SmartHome System Design Documentation

The Customer

Prof. Houser, of the Johns Hopkins University, is the effective customer for this design. This is being done as my "capstone semester project" course EN.525.743 during Fall 2015 semester. The design details were discussed and negotiated with Prof. Houser at the beginning of this semester, and his suggestions for component selection and feature requirements are considered to be customer requirements, with me in the role of the contract design engineer. Any potential conflicts or scheduling issues, etc. were discussed with Prof. Houser, and any changes were made with is approval.

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. The intention is to complete a project comparable to popular X-10 systems, but which will send control commands wirelessly instead of signaling over the 120V power line.

This project evolved, at the customer's direction, from an initial concept of a simple "wireless light switch" control, 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 or other components done by other students, nor do other projects depend on this.

Control messages will be sent wirelessly from user input control units to target load driver units. The wireless communication will be using the Zigbee Mesh protocol. Xbee Series 2 modules will be used, in API frame mode and with built-in encryption capability enabled.

Wall “switch” user input 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 load 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.) At this time, a 2.8 inch diagonal size LCD touchscreen is used for prototyping this system. Ideally, a smaller size LCD touchscreen would be used that would naturally fit into a standard Decora™ style wall gang and switch wall-plate, but that will be left as an improvement for commercial productization of this concept, as no such LCD touch panel was conveniently available for this university project. The wall-switch user input control will be an Arduino Mega 2650 board using an Atmel AVR microcontroller.

The Arduino Mega replaces my original plan to use an Arduino Due, as the Due microcontroller presented compatibility issues with the LCD touchscreen libraries and Zigbee messaging. The Mega has enough flash and RAM resources to work with the libraries planned to be incorporated into these units, whereas the Arduino Uno is too small for this amount of code. Mega also has on-board EEPROM, which Due does not, which may or may not be a useful feature. Multiple controllers can select the same node at the same time, and will keep in sync regarding the current intensity gauge level displayed on the LCD GUI. Each controller watches the SmartHome messaging traffic to see if its selected load transmits a new level and powered state status message, and will update the display accordingly. When the controller selects a different load, then it will send a message to that load to inquire about its current level and state, and update the display accordingly.

The load items being controlled will use an Arduino Uno board, using Atmel AVR microcontroller, as the Zigbee command receiver and target load driver. Dimmable light fixtures or speed-controllable motor-driven load items will use a common Triac-based circuit to control the load as desired by the user. This Triac relay function will be provided by the combination of a Zero-Cross Tail and PowerSSR Tail units, at customer's request. The Zero-Cross Tail unit provides a signal indicating when the 120V AC line waveform crosses its Zero volt level, which happens twice during each AC sine wave, and this provides timing information used by the light dimmer and motor speed control components.

A central "gateway" or bridge computer will log user command events. This gateway will also act as the Zigbee wireless network “coordinator” node, in charge of the Zigbee mesh network. This gateway will also have an LCD touchscreen panel acting as an event log viewer, as well as acting as another user input control. In the future, this Linux node can provide a gateway interface for distantly remote items, such as Android smart-phones, to control the various loads, via the internet. This gateway runs on 64bit Lubuntu, a Linux operating system optimized for small resource x86/x64 PC hardware, on an AMD Gizmo-2 embedded PC. This Linux Gateway node is not required to be present, it is yet another controller, and it spies on SmartHome messaging traffic to keep a log of events, but if this computer was absent then the other wall control and load driver nodes would continue to operate, so long as there was a Zigbee Coordinator somewhere. Placing the Coordinator module with the Linux gateway is merely a convenience in remembering where that particular Zigbee module is, rather than leaving he user to try and remember which wall switch or light fixture it is installed in.

This project will also be designed to allow future expansion, with possibility to add new kinds of load items to control and new types of controls. 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

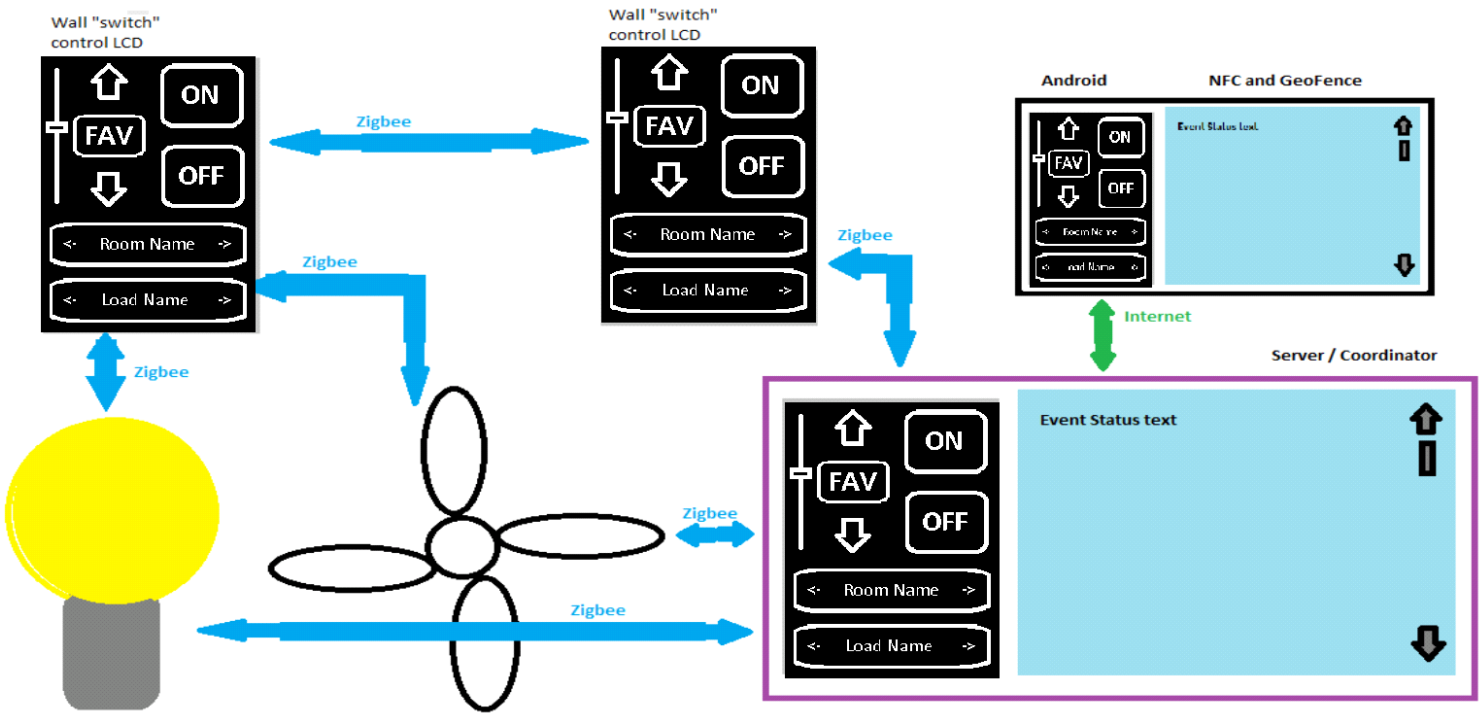
soldering stations, in case you forget to turn them off when you leave home, but these examples were not included in this semester's project effort.

