Project Code: MQP SJB 3A17

Hike Safe

A Report
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degrees of Bachelor of Science by

Natalie Correa
And
Melvin Moore

Electrical and Computer Engineering
Major Qualifying Project

Advised by
Professor Stephen J. Bitar

Abstract

As one of the United States favorite pastimes, hiking has long enjoyed popularity among people of all ages and backgrounds. While hiking is a hobby for many, it is considered serious business due to the preparation and planning that is required to ensure it is done safely. Even with preparation and planning, the most experienced hikers have found themselves lost or injured. With the lack of cellular towers in national and state parks, communication with park rangers or other emergency personnel is limited. The goal of this Major Qualifying Project is to design a device that improves the safety of hikers. The device is portable and transmits important data without the use of a designated WIFI or cellular network. This device will allow the hiker to alert the park staff that an emergency has occurred, so the staff can send help to the location of the lost or injured hiker.

Acknowledgements

The team would like to thank Professor Stephen Bitar for his advice, guidance, and support as we worked to create this device.

Executive Summary

Personal safety is a major concern for hikers who frequent national and state parks. An individual hiker visiting one of these parks has a 1 in 19 chance of becoming a victim of a serious crime. This problem is compounded by the lack of cellular coverage in these areas which makes it impossible for hikers to notify emergency personnel when they get into trouble. Although most parks provide information pamphlets, maps and websites to help plan a visit, few have adequate resources to patrol wide areas to ensure safety. This leaves individual hikers vulnerable when they encounter dangerous situations.

This Major Qualifying Project proposes an emergency notification system capable of notifying park personnel when a hiker gets in trouble. This system can be broken down into three parts – a portable transmitter or tag to be carried by individual hikers, a number of emergency beacons located throughout the park and a central base station where emergency personnel receive notification when a hiker activates the tag. This system is shown in Figure 1.

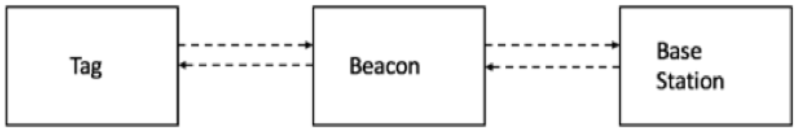


Figure 1: System Block Diagram

The tag is designed to be small, portable and lightweight. The tags are battery powered and transmit real time GPS coordinates to the nearest beacon when activated. In the proposed design, they are cable of transmitting continuously for 30 hours, if needed. The block diagram for this portion of the system is shown in Figure 2.

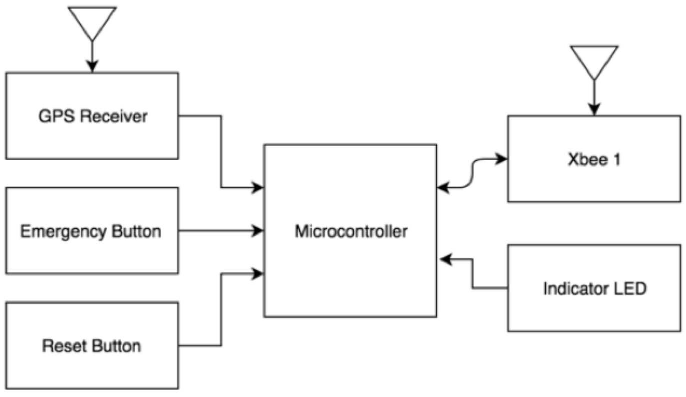


Figure 2: Tag Block Diagram

Beacons are rugged and will be installed every 2 miles in order to form a mesh network. They are capable of relaying information to other beacons as well as to the base station. The figure below shows the Beacons that would be setup along the hiking trails as well as the large circles indicating their range where the tags can be detected.

