Wireless-Enabled Smart-Lights Hub Prototype

DESIGN DOCUMENT

Senior Design December 2019 Team 16
Dr. Ravikumar Gelli
Dr. Manimaran Govindarasu
Alexander Beaver
Ryan Bush
Aaron Ramsey
Logan Zasada

http://sddec19-16.sd.ece.iastate.edu/

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

Our faculty adviser/client is Dr. Manimaran Govindarasu and is a mentor for the project by helping us manage our time and progress, providing funding, and giving guidance for the direction of the project and any problems we face.

Manimaran's post-doctorate, our main client is Dr. Ravikumar Gelli, who helps make large decisions in regard to the direction of the project, as well as gives feedback on ways we can improve the project as it is created.

In addition to Drs. Govindarasu and Gelli, a student named Alexander Nicklaus was a student working for the PowerCyber lab over the summer. Alexander picked up where we left off in the spring, continued the project to the best of his ability, and transferred work and responsibilities back to us at the beginning of the fall semester.

1.2 Problem and Project Statement

Presently, the PowerCyber lab uses several relay boxes that are hardwired to light bulbs to display the status of power components in a simulation. This system is used as an outreach tool for the general public and a training tool for industry partners. The main problems with this system are that it can't be connected to enough lights to represent larger simulations, and it's not portable enough to bring to most outreach locations. To address these problems, our team is developing a system that will take inputs from the relay simulation and wirelessly control a portable display of wireless lightbulbs.

The light bulb display system can be made as large as it needs to be by adding or removing light bulbs from it. This approach wouldn't work with the existing system because the light bulbs have to be hardwired to the relay boxes and there isn't enough space near the boxes to neatly arrange the bulbs. Because the new system utilizes wireless lightbulbs, it will be portable enough to take to any outreach location. By implementing this system, we increase the PowerCyber labs' ability to perform outreach and train industry partners.

1.3 Operational Environment

The light bulb display system will only be operated indoors. The temperature of the system is expected to be around room temperature and not fluctuate. Some dust may accumulate on the system if it remains in one location for an extended period of time. Overall, the expected operational environment is stable and doesn't pose any significant design challenges.

Since the system is wireless, the only equipment/resources necessary for functionality are 1 120V plug to keep a laptop charged and a wireless connection to retrieve light statuses from the server in the PowerCyber lab.

1.4 Intended Users and Uses

For the proposed system there will be two groups of users: primary and secondary. The primary user group are the members of the PowerCyber lab. The primary users will be the ones that configure, maintain, and transport the system. System configuration will consist of mapping relay system outputs to specific light units. System configuration will also consist of routine setup tasks such as placing light units on to the magnetic mounting board. To maintain the system, the primary users will need to make sure the light units are charged before the system is deployed and verify the mapping between the relay outputs and the light units. To move the system, the primary users will install the magnetic mounting board and transport the light modules and host laptop to whatever location the system is to be used.

The secondary users of the system will be industry partners and the general public. The primary interaction of the secondary users will be observing the system and presentations from the PowerCyber lab to learn about cyber-security in a power systems context. There are also interactive buttons on the light modules that allow the general public to "hack" the relay system, which in effect tells the PowerCyber lab that a certain relay was "hacked" and the PowerCyber lab's software decides what relays are affected by that, consequently updating relevant light statuses. Industry partners will also learn how cyber-security principles and intelligent grid management can be used to improve the quality of their operation. Because the secondary users will only be observing the system it's important that the system will have pleasing aesthetics to make sure they're satisfied.

The following case diagram simply illustrates how each user is intended to use the system.

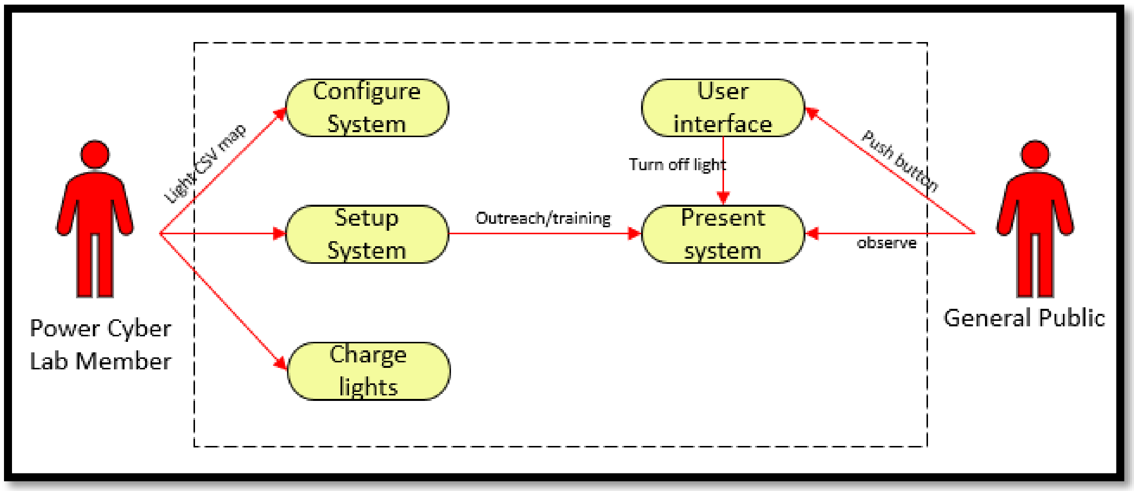


FIGURE 1: CASE DIAGRAM OF LIGHT SYSTEM

1.5 ASSUMPTIONS AND LIMITATIONS

A few things can be assumed about this project both during design and the implementation. The project is funded and supplied explicitly by Dr. Manimaran and ETG. The only people handling the system are PowerCyber Lab members. When the system is in use, internet connectivity will always be present along with a power outlet for the server-to-light hub. Upon completion of the project, maintenance and care of system will transfer to Dr. Manimaran Govindarasu.