While the ideal would be for the wall “switch” controls to fit into a standard Decora™ wall-plate and housing gang area, or perhaps in a somewhat larger Legrande "Adorne" style wall-plate, and Load Driver units 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 (power conversion for 120V AC to 5V or 3.3V DC, custom physical volume-optimized Triac and Zero-Cross circuitry, etc.), 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/proof of concept.

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 gateway 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. As previously mentioned, the Android portion of this project was removed from the list of goals.

Diagram 1: Example configuration of two wall controls, one Linux gateway node, on elight fixture and one ceiling fan, plus a potential future Android mobile remote control node via internet.



At boot time, any Arduino based node (either Uno or Mega), 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 Uno Load Driver node microcontroller will initialize to all loads being OFF, and with some hard-coded FAVorite intensity level, and then begin listening for command messages from the Zigbee network.

A wall control Arduino Mega node will proceed from the Xbee config check to gather information about controllable load units, which Load Driver 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 and which default room location it is in, 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. The control node will watch for incoming Command Complete messages, and if found, then check if it should update the displayed intensity level, should the message be coming from the same load as this controller is currently selected to. If a valid touchscreen (GUI button) event is detected, then one of two things can happen.

The first type of response to a GUI button press is to change the selected room location and/or selected load. Changing to a different room will then change to that room's default load. A change of room and/or load will pull in the newly selected ID data from microSD card files and then update the node tracking structures in memory as well as update the LCD display appropriately. If the load has changed, then assemble and transmit a Zigbee frame to ask the new load what it's current intensity level and isPowered state are, update the current intensity level gauge as indicated by the Load's response message, and then return to waiting for incoming Zigbee messages and for touchscreen events.

The second type of response to a GUI button press is to assemble an appropriate Command Init message into a Zigbee frame, and transmit that to the load Driver recipient node. Then wait for a Command Completed response over Zigbee, and update the intensity level meter to the value indicated by the load Driver node in it's response message, and then return to waiting for incoming Zigbee messages and touchscreen events.
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 gateway will boot up, prepare the serial port for non-blocking access and Zigbee networking, open the even log file for appending, read and display any previously saved event log text data, read from storage the default room and load selections and the relevant node data, display the default room and load names on the LCD, send a Zigbee message asking the default load's current intensity level, and paint the now current intensity level to LCD, for the now selected default load ID. Then begin listening for Zigbee commands, listening for commands from the Internet, as well as waiting for touchscreen inputs.

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 Zigbee standard API frame, as well as the SmartHome message payload, use Big-Endian (BE) format for multi-byte data fields. As the frame is transmitted or received, the MSByte is transmitted or received first, followed by any middle bytes, and finally LSByte comes last, in Big-Endian standard format throughout. The AVR microcontrollers as well as the AMD processor used in this project are all Little-Endian format devices, and so byteswapping by some means will be required to sort the Zigbee and SmartHome data fields appropriately into local memory data structures for use.

The Zigbee module will be accessed using a serial port running at 9600 baud, 8N1 configuration. Zigbee API mode frames will be used, with both the 64bit and 16bit destination addresses set to the special "broadcast" value, so that any frame transmitted will be received by all other nodes on the Zigbee network. Then, the SmartHome message content will be used to determine the SmartHome sender and recipient IDs, so that there is potential to move to different transport methods if needed. Encryption of the SmartHome message payload by the Xbee module is optionally supported, but is disabled during development and debugging. It can be enabled when the Zigbee communication is working well, but must be consistently enabled for all nodes, or disabled for all nodes. This setting can be done in software, setting a bit in the Zigbee API frame control bytes.

The Zigbee PAN ID used in this project is the value "42", for all Xbee modules.

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

1. Sender sends command to recipient
2. (not yet implemented) Recipient Acknowledges command received, asks sender for confirmation
3. (not yet implemented) Sender sends confirmation
4. Recipient indicates completion success or failure, and if successful tells the control node what the new current intensity level and isPowered states are.

Diagram 2 shows the Arduino and shield configuration for an LCD touchscreen wall control node. The bottom blue bar new represents an Arduino Mega board, not an Uno or Due board.

