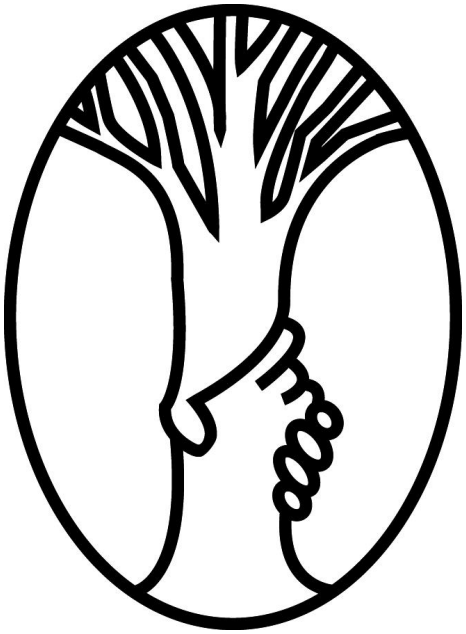
MICHIGAN STATE
UNIVERSITY



ECE 480: Senior Design
Design Team 6

Bluetooth Low Energy Beacon

Facilitator

Dr. John Deller

Team Members

Nicholas Blackledge

Chunyang Chu

Steven Le

Matt Smania

Bowei Yu

Sponsor

Resource Center for Persons with Disabilities(RCPD)

Project Liaison

Stephen Blosser

Aditya Mathew

Executive Summary

Navigating a large environment like the Michigan State University campus is a challenge for the blind and visually-impaired. This project's goal is to design a Bluetooth Low Energy (BLE) beacon, based on Apple's iBeacon standard, for assistive navigation. The primary objective is a low cost BLE beacon to allow large scale deployment across the Michigan State University campus. The Texas Instruments cc2540 was selected as the core component of the design. After testing, the designed beacons performed at the same level of accuracy and signal range as an iPhone acting as an iBeacon. Mobile applications for Apple's iOS and Google's Android operating systems were also developed to demonstrate the functionality of the BLE beacons. The final beacons have a component cost of roughly $10 and are about 1" x 2" in size.

Acknowledgements

We would like to thank the Michigan State University Resource Center for Persons with Disabilities, specifically Stephen Blosser and Adi Mathew, for the opportunity to work on this project and the assistance they provided.

We would also like to thank Dr. Deller, our facilitator, for his mentorship during the project; his advice and feedback made us a better team and contributed greatly to our success on this project.

In addition, we would like to extend our thanks to Professors Timothy Grotjohn and Lalita Udpa, as well as Gregg Mulder and Brian Wright from the ECE shop, for the guidance and help they provided during the senior design process.

Lastly, we would like to thank Radius Networks for providing the use of their Android API and reference application for iBeacons which made it possible to build a fast and reliable software package on the Android platform.

Table of Contents

Chapter 1 - Introduction and Background……6
1.1 Introduction……6
1.2 Background……7
Chapter 2 - Objectives and Approach……9
2.1 Design Specification……9
2.2 Conceptual Design Description……9
2.3 Conceptual Design Rankings……10
2.4 Solution Description……12
2.5 Budget……13
2.6 Proposed Schedule……14
Chapter 3 - Technical Implementation……17
3.1 Hardware……17
3.2 Software……18
Chapter 4 - Tested Data and Results……21
4.1 Testing……21
4.2 Result……22
Chapter 5 - Conclusion……26
5.1 Final Cost……26
5.2 Schedule……26
5.3 Project Fiding……26
Appendix 1 - Technical Roles……28
Appendix 2 - Reference Materials……33
Appendix 3 - Technical Documentation……35

Chapter 1 - Introduction and Background

1.1 Introduction

The Michigan State University East Lansing campus consists of 538 buildings spread out over 8.125 square miles. Navigating the campus is a daunting task, even for normally-sighted people. The challenge for blind and visually-impaired members of the Michigan State community is significantly greater while they are becoming familiar with the campus. Twenty minute class change intervals are inadequate for blind students and the risks of injury to anyone are increased by rushing through unfamiliar locations.

Currently, there are limited accessibility options regarding navigation for visually impaired people at Michigan State. On campus, there are audible crossing signs implemented at most intersections, which provides an audible notification of the street name and when the walk sign is on. In addition, there is a outdoor kiosk on campus containing a button that will assist the visually impaired. There are also handheld audible GPS devices that are designed specifically for the blind, but can cost over $700 and would not work as effectively indoors.

In order to make campus more accessible to the visually impaired, design team 6, along with the Michigan State University Resource Center for Persons with Disabilities (RCPD), has developed an interactive positioning system using Bluetooth Low Energy (BLE) beacons. Multiple beacons can be set up around campus that will allow communication with BLE equipped devices such as smartphones or tablets. Using BLE beacons provides a navigation solution at low cost and low power usage. Each beacon will send a signal that can be used to approximate a beacon's distance from the BLE equipped smartphones within a 50 meter range. Once a device receives the signal, an alert can be triggered to give the user a notification or location information. These beacons can be installed at building entrances and exits, in hallways, and inside rooms to provide a useful positioning system.

A benefit for the users of this positioning system is that the Bluetooth communication systems are already implemented in many smartphones, therefore the user will not have to pay for additional hardware. The user will only have to download an application to interface with the beacons. The application can include accessibility options that would allow navigation of the menus and voice notifications.

The cost of the final design arrived at roughly $10. If the beacons were to be used on campus, the number of beacons needed would vary depending on how many points of interest the RCPD wants. In order to distinguish each building at Michigan State, at least 538 beacons are needed and if each entrance and exit needed differentiation, thousands of beacons would be needed. The large scale deployment of the beacon would lower the overall price of individual beacons since components of the design will cost less in bulk orders.

1.2 Background

BLE beacons communicate by broadcasting a packet of data at an interval ranging from 20 milliseconds to 10 seconds using the spectrum range of 2.4-2.4835 GHz. Unlike classic Bluetooth, BLE allows the signal to be broadcast on a range of profiles that can be specified in order to benefit in a particular application. As the name suggests, BLE uses minimal power to broadcast and receive signals allowing extended use of wireless devices without a major effect on battery life. A device will determine the distance from the beacon based on the signal strength. Apple’s operating system, iOS, for iPhones and iPads, has BLE beacon support built in. Google’s Android operating system has not yet implemented native BLE beacon support, but some companies, such as Radius Networks, have libraries available for use on Android.

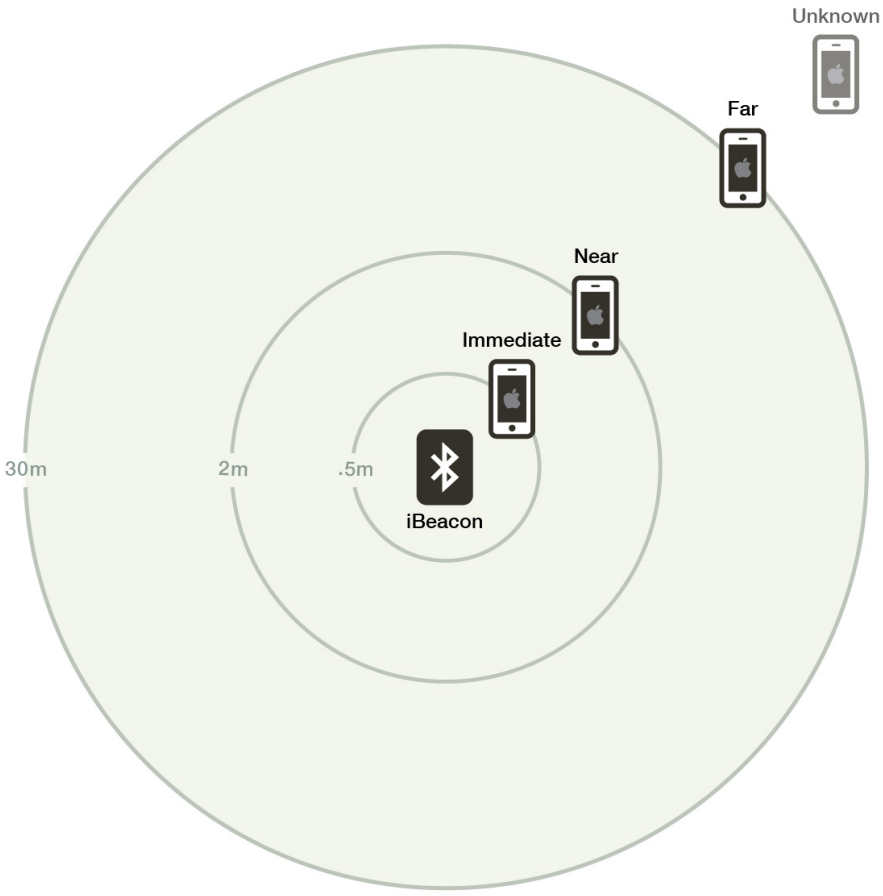


Figure 1: iBeacon Interacts with iDevices

There are several BLE beacons currently on the market, however the cost of these beacons is too high for large scale deployment. One example is the Estimote Beacon which retails for $99 per beacon. There is also a Raspberry Pi based beacon which would cost approximately $40, however software for it is not fully developed. Designs like the Raspberry Pi based beacon add functionality which is not necessary since one of the primary objectives is to design the beacon to be low cost with only the necessary components to accomplish assistive navigation. Qualcomm has brought to market the Gimbal series 10 BLE beacon, which currently cost $5 a beacon, but are not compatible with the iBeacon protocol. Qualcomm is planning to release the Gimbal series 20 which will support the iBeacon protocol, but cost more than the series 10.

Chapter 2 - Objectives and Approach

2.1 Design Specification

This project's goal is to design a Bluetooth Low Energy based beacon to be used for assistive navigation. This beacon will need to be compatible with Apple's proprietary iBeacon technology; the beacon must work with Apple's current iBeacon infrastructure and software APIs. Based on component prices, the cost of the beacon should be less than $20 per beacon, which would be suitable for campus wide deployment. Power will be provided via AC outlets located near the building entrances and exits. Development of mobile applications for iOS and Android devices is also required to demonstrate the functionality of the beacons.