Along with assumptions, there are also limitations to the project. One major limitation for the project is time, since we only have two semesters to complete it in total. Individuals outside of those authorized, will be unable to initiate any wireless communication.

Another major limitation for the project is only having one software proficient team member.

1.6 Expected End Product and Deliverables

At the end of the Fall 2019 semester, we will deliver multiple functional light modules enclosed in an aesthetically pleasing housing. In addition to this, we'll also deliver the files necessary to print and assemble as many more light modules as the client wishes.

On the software side, we'll also deliver functional programs to the client that interface with the PowerCyber lab's software as well as our light module hardware.

As users travel to different locations with our system, potentially outside the reach of the PowerCyber network, they will need a way to connect to it so they can grab the light statuses from the server. The hub works as this bridge, and will consist of a laptop with a USB port and the ability to run a command-line C program, connected via ethernet to our server, and connected via serial communication to a ZigBee coordinator. The laptop communicates via the coordinator to each light module the users bring with them.

Each light module contains a ZigBee Xbee module, as well as an LED that the ZigBee module controls. Therefore, the light module will contain some sort of diffusion to make it obvious when the LED is illuminated. These lights modules will be powered by rechargeable batteries, and the outside of the module's housing will have a magnet so users can stick the light module on a large magnetic board for easy demonstration.

2. Specifications and Analysis

2.1 Final Design

The final system is composed of 3 components. The IOT light modules, the light module coordinator, and the software that interfaces between the coordinator and the simulation.

The IOT light modules have hardware on them that allows them to receive ZigBee signals from the coordinator and toggle the status of a three-color LED on board. The light modules also have pushbutton switch that functions as a user interface for observers to interact with the system. By toggling the switch, observers can simulate cyber-attacks on the simulated grid. The light modules are intended to be configured once, using Digi's XTCU application and the "brains" of the application are on the coordinator side. The light modules are also be battery operated and include a micro-USB port for charging purposes.

The light module coordinator has hardware to interface with the IOT light modules via ZigBee and a central laptop via USB. The light module coordinator essentially acts as a signal buffer to put information from the simulation into the IOT light modules and forward "cyber attack" data from to IOT light modules back to the simulation.

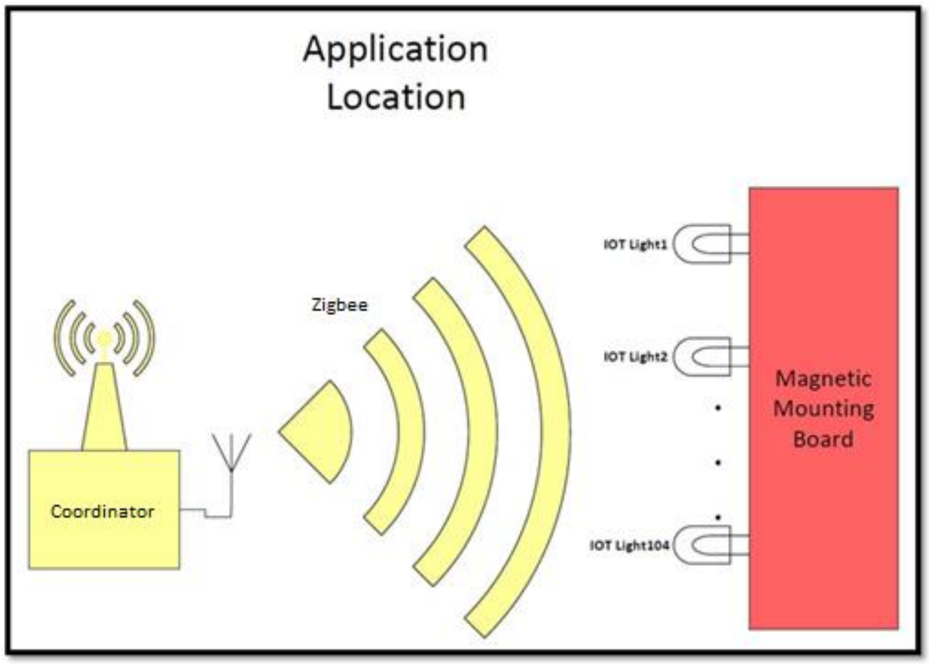


FIGURE 2: COMMUNICATION BETWEEN COORDINATOR AND INDIVIDUAL LIGHTS

The software system consists of a set of CSV files, a C-program, and a Python script. The CSV files hold the status of simulation components and IOT module user interface components. The coordinator connects to the C-program which then updates the user interface CSV file or forwards information from the simulation component CSV file. The python script will put information from the simulation into the simulation component CSV file and send information from the user interface CSV file to the simulation.

2.2 DESIGN ANALYSIS

For the first semester, most of our efforts were focused towards planning, communication of different roles, learning about how to interface the different parts of the system, and writing the associated documentation and making the diagrams.

We ordered many different parts/components to use for this project in order to determine which plan of action would be the most practical. We ordered a few different types of lights. The issue with our design was that we originally planned to buy a DC powered light with Zigbee capabilities, but we have not been able to find one with DC capabilities, only AC powered lights.

We decided to create our own ZigBee lights. This included a ZigBee module (ISM band - frequencies clustered around 2.4GHz), a PCB, a battery, a battery protection unit, an LED, a press button, and simple microcontroller that will all be encased in a single enclosure.

The design of the system consists of three main sub-systems. The first sub system is the communication of data from the server to the .csv file to keep track of the light input for each light module and there is another .csv to track the button status of each module (used to represent a “cyber-attack”) which communicates its data to the server. Any server to .csv or .csv file to server communication is done using python.

