Executive Summary

IoT Beagle Bone Cape

DESIGN REPORT

Team 7
Client: Texas Instruments
TI Representative: Mark Eastley
Faculty Advisor: Nathan Neihart

Developers
Taylor Weil - Software design
Parker Larsen - Hardware design
Sean Griffen - Software design
Sterling Hulling - Hardware design,
Team Email: sddec21-07@iastate.edu
Team Website: https://sddec21-07.sd.ece.iastate.edu/

Revised: December 8, 2021

Table of Contents

1 Introduction

1.1 Acknowledgement

- Advisor Nathan Neihart: Providing us direction, helping us to stay motivated throughout the project design and eliminating roadblocks.
- Client representative Mark Easley: Providing valuable resources, clarification of project requirements and ideas, as well as expertise with our hardware.

1.2 Problem and Project Statement

Our original problem statement involved more of an automotive theme. As the project progressed the theme of the project was changed to a more all purpose design by removing the stretch goal of adding an OBD interface and added a relay and switch board in its place. The original problem statement is referenced throughout the document and updates are provided.

The Zigbee Cape will allow the user to buy, create and connect to various Zigbee integrated devices and modules to the BeagleBone Green Gateway embedded computer. The Cape will physically connect to the Green Gateway to allow the user to write programs to connect, track and send sensor data to either a phone, or an external hub over WiFi or bluetooth communication. Additionally, other developed boards will give the user the ability to receive temperature or light readings wirelessly from another location or trigger a relay wirelessly using a board with switches.

This project will not only give tinkerers another toy to play with,, but it will also provide them with an easy method to learn how to communicate using Zigbee. We will also provide a few external modules that will be able to connect wirelessly to the Cape.

This project will include three main objectives and two stretch objective:

Main:

- Create a Cape to add Zigbee functionality to the BeagleBone Green Gateway.
- Connect to a daughter board or a second Zigbee device with rudimentary sensors (temperature, humidity, and/or accelerometers) to the Cape over Zigbee.
- Provide an intuitive demonstration and visualization of data collection from the Cape’s Zigbee network.

Stretch:

- Either add an OBD2 port to the Cape to access a vehicle's CAN bus, or create a daughterboard with similar functionality. This is to target this project towards automotive applications.
- Add 2 additional daughterboards:
    - RelayBoard - 4 high power relays and 4 low power relays
    - SwitchBoard - 8 switches to control the 8 relays on the above board.

1.3 Operational Environment

Our original design document talked about the extreme temperatures and conditions that a vehicle might undergo in the Midwest. Due to time and resource constraints, these conditions were not able to be tested. The design also changed to a more all-purpose design which removed

the automotive constraint. This board was designed to be used by tinkers and other people interested in IoT applications in normal, indoor environments.

1.4 Requirements

This project is to be an open-source demonstration of the TI BeagleBone Green Gateway as an Internet of Things hub. As such, it needs to demonstrate the ability to be utilized effectively as a hub for WiFi and Zigbee enabled sensors and devices. We need to have a developed library of software pieces that can enable a user to quickly and easily demo our project and continue development. It is open source, so heavy documentation that is user friendly is a priority. This project will be freely available for use and expansion on GitHub and Hackster.io, under the permissive MIT License.

1.5 Intended Uses and Users

The goal is to provide a given user with a modular Zigbee platform on the BeagleBone Green Gateway from which to develop further implementation and derivative projects. Intended users are interested in IOT, and are relatively familiar with programming languages such as C, JavaScript, and/or Python, and can edit and compile their own code. They are assumed to have the interest and ability to assemble a printed circuit board and mount it onto a BeagleBone in the form of a Cape, and supply sufficient power to the device. It is assumed that users do not use this device for industrial purposes, rather for personal use.

1.6 Assumptions and Limitations

- Assumptions:
    - Open source projects can be used for commercial, for-profit and private sector purposes.
    - The number of Zigbee connections at a given time is limited to the max that the Zigbee controller can handle (240 devices).
- Limitations:
    - The Zigbee Cape is designed to be added to another board such as a BeagleBone or one of the daughter boards. Due to this, the Cape does not have any onboard voltage regulation or voltage protection circuitry.
    - The daughter boards each have linear regulators designed for low power applications and have a limited input voltage of between five and eighteen volts
    - The Zigbee protocol used is limited and fairly specific for TI's ZStack implementation. Any cross-platform/software stack compatibility is limited to what the BeagleBone can translate.

1.7 Expected End Product and Deliverables

- The Cape and daughter boards will be listed publicly on GitHub and hackster.io as an open source project that anyone can access and download files from. They can then send those files to be manufactured, and they'll have their very own Cape and daughter board like us.
- The technical documentation for each PCB shall contain a kicad file, a parts list and a wiki description.
- Any other items that will be delivered to the client shall also be included and described, unless their definition and description are covered by existing documentation (i.e: no documentation needed for the Beaglebone, micro USB cable, or 5 v power adapter).

- Examples might include a household power supply to eliminate the need for batteries, a user's manual, or other project reports.
- There shall be at least a one-paragraph description for each item to be delivered.
- Soldering of PCB boards by the beginning of August. We want to have our 1st iteration PCB designs back from printing, and be assembling and testing them during the first weeks of August to ensure we have sufficient time for iteration.