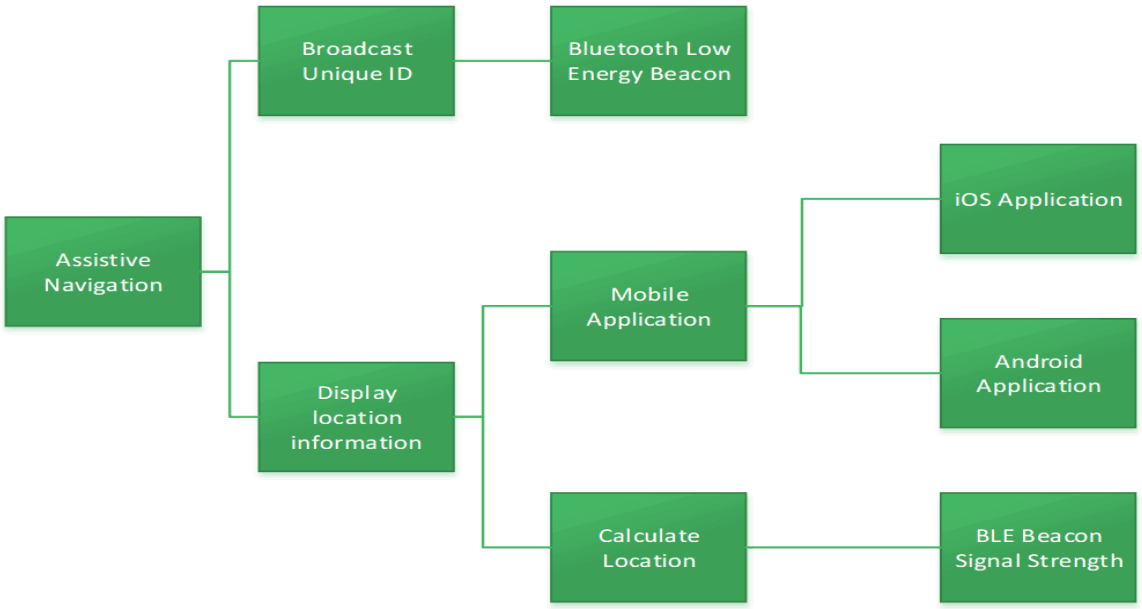


Figure 2: Function Analysis System Technique Diagram

2.2 Conceptual Design Description

Three possible hardware designs have been investigated as possible solutions to the design problem. Each of the designs is based on a different BLE chip. Similar support

hardware is necessary to implement the three chips, the differences among the designs are largely cost of the BLE chips and what functionality they have built into them.

- Design one: Texas Instruments based design
    - This design uses a TI cc2540 BLE chip to implement the Bluetooth functionality. The TI cc2530 is a system on a chip (SoC), meaning no external microcontroller will be needed. The cc2540 BLE chip does need an external antenna to operate.
- Design two: Bluegiga based design
    - This design uses a Bluegiga BLE112 BLE module designed to make implementing Bluetooth functionality straightforward. The BLE112 has a built in antenna and is based on the TI cc2540 SoC BLE chip. The BLE112 requires the fewest number of external components, but has the highest cost.
- Design three: Nordic Semiconductor based design
    - This design is based on the Nordic Semiconductor nRF8001 BLE chip. The nRF8001 has the fewest features of the possible BLE chips as it requires both an external microcontroller and an external antenna. This chip's main advantage is its minimal cost even at low volumes.

2.3 Conceptual Design Rankings

To settle on one of the the three possible designs, the major design criteria of cost, size, and security are ranked by importance on a scale of one to five. Each of the designs are then ranked on how well they meet the design criterias on a one, three, nine scale. These rankings are used to compute a final value which indicates the desirability of each design.

Engineering Criteria	Importance	Bluegiga module design	TI chip design	Nordic chip design
Cost	5	3	9	9
Size	1	9	9	3
Security	3	9	9	9
Total		51	81	75

Table 1: Solution Selection Matrix

According to the table 1, the total score of three designs conclude that the best choice for the product would be the TI chip design. Its major advantage would be its low cost, size, and similar performance which would match other two design. However, the scores of the Nordic and TI designs are relatively close.

The feasible designs are ranked based on engineering criteria of cost, size, security, development time, and manufacturability. The importance of the criteria is rated on a scale of one to five. Then the designs are rated based on how well they meet the criteria on a one, three, nine scale.

Engineering Criteria	Importance	TI chip design	Nordic chip design
Cost	5	9	9
Size	1	9	3
Security	3	9	9
Development time	5	9	3
Manufacture	3	3	3
Total		135	99

Table 2: Feasible Design Matrix

Table 2 indicates that the TI design is the most feasible design. The TI based design requires less development time because the Bluegiga module can be used for developing the software and firmware while the hardware for the TI design is being developed. The Bluegiga module uses the TI cc2540, the same programmer as the TI design, and includes a built in antenna. Because the Bluegiga module uses the TI cc2540, the firmware can be easily ported over to a TI based design, allowing parallel development of the TI hardware and firmware.

2.4 Solution Description

Based on the design requirements and the rankings of the possible designs, the TI cc2540 based design was selected. Initially a prototype beacon was built using the Bluegiga BLE module because it includes all the necessary components to begin software development. This allowed the group to become familiar with the BLE and the iBeacon protocol while concurrently developing the more complex hardware of the TI design.

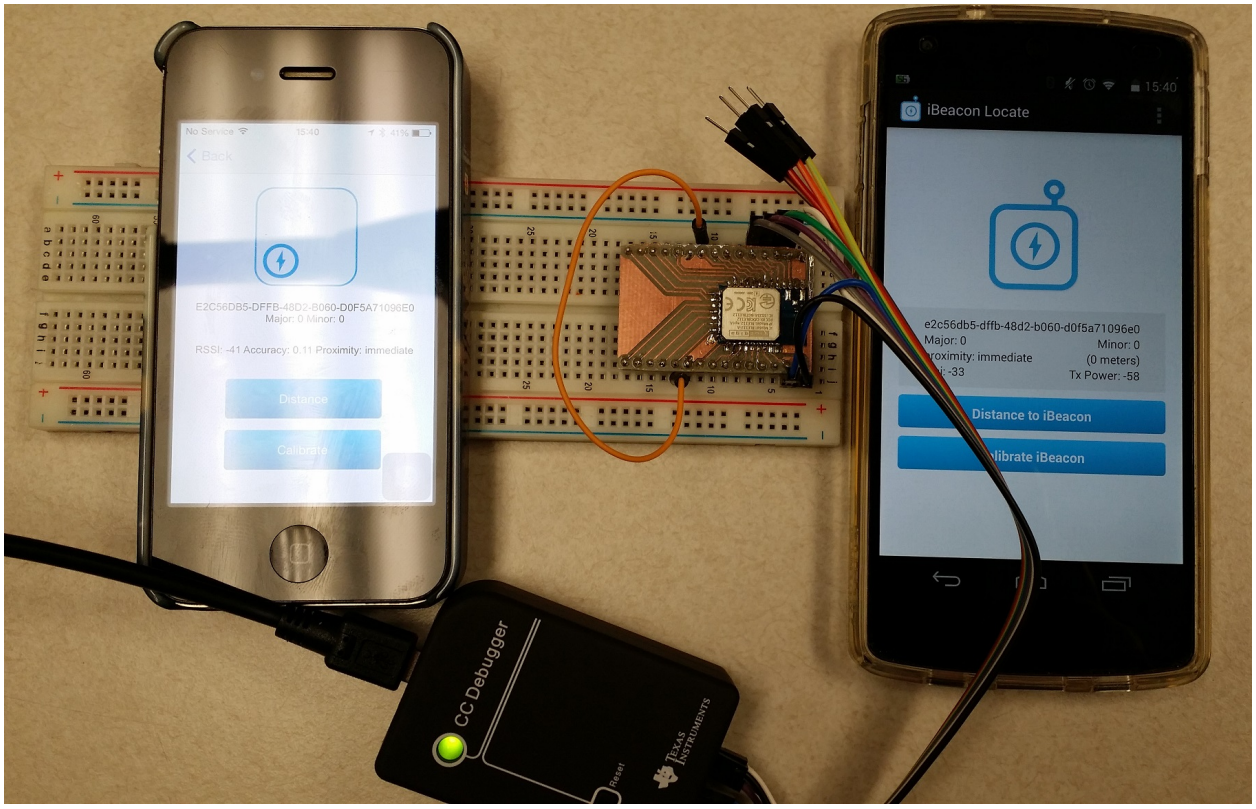


Figure 3: Bluegiga BLE module Interacting with iPhone and an Android Phone

The final beacon design focused on creating a low cost solution, with minimum impact to the performance. A custom PCB was created to implement the support circuitry required by the cc2540. Cost saving measures such as using a PCB antenna instead of a discrete IC based antenna and a 2-layer PCB instead of the 4-layer PCB common in RF designs. The simple operation of the BLE beacon helped keep the component cost low.

2.5 Budget

The project came in under the proposed budget by almost $60 (refer to Table 3 for the final budget and Appendix III - Table 1 for the projected budget). This is mostly attributed to consolidating orders to reduce shipping costs and using the same programming and debugging tool for all of the prototypes and final design.

Item	Cost
Bluetooth Dev. Boards	$63.95
Programming / Debug Tool	$47.50
Electrical Components	$140.50
Shipping	$20.83
TOTAL	$272.78

Table 3: Final Budget

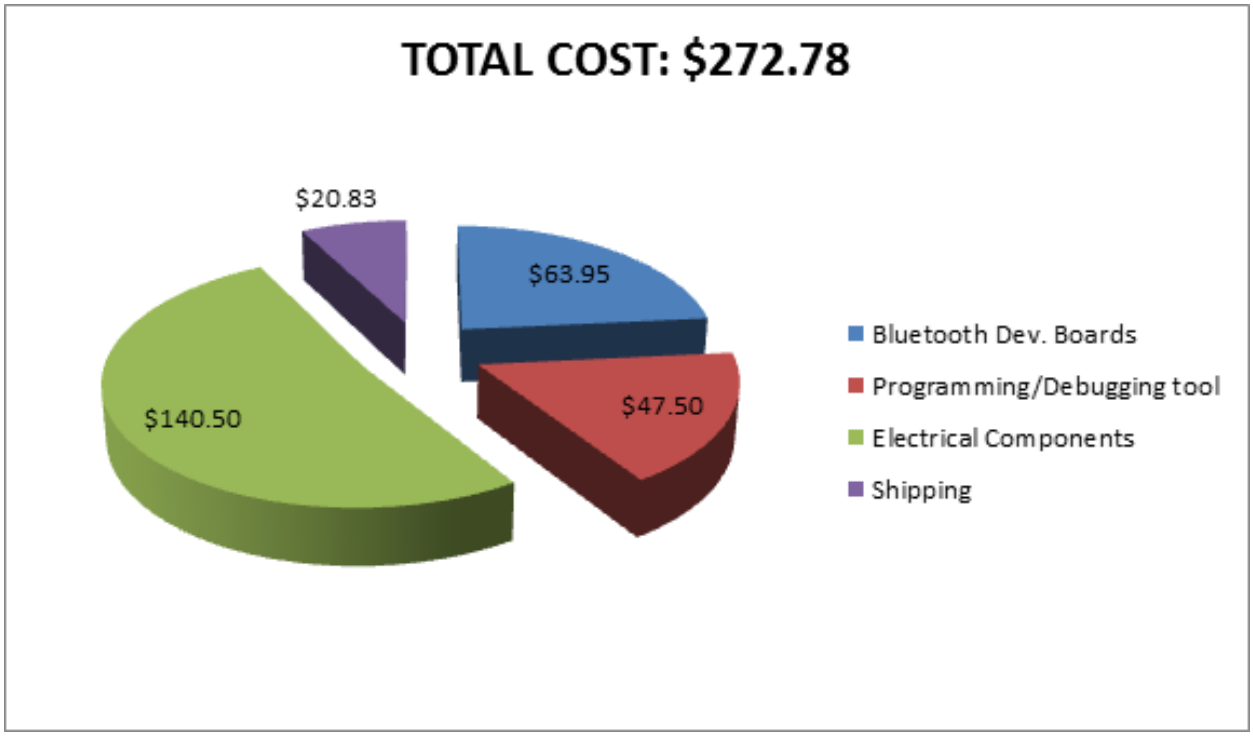


Figure 4: Project's Budget

2.6 Proposed Schedule

The original schedule was ambitious, requiring the development and testing of 4 prototypes and 1 final design in the span of a month and a half (refer to Appendix III figures 2 and 3 for original gantt chart). This was paired down to a more reasonable schedule as it became apparent that there was not a need for 4 prototypes.

