Critical Design Review (CDR)

Project Description

This project will design and implement a Smarthome lighting and ceiling fan control system. Controls will allow light fixtures to vary their brightness in a dimmable fashion, and ceiling fans driven by single-phase AC induction motors will be speed controllable.

Control messages will be sent wirelessly. The wireless communication will be using the Zigbee Mesh protocol. Xbee Series 2 modules will be used, in API mode and with built-in encryption capability enabled.

Wall “switch” controls will be small LCD touchscreens, communicating with the items they control via wireless command messaging. The LCD will allow the user to select a room, and then which target item in that room to view current status of, or to change it's state. (ie. The selected item in the selected room will get brighter, dimmer, slower, faster, etc.)

The target items being controlled will use a microcontroller as the Zigbee command receiver. Dimmable light fixtures or speed-controllable motor-driven target items will use a common Triac-based circuit to control the target as desired by the user.

A central computer server will log events, control “vacation mode” away events, and act as a gateway for automatic and manual controls sent from distant control devices carried by the home residents. This server will also act as the Zigbee wireless network “coordinator” node, in charge of the Zigbee mesh network. This server will also have an LCD touchscreen panel for

Android smartphones will offer both automatic and manual controls sent via the Linux server, and also a way to remotely view the server's log data in text form. Current status of the home targets will be visible in a GUI which mimics that seen on the LCD touchscreen wall “switches”. The mobile devices will primarily use a GPS based geofence to indicate whether the user is home or away. An alternative control method will be with the use of NFC tags at the front door of the home, so the user can manually swipe his or her mobile device across a tag to manually indicate that they are leaving or returning home. These two control capabilities allow the server to automatically enter and leave an automated, quasi-random “vacation mode”, where it will turn lights on and off at reasonable times to emulate the residents being home for added security from burglars.

This project will also be designed to allow future expansion, with possibility to add new kinds of target items to control and new types of controls. The intention is to complete a project comparable to popular X-10 systems, but which will send control commands wirelessly instead of signaling over power line. Some example ideas for future additions are garage doors, electronic entry door locks, thermostat HVAC environment control, or safety shutoff for electrics ovens or lab soldering stations, in case you forget to turn them off when you leave home, but these examples are not included in this semester's project effort. Additionally, other mobile platforms, such as iPhones or Windows phones, can be added in the future, but are not part of this project effort.

The major motivation to choose a wireless signal messaging system is to allow some effective changes to the home wiring, without needing to cut into walls or ceilings, etc. to reroute Romex 120V cables. For example, an incorrectly done set of three switches can be replaced so that the three switches do all work on the same load together as intended. (A correct classic 3-switch network needs a 3-way switch on each end, and a 4-way switch between them. This may have been done wrong such as to use a simple on off switch on one end, which keeps everything off unless it is on.) This works well where switches are already present, with Romex 120V AC cabling present to provide power to the new smarthome switch units. This system would also allow to "move" the switch responsible for a particular load to a different switch location, such as to allow a switch near the front door to control a light, which is currently controlled only by a switch on the opposite side of the entry room. This system would also allow additional switches in different locations to work in addition to a load's current control location, as a "companion" rather than a true "replacement" of the original.

While the ideal would be for the wall “switch” controls to fit into a standard Decora ™ wall-plate and housing gang area, and target controls would fit into common ceiling fan wiring canopies or other light fixture wiring boxes, this project during the semester will be made using prototyping computing and interface boards, which will not cleanly fit such common switch housing, and so will not implement a cosmetic ideal. To cosmetically fit into the popular housings would require custom PCB design and assembly to compact the various components used, as well as alternate choice of LCD touch panel of proper size, which may not be supported by the intended Arduino code libraries available, and require software porting to make use of. These additional tasks are beyond the scope of this semester's effort to create a working prototype.

