ECE 480 Team 3
Final Report
Home Automation Control Panel

Date: 4/29/2009

Team Members:
Rituraj Behera
Samuel Flynn
Da Ke
Sungsoo Kim
Eric Myers

Faculty Facilitator:
Michael Shanblatt

Sponsor: Texas Instruments
Ram Sathappan
Jay Shastry
Executive Summary

Home automation systems are an emerging trend that will soon become standard in all modern houses. In order to help Texas Instruments anticipate and take advantage of this trend, our team has designed a sophisticated but yet very intuitive and low-cost home automation control panel system based on Texas Instruments' latest OMAP3 processor. We have created an interactive user interface that will allow our users to obtain and utilize real-time information collected from different devices in their house. We made the interface look very presentable, aesthetic and efficient. Our modular design allows for multiple devices, including lights and a climate control fan, to be plugged and unplugged as needed. All this was done on a low-cost Beagle Board, which incorporates the TI OMAP3530 processor. With the help of the various ports on the Beagle Board, all the various electronic modules work on the control panel concurrently. In addition, our system automates the control process by applying an input-driven feedback system. For example, devices such as climate control are managed by our system automatically, based on current temperature in the house. All the information one needs to see and manipulate in the household can be done with greater simplicity.

Our Home Automation system demonstrates the TI OMAP3 processor's capabilities as a potential industry leader in the field of home automation due to its low cost and high performance.
Acknowledgments

Team 3 would like to thank following people for their guidance and help:

Dr. Michael Shanblatt - Facilitator
Ram Sathappan - Texas Instruments
Jay Shastry - Texas Instruments
Prabindh Sundareson - Texas Instruments India
Karthik Swaminathan - Texas Instruments India
Amit Chatterjee - Texas Instruments India
Khasim Syed Mohammed - Texas Instruments India
Pratheesh Gangadhar - Texas Instruments India
# Table of Contents:

1. Chapter 1 (Background and Introduction) .................................................. 5  
   - Background ................................................................................................. 5  
   - Introduction ............................................................................................... 5  

2. Chapter 2 (Problems and Solution) ............................................................. 7  
   - Exploring Solution space ........................................................................... 7  
   - Budget ......................................................................................................... 9  
   - Gantt Chart ................................................................................................. 9  

3. Chapter 3 (Technical Work) ................................................................. 12  
   - Software  
     - Operating system Configuration .......................................................... 13  
     - Java Coding ........................................................................................... 13  
   - Hardware  
     - Hardware Development ........................................................................ 20  

4. Chapter 4 (Testing and Results) ................................................................. 22  
   - Testing and Results ................................................................................... 22  

5. Chapter 5 ........................................................................................................ 24  

6. Appendix 1 (Technical Roles) ................................................................. 25  
   - Da Ke ........................................................................................................ 25  
   - Eric Myers ................................................................................................. 26  
   - Samuel Flynn ............................................................................................ 27  
   - Sungsoo Kim .............................................................................................. 28  
   - Rituraj Behera ........................................................................................... 29  

7. Appendix 2 (Web References) ............................................................... 31  
   - Web References .......................................................................................... 31
Chapter 1- Background and Introduction

Background

Home automation control panels have been in the marketplace for a few years now. They are currently installed only in luxury homes, and tend to be very costly to set up. This high cost mostly stems from an expensive embedded system and complex device controllers that need to be customized for every device a customer needs automated. Our task is to make a simpler control panel, with lower cost and easier installation.

Before this can be a successful product, though, it needs to have consumer appeal. In order to appeal to a wide consumer base, the control panel needs to have an interactive graphical user interface (GUI) with which the users can issue commands and receive information about their house. The most important goals of a good GUI are aesthetics and user-friendliness. With the advent of more powerful computers and embedded systems, graphical systems on user interfaces have become increasingly complex. For example, modern cellular phones have very intricate menus, full of animations, colorful displays, and other visual effects. This control panel must match the user appeal of systems like these.

Introduction

Modern consumer electronics are also characterized as being intuitive and easy to use. Users today expect to be able to use their product right out of the box, without having to read instruction manuals. This control panel must also conform to this trend. While most users will probably know how to use a keyboard and mouse, the GUI system must be usable by those who don't. Therefore, a touch screen interface will be implemented. By putting large, well-labeled buttons on the screen with a touch screen input, users can interact with buttons and menus directly. Combined with simple and well-organized menus, a user can find the information or commands they seek with just a couple of button presses.

Another important aspect of our control panel is its ability to connect to various devices around the home quickly and without expensive installation procedures. Two of
the most prominent systems to control in a house are the lighting and climate control systems. Many existing control panel systems require the homeowners to hire professional installers to custom build a system that only works on their specific home. Our task is to simplify this by using an embedded system and some simple circuitry to control power to these basic systems. By controlling the power to the lights and climate control system directly, rather than controlling the switches that control them, fewer parts are needed and installation becomes much simpler.