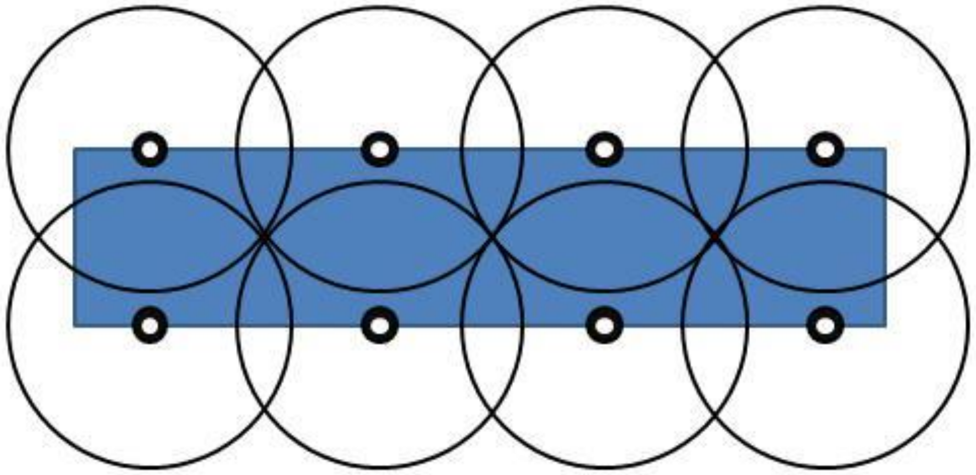


Figure 3: Beacons Spaced Along the Trail

Beacons are battery powered and rely on solar panels for recharging so that no hard wiring is necessary. The block diagram for a typical beacon is shown in Figure 4.

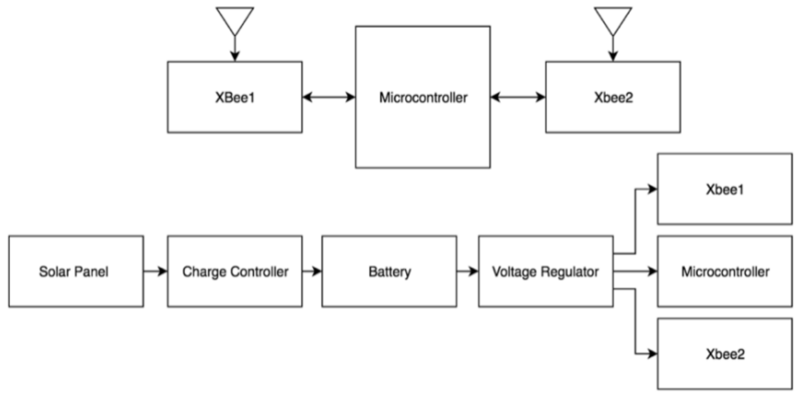


Figure 4: Beacon Block Diagram

At the base station, the received GPS coordinates are used to indicate the location of the hiker on a map of the park so the emergency personnel can respond. The GPS coordinates will continue to appear on a computer screen until a reset signal from the tag is transmitted to the base station as shown in Figure 5.

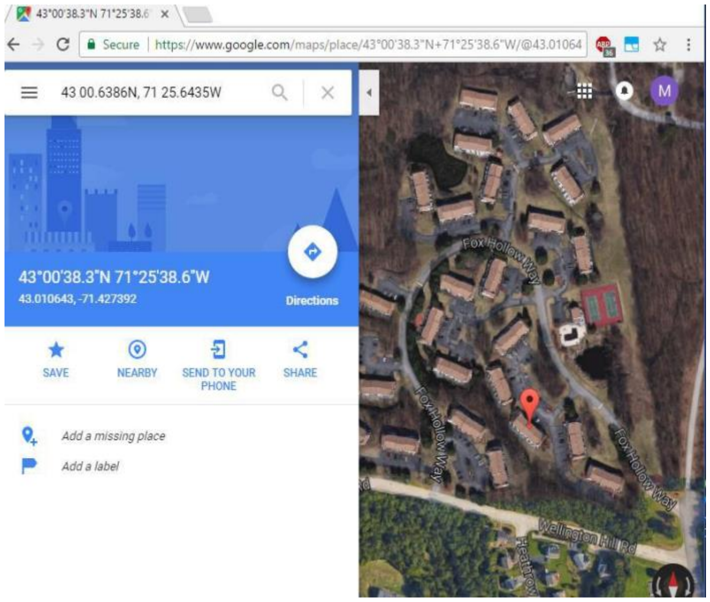


Figure 5: GPS Coordinates displayed on Base Station

In this way, the tag, the beacons and the base station, work together to create a system that is able to better protect hikers in emergency situations.