2 Project Plan

2.1 Task Decomposition (in no particular order)

1. PCB design and manufacturing
    a. Component selection and testing
    b. Design validation in software
    c. Manufacturing spec validation
2. Zigbee communication library
    a. Connection establishment between hub and peripheral devices
    b. Data validation on send/receive between 1 hub and 1 peripheral device
    c. Parallel communication between 1 hub and multiple peripheral devices
3. Data management/offloading
    a. Connection establishment between WiFi router and BeagleBone
    b. Data validation on send/receive for WiFi
    c. Offload sensor data to an off-site web server
    d. Display sensor data on an interactive web page
4. Integrations with 3rd party services (Stretch Goal)
    a. Connection to open-source home automation software (HomeAssistant, OpenHAB, etc)
    b. Connection to Amazon, Microsoft, and/or Google cloud APIs for data storage/access
5. CAN communication library (Stretch Goal)
    a. Connection establishment between Cape and attached CAN bus
    b. Data read/parsing validation through OBD2 port
    c. Data write validation through OBD2 port
    d. Safety interlocks
6. Create Additional Daughter boards (Stretch Goal)
    a. Design any additional daughterboard or 2 to demonstrate an additional use case
    b. Add Zigbee capability through on board logic or mount to a Zigbee Cape
    c. Test interaction between Zigbee connected devices

2.2 Risks And Risk Management/Mitigation

Since our devices operate in the 2.4GHz frequency range, this could impact other devices not operating in our network, although the Zigbee protocol tends to avoid conflict with other 2.4GHz protocols such as WiFi or BLE

Due to the change from automotive to a multipurpose design, the following requirements are no longer applicable.

- Perhaps put the Cape and board in an enclosure so it can't get wet easily, or a conformal coat to protect from corrosion.
- If CAN communication gets implemented, and writing to the CAN bus is feasible, ensure no data writing occurs while vehicle is in an unsafe state (i.e. on or out of park depending on the amount of state feedback is available), and that reading data from the CAN bus does not cause any overloading

- Daughterboards can be off-the-shelf components
- Ensure passive power draw of Cape and beagleboard do not cause excessive battery drain of the vehicle

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

1. PCB design is manufactured and validated
2. 2-way parallel Zigbee communication is achieved between Cape and peripheral devices
3. 2-way CAN communication is achieved between vehicle and Cape (Stretch Goal)
4. 2-way communication between daughterboards and BeagleBone through Zigbee network (Stretch Goal)
5. Cape can connect to 3rd party services and off-load data through WiFi and can connect to a phone over Bluetooth and display data through a companion app (Stretch Goal)

2.4 Project Timeline/Schedule (Starting week of March 8, 2021)

Week 1: Research Datasheets and dev boards.
- Select 2x sensor types, order

Week 2-3:
- Receive hardware and work on dev boards

Week 4:
- Know what hardware is needed to build out software, breadboard prototypes.

Week 5-6:
- Design software libraries and hardware design for Cape and daughter boards

Week 7:
- Finalized PCB design, Prepare to manufacture. Demo Prototype.

Week 8:
- Send off PCB design for manufacturing, order SMD/bespoke components

Summer Break:
- Receive PCB's, solder and test , iterate PCB's if necessary

Week 9:
- Demonstrate on a real PCB.

Week 10:
- Develop documentation and user libraries.

Week 11:
- Evaluate next PCB iteration

Week 12:
- Document 1x sensor implementation

Week 13:
- Document 2nd sensor implementation

Week 14:
- Develop software libraries for multi-function

Week 15:
- Send for manufacture PCB rev2.0

Week 16:
- Software development for data-collection

Week 17:

- Data processing

Week 18:

- Portfolio and packaging.

Week 19:

- Full-feature benchtop demonstration

Week 20:

- Real-world demonstration.

Week 21:

- Submit final project.

2.5 Project Tracking Procedures

Timeline and requirement progress tracking will be mainly done using Trello. Git commit messages in the official repository will show a direct timeline of what code is changed when.

2.6 Personnel Effort Requirements

Task	Software Team Hours	Hardware Team Hours	Explanation
Meetings	120 H	120 H	Each week, our group meets with our advisor and our TI rep to ask questions and give updates
Standup Discussions	60 H	60 H	Each week, our group spends a few hours catching each other up on progress and collaborating
BeagleBone setup	15 H	0 H	Getting the beaglebone hardware into a working state and learning the programming interface
BeagleBone UART testing	10 H	0 H	Programming the beaglebone as a UART master device
Zigbee dev board testing	20 H	0 H	Learning how to program the zigbee controller by using TI software and tutorials
KiCad Setup	0 H	15 H	Learning how to use KiCad and setting up the libraries on Git
Schematic Symbols	0 H	50 H	Creating schematic symbols for parts
Schematic	0 H	50 H	Creating the schematic for the Zigbee Cape

Layout	0 H	60 H	Creating the layout for the Zigbee Cape
Ordering Parts and Boards	0 H	5 H	Ordering Parts and Boards from various suppliers
Assembling Boards	0 H	10 H	Assembling boards by hand and reflow

2.7 Other Resource Requirements

In addition to software development, we will also be designing and assembling the hardware for our project. We will require development boards to test our and develop our software on, and then transition to building custom boards using similar components. We will need access to tools such as soldering irons and a reflow oven to assemble boards. For software development we will be using examples provided by Texas Instruments in Code Composer Studio to program Zigbee development boards and the BeagleBone.

2.8 Financial Requirements

Cost per board is approximately $50, based on current part pricing. We have purchased and assembled 5-6 boards. This means our financial requirements are $300-500. This estimate does not include the cost of the prototyping hardware, of which we have already received from TI. Bulk production costs will be markedly lower, as much of our cost comes from shipping and low quantity parts.

