aLife – Home Monitoring System

Amos Kittelson, Jacob Peery, Todd Denton, Tim Tewolde

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

Abstract - In this paper we present the design of the aLife home monitoring system. The main functionality of the system is use the Android operating platform to allow for the use of notifications to be sent to the user. They will be sent via a base station whenever the Zigbee wireless network replies that an event has been triggered. The notification will contain information about the event and possibly a set of actions that could be taken for it.

Index Terms --- home monitoring, wireless networks,

I. Introduction

In our hectic lives there exists a need: Simplify our lives, please! The aLife (Advanced Living Integration for Education) project will aim to help by taking information from in and around your home, run it through a filter and give your gray matter a break. The system will be smart enough to know when to give feedback to the user and when not too.

People's lives are increasingly complicated, especially in managing their homes. You have to remember the shopping list, when to take the kids to soccer practice, whether you locked the front door, if the garage door is closed, when the dog needs more water, to turn off the lights when you're not using them, turn the coffee pot off, keep the AC at the correct temp, and a multitude of other things that eventually lead to some degree of information overload and stress. While there are devices that can increase our standard of living and give us more ways to be lazy (clap on lights), what would be far more useful is a way to make managing our lives more efficient, with respect to time, electrical energy, and mental energy. A "one stop shop" system of monitoring and control of all of the devices (and even some non-electronics) in your home that only presents information to you when pertinent, allow you to set automatic settings for electronics such as lights, and even monitoring power consumption of electronics and disconnect them remotely, saving you money and stress when the electric bill comes. It's like autopilot for your home so you can focus more on living.

For our project, we will develop a prototype system with the potential to do all these things. It will consist of 3 major parts: An in home base station, wireless appliance control modules, and a remote user interface device such as a cell phone. The base station will act as a repository for all of the information about the status of devices in your house. It will pass information back in forth between the user interface and the modules that actually control your home appliances via Ethernet, ZigBee, and USB. The remote modules are designed to be simple, cost effective devices that monitor and control household appliances in a non-intrusive way. They are designed to be as flexible as possible so they can be interfaced with a wide variety of common household items. They will each have a ZigBee transceiver for communicating with the base station and basic I/O hardware to perform monitoring and control functions. The user interfaces will be a wall mounted touch screen LCD that is hard wired to the base station, as well as applications that run on any any Android equipped cell phone to allow remote control connection with the base station over the Internet. The user interfaces will present the user with intuitive, concise menus that display information about items in your home and allow the user to change the operation of items in their home. Although there may be several items in your home that are connected, information about them will be integrated and displayed in a central Android app in order to keep the information organized.

II. Base Station Hardware

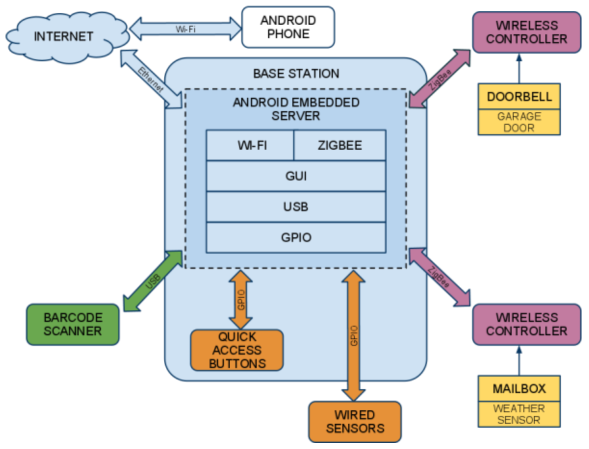


Fig 1. Functional Diagram of the Base Station.