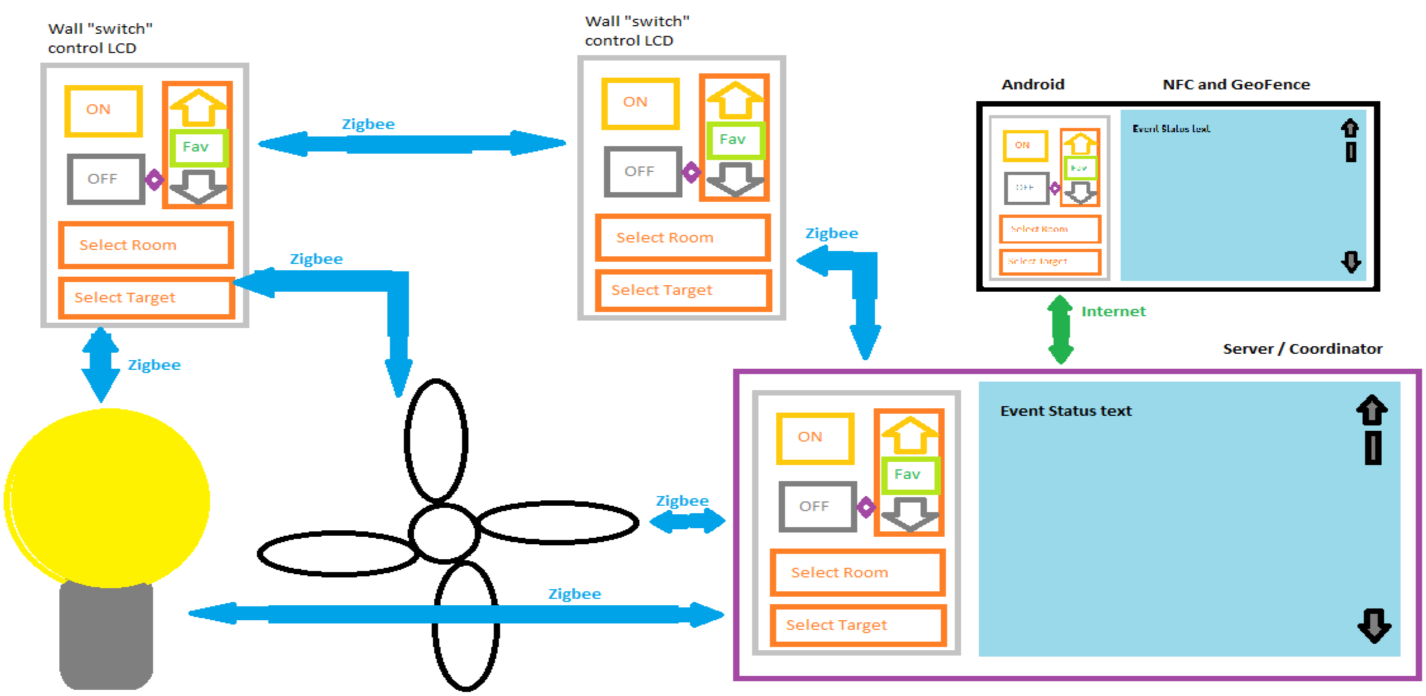
This project evolved from an initial concept of a wireless controlled light switch, and complexity was added to bring that up to an acceptably complex project for this course.

This project is an individual effort, and does not depend on other projects, nor do other projects depend on this.

Functional Description

Diagram 1 shows the three types of user control units: 1) The wall-mounted LCD touchscreen controls, located in the building as any other type of light switch would be, and likely replacing older model switches as this system is installed, 2) The central Linux server and coordinator panel, which could be wall mounted or in an enclosure resting on a desk or table, and 3) The Android mobile device App. The possible communication participants are indicated with blue arrows to show a Zigbee connection, or green arrows to show an Internet connection.

Diagram 1:



At boot time, any Arduino based node (either Uno or Due), will first check if a hardware push-button is asserted, to indicate that the user wants to use a PC on the board's USB port to configure the Xbee module, using the XCTU utility software. If asserted, then the Arduino board will enter a non-active microcontroller state to avoid contention on the USB pins. Otherwise, the board will continue to boot into operational mode.

An Arduino target node (controlling some load such as a light fixture or a ceiling fan, and potentially more than one load) will look to stored values in EEPROM or Flash to determine the last running intensity value for the load. Each load will then be set to be at that last saved intensity value. This will allow a quick recovery from a power outage, without a delay of waiting for the Linux server to boot, all the other Arduino nodes to boot, and receive, load ID by load ID, setting commands from the server based on the most recent settings found in the event log.

Once the last saved intensity value is restored to each load, then the target node microcontroller will begin listening for command messages from the Zigbee network.

A wall control Arduino node will proceed from the Xbee config check to gather information about controllable load units, which target nodes they are controlled by, room locations of loads, and what the “default” load ID and location are assigned to it.

Then, the wall control will begin polling the load IDs to learn their current intensity level or other status value.

The wall control then updates the LCD display to indicate the default load ID, location, and current intensity setting.

After the LCD is displaying the current situation for it's default load, the wall control unit will then begin listening for commands from the Zigbee network as well as from the touchscreen and any debug controls.

When a manual, debug or Zigbee control is received, any Zigbee node will carry out the messaging conversation as defined for that command type, and when completed will return to a state of waiting for another control input or message.

The Linux server will boot up, prepare Zigbee networking, prepare Internet networking, open the even log file for appending, read last known status for each node from event log, check what other nodes are expected to exist and check they are all booted and available with a Read Intensity command via Zigbee, note any nodes not responding, log any updated current values from the target nodes if not matching the last known value in the event log, determine the default load to display on the LCD and to control, configure the LCD display to show the now current intensity level for the default load ID, read and then display the event log text data. Then begin listening for Zigbee commands, listening for commands from the Internet, as well as waiting for touchscreen and remote Android inputs. On GPS geofence events provided by the Android phones, the server will update a server panel status indicator, showing who is home and who is away.

The Android app will determine the related default load ID, prepare Internet networking, and attempt to connect to the Linux server via Internet.

If not successful in connecting to Linux Server, then draw to LCD an indication that connection is not available. Continue attempting connection periodically.

