IoT-SmartLights: Remotely Controlled Advanced IoT Smart Lights Hub Prototype

FINAL REPORT

Sr. Design December 2021 Team 20
Client: Govindarasu Manimaran
Adviser: Gelli Ravikumar

Team Members and Roles:
Hamza Mostafa Sayed Mahmoud - Hardware
William Gavins - Hardware
Xinlei Yu- Software
Nathan Orts - Software

Team Email: sddec21-20@iastate.edu
Team Website: https://sddec21-20.sd.ece.iastate.edu

Final edition: December 18, 2021

Executive Summary

Development Standards & Practices Used

This system will use the Zigbee wireless communication protocol for communication between the smart lights and the Zigbee controller.

Summary of Requirements

- Lights must be able to operate continuously for 8 hours.
- Lights must be able to change to at least 4 different colors.
- Lights must operate wirelessly while being controlled from a host computer connected to the PowerCyberLab simulation.
- Latency between a relay status change and light status update must be less than one second.
- Light enclosures must be entirely self-contained.
- Software interface between the user and the light system must be easy to use and understand.

Applicable Courses from Iowa State University Curriculum

- CPRE 281, 288, 430, 450, 489
- COM S 309 & 319
- EE 201, 230 & 333

New Skills/Knowledge acquired that was not taught in courses

- PCB design is by far the most important skill we are to acquire outside of class.
- Embedded IoT communication protocol (i.e Zigbee) usage is also another skill we are to learn
- Programming languages such as Python, HTML, CSS
- Programming tools such as Dajongo

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

We would like to express our sincere gratitude to Dr. Gelli Ravikumar as our advisor and Dr. Govindarasu Manimaran as our client and secondary advisor. They both provided invaluable guidance, feedback, and resources towards the successful completion of this project.

1.2 Problem and Project Statement

- Problem Statement

Currently, when running power grid simulations in the PowerCyberLab in Coover Hall, the status of relays within the simulated power grid is shown by twelve lights mounted on the hardware running the simulations. In the past, when doing outreach demonstrations or presentations, there would be a livestream of the lights via a video camera in the lab. Though this worked, it is hard for viewers to visualize the simulated power grid and the status of the relays when the status lights are fixed to the hardware, rather than being arrayed over a map of the simulated power grid.

- Solution Approach

Our goal is to provide a way to more easily demonstrate the power grid simulations in outreach presentations. We would like to use wireless, magnetic lights in an internet of things environment such that the simulated power grid can be projected onto a surface and our lights can be placed where the relays are located in the grid, providing a better visual aid for the demonstrations.

The deliverables for this project are a wireless, magnetically mountable LED light, a Zigbee coordinator that controls the lights, and software that configures the lights to fit the current simulation and transmits the relay statuses to the lights to update the demonstration in real time.

The Light display system needs to have enough nodes to fit the simulations possible within the PowerCyber lab, as such this network scheme for the IOT lights needs to be highly scalable. Each node will be able to be configured to represent individual relays in the PowerCyber Lab

1.3 Operational Environment

Our wireless light system will be used in indoor presentation venues, such as classrooms, lecture halls, or electronics labs. Our system must have access to the internet so that it can communicate with a server on campus that serves as the intermediary between the light system and the power grid simulation in Coover hall.

1.4 Requirements

Previous team's requirements	Our team's requirements
Functional Requirements ➢ Accept input from 100 relays. ➢ Output to 100 IOT lights. ➢ Capability to map relay inputs to light outputs via UI. ➢ IOT lights should be magnetically mountable. ➢ IOT lights should be able to operate continuously for 8 hours. ➢ IOT lights should come in at least 4 different colors. ➢ IOT lights must be operated wirelessly. ➢ System should be capable of charging at least 8 lights at once.	Functional Requirements: ➢ Host PC communicates with PowerCyber Lab Simulation. ➢ Hardware Coordinator communicates with the host PC. ➢ IOT Light devices communicate with the Hardware Coordinator. ➢ Light devices are configured on the host PC to represent relays or nodes in a simulation. ➢ Light device shows the state of a relay or node in a simulation using an RGB LED. ➢ Light devices are battery powered, with a battery life of eight hours or more. ➢ Light devices must be operable while charging. ➢ Light devices must be operated on a scalable network topology. ➢ Light devices are able to send integer data to the host PC to trigger an attack on their respective relay.
Non-Functional Requirements ➢ System should be easy to move through standard door frames. ➢ Magnetic mounting board should not obscure any image projected onto it. ➢ Light status should update in less than a second when the corresponding relay is updated. ➢ Relay interface components should be easy to connect. ➢ IOT lights should be similar in size to existing solution(s).	Non Functional Requirements ➢ Light enclosure is aesthetically pleasing and relatively small. ➢ Light enclosure must be entirely self-contained. ➢ All system components together must be portable enough to fit through doors. ➢ Access to white board or magnetic board as a mounting surface ➢ Mounting surface should not obscure any image projected upon it. ➢ Light devices update to reflect changes in the simulation in less than one second . ➢ Light devices must be magnetically mountable. ➢ Relay interface components must be easy to connect. ➢ Software interface between the user and the light system must be easy to use and understand.

(Table 1 list of requirements)

1.5 Intended Users and Uses

This system is intended for use by members of the PowerCyber lab at Iowa State University to visually represent the status of relays in power grid simulations, though the system could be easily adapted to visually represent of the status of nodes in any graph-type system, such as city streets, wireless networks, or supply chains.

1.6 Assumptions and Limitations

Assumptions:

- Users can read and understand english.
- Users have their own computer(s) that can run our software.
- Users are in an environment with an internet connection.
- If outside of the Iowa State University campus, users will have access to the Iowa State University VPN.
- Users have access to electricity when needing to charge the wireless lights.
- The system will be used indoors and will not be exposed to the elements.
- The system will be kept dry at all times.
- The colors for the following statuses will be: blue for functional, yellow for <intermediate>, and red for non-functional or non-responsive.