3 Design

3.1 Previous Work And Literature

All of the technology we are implementing into this project already exists. We will be utilizing the Zigbee wireless protocol, a common technology used in IoT projects. Additionally, we will be interfacing with a variety of sensors using our selected Zigbee controller. Our aim is to utilize these pieces of technology to create a hub as an interface for hobbyists and future developers to expand on our project for their own personal use.

The greatest standout of our project compared to similar designs, is that we will be creating an out-of-box-experience. Our users can print out circuit boards and download our code and immediately start utilizing the hardware for themselves. While the Zigbee communication will be open-source and available to the user, they will not need any understanding or programming knowledge in order to get started. In order to accomplish this goal, we will utilize datasheets of development boards already in place, research best practices for wireless technology, and provide detailed documentation for any software libraries we create.

3.2 Design Thinking

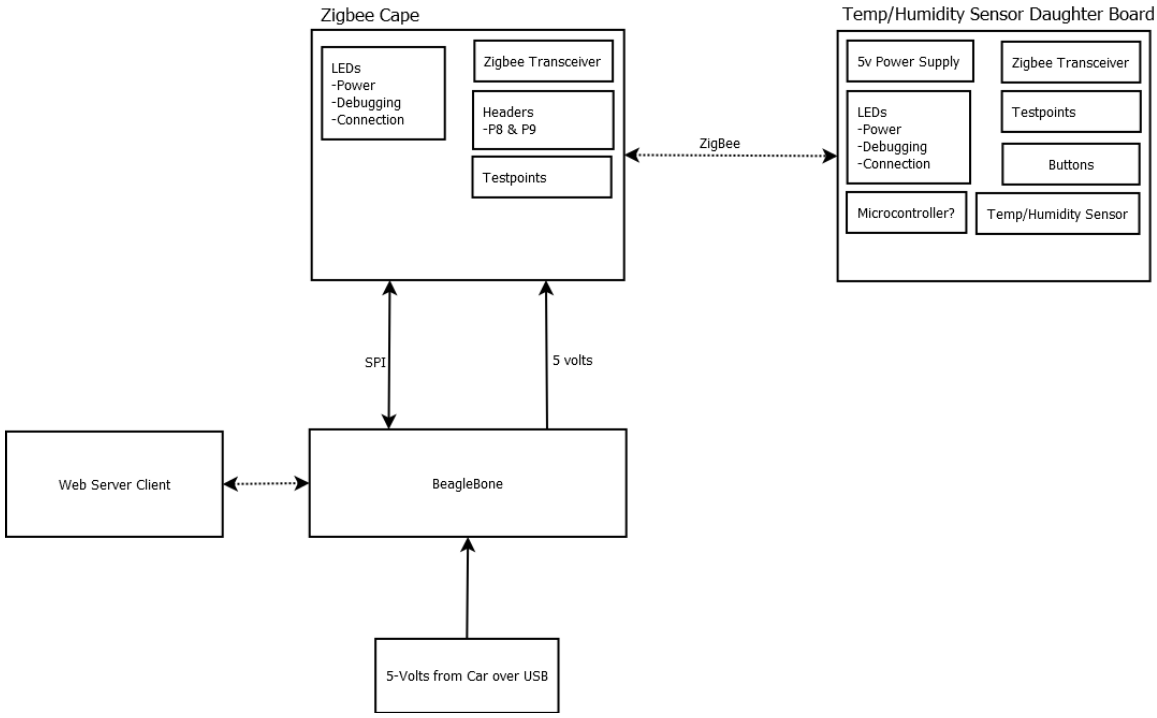
The Cape needs to be in the same form-factor as the BeagleBoard it's attaching to, and needs to operate on relatively low power. Additionally, the software libraries used and third party services should be widely available and well documented as we keep the end-user in mind.

We had proposed additional ideas that would increase the functionality and usefulness of being in a car. One idea was to connect to the car's canbus OBD2 port to utilize onboard signals. This idea was tabled until we feel confident about Zigbee and utilizing external sensors. We hope to make a daughter board that features a canbus OBD2 port that transmits data to our Cape design in the future.

For semester two, we decided to drop the idea of including an OBD2 port to interface with the onboard CANBUS of a vehicle as a main goal in favor of focusing on data aggregation and building out our wireless communications. We instead designed and constructed a relay board and a switch board that would use a reproduced Cape design as the communication component. This is further elaborated on in section 5.1.2.

3.3 Original Proposed Design

Our proposed Cape design will incorporate a Zigbee microcontroller and numerous feedback LEDs and debugging endpoints. A high-level schematic for a complete system is provided below:



This schematic shows how our Cape connects to the beaglebone, and how wireless communication is handled between the Cape and the daughter boards, and how the beaglebone connects to third-party clients.

This Cape has a dedicated CAN transceiver, pass-through GPIO from the BeableBoard it's connected to, a dedicated Zigbee MCU, and a dedicated on-board sensor of some kind (temperature, light, humidity, etc). Designing the Cape with a dedicated Zigbee antenna allows a user to connect any Zigbee certified device to control and get data from. Having a CAN controller allows a user to access diagnostic messages from the car and even send actions for the car to perform like turn on, turn on the a/c, heat, headlights, windshield wipers, etc. The BeagleBoard already has WiFi and Bluetooth built in, so we don't specifically need to integrate those into out Cape design.