The Base Station is designed to be a wall mounted unit that is always active and able to respond to requests from both the remote client and the ZigBee network. It will also have a built in GUI that will allow the user some access for changing and adding devices to the ZigBee network from the Base Station. In processing requests from the ZigBee network side of the system, it will provide a way to determine upon receiving a signal from what device it came from and to alert the user of this by sending a notification to the remote device with this information. From the remote device it will be able to receive commands on the software side of the Base Station and then to transfer those commands into instructions for the specified ZigBee network device to decode and follow. The hardware will allow for multiple connections to be made for remote devices and for multiple sensors to be set up on the ZigBee Network. It will also have the means of communicating through the Internet and updating the database about notifications being received, transmitted, or handled according to what the notification was sent about.

The Base Station hardware is composed of three main components. Our microcontrollers, the NXP LPC3250, will serve as the main controller for our system. Some of the attributes of the microcontroller are listed below.[1]

Processor	**ARM926EJ-S Core @266 Mhz**
RAM	256 KB
Serial Connections	7xUART, 2xI2C, 2xSPI
Other Features	Ethernet driver USB 2.0 support

With the microcontroller defined, we can then move on to integrating that with the rest of the board. The next hardware section includes the microcontroller placed into the LPC3250 OEM Board. This allows us to interface with the final base board that we will be using, and that will be covered later in the paper. The following picture is microprocessor mounted on the OEM Board[3]

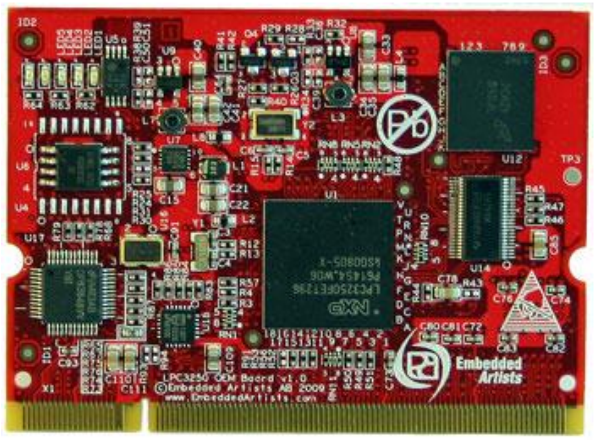


Fig 2. ARM9 processor on the OEM board

The next component is the actual base board that will make up the base station. The board we are using is the QVGA Base Board. Some of the features that it offers are listed below.

Power	**Either 3.3 (USB) or 9-15**
Interfaces	USB, RS232
Connectors	Ethernet, MMC/SD
Other Features	USB-to-Serial Bridge 240x150mm size

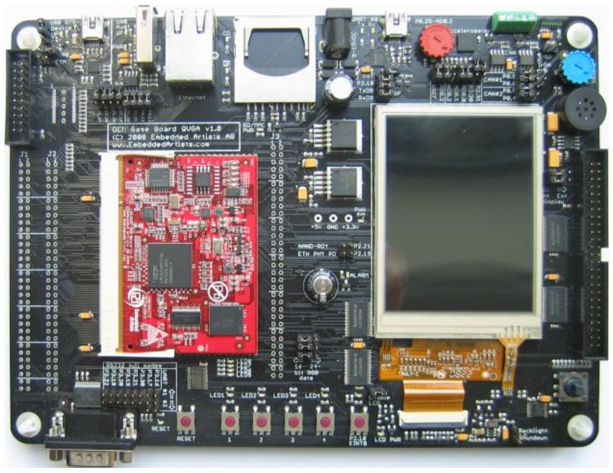


Fig 3. OEM board mounted in the QVGA base board

The last key feature of our system is the touchscreen itself. We are using the QVGA TFT LCD screen that is included with the base station that we are using. The main reason for this is to allow the user some direct access to the system without having to user the serial or USB inputs. This will help when registering devices or interacting with the system without the use of a remote client

The Board Support Package for the base station included both the Linux kernel and the Android operating system. The version of Android that is running on the board is 1.5 (Cupcake).