Limitations:

- System is capable of two way communication.
- System must be chargeable and usable at the same time
- System in its entirety must fit through a standard door.

1.7 Expected End Product and Deliverables

The primary deliverables for this project are the wireless LED lights, the software that runs on the user's computer which is used to configure the system, and the Zigbee coordinator that acts as the intermediary between the software and the wireless lights. We will also include a way to charge multiple lights at the same time, in the form of a charger hub.

2 Design and Implementation

2.1 Previous Work And Literature

Our project is an enhancement of a previous team's work (https://sddec19-16.sd.ece.iastate.edu). Our purpose is making the design more usable by adding more features and enhancing the communication protocol and battery time. The previous team was able to set up a one way communication using Zigbee protocol and Xbee chips. They designed the PCB and the housing of the Smart light and was successfully mountable on a white board using magnets.

In the previous team's design the battery only lasted for 8 hours. Another large drawback for the hardware was that it did not fit well inside the enclosure, which got in the way when a user needed to charge the device. The device had a physical switch that needed to be toggled to switch from charge and discharge modes, which made it harder to use. The device also only had one RGB light for presenting data, with very little variance in color selection. One final issue was the lack of user input. Users were unable to interact with the active simulation represented with the lights, which calls for two way communication back to the simulation. This is not to understate the fact that the previous team successfully created a device which could functionally represent the state of the simulation, rather to outline some potential targets our team is looking for improvement on the project.

(Figure 1: Previous team's light module)

2.2 Design Thinking

Our Design uses the Zigbee wireless communication protocol for simulating processes running in different locations. In our design thinking, group members suggested using different protocols, such as LoRa and Z-Wave. Another proposed idea was to use Bluetooth Low Energy instead of a predefined IOT protocol. However, we found ZigBee to be the best choice as it consumes the least amount of energy and allows a wider range of devices to be connected on its own network.

	Owner	Frequency (MHz)	Range	Power requirement	Security	Compatibility
Zigbee	Zigbee Alliance	868 - 868.6 (Europe) 902 - 928 (US)	10–100 meters line-of-sight	Low-Power, Potential Batteryless	Low, basic encryption	Compatible across Zigbee devices. DotDot OS.
Lo-RaWan	LoRa Alliance	169, 433, 868 (Europe) 915 (US)	Up to 6.2 miles or 10 km.	Low-Power	Basic 64-128 bit encryption	Depends on OEM
LTE-M	GSMA - Cellular Carriers	LTE Bands: 450-2350 (uplink)	Global	Band dependant	NSA AES-256	Application dependant
IEEE 802.11af (White-Fi)	Open - IEEE Certified	470 - 710 (Digital Dividend)	Short, up to 100m	Low	WPA	Application dependant
IEEE 802.11ah (HaLow)	Open - IEEE Certified	850 (Europe) 900 (US) 700 (China)	Up to 13 miles or 20 km.	Medium	WPA	Application dependant

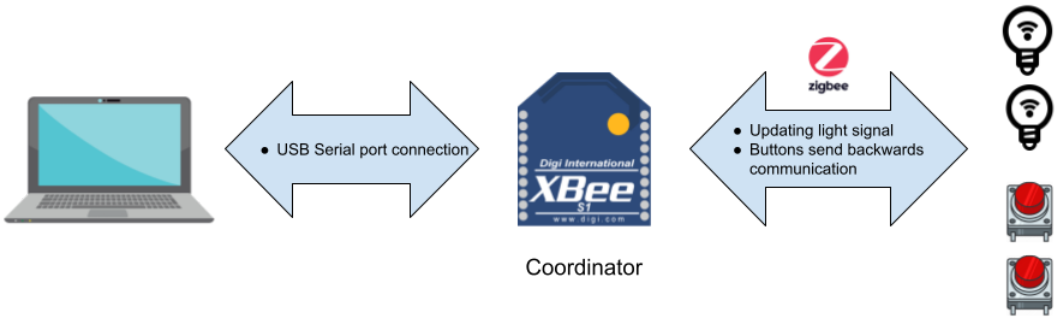
(Table 2: Wireless communication protocol comparisons)

Our team also came up with the idea of using smart batteries that have a built-in charging circuit. Another approach was designing our own circuit and using another battery with more capacity. The team decided to go with the second option. One of the limitations of our project is size and therefore we will be using a small battery. We reorganized our components on the PCB in order to accommodate a battery charging circuit. Power usage can be a lot higher in a situation where the smart light modules are transmitting and receiving information rather than just receiving information. As such, power consumption was weighed more heavily than in the previous context.