3.4 Design Plan Changes

The plan for the BeagleBone and Zigbee Cape connection did not change but the use case and the additional daughter boards did. The final design, while useful in an automotive setting, was more structured towards an all-purpose setting. The BeagleBone no longer needs to connect to the car's usb and the CAN application was not completed. Instead, two additional daughter boards were created. The switchboard and relayboard contain headers that connect to a Zigbee Cape to allow it to connect to the Zigbee network. Both of these boards connect to the Zigbee Cape over SPI in the form of an IO expander. The switch board contains 8 switches that can be linked up to the relayboard. The relayboard has 4 high power relays for low power and 4 high power relays that can be used for higher power applications. The software also needed to be adapted to align with the new approach. The primary change from our initial design was simplifying the project to be more focused around collecting and reporting data from sensors. Additional changes and implementation details can be found in the software section of the user's guide 6.3.

3.5 Technology Considerations

No technology limitations have come up during the design phase, but any that do will be listed here in the future.

For data reporting and aggregation, we have chosen to focus on connecting to a local web server rather than an Amazon Web Services Bucket. A web server will serve well enough to prove our design and will provide a starting point to build out other HTTP connections.

3.6 Design Analysis

We have brainstormed several ideas for implementing our Zigbee transceiver into a Cape. One of our promising ideas that we decided against was to have the Zigbee transceiver on its own removable board that could double as a remote sensor. This would have the effect of streamlining the creation of remote sensors, as they would have the same PCB. We decided against this as it would still require us to design two PCB's, and wouldn't be relevant due to the prolific availability of off the shelf Zigbee compatible sensors. We have decided to have our Zigbee integrated into our BeagleBone Cape.

3.6 Development Process

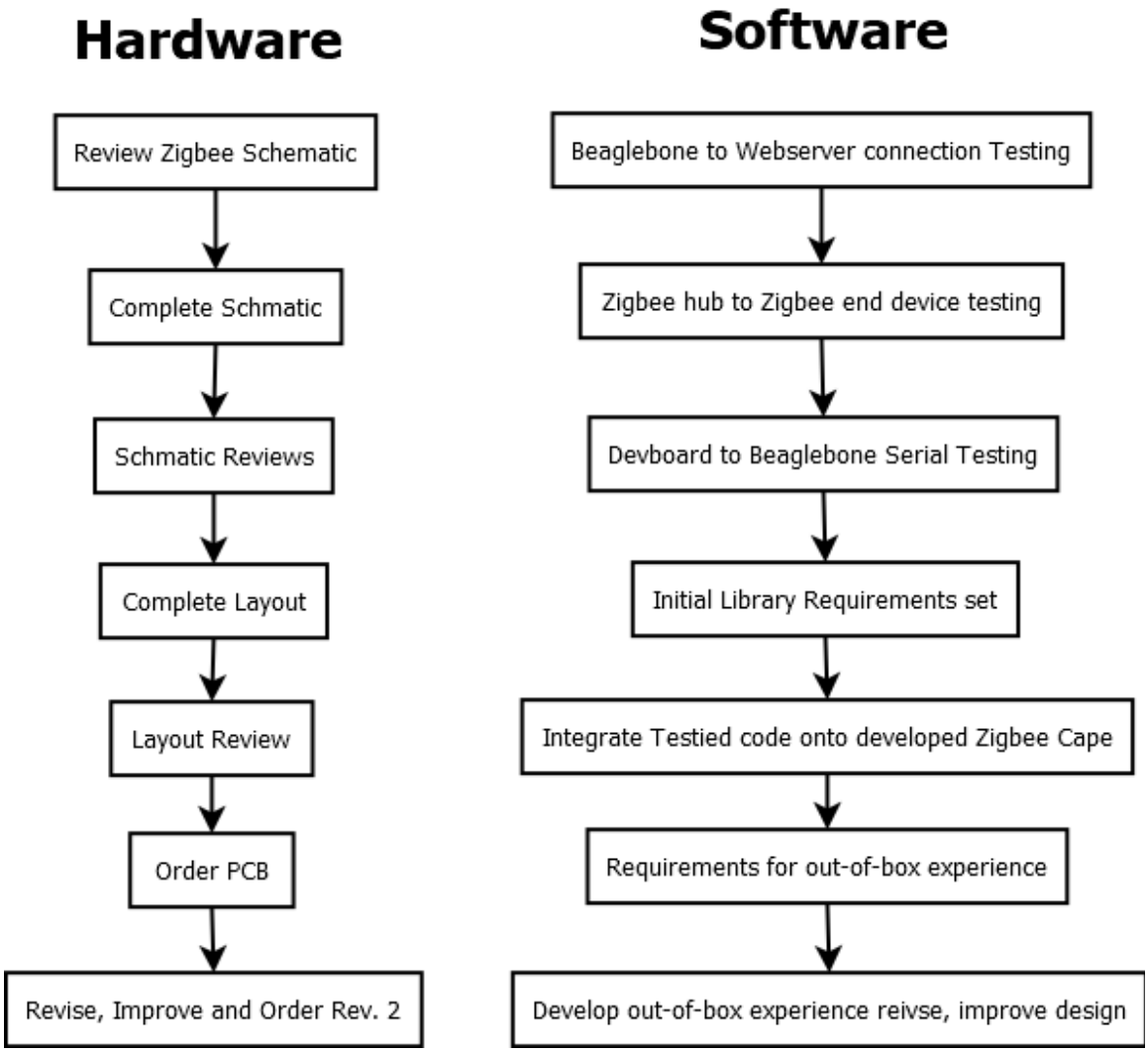
3.6.1 In Plan

We are trying to follow a pseudo-agile process, using Trello as a time and task keeping resource, and using git and it's methods (commit messages, and an official changelog) as a version control system for hardware designs, and software versions.