When successfully connected to Linux server, then get the now current states for target nodes, get the current event log content, draw the control panel indicating the current level for the default load as well as the default portion of the event log text data. Status will be updated periodically, not continuously. Then wait for Android touchscreen input from user, and also periodically update node states and event log data from server. A GUI status indicator will show if the app has a connection to the server. The app will also periodically check GPS location, determine if the current location is at home or away, and update the server with that information if a change is detected.

Protocol

This system will use a Zigbee wireless method to communicate with other nodes. The message content will be encrypted by the Xbee module firmware, so our software will not need to include an encryption library.

The communication protocol will be a command made in 4 steps:

1. Sender sends command to recipient
2. Recipient Acknowledges command received, asks sender for confirmation
3. Sender sends confirmation
4. Recipient indicates completion success or failure

Diagram 2 shows the Arduino and shield configuration for an LCD touchscreen wall control node.

Diagram 2:

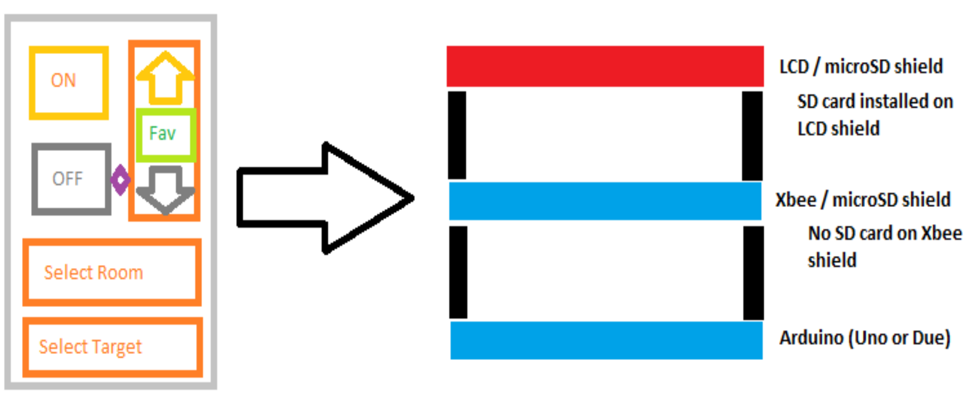
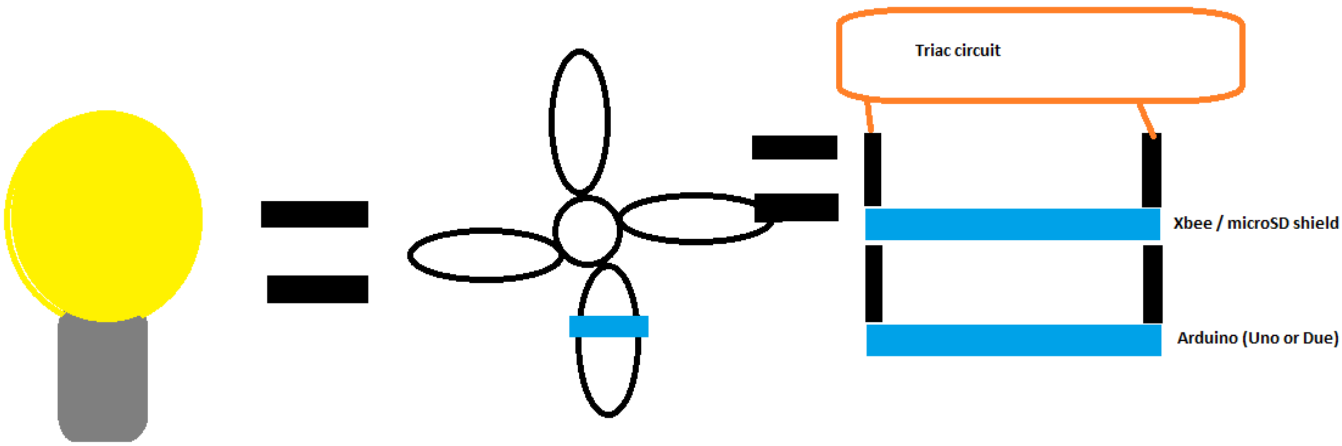


Diagram 3 shows the Arduino and shield configuration for a target node. This is identical for a light controlling node, a ceiling fan controlling node, and possibly some other node types which could be added in the future. One target node can actually control two, or possibly a few more than that, loads, so that one unit can control both a light and a fan.

Diagram 3:



Interface Description

Electrical Connections

The Arduino board has 5V IO signaling, while the Xbee module, SD card, and LCD panel are natively 3.3V components. The Arduino shields include voltage shifting from Uno 5V outputs to the 3.3V inputs of these other components. In the reverse direction, no shifting is provided, but 3.3V outputs from these devices is sufficient to work with the threshold voltage of the Uno 5V inputs. This allows signaling between the devices and Arduino Uno to work properly and safely.

The Arduino Due board has 3.3V IO signaling. The same shields will work with this board, as the Due's 3.3V outputs are sufficient to drive the 5V input side of the level-shifters on the shield boards. These shifters then output 3.3V signal into the other components. These other components then have their 3.3V outputs connected to the 3.3V inputs on the Due, and all signaling works properly in this configuration as well.