The second sub-system consists of the communication between the .csv files and the coordinator. The data held in the light input .csv is sent to the coordinator and the button status .csv file receives the data from the coordinator. This communication is done and setup with C code.

The third and final sub-system consists of the communication of data about the status of the lights and the status of the buttons between the coordinator and the endpoints. This communication is set up using XCTU, the software used to setup the ZigBee modules on the same channel and setup whether the individual module is a coordinator or endpoint.

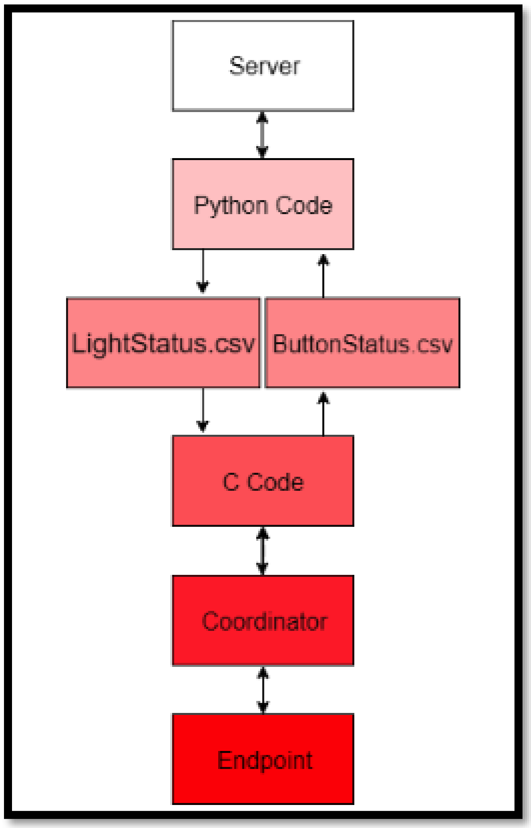


FIGURE 3: SOFTWARE BLOCK DIAGRAM

3 Testing and Implementation

3.1 IMPLEMENTATION DETAILS

As of the end of the Fall 2019 semester, all functional and non-functional testing will be complete and the project will be handed over to the client. At the time of the handover, there will be 3-4 functional light modules, fully housed. The backing software will be completely written and functional. The client will have the files necessary to create as many light modules as they wish, and they will know how to assemble the light modules, set them up for use, and run the system.

3.2 Similar Products

There are a number of similar products out there on the market. We investigated using Samsung SmartThings hubs, Philips Hue light bulbs, and other home automation tools to implement this system. Since these products all used Zigbee, we hoped they'd be able to be integrated nicely into our project.

The biggest issue with these products is the lights. The most readily available lights are AC lights meant to be put into a house. This causes 2 issues: size and power. The size of the lights is the size of a standard consumer lightbulb, and are bigger than the client requested. In addition to this, we would have to make a circuit that converted DC battery power to AC power.

We hoped to search for non-AC lights, but we were in a pretty tight niche where we couldn't find something small, battery powered, and wirelessly controlled. We could find multiple lights that satisfied 2 of those 3 requirements, but none that satisfied all. After concluding that this part of the system couldn't be bought, we decided to build the modules ourselves.

Interestingly, however, the housings of our light modules are made from purchasable wireless puck lights. These lights are controlled by an RF remote, and won't work as intended, but we ended up stripping all circuitry from the lights and using their housing as the housings for our light modules.

3.3 Interface Specifications

The project has 2 main interfaces: The server-to-coordinator interface and the coordinator-to-light interface. For the server-to-coordinator interface, the coordinator is hooked up to a laptop through USB and the laptop reads files off the server. When a text file for the current light status changes, a C program reads the text file, builds one or more XBee packets, and sends it through a serial port to the XBee coordinator. The interface is also able to provide communication in the reverse direction. If a user were to press a button on one of the light endpoints, the serial port reads the coordinator packet and updates a csv file with the MAC address and button status of the updated light.
The coordinator-to-light interface is dictated by the ZigBee standard. When an individual ZigBee module gets a packet from the coordinator, it turns a digital port off or on depending on what the digital data field of the packet says. Additionally, a button on each ZigBee module can be pressed which sends a packet to the coordinator.

3.4 Hardware and software

Below is a list of all hardware and software tools used in the development process and created/used in the final implementation.

3.4.1 Hardware to be used

- Breadboard
    - To prototype all the circuits we designed
- Light Module – PCB

    - We designed a PCB for the light modules that gets printed and assembled, then we house the PCB in the housing we create
    - PCB contains (main functional components)
        - Battery
        - ZigBee module
        - LED
        - Battery protection/charging circuitry
- Light Module – Housing
    - We bought off-the-shelf puck lights and stripped the inside to fit our PCB inside
- Local Hub Computer
    - The local hub computer runs a C-program that reads from the .csv files that the server updates and communicates with the coordinator based on the data in those files
    - In addition to communicating from the server to the coordinator, the C-program also reads button status updates from the coordinator and writes these to a separate .csv file that the server then reads and reacts to
- ZigBee Coordinator
    - The coordinator takes information from the hub computer and C program, and sends ZigBee packets to light modules as necessary.

3.4.2 Software to be used

- Python
    - We will be using a Python script on the server communication side in order to parse csv files that the client reads from and writes to.
- C
    - We will be using C to send and receive packets from the coordinator. Furthermore, the Windows API will be used to facilitate serial communication.
- XCTU
    - We will use XCTU to discover new light nodes on the system so that the coordinator can talk to the nodes. If needed, new light endpoints can be programmed with the software.

3.5 Functional Testing

As each aspect of the project was completed, we needed to test our project thoroughly to ensure smooth transitions and a satisfactory final delivery. When each individual module was completed, we performed unit tests on that module to verify their functionality.

3.6 UNIT TESTING

Unit testing was broken up into two modules: the simulation-to-coordinator and the coordinator-to-light module.