Diagram 2:

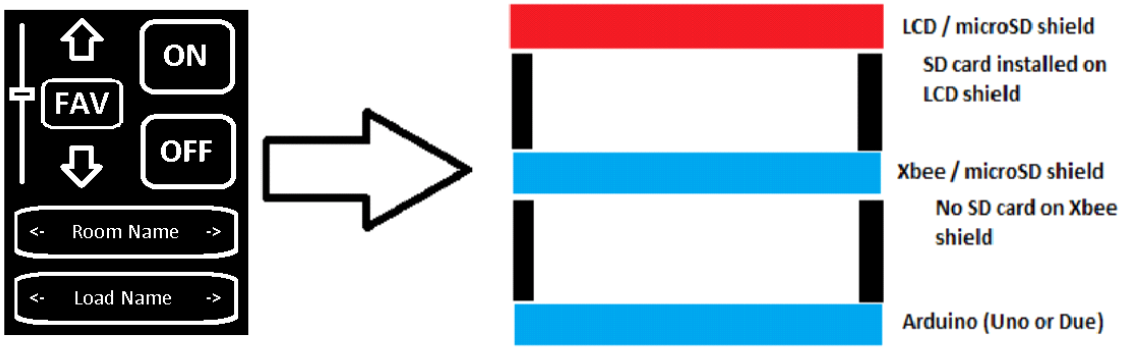
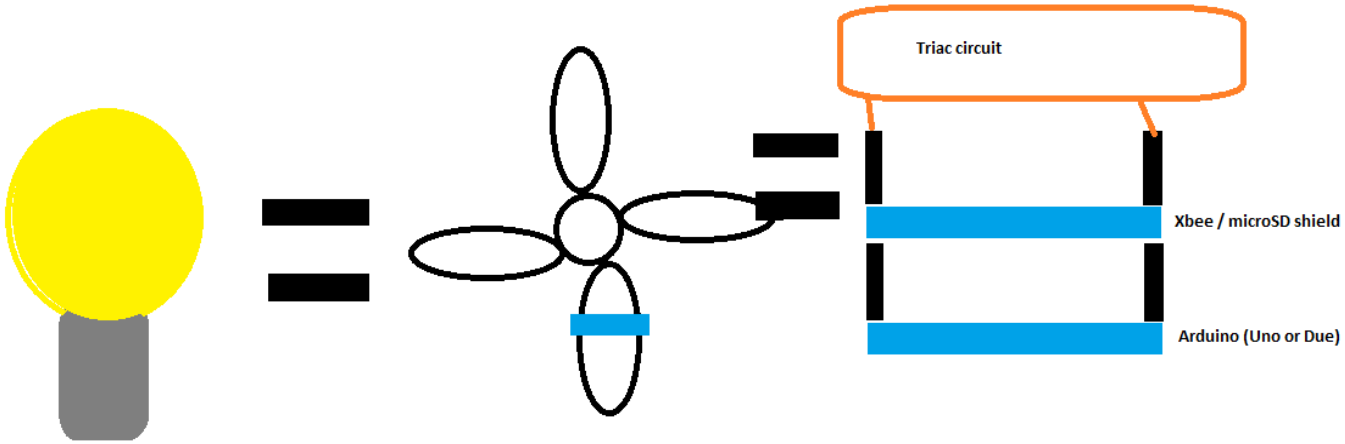


Diagram 3 shows the Arduino and shield configuration for a Load Driver node. This is identical for a light controlling node, a ceiling fan controlling node, and possibly some other node types which could be added in the future. One Load Driver node can actually control two, or possibly a few more than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

Diagram 3:



Interface Description

The most commonly used interface is 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 location and load device, view current load state (as the target load may be located in another room and not directly visible)
- Control buttons to increase or decrease load brightness or speed, full on or off, or set to some favorite intensity level for that target load. An indicator will move up and down in a small track to indicate the currently set intensity level.
- A Linux Gateway computer, with a larger LCD touchscreen, will provide a similar user interface, and also add a text area to display logged command event information.
- A set of buttons will allow to change what room the controller is looking for load devices in, as well as to change which load in the selected room is to be controlled. The available rooms and loads are stored on a wall control unit's microSD card, or on local disk storage (mSATA) for the Linux Gateway node.

The Linux Gateway GUI and an Android app will display the same control panel as described in Diagram 1 for the wall controls, but adds visibility of the event logging data as well. This keeps things 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 on the small wall switch controls.

A data structure is used to hold the currently selected load's SmartHome node ID, type, Arduino pin controlling that load, the current intensity and FAVorite intensity levels, the isPowered state, a SmartHome message structure, and whether or not a message is pending transmit or has just been received. Putting a SmartHome message data structure inside the node info structure allows an Arduino unit to have multiple loads connected, each with its own node info struct, and thus its own SmartHome message buffer. This way, messages can potentially come in from various other nodes in different ordering, such as if two controls happen very close together in time, one does not have to wait for the other to finish, they can send their messages in a sort of "interleaved" fashion. Each node will only track relevant messaging, and ignore unrelated messaging, which might go to a different load on the same Arduino unit.

Use standardized data type names, such as uint8_t for 8bit unsigned char, uint16_t for 16but unsigned int, uint32_t, etc. for easier code-sharing across different platforms. Some data structures and some code is directly shared on both 8bit AVR and 64bit x64 processors and their different compilers. Using these standardized types helps avoid inconsistencies of other data types, which can potentially be different and incompatible across different compilers or platforms, for example, an "unsigned int" might be 16bits from one compiler or 32bits in another compiler.