The reason that we went with the baseboard for this project is that it meets the requirements for running Android. All Android versions require the same hardware specifications as 1.5, so this is useful for us because we know that this board meets minimum system requirements.[2]

III. Base Station Software

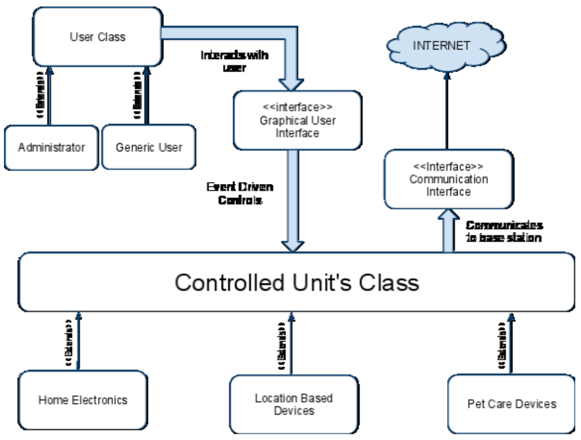


Fig 4. Functionality of the Base Station software. It shows how all of the different classes interact throughout the system.

The Base Station Software is intended to be an "always on" process that works with little to no user intervention. The system is responsible for responding to service requests sent from remote client devices in a reliable and time efficient manner. The software will also have the responsibility of periodically polling certain controlled devices and sending their information to its specified table in our database. The software must also provide the user a basic security mechanism, basic log in and log out functionality, which will allow access into our system. The only feature that will be enabled to the user through the base station software will be the ability to add a device to the ZigBee network. Any additional services provided to the user will be done by the remote client device. The Base Station must also be able to accept full duplex TCP socket connection requests from one, or up to five users, with a high degree of reliability. After connecting with a remote client device the system must be able to handle any service request sent from the said device on a first come, first serve basis. Any duplicate request, simultaneous request, or unsuccessful requests must be handled accordingly and a notification must be sent to said remote devices if the issue was not resolved. The Base Station software is also responsible for sending notifications to one, or all, remote client devices. These notifications are based on a user's specific notification settings.

The idea behind the model was to make the base station software as similar as possible to the remote client software. It must also be able to accept up to 5 simultaneous service requests on a first come, first serve basis. Other functional requirements we set up for the software were that multiple requests for the same notification must be serviced only once, and be able to store information on an embedded SQLite database. We also wanted to add some type of security to the system by adding limited number of login attempts, registering wireless devices before allowing them to send and receive information, and using a security scheme over the ZigBee network itself.

For developing the software we used many different platforms. Eclipse with the Android Development Tools (ADT) plug-in was used to program the software in, and a SQLite database was implemented in java to store any information that might need to be polled from the wireless devices.

The following Figures shows the actions that our system will take when dealing with the system. The first Figure shows the steps that the new user will undergo to register for the system. The second picture is that will happen when a new device is detected and added to the network for later use. The final picture is the fulfill service requests, and it shows how requests will be processed from the remote client to the base station. These three pictures represent the base functionality for our project and how it interacts with the users and clients.

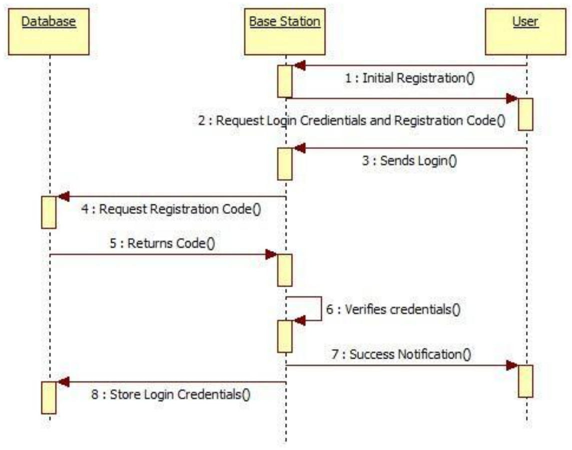

Database
Base Station
User
1 : Initial Registration()
2 : Request Login Credientials and Registration Code()
3 : Sends Login()
4 : Request Registration Code()
5 : Returns Code()
6 : Verifies credentials()
7 : Success Notification()
8 : Store Login Credentials()