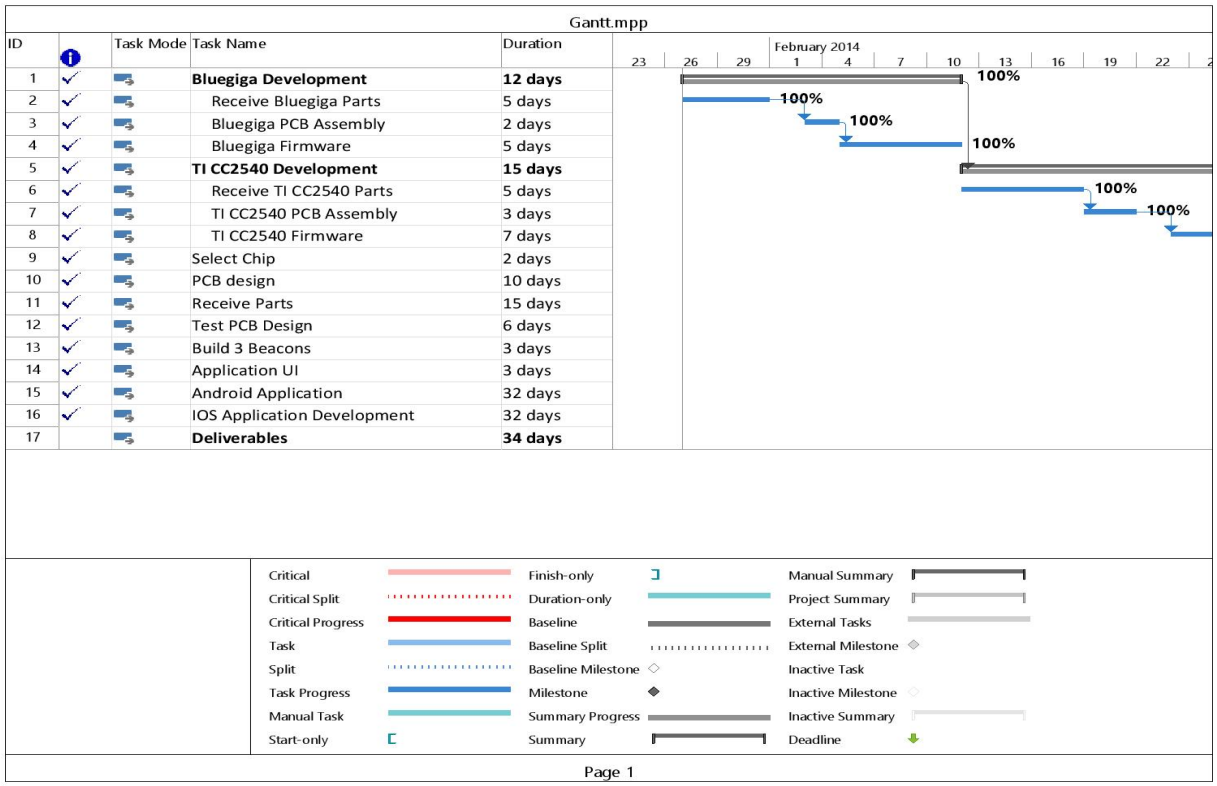


Figure 5: Page 1 of Final Gantt Chart

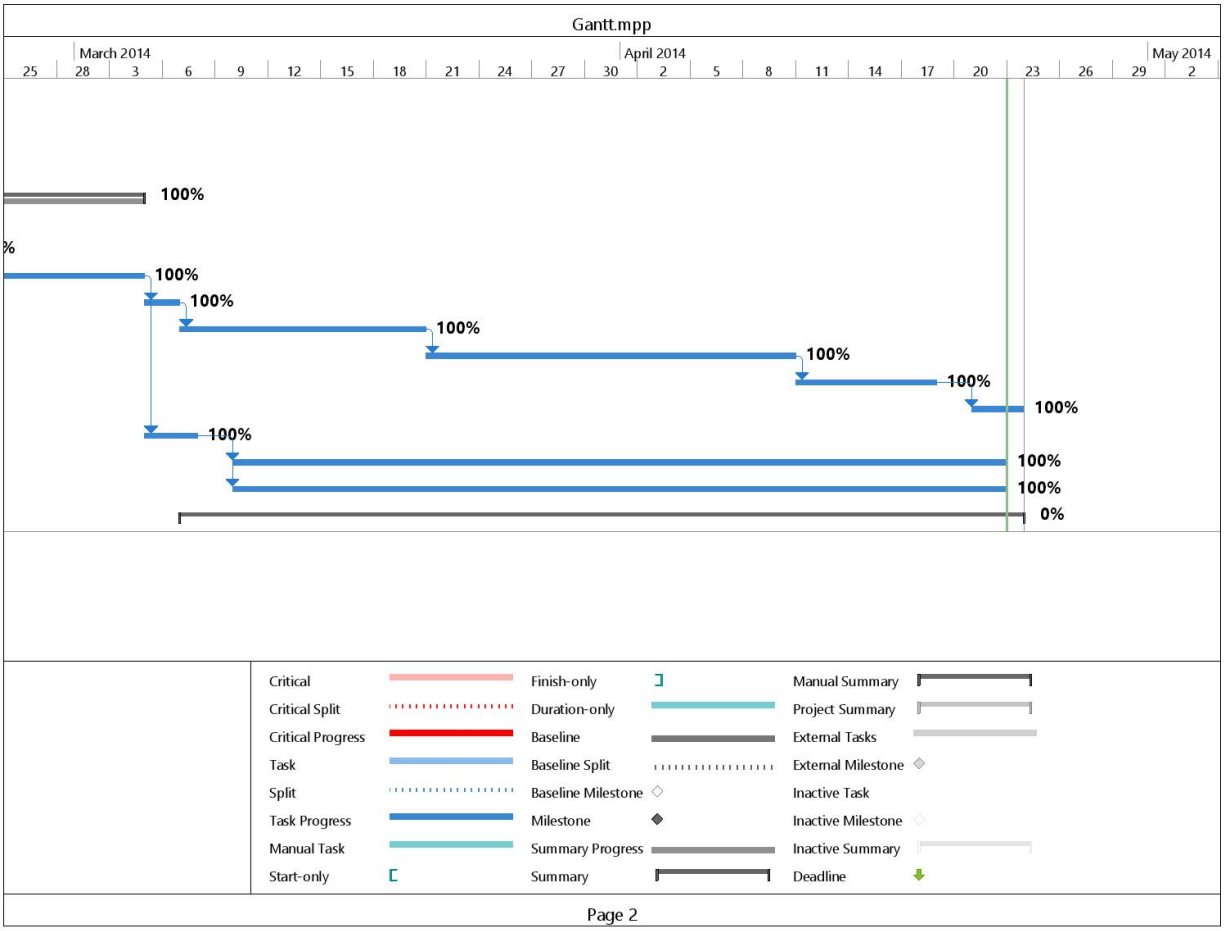


Figure 6: Page 2 of Final Gantt Chart

The time required to create and manufacture a PCB was also underestimated in the original schedule. Only 1 day was allocated for designing the PCB and 5 days for receiving parts and the PCB from the manufacturer. The PCB design process ended up taking 10 days due to the complexity of designing a PCB which is capable of broadcast RF without distortion. Manufacturing the PCBs took 2 weeks from the point the files were sent to the manufacturer until the PCBs were received. To complete two prototypes during the design period, pre-made development boards were utilized in combination with custom made design adapter PCB which were simple enough to be milled by the ECE shop, reducing the manufacturing time to one day. This strategy allowed for two hardware prototypes to be created during the design period and still leave enough time for software development and testing.

Chapter 3 - Technical Implementation

3.1 Hardware

Two prototype beacons were necessary to complete the design of the final beacon. Each prototype was based around a different BLE module and required the design of a custom adapter to facilitate prototyping.

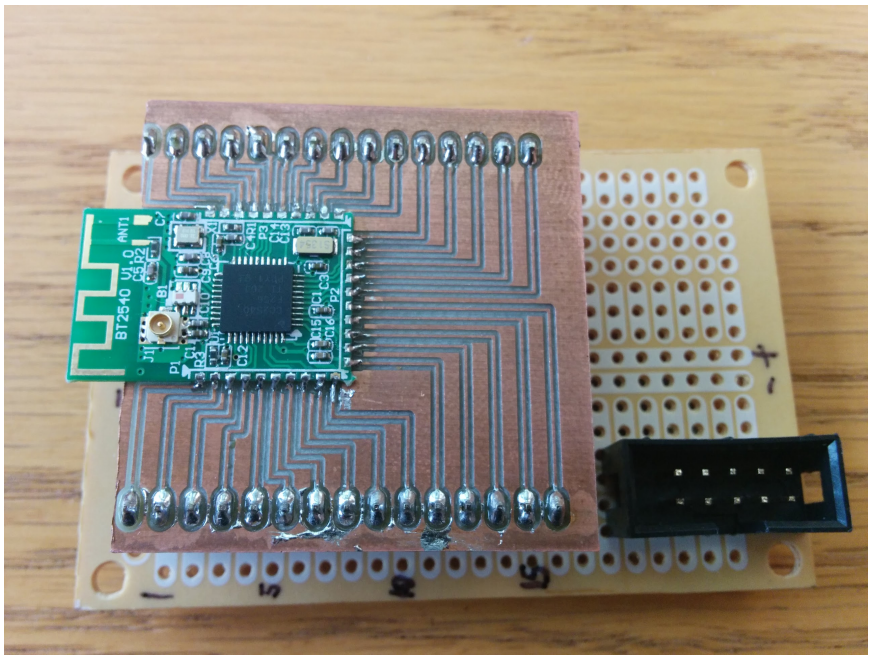


Figure 7: TI cc2540 based prototype with programming adapter

The final design required the design and manufacturing of its own custom PCB so the low-cost objective can be achieved. In order to minimize the final size and cost of the PCB a programming header was not included, instead the pins required for programming were broken out to vias on the PCB. To program the beacon a separate programming PCB was designed with pins which mate with the programming vias on the main PCB.Two versions of the final design were constructed to test the difference between using a PCB based antenna and a discrete IC based antenna and select the best one for different scenarios.