The SmartHome Message data structure:

```
typedef struct
{
   volatile uint16_t SHothrID;       // The "other node ID" in this message,
                                     // either the destination this node
                                     // sends to OR the source node the
                                     // message was received from.
   volatile uint8_t  SHmsgType;      // which of the 4 message types
                                     // is this (or 5th option is IDLE)
   volatile uint8_t  SHcommand;      // the SmartHome command
   volatile uint8_t  SHstatusH;      // used to pass the FAVorite level
   volatile uint8_t  SHstatusL;      // used to pass the isPowered state
   volatile uint16_t SHstatusID;     // 16bit ID alternative to SHstatusH and
                                     // SHstatusL but represents same bytes
                                     // in message. (Not yet used)
   volatile uint8_t  SHstatusVal;    // current/new load intensity level or
                                     // failed satus
   volatile uint8_t  SHreserved1;    // not yet used
   volatile uint8_t  SHreserved2;    // not yet used
   volatile uint8_t  SHchksum;       // from this node or defined by another node
   volatile uint8_t  SHcalcChksum;  // SmartHome message checksum calculated
                                      // for comparison to the sender's value
   volatile uint8_t  SHstatusTX;     // status of attempt to transmit a Zigbee API
                                     // frame (not used)
   volatile uint8_t  SHstatusRX;     // status of receiving a Zigbee API frame
                                     // (not used)
} SHmessage, *prtSHmessage;
```

The SmartHome Node ID data structure:

```
typedef struct
{
   volatile uint16_t  SHthisNodeID;         // this load's node SH ID, might be one
                                            // of multiple loads on this node
   volatile uint8_t   SHthisNodeLoc;        // what building room is this node in?
                                            // (8bit room ID)
   volatile uint8_t   SHthisNodeType;       // 0=ctrl, 1=light, 2=fan
   volatile uint16_t  SHthisNodePin;        // ONLY used on Uno Load Driver node
                                            // What Arduino board pin is controlled
                                            // by this load Node? This is 16bit int
                                            // value such as D6, D9 etc. #defines
   volatile uint8_t   SHthisNodeIsPowered;      // is ON to some brightness/speed
                                                // level, but something more than
                                                // full-OFF, or full-OFF
                                                // 1=ON, 0=OFF
   volatile uint8_t   SHthisNodeLevelCurrent;  // current dim/speed level
   volatile uint8_t   SHthisNodeLevelFav;      // FAVorite dim/speed level
   volatile uint16_t  SHothrNodeID;         // other node in this SH message
                                            // conversation, 0 if idle
   volatile uint8_t   SHmsgCurrentState;  // which of 4 message stages are we
                                            // in now, or 0=idle?
   volatile uint8_t   SHmsgNextState;       // which of 4 message stages are we
                                            // in now, or 0=idle?
   volatile uint8_t   SHmsgCmd;             // current message command
   volatile uint8_t   SHmsgStatus;          // current message status for this node
   SHmessage       SHthisNodeMsg;           // SmartHome message fields for this
                                            // node ID, to be in parallel to message
                                            // fields of other Node IDs
   volatile uint8_t   newSHmsgTX;           // YES if a SH message is waiting
                                            // to be sent, NO if not
   volatile uint8_t   newSHmsgRX;           // YES if a SH message has
                                            // been received, NO if not
} SHnodeInfo, *prtSHnodeInfo;
```

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. Load Driver nodes will not include the LCD shield, and so there is no conflict on pin 4. Also, due to driver library issues, the 2.4inch LCD was not used, so those two columns of pins can be ignored below. A 2.8inch LCD was used instead, which makes use of the bottom-row 6-pin SPI interface instead of the side column digital and analog pins.

Diagram 4:

Arduino pinouts

Triac/Misc	Wireless/SD	LCD/SD (2.4inch LCD)	Mega	Uno R3	pin ID		pin ID	Uno R3	Mega	LCD/SD (2.4inch LCD)	Wireless/SD	Triac/Misc
							17					
							16					
							AREF / 15					
							GND / 14					
					NC		13			SD_SCK	SD	
					IOREF		12			SD_DO	SD	
					Reset		11			SD_DI	SD	PWM OC2A
	3V3	3V3			3V3		10			SD_SS (CS?)		PWM OC1B
	5V	5V			5V		9			LCD_D1	(OC1A) →	PWM OC1A
					GND		8			LCD_D0		
	GND	GND			GND							
					Vin		7			LCD_D7		
							6			LCD_D6	(OC0A) →	Light
btn On/Off or	ON	LCD_RD			A0		5			LCD_D5	(OC0B) →	Fan
btn Up		LCD_WR			A1		4			LCD_D4	SD_CS	
btn Down		LCD_RS			A2		3			LCD_D3		PWM OC2B
Xbee config s	tatus	LCD_CS			A3		2			LCD_D2		
AC 0-cross /	btn fn/lt sel	LCD_RST			A4		1				TX	
Xbee config o	r OFF				A5		0				RX	
					A6							
					A7		14					
							15					
					A8		16					
					A9		17					
					A10		18					
					A11		19					
					DAC0		20					
					DAC1		21					
					CANRX							
					CANTX							
				Adafruit LCD uses SPI port along bottom								

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 Push-buttons and other in/output signals:

Description of Misc and Triac Signals	
Signal Name	Description
Xbee config (also btn fn/lt sel??) pin A5	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 pin A3	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 pin A4	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 load, before adding LCD touch shield support. Once LCD shield is added, this button must be removed from wall control.)
btn On/Off pin A0	Active-Low – When asserted, change the load power applied state. Like a “bang-bang” power enable control. (Also used on wall control node development to test communication and control of load, before adding LCD touch shield support. Once LCD shield is added, this button must be removed from wall control.)
btn Up pin A1	Active-Low – When asserted, increase intensity (brighter/faster) to load. (Also used on wall control node development to test communication and control of load, before adding LCD touch shield support. Once LCD shield is added, this button must be removed from wall control.)
btn Down	Active-Low – When asserted, decrease (dimmer/slower)

pin A2	to load. (Also used on wall control node development to test communication and control of load, before adding LCD touch shield support. Once LCD shield is added, this button must be removed from wall control.)
load 0 (light) pin 6	Software bit-banged control of triac circuit for the Light load
load 1 (fan) pin 5	Software bit-banged control of triac circuit for the Fan load
AC 0-cross pin A4	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.

