actually buy a kit to run some tests with. We will need to make sure that this is compatible with our original design. We have already found a launchpad booster pack to run tests with and actually run some test programs on it to see if we can get this technology to work and how well it performs. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by researching the requirements for radio frequency communication. Next we will select parts that are compatible with the design and buy some parts to test the products with. The second step will require the most amount of time since the bulk of the work in sprint 5 is testing the product and coding tests to run. Finally we will make any changes to our past design if we find that this part is insufficient.

8.1.6 - Sprint 6
During sprint six we plan to program our test Stellaris with lightweight internet protocol as well as implement dynamic domain name service. Figure 6 shows
our three step process we hope to do in this weekly sprint. We will first need to get lightweight internet protocol up and running which might take a week alone. The next requirement will be to build a landing space on the Stellaris that can be seen from a computer via the dynamic domain name service. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by researching the requirements for dynamic domain name service. Next we will implement the lightweight internet protocol stack and code in a custom domain name service in the stack. The second step will require the most amount of time since it will actually be implementing two separate programming assignments that are coupled together. Finally we will make any changes to our past design if we find that the Stellaris is insufficient for this application or if dynamic domain name service is not what we want.

![Sprint 6 Timeline](image)

### 8.1.7 - Sprint 7

During sprint seven we plan to get the Google App Engine up and running. Figure 7 shows our three step process we hope to do in this weekly sprint. We will need to learn how the Google App Engine functions in the Java environment. We will also evaluate the python environment to see if it is a better match however we do have more experience coding in Java. We will also need to become more familiar with their Big Table database system because it is not a full relational one like we are used to using. If we use the Java environment we will need to learn how to use Java Data Objects because this is what the Google App Engine uses for data management. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by researching the requirements for the Google App Engine. We will need to program separate servlets within the Google App Engine and test how their flow goes once on the server. Another goal for Sprint 7 is developing the data structures required for the project as well as the advanced programming interface to handle the structured query statements we will need to call and get data from these tables. Finally we will make any changes to our past design if we find that the Google App Engine is insufficient.
8.1.8 - Sprint 8
During sprint eight we plan to build a simple android application with a few test commands. Figure 8 shows our three step process we hope to do in this weekly sprint. We will want the Android device to be able to first communicate with the Google App Engine and execute a command. During this sprint we will also need to build some application programming interface driven structured queries statements and test its functionality as well as performance. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by researching the requirements the Android application. Next we will need to create the hypertext transport protocol requests to the Google App Engine. Sprint 8 is not concerned with graphics or design but more functionality testing. Finally we will make any changes to our past design if we find that Android and Google App Engine are not a fitting pair.

8.1.9 - Sprint 9
During sprint nine we plan to do a complete communication test run. Figure 9 shows our three step process we hope to do in this weekly sprint. We have called this “Trigger a Command!” in our proof of concept section. It will send a
message from the Android device to the Google App Engine. The Google App Engine will then send a command to the Stellaris who will then turn on its chime. We will need to build our lightweight internet protocol advanced programming interface at this point as well. After one week we will hold a meeting to discuss our progress and problems.

![Sprint 9 Timeline](image)

**Figure 8-9: Sprint 9 Timeline**

### 8.1.10 - Sprint 10
During sprint ten we plan to assemble our lighting system. Figure 10 shows our three step process we hope to do in this weekly sprint. We figure by now our printed circuit boards have come in and we are ready to start assembling our separate subsystems. Our first subsystem we want to build is the lighting subsystem. We believe this is the easiest to get working and once we get the first subsystem working it will act as a model for the rest of the subsystems. This will be a fully functional application from Android to subsystem. We anticipate to spend more than one week working on this sprint because this will be the first time we will be programming the Stellaris main controller to communicate with the MSP430 subsystems. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by assembling our lighting subsystem. Next we will program the required advanced programming interface for the complete link from Android to light switch. We will also test the dimming feature we have implemented to see how well it works.
8.1.11 - Sprint 11
During sprint eleven we plan to build the heating ventilation and air conditioning subsystem. Figure 11 shows our three step process we hope to do in this weekly sprint. We will want to build two different scenarios for the heating ventilation and air conditioning subsystem. One for at home where it actually interfaces with our heating ventilation and air conditioning unit in the wall and another for the purposes of demoing our project at the end of the semester. Since we can not bring in an actual heating ventilation and air conditioning system to the demo we will build a small demo system showing its digital logic and why this would work on an actual application. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by assembling our heating ventilation and air conditioning subsystem. Next we will program the required advanced programming interface for the complete link from Android to heating ventilation and air conditioning subsystem. We will also test the digital logic we have implemented to see how well it works.