3.6.2 In practice

We began using the Trello board to plan out the tasks that needed to be accomplished and our timeline for doing so. This allowed us to break up our tasks and delegate while working remotely. We could see progress taking place and easily review the status of the project. Over the course of the project, we began using Trello mainly to keep track of main project goals, and got less detailed in our cards, as we were able to keep each other updated and informed using our groups Discord Server.

3.7 DESIGN PLAN



4 Testing

Preliminary testing was done on development boards and with example programs in order to gain an understanding of the libraries and technologies we will be using. As the project progressed, we slowly adopted our custom hardware into the build to test each component.

4.1 Unit Testing

4.1.1 Hardware

Once we receive the board back from manufacturing, we will need to make sure everything is operational. There are many different parts but most of them can be split into 4 different groups:

- Power and grounds: Once we receive the board back manufacturing, the first thing that we will need to check will be the power and ground nets. If these nets are not all connected and at the same voltage level, there is a very low chance that the other part of the circuit will work as expected. In order to prepare for this step, it will be very important to expose these nets at critical locations or test points
- Headers and BeagleBone connections: The connection between the BeagleBone and Cape will be completed through the use of headers. We will need to make sure that the boards make a good connection both physically and electrically.
- Programming/JTAG: We will need to make sure that the Cape can be programmed successfully. This will be completed through the use of the onboard LEDs.
- ZigBee communication: This will be one of the last things that we will test when we bring the board up. This will involve using a known working transceiver and attempting to communicate with it.

4.1.2 Software

Initial library design and functionality can be completed without the use of custom hardware being designed. Testing this on our provided development hardware will involve the following:

- Sending multiple different types of data between Zigbee end-devices and the Zigbee hub.
- Sending data from multiple end devices at once.
- Ensuring data sent over Zigbee is property transferred to the beaglebone.

Once the custom hardware is deemed working, software developers will port working code to the new hardware and perform similar tests. At this point work can be completed on the “out-of-box” experience aspect of our design.

4.2 Interface Testing

In order to verify our working software is compatible with our working hardware, we replaced one component at a time into the working system. To ensure each component worked both individually and within the system we did the following:

- Check hardware: Make sure the PCB is working properly on its own. Perform tests described in section 4.1.
- Check software: Make sure that the software is working correctly on the test board. Perform tests described in section 4.1.
- Add to out-of-box functionality to increase immediate usefulness. This will also ”stress test” key aspects of our hardware and software designs.
- Bring features together incrementally and make corrections as needed. At this point we will have clear design and functionality objectives to compare actual test results to.

4.3 Acceptance Testing

Throughout the process of this project we will complete a list of requirements or tests that we must pass in order for all parties to accept our project. This list will be approved by every member of the team, project advisor Nathan Neihart, and the client that we are working with Texas Instruments represented by Mark Easley These requirements will include but are not limited to functionality of the final product, documentation for future users, public release of project source code and design resources, and presentations showing the completion of this project.

4.4 Results

4.4.1 Hardware

As we complete the initial PCB design, we are making detailed notes and documentation of our design decisions and changes. This way we can have more productive design meetings and we can better understand our results. As we progress through this project it will be really important to reference and update these documents to better learn from our mistakes and prevent us from backtracking.

4.4.2 Software

In our testing with custom hardware we had a number of wins and losses. Our team was relatively successful in assembling the board, so flashing our program to the CC1352 was not a problem. We did have difficulties connecting the zigbee device while the UART was connected to the BeagleBone. Some of our iterations were more successful than others, but we were unable to get a Cape with both of these features reliable in time for the final demo.

5 Implementation

5.1 Hardware:

5.1.1 Semester One: Plan

After the conclusion of the first design of the printed circuit board, we are planning to start out the semester by powering up the circuit board. We will test all the components separately and then hopefully together if that goes well. Depending on the results, we will make corrections and improvements to the board in an attempt to improve performance. Our goal is to get a working PCB that the software team can successfully program and communicate through.

5.1.2 Semester Two: Hardware Implementation

At the beginning of the semester we fabricated revision 0 of our Zigbee Cape, and discovered some assorted design errors. The CC1352P Zigbee microcontroller that we ordered was chosen based on parts availability and was understood to be pin-compatible with the CC1352R. However, we discovered that the -P variant contains a high-power amplifier section for the output to the Zigbee antenna. This was not designed around for our first revision, and we quickly discovered that our transmission pins were not transmitting. We could receive signals as the RX pins were working, but since we did not hardware initialize the high-power circuit on the -P variant of the Zigbee chip it did not transmit in a default state. We designed a revision 1 based off of this new information, and have fabricated this. We also designed and fabricated a pair of daughter boards; SwitchBoard and RelayController.

SwitchBoard is an accessory Cape that is pin compatible with BeagleBone, but is designed to attach atop our Zigbee Cape. It is intended to allow a user to select 8 slide switches to control the RelayController. These slide switches are translated into I2C by an expander chip, and piped to the Zigbee Cape to send communications. The intent is to allow the switch board to provide the control side of a remote, so the user can wirelessly control accessory functions. Up to four of these boards can be connected to a single Zigbee, but physical limitations prohibit this. Our first revision of this design has a minor error in the reset pin being tied to ground instead of a pull-up.