The relay portion of the AC control circuit will be a Triac device capable of working in a 120V AC, 60Hz environment. One could potentially select a Triac capable of working at 240V, for other country standards, but I will focus on the United States standard of 120V at 60Hz.

Arduino Uno boards are planned to be used as the target control nodes, so they will be connected to and in control of the triac AC relay circuit. An opto-isolated Diac will be used to electrically isolate the AC side of the relay circuit from the Arduino, xbee and SD card components. The opto-Diac will be selected to be controlled by a 5V digital input signal from the Arduino Uno. Should an Arduino Due board be used as a target control node, then if the opto-Diac also works with 3.3V input signal, then no change to the Diac would be required. If 3.3V would not work with the initially selected 5V input opto-Diac, then the Diac part would need changed to a 3.3V input compatible part. Ideally, an opto-Diac compatible with both voltage levels would be selected, or a level shifter used in the input path to allow either voltage to control the Diac. The Diac must work with 120V on the AC side.

Similarly, the AC Zero-cross detection circuit, which provides an input into the target Arduino microcontroller, should provide a 3.3V and 5V compatible signal, so that either model of Arduino board can be used safely and reliably.

The AMD Gizmo-2 board will be used in conjunction with the Explorer board add-on. This add-on. offers a UART connection, but I have not yet found documentation about this, such as voltage level or connection location. If this is found to be a 5V signal from Gizmo, then level shifting will be required to the 3.3V Xbee module. If the port is 3.3V signaling, then no shifting is required. For convenience, I would like to make use of an Arduino Xbee shield, which already includes level shifting to work with both signal voltages, connecting via a breadboard and a small 5V/3.3V level shifter for Xbee.. Optionally, a USB serial adapter for Xbee may be used in one of the Gizmo's USB ports.

There are no electrical concerns for the Android mobile devices involved, as they will not be electrically modified for this project. They are used as-is from vendor.

Diagram 4 shows the pin mapping of the two types of Arduino microcontroller boards and the different “shield” boards I plan to connect to the Arduinos. This diagram shows that the LCD touchscreen wall “switch” nodes have a conflict on pin 4 between the LCD shield and the Wireless/SD shield. This is not a significant problem, as for these nodes, I will instead use the SD slot located on the LCD shield to avoid any conflicts. Target nodes will not include the LCD shield, and so there is no conflict on pin 4.

Diagram 4:

Arduino pinouts

Triac/Misc	Wireless/SD	LCD/SD	Due	Uno R3	pin ID		pin ID	Uno R3	Due	LCD/SD	Wireless/SD	Triac/Misc
3.3V	3.3V	3.3V	3.3V	5V				5V	3.3V	3.3V	3.3V	3.3V
							17					
							16					
							AREF / 15					
							GND / 14					
					NC		13			SD_SCK	SD	
					IOREF		12			SD_DO	SD	
					Reset		11			SD_DI	SD	
	3V3	3V3			3V3		10			SD_SS (CS?)		PWM
	5V	s			5V		9			LCD_D1		PWM Fan
					GND		8			LCD_D0		
	GND	GND			GND							
					Vin		7			LCD_D7		
							6			LCD_D6		PWM Light
btn On/Off		LCD_RD			A0		5			LCD_D5		PWM
btn Up		LCD_WR			A1		4			LCD_D4	SD_CS	
btn Down		LCD_RS			A2		3			LCD_D3		PWM
Xbee config status		LCD_CS			A3		2			LCD_D2		btn fn/lt sel
AC 0-cross		LCD_RST			A4		1				TX	
Xbee config					A5		0				RX	
					A6							
					A7		14					
							15					
					A8		16					
					A9		17					
					A10		18					
					A11		19					
					DAC0		20					
					DAC1		21					
					CANRX							
					CANTX							

Yellow =	Applies to both the LCD touchscreen wall control and the target receiver
Pink =	Apples to target receiver ONLY
Purple =	Applies to LCD touchscreen wall control ONLY

Table 1 describes the additionally required signals in and out of the Arduino units, which are required to interface to the 120V AC loads and to debug push-buttons and LED indicators.

Table 1:

Description of Misc and Triac Signals	
Signal Name	Description
Xbee config (also btn fn/lt sel??) Input 3.3V	Active-Low – When asserted at boot time, Arduino goes into a dummy loop or does an exit, so that USB can be used to the Wireless/SD shield's Xbee module for configuration using XCTU software on a PC. Otherwise has no function. (might optimize to share this input with btn fn/lt sel, which is only effective in run-time)
Xbee config status Output 3.3Vor5V	When lit, indicates that the Arduino board is in an inactive state so that a PC computer can communicate with Xbee module over USB for configuration, using XCTU utility software.
btn fn/lt sel Input 3.3V	When High (default by pullup), tells Arduino to apply any other manual button inputs to the Light control. When Low, tells Arduino to apply any other manual button inputs to the Fan control. (Also used on wall control node development to test communication and control of target, before adding LCD touch shield support. Once LCD shield is added, this button must be removed from wall control.)
btn On/Off Input 3.3V	Active-Low – When asserted, change the target power applied state. Like a “bang-bang” power enable control. (Also used on wall control node development to test communication and control of target, before adding LCD touch shield support. Once LCD shield is added, this button must be removed from wall control.)
btn Up Input 3.3V	Active-Low – When asserted, increase intensity (brighter/faster) to target. (Also used on wall control node development to test communication and control of target, before adding LCD touch shield support. Once LCD shield is added, this button must be removed from wall control.)
btn Down Input 3.3V	Active-Low – When asserted, decrease (dimmer/slower) to target. (Also used on wall control node development to test communication and control of target, before adding LCD touch shield support. Once LCD shield is added, this button must be removed from wall control.)