Table 2: Zigbee Payload / SmartHome Message

Payload / Message Offset	Description offset 0 is first byte sent/received, offset 11 is the last one
0 1	SmartHome Destination address MSByte (big-endian BE) SmartHome Destination address LSByte (big-endian BE) 16bit address of a user input control unit (LCD touchscreen Arduino Mega), a load to be driven (light or fan), or the Linux gateway. An Arduino Uno load driver may have multiple SmartHome Node addresses under it's control, so this is not an address per Arduino board, this is an address per Node. Each load driven is a unique "Node" with its own SmartHome address. Linux gateway is at special address of 0x0000 Linux gateway logs ALL SmartHome events, and it will use this address information in the event log.
2 3	SmartHome Source address MSByte (big-endian BE) SmartHome Source address LSByte (big-endian BE) 16bit address tells the message recipient where to send

	response stages of a full command conversation. Linux gateway logs ALL SmartHome events, and it will use this address information in the event log.
4	SmartHome message type (Command Request/Init, Acknowledge, Confirm, Complete) This field tracks what stage of the SmartHome messaging protocol "conversation" is the current stage. Currently, the Command Request/Init and Complete messages are implemented in this protocol.
5	SmartHome Command
6	Status H byte, MSByte (BE) if have a 16bit value, or a second 8bit status Status-2. This so far has been used to pass the FAVorite intensity level value between control and load nodes, for both the Read Current and Read Favorite commands.
7	Status L byte, LSByte (BE) if have a 16bit value, or a third 8bit status Status-1. This has so far been used to pass the isPowered state between control and load nodes, for both the Read Current and Read Favorite commands. Value 1 means ON, value - means OFF.
8	Status Value, 8bits. This is usually used to pass the current intensity level value between control and load nodes, such as when the load tells a control unit what current level to display on the level gauge part of GUI display.
9	Reserved for future use (Reserved1)
10	Reserved for future use (Reserved2)
11	Checksum of the previous 11 SmartHome Message bytes = 0xff - Sum(0:10)

Table 3: SmartHome Command codes

Command	8bit Encoding	Description
No OPeration (nop)	0x00	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	Go to FAVorite brightness/speed
Store current level as Favorite	0x06	Save current intensity level as FAVorite brightness/speed (not implemented)
Read FAVorite level	0x07	Ask target to say what the load's stored Favorite intensity level is
Read current level	0x08	Ask target to say what the load's current intensity level and isPowered state are

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 load drivers 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 Load Driver end will also require a light and a ceiling fan to control. An Xbee module will be required for each Arduino and Load Driver node.

A 2.8inch LCD touchscreen+SD module was chosen so that these prototypes are not very close to a standard Lenovo Decora™ wall-plate slot. A commercial product would need to refine into a more exact LCD package or some other wall gang/plate standard.

I have a few Arduino Uno boards already for use in this project. I do not have any Arduino Mega boards. I ordered one Arduino brand, and ordered some much less expensive knockoffs for additional wall controls. The one brand name unit was receivable much sooner due to faster shipping availability to get vendor support if needed and to get development moving while I waited for the less expensive companion units to arrive.

The Linux gateway will be built using an AMD Gizmo-2 embedded computing board, a 7 inch LCD touchscreen, an mSATA storage card, and a USB adapter for 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 purchased the least expensive ceiling fan/light combo unit at my local hardware store, to act as my two loads to be driven for this project. One load is the light, the other load is the fan motor i this unit. I also purchased a touch-controlled light dimmer unit, marketed as a replacement or add-on for metal-framed lamps, as a test configuration recommended in online forums to see if the inductive style motor could be reliably driven by something comparable to the more expensive PowerSSR Tail solid-state relay unit, since the PowerSSR Tail description said it did NOT support inductive loads, such as a ceiling fan motor. This test configuration worked, and so did the PowerSSR Tail.

Table 4 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 4:

Component	Needed (initial est.)	Already Have	Price Each	Total	Actually Used	Total Used	Description
Arduino Uno	3	3	$4	$12	1	$4	Microcontroller directly driving target load units. Cheap clones from Ebay.
LCD Touchscreen/ SD shield	3	0 0	$35 $5	$15	2 0	$70	Adafruit.com 2.8inch resistive Mcufriend.com (didn't work) For wall “switch” control nodes.
Arduino Mega 2650	3	0	1@ $47 2@ $8	$47 $16	2	$55	Microcontroller for wall “switch” LCD touchscreen controls. Arduino Due boards proved problematic. One is Arduino brand, others are Cheap Ebay Chinese clones.
Wireless/SD shield	6	0	$8	$48	3	$24	For ALL Arduino nodes. (both Unos and Megas)
2GB microSD cards	6	0	$2	$12	2	$4	“Mass” storage for graphics, system node configuration, room locations, loads etc. on Mega wall control boards.
Linux Gateway	1 1 1	2 1 0	$200 $30 $45	$200 $30 $45	1 1 1	$275	AMD Gizmo-2 embedded PC Msata SSD storage 7inch LCD touchscreen
Light fixture	2	0	$5	$10	0		Simple and cheap bulb holder
Ceiling Fan/Light	1	0	$32	$32	1	$32	Cheapest at local store combo fan/light unit
Android mobile device	2	2			0		One of ours supports NFC feature, the other does not. Both support GPS and Internet data networking. (Not Used)
Zigbee module in Xbee module format	7	0	$26	$182	4	$104	Wireless networking module for Arduinos etc. to fit Xbee module onto.
Xbee to USB adapter	1	0	$30	$30	1	$30	
Total				$679		$598	approximate