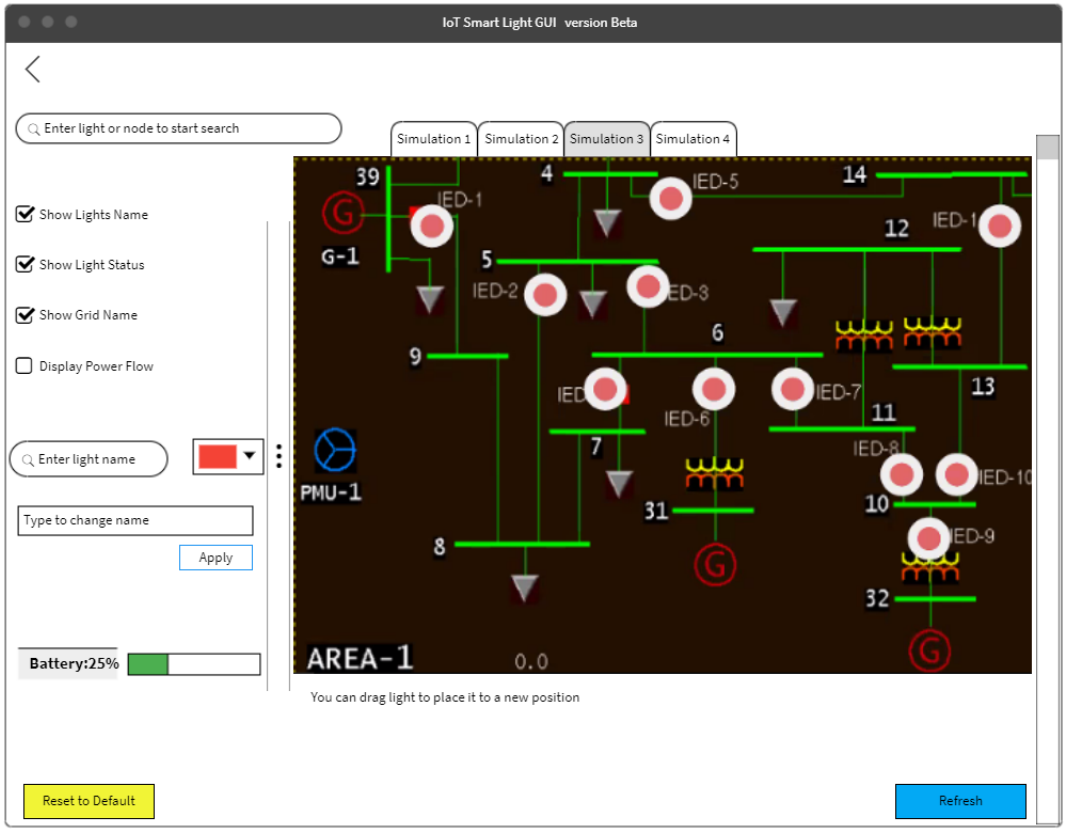
2.3 Proposed Design

When approaching this project, we divided our requirements into two categories, the core requirements of which we're aiming for, and the non-functional requirements of which we must satisfy. Each of these requirements was used to help arrive towards our prototype platform conclusion.

From these requirements we decided to move forward with Xbee 2 as the platform from which we develop our smart lights. The Xbee 2 is capable of several types of network topologies which allow for network scalability and flexibility for the number of devices during a PowerCyber simulation. The Xbee s2c features an implementation of Zigbee, which can be used for low power wireless communication. For now, the smart lights are broken up into singular nodes, and a coordinator connected to a host PC is used to handle the inputs and outputs of the smart lights network and interface with the simulation.



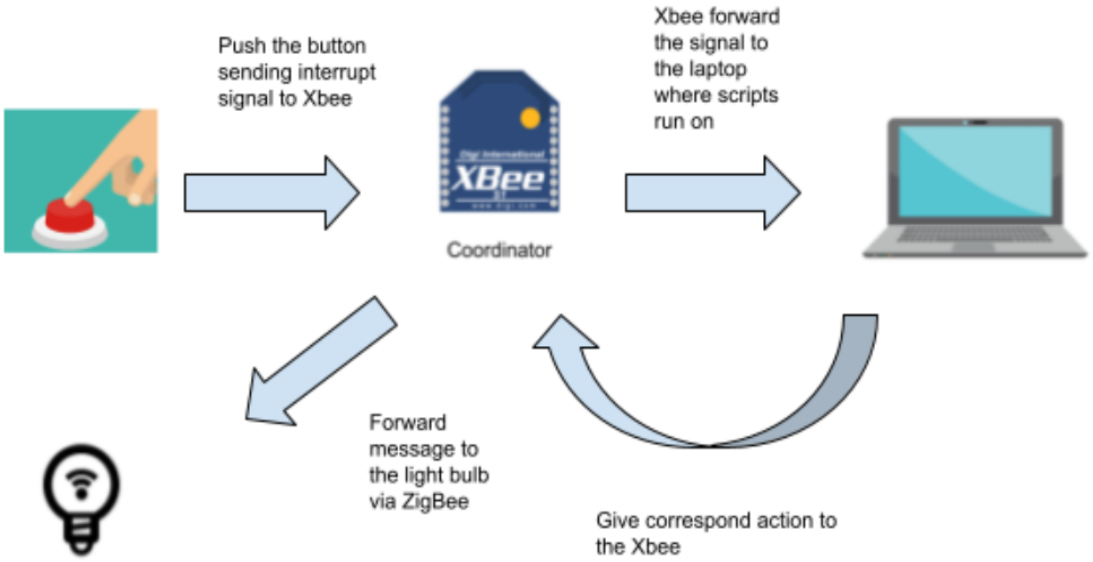
(Figure 2: System diagram of IoT devices and host machine)



(Figure 3: Proposed GUI page for showing simulation)

2.4 Coordinator

The Coordinator is responsible for receiving and sending information to the host simulation consisting of status updates from the smart lights that would be triggered by an end user, as well as the current status of the relay that any given smart light may represent during the simulation. This Coordinator communicates with the host PC software via usb serial, and communicates with the xbee 2 based nodes via the Zigbee wireless protocol.



(Figure 4: Coordinator)

2.5 Smart Light Endpoints

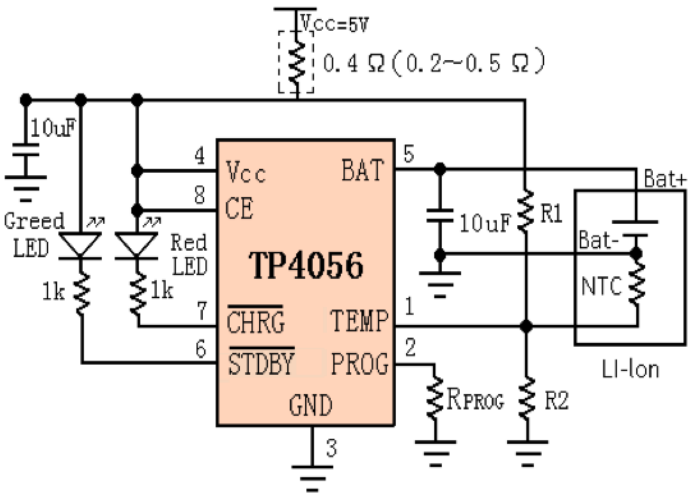
Originally the design was focused on taking advantage of the programmable Xbee s2c's that supported micropython, However, it became apparent that the hardware that was already owned by the powercyber lab lacked this functionality. Because of this we decided to implement the project using the existing hardware from the previous revision of the modules. The tradeoff was a more complicated workaround through the Xbee python API, but with a simpler 6 pin RGB LED.