3.6.1 SERVER-TO-COORDINATOR

To individually test the simulation-to-coordinator module, we emulated the output of the python script. Then we send test light statuses to the local coordinator via our C-code which will allows us to verify that the coordinator is receiving the correct packets from the C-code.

3.6.2 COORDINATOR-TO-LIGHT

To test the coordinator-to-light module, we can check that directly sending a packet through XTCU produces the results we would like. Reverse communication was tested by programming the endpoint, pressing the button, then seeing if the coordinator received the correct packet.

3.7 INTEGRATION TESTING

Integration testing consists of putting the server-to-coordinator and the coordinator-to-light interfaces together. In order to ensure that the system preforms as required, we need to make sure that we can send a packet from the program and see the lights turn on or off. Once that is done, we can make sure that the program correctly catches the packet.

3.8 SYSTEM TESTING

System testing will basically be a combination of the two integration tests with the addition of the server files. For a successful test, we need the same exact results as the integration tests.

3.9 ACCEPTANCE TESTING

To see if our client and adviser are satisfied, we will basically perform a system test in their presence, teach them how to use each aspect of the system, and take their feedback. If the user interface of any module is too tricky or ugly, we will take that feedback and if we have time, fix it.

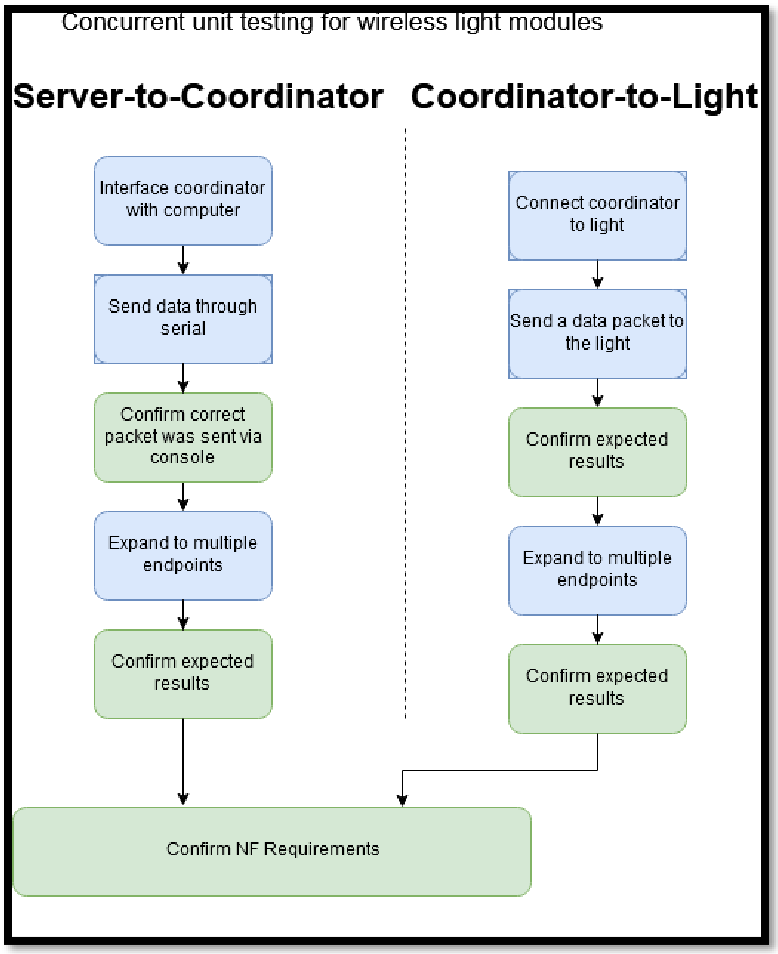
3.10 NON-FUNCTIONAL TESTING

The final non-functional testing of the system happened after the subsystems had been fully developed and integrated. At this point, all compatibility issues were resolved, so we needed to verify the system's usability and performance.

To verify the usability of the system, we had the client work with the system for each use case. During this testing we found several issues, but made plans to resolve them all. The final design delivered to the client will be usable by their standards; therefore, the system's usability has been qualitatively verified.

To test the systems performance, we also had the client work with the system and noted key performance identifiers while performing the unit testing. We found that the system's performance was on par with our design specifications.

3.11 Process



Concurrent unit testing for wireless light modules
Server-to-Coordinator
Coordinator-to-Light
Interface coordinator with computer
Send data through serial
Confirm correct packet was sent via console
Expand to multiple endpoints
Confirm expected results
Connect coordinator to light
Send a data packet to the light
Confirm expected results
Expand to multiple endpoints
Confirm expected results
Confirm NF Requirements