Table 5 shows the Various software products used in this project

Table 5:

Software Tool	License	Description / Usage
Arduino IDE	Free, open-source	Programming/development environment for Arduino boards.
Eclipse IDE	Free, open-source	Programming/development environment. I'm mostly using Eclipse as one interface to GitHub repository for this project, and for doing diffs between file revisions.
Code::Blocks	Free, open-source	Programming/development environment. CodeBlocks includes a GUI builder, called wxSmith, for applications using the wxWidgets framework, which allows you to "draw" your GUI interface and get C++ code template as output. I'm using this as the coding environment for the Linux Gateway/Control application.
Android Studio	Free	Programming/Development environment for mobile/smartphone app.
gitk / git gui	Free, open-source	Used in Linux to push code changes up to the git repository in Github.
Github desktop tool	Free	Used in MS Windows to push code changes up to the git repository in Github.
XCTU	Free (no cost) proprietary	Digi tool to configure Xbee wireless modules, to generate example Zigbee API frames, and to decode and display Zigbee API frames.
HxD	Free, open-source	Hexadecimal file editor, used to create binary-only files with SmartHome Node information to be used on microSD card at the LCD touchscreen and Linux Gateway units. The load driver units will use EEPROM to store the related information that they need.
Microsoft Paint	no cost, included with Windows	image editor tool, used to create the GUI buttons for Arduino and Linux gateway displays. These buttons are up, down, right or left arrow buttons, ON, OFF and

	OS	FAV buttons, and lines/bars, box shapes used to build the user GUI interface.
GIMP (Gnu Image Manipulation Program	Free, open-source	Used to manipulate images for the GUI display.

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

Due to one library used by the Wall Control Arduino unit, the SD library to access files on microSD card, this part is under the GPL v3 license (NO "or later version" clause). I'm applying the LGPL 2.1 "or later version" license to the files created for this project. As neither the load driver nor the Linux Gateway application make use of this 3rd party GPL3 library, they are not affected by the GPL3 terms, and those portions of this project can remain as LGPL 2.1 "or later", or as wxWindows licensing, although some future possibilities could, in the future, bring the load driver program under this GPL3 terms as well.

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

1. Arduino Load Driver
    - Zigbee communication transmit
    - Zigee communication receive
    - Control of AC line triacs, using the Zero-Cross Tail and PowerSSR Tail units as my Triac circuit. This initially is done using manual push-buttons for user control, until there is a separate control unit working over Zigbee to send the commands wirelessly.
2. Arduino Wall Control
    - Share as much Zigbee communication code as possible with Load Driver
    - SD card storage file access (contains some node configuration data and image files)
    - LCD GUI implementation - This was done using a modified "spitftbitmap" example program from the LCD display driver library to send various BMP format files to different locations of the LCD, to build the GUI interface. I discovered that the X coordinate for

where to begin painting an image to LCD must be X=0, or else the image will not be painted correctly. So images and locations of elements in the GUI were done with this limitation in mind.

- Touchscreen input events. I combined the "TouchTest" example program from the touchscreen driver library with the "spitftbitmap" example program from the LCD display driver, to display a fixed picture of the final wall control GUI, and display to serial monitor the touchscreen coordinates for each press, and then mapped out the coordinate boundaries for each button. These boundaries became the things to check for user input events, in the final Wall Control program.
- 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 Mega 2650 boards for the LCD touch control nodes. Uno's continued to be suitable for the Load Driver nodes, as those do not require LCD graphics related libraries or touch drivers, and have a much smaller software memory footprint. Indeed, fitting the wall control code into an Uno would have been problematic. Another student in this class encountered a problem, where things started breaking when memory utilization exceeded 74% of available, which is apparently a known phenomena. The 3rd party libraries I used for the wall control unit together used very close to that 74% of an Uno mark, without any of my own code added. Uno was fine for the load driver node, and I did use an Uno for that purpose.

2. Linux Gateway (AMD Gizmo-2 embedded project board)
    - base application with GUI, using wxWidgets framework
    - Zigbee coordinator
    - Zigbee communication with Load Drivers
    - Implement event logging

The initially anticipated schedule is shown in Table 6. Schedule slips on earlier features ended up removing the Android app as a goal for this project, during this semester. This was discussed with and approved by the customer, Prof. Houser, when it became apparent that an Android app would not be a successful component before the given deadline.

Table 6:

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 graphics																
Arduino LCD touch sensing																
Server Zigbee																
Server event logging																
Server vacation mode events (not completed)																
Server LCD panel																
Server LCD touch																
Arduino triac interface for lights/fans	Not Needed due to use of Zero-Cross an dPowerSSR Tail units															
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 3 major milestones.

1. Arduino Uno target load driver/controller
2. Arduino Mega wall-plate LCD touchscreen controller
3. Linux Gateway/Coordinator
4. (deprecated) Triac based 120V AC line relay circuit
5. (deprecated) Android Smartphone Application

I had initially planned for fourth and fifth major milestones, to design the Triac relay circuit , and an Android app for remote control via the internet. The Android remote app was removed from the project requirements per a discussion with the customer about scheduling. The Triac effort was nullified due to the use of the PowerSSR Tail and Zero-Cross Tail products, and the customer encouraged the use of these rather than creating a new circuit design for this purpose. This was a safety concern for the class project, and also ended up being a benefit to a slipping schedule.