Another cost saving measure was using a 2-layer PCB; RF designs normally employ a 4-layer design to allow for an unbroken ground plane for EMI reduction. To keep the 2-layer

design from impacting the RF performance, all the traces were done only on the top layer to allow the bottom layer to be used exclusively for the ground plane.

Besides the solid ground plane, the final design also employ the use of via stitching around the RF path to isolate it from other noise on the PCB. The RF path itself is kept short to reduce its exposure to EMI before it reaches the antenna. The RF path uses an inductor and capacitor based impedance matching network. This is used to reduce any reflection of the RF signal which would happen if the path was not impedance matched. Another option was to use a BALUN, but the inductor and capacitor method was chosen to reduce cost.

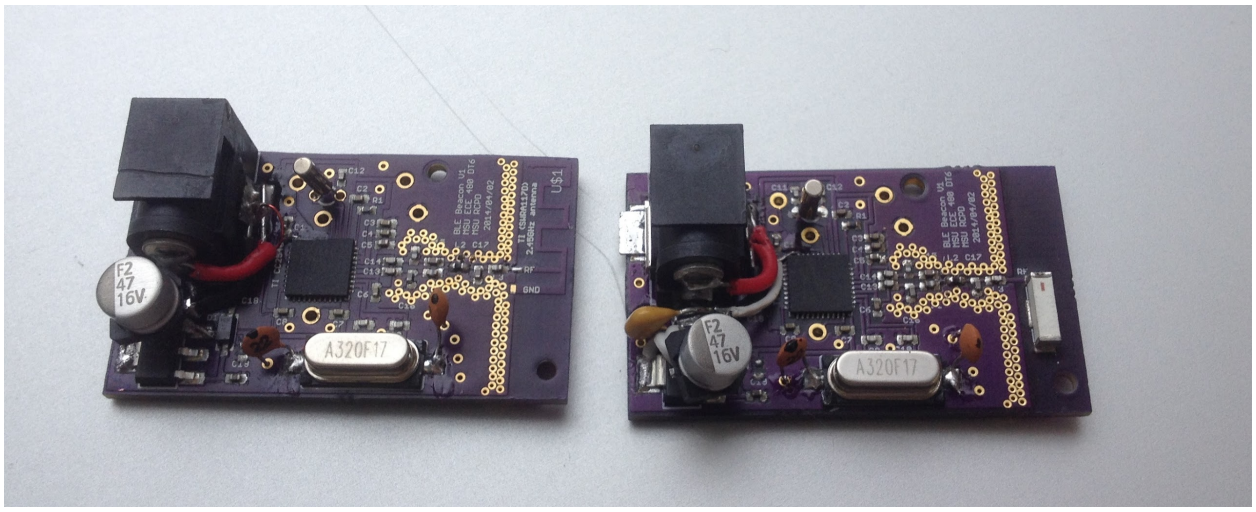


Figure 8: Final Design. From left to right, the PCB antenna version and the discrete antenna version

3.2 Software

Several software components were required to complete this project including the firmware, iOS, and Android application. Firmware is an important component, as it serves as a bridge between our hardware and mobile phone application, and controls the operation of the cc2540 BLE chip. The cc2540 does not require an external microcontroller, instead the BLE chip itself is capable of running code. The firmware is written in C and is based on an example application created by TI called the *Simple BLE Broadcaster*. To implement the iBeacon protocol several changes were made to the existing program. First the generic access profile was modified to allow the beacon to

continuously broadcast the iBeacon standard packet and the scan response was removed as iBeacons don’t respond to scan requests with a data packet. Next the advertising data for the iBeacon packet is added, this includes the Preamble, Universally Unique Identifier(UUID), Major, Minor, and 1 meter TX power according to the calibration of the final hardware. Other changes were made to improve performance such as changing the advertising interval to 50 milliseconds and the broadcast power to 0 dB. In order to reuse the written firmware to program multiple devices, a python script was created. This script allows the UUID, major, minor and TX power to be changed without recompiling the firmware and can be applied these new settings quicker on the ibeacon.

Both the Android and iOS applications read the information of any broadcasting beacons in the vicinity. These beacons are divided into proximity groups of immediate, near and far. Beacons that are immediate are within 0.5 meters, beacons that are near are within two meters, and beacons that are far are beyond two meters away from the device. Since each beacon is uniquely identified by the UUID, major, and minor, the applications can customize reactions to the device’s proximity to the beacons. This is a feature exemplified by the application opening the design teams website when the device is within 0.5 meters of the beacon with major: 1 and minor: 2.

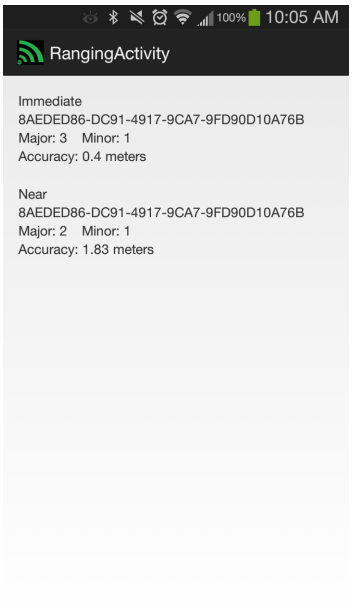


Figure 9: Android Application UI

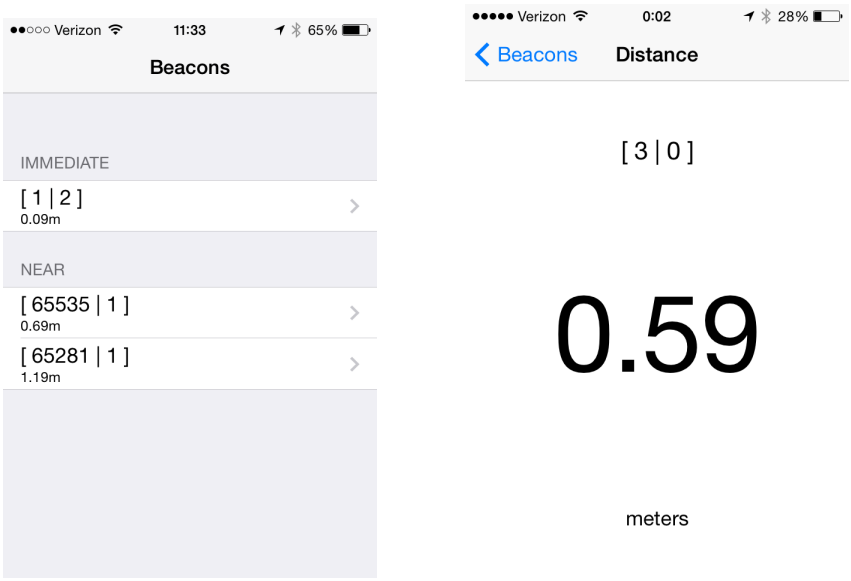


Figure 10 & 11: iOS Application UI

The iOS application uses an API directly from Apple. The Android application developed to use with the beacons was developed with Radius Networks' Android Beacon Library. This library, along with a minimalist reference application, again from by Radius Networks, provides a simple to use API and existing code to aid in creating a customized application. There are some small, noticeable differences in the 2 APIs, the most obvious is that the Apple averages the values of the distance over time so that the distance readings are more consistent. This is overcome in Android by programming an average to be calculated over time. The end result is the same, but in iOS the functionality is built in. As with the hardware limitation of a BLE capable device, the software also has device limitations. The iOS API requires iOS 7.0 or later, while the Android API provided by Radius Networks requires Android 4.3 or later. Therefore, these applications would work on 87% of iDevices and 14.2% of Android Devices.

Chapter 4 - Test Data and Results

4.1 Testing

Testing was completed in several phases to ensure that the beacon complied with the iBeacon protocol and had a performance roughly equivalent to other beacons. The first test, packet capture test, was a packet capture test. A USB BLE adapter was used in conjunction with TI's SmartRF Packet Sniffer application to get a view of the raw packets being broadcast by the beacon. These were compared with packet capture from iOS devices acting as beacons and packet captures of the two prototype beacon. The beacon's identification information (UUID, major, minor) was modified several times to test that the beacon properly broadcast as an iBeacon in the full range of possible major and minor values.

The second test, connectivity test, was a basic connectivity verification procedure. Several devices were used to confirm that they could detect the iBeacons being broadcast. The beacon was tested with an iPhone 4S, iPod Touch 5th Generation, HTC one, Samsung Galaxy S3, LG Nexus 5, and Google Nexus 7. Again the beacon's identification information (UUID, major, minor) was modified several times to test that the beacon properly broadcast as an iBeacon in the full range of possible Major and Minor values.

The third test, distance accuracy test, was to determine the accuracy of the distance measurements. The beacons were placed in a fixed location and several devices including an iPhone 4s and other Android devices were placed at measured distances away from the beacons. The tests were conducted at 1 meter, 2 meters, and 5 meters. The devices were allowed several seconds to stabilize on a distance reading and these readings were compared to the actual distance from the beacon. This test evaluated the accuracy of the final beacon compared with the prototype and iPhone beacon.

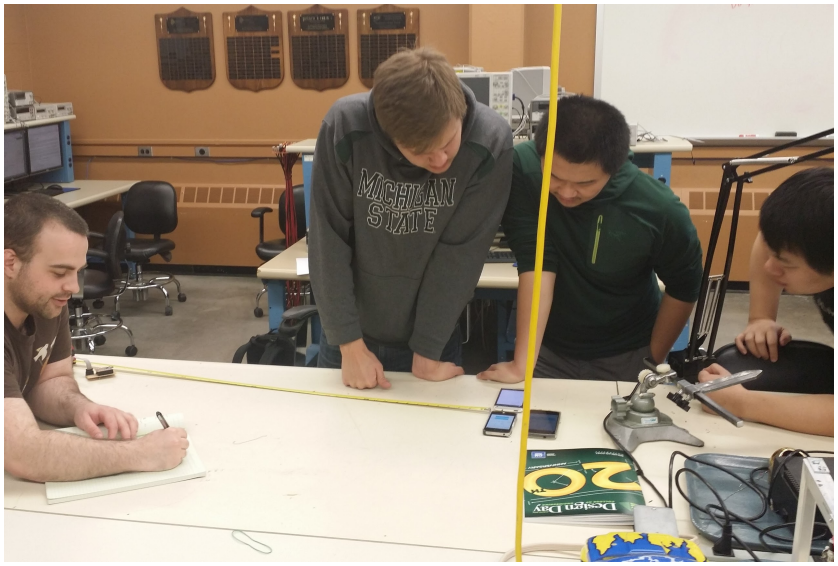


Figure 12: Team working on testing

The fourth test, maximum distance test, was to compare the relative maximum distance of the beacon. Three beacons were setup in the same area, each with different identification information. A Google Nexus 7 and LG Nexus 5 were used to detect the beacons. The Nexus 7 and Nexus 5 were then moved away from the beacons until the beacons could no longer be detected. This test evaluated the maximum broadcast distance of the final beacon in comparison to the prototype beacons and iPhone beacon.