The smart lights work very similar to the coordinator, and are also based on the XBee 2 platform. Each node, powered by a Digi Xbee s2c, has access to the full zigbee stack, and XTCU Configurations. Based on the out of the box XBee firmware, python code written on a host PC can communicate with the coordinator, which then communicates with each other Xbee module on the Network.

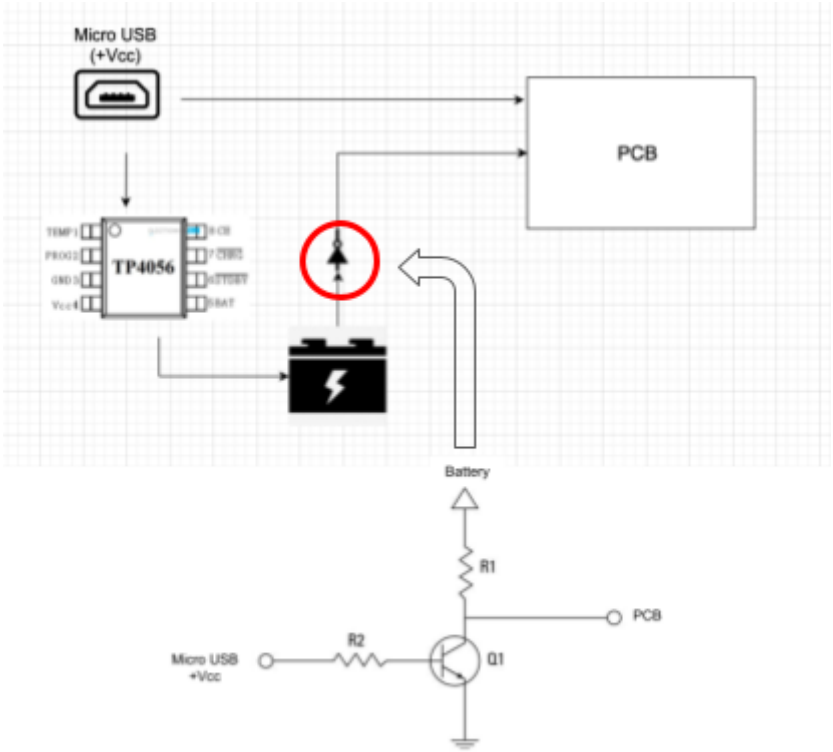
In addition to this, we're also aiming to improve the battery charging circuitry. In its current state the charging circuitry offers charging only after the toggling of an onboard switch, meaning that charging requires hardware teardown for PCB access, and gates the user from using the node during this time. Using a sophisticated hardware design that allows for concurrent usage is ideal, and several transistors as well as a mcp73832 are used to satisfy this requirement. Addition of hardware buttons and a dial are also needed for the end users to simulate a "hack" on the given relay, this requires a dial to select what type of attack is to be simulated, as well as a button to actually trigger the update to the back-end server.

2.6 Charging circuit

We initially designed our charging circuit using the TP4056 IC (figure 8). The IC will help both charge the battery and indicate if battery charge is full. We will also implement an automatic alternating design as shown in figure 8 using a not gate in order to alternate the power source between battery and USB while the battery is charging.



(Figure 5: circuit diagram)

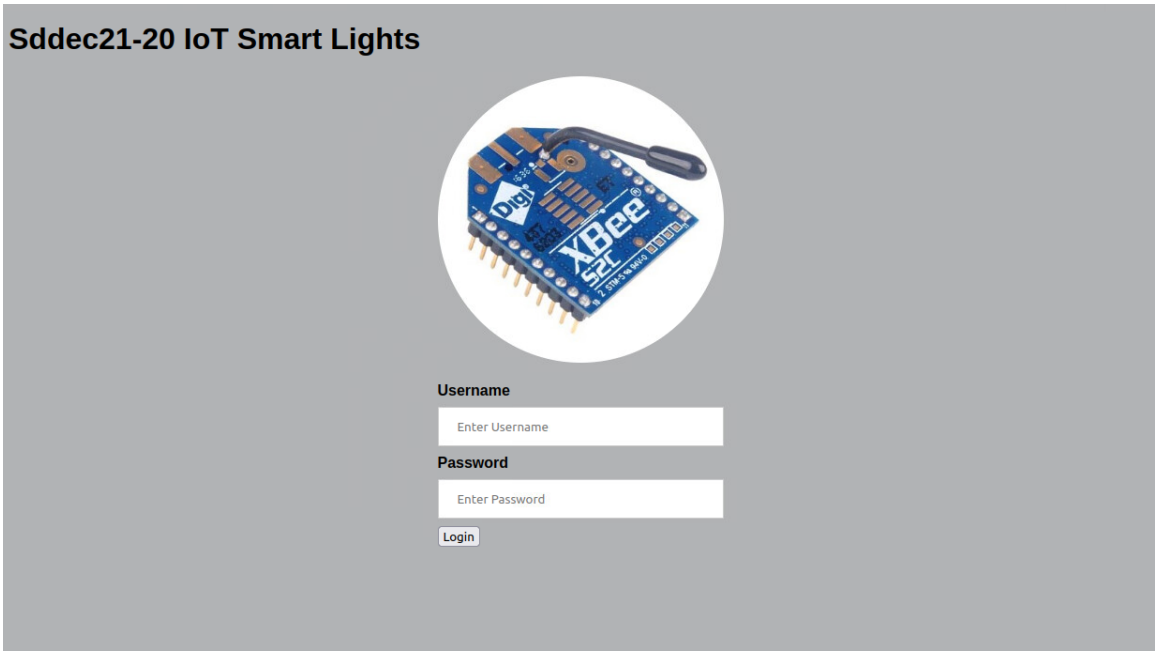


(Figure 6: charging circuit diagram)