8.1.12 - Sprint 12
During sprint twelve we plan to build the television subsystem. Figure 12 shows our three step process we hope to do in this weekly sprint. We believe this
subsystem is going to be the most difficult because it requires infrared codes to be sent from the Google App Engine. We believe it will be more efficient for us to store all of the television codes on the Google App Engine and then send them to the MSP430 rather than hard coding them directly onto the chip itself. This will also help on maintenance since we can just inject codes directly into our database. After one week we will hold a meeting to discuss our progress and problems.

As seen below, we will start by assembling our television subsystem. Next we will program the required advanced programming interface for the complete link from Android to television. We will also a wide range of codes and see how many we can fit on the screen. We hope to get some type of graphical user interface implemented in sprint 12.

![Sprint 12 Timeline](image)

**Figure 8-12: Sprint 12 Timeline**

### 8.1.13 - Sprint 13

During sprint thirteen we plan to implement our “Bag of Words” idea. Figure 13 shows our three step process we hope to do in this weekly sprint. This will be the voice control portion of our project. We will need to do lots of research before we can implement this and will have to choose the right keywords to say. This will also be a time to make our Android application look better and tie everything together for our presentation/demo. This will also be the time to expand on anything we feel necessary before finalizing the project. After one week we will hold a meeting to discuss our progress and problems.
8.2 - Budget and Finances
Our project has been self funded because we do not have a sponsor. We have carefully selected parts to minimize cost around the board.

8.2.1- Budget management
We have selected low cost value line MSP430’s as well as only using one high powered stellaris microcontroller. We have even found plastic cases that only range around $5 each. Google App Engine is a free platform to develop on. Android is another open source platform we are using because it is completely free.

8.2.2- Finances
The main source of money will be John since he wants to keep the final product. We do not plan to spend more than $1,000 but are prepared to go beyond if necessary. Another option is John’s parents because they are devoted to his education.
Appendices

Appendix A – Copyright Permissions

Google App Engine:

Datastore Inheritance Strategies:

On Sun, Apr 1, 2012 at 4:49 AM, Andy Jefferson <andy@datanucleus.com> wrote:
Hi Jimmy,

> I am part of an Electrical Engineering/Computer Engineering senior design
> team at the University of Central Florida. We are currently designing and
> researching on how to implement a home automation system. For user
> communication with various appliances, we are planning on using an Android
> application with Google App Engine support. For the data storage aspect of
> the software, we are planning on using JPA with the App Engine.
>
> In doing so, I was planning on using documentation from your website about
> inheritance strategies, concerning data storage. I was wondering if I
> could have permission to reference your documentation and use some of the
> pictures on the "Inheritance Strategies" page. This is a non-profit,
> university sanctioned assignment.

Referencing the DataNucleus documentation is no problem, and using some of the
inheritance strategies images would be ok, as long as you put a footnote of
their origin.

Will your assignment be published on a (public) website? if so, would it be
ok (when you have something published) to include your project in
http://www.datanucleus.org/project/usage.html with a brief description?
Obviously wouldn't need to do that until you have something ready.

Regards
--
Andy
DataNucleus (http://www.datanucleus.com)

Television Remote Control:

Bassuk, Larry l-bassuk@ti.com
to me:

Thank you for your interest in Texas Instruments. We grant the permission you request in your
email below.

On each copy, please provide the following credit:

Courtesy Texas Instruments

Regards,

Larry Bassuk
Deputy General Patent Counsel &
Copyright Counsel
Texas Instruments Incorporated
214-479-1152
From: Heston Posner [mailto:heston1919@gmail.com]
Sent: Tuesday, April 03, 2012 3:00 PM
To: copyrightcounsel@list.ti.com - Copyright and trademark web requests (May contain non-TIers)
Subject: [Requests & questions from ti.com] Permission Request

Hello,

I am an Electrical Engineering undergraduate at the University of Central Florida working with a team on a senior design project. We are designing a home automation system in which I plan to use the infrared remote MSP430 for the television subsystem.

Since we will be using this remote control transmitter I would like to request permission to reference your documentation on this product as well as use some of the schematics/figures or parts of them. The exact page I am referring to is http://www.ti.com/lit/an/slla175/slla175.pdf. This is a non-profit, university sanctioned assignment.

Thank you,

Heston Posner
Heston1919@gmail.com

From: Kyle D’Arcangelis

to rkirchhof

I am an Electrical Engineering student at the University of Central Florida who is doing a Senior Design project that is very similar to yours. Part of the report requires us to reference existing projects. If possible, would you be able to give me permission to use some of your information posted on your website? Any parts used will be credited accordingly.