4.2 Results

The packet capture test showed the final beacon is broadcasting iBeacon packets consistent with the iBeacon protocol. The packets were formed properly with the required iBeacon information such as the preamble, major, minor, 1 meter TX value, and checksum.



Figure 13: Beacon Packet Capture

The connectivity test also yielded positive results. All devices tested were able to detect the beacons. Changing the identification information worked as expect with all the devices displaying the new major and minor values of the beacon. This confirmed what the packet capture showed, that the beacon was broadcasting iBeacons as expected.

The distance accuracy test results showed the beacon's accuracy varied widely based on which device was taking the distance measurements. Devices based on iOS are showing a more stable and accurate result. This is due to their iOS provided APIs being more robust and their use of averaging. Android devices using the Radius Networks iBeacon API showed large jumps in the distance measurements, so averaging was utilized inside the Android application to come up with a more consistent distance measurement. The final beacon had an accuracy comparable with the other prototypes and iPhone 4S based beacon.

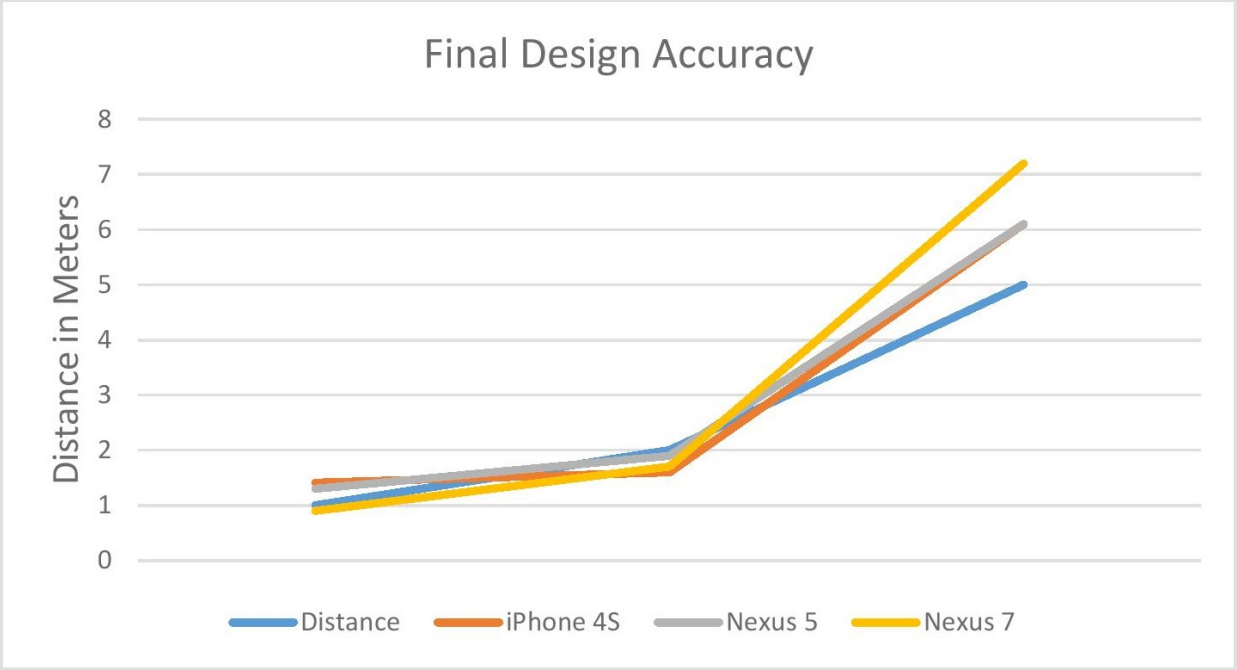


Figure 14: Final Design Accuracy Test Results

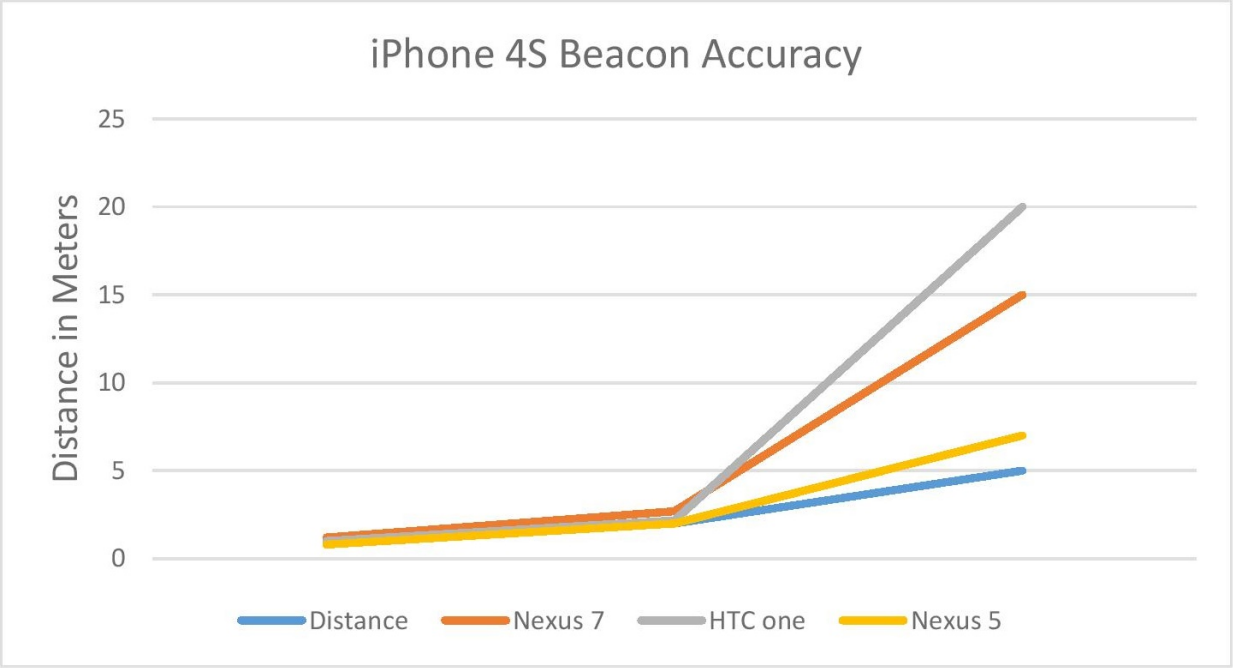


Figure 15: iPhone 4S Beacon Accuracy Test Results

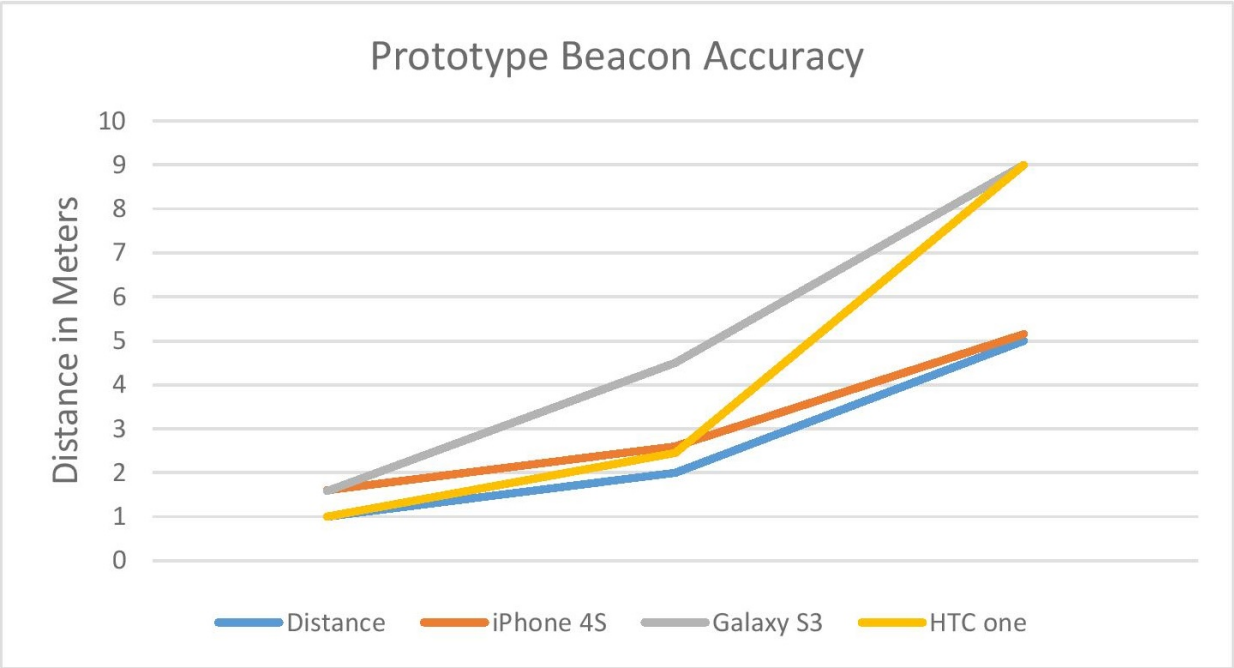


Figure 16: Prototype Accuracy Test Results

The maximum distance test also yielded results comparable to other beacons. The devices failed to detect the final beacon design within 2 meters of losing contact with the iPhone 4S and prototype beacon. The devices were reading a distance of an average of 50 meters when it failed to detect. This shows that the final beacon has a maximum distance that is nearly the same as the other beacons.

Chapter 5 - Conclusion

5.1 Final Cost

In total, after developing two prototypes and the final design, the final cost of the project was $272.78. The proposed cost of the project was $330, therefore the project was completed $60 under budget. The actual cost of the final design is $16.45. It fits in the price range given by the sponsor RCPD and If the beacons were used on a large scale throughout campus the price per unit of the beacon would go down due to the discounted prices of components ordered in bulk. The price per unit for constructing 500 beacons would be $9.92, which meets the design objective better.

5.2 Schedule

The schedule changed significantly from the initial proposed schedule. The original plan of creating 4 prototypes plus the final design was not feasible in the allotted time due to the time required to manufacture and redesign a PCB. The final schedule reduced the number of prototypes to 2 and focused on using pre-made hardware for the prototypes to reduce the time required to manufacture them. This strategy allowed the hardware prototypes to be created during the design period and still permitted sufficient time for software development and testing.

5.3 Project Findings

Overall, the project is a success since all of the design specifications were achieved. The cost per beacon is well under $20 and the beacon allows for reusability of the code and design by the RCPD for future developments. The beacon performs well enough for it designed purposes however the accuracy of the distance from beacon to receiver was not as stable as expected.