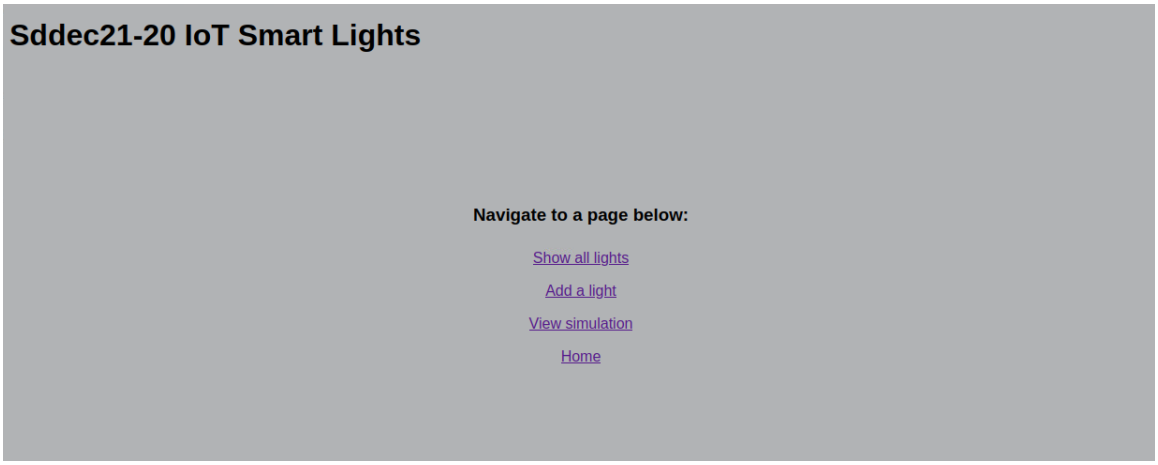
2.7 Wireless Charging

In relation to the initial charging circuit we also discussed the addition of wireless charging, which would simplify the charging using both an inductive charging station and the addition of an inductive charging circuit on the smart lights. This would work on top of the previous circuit. The largest issue here would be the voltage requirements needed by the microcontroller and the rest of the devices. As it stands the battery requires 4.2 V to charge, which puts any 3.3V Inductive charging set out of bounds. The next solution would be using a 5V circuit with a power step down module. This addition was added later on in the design phase, which can pose some issues but research will be done prior to the implementation to determine if this is feasible. This would require some hardware added to the base of each smart light as well as a basic table fitted with the correct inductive charging circuit and a power supply.

2.8 Graphical User Interface



(Figure 7: Smart Light GUI diagram)



(Figure 8: Smart Light GUI diagram)

The GUI allows the user to configure the IoT light devices and upload an image of the simulated power grid. Users must log in to view any of the pages other than the login page. It is integrated with the control software so that the light module database can be accessed and modified without accessing the command line. Users can add a light module or view all the light modules in the database.

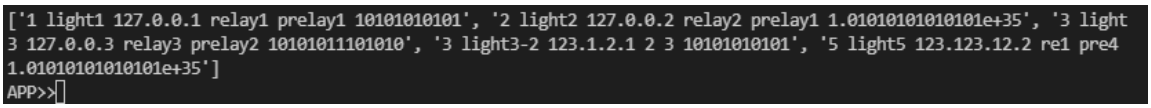
The GUI was designed using Django, a python-based web development framework. Django is similar to ReactJS in that HTML code is served to the browser but python (for React it is java) is used for the work in the background.

2.9 Control software

Control Software/Database allows easy configuring setups, manipulating mySQL database, and interfacing with the Coordinator Module and APIs for GUI to store, retrieve and etc. data from the database. The Control Software uses Python as a programming language and Visual Studio as an IDE. The main function runs a while loop and input function that can pull user's inputs from terminal/command line like a shell. Maintainers can interact with the shell program with built-in commands to manipulate the control software and database. The built-in functions include status, getCfg, print, select, add, add-file, delete, create, and etc. Once you launch the program, it looks like below.

(Figure 9: Control Software Demo)

The initial plan was using MySQL as a database management system; however, MySQL is kind of heavy for our project and not easy to integrate with other components. A better solution would be using Python Sqlite3 library, a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a SQL query language. In addition, it is more joyful to write APIs with Python sqlite3 to provide database functionality for GUI and light modules in our project. An example data field stored in the database includes: light id, light name, light relay, parent relay, and lightstatus/data. Example database outputs shown below.



(Figure 10: Database Interaction Output Demo)

2.10 Technology Considerations

The proposed design will be able to simulate a running process in a different location using multiple smart lights. The design still lacks the ability to run for longer than 8 hours, however with adding the charging and using at the same time feature, we will be able to overcome this issue. Some trade-off has to be made as the PCB is very limited in size as per the client's requirements, and therefore a limited amount of features can be added. Such an issue will be overcome by getting more creative with different ways to present data using only an LCD screen or multiple RGB lights.

The design will have the ability to perform two way communication which will allow us to add more features.