Thanks,
Kyle D’Arcangelis
University of Central Florida
Appendix B – Datasheets
Digital Temperature Sensor with I²C™ Interface

FEATURES
- DIGITAL OUTPUT: I²C Serial 2-Wire
- RESOLUTION: 9- to 12-Bits, User-Selectable
- ACCURACY:
  ±2.0°C from −25°C to +85°C (max)
  ±3.0°C from −55°C to +125°C (max)
- LOW QUIESCENT CURRENT:
  45μA, 6.1μA Standby
- WIDE SUPPLY RANGE: 2.7V to 5.5V
- TINY SOT23-6 PACKAGE

APPLICATIONS
- POWER-SUPPLY TEMPERATURE MONITORING
- COMPUTER PERIPHERAL THERMAL PROTECTION
- NOTEBOOK COMPUTERS
- CELL PHONES
- BATTERY MANAGEMENT
- OFFICE MACHINES
- THERMOSTAT CONTROLS
- ENVIRONMENTAL MONITORING AND HVAC
- ELECTROMECHANICAL DEVICE TEMPERATURE

DESCRIPTION
The TMP100 and TMP101 are two-wire, serial output temperature sensors available in SOT23-6 packages. Requiring no external components, the TMP100 and TMP101 are capable of reading temperatures with a resolution of 0.0625°C.

The TMP100 and TMP101 feature SMBus and I²C interface compatibility, with the TMP100 allowing up to eight devices on one bus. The TMP101 offers SMBus alert function with up to three devices per bus.

The TMP100 and TMP101 are ideal for extended temperature measurement in a variety of communication, computer, consumer, environmental, industrial, and instrumentation applications.

The TMP100 and TMP101 are specified for operation over a temperature range of −55°C to +125°C.
MIXED SIGNAL MICROCONTROLLER

FEATURES

- Low Supply-Voltage Range: 1.8 V to 3.6 V
- Ultra-Low Power Consumption
  - Active Mode: 230 µA at 1 MHz, 2.2 V
  - Standby Mode: 0.5 µA
  - Off Mode (RAM Retention): 0.1 µA
- Five Power-Saving Modes
- Ultra-Fast Wake-Up From Standby Mode in Less Than 1 µs
- 16-Bit RISC Architecture, 62.5-ns Instruction Cycle Time
- Basic Clock Module Configurations
  - Internal Frequencies up to 16 MHz With Four Calibrated Frequency
  - Internal Very-Low-Power Low-Frequency (LF) Oscillator
  - 32-kHz Crystal
  - External Digital Clock Source
- Two 16-Bit Timer_A With Three Capture/Compare Registers
- Up to 24 Touch-Sense-Enabled I/O Pins
- Universal Serial Communication Interface (USCI)
  - Enhanced UART Supporting Auto Baudrate Detection (LIN)
  - IrDA Encoder and Decoder
  - Synchronous SPI
  - I²C™
- On-Chip Comparator for Analog Signal Compare Function or Slope Analog-to-Digital (A/D) Conversion
- 10-Bit 200-kSps Analog-to-Digital (A/D) Converter With Internal Reference, Sample-and-Hold, and Autoscan (See Table 1)
- Brownout Detector
- Serial Onboard Programming, No External Programming Voltage Needed, Programmable Code Protection by Security Fuse
- On-Chip Emulation Logic With Spy-Bi-Wire Interface
- Family Members are Summarized in Table 1
- Package Options
  - TSSOP: 20 Pin, 28 Pin
  - PDIP: 20 Pin
  - QFN: 32 Pin
- For Complete Module Descriptions, See the MSP430x2xx Family User’s Guide (SLAU144)

DESCRIPTION

The Texas Instruments MSP430 family of ultra-low-power microcontrollers consists of several devices featuring different sets of peripherals targeted for various applications. The architecture, combined with five low-power modes, is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 1 µs.

The MSP430G2x13 and MSP430G2x53 series are ultra-low-power mixed signal microcontrollers with built-in 16-bit timers, up to 24 I/O touch-sense-enabled pins, a versatile analog comparator, and built-in communication capability using the universal serial communication interface. In addition, the MSP430G2x53 family members have a 10-bit analog-to-digital (A/D) converter. For configuration details see Table 1.