Fig 5. New User Registration

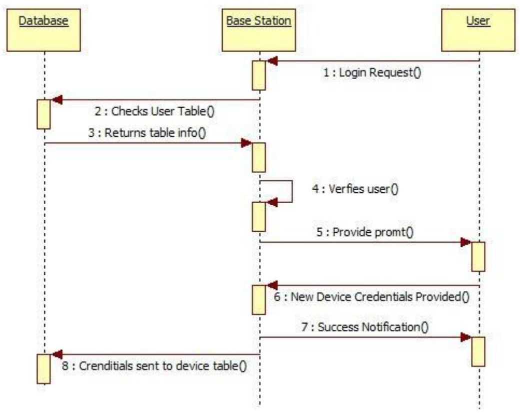


Fig 6. Add a New Device

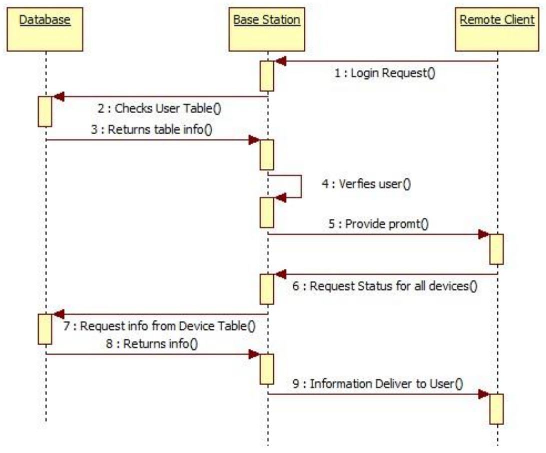


Fig 7. Fulfill service request.

The main functionality of the software can be divided into two categories: user class and device class. The user class is going to be composed of the GUI, the interaction between users, and fulfilling any autonomous service requests required of the base station. The device class, as seen below in the multiple class types of devices that the base station will service, are specific to devices and are made to incorporate the type of data we plan on receiving from them. The following lists the types of methods that we are seen with the corresponding devices.

A. Power Device

- PowerDevice(): Constructor to create object. Will have generic and specific constructors.
- turnOn(): Takes in no parameters. Sends message to ZigBee device to turn on
- turnOff(): Takes in no parameters. Sends message to ZigBee device to turn on
- getTemp(): Takes no parameters. Polls ZigBee device for temperature.
- setTemp(Int Temp): Sets ZigBee device with provided temperature. Does not return anything.
- getWatt(): Takes no parameters. Polls ZigBee device for wattage.
- getTimeframe(): Takes in no parameters. Returns the timeframe field.
- setTimeframe(Int Timeframe): Sets the timeframe field. Does not return anything.

B. Security Device

- SecurityDevice(): Constructor to create object. Will have generic and specific constructors.
- turnOn(): Takes in no parameters. Sends message to ZigBee device to turn on
- turnOff(): Takes in no parameters. Sends message to ZigBee device to turn on
- isOpen(): Takes in no parameters. Returns true if ZigBee device is open.
- IsLocked(): Takes in no parameters. Returns true if ZigBee device is locked.

C. Control Device

- turnOn(): Takes in no parameters. Sends message to ZigBee device to turn on

- turnOff(): Takes in no parameters. Sends message to ZigBee device to turn on

The last component of the base station software is the database that will be used to store information about current users, devices, power history, notification history, and notification set ups. Each of these elements will have their own table in our database with their own specified fields.