By combining a professional user interface, a low-cost system, and easy installation, this system will transform the home automation control panel from a luxury commodity to a commonplace appliance. This product will become as commonplace as computers in modern households.

As the technology keeps improving and systems get more and more sophisticated, it is important that we take advantage of such technologies and integrate them into our design to create something useful and efficient. Wireless communication has been improving over the past years. So, taking into consideration that even common electronic devices can now be controlled remotely, our team has designed a Control panel that users can control from anywhere, using the internet. So, a person can easily turn on or off even his room lights or check the status of appliances like the washing machine on the Control Panel screen.

The Home Automation Control Panel that we have designed and developed with the help of latest hardware and software configurations makes home a much convenient and sophisticated place to stay in. We definitely believe that our team has been successful in the development of next level Control Panel. With the help of the combination of touch screen technology, big icons on the GUI and user friendly menu and sub menu implementations, it is easy for residents to use this product for a range of electronic household devices.
Chapter 2 – Problems and Solutions

Problems and Solutions

Problem 1: An embedded system capable of running the control panel

Solution 1: Beagle Board

While there are many embedded computing platforms that exist in the market, the Beagle Board from Texas Instruments stands out from rest of the crowd with its ultra-low cost and energy efficiency. The Beagle Board, an OMAP3530 platform designed specifically for open source community, has been equipped with a set of features to allow the user to experience the power of the latest OMAP3530 processor. By applying standard ports and sockets, the Beagle Board is highly extensible to add many features and interfaces.

The OMAP3530 processor with a clock frequency of 600 MHz is based on the market's first board offering of the ARM Cortex A8 Core processor. This provides an unprecedented combination of laptop-like performance at hand held power levels on a single chip. This processor is integrated to a 0.4mm pitch Package on Package (POP) package. The POP technique is useful in integrating the main memory with the processor. This makes the Beagle Board portable while still providing a memory system of 256MB NAND RAM and another 256MB of DDR RAM to work with our Home Automation Human Machine Interface (HMI) demo. The Beagle Board also provides two different types of power connectors, one for the USB Power used in all our various I/O devices like Touch screen driver and keyboard, and a DC power for additional power supply using an AC to DC converter. In addition, another 3.3V regulated voltage supply is provided for the DVI-D encoder, which is used for the Main Panel display on the Touch Screen monitor.

The Beagle Board is also quite inexpensive. At $150, it is an affordable way to provide plenty of processing power. Compared to the $1,500 price tag of the OMAP3530 EVM, it is a far better choice for implementing a low-cost system.

The above reasons clearly suggest why the Beagle Board was the most ideal embedded system for our Home Automation HMI Demo design.
Problem 2: A graphical development toolkit capable of handling the requirements of the control panel

Solution: Java Virtual Machine with the Swing Toolkit

For the development of our GUI we decided to use the Java Swing Toolkit as the graphical package for the Control Panel. The entire software design that we needed to implement was done on the java Swing platform due to its high level of convenience and advanced development platform. Also the java swing package provides a set of "lightweight" (all-Java language) components that, in most cases, work the same on all platforms with a Java Virtual Machine (JVM). Java Development Kit (JDK) makes it easy and desirable to create media-rich Java applets and Panels. Also, if coding becomes extremely complicated, Java Netbeans, another GUI development kit provided by Sun Microsystems can be easily used by just dragging and dropping java applets, panels and labels. Hence the software development becomes more flexible and either Eclipse or Netbeans can be used as Integrated Development Environment (IDE) for GUI design implementation.

Problem 3: Connect and Control External Devices

Solution 3: General Purpose Input/Output (GPIO) Pins on the Beagle Board's expansion slot

The problem of connecting external devices to the Beagle Board had many potential solutions. At first, we pursued a way to create a custom USB device, which would connect to the Beagle Board and switch lights and fans on and off. This offered several benefits, as a USB device could potentially be used on other systems, such as home personal computers. Unfortunately, this option also had a few challenges. In order to create a custom USB device, a second microcontroller would be needed to handle the communications with the Beagle Board. Getting another microcontroller system set up would have been too costly and time consuming, so another option was needed.

The other option was to use the GPIO pins provided by the Beagle Board expansion slot. This had the advantage of being much simpler than the USB device, as the GPIO
pins didn't require advanced communication procedures. The disadvantage of this system is that it offers much lower resolution, as signals had to be sent in parallel, rather than in series. This was not a very serious an issue, though, as we were simply controlling whether the lights were on or off, not their brightness. Therefore, we decided to hook the lights and climate control fan up to the GPIO pins.