PWM Light Output 3.3V/5V	Pulse-Width Modulated control of triac circuit for the Light target
PWM FAN Output 3.3V/5V	Pulse-Width Modulated control of triac circuit for the Fan target
AC 0-cross Input 3.3V	Active-High - Input pulse to indicate the 120V AC line waveform has crossed the 0V line, and a new waveform period begins.

Zigbee Message Format

Each Zigbee message will have a twelve byte logical payload, as described below. The “physical” payload, in the Zigbee total frame, may be a few bytes larger, as I understand that the Xbee firmware encryption feature requires this to be done.

Each node will have one or more 16bit ID values, which will be the first two bytes in the Zigbee message payload. The first byte in the payload is the ID MSByte, and the second byte is the ID LSByte. Each node can potentially have more than one target load ID, such as a node that controls both a light and a fan motor. One load ID will relate to the light, and another load ID will relate to the fan. So there are multiple kinds of IDs that will be included in the total Zigbee frame, the Zigbee address ID, which is not part of the payload, and the target load ID, which is part of the payload. While a 16bit value may sound excessive to ID a particular load, consider that a commercial building could potentially consume more than the 256 ID values, if I had chosen a single byte ID. An individual home would not likely consume 256 IDs, even if every outlet, light switch and light bulb in the house was uniquely addressable. But I'd like this to have potential beyond the home market, and make it appealing to businesses as well.

The third and fourth bytes will be the MSByte and the LSByte, respectively, of the node sending the command. This will tell the recipient where to send response stages of a full command conversation.

The fifth byte in the payload indicates the current message type within a single multi-message communication. Such as this indicates the initial command request, or the acknowledgment/confirmation request, or the confirmation, or the command completion and status notice.

The sixth byte in the Zigbee message payload will be a command code. There will be several commands that can be made to the recipient. They will be encoded as a byte value for convenience in the case of Arduino Uno, which is an 8bit microcontroller with small memory resources. To help keep the stack and code compact, I will not choose to subdivide a byte into multiple different fields and then add code to mask and shift these fields around. This command code should remain fixed through all message stages of a single command communication.

Command	8bit Encoding	Description
No Operation (nop)	0×00	Tell recipient to acknowledge and complete this communication, but otherwise do nothing.
On	0x01	Tell target to power on the load
Off	0x02	Tell target to power off the load
Increase Intensity	0x03	Set load to be one step brighter/faster
Decrease Intensity	0x04	Make light one step dimmer/slower
Set to Favorite level	0x05	Set target load to stored favorite level
Store current level as Favorite	0x06	Save current intensity level as Favorite
Read Favorite level	0x07	Ask target to say what the stored Favorite intensity level is
Read current level	0x08	Ask target to say what the current intensity level is
Event Notice	0xFF	Inform the Linux server that a command event has been completed, and what the new status is, for log purposes. Inform the server what the command was, what the target load ID was, and the status value.

The seventh and eighth bytes will be the MSByte and LSByte, respectively, for any message passing a status value, a Favorite or current level read response, or event update notice.

The ninth byte will be the status value or the intensity level value asked for.

The tenth byte will be a CRC check on the first seven bytes of the payload.

The eleventh and twelfth bytes are reserved for future use.

Graphical User Interface

The most commonly used user interface will be a Graphical User Interface on a small LCD touchscreen, mounted about the home (or business) in wall-switch wall-plate locations.

The LCD will display

- A Graphical User Interface to select the room and target device location, view current target state (as the target may be located in another room and not directly visible)
- Control buttons to increase or decrease target brightness or speed, full on or off, or some favorite intensity level for that target. An indicator will move up and down in a small track to indicate the currently set intensity.

The Linux Server and the Android app will display the same control panel as described in Diagram 1 for the wall controls, with the addition of visibility of the event logging data as well. The Linux server and android app will also display whether mobile device owners are home or away. This keeps load controls very consistent and familiar, so that the user only needs to learn to use one control interface arrangement. It would be confusing and irritating if each control type looked significantly different from the others. The only variance will be the omission of event log visibility and home/away status on the small wall switch controls.

Material/Resource Requirements

There are no dependencies from other projects. This project does not provide any dependencies to other projects. It is a completely independent effort.

This project is an individual effort. So there is no particular assignment of certain tasks between multiple team members, all project tasks are assigned to this single individual, Bill Toner.