It was important to start with the Arduino load drivers and wall controls, as without them there is nothing for a Linux gateway or Android app to do. After the initial wall control LCD touchscreen and Arduino load driver communications

testing (using LEDs in place of the AC line loads), I added the ZeroCross Tail and PowerSSR Tail units to interface with the 120V light bulb and ceiling fan motor. The next major priority is the Linux based gateway, to coordinate the other Zigbee nodes, observe and control target Load Driver events, etc. This gateway, along with the Arduino nodes, together make up the bulk of this project's core functionality. At this point, within the home is a fully functioning system with “real” loads 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 gateway roles in this project, and so if anything should not get completed, this app is the most reasonable choice to be worked on last.

The only Arduino shield modifications I needed to do was the LCD shield trace cut and solder something else one to put it into "Arduino Mega" mode. I did not encounter any pin conflicts between the LCD and Zigbee shields that needed correcting.

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.
- 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 await a reply to this request. I rarely take time off work, I now have a second person assisting in my role, and my overlords have a history of being very supportive of the time off requests that I have made in the past, so I do not anticipate that this training event will have a significant impact on this project schedule. I was able to gain 6 days of PTO from work to focus on this project, and was also able to

spend some time after hours during the trip and on the airplane working on the wxWidgets/Linux application part of this project.

Assembly Instructions

The wall control units are pretty simple to assemble. These consist of an Arduino Mega 2560 R3 microcontroller board, an Xbee wireless adapter shield with Xbee Zigbee module, and an Adafruit 2.8inch touchscreen LCD shield. Simply stack them together, with the Xbee shield attached to the Mega, and hte LCD shield attached to the Xbee shield. The LCD shield needs modified to use the SPI pins from the Mega board's location, as described at Adafruit website, at
https://learn.adafruit.com/adafruit-2-8-tft-touch-shield-v2/connecting

The Load Driver unit is more complicated. It consists of an Arduino Uno microcontroller stacked with an Xbee wireless shield and Xbee Zigbee module, plus the 120V AC relay interfacing. Each load driver microcontroller unit has one associated ZeroCross Tail to provide the power line sinewave zero-crossing timing signal as an input to the microcontroller. Then, each load driven by the microcontroller has its own PowerSSR Tail unit, so that a Load Driver board driving two loads will have two PowerSSR Tail units. The various *Tail units are plugged into an AC power socket, such as in a power strip, and the loads to be driven plug into the other end of the PowerSSR Tail units. The Zero-Cross Tail unit does not need to have anything plugged into the other side. The control signals to the various *Tail untis are connected to the Load Driver Arduino board as described in Diagram 4, at the Light, Fan and AC-0cross pins. The ground side of each *Tail control connection can connect to any Arduino ground pin.

The Linux Gateway is made up of an AMD Gizmo-2 embedded PC board. An mSATA SSD storage card is added to this as a "hard drive", and a plastic "zip-strip" was used to secure the mSADA card to the main board, as there are no mSATA standard screw offsets provided on the Gizmo board. The Sain brand 7inch LCD panel was assembled per instructions from the vendor, and connected to Gizmo by an HDMI cable and a USB cable. A wireless keyboard/mouse combination wsa used with a USB dongle attached to the Gizmo, a WiFi USB dongle was used for internet access, and a USB to Xbee adapter was used to connect the Zigbee module as the /dev/ttyUSB0 device. A USB external CDROM drive was connected to Gizmo during installation of the Lubuntu operating system, then after that it can be removed. After SmartHome software installation and configuration is complete, then the keyboard/mouse dongle can be removed as well. The WiFi dongle is currently not used after final installation and configuration, but can be left connected for future use as an Internet gateway.

Performance and Test Data

The Arduino Uno based Load Driver node, as in my prototype test platform, occasionally locks up and needs a power cycle to the microcontroller to reset and begin working again. This is infrequent, and I have not yet discovered the cause. it can run well for a couple days before freezing.

There is a noticable delay after pressing a touchscreen control and observing its effect, of a couple seconds. This takes a little getting used to at first, as an impatient user might wonder if it's working or not. The wall control units function well, and this delay isn't a critical issue for this application.

When the full system is powered on, and all load driver and control nodes are operating, it can take a bit of fiddling to get the load responding reliably to user inputs. I believe this is a mismatch between default levels/states at the control nodes and the load driver nodes, where one party thinks the load is ON, while the other thinks it is OFF. This can be synched up by an increase, decrease intensity or the FAVorite level command. The control units should be improved to inquire with their default load what its state is on bootup as a correction to this issue, but this had been overlooked earlier.

The Linux gateway application functions well, but can feel sluggish. Part of this is due to the Zigbee communication delay as mentioned above, but there is also a local component to performance. The current implementation makes sure that the "OnIdle" event function is called on very frequently, basically constantly when no other event (such as a GUI touch event) is in process. This function is effectivley the "main loop" for this software, which checks for serial data received or ready to be transmitted, updates the event log text to the display if needed, and any other things that need to be checked or done periodocally. This means that the AMD CPU is pegged at or very close to 100% utilization, which is more than necessary for this class of application. A sleep timer of some sort could help reduce CPU utilization to a minimal level for functionality and save some power consumption.

On occasion the Linux gateway misses an event and does not log it to file or display. More investigation is needed to debug this infrequent issue.