Even with success, there are still areas of improvement that can be made on the beacon. Improvements can be done on the PCB layout for the components.There are four layouts for capacitors that need to be added on the PCB layout to allow the beacon to properly function. In addition, the polarity layout for the DC jack needs to be reversed. The team was able to fix these layout errors by soldering components over the PCB but future revisions should fix the errors on the actual PCB layouts. For future iterations, the beacon can be design to be battery powered.

In regards to software, a database needs to be implemented in order to match the UUID, major byte and minor byte to a name for a precise location. More accurate averaging algorithm is needed for the Android side to further increase the distance measured from the beacons and triangulation could be added to improve it even more. On the firmware side, it may be possible to update the firmware remotely and prevent tampering at the same time using OAD method, but it requires more work on the encryption of the firmware on both ends.

Appendix I - Technical Roles

Nicholas Blackledge



Is Design Team 6's Manager and created the PCB designs. As the Manager he was responsible for keeping the team organized, focused, and on schedule. He managed communications with the sponsor, the Michigan State University Resource Center for Persons with Disabilities, along with the team's facilitator and the ECE shop. He worked closely with Adi Mathews from the RCPD, who created the iOS application to demo the beacons. Nicholas designed the PCB required for both prototyping and the final board. The final PCB required the use of techniques to reduce the impact of EMI on the beacon's performance. This required learning about proper RF PCB design. Nicholas also help reduce the cost and manufacturing time of the final design by reducing the 4-layer PCB design to a 2-layer PCB without a significant impact on the RF performance. He all the required traces on a single layer in order to reserve the bottom layer for a solid ground plane. Nicholas was also in charge of selecting and ordering the parts for the prototypes and final designs, selecting parts based on the required specifications while keeping the final cost down. As well as assembling the boards which utilized 0402 surface mount parts and a 40-pin QFN package for the main cc2540 chip.

Nicholas also worked on creating the firmware for the TI cc2540 beacon and the Bluegiga prototype. Helping to implement the iBeacon protocol in the Bluegiga scripting language and then porting that code to C as required for the cc2540. He also created a python script to allow for easy customization of the final firmware without recompiling. He used a HEX editor to find the values of the UUID, Major, Minor, and TX value and then wrote a python script which also those values to be altered, recalculates the necessary checksums, and outputs the customized firmware.

Chunyang Chu



As the web coordinator of the team, Chunyang's non-technical role was to prepare and maintain the team's webpage. He was also in charge of designing the team's poster of Design Day.

Apart from his non-technical roles, Chunyang's technical contributions to the team included several aspects. He was in charge of the iOS development for demonstrating functionalities of the team's hardware design products. Chunyang had no previous experience with iOS development before working on this project, but had taken several programming courses for Python, C and C++. He worked together with Aditya Matthew from MSU RCPD to develop a basic iOS application, which can detect several beacons in range, display their UUID, major and minor bits as well as Tx Power value, then calculate the approximate distances from the iDevice to these beacons.

Chunyang helped the team with researching the Bluetooth Low Energy packet details as well. The packet details were crucial for several reasons. It needed to be set so that the beacons are in correct advertising mode. More importantly, it determined how the information should be transmitted to ensure the beacons obey iBeacon protocols. Chunyang also participated in the testing phase of the project. Along with his teammates, Chunyang tested the performance of their design focusing on the accuracy of the distance calculation, and the maximum range the beacons can detect. By performing various tests with different mobile devices including iPhone 4S, Google Nexus 5 and Nexus 7, HTC One and Samsung Galaxy S3, he and his teammates were able to confirm that their design worked nearly as well as the iDevices as beacons.

Steven Le



As the presentation preparer, Steven was responsible for setting up and organizing the PowerPoint for presentations. The PowerPoint presentations were design to only show the necessary information and not clutter the slides.

Aside from Steven's non-technical role, he had a technical role to incorporate a low cost antenna that would a low cost antenna that would work with the Bluetooth Low Energy beacon design. In order to meet design criteria and constraints, Steven researched to find the best solution. He found two suitable designs, using a PCB trace antenna and a ceramic chip antenna. The PCB trace antenna was the antenna chosen for the final design since it provides low cost implementation. The next stage in development that Steven dealt with was regarding impedance matching circuit with the antenna to prevent power losses. In addition to the PCB trace antenna, the other design that Steven found, using the ceramic chip was pursued for a backup beacon. Steven's implementation of the PCB trace antenna made it possible to lower the overall price of the beacon. In addition, Steven assisted in troubleshooting other hardware related issues that came up in the project.

A problem that Steven encountered during his role was he was not able to test the antennas implemented on the final designs. His previous experiences with the spectrum analyzer and past electromagnetic courses would allowed him to test the antennas, however without access to a spectrum analyzer that could go as high as 2.4Ghz, testing on the individual design could not be done. In the end, Steven's design did work for its function however the performance was not measured.

Matt Smania



Matt's team roles included document preparation and android programmer. As document preparation manager, he proof read the many assignments throughout the semester after the content was written. Since he was always looking at the schedule to see when assignments were due, Matt would make sure that the team knew of any assignments, checkpoints, and presentations that were due in the coming weeks. Matt contributed to every document and assured it was of a professional quality.

As the lead Android programmer, and one of two Computer Engineers, Matt utilized skills learned throughout his college career. Since Matt was concurrently taking CSE 476, Intro to Mobile App Development, he learned many of the basic skills he needed to develop an Android application that was comparable to that of an iOS device. He did a thorough search through the Radius Networks' website to find the necessary information to use their Android iBeacon Library. Matt based the final application on the Minimalist Reference Application from Radius Networks. After reading through that application and with inspiration from Adi Mathew, Matt created an application which would display the uses of iBeacons on Android. His brief history with android applications allowed him to create and application with an intuitive design. The application is clearly commented so that when handed over to the RCPD, future developers of the project will know clearly what is happening throughout the program. After finishing the application, he also was able to suggest ways to improve upon the application for the future iterations. Matt's ever growing knowledge of Android was unmatched in the group and made him an irreplaceable part of the team.

Bowei Yu



As the lab coordinator and circuit designer, Bowei Yu contributes a lot in this project in terms of hardware research and firmware exploration. He helps with the poster design of the final design day, making updates and formatting varies kinds of documents needed for this project including proposal, application note and final report. In the early stage of this project, he dedicated in researching any possible solution to achieve the objectives obtained from the sponsor RCPD. Also, after done with the research, he aided the team to decide which proposed solution should the team to dig deeper by updating the thread setup on the Google group.

After selecting three candidates, Bowei contributes more in the circuit designing part. He works with the team to verify and test the beacon layout and final beacon. He also setups up and collaborates with team members on the testing work for the prototype and final beacon. Four testing are done in an accurate way and helps the calibration of the final ibeacon to be more precise on distance measurement. On the software side, Bowei contributes to the software UI design and firmware programming. He discussed with team members and contributes part of what should the application UI look like on the mobile phones and research into algorithms that help to stabilize the rssi received from the ibeacon. On the firmware part, Bowei also contribute in programming and improving the code. He read through the TI software guide and verifies every line of the firmware code to be functional and reasonable. The original example comes from TI official website and the program is containing a lot of redundant codes that are not necessary for this project. Bowei does a lot of work coding this part and after doing that, he tests the final version of the firmware to be fully functional on the ibeacon model.

Appendix II - Reference Materials

"Apple Developer." *App Store Distribution*. Apple, Inc., 20 Apr. 2014. Web. <https://developer.apple.com/support/appstore/>

"Apps That Know Where You Are: Our Experimentation With Apple's IBeacon Technology - Inside the Nerdery." *Inside the Nerdery*. Nerdy, 19 Nov. 2013. Web. Jan. 2014 <http://blog.nerdery.com/2013/11/nerdery-labs-ibeacon-experiments/>

"BLE112 Bluetooth Smart Module." BLE112 Bluetooth Smart Module – Bluegiga. Bluegiga. Jan. 2014 <http://www.bluegiga.com/en-US/products/bluetooth-4.0-modules/ble112-bluetooth--smart-module/>.

"CC2540 2.4GHz Bluetooth Low Energy System-on-Chip Solution." Bluetooth Low Energy - CC2540. Texas Instruments. Jan. 2014 <http://www.ti.com/product/cc2540>.

"CLBeacon Class Reference." *CLBeacon Class Reference*. Apple, Inc., 18 Sept. 2013. Web. <https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CLBeacon_class/Reference/Reference.html>

"Gimbal Proximity Beacons." Qualcomm Gimbal Beacons. Qualcomm. 30 Jan. 2014 <http://www.qualcomm.com/solutions/gimbal/beacons>.

"How to Choose a Bluetooth Smart / Low Energy Development Kit." Bluetooth Smart. 5 Oct. 2011. Jan. 2014 <http://blog.bluetooth-smart.com/2011/10/05/bluetooth-low-energy-development-kits-2/>.

Johannsen, Jens Willy. "Jenswilly.dk." Jenswilly.dk Microcontrollers and embedded electronics. 3 Sept. 2013. Jan. 2014 <http://jenswilly.dk/2013/09/ibeacons-ready/>.

Kosner, Anthony Wing. "The Internet of iThings: Apple's iBeacon Is Already In Almost 200 Million iPhones And iPads." Forbes. 15 Dec. 2013. Forbes Magazine. Jan. 2014 <http://www.forbes.com/sites/anthonykosner/2013/12/15/the-internet-of-ithings-apples-ibeacon-is-already-in-almost-200-million-iphones-and-ipads/>.

"NRF8001, µBlue, Bluetooth low energy." NRF8001 Bluetooth Low Energy. Nordic Semi Conductor. 30 Jan. 2014 <http://www.nordicsemi.com/eng/Products/Bluetooth-R-low-energy/nRF8001>.

"Platform Versions." *Android Developers*. Google, 1 Apr. 2014. Web. <http://developer.android.com/about/dashboards/index.html>

Townsend, Kevin. "PiBeacon - DIY iBeacon with a Raspberry Pi." Learn Adafruit. Adafruit. Jan. 2014 <http://learn.adafruit.com/pibeacon-ibeacon-with-a-raspberry-pi/overview>.

Young, David G. "Radius Networks." Radius Networks. 1 Oct. 2013. Radius Networks, inc. Jan. 2014 <http://developer.radiusnetworks.com/2013/10/01/reverse-engineering-the-ibeacon-profile.html>.