Figure 4: Pre-integration testing for wireless light modules

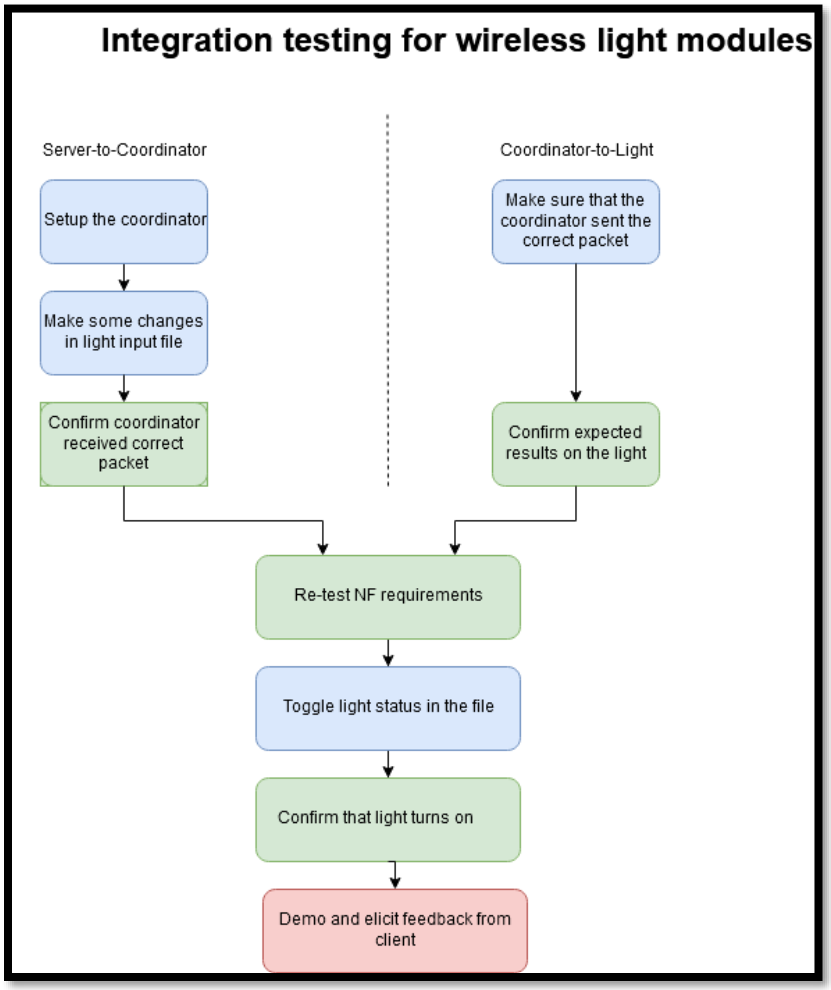

Integration testing for wireless light modules
Server-to-Coordinator
Coordinator-to-Light
Setup the coordinator
Make some changes in light input file
Confirm coordinator received correct packet
Make sure that the coordinator sent the correct packet
Confirm expected results on the light
Re-test NF requirements
Toggle light status in the file
Confirm that light turns on
Demo and elicit feedback from client

FIGURE 5: INTEGRATION AND POST-INTEGRATION TESTING FOR WIRELESS LIGHT MODULES

The figures above illustrate how we plan on testing our proposed solution. To test our relay to server communications we first tested the circuits that communicate from the relay to the transmitter junction box. Then we tested the communication scheme for the transmitter junction box to the server. To test the server, we used scripts to emulate the expected input from the transmitter junction box and verify that the light statuses in the server changed as expected. To test the server to light communication scheme, we manually manipulated the light statuses in the server and verified that the server to light communication module changed the physical light statuses. Finally, we put everything together and used switches to emulate the relays. By toggling the switches, we can verify the functional performance of the system. Unfortunately, it takes a while for parts to get here so we are still working on testing our proposed implementation method for each subsystem. As this testing progresses and we validate/invalidate various methods it's possible that our testing process will also change.

3.12 Results

3.12.1 Python Script Test Results

The python script was developed by the client and one of their grad students over the course of summer 2019. During this time, it was verified that the script was functionally capable of updating a CSV file with the status of simulation components.

3.12.2 Light Unit Testing

To test the light units, we used DiGi's XTCU software to create experimental configurations for the ZigBee light modules and the zigbee coordinator. By using the coordinator to send experimental data frames, we were able to verify the functionality of the LED circuits used on the light module. By pushing the button on the light module, we were also able to verify that the coordinator is receiving data packets from the light module's user interface circuits and that there will always be a packet sent whenever a button is pushed. Testing the battery charging circuit was tricky, as we didn't have any constant voltage electrical loads that we could use. We had to design a circuit that would allow us to verify the performance of the chargre. The circuit we used can be seen below.

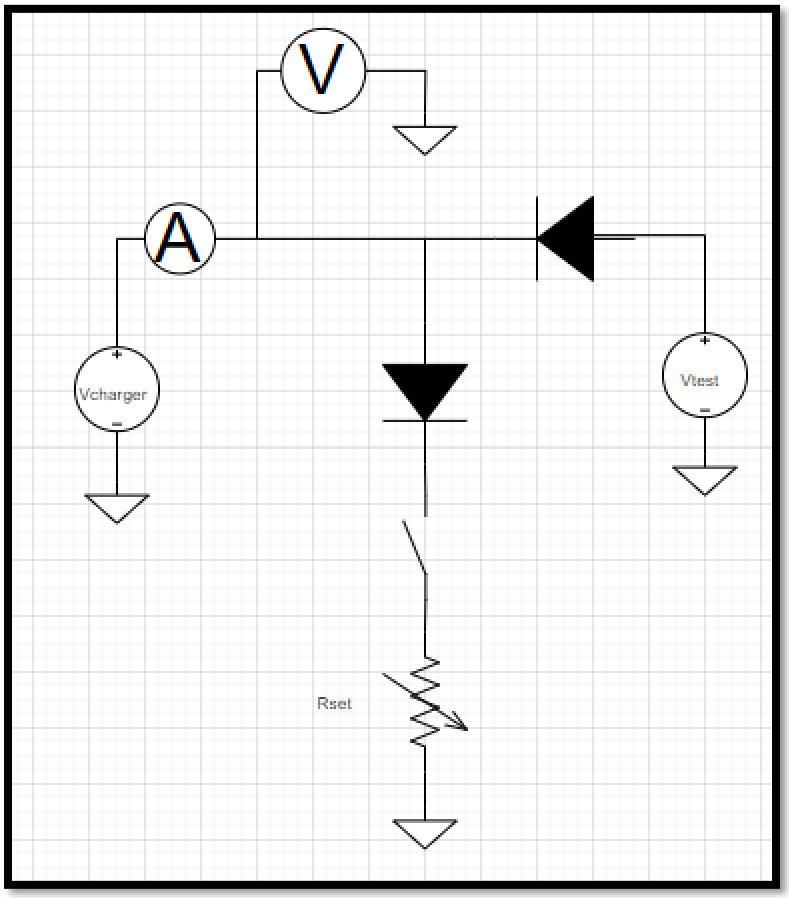


Figure 6: Battery Charger Test Schematic

bTo use this circuit, we used Vtest to set the charging circuits output voltage, and Rset to simulate the impedance of the battery. Using this circuit, we were able to verify that the charger outputs 0.5A over the range of charging voltages and as the battery's impedance increases the charge current will lower. With these tests, we were able to verify the light modules performance.