IV. Wireless Network

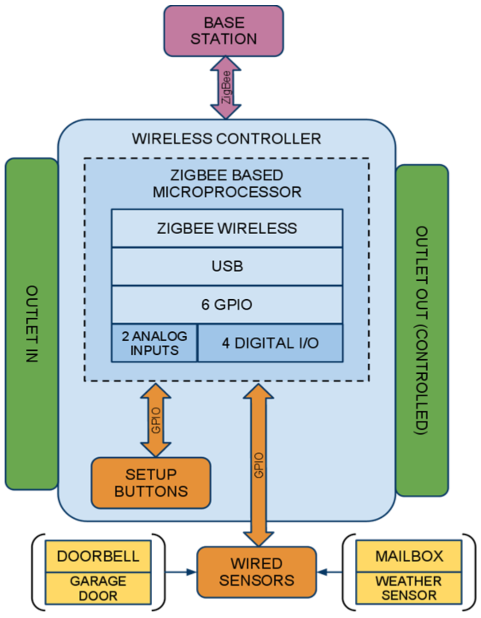


Fig 8. Functionality of our Wireless Modules

The purpose for all of the modules is to serve as a sensor and/or control device for the main system. They are the entry point for all home status information, and the end points for control commands. For instance, these devices will monitor the actual current being consumed by an appliance, physically connect or disconnect an appliance, or have a wired connection to a sensor. The native aLife modules and 3rd party modules will operate with the same wireless protocol, so the interface with the base station is identical. Example 3rd party devices are ZigBee thermostats, window and door sensors, and electronic door locks. This allows a high degree of flexibility and expandability in the overall system, allowing the user to select any compliant ZigBee device and add it to the system for their specific application.

In operation, the wireless modules would essentially act as slave devices to the master base station. Configured as a sensor, the module would continuously monitor the sensor data and report the data back to the base station when polled (ex for power monitoring), or if there's an interrupt even that requires immediate notification to the base station (ex. window or door open sensor). If configured as a control (output) device, they would wait for instructions from the base station and maintain a constant output state until the base station commands them to do otherwise.

This section outlines the hardware requirements of all of our ZigBee remote modules. The following lists cover some of the hardware components that are going to be included with the MCU part of the wireless sense and control modules.

MCU
Low voltage MCU with 40 MHz HCS08 CPU core
Dedicated SPI for Zigbee interaction
2 SCI
60K Flash and 4K Ram

Power Supply
120 VAC to 9VAC transform
9VAC full wave bridge rect. to create 7VDC rail
7VDC input into 300mA 550kHZ switching power supply

The wireless modules will also contain a ZigBee chip that will serve as our wireless home network. The specifications for the chip can be found in the list below.

ZigBee Transceiver
802.15.4 compliant
Operates on 2.4 GHz ISM band
<-92 dBm receive sensitivity
Three low power modes
Supports packet and streaming data
Programmable clock frequency
Seven GPIO for use by the user

V. Remote Client

Fig. 9 Picture of our remote client

The following table lists out some of the hardware specifications for the remote client.[4]

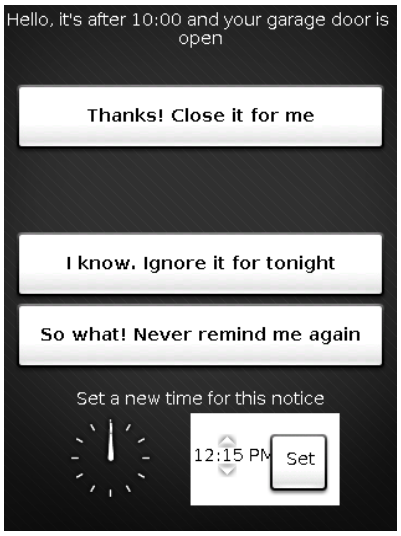
Processor	**Qualcomm® MSM7201A™, 528 MHz**
Platform	Android™
Memory	ROM: 256 MB RAM: 92 MB
Display	3.2-inch TFT-LCD flat touch-sensitive screen with 320 x 480 (HVGA) resolution
Network	HSPA/WCDMA: 2100 MHz Up to 7.2 Mbps down-link (HSDPA) and 2 Mbps up-link (HSUPA) speeds Quad-band GSM/GPRS/EDGE: 850/900/1800/1900 MHz (Band frequency, HSUPA availability, and data speed are operator dependent.)
Device Control	Trackball with Enter button
Keyboard	Slide-out 5-row QWERTY keyboard
GPS	GPS navigation capability with Google Maps™
Connectivity	Bluetooth® 2.0 with Enhanced Data Rate Wi-Fi®: IEEE 802.11b/g HTC ExtUSB™ (11-pin mini-USB 2.0 and audio jack in one)