Problem 4: Remote Control
Solution 4: Internet (SSH, HTTP, VNC)

Being able to control appliances while away from home is certainly an appealing feature to potential customer. This is a high cost feature in most of the existing home automation system. Fortunately, there are a lot of tools from open-source community that allows us to integrate this great feature into our system with virtually no cost. We were able to have this feature by setting up a SSH or VNC server on our Beagle board and connect to it from portable devices like laptop computer and cell phone.

Budget Estimation

At the outset of the project, we came up with a rough budget estimate of what our project would cost.

<table>
<thead>
<tr>
<th>Description</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beagle Board and Accessories</td>
<td>$250</td>
</tr>
<tr>
<td>Touch screen Monitor</td>
<td>$500</td>
</tr>
<tr>
<td>Interfaced Physical Devices</td>
<td>$50</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>$800</strong></td>
</tr>
</tbody>
</table>
Chapter 3- Technical Work

Operating System Configuration:
Once we decided on the Beagle Board for our embedded system, it was time to begin configuring the Angstrom operating system so that the Beagle Board could boot it and run our various programs. We installed the Angstrom Linux on the Beagle Board with the help of an external 8GB SD Card which is used as a hard drive for our Board. Several software packages needs to be installed in order to make our HMI control system work properly. This includes:

- Java Virtual Machine (Cacao)
- Video for Linux (Driver for our video camera)
- Mplayer for Webcam (For real time video player)
- Motion (For motion detection using our video camera)
- OpenSSH or VNC Server (For remote control)
- Touch screen driver provided by TI (To get the touch screen working)

Java Coding:
The programming running from our Java code is what the user sees when they use the control panel. Therefore most of the product's user appeal is highly dependent on the Java running properly, efficiently, and without major bugs. Programming the control panel made up the majority of the technical work. Luckily, Java runs using a Virtual Machine, which uses intermediate code, rather than machine-specific assembly code. This means that code can be developed and compiled onto one system, and then transferred to another system without the need to re-compile. This gives the system a great degree of portability. It can be moved to any system capable of running Java.

In order to get the programming started, though, we had to set up a window on which the menus would be displayed. This was done by creating a class called MainPanel. This class starts up a JFrame and a JPanel, which is the window that pops up when the program is run. Once it was successfully, JButtons and JLabels, which are the Swing toolkit's standard buttons and labels, were added to the JPanel. They
were not appearing on display where the code had placed them, however. In order to have explicit control over the size and placement of GUI elements, the JFrame's layout managed needed to be disabled.

Once that was done, ActionListeners were added to the buttons. These ActionListeners would run a function every time their button was pressed. Initially, they were just used to change the text on JLabels, but later were used to do any number of tasks, such as hiding or showing JPanels, changing internal settings, or launching new applications through kernel console commands.

Unfortunately, all of the JButtons and JPanels used until this point used default Java images. While these images were functional, they were not very aesthetically interesting. The dull gray colors would not have generated the user appeal that was key to the control panel's success. Therefore, our team had to create all new images that would stand out to the users. Button images were created with reflective surfaces and menu borders were redone to appear to have raised surfaces and to cast shadows. Every visual element was redone, so no standard Java images were shown.

Unfortunately, all of the new graphics caused some performance issues. All of these new pictures needed to be loaded into Image objects. The images themselves were not loaded into the system's memory, though, and had to be reloaded from the Beagle Board's SD card every time it was drawn onto the screen. In order to fix this problem, BufferedImage objects needed to be created. BufferedImages can't load files directly, but can have other visual elements drawn onto them. By loading an image file into an Image object, then drawing that Image object onto a BufferedImage, the file would be effectively loaded into memory. An ImageManager class was created in order to handle this task. Before the program started up, the ImageManager would create a BufferedImage for every image and save it in a hash table, so they could be easily looked up by other parts of the program. This ImageManager system greatly improved performance at the cost of a long loading time for the program, as the cost of drawing an Image onto a BufferedImage is quite high.

Once the system for handling custom images was in place, it was time to begin creating the class structure for the panel. Our team decided to use a submenu system. The front panel would display several buttons, each of which launched a submenu. A
submenu would pop up in the center of the screen and display several information or command options to the user, depending on that submenu's particular function. The user could then navigate back to the front panel by pressing the "Home" button, or could open another submenu by pressing its button.

In order to begin implementing the submenu system, a SubMenu superclass was created. Each object contained the menu's name, which was used to determine what images were associated with it. It also contained a JButton, which was displayed on the front panel. The JButton contained two default ActionListeners, a submenu closer, and a submenu opener. The submenu closer hid and disabled all other submenus, which prevented two menus from being visible at the same time. The submenu opener opened the submenu that owned that particular button. Therefore, pressing any submenu button would close all open submenus and display a new one.