3.13 Implementation Challenges and Issues

Some of the biggest challenges we faced were lead time, communication, and our background knowledge.

Lead time is a challenge for every team, but towards the end of the fall semester we struggled with the lead time of PCB manufacturing. We wanted to be able to push 3 revisions of boards through with enough time to test and fix issues on our boards before we ordered new boards. Due to the underestimated length of lead times, we were only able to see 2 revisions through, and the final revision was handed off to the client for them to have printed themselves.

Communication was difficult on multiple fronts. In the beginning, it was difficult to elicit requirements clearly enough for us to come up with a comprehensive plan for the project. As the project progressed, we didn't do the best job of communicating the current status with the client and adviser, which resulted in work often being thrown away because we assumed the client's opinion and continued working.

Our background knowledge was the final large challenge. Entering the project, we had no software engineers, but our computer engineer had to learn multiple extra languages and interfaces to effectively complete the project. In addition to this, none of us knew anything about ZigBee, so we had to learn how that communication worked as we designed the project.

4 Appendix I – Operational manual

4.1 Necessary Items / First-time setup

To set up the system for a demonstration, you'll need the following components:

- A hub computer with a USB port and the ability to connect to the internet
- As many light modules as you want to bring. We'd recommend bringing a few extra in case batteries are dead or some are non-functional
- A ZigBee coordinator with a USB cable to connect to the hub

XBEE S2C Module setup with XCTU:

(Note that all setup directions for setting up a coordinator module and endpoint modules can be found at the digi website: https://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf)

Digital Pin on XBEE	LED Color/Use
DIO0	Red LED

DIO1	Green LED
DIO2	Blue LED
DIO4	Connection to Toggle Button

TABLE 1: DATA PIN TO USE MAPPING

4.1.1 Protocol Used:

We used the **ZB** protocol instead of the **802.15.4.** This allows for the Digi Box (Coordinator), which uses the ZB protocol, to be the coordinator versus the laptop with ZigBee coordinator.

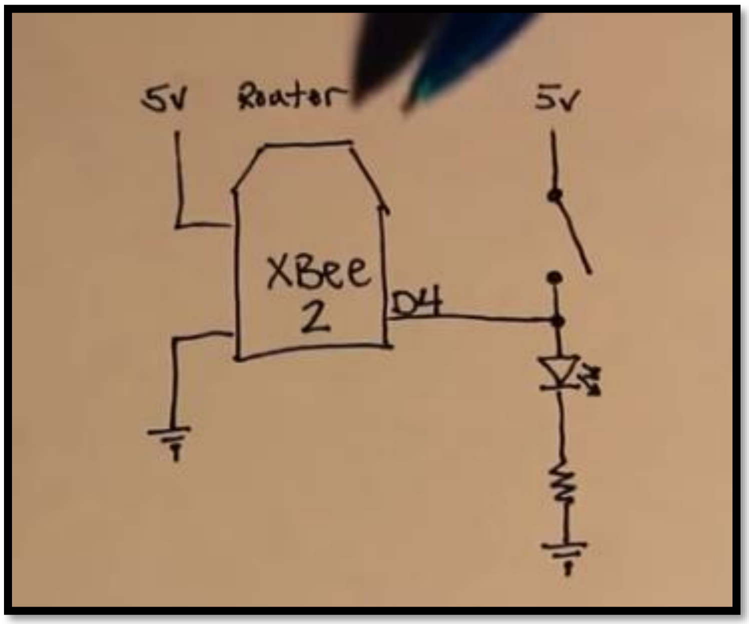


FIGURE 7: DIAGRAM FOR TOGGLE BUTTON CIRCUIT CONNECTED TO THE ZIGBEE DEVICE

DIO4 value [3] makes it an input for the toggle button.

4.1.2 How to observes changes to the toggle pin

To observe changes to the toggle pin (DIO4) we write the IC parameter of the Endpoint to 0x0010 (same as 0b0001_0000)

AT- (attention commands) want to be able to use these to read the state of a pin.

Pin Configuration for XBEE S2C Module:

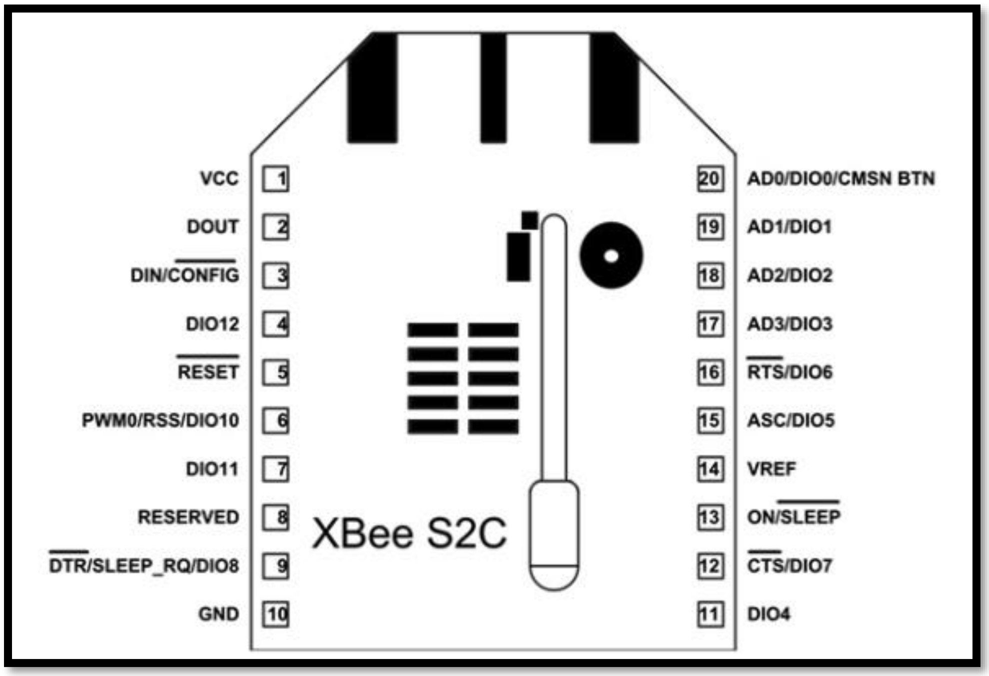


FIGURE 8: XBEE S2C PIN CONFIGURATION

https://www.digi.com/resources/documentation/Digidocs/90001456-13/tasks/t_obtain_data_from_sensor.htm

4.1.3 Automatic sampling

Once you have set up the pin, the remote module must be configured to automatically transmit the sensor information to the main XBee module. The remote XBee module needs to know:

1. Where to transmit the sensor data: define this information for the module receiving this information by the destination address (**DH** + **DL**) parameters.
2. When to transmit the sensor data:
   - Periodically: The XBee can send the information read from the sensor at a specified interval.
   - By change detection: When a pin or several pins change status.

- Configure parameters IO Sampling Rate (**IR**) and Digital IO Change Detection (**IC**) to automatically transmit the sensor data.

Note These two features can work in combination with each other, depending on your requirements. For example, you could choose to receive an IO sample every minute (**IR**) but also when a certain pin changes state (**IC**).

Digital IO Change Detection (IC)

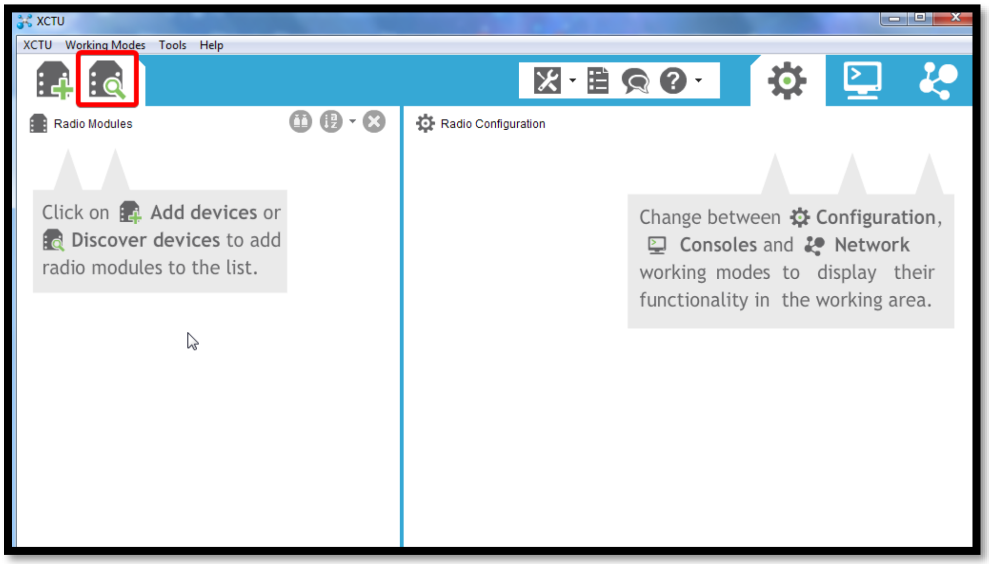
The **IC** parameter allows you to set which pins to monitor for change detection. When the state of the monitored pin(s) changes, a sample is immediately sent to the destination address.

4.2 Setup

4.2.1 Coordinator Setup

When you have everything you need on-site, you need to connect the coordinator to the endpoints. To do that:

1. Ensure that every endpoint is turned on
2. Plug the micro-USB into the port on the coordinator
3. Plug the other end of the USB port into a free port on the computer
4. Open up XCTU
5. Click discover devices



Click on Add devices or Discover devices to add radio modules to the list.

Change between Configuration, Consoles and Network working modes to display their functionality in the working area.

Figure 9: XCTU Interface

6. Select the COM port the coordinator is on and Hit "Next"
7. Confirm that the baud rate is set to 9600 with 8 data bits, no parity, and one stop bit.
8. Wait for XCTU to scan the COM port on the computer. If a coordinator is found, click "Add Selected Devices"
9. On the left side of the screen next to the coordinator details, click the center button
10. The coordinator will now scan for devices in the personal area network. If any are found, click “Add Selected Devices”
11. Make note of the endpoint MAC addresses

4.2.2 Setting up the program

1. Open up Cygwin
2. cd to where the program is installed

3. Type gcc-o <your exe name> forward.c, replacing <your exe name> to what exe you want gcc to compile the code to.
4. Repeat step four for reverse.c
5. Open up mac_addresses.txt and enter one MAC address of each endpoint per line, ensuring that there is a blank line in the end
6. Open up LightStatus.txt and input one digit (0-8) per MAC address.

4.3 Running

Once the system is set up, the operation is fairly simple. Monitor the battery level of each light, and if any light seems to be low on battery, turn it off and charge it. Other than that, pressing the button on the side of the light module will simulate a "hack" on that specific power station and this will be communicated to the PowerCyber lab's system. Note that the button is a toggle button, and be sure to toggle the button back to off if you want to simulate the power station being fixed.