RelayController is another Zigbee Cape that can intercept this communication and send I2C to a decoder chip that expands it out into 8 output pins. These pins drive a set of solid state relays. Four of these relays are directly connected to headers that allow the user to trigger low-power items, such as a lamp or fan. The other four relays are used to trigger 12v high power relays to drive higher power loads such as motors or pumps. The intent is to allow the relay board to provide remote control of vehicle functions such as coolant pumps or window motors. It could also be used to control accessory lighting, removing the need to run multiple wires to switches within the vehicle. Up to four RelayControllers can be connected to a single I2C bus and differentiated by hardware address pins. The physical design requires an extra set of header pins to be installed to physically connect multiple relay boards, or another Cape to be installed between layers. Our first revision of this design has a minor error in the footprint for the high power relays. There are two SKU's for the relay, and the SKU ordered had pins mirrored from

how it was laid out. We can fix this by either mounting the relays upside down underneath the board to mirror it, or by installing a jumper wire to bridge the mirrored contacts. We have opted to install the jumper wires.This board still needs to be programmed to complete the stretch goal.

5.2 Software:

5.2.1 Semester 1: Software Plan

Following the testing, we plan on writing a specific library of code that expands off the example code provided by Texas Instruments. Instead of sending a signal to toggle a light, we will be sending a JSON payload over Zigbee with data collected from the sensor units. Additionally, we will utilize our findings of communicating between the Zigbee controller and the beaglebone to forward that message back to the main device. Here, the data is ready to be displayed or forwarded to a cloud service. This implementation will use the custom PCB once it is designed and manufactured.

5.2.2 Semester 2: Software Implementation

The software implementation followed our plan with slight variations. We did continue to build off of the TI sample projects, by the end we were able to clean up some of the boiler-plate code and personalize the project to our needs but a large part of the file system was left as we found it. A small change to the zigbee implementation was to instead only send the data read from the sensors as an integer and build the JSON string on the Cape using the device ID from the network package header. One thing we could not get to in time was sending data back up to the end device from the BeagleBone as we focused on the data collection. We were able to successfully aggregate the data and send it to a cloud service, a web server in this case, for our final demo.

6 *Operation Manual*

The BeagleBone Cape IoT project is completely open source and available for anyone to jump in where we left off. As previously mentioned, this project is intended to lay the foundation to any future IoT project where a user wishes to utilize zigbee with the BeagleBone platform. To reach parity with what we have set up to demo you are going to need to set up and implement the hardware and the software as enumerated below.

1. **Setup**
    a. Acquire the hardware
    b. Install Development Environments
    c. Download Resources
2. **Hardware**
    a. Design Cape
    b. Design Daughter Boards
    c. Order Parts
    d. Assemble boards
3. **Software**
    a. Setup Zigbee projects for the CC1352P
    b. Setup the BeagleBone
    c. (Optional) Setup web server / IoT Endpoint
4. **Using Your Devices**
    a. Understanding the Platform
    b. Using the Zigbee Devices
    c. Using the BeagleBone

6.1 Setup

Before you can get going on your own BeagleBone Zigbee IoT project, you will need to do some preparatory work.

6.1.1 Acquire the Hardware

A number of devices are used to develop and test your custom design and those should be ordered early to ensure that you can begin programming them ASAP. The required devices are:

6.1.2 Install the Development Environments

The BeagleBone will need to be programmed with Cloud9 IDE. Cloud9 is a built in IDE that can be accessed from the local server hosted by the BeagleBone itself. It is recommended to follow the setup instructions here:
https://beagleboard.org/getting-started

The CC1352 microcontrollers will be programmed using TI's Code Composer Studio. This will need to be downloaded and installed on your local machine. There are additionally a number of packages used for the projects specifically, you will be prompted to download these automatically if you try to import a sample project from the device explorer for the CC1352. Note, all required dependencies are included in project files or from a TI resource.

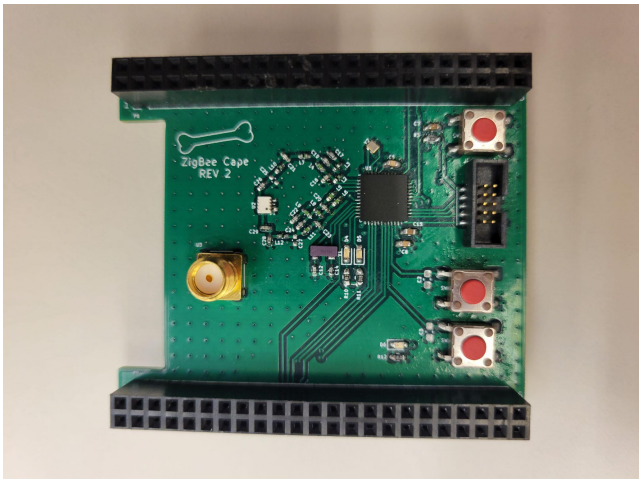
6.1.3 Download Resources

Everything that you will need to design has been listed and documented in the resources section of our closing materials. If you navigate to the team's github page you can download the source code for all of the projects used in this demonstration. There should be 2 code composer projects, one Cloud9 project, a Java SpringBoot project, and a React project. The last two are optional and are only used for demonstration purposes. Make sure you can build the two Code Composer projects at this point.

6.2 Hardware

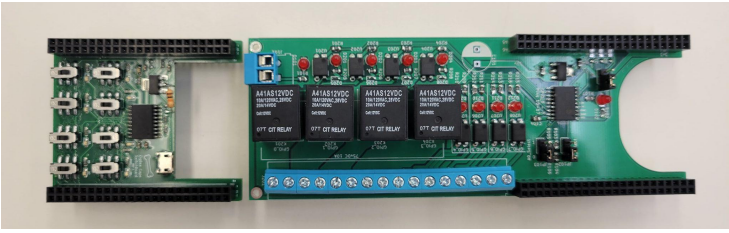
6.2.1 Design Cape

The Cape will utilize the CC1352P microcontroller to allow the Cape to communicate with the BeagleBone over UART and additional Zigbee devices over Zigbee. The Cape will also utilize a series of buttons and LEDs to allow for an easier user interface for users and developers. The CC1352P microcontroller can be programmed using the onboard JTAG connector.