There are three major components of the smarthome system. There are also some general household items, such as light fixtures, ceiling fan units, wiring boxes and power cabling which will be required.

Both the controls and the targets in the Arduino class will require an Arduino microcontroller board and the Wireless/SD shield, which holds an Xbee module and possibly a microSD card. The controls will also require an LCD touchscreen/SD shield, and the targets will also require the triac based AC line control circuitry. The target end will also require a couple lights and a ceiling fan to control. An Xbee module will be required for each Arduino and target node. A 2.4inch LCD touchscreen+SD module was chosen so that these prototypes are close to a standard Lenovo Decora™ wall-plate slot. A commercial product would need to refine into a more exact LCD package.

I have not yet purchased components for the triac AC line control circuits, as I will need more time to evaluate available app notes and examples to determine appropriate part numbers to use in this design. I will do this during the weeks leading up to my scheduled time to work on this part of the system, leaving a week or so for shipment and delivery. If delivery is for some reason delayed a little, then I will begin assembly of the Fan and light fixtures with some wiring and electronics enclosures in preparation of the triac circuits coming, and then begin work on the Android app until the AC control components are received.

I already have a couple Arduino Uno boards, have ordered one each of a few items for fast delivery, and ordered additional copies of some items at lower cost with slower delivery. I have now received Wireless+SD shield and LCD touch/SD shield. I also have ordered a few Arduino Due boards, and already have one in possession. Additional Unos and Dues are in shipment from Ebay sellers located in China. I have already received some items from Chinese sellers, and anticipate the remainder of my orders arriving by the end of the first week in October, 2015.

The Linux server will be built using an AMD Gizmo embedded computing board, a 7 inch LCD touchscreen, a USB WiFi dongle, an Msata storage card, and some custom connectivity to the Xbee module. I already have the Gizmo kit. I had purchased a Gizmo kit a few months ago in order to get an AMD jtag probe for unrelated personal interest. I will likely need to do some bread-boarding

For the Android class, I will be using the smartphones that My wife and I already own and use. I will also use a home Wifi network router that I already own as the backbone of my simulated internet between the Android devices and the Linux server.

I have acquired a couple simple, inexpensive light fixtures, wall electrical boxes, and an inexpensive ceiling fan unit, to act as my targets to be controlled by this system. If this fan unit, the least expensive one I could find amongst my local hardware stores, proves unsuitable, I do have another ceiling fan unit of higher quality pending a small home improvement project, which could be used as a backup in this project. I don't expect this to be necessary as there are several online examples and app notes in general agreement for how to use triacs to control ceiling fan motors. While a few discussions indicate that a capacitor and/or frequency based speed control circuit may be more ideal, I have not found any app notes or examples for this approach, but I will keep it in mind as an alternative should I have significant problems with the triac design. I will initially focus on the triac concept due to apparent popularity.

I plan to assemble the triac circuit I will use in an Ltspice simulation. Then, once satisfied with function and performance, I will use a suitable oscilloscope to observe the AC waveform applied to the initial test load, by a prototype hardware setup with an inexpensive incandescent light bulb. This will start out as an Arduino Uno being hard-coded for speed and/or being controlled by a PC laptop. I also have a commercial wired style light dim and fan speed control pending a home improvement project, which I can observe with a scope to learn behavior if I have problems with triac and need some ideas.

Table 2 shows various components required, how many of each, and the price. An estimated total cost is then shown as a sum of these items. The software development environments and the Linux operating system are all available free of charge, and without an expiration date.

Table 2:

Component	Needed	Already Have	Price Each	Total	Description
Arduino Uno	3	3	$3.67		Microcontroller in direct control of target load units. Cheap clones from Ebay.
LCD Touchscreen /SD shield	3	0	$4.60	$13.80	For wall “switch” control nodes. Alternative is Adafruit 2.8 inch model at $30 each. I already have one of those, and would need minimum two more for demo.
Arduino Due	3	0	$16	$48	Microcontroller for wall “switch” LCD touchscreen controls
Wireless/SD shield	6	0	$8.16	$49	For ALL Arduino nodes
2GB microSD cards	6	0	$2	$12	“Mass” storage for graphics, system node configuration, room locations, etc.
AMD Gizmo embedded computing board	1	2	$200		Linux server
Light fixture	2	0	$5	$10	Simple and cheap
Ceiling Fan	1	0	$32	$32	Cheapest at local store
Android mobile device	2	2			My wife and I already have Android smartphones. One supports NFC feature, the other does not. Both support GPS and Internet data networking.
Xbee 5V/3.3V breakout board	1	0	$4.50	$4.50	Level shifting and breadboard fitting for Xbee on Linux server node.