4.4 Teardown

After a conference or session of use, there are a few tasks that need to be done to make sure that the system is properly stored and ready for the next demonstration.

1. Turn off all light modules via their main power switch
2. Unplug coordinator from hub and store properly
3. Plug a micro-USB cable into any light used in order to charge it. Monitor the battery temperature and the status LEDs. Red LED signifies that the battery is charging, and a green LED signifies that the battery is fully charged
4. As batteries reach full charge, unplug them and store them in a safe, temperature-controlled space

5 Appendix II – Other Versions of design

5.1 ISM Band Prototype

In the first semester of the project, we were not confident in our knowledge of the ZigBee communication protocols and our ability to learn them in a timely manner. In an effort to keep progress on the project moving, we worked on ZigBee in parallel with another implementation that used the ISM band. This prototype used an Arduino variant with a wireless communication module called a Lora. The Lora communicates over the ISM band, and one of the members of the team had experience using it with Arduino. We delivered 2 prototypes of light modules that could be controlled by a third Arduino that acted as a hub. Below is a picture of a demo we created.

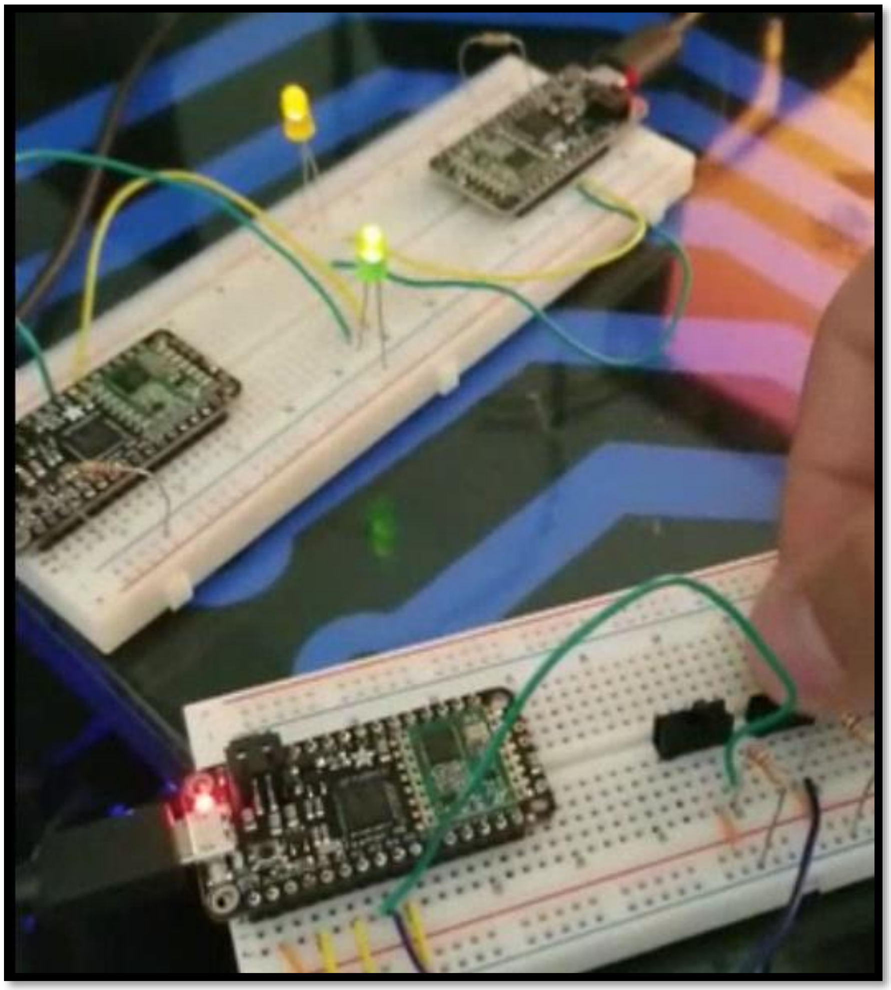


FIGURE 10: ISM BAND PROTOTYPE

In the picture above, the top breadboard is 2 separate light module prototypes. The big circuit board is an Arduino variant with the Lora built in, and then the LEDs are connected to a data port. The bottom breadboard is the same Arduino variant, but instead of LEDs having switches on the data ports. At the time of building the prototype, we didn't know the exact implementation of the data changing, so we used switches to simulate the data updates so we could demonstrate functionality.

In the video this picture is taken from, we show that flipping 1 switch on the lower breadboard results in a change in light status on the corresponding light module within a second or two.

This prototype was eventually halted because the client wished for the project to use Zigbee, as the client has plans in the future for projects involving ZigBee.

6 Closing Material

6.1 Conclusion

With the current setup in the PowerCyber lab, the existing lights are inadequate in number and in representation. Relay boxes output 8 signals and the current setup is directly hard-wired to the light units only. It is difficult to manage and operate this hard-wired setup as it scales up. Also, it limits aesthetic and informative visualization of these light units with reference to a power grid map as they are not flexible to move as well as improper space to project a power grid map underneath the light units.

Our project, in an effort to fix the two aforementioned problems, will be a new system where there is a large magnetic board with a geographical map projected onto it. Users will place a wireless-enabled light at a point on the map that they want to represent a certain power station. They will take the ID of that light and tell our system which power station (meaning which relay from the PowerCyber lab's system) that light corresponds to. Our system will take in the current status of that power station from the relay boxes in the PowerCyber lab, send it wirelessly to a local hub at the presentation site, and update the status of that power station with the corresponding light. There will also be reverse communication, where the user can press a button on the physical light module, and the light module will communicate that button press to the PowerCyber system, which simulates a "hack" on that specific power station. This will allow presentations to simulate a hack in one power station and show the effects of that power station being compromised on other power stations.