The aLife remote client software will be designed to give the user an intuitive, responsive and fluid interface to the aLife system. The interface design will mix text, icons, scrollable lists and gestures for the user to interact with. The user interface look similar across platforms. The iPhone remote client software should have the same look and feel of an Android version of the remote client software. Since the scope of this project is focusing on Android, the following user interface specifications will focus on Android. Remote client software build on other platforms in the future will use the Android application as a design standard.

The aLife remote client software will display an icon in the top left of the menu bar to indicate that there is an notice for the user to review. The user will pull the menu bar down to review what the event is related to. Selecting the notification will take the user to the aLife remote client application where the user will be presented with more detailed information and options. Examples of notifications are energy usage, temperature issues, security and lighting. The aLife notification icon will be supplemented with overlay graphics that indicate what type of notification is waiting for the users review. If more than one notification is pending then the aLife icon will be over-layed with a plus symbol to indicate multiple notifications. The icon will have different graphic overlays to indicate security, energy and temperature notifications.

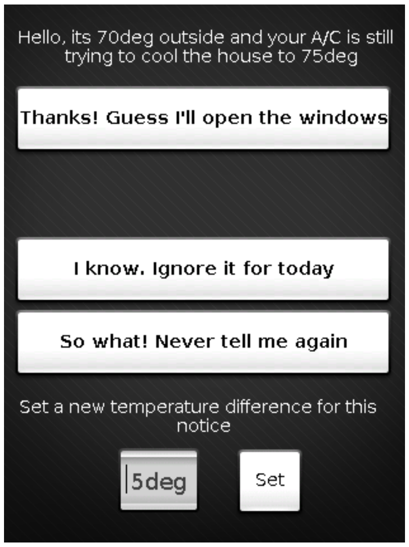
For example, if the aLife system is set to make sure the house is secure at 10:00pm and the garage door is open then the user will receive a security notification in the top left the Android user interface. And audible tone may also sound as described in the notifications subsection of the research section of this document. When the user pulls down the menu bar the notification window will indicate a security issue with the garage door. When the user selects the notification the user will be presented with a screen similar to the one in Illustration 39. The aLife remote client software will tell the user that the garage door is open and that the reason the notification was presented is because it is set to do so if the time is 10:00pm and the garage door is open. The user will have the option to close the garage door, set the system to ignore the garage door for the rest of the night or never remind the user in future.

The remote client will have its own interface to the aLife system. Below are a few examples of what notifications would look like when they are received by the system. The first is an alert about the garage door being open and options about how to fix it. The second is an alert about controlling the temperature in your house.


Hello, it's after 10:00 and your garage door is open

Thanks! Close it for me
I know. Ignore it for tonight
So what! Never remind me again
Set a new time for this notice
12:15 PM
Set

Fig. 10 Picture of a garage door event as a notification


Hello, its 70deg outside and your A/C is still trying to cool the house to 75deg

Thanks! Guess I'll open the windows
I know. Ignore it for today
So what! Never tell me again
Set a new temperature difference for this notice
5deg
Set

Fig. 11 Picture of a thermostat triggered event to the system

Selecting a device will display a basic graph showing the device history. History length will be determined at testing. Devices that have an energy history will show their energy graph. Devices that have on/off states will show their on/off history. Below the history will be buttons related to individual devices. Button behavior will be based on the control-ability of the device and its options.