Appendix III - Technical Documentation

Item	Estimated Cost
Bluetooth Low Energy Chips	$60
Other electrical components	$70
Programming / Debug tools	$100
Three demonstration beacons	$50
Shipping	$50
TOTAL	$330

Table 1: Projected Budget

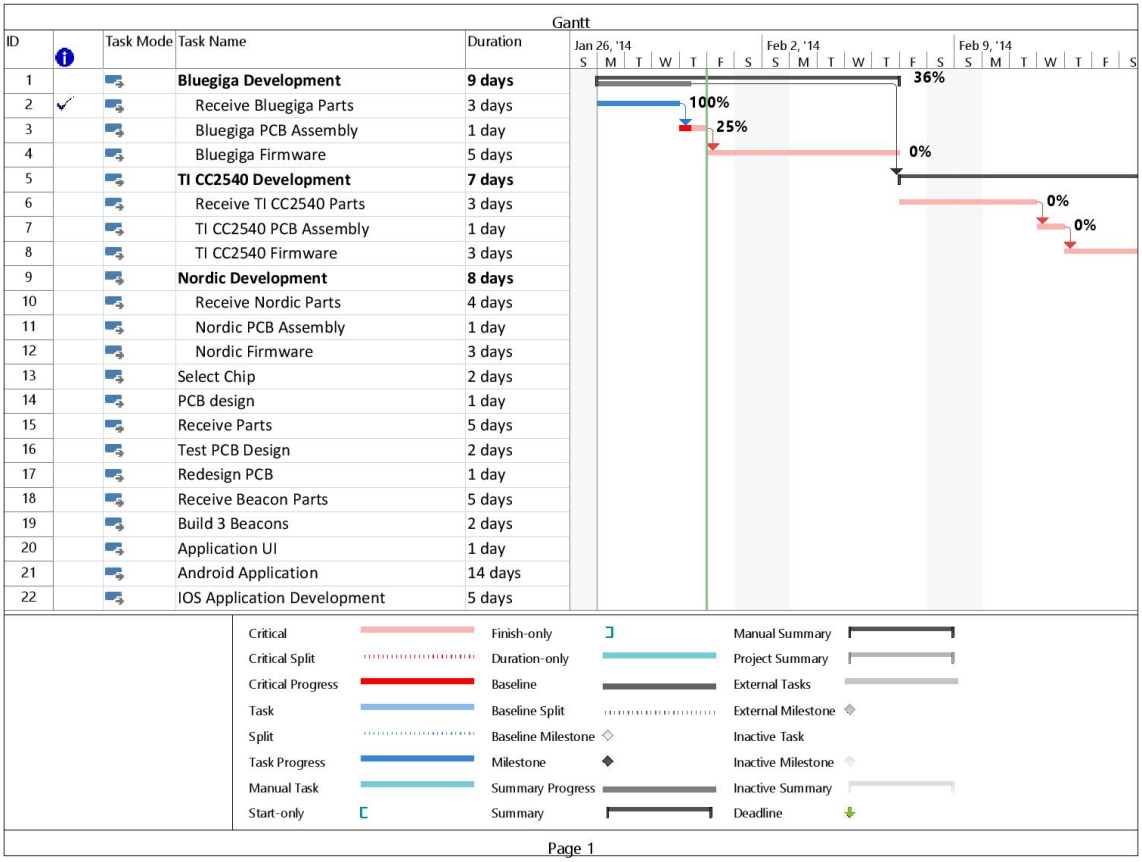


Figure 1: Page 1 of Original Gantt Chart

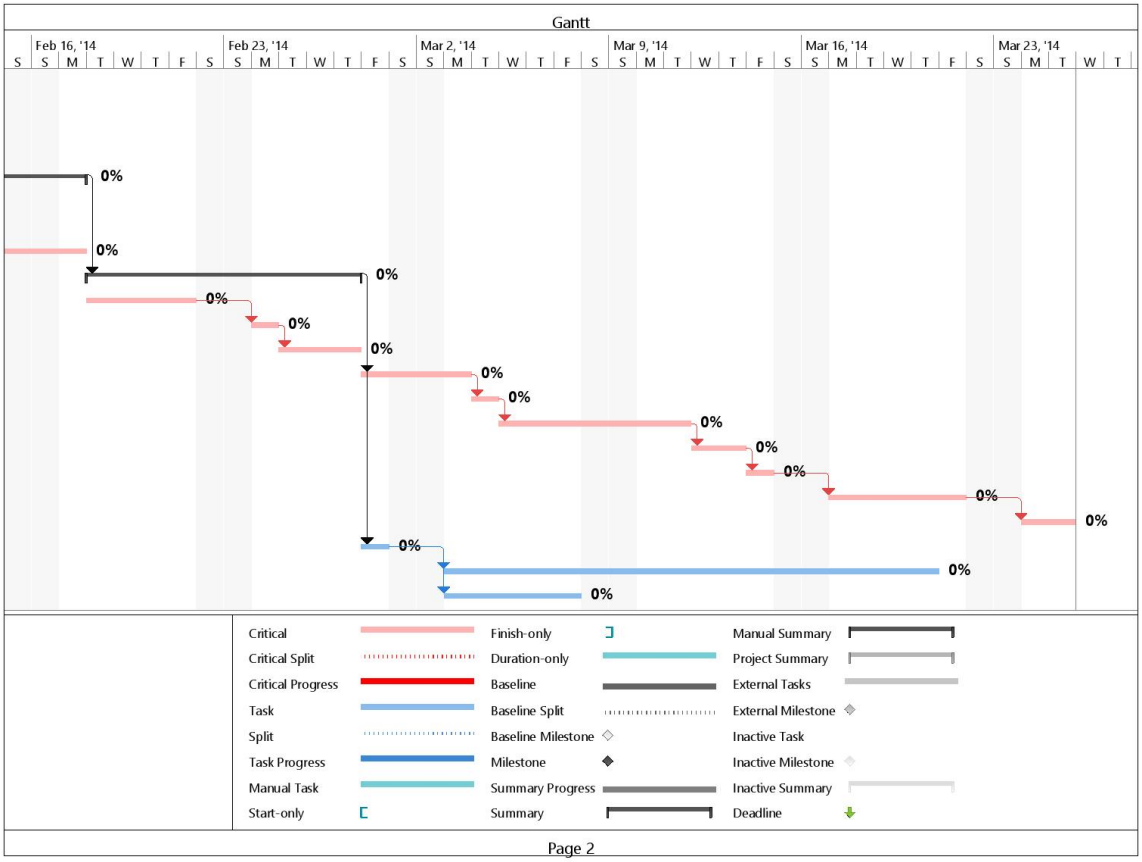


Figure 2: Page 2 of Original Gantt Chart

Component	Quantity	Cost per unit for order of 1	Cost for 1 Beacon	Cost per unit for order of 500	Cost for 500 Beacons, per beacon
32 MHz Crystal	1	0.43	0.43	0.225	0.225
32.768 kHz Crystal	1	0.25	0.25	0.116	0.116
LM3940	1	1.53	1.53	0.728	0.728
cc2540	1	5.75	5.75	3.07	3.07
100 nF Capacitor	6	0.01	0.06	0.01	0.06
1 pF Capacitor	3	0.02	0.06	0.02	0.06

1 uF Capacitor	1	0.10	0.10	0.10	0.10
12 pF Capacitor	2	0.16	0.32	0.016	0.032
15 pF Capacitor	2	0.08	0.16	0.04	0.08
18 pF Capacitor	2	0.08	0.16	0.04	0.08
47 uF Capacitor	1	0.17	0.17	0.044	0.044
220 pF Capacitor	1	0.32	0.32	0.104	0.104
0.47 uF Capacitor	1	0.11	0.11	0.027	0.027
56 kOhm Resistor	1	0.08	0.08	0.021	0.021
2 nH Inductor	2	0.10	0.20	0.03	0.06
1 nH Inductor	1	0.10	0.10	0.051	0.051
3 nH Inductor	1	0.10	0.10	0.033	0.033
Diode	1	0.39	0.39	0.153	0.153
DC Jack	1	1.48	1.48	1.23	1.23
Charger	1	4.68	4.68	3.65	3.65
TOTAL			$16.45		$9.924

Table 2: Component Cost of Beacons

Firmware for cc2540 Beacon

```
/*********************************************************************
 * INCLUDES
 */

#include "bcomdef.h"
#include "OSAL.h"
#include "OSAL_PwrMgr.h"

#include "OnBoard.h"
#include "hal_adc.h"

#include "hci.h"
#include "gap.h"

#include "devinfoservice.h"
#include "broadcaster.h"

#include "simpleBLEBroadcaster.h"

/*********************************************************************
 * CONSTANTS
 */

// What is the advertising interval when device is discoverable (units of 625us, 160=100ms)

#define DEFAULT_ADVERTISING_INTERVAL          160

// Company Identifier: Texas Instruments Inc. (13)
#define TI_COMPANY_ID                         0x000D

// Length of bd addr as a string
#define B_ADDR_STR_LEN                        15


static uint8 simpleBLEBroadcaster_TaskID;   // Task ID for internal task/event processing

  // GAP - Advertisement data (max size = 31 bytes, though this is
// best kept short to conserve power while advertisting)
static uint8 advertData[] =
{
  // 25 byte ibeacon advertising data
  // Preamble: 0x4c000215
  // UUID: 8aeded86-dc91-4917-9ca7-9fd90d10a76b
  // Major: 1 (0x0001)
  // Minor: 1 (0x0001)
  // Measured Power: 4 db (0x04)
  0x1A, // length of this data including the data type byte
  GAP_ADTYPE_MANUFACTURER_SPECIFIC, // manufacturer specific advertisement data type
```

```
  0x4c, 0x00, 0x02, 0x15, //preamble
  0x8A, 0xED, 0xED, 0x86, 0xDC, 0x91, 0x49, 0x17, 0x9C, 0xA7, 0x9F, 0xD9, 0x0D, 0x10, 0xA7, 0x6B, //UUID
  0x00,  0x01, //major
  0x00,  0x02, //minor
  0x04
};


/*********************************************************************
 * LOCAL FUNCTIONS
 */

/*********************************************************************
 * PROFILE CALLBACKS
 */

// GAP Role Callbacks
static gapRolesCBs_t simpleBLEBroadcaster_BroadcasterCBs =
{
  NULL                          // When a valid RSSI is read from controller (not used by application)
};

/*********************************************************************
 * PUBLIC FUNCTIONS
 */

/*********************************************************************
 * @fn      SimpleBLEBroadcaster_Init
 *
 * @brief   Initialization function for the Simple BLE Broadcaster App
 *          Task. This is called during initialization and should contain
 *          any application specific initialization (ie. hardware
 *          initialization/setup, table initialization, power up
 *          notificaiton ... ).
 *
 * @param   task_id - the ID assigned by OSAL.  This ID should be
 *                    used to send messages and set timers.
 *
 * @return  none
 */
void SimpleBLEBroadcaster_Init( uint8 task_id )
{
  simpleBLEBroadcaster_TaskID = task_id;

  // Setup the GAP Broadcaster Role Profile
  {
      // device starts advertising upon initialization
      uint8 initial_advertising_enable = TRUE;

    //uint8 advType = GAP_ADTYPE_ADV_NONCONN_IND;   // use non-connectable advertisements
```

```
    uint8 advType = GAP_ADTYPE_ADV_SCAN_IND; // use scannable unidirected advertisements

    // Set the GAP Role Parameters

    GAPRole_SetParameter( GAPROLE_ADVERT_ENABLED, sizeof( uint8 ), &initial_advertising_enable );
    GAPRole_SetParameter( GAPROLE_SCAN_RSP_DATA, sizeof ( scanRspData ), scanRspData );
    GAPRole_SetParameter( GAPROLE_ADVERT_DATA, sizeof( advertData ), advertData );
    GAPRole_SetParameter( GAPROLE_ADV_EVENT_TYPE, sizeof( uint8 ), &advType );
  }

  // Set advertising interval
  {
    uint16 advInt = DEFAULT_ADVERTISING_INTERVAL;

    GAP_SetParamValue( TGAP_LIM_DISC_ADV_INT_MIN, advInt );
    GAP_SetParamValue( TGAP_LIM_DISC_ADV_INT_MAX, advInt );
    GAP_SetParamValue( TGAP_GEN_DISC_ADV_INT_MIN, advInt );
    GAP_SetParamValue( TGAP_GEN_DISC_ADV_INT_MAX, advInt );
  }

  // Setup a delayed profile startup
  osal_set_event( simpleBLEBroadcaster_TaskID, SBP_START_DEVICE_EVT );

  // Setup custom tx power
  HCI_EXT_SetTxPowerCmd(HCI_EXT_TX_POWER_4_DBM);
}

/*********************************************************************
 * @fn      SimpleBLEBroadcaster_ProcessEvent
 *
 * @brief   Simple BLE Broadcaster Application Task event processor. This
 *          function is called to process all events for the task. Events
 *          include timers, messages and any other user defined events.
 *
 * @param   task_id  - The OSAL assigned task ID.
 * @param   events - events to process.  This is a bit map and can
 *                   contain more than one event.
 *
 * @return  events not processed
 */
uint16 SimpleBLEBroadcaster_ProcessEvent( uint8 task_id, uint16 events )
{

  VOID task_id; // OSAL required parameter that isn't used in this function

  if ( events & SYS_EVENT_MSG )
  {
    uint8 *pMsg;

    if ( (pMsg = osal_msg_receive( simpleBLEBroadcaster_TaskID )) != NULL )
```

```
    {
      // Release the OSAL message
      VOID osal_msg_deallocate( pMsg );
    }

    // return unprocessed events
    return (events ^ SYS_EVENT_MSG);
  }

  if ( events & SBP_START_DEVICE_EVT )
  {
    // Start the Device
    VOID GAPRole_StartDevice( &simpleBLEBroadcaster_BroadcasterCBs );

    return ( events ^ SBP_START_DEVICE_EVT );
  }

  // Discard unknown events
  return 0;
}

/*********************************************************************
*********************************************************************/
```