Table of Contents

List of Figures

List of Tables

Introduction

The National Park system encompasses 417 national parks in the United States. They span across more than 84 million acres in each state, extending into territories including parks in Puerto Rico, the Virgin Islands, American Samoa and Guam. Threats to the visitors of these national parks range from thefts to murders and in recent years the crime rates in these areas have remained high. In Table 1.1, it can be seen that the number of crimes occurring in National Parks have stayed stagnant. The following table, derived from the Federal Agencies website portrays two years of crimes from 2014 to 2016.

TABLE 1.0: NATIONAL PARK SERVICE CRIMES, 2014 TO 2016

Year	Violent Crime	Murder and Nonnegligent Manslaughter	Rape	Robbery	Aggravated Assault	Property Crime	Burglary	Larceny Theft	Motor Vehicle Theft	Arson
2014	369	16	62	83	199	95	645	158	92	69
2015	200	9	72	84	85	87	543	541	92	41

The lack of preventative actions being taken were the cause of the stagnant crime rates. Through research it was found that only preemptive actions are taken such as websites or pamphlets that advise individuals on how to stay safe during their visit. Once the hiker is in danger there are limited options to contact emergency personal. One of the only ways to contact emergency personal is a cell phone but with limited cell service this proves to be a reliable option, leaving the hiker vulnerable.

Due to the lack of safety measures taken for hikers, we chose to fulfill our senior project requirement of demonstrating abilities in electrical and computer engineering by designing a product that would help create a safe environment for the visitors of the State and National Parks. The system designed is a solution that provides visitors of State and National Parks with simple remote access to park service emergency personnel. When the device is activated, the device sends GPS location data via a modular relay system to Park Service Headquarters. The information would not only bypass the need to use a personal cell phone to dial the police but it would also provide useful data to the dispatcher, therefore helping first responders to get to the scene of a potential crime in a timely manner.

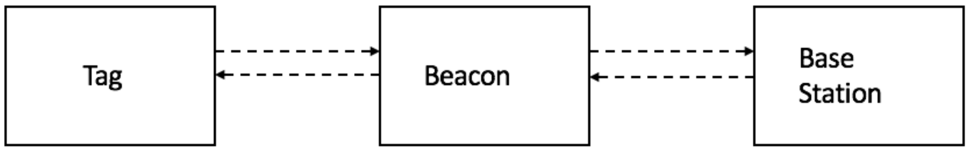


Figure 1.1: System block diagram

The system can be broken down into the following parts: a Tag, a Beacon and the Base Station. The device would need to inform park service emergency personnel of the user's location at the press of the button. Since the user simply borrows the device for a certain time period there is no maintenance required on the user side. The network of Beacons along the Park trails would receive the emergency signal from the wireless module of the Tag and transmit it along the shortest route possible to the Park Service Headquarters otherwise known as the Base station. Finally, in order to keep track of the Tags given to the visitors upon the beginning and ending of their visit a system would need to be developed that would be capable of tracking the rented devices and maintaining them upon return.

Since a National Park such as Yellowstone spans roughly 2.2 million acres with more than 900 miles of hiking trails it was more realistic to test our project on a smaller scale. Massachusetts is home to over 145 State Parks spanning over 2,000 miles of trails which was slightly more realistic to test the project which can be seen depicted in the figure below.