Ideas for future Enhancements and Additions

- Enable load driver EEPROM usage to record current level, isPowered state and FAVorite level to survive a power outage, and restore to that state when power returns.
- Add the internet gateway capability, for controls to come from distant locations via internet.
- The logging is relatively simple, using the hexadecimal node ID codes. A database could be added to retrieve room and load names from, for a more human-readable log message.
- In the future, consider changing the Linux Gateway node to use its microSD card slot as a way to program the microSD card for new wall control or load driver nodes, to be added to the system. For load drivers, this would also require adding SD card support to that particular software. Currently, only the wall control units use the microSD cards, and load drivers are hard-coded in the compiled software.
- Create another Linux applciation, or an extension or different mode to the current Linux application, to more conveniently create the node data files used by the Linux control panel or wall control units (and potentially load drivers in the future as well) for node ID, node type, room and load names, etc. Currently this is a tedious, manual effort, using several different tools (hex editor, image editor, text file editor at minimum).
- Additional things to be controlled, such as garage door, thermostat, a simpler toggle switch controller (no LCD/touchscreen), etc.
- Add save favorite level capability. For now, the favorite level is hardcoded into the load driver node.

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
- wxWidgets (for Linux GUI in C++) - http://wxwidgets.org/
- Code::Blocks IDE / wxSmith - http://www.codeblocks.org/
- Eclipse IDE - http://www.eclipse.org/
- 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

I plan to make use of available libraries and examples as much as possible for various components of the software, including:

Arduino Libraries:

- Arduino Serial Library - LGPL 2.1 "or later" -
  - https://www.arduino.cc/en/Reference/Serial
  - Used for debug and information data to serial monitor/terminal, as well as for communicating with the Zigbee module.
- Arduino SD library - GPL3 -
  - https://www.arduino.cc/en/Reference/SD
  - Used to read node configuration data from microSD card
- Adafruit_ILI9341-master TFT Library for LCD panel driver - MIT -
  - https://github.com/adafruit/Adafruit_ILI9341
  - https://github.com/adafruit/Adafruit-GFX-Library
  - https://www.arduino.cc/en/Reference/TFTLibrary
- Adafruit-GFX-library-master - BSD -
  - Draws lines and shapes to LCD screen
- Adafruit_STMPE610-master Touchscreen driver library - MIT -
  - https://github.com/adafruit/Adafruit_STMPE610
- Arduino TimerOne library (modified) - Creative Commons Attribution 3.0 United States License -
  - https://github.com/PaulStoffregen/TimerOne
- Arduino spitftbitmap picture display application - MIT -
  - example application reworked into a library function - as found in the Adafruit_ILI9341-master library package
  - Draws 24bit BMP image files to LCD screen. Limitation that X coordinate MUST be 0, it doesn't seem to work otherwise, either drawing garbage or nothing at all.
- Arduino EEPROM library - LGPL 2.1 "or later"
  - https://www.arduino.cc/en/Reference/EEPROM
  - For use in storing current intensity (brightness/speed) and FAVorite levels and isPowered state to NV memory, to restore state after power outage.
- GNU Byteswap - LGPL 2.1 "or later"
  - Provides macros to byteswap data between the Little Endian and Big Endian data formats.

Arduino Shield documentation:

- Xbee+microSD -
  - https://www.arduino.cc/en/Main/ArduinoWirelessShield
- LCD touch+microSD Adafruit Mega/ICSP connections -
  - https://learn.adafruit.com/adafruit-2-8-tft-touch-shield-v2/connecting

wxWidgets Framework - wxWindows Library License 3.1 (LGPL with an exception)

- GUI application for Linux Gateway node
- website dev information - http://docs.wxwidgets.org/trunk/modules.html
- "wxBook" : Cross-Platform GUI Programming with wxWidgets
  - http://www.amazon.com/Cross-Platform-Programming-wxWidgets-Julian-Smart/dp/0131473816/ref=sr_1_1?ie=UTF8&qid=1447737672&sr=8-1&keywords=wxwidgets

Digi Xbee Zigbee wireless modules

- 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
- Building Wireless Sensor Networks: with Zigbee, Xbee, Arduino and Processing book - http://www.amazon.com/Building-Wireless-Sensor-Networks-Processing-ebook/dp/B004GTLFHI/ref=sr_1_1?s=books&ie=UTF8&qid=1450237848&sr=1-1&keywords=zigbee

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 AVR datasheet (Arduino Mega 2650)
- Triac AC line control:
  - PowerSSR Tail 120V AC solid-state relay http://www.powerswitchtail.com/pages/powerssrtail.aspx
  - Zero-Cross Tail - Provides a pulse signal when the 120V AC line sinewave crosses the Zero volt center. http://www.powerswitchtail.com/Pages/ZeroCrossTail.aspx
- **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 Gateway/Control panel

- http://www.sainsmart.com/7-inch-tft-lcd-monitor-for-raspberry-pi-touch-screen-driver-board-hdmi-vga-2av.html
  - This worked almost flawlessly with standard Linux installation, and no additional drivers. The one imperfection is that the Y axis is inverted, so that if you touch near the top of the screen, the pointer goes proportionally near the bottom, and vice-versa. The X axis appears fine as default. Support inquiry to the vendor brought me to an X11 configuration tool that was very helpful in correcting this issue.
  - http://www.freedesktop.org/wiki/Software/xinput_calibrator/
  - https://wiki.ubuntu.com/Touchscreen
  - https://www.thefanclub.co.za/how-to/how-ubuntu-1204-touchscreen-calibration
  - This 7 inch display has a resolution of 800x480, which is not a standard resolution selectable in Lubuntu preferences. Use cvt and xrandr commands to create the modeline for this and then activate it for use.
  - http://askubuntu.com/questions/377937/how-to-set-a-custom-resolution
    - cvt 800 480 72
    - xrandr --newmode ...
    - xrandr --addmode HDMI-0 800x480_72.00
    - logout/login and then select new resolution in monitor preferences.