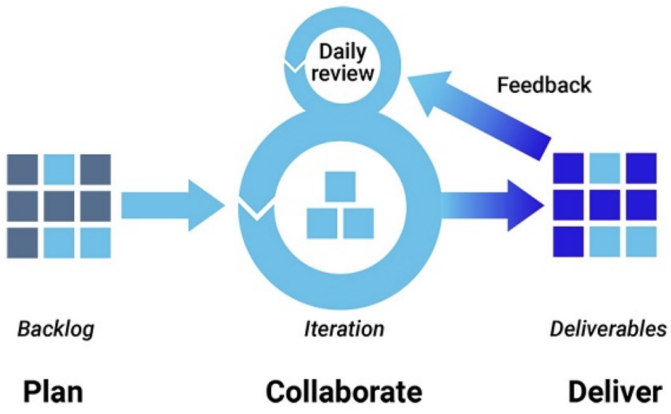
2.11 Design Analysis

Thus far, most of our efforts have been 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.

When looking at the previous work we decided to try to iterate on the design, as we agreed that using a lot of the same core hardware would yield better results with some slight changes to the design. As a team we agreed that some improvements could be made, especially with the previously stated issues regarding the lack of two way communication between the IOT smart lights and the coordinator. This requires additional buttons and a dial. Another aspect is more configurable rgb lights for the xbee modules, through the usage of ws2812bs, which are a bright module that is capable of 255^3 colors, a significant improvement from the static nature of the last version.

2.12 Development Process

Our team followed the Agile development process. We used this process as our team needs to be able to react smoothly to changes in requirements and features. Therefore, throughout the semester, at the beginning of the week we planned what we would do for the week and at the end of the week we would meet to discuss what we had accomplished and what still needed to be done. We discussed what challenges each person had and how we could work together to overcome those challenges. We would then, over the course of the next week, iterate on those issues and try to solve them.



(Figure 11: Agile development process)

3 Testing

Testing is an extremely important component of any project, whether it involves a circuit, a process, or software.

The project was divided into phases, such that each phase would have time to debug and catch any large time consuming errors that could hold the team back. Because of the nature of IOT, there were several hardware design revisions. Getting some level of debugged hardware prototype functioning was the first step for testing on all levels.

Testing depended on what modules of software and hardware were complete, as well as what revision of hardware was currently functioning. This iterative process allowed changes to be made to hardware to prevent lockups in the development process.

3.1 Unit Testing

Tests were broken down into units to emphasize modular testing, this way the team could debug stages one level at a time to prevent confusion. This allowed software to be broken up for easy testing

Each module was tested individually before integrating into more complicated sets. Between each unit test there were interface tests to ensure that the two elements were communicating correctly.

After a PCB Prototype had been fabricated, it was important to ensure that the printed and soldered prototypes were matching the circuit diagram completely. This required testing each connection and component to the parts specifications and verifying the design to the original specification.

3.2.1 Smart Light Modules

Each revision of the endpoint modules needs the following verified:

The module should be able to send data to the host regarding its updated state based on a button press. This will send a packet to the server via zigbee. Testing is needed to ensure that ranges are clear and packet loss is minimized.

The module can receive data from the host based on its changing state from the backend simulation.

The module will also be tested on whether or not the battery can charge and what the discharge rate of the module is.

The module must be able to correctly cycle through all possible colors on the LED. For example when the module is currently red, it can then change the color to any other color.

In the early phases this will be done on a breadboard to make sure that hardware is both functional as well as modular for rapid design changes. Testing the functionality is repeated for each iteration of the PCB.

Early stages of testing will involve this module as well as the Coordinator Module to test wireless communication and scalability.

3.2.2 Coordinator Module

The broadcaster module designates the iot smart light modules to reflect the simulation that takes place within the PowerCyber Lab. In order to prove that the module works when a signal is received, a simulated signal will be sent to show that the backend updates based on the input. Ensuring that the module can trigger changes in the backend is the foundation of the two way communication needed for a successful deliverable. Simulated functions used to make it "appear" that a signal is received or sent will be used to mock test this functionality.

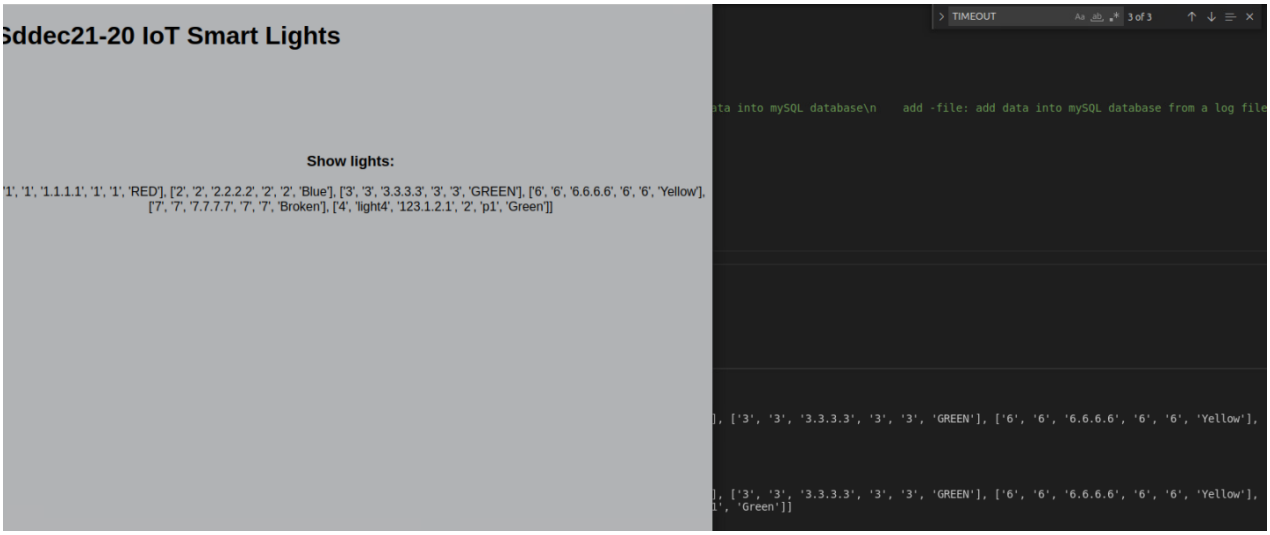
This Module will be connected to a host PC, and will be sent information regarding the intended status of the Smart Light Modules over serial communication, and will communicate updates through the same interface back to the host PC.