6.2.2 Design Daughter Boards

A switchboard and relayboard were created to demonstrate a use case for this system. The switchboard had a switch for each relay on the relayboard. In order to simplify the design, the daughter boards were attached to the Zigbee Cape through the use of headers. This design allowed us to only need one Zigbee circuit to work.



6.2.3 Order Parts

The parts for the BeagleBone and daughter boards were ordered through a combination of Digikey and Mouser. Due to the current parts shortage it was critical to

order parts ahead of time to make sure that the parts were available when designing and ordering the boards. Additional quantities of critical and low stock components were ordered in the case that they went out of stock and an additional revision was needed. All boards were ordered with default settings through JLCPCB. The Bill Of Materials can be found within the design folders, or generated from the KiCad Project.

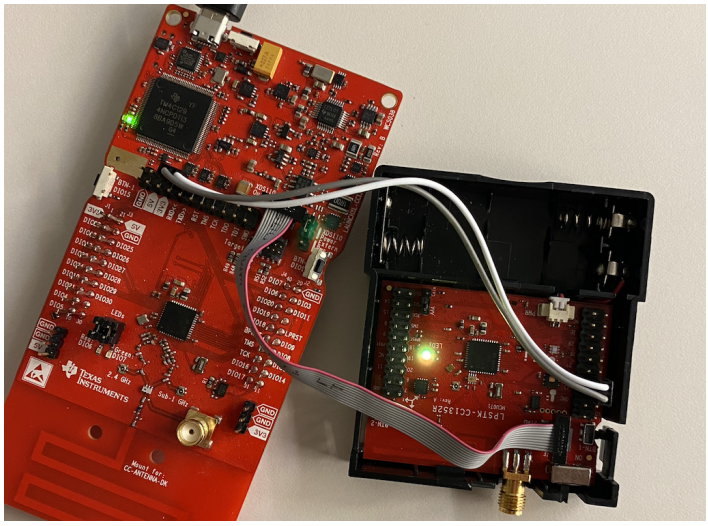
6.2.4 Assemble boards

The boards and parts were bought separately so they needed to be assembled in the lab. Due to the size and type of components, reflow soldering was chosen to simplify the process. For the 0402 SMD parts, and the zigbee chip itself, on the zigbee Cape a microscope was used. It is advisable to order solder mask stencils if choosing to reflow. The switch board is mostly surface mount parts but these can be installed with the help of a magnifying glass, or with the use of a reflow oven. The relay board has many through-hole parts and can be hand soldered in less than two hours. It has 0804 parts but they are spaced well and are not hard to work with if you do them first. Be careful of the orientation of the protection diodes, as they are easy to install backwards. It is advisable to assemble at least one of each board, and at least two zigbee boards the full wireless capabilities are to be realized.

6.3 Software

6.3.1 Setup the Zigbee Projects for the CC1352

The CC1352 is a microcontroller being used to read sensor data, as well as communicate with the BeagleBone over UART. There will be two separate devices, the Cape and a sensor tag, that will communicate over Zigbee and have separate (although similar appearing) projects. Once you have downloaded the resources from our GitHub page, you should open up Code Composer Studio and add these two projects as Code Composer Projects. You should be prompted to download the proper resources, make sure you install these and are able to build the project. To flash the launchpad device, you will need to connect it to your computer with a USB cable, build the project, and select flash. If the project is not automatically selected as active then you may need to run as debug first. To run the sensor tag project, you will need to attach the JTAG connector to the sensor tag and the JTAG connector on the launchpad closer to the USB. Also make sure to remove the jumpers that would be programming and powering the launchpad's onboard microcontroller. To view the UI for the sensor tag, connect the pins for RX to TX and vise-versa. The connection for the sensor tag should look like this:



Launchpad (left) with CC1352 Sensor tag (right)
connected with JTAG and UART jumpers

6.3.2 Setup the BeagleBone Green

To set up the BeagleBone Green, follow TI's Getting Started instructions provided at https://beagleboard.org/getting-started. Be sure to follow the instructions for flashing the BeagleBone's onboard eMMC memory (found at http://elinux.org/Beagleboard:BeagleBoneBlack_Debian#Flashing_eMMC), and install a "Flasher" version of the Debian linux image (found at https://beagleboard.org/latest-images). After the BeagleBone is successfully flashed and powered on, go to http://192.168.7.2 to access the Cloud9 IDE. This will be used to access the BeagleBone's terminal and run any project files. Using the terminal, update your BeagleBone following TI's Upgrade instructions provided at https://beagleboard.org/upgrade. After updating is completed (this can take anywhere from fifteen to thirty minutes), create a folder in the main directory of the Cloud9 IDE and copy the files from the 'firmware/beaglebone' directory on the project repository into the new folder on the BeagleBone. After this, inside the `app` directory, run the file 'main.js'.

6.3.3 (Optional) Setup web server / IoT Endpoint

In order to view the data that is being captured by the BeagleBone, you're likely going to need a database with a server to interface with. Included in our repo in the software directory will be two projects, a web server and a web app. These projects are example projects you could base your project on or use to verify everything is working as intended.

To set up the web server application you'll need to open up the given SpringBoot project in Eclipse or Intellij IDE. After this, install any Maven dependencies used by this project (this will most likely happen automatically) and configure the MySql database location in 'src/main/resources/application.properties'. A placeholder configuration exists that will get you started, but configure the MySql database endpoint as needed. After the configuration has been verified, run the startup file 'WebHostApplication.java'. This will

run the SpringBoot instance, connect to the specified database, and create a HTTP endpoint for the UI application to connect to. To check that the project is running correctly, a landing page will be live at http://localhost:8080/.