Once the SubMenu superclass was created, individual submenu classes, which derived from the superclass, were created. By deriving from SubMenu, all of the new classes had the same properties, like images and buttons, which the superclass had.

The first submenu to get a concrete definition was the climate control. The purpose of the climate control menu was to display the house's temperature, the temperature setting, and the status of the system. In order to provide a familiar and intuitive control system, the menu displays an old, rotary-style thermostat. The user changes the temperature setting simply by touching the thermostat and rotating it around its center. The display updates to the new setting as the user moves their finger, and, when the setting gets too far from the actual room's temperature, the system status changes from off to heat or cool.

While this appears to be a simple system to the user, the mechanics behind it are quite complex. First, the submenu needed to determine the angle of a finger touch with respect to the thermostat's center. Then, as the finger was dragged, the angle needed to be continuously updated until the finger was released. Finally, as the angle of the finger touch changed, the image that displayed the thermostat's pointer needed to be rotated. While this could have been done by creating a new BufferedImage object and drawing on the old BufferedImage with an angle offset, it turned out to be very slow on the Beagle Board. Instead, an AffineTransform operation was used.
Rather than affecting how the image is stored in memory, the AffineTransform affects how it's drawn to the screen. By overriding the default paint function and adding the AffineTransform, the image was rotated with the exact same angle as the user’s finger.

The next submenu to be implemented was the lighting controller. The lighting menu displays a floor plan of the house, with boxes around each room. By default, each room is colored yellow, to signify the lights in the room are on. To toggle the lights in any given room, the user simply needs to touch that room. While this was originally done by drawing a solid color behind the floor plan and an invisible button on top, the order in which the Java Virtual Machine would draw the images was unpredictable. Instead, a button with a semi-transparent color was placed over the floor plan. The floor was still visible behind the button, and it was immediately obvious if the light was on or off.

The security submenu was the next one finished. This menu's purpose is to allow the user to enter a security code to arm or disarm the system. In order to do this without the use of a keyboard, a number pad needed to be implemented. However, a number pad was going to be needed to allow the user to change their zip code and the settings of their appliances, too. Therefore, rather than building the number pad into the security submenu, a NumPad class was created. Once created, the numpad class could be added to the security submenu, the appliances submenu, and the settings submenu. In the security menu, the user enters a number, which is saved and causes the system to arm itself. In order to disarm the system, the same number must to be entered again. However, if a wrong number is entered more than twice, the security system is tripped and goes into alert status.

There was one major problem in implementing the security system. The MainPanel class contained a security submenu, but the security submenu needed to contain a pointer to the MainPanel in order to send alert messages and function calls. This causes a problem known as circular dependency, where two classes directly reference the other. This is a very undesirable setup, as it can be unstable and makes changes difficult. In order to fix this problem, the MainPanel needed to implement an interface. The interface, called Lockable, forced the MainPanel to have two functions, Lock and
Unlock. By forcing the MainPanel to implement these functions, the security submenu could contain a pointer to a Lockable object, rather than a MainPanel. The security menu could then call Lock and Unlock on the MainPanel without knowing exactly what kind of class MainPanel was. This broke the circular dependence and made the system much more stable.

The settings submenu was the next one to be implemented. Unlike the previous submenus, the settings have several categories which the user can choose between. They can view the current settings, change their zip code, or choose between metric and English units. This presents too much data and options to be displayed on the screen at once, so it had to be broken up into separate panels. These new panels are displayed only when their button is pressed in the settings submenu. Much like the submenu closer ActionListener all Submenus carry, these smaller panels close all other panels before displaying themselves.

Next to be finished was the weather submenu. The weather submenu takes advantage of a service provided by www.weather.com in which an XML table containing weather information is downloaded through an internet connection. Unfortunately, the XML table needed to be parsed in order to extract useful information from its various data fields. To do this, a class that could get specific values from the table was needed. This took the form of a class called WeatherInfoManager and a class called XMLParser. The WeatherInfoManager determined what information was needed and made a call to the XMLParser. The XMLParser then used the javax.xml.parsers package to extract that information from the XML file. For example, the high temperature for the following day is located in the XML table under weather/dayf/day d=1/hi/. The WeatherInfoManager would then call the XMLParser and tell it to get the string under that directory. The XMLParser would access the XML file, get the information, and pass it back to the WeatherInfoManager, which would then pass it to the weather submenu. When this process is repeated for every field in the entire submenu, they are all updated with the weather information from the XML table.