3.3 Software Testing

To test the functionality of the GUI we ensured that it would function on both Linux and Windows systems. We used the integrated tools within Visual Studio Code to run tests on the GUI. For testing the connection between the GUI and the control software it was useful to use the command line interface that was originally implemented.

Testing integration on GUI and Database

1. Runs Database on the background and launches GUI
2. Adds lights infos through GUI page and passes to Database
3. Retrieves the datas from Database through GUI page
4. GUI displays the lights information



(Figure 12: GUI and Database Integration Demo)

4 Results

Functional Requirements	Done	Not Done
1. Host PC communicates with PowerCyber Lab Simulation.		☑
2. Hardware Coordinator communicates with the host PC.	☑	
3. IOT Light devices communicate with the Hardware Coordinator.	☑	
4. Light devices are configured on the host PC to represent relays or nodes in a simulation.		☑
5. Light device shows the state of a relay or node in a simulation using an RGB LED.	☑	
6. Light devices are battery powered, with a battery life of eight hours or more.	☑	
7. Light devices must be operable while charging.		☑
8. Light devices must be operated on a scalable network topology.	☑	
9. Light devices are able to send integer data to the host PC to trigger an attack on their respective relay.		☑

(Table 3 Functional Requirement Result)

Non Functional Requirements	Done	Not Done
1. Light enclosure is aesthetically pleasing and relatively small.		☑
2. Light enclosure must be entirely self-contained.		☑
3. All system components together must be portable enough to fit through doors.	☑	
4. Access to white board or magnetic board as a mounting surface	☑	
5. Mounting surface should not obscure any image projected upon it.	☑	
6. Light devices update to reflect changes in the simulation in less than one second .	☑	
7. Light devices must be magnetically mountable.		☑
8. Relay interface components must be easy to connect.	☑	
9. Software interface between the user and the light system must be easy to use and understand.	☑	

(Table 4 Non Functional Requirement Result)

4.1 Development Challenges

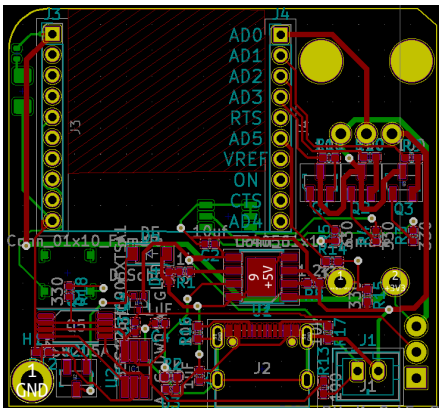
4.1.1 Software

Our biggest challenge in software development was integrating the different pieces of software, such as the GUI, the control software, and the Coordinator software. The control software was originally designed to operate on the command line since it was inherited from the previous group that had not created a GUI. It therefore did not return information on method calls and would instead print to the shell. That had to be changed for the GUI to be able to use those methods. There were also challenges in learning multiple new programming languages and technologies, such as Python, Django, and HTML.

4.1.2 Hardware

Hardware design fell into several challenges, ranging from parts shortages and inexperience. The original plan was to re-invent the original hardware schematic from the previous team, but we found that it lacked several available components from battery protection, to charging ICs.

The initial design fell flat when it came to assembling the components on the board. Due to parts shortages, we had changed the charging design from a tp4056 in favor of an mcp73832/1. Initially when testing, Soldering the components proved to be difficult in small form factors, and high soldering iron temperatures were likely what caused the first design to fail.



(Figure 13: the 2nd revision PCB)

The second revision sought to avoid this downfall. JLCPCB, the company used to manufacture the PCBs, offered an assembly service that allowed a parts placement machine to place much smaller components with vastly higher precision. After redesigning the board, and choosing to revert to the previous charging design based on the TP4056, we were almost ready to assemble the PCB through JLC, however as it turned out several parts changed status, and minimum quantities were handled in reels rather than in individual components, meaning that for 3 components to be placed, the

other 3997 would need to be purchased. The cost of only a few boards would balloon over a thousand dollars.

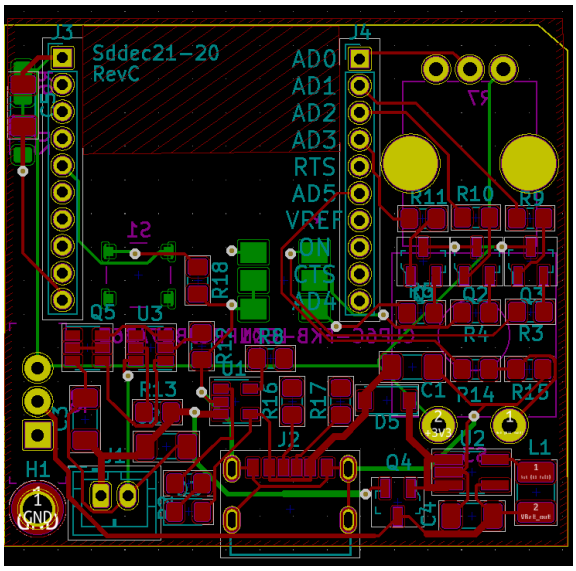
The 3rd revision was done to avoid the cost of large quantities of parts for low quantity boards. Because of the loss of an assembly service, it was decided to assemble by hand using solder paste and a heat gun.

At this stage usb C was also added as a design constraint, meaning the board would also need to meet new compliance standards.