To run the UI web application you will need to open up the React project and make sure Node.js is installed on your machine. I recommend opening the project with VSCode and opening a terminal, but any command line will work. Start by running the command: `npm install` followed by `npm start`. At this point the server hosting the web page should be live at http://localhost:3000/

6.4 Using Your Devices

6.4.1 Understanding the Platform

In order to get a better understanding of how the zigbee network is configured, I would recommend opening the device's user interface. You should already have your device plugged into the computer for programming by interfacing through the launchpad. You should be able to find the serial port being used by your computer and monitor it using a program such as PuTTy on windows or Screen on linux machines. You will need to set the baud rate to 115200. If your program is running correctly you should see a UI like the one below.

```
Sensor Tag Project
Press Enter for Help
<      HELP      >

Device Info: [IEEE Addr] 00124B001CAB7
   NWK Info: [PAN ID] 0x65c3 [Channel] 11 [Short Address] 0xde87
   NWK Info: [Parent Address] 0x0000
   ZDO Info: [Logical Device] End Device [State] In the network
   BDB Info: -- Id000 Srch000/00
  Bind Info: --
Sensor Info: [Status] READING [Data] 40
```

Sample Serial UI for the Sensor Tag

In order to understand how the devices communicate with each other, it's important to understand the structure of a zigbee network. At a high level, the network is made up of a Coordinator, Routers, and End Devices. You can think of the coordinator as your wifi router, the center aggregating device that creates the network and knows all of the connections. The Cape will take the role of the coordinator (ZC), and the sensor tags will take the role of the end devices (ZED). There can be a number of end devices but they will not create or repeat a network unlike in a mesh network. Each device is given a short address which will be reported in each transmission to help identify the source.

6.4.2 Using the Zigbee Devices

Once the projects are flashed and running correctly we can connect the devices and begin sending data. As mentioned in the previous section, the Cape will be a coordinator and the Sensor Tag will be an end device. To get the coordinator on the network you should open the device's UI, use the arrow keys to change the menu option to "Commision", and press Enter. You should see a 3 minute countdown start and the network is now accessible. Repeat this process on an end device and you should see the status change to discovering, and then to "In The Network". If it goes back to "initialized" or "discovering", there may be something wrong with the configuration or

the hardware. Once the end device is in the network, you should see a green LED on the board turn on to verify an active connection. Also, each network is randomly assigned an ID displayed as “PID”. This value will be the same for all devices on the network. At this point you are good to go! There is no more setup needed to send data, your sensor will automatically start collecting and sending information as configured in the project you uploaded. Additionally, any end device will automatically attempt to reconnect to a previously connected network, if you wish to reset a connected network on either device you can choose the “Reset” menu option on the UI and even specify a PID to join. If you wish to manage sensor configuration without reflashing, we have included ideas for communicating back to the end device in our future improvements section 7.2.

6.1.4.2 Using the BeagleBone

The BeagleBone is the most customizable part of the system. For our example, we have set up a web server that communicates with the BeagleBone over a websocket, but you should build onto the BeagleBone project as your needs desire. What you should know is that the BeagleBone will communicate with the Cape over a wire UART connection via the pin headers it will be connected to. On startup, the devices will automatically connect and transmit data when it comes available.

7 Closing Material

7.1 Conclusion

This project has been a challenge that came filled with a number of learning opportunities and has helped refine us to think as engineers. Reflecting on the design process we noticed a number of changes that were made along the way. Knowing what we know now, we could have focused more on implementing the zigbee network sooner and had more time to expand on the project. While we did not accomplish all that we had hoped to, the breadth of learning was great to all of the members on the team.

7.2 Additional Considerations and Improvements

Even though we accomplished our minimum viable product, there were still lots of features we could have improved upon. In our first semester we talked about working in an OBD2 interface that communicates over zigbee. If we continue to work on this project after graduation or someone else wishes to iterate on the project, we would recommend looking into this as a feature and narrowing the focus down to automotive data aggregation and automation. Another consideration we passed up was additional sensor types. To build off of this we would research other i2c sensors that could be surface mounted and build a custom sensor tag. We also wish we could have put more time into the relay board / switch board. Since the development of the hardware pushed near the end of the semester we were unable to bring its connectivity into the rest of the project ecosystem. Additional software improvements can be made to configure sensor polling rates and configuration. One stretch goal we discussed was having a board with multiple sensors where the web app could select which sensor was going to be active. This could be accomplished by adding another zigbee data attribute with “reportable” set on the cape.

7.3 Resources

Source Code:

https://github.com/iowa-state-senior-design-dec-2021-proj7/BBGreenZigbeeCape

Hackster:

https://www.hackster.io/iowa-state-senior-design-dec-2021-proj7/beaglebone-green-wireless-zigbee-cape-c2da61

7.4 References

"LAUNCHXL-CC1352R1." LAUNCHXL-CC1352R1 Development Kit | TI.com www.ti.com/tool/LAUNCHXL-CC1352R1#order-start-development.

"CC1352R1 LaunchXL." CC1352R1 LaunchXL - Zephyr Project Documentation, developer.nordicsemi.com/nRF_Connect_SDK/doc/1.0.0/zephyr/boards/arm/cc1352r1_launchxl/doc/index.html.

Zuo, Baozhu. "Seeed Studio BeagleBone® Green Gateway." *Seeedstudio*, wiki.seeedstudio.com/BeagleBone-Green-Gateway/.