Zigbee module in Xbee format	7	0	$26	$182	Wireless networking module for Arduinos etc.
Total				$361	approximate

Development Plan

The source code, circuit diagrams, documentation, and any other digital data related to this project will be committed and tracked using a Design Management (or Configuration Management in some vocabularies) repository using the GitHub service, located at the link below. This is set as a private repository, and I'll add professor and class members who may be interested, on request, and once I figure out how to do that.

https://github.com/amigabill/jhu_EN525.743

I plan to work on the various system components in the following order:

1. Arduino
    - SD card storage (contains some node configuration data)
    - Zigbee communication (With PC laptop as coordinator at this point)
    - Target control of AC line triacs (with at least some visible output indication, such as a small LED, in place of final triac circuit)
    - Touchscreen input events, using some pre-planned location for test "buttons"
    - LCD GUI implementation
    - Anticipating the possibility that the available libraries I plan to use may not all fit together into an Arduino Uno memory space, I plan to use Due boards for the LCD touch control nodes. Uno's are likely to be suitable for the target nodes, as those do not require LCD graphics related libraries or touch drivers, and will have a smaller memory footprint.
2. Linux Server (AMD Gizmo embedded project board)
    - Zigbee coordinator
    - Communicate with targets
    - Communicate with wall touchscreens
    - Implement event logging
    - GUI using wxWidgets library
    - Networking with internet, using a home WiFi router to emulate the internet. (Overlaps with Android App communicating with Linux server)
3. Android smartphone app
    - Communicate with Linux Server
    - GUI indication of targets current states
    - GUI manual control of targets
    - NFC tag manual location indications (easier to demo than Geofence)
    - GPS Geofence locations (somewhat harder to demo than NFC)

The anticipated schedule is shown in Table 3.

Table 3:

Milestone Number(s)							1	2			3	4				5
week of semester (Fall 2015)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
date	20150831	20150907	20150914	20150921	20150928	20151005	20151012	20151019	20151026	20151102	20151109	20151116	20151123	20151130	20151207	20151214
First class - initial ideas																
Second class - review ideas and commit																
PDR																
CDR (Critical Design Review)																
Software tools install																
Arduino Zigbee communication																
Arduino SD card data access																
Arduino LCD touch sensing																
Arduino LCD graphics																
Server Zigbee																
Server event logging																
Server vacation mode events																
Server LCD panel																
Server LCD touch																
Arduino triac interface for lights/fans																
Android connection to server/status/logs																
Android manual events to server																
Android NFC events to server																
Android geofence events to server																
Demo																
Travel for Work																

There are 5 major milestones.

1. Arduino Uno target controller
2. Arduino Due wall-plate LCD touchscreen controller
3. Linux Server/Coordinator
4. AC line load controller circuit, which is controlled by #1 above.
5. Android Smartphone Application

It's important to start with the Arduino targets and controls, as without them there is nothing for a Linux server or Android app to do. The next major priority is the Linux based server, to coordinate the other Zigbee nodes, observe and control target events, etc. This server along with the Arduino nodes then make up the bulk of this project's core functionality. Once the server is complete, then I will implement the triac circuitry to control the AC line to a real target light or fan motor in place of the earlier placeholder status LED. At this point, within the home is a fully functioning system with “real” targets working. Finally, I will move on to implementing the Android app. Should scheduling slip, the Android app is not as critical as the Arduino nodes or Linux server roles in this project, and so if anything should not get completed, this app is the most reasonable choice to be worked on last.

Some modifications may be required to the shields to improve signal quality on Arduino pins used on both modules, such as D4 which is SD_CS on the Xbee/SD shield, and an LCD data pin on the LCD shield. Cutting the D4 and other SD slot traces on the Xbee/SD shield, and not using SD on the Xbee/SD shield can help. The LCD shield has another microSD slot on it, using D10 for SD_CS instead of D4.

There are some risks in this schedule.

- I have not worked with Arduino or AVR microcontrollers before. During my previous experience with Atmel Sam-4 ARM devices, I worked some with SD protocol, in both C and Verilog, which would be beneficial to this project. I also worked with other peripherals such as Usart and Duart, parallel ports, interrupts, and I also had experience programming the Rabbit3000 and Xilinx Microblaze in prior courses at Johns Hopkins University, to show that I can move to different library APIs/ABIs in an acceptable time-frame for comparably complex projects.
- I have not worked with wxWidgets for a GUI application in Linux before. There are Rapid Application Development tools available, which I will make use of to assist in building the general code framework and creating the GUI interface. There is a thriving online support community for wxWidgets, and I have been looking at the API a little while pondering an unrelated project to port the wxWidgets framework to an operating system it is not currently available for, so I'm not going into that with a completely clean slate. I have not worked with other GUI frameworks either, such as QT, GTK etc. and I have less general familiarity with them than I do with wxWidgets, which makes wxWidgets a slightly better choice due to that mild familiarity.
- I have a small amount of experience with internet networking, such as the communication between Android app and Linux server. This experience is from the JHU course EN.525.742 System on Chip FPGA Design Lab, which I took a couple years ago. There I learned about ethernet message packets using Xilinx Microblaze microcontroller and a minimal library to hit the hardware, and the use of WireShark for debugging packets coming out of the FPGA board. I will be sending simpler messages in this project.
- How to demo geofence within range of WiFi router in the classroom? I will see if I can bribe my wife into being a remote participant during demo night, using her Android device over cell data network communicating with the Linux server via the WiFi hotspot in my own Android device, which will act as my in-classroom “home internet router” for the Linux server.
- I have not programmed in Java before. I plan to use a Rapid Application Development tool to assist with general code framework and creating the GUI interface. From there, I will have to be able to take a general pseudo code of my intended application and quickly learn the Java syntax to accomplish that design. Online Java examples abound, and there are several online support communities available when I have a question.
- During week 5, after the PDR presentation, my supervisor at work has asked me to attend a week-long training seminar in the month of November, along with everyone else in the group under him. It will be difficult to keep up with my normal work, the training, evening group meetings and other activities, and also continue a good pace on this project. I have requested permission to take a few days vacation to focus on this project to compensate, and also described the situation regarding this course and scheduling, and this request has been approved.