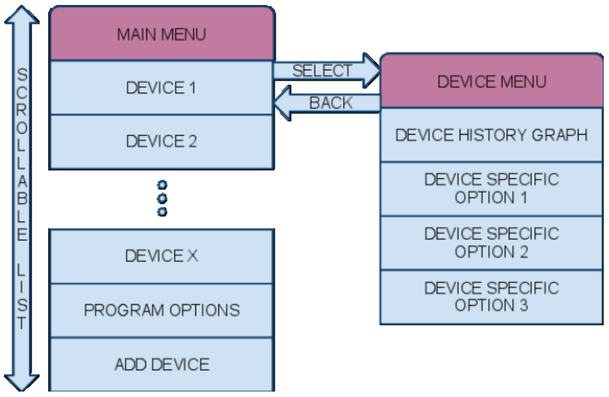


Fig. 12 Device menu for the selected device

The program options page (Figure 13) will give the user general the ability to control whether or not to maintain a constant connection with the server or periodically check the server for updates to save battery life. The user will also be able to mute just the aLife audio feedback, control if the service starts with the phone, and input user name data.

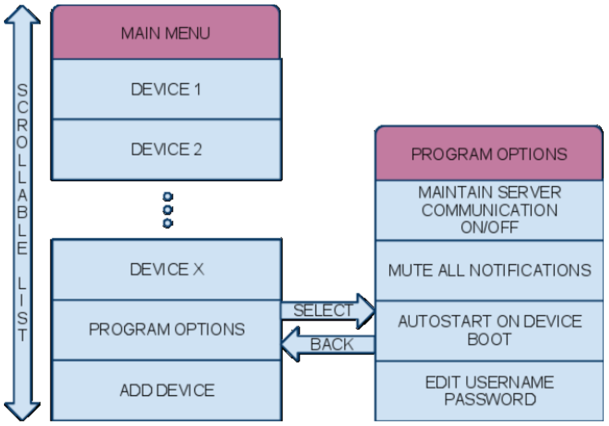


Fig 13. Program options for the remote client

VI. Conclusion

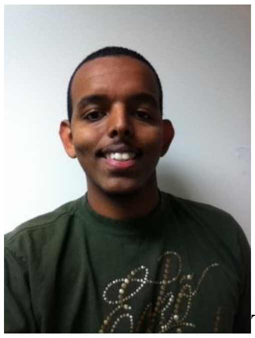
With all of the information listed, we now have a general overview of what the aLife is capable of doing. From the bottom level of the Zigbee protocol to the higher level of notification passing and internet communication, we have detailed our senior design project.

VII. The Engineers

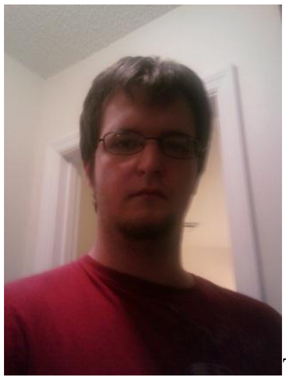
Amos Kittelson is currently a senior computer engineer at the University of Central Florida. After Senior Design, he plans on moving to Europe to continue his education and to finish up graduation

Jacob Peery is currently a senior electrical and computer engineering student at the University of Central Florida. After graduation he plans on trying to get a job at his current company, Advantor.

Tim Tewolde

Tim Tewolde is currently a senior computer engineer at the University of Central Florida. After graduation, he plans on looking for a job in the software development field and hopes to be working on android in the future as well.

Todd Denton

Todd Denton is currently a senior computer engineering student at the University of Central Florida. After graduation he plans on applying to University of Central Florida again to continue with a master's degree in Artificial Intelligence.

Acknowledgments

We would like to thank Dr. Richie for taking the time and helping us with our project, and overall keeping us motivated to do the best we could on it.

We would also like to thank Jacob's company Advantor for lending us the boards that we used in our project and providing some support for troubleshooting issues that we had.

Lastly we would like to thank Freescale for offering many reference guides online for their products and a good support line for help most any time of the day.

References

[1] http://ics.nxp.com/products/lpc3000/lpc32x0/

[2] http://www.talkandroid.com/android-forums/android-hardware/2-android-minimum-hardware-requirements.html

[3] http://ics.nxp.com/products/lpc3000/datasheet/lpc3220.lpc3230.lpc3240.lpc3250.pdf

[4] http://www.htc.com/www/product/g1/specification.html