Once the system for the weather was in place, the next logical step was to introduce the news submenu. Like the weather system, the news submenu relies on XML tables.
retrieved over the internet to get up-to-date new information. Because the news XML table is structured differently from the weather one, a new information manager was needed. The NewsInfoManager is very similar to the WeatherInfoManager in that it uses the XMLParser class to get at the data in the XML table. The only difference is the directories it looks under. For example, to get the headline's title, the NewsInfoManager gets the data under channel/item/title.

The last of our traditional submenus was the appliances menu. Like the security submenu, the appliances submenu is purely demonstrative, with no external connections to physical devices or sensors. When first brought up, the menu displays a floor plan just like the lighting submenu. When a user clicks on a room, the status of all of the room’s appliances is displayed. From there, the user can select an appliance and manipulate its controls. For example, the den has a TV, a Tivo, an audio player, and a DVD player. For each of these devices, the user can toggle the device on or off and change the channel or track using a number pad. Other rooms contain different appliances. The kitchen has a refrigerator, a microwave oven, a cooking range, a conventional oven, and a dishwasher, while the utility room has a washer and dryer to control.

The only remaining submenu is a little different from the rest. Rather than controlling internal options or manipulating displays, the video submenu needed to launch a video player. On the Angstrom system, the primary application capable of playing a webcam feed is called mplayer. The other application that can use a webcam to retrieve useful information is called motion. Instead of launching a window that plays a live feed, motion runs entirely in the background. Whenever the picture from the webcam changes significantly, motion captures a picture and saves it to the SD card. This effectively created a video recording system with a motion detector.

Both the motion detector and the traditional mplayer feed needed to be implemented in the video submenu. Therefore, only two buttons were added to the submenu, one to launch motion and one to launch mplayer. However the functionality behind these two buttons is much more complex than the simple data manipulation most others use. These buttons needed to launch a new process that ran a Linux
kernel command. While there are built-in Java libraries to handle such a process, there were serious problems limiting its effectiveness. Unlike most modern personal computers, the Beagle Board cannot use memory swapping. When most personal computers run out of physical memory, they save some memory to a hard disk and free up some memory space. The Beagle Board does not have that option, so when it runs out of physical memory, no new applications can be launched. Until this point, the process of loading all of the images needed for the GUI had taken up all of the physical memory. The solution to this was twofold: compress all images to GIF files, and upgrade the Beagle Board from revision B7 to C2. The compressed GIF files dramatically reduced loading times and memory usage. Revision C2 of the Beagle Board improves on the older versions by doubling the physical memory from 128 megabytes to 256. These two improvements not only allowed the video players to launch, but also freed up enough system resources to allow for the remote control protocols.

With all of the submenus ready to go, the only thing left to do was to configure the control panel to manipulate the processor's GPIO ports. In order to do this, the MainPanel class had to first launch processes that configured the GPIO ports for manually controlled output. To do this, it runs the echo command with the GPIO number on an export program. For example in order to configure port 139, the MainPanel would run the console command "echo 139 > /sys/class/gpio/export". Next when the program closes, the MainPanel needed to send the GPIO numbers to the unexport program.

Once the GPIO ports were ready to use, the climate control and lighting submenus needed to be updated to use them. In order to toggle the ports' status, more console commands were needed. This involved passing a string into a new program, direction, which was created when the ports were exported. For example, in order to set port 139 to high, the code runs the command "echo "high" > /sys/class/gpio/gpio139/direction". Each of the lights under the lighting submenu was assigned their own GPIO port. By doing this, the GPIO status now toggles every time the user presses a light under the submenu. Likewise, the climate control system was given its own port. When the climate control status is off, its port is set to "low."
When the temperature goes more than five degrees away from the setting, the port is set to "high." By running these low and high signals out of the Beagle Board, they could control actual devices.

**Hardware Development and Testing:**

Hardware had to be developed to test the GUI. Since a 1.8V signal from the GPIO ports of the Beagle board would be used to control whether a device was on or not, a switch would have to be used to determine if that signal was being received. The switch is a transistor, a TIP31A NPN transistor, with the emitter connected to ground, the base connected to a 100 ohm resistor which was connected to the 1.8V signal from the GPIO ports, and the collector as the output. It would be better to test more than one device because the Human Machine Interface (HMI) in an actual home would not just control one item. Therefore, a Printed Circuit Board (PCB) was developed that incorporated twelve switches (See Appendix 5). Each switch controls a USB port. The USB port was designed to simply turn on or off based on the input into the switch; no data is transmitted to the USB port. A voltage is constantly sent to the USB socket, but the switch controls whether or not the ground is connected to the USB socket. Every USB socket was connected to the same power source, but each individual USB socket was connected to a separate switch. After the connection of the USB sockets, each socket was glued into a hole in the side of an enclosure to prevent any parts from moving or being damaged accidently.