- My initial choice of an inexpensive 2.4 inch LCD touchscreen shield for Arduino is proving to be a challenge. It seems that the manufacturer may have recently changed which LCD driver chip is used, and the previously indicated Arduino driver libraries are not yet giving me a picture from the demo programs. If this is not resolved by finding the correct different library version, then I will change to a better supported 2.8 inch LCD touchscreen from a different vendor, and with a higher price tag. As this design is to prototype the concept, and any commercialized implementation would likely redo certain things such as to change to an LCD panel fitting more ideally with a standard Decora™ wallplate, which would most likely be done if using the 2.4 inch LCD for this prototype anyway, then this change should not be seen as a significant problem.

References

The programming environments, operating systems, and support libraries that I plan to use are all available at no cost. There are no expensive licenses to buy or to expire at an inconvenient time.

Programming IDEs, SDKs, and Frameworks:
- Arduino - https://www.arduino.cc/en/Main/Software
- Android Studio/SDK for Java - https://developer.android.com/sdk/index.html
- Android NDK for C/C++ - https://developer.android.com/ndk/index.html
- Linux for AMD Gizmo board - http://www.timesys.com/register/gizmo
- wxWidgets (for Linux GUI in C++) - http://wxwidgets.org/

I plan to make use of available libraries as much as possible for various components of the software.

Arduino Libraries:
- Arduino SD library - https://www.arduino.cc/en/Reference/SD
- Arduino TFT Library for LCD panel driver, or relevant equivalent for the LCD controller used - https://www.arduino.cc/en/Reference/TFTLibrary
- Yamsam LCD https://github.com/yamsam/TFTLCD_ST7781

Arduino Shield documentation:
- Xbee+microSD - https://www.arduino.cc/en/Main/ArduinoWirelessShield
- LCD touch+microSD
    - http://www.smokeandwires.co.nz/blog/a-2-4-tft-touchscreen-shield-for-arduino/ (for same LCD controller info and library)
    - https://github.com/yamsam/TFTLCD_ST7781
    - SmokeandWires LCD http://www.smokeandwires.co.nz/blog/a-2-4-tft-touchscreen-shield-for-arduino/
    - Adafruit Graphics Core library - https://learn.adafruit.com/adafruit-gfx-graphics-library/overview

Digi Xbee
- http://ftp1.digi.com/support/documentation/90000976_W.pdf
- http://knowledge.digi.com/articles/Knowledge_Base_Article/What-is-API-Application-Programming-Interface-Mode-and-how-does-it-work
- https://github.com/andrewrapp/xbee-arduino
- http://ftp1.digi.com/support/utilities/digi_apiframes2.htm
- http://arduino.stackexchange.com/questions/1500/how-to-make-xbee-module-interrupt-wake-arduino

Ltspice
- http://electronicdesign.com/power/model-diacs-and-triacs-ac-line-control
- http://electronics.stackexchange.com/questions/87546/ltspice-mac97a8-triac-model-vgs-error

There are several online app notes and example designs which will be relevant to this project:

- Atmel AVR datasheet (Arduino Uno) - http://www.atmel.com/Images/doc8161.pdf
- Atmel SAM3 Cortex-M3 datasheet (Arduino Due) - http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf
- Triac AC line control:
- **Microcontrollers: From Assembly Language to C Using the PIC24 Family** http://www.freescale.com/files/microcontrollers/doc/ref_manual/DRM039.pdf
- http://cache.freescale.com/files/microcontrollers/doc/app_note/AN3471.pdf
- http://www.st.com/web/en/resource/technical/document/application_note/CD00003820.pdf
- http://ww1.microchip.com/downloads/en/AppNotes/91094A.pdf
- http://www.mouser.com/catalog/specsheets/stevalill004v1.pdf
- https://books.google.com/books?id=ndALAAAAQBAJ&pg=PA683&lpg=PA683&dq=120V+AC+triac+3.3v&source=bl&ots=I3s7rIZYNT&sig=KcAZ0yObYB2d9QJC0kRAYhIQLkU&hl=en&sa=X&ved=0CCYQ6AEwAWoVChMItZzzj_yPyAIVgyaUCh0C8Az7#v=onepage&q=120V%20AC%20triac%203.3v&f=false

LCD touchscreen for Linux Server

- http://www.sainsmart.com/7-inch-tft-lcd-monitor-for-raspberry-pi-touch-screen-driver-board-hdmi-vga-2av.html