After eliminating soldering errors during assembly of the board several issues were found, however some subsystems worked correctly. One notable failure on the design was the LED Driver transistor pin placement on the schematic to the layout. The other large issue was the switching voltage regulator was lower than expected, pulsing 2.2V instead of the expected 3.3V. The wireless module was still able to briefly connect with the coordinator when testing, but was intermittent.

The battery charging was also suboptimal, After reviewing the layout and schematic, the status input for the MCP73832 was found to be pulled down rather than pulled up. This could have been avoided with more careful review when converting the charging design from the TP4056 to the mcpMCP73832.

After reviewing and correcting the footprints the design should work minimally, however the power dropoff still has yet to be understood. As of now the layout has been updated based on the changes needed to make the 3rd revision work correctly.



(Figure 15: the 3nd revision PCB)

(Figure 14: the 2nd revision PCB)

5 Closing Material

5.1 Conclusion

Our team planned on taking the material that the previous team produced and coming up with an improved design that would perform better and be easier to use. Our design has many features that the previous one did not which make the system far better than it was, such as a graphical user interface, USB-C charging interface, two-way communication, and improved battery life. We believe that if the design is implemented properly all requirements set forward by the client will be fully met. Unfortunately, we made some mistakes and ran into some problems throughout the semester.

One of our mistakes was that we started rather late in the semester. Our first meeting with our stakeholders was delayed due to lack of communication between the team members. This could have been solved with the appointment of a designated team leader or communications officer. As it stands, all team members are on equal footing and no one member was designated a leadership role. This caused some confusion when expectations were not fully understood. Another challenge was a lack of members. At one point in the semester one of the team members was not able to contribute as much as they would have liked due to extenuating circumstances out of their control. This forced the work to be redistributed to the other members. As there were only three of us our workload would, ideally, only have increased by 33% each. Unfortunately, reality was different and some members had to take up more of the slack. This could have been avoided if this project had been allotted more personnel.

Another big problem was part shortages caused by the COVID-19 global pandemic. Some of the proposed components were not available or were only purchasable pre-installed on other boards. Within weeks of creating a design, some parts would go out of stock, and force a redesign around a similar component. An example of this is our original charging circuit, the TP4056 and the accompanying battery protection circuitry. This component was only available from JLCPCB's assembly service, and was out of stock on virtually every storefront. This forced us to find a new part to take its place, which took time, and order it which took longer still. We could have avoided this issue by coming up with a list of alternative options for each part in case we were unable to acquire one.

5.2 References

List technical references and related work / market survey references. Do professional citation style (ex. IEEE).

[1] Augustin, Aloÿs, et al. "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things." Sensors, vol. 16, no. 9, 9 Sept. 2016, doi:10.3390/s16091466.

[2]Digi, Setup Devices, 2018, [Online]. Available: https://www.digi.com/resources/documentation/digidocs/90001526/Default.htm#containers/cont_setup_devices.htm%3FTocPath%3DSet%2520up%2520%2520your%2520XBee%2520devices%7C_____0

[3]"LoRa Specification." *LoRa Alliance Resource*, Nov. 2020, lora-alliance.org/wp-content/uploads/2020/11/lorawantm_specification_-v1.1.pdf.

[4]"What Is Z-Wave Long Range and How Does It Differ from Z-Wave? - Z-Wave Alliance." *Z*, 17 Dec. 2020, z-wavealliance.org/what-is-z-wave-long-range-and-how-does-it-differ-from-z-wave/.

[5] XBEE® AND XBEE-PRO® ZIGBEE, Digi, 2016, [Online]. Available: https://cdn-shop.adafruit.com/product-files/967/p967b+datasheet.pdf

[6] XBee®/XBee-PRO S2C Zigbee®, Digi, January 2020, [Online]. Available: https://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf

[7]"Zigbee." *Zigbee Alliance*, 9 Dec. 2019, zigbeealliance.org/solution/zigbee/

[8] Digi. (n.d.). "Xbee Python library". Digi XBee Python library 1.4.1 documentation. Retrieved December 9, 2021, from https://xbplib.readthedocs.io/en/latest/

5.3 Appendices

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar data that does not directly pertain to the problem but helps support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc,. PCB testing issues etc., Software bugs etc.

Appendix I

1. Operation Manual: GUI Software
    1.1. Clone the repository at https://git.ece.iastate.edu/sd/sddec21-20.git
    1.2. Open terminal/cmd and navigate into the Nathan-Dev folder.
    1.3. In the same terminal window, run the command `python mange.py runserver`
    1.4. There will be a URL that, when clicked, opens the GUI in a browser window.

Appendix II

Appendix III

Hardware Redesign concerns

As it stands, the Xbee s2c is capable of meeting the minimum requirements, but the assumption that the provided modules had the functionality of the programmable versions was a significant drawback in the design process. The original design was thought to be using primarily micropython to drive the modules, when in actuality we ended up using a series of calls to and from the coordinator to manage the endpoints. A core part of the project was to have the modules on their own network that wouldn't interfere with normal 802.11 network, which Xbees are capable of, but they don't open up any extra functionality besides bare minimum communication. This could be solved with adding a microcontroller to the devices that has greater programmable functionality than the xbees, but this also increases the power consumption and PCB design difficulty as a drawback. Several requirements had to be dialed back because of our inexperience with PCB design in this regard. A solution that has recently been announced that could be implemented is the Esp32-H2, a new product which is able to use Thread networks, which can replace the zigbee protocol in this case. At the time of designing the smart lights it was not a finished product, and thus was not an option, however upon release would be a much more configurable solution.

Appendix IV

- All codes are pushed on Github @ https://git.ece.iastate.edu/sd/sddec21-20
- For high resolution Gantt charts.pdf file, please visit https://drive.google.com/file/d/1k6aQkD4UcFIDgFBK1CtHrsNlCSI92nlV/view?usp=sharing