It would be quite a hassle to run individual loose wires between our Beagle board and the PCB. Fourteen wires would be needed to have the twelve 1.8V GPIO signals, one voltage signal, and one ground signal transmit the needed information. So that many individual loose wires were not used, a 16 wire ribbon cable was used. This allowed for the needed fourteen wires, but also provided room in case something happened to one of the wires in use. A lengthy female ribbon cable was soldered onto the PCB, using twelve wires for GPIO input, one for ground, and one for power. A shorter female ribbon cable was soldered onto the Beagle board so the female part could be glued into the enclosure for the Beagle board. A connector, with
both sides being male, is then used to connect the female ends from the PCB and Beagle board.

Devices needed to be created or used to connect to the USB ports. Devices that simulate different submenus of the HMI. Two of the submenus had parts chosen that showed the effectiveness off the HMI. Firstly, a USB fan was purchased. This fan already came completely made. It simply had to be connected to a B-type USB port. The fan simulates a climate control device, such as an air conditioner. The second device that was created was a small Light Emitting Diode (LED) circuit (See Appendix 6). This circuit was developed to simulate the lighting of a house. On the lighting submenu, ten different lighting areas can be turned on or off. Because of this, ten LED circuits were created. Each circuit was connected to an A-type USB cable so they could be used with the USB ports. The ground of the USB cable was connected to the cathode of the LED. The anode of the LED was then connected to a 270 ohm resistor. This was done because the LED is rated for 3.3V and the voltage coming in from the USB cable is 5V. The resistor was then connected to the power side of the USB cable.
Chapter 4 - Testing and Results

Several tests needed to be done to correctly design the hardware, as well as make sure the hardware worked correctly. The first test that was done was to determine what the resistor value for the switch should be. This testing was done by simply testing a value and making sure that the switch works as intended. A 100 ohm resistor ended up working.

There was not much of a data sheet that came along with the USB sockets, a test needed to be done to determine what was power and ground for both A-type and B-type. First, the internet was consulted to see what leads are power and ground in general. Using a multimeter, one end was attached to a lead of the USB socket and the other end was attached to a wire from the end of the USB cable. If there was a readout, then the lead from the socket matched the wire from the USB cable. It was determined which leads were power and ground.

The PCB was the next thing that needed to be tested. After everything was soldered on, the power and ground outputs that would be going to the USB sockets needed to be tested. This was done using an oscilloscope. The oscilloscope measured if the wire had the correct output. This tested to make sure the soldering was correct and that the wires were both electrically connected and did not have shorts. There were a few problems during the first test of this. Half of the transistors and one quarter of the resistors were not electrically connected where it was needed. They were resoldered to fix that problem. When the second test was done, only one of the ground wires was not receiving the correct output. There was a bridge of the solder that was sending a voltage to the ground wire. A little solder was removed in order to fix this problem.

The next test was of the ribbon cable attached to the PCB. An input voltage was put into each of the ribbon cable ports. This determined which port correlated to which switch on the PCB so tests would be much easier in the future. The ribbon cable has a red line on one of the wires to differentiate between the sides. Also, this testing allowed
for making the remaining tests much easier.

Next, the PCB needed to be tested to make sure that using the GPIO 1.8V signals would turn a device on or off. The device used was a small motor that the power and ground wires that were to be attached to the USB socket were plugged into the motor for each switch. The GPIO signal was then connected into the corresponding port in the ribbon cable. The first test of this resulted in half of the ports not working. After resoldering, two of the ports were fixed, but four still had problems. These problems were in four ports that were in a row. After checking to determine what the problem was, it was found that there was a problem with the 5V signal having been bridged with the copper plate. After fixing that problem, two of those ports still did not work, as well as a port that had been working before was not working. That was when the realization came that some of the solder was not thick enough to hold after being bumped a little bit. Places where the solder appeared not to have enough were resoldered to fix those problems. At this point, all the ports worked, though two had problems still when bumped. For one of those ports, a wire was changed and it worked fine. For the second port, the transistor was not fully electrically connected. Once that happened that port worked, meaning all ports worked.

The completed LED circuits were then tested to make sure they were wired and soldered correctly. The test was to make sure the LED turned on when the circuit was connected to 5V and Ground. Every LED did work when a 5V signal was applied to the positive wire and when ground was applied to the ground wire.
Chapter 5 – Conclusion

The final product for our Home Automation HMI Demo included mainly the Beagle Board, Beagle Board accessories, 7" Touch screen monitor, Web camera, Power Hub and various other I/O devices for the hardware. In terms of the software, the installation and use of Java swing required for our GUI development was done at absolutely no expense. So, the total expenditure that our team had to do was only for the hardware devices. The breakdown of cost for all hardware parts used is listed on our Budget. Hence, total cost that was required to be spent for a working Home Automation HMI demo was approximately $967 dollars.