Firmware Modification Python Script

```
import mmap, math, shutil, sys

#requires python v3
if sys.version_info[0] != 3:
    print("Please use python version 3 to execute this script")
    exit()

#open generic firmware hex file
shutil.copyfile('./Generic_Firmware.hex', './Custom_Firmware.hex')
file = open('Custom_Firmware.hex','r+b')
maped = mmap.mmap(file.fileno(),0)

#beginning of segments of the firmware that will be modified
seg_1 = 1109
seg_2 = 1154

#segment offset till to the UUID, major, minor, tx
uuid_offset_1 = 20
uuid_offset_2 = 8
major_offset = 20
minor_offset = 24
tx_offset = 28
checksum_offset = 40

#length of the UUID, major, minor, tx in the segments
uuid_len_1 = 20
uuid_len_2 = 12
major_len = 4
minor_len = 4
tx_len = 2

#calculate the checksum value and return as a string in hex
def checksumCal(start):
    val = 0
    for i in range(0,20):
        val = val + int(maped[start+2*i:start+2+2*i],16)
        twosCompString = twosComp(val%256)
    return bytes(twosCompString, 'utf-8')

#calculate the two's compliment of an integer value and returns the hex value in string
format
def twosComp(intVal):
    return format(((intVal)^(0xff))+1, 'x')

#displays a section of the firmware
def display_seg(start, char):
    print(maped[start:start+char].decode(), end='')

def newline():
    print()

#checks to make sure the passed variable is Hex (true = hex)
def checkForHex(hexIHope):
    try:
        int(hexIHope, 16)
        return True
    except ValueError:
        return False
```

```
#checks to make sure the passed variable has a specific length (true = has length x)
def checkLength(lengthIHope,length):
    if len(lengthIHope) == length:
        return True
    else:
        return False

#used to enter in a new UUID
def updateUUID():
    newline()

    #first 8 characters of UUID
    while True:
        userInput = input("Enter the first 8 characters of UUID:")
        if checkForHex(userInput) == True:
            if checkLength(userInput, 8) == True:
                break
            else:
                print("Please enter a 8 character hex value. For example: 00000000")
        else:
            print("Please enter a 8 character hex value. For example: 00000000")
    byteInput = bytes(userInput.upper(), 'utf-8')
    maped[seg_1+uuid_offset_1:seg_1+uuid_offset_1+8] = byteInput

    #next 4 characters of UUID
    while True:
        userInput = input("Enter the next 4 characters of UUID:")
        if checkForHex(userInput) == True:
            if checkLength(userInput, 4) == True:
                break
            else:
                print("Please enter a 4 character hex value. For example: 0000")
        else:
            print("Please enter a 4 character hex value. For example: 0000")
    byteInput = bytes(userInput.upper(), 'utf-8')
    maped[seg_1+uuid_offset_1+8:seg_1+uuid_offset_1+12] = byteInput

    #next 4 characters of UUID
    while True:
        userInput = input("Enter the next 4 characters of UUID:")
        if checkForHex(userInput) == True:
            if checkLength(userInput, 4) == True:
                break
            else:
                print("Please enter a 4 character hex value. For example: 0000")
        else:
            print("Please enter a 4 character hex value. For example: 0000")
    byteInput = bytes(userInput.upper(), 'utf-8')
    maped[seg_1+uuid_offset_1+12:seg_1+uuid_offset_1+16] = byteInput

    #next 4 characters of UUID
    while True:
        userInput = input("Enter the next 4 characters of UUID:")
        if checkForHex(userInput) == True:
            if checkLength(userInput, 4) == True:
                break
            else:
                print("Please enter a 4 character hex value. For example: 0000")
        else:
            print("Please enter a 4 character hex value. For example: 0000")
```

```
    byteInput = bytes(userInput.upper(), 'utf-8')
    maped[seg_1+uuid_offset_1+16:seg_1+uuid_offset_1+20] = byteInput

    #Last 12 characters of UUID
    while True:
        userInput = input("Enter the first 12 characters of UUID:")
        if checkForHex(userInput) == True:
            if checkLength(userInput, 12) == True:
                break
            else:
                print("Please enter a 12 character hex value. For example: 000000000000")

        else:
            print("Please enter a 8 character hex value. For example: 000000000000")
    byteInput = bytes(userInput.upper(), 'utf-8')
    maped[seg_2+uuid_offset_2:seg_2+uuid_offset_2+12] = byteInput
    checkUUID()

#used to enter in a new Major
def updateMajor():
    newline()
    while True:
        userInput = input("Enter new Major value (4 characters hex value): ")
        if checkForHex(userInput) == True:
            if checkLength(userInput, 4) == True:
                break
            else:
                print("Please enter a 4 character hex value. For example: 0000")
        else:
            print("Please enter a 4 character hex value. For example: 0000")
    byteInput = bytes(userInput.upper(), 'utf-8')
    maped[seg_2+major_offset:seg_2+major_offset+major_len] = byteInput
    checkMajor()

#used to enter in a new Minor
def updateMinor():
    newline()
    while True:
        userInput = input("Enter new Minor value (4 characters hex value): ")
        if checkForHex(userInput) == True:
            if checkLength(userInput, 4) == True:
                break
            else:
                print("Please enter a 4 character hex value. For example: 0000")
        else:
            print("Please enter a 4 character hex value. For example: 0000")
    byteInput = bytes(userInput.upper(), 'utf-8')
    maped[seg_2+minor_offset:seg_2+minor_offset+minor_len] = byteInput
    checkMinor()

#used to enter in a new Tx value
def updateTx():
    newline()
    while True:
        userInput = input("Enter new Tx value in 2's compliment form (2 characters hex
value): ")
        if checkForHex(userInput) == True:
            if checkLength(userInput, 2) == True:
                break
            else:
                print("Please enter a 2 character hex value. For example: 00")
```

```
        else:
            print("Please enter a 2 character hex value. For example: 00")
    byteInput = bytes(userInput.upper(), 'utf-8')
    maped[seg_2+tx_offset:seg_2+tx_offset+tx_len] = byteInput
    checkTx()

#Displays UUID and asks if it should be changed
def checkUUID():
    print("The current UUID is:")

    #display the current UUID in a easily readable form
    display_seg(seg_1+uuid_offset_1, 8)
    print("-", end='')
    display_seg(seg_1+uuid_offset_1+8, 4)
    print("-", end='')
    display_seg(seg_1+uuid_offset_1+12, 4)
    print("-", end='')
    display_seg(seg_1+uuid_offset_1+16, 4)
    print("-", end='')
    display_seg(seg_2+uuid_offset_2, uuid_len_2)
    newline()
    userInput = input("Would you like to change the UUID?(y/n)")
    if userInput == "Y" or userInput == "y" or userInput == "yes" or userInput == "Yes"
or userInput == "YES":
        updateUUID()

#Displays Major and asks if it should be changed
def checkMajor():
    print("The current Major is:")
    display_seg(seg_2+major_offset, major_len)
    newline()
    userInput = input("Would you like to change the Major?(y/n)")
    if userInput == "Y" or userInput == "y" or userInput == "yes" or userInput == "Yes"
or userInput == "YES":
        updateMajor()

#Displays Minor and asks if it should be changed
def checkMinor():
    print("The current Minor is:")
    display_seg(seg_2+minor_offset, minor_len)
    newline()
    userInput = input("Would you like to change the Minor?(y/n)")
    if userInput == "Y" or userInput == "y" or userInput == "yes" or userInput == "Yes"
or userInput == "YES":
        updateMinor()

#Displays Tx value and asks if it should be changed
def checkTx():
    print("The current Tx Power is:")
    display_seg(seg_2+tx_offset, tx_len)
    newline()
    userInput = input("Would you like to change the Tx Value?(y/n)")
    if userInput == "Y" or userInput == "y" or userInput == "yes" or userInput == "Yes"
or userInput == "YES":
        updateTx()

checkUUID()
checkMajor()
checkMinor()
checkTx()
```

```
#compute and update checksums
maped[seg_1+checksum_offset:seg_1+checksum_offset+2] = checksumCal(seg_1).upper()
maped[seg_2+checksum_offset:seg_2+checksum_offset+2] = checksumCal(seg_2).upper()

maped.close()
file.close()
```