Typical applications include low-cost sensor systems that capture analog signals, convert them to digital values, and then process the data for display or for transmission to a host system.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.
MSP430x12x
MIXED SIGNAL MICROCONTROLLER

SLAS312C – JULY 2001 – REVISED SEPTEMBER 2004

- Low Supply Voltage Range 1.8 V to 3.6 V
- Ultralow-Power Consumption:
  - Active Mode: 200 µA at 1 MHz, 2.2 V
  - Standby Mode: 0.7 µA
  - Off Mode (RAM Retention): 0.1 µA
- Five Power Saving Modes
- Wake-Up From Standby Mode in less than 8 µs
- 16-Bit RISC Architecture, 125 ns Instruction Cycle Time
- Basic Clock Module Configurations:
  - Various Internal Resistors
  - Single External Resistor
  - 32 kHz Crystal
  - High Frequency Crystal
  - Resonator
  - External Clock Source
- 16-Bit Timer_A With Three Capture/Compare Registers
- On-Chip Comparator for Analog Signal
  Compare Function or Slope A/D Conversion
- Serial Communication Interface (USART0)
  Software-Selects Asynchronous UART or
  Synchronous SPI
- Serial Onboard Programming,
  No External Programming Voltage Needed
  Programmable Code Protection by Security
  Fuse
- Family Members Include:
  MSP430F122:
  - 4KB + 256B Flash Memory
  - 256B RAM
- MSP430F123:
  - 8KB + 256B Flash Memory
  - 256B RAM
- Available in a 28-Pin Plastic Small-Outline
  Wide Body (SOWB) Package, 28-Pin Plastic
  Thin Shrink Small-Outline Package
  (TSSOP) and 32-Pin QFN Package
- For Complete Module Descriptions, See the
  MSP430x1xx Family User’s Guide,
  Literature Number SLAU049

description

The Texas Instruments MSP430 family of ultralow power microcontrollers consist of several devices featuring different sets of peripherals targeted for various applications. The architecture, combined with five low power modes is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that attribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 6 µs.

The MSP430F12x series is an ultralow-power mixed signal microcontroller with a built-in 16-bit timer and twenty-two I/O pins. The MSP430F12x series also has a built-in communication capability using asynchronous (UART) and synchronous (SPI) protocols in addition to a versatile analog comparator.

Typical applications include sensor systems that capture analog signals, convert them to digital values, and then process the data and display them or transmit them to a host system. Stand-alone RF sensor front end is another area of application. The I/O port inputs provide single slope A/D conversion capability on resistive sensors.

<table>
<thead>
<tr>
<th>TA</th>
<th>PACKAGED DEVICES</th>
</tr>
</thead>
<tbody>
<tr>
<td>-40°C to 85°C</td>
<td>PLASTIC 28-PIN SOWB (DW)</td>
</tr>
<tr>
<td>DW</td>
<td>MSP430F122IDW</td>
</tr>
<tr>
<td>PW</td>
<td>MSP430F123IDW</td>
</tr>
</tbody>
</table>

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers there are appears at the end of this data sheet.
Figure 1-1. Stellaris LM3S8962 Microcontroller High-Level Block Diagram
CC1000
Single Chip Very Low Power RF Transceiver

Applications
- Very low power UHF wireless data transmitters and receivers
- 315 / 433 / 868 and 915 MHz ISM/SRD band systems
- RKE – Two-way Remote Keyless Entry
- Home automation
- Wireless alarm and security systems
- AMR – Automatic Meter Reading
- Low power telemetry
- Toys

Product Description
CC1000 is a true single-chip UHF transceiver designed for very low power and very low voltage wireless applications. The circuit is mainly intended for the ISM (Industrial, Scientific and Medical) and SRD (Short Range Device) frequency bands at 315, 433, 868 and 915 MHz, but can easily be programmed for operation at other frequencies in the 300-1000 MHz range.

The main operating parameters of CC1000 can be programmed via an easy-to-interface serial bus, thus making CC1000 a very flexible and easy to use transceiver. In a typical system CC1000 will be used together with a microcontroller and a few external passive components.

Features
- True single chip UHF RF transceiver
- Very low current consumption
- Frequency range 300 – 1000 MHz
- Integrated bit synchroniser
- High sensitivity (typical -110 dBm at 2.4 kBaud)
- Programmable output power ~20 to 10 dBm
- Small size (TSSOP-28 package)
- Low supply voltage (2.1 V to 3.6 V)
- Very few external components required
- No external RF switch / IF filter required
- RSSI output
- Single port antenna connection
- FSK data rate up to 76.8 kBaud
- Complies with EN 300 220 and FCC CFR47 part 15
- FSK modulation spectrum shaping
- Programmable frequency in 250 Hz steps makes crystal temperature drift compensation possible without TCXO
- Suitable for frequency hopping protocols
- Development kit available
- Easy-to-use software for generating the CC1000 configuration data

This document contains information on a pre-production product. Specifications and information herein are subject to change without notice.