We succeeded in making a system that is both easy for homeowner to own and to use. By using a low-cost embedded system, the hardware is much less expensive than home automation control panels currently on the market. In addition, the organization and aesthetics of the user interface gives it high user appeal and renders it usable by a wide consumer base.

In addition, we ended up with many more features than we expected when we began. The news system, remote control, and motion sensors are all additional features unique to our system.
Appendix 1 - Technical Roles:

Da Ke:

My technical responsibility for this design project include install and configure the operating system on the Beagle board, interface different hardware devices with our actual GUI components. A working operating system is essential to the success of our project. Therefore, I spent great amount of time to install and configure the Angstrom Embedded Linux on the Beagle board as soon as it arrived. This is builds the foundation for our group to test our initial GUI application and identify different software and hardware constraints in our system. After setting up the system, I worked with Sam with the design and implementation of the GUI. Based on Sam’s initial GUI design, I went ahead and improve the software with more robust and expandable design. With implementation of this new design, our programming team found the GUI to be more organized and efficient.

After finding it’s impossible for our team to get a working weather band receiver in our system, we decided to get the weather information from the internet. To achieve this, I collaborated with Sungsoo on developing a new piece of software that will collect and extract weather information from weather.com. Our work around to weather band receiver turn out to be much simpler and less costly while allowing us to fulfill the project requirement that our sponsor demanded. In term of hardware interfacing, I successfully integrated our video camera into our system. Initially we had problem with the touch screen, but TI was able to develop a driver and make it work with our Beagle board. I installed touch screen driver provided by TI and configure it to work with our
system. I also explored different ways that we can access our system remotely. I found out that standard Linux protocol like SSH and VNC will work just fine for our purpose. I also did extensive testing on our GUI to make sure everything is working. I created script to allow us easily deploy our newest application to our Beagle board via the internet.

**Eric Myers:**

The GUI that we created needed to be tested. To do this, I had to come up with hardware that would implement a control signal. After doing some research, I created the switch that would be needed to be able to use the control signal. The switch used a TIP31A NPN transistor. Since we had to use many switches to go with many GPIO signals, a Printed Circuit Board needed to be created. I created the PCB to have twelve switches that used the 1.8V signal from the GPIO ports in order to control when the ground is output. The voltage signal itself is completely separate from the switches. The voltage runs directly to the USB sockets.

We also needed a way to deliver the 5V and ground signals that the switch controlled. I decided to use USB because USB is a practical way to transmit power and ground. It is already widely used and buying cables and sockets creates an easy connection. The USB cables were attached to the external devices used for testing and the sockets were connected to power and ground wires that came from the PCB. The USB wires in the cable are easily stripped and distinguishable in the cable so it is easy to connect them to a circuit. The USB sockets use a constant 5V sent to the socket and the ground is used when the GPIO sends a 1.8V signal to the switch corresponding to that socket. I connected the USB wires to an LED circuit that I created. This circuit lights up the LED when the GPIO signal is high. I created the LED circuit using a 5mm high intensity while LEDs.
I also did the testing of these hardware devices. The hardware needed to be tested thoroughly in order to make sure of two things. The first reason was that the hardware had to work. If the hardware was not working correctly then we would not be able to show that out HMI works. The second reason for the extensive testing was to ensure there was no voltage coming back through the ribbon cable to the Beagle board. If this did happen, then the Beagle board would most likely be destroyed and would ruin our entire project. The testing took several days because it was so imperative that everything was tested so thoroughly.

Samuel Flynn:

In order to ensure all parts of the system worked with one another, I worked on all parts of our solution. At the project's outset, I began to research all of the various Java packages and which ones would be needed in order to complete the project. Once we decided on the Swing Toolkit, I began to set up the basic class structure, with the MainPanel class launching the JFrame and primary JPanel. As we began to populate the JButtons and JLabels and realized that original graphics were needed, I took charge of graphics creation. I started with some basic buttons, trying various shapes, colors, and reflective and 3D effects. Once we decided on a red button with a raised glass effect, I created similar menu borders and more complex buttons for specific purposes. When it came time to program the submenus, I took charge of three of them: the climate control, the security, and the lighting. The security and lighting were simple enough, but the climate control system required much more research on MouseListeners and the procedures Java uses to draw objects to the screen. I tried using a rotated BufferedImage system for rotating the dials, but it turned out to be too slow on the Beagle Board, so I had to switch over to AffineTransforms. Unfortunately, the AffineTransforms behave differently under different Java Virtual Machines. The code had to be carefully
configured so it drew correctly on the Beagle Board, even if it was incorrect on lab PC's.

As the Java development continued, it came time to figure out how to control physical devices from the Beagle Board. I worked with Eric to research a solution to this problem. Once we decided on using the Beagle Board GPIO ports, I figured out exactly what kind of voltage and power these ports were capable of delivering. Because the power wasn't enough to run most devices, I assisted Eric with a system that would boost our 1.8 volts up to 5 volts. Together, we designed a modular power system that used USB plugs to allow up to twelve devices to be attached.

While Texas Instruments eventually provided working touch screen drivers, Da and I worked on getting the drivers that came with the screen to compile. The Beagle Board was missing several packages that the compiler needed, so we attempted to obtain them. We were stopped in our tracks, however, when the operating system build headers were required, as these were unavailable to us. Luckily, Texas Instruments managed to find a solution and provide us with working touch screen drivers.

**Sungsoo Kim:**

In the beginning of the project, I worked on the Graphic User Interface design for our team. Considering the size of the monitor, seven inches, I designed the GUI that is convenient to use two hands by placing sub menus at each side.

My main technical responsibility of this project is displaying weather information on the Beagle Board. To accomplish this goal, I started with researching on weather band IC to receive the weather band information. Silicon Lab's Si4707 IC is dedicated to receive weather band signal. However, Si4707 was in Non-Disclosure Agreement. Because of our time frame, the plan of using IC to receive the weather band signal was canceled. Instead of using IC, we decided to use Ethernet connection to retrieve weather information from the web. I changed the direction of the research to retrieve the weather information from the web. As Da implementing XML parser to parse the information, I
designed the GUI for displaying weather information and wrote the code to retrieve and display the weather information on the Beagle Board. Because displaying the weather information was one of the main goals, I put major efforts on putting more feathers to weather sub menu. I made Setting sub menu so that user set their zip code and preferred metric system, which can be shared by other sub menus. After we decided to use Ethernet, I came up with new idea that retrieve the Headline News from the web and display on the Beagle Board. The project is related to home automation, so I made sub menu for displaying current status and controlling each appliance of each room. For time issue, we could not make appliance to communicate with the Beagle Board. However, this feature will be the minimum requirements for future home automation system.

Meanwhile as a lab coordinator, I put great amounts of efforts on researching touch screen monitors for our Beagle Board. The Beagle Board Revision C2 supports raw LCD outputs while the Beagle Board revision B7, which our team have developed on, does not support raw LCD outputs. There were lots of difficulties for finding touch screen for the Beagle Board and Angstrom Linux that was sold in the market. At best, I was able to find the seven inches touch screen monitor supporting Linux, which has the same kernel version as Angstrom Linux. After selection and purchasing of the monitor, Da worked on installing driver for this monitor. With the help of TI, our team succeeded to make touch screen work on the Beagle Board.

Rituraj Behera:

The design process that our team had to go through for designing the Home Automation Control Panel was very informative and I learned a lot about how working experience is as a team in designing something new and completely from scratch, at least in terms of software. I was responsible for the conceptual design of the GUI, how the home page would look, the sub-menus for weather, lighting, climate control and security
layout would look like and also implement some of the code to make things work in the GUI like the digital clock and set it on the Main Panel. There were a few constraints in terms of the amount of space that I could work with for the conceptual design of the layout of all the menus and sub-menus. So, the buttons and icons had to be large enough and also fit in a certain framework. A lot of calculations had to be made especially for the sub-menus to make all the attributes fit perfectly in the given space. So, for the first half of the design process, I was mostly involved in conceptual design and writing java code for implementing some of the functionalities of some menus or buttons. In the second half of the project too I had to work mostly on java coding for the GUI and did not take much part in the hardware interfacing of the Control Panel. The work was distributed in a way that those who took interest in software did the GUI coding and one’s interested in hardware did the hardware interfacing with the Touch Screen and the Beagle Board. I also worked together with Samuel Flynn in deciding what I/O devices and other hardware devices to take into consideration to be purchased with the help of a matrix table to decide the most optimum hardware devices to go with so that the project is most cost efficient. In addition to my own responsibilities, I also helped Sam with organizing and optimizing the java code he had already written for the Main panel. This is because, even if the java code worked for the GUI, it was not always the most efficient and organized and so I had to monitor his written code to find any mistakes or optimizing required for a better Object Oriented (OO) Design.
Appendix 2 - Literature:

Web References:

- Java Swing Tutorial - Creating a GUI with JFC/Swing
  <http://java.sun.com/docs/books/tutorial/uiswing/>

- Weather information collection (XML) and other references related to the research
  - <http://www.weather.com/services/xmloap.html>

- Eagle software for PCB
  <http://www.cadsoftusa.com>

- USB socket help
  <http://en.wikipedia.org/wiki/USB>