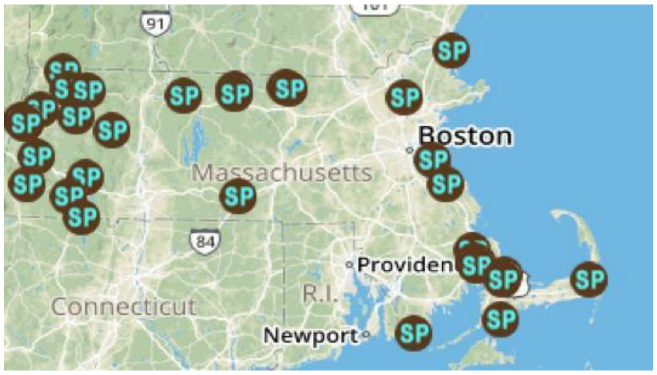


Figure 1.2: State Parks located in Massachusetts

It can be seen from the chart depicted that crimes are more likely to occur in a Forest Park. For every 100,000 people, there are 14.84 daily crimes that occur in Forest Parks. This means that in a Forest Park you have a 1 in 19 chance of becoming a victim of any crime. That means that you are more likely to become a victim of any crime in a Forest Park than on the streets.

Table 1.1: Reported Annual Crime in Forest Park

Statistic	Forest Park /100k people	Massachusetts /100k people	National /100k people
Total Crime	5,416	2,082	2,860

2. Background

This section provides a foundation for understanding the design process for a device that informs park service emergency personnel of the user's location at the press of a button. This section also provides information regarding hiking popularity, the dangers of hiking and what types of preventative actions are available for hikers. It will also briefly explain how many people are affected yearly by hiking related crimes. The chapter will also explore existing devices that are currently on the market. Finally, this chapter will provide information on options considered for the creation of the different parts of the device.

2.1 Hiking Popularity

Hiking is the most accessible sport in the world. It requires no special equipment or training and offers health benefits as well as therapeutic benefits that help provide a stress free lifestyle for the working class. For the working class an escape from their busy lifestyles and technology is needed. With hiking providing plenty of perks such as nice views, fresh air, smell of nature, improvements with blood pressure, balance and weight it is no wonder that people are choosing to engage in this activity. As shown in Table 2.0, the number of people participating in hiking has increased 7.64 million from 2011 to 2016.

Table 2.0: Number participants in hiking in the United States from 2011 to 2016

Year	Number of participants in millions in the United States
2011	34.49
2012	34.55
2013	34.38
2014	36.22
2015	37.23
2016	42.13

This increase of hiking has some detrimental effects. With the increase of hikers, the increase of the potentials hazards has followed.

2.2 Hiking Hazards

Whether the hiker is taking a casual stroll in a State Park or hiking the Maze in Utah there is a potential risk of getting injured or ill. The four main hazards that could put a hiker at risk are topographical, climatic, human, and mountain environment. While there are rules such as not hiking alone, staying near the designated trail and hiking with a well-prepared pack, humans seldom follow these and find themselves in danger. The top five causes of death in National Parks from 2007-2013 were drowning, falling or slipping, poisoning, and wildlife or animals. This is why from 2007-2013 in all 59 National Parks there were 1,025 fatalities. On average 160 visitors per year die while visiting national parks. While this sounds like a miniscule number compared to the number of visitors the fact is that hikers are still at risk.

2.3 Hiking Injury Preventions

For any hiker, the best and easiest way to stay safe is to prepare prior to visiting a national or state park. There are many websites such as hikesafe.com that offers tips on how to stay safe and prepare. Some of the suggestions that many of these websites inform hikers to do are to remain on the marked trail, to inform other of your plans prior to going, to hike in groups, to turn back in case of bad weather, and to know what to do in case of an emergency. These websites often inform potential hikers to know how to rescue themselves in case of an emergency due to no guarantee of being rescued. With lack of cell service, preparation before hiking is the only real way to stay safe and prevent injury.

2.4 Research of Available Products

In the case of an emergency, the lack of experience for some hikers and the inability to contact someone can be dangerous. In order to address this gap, the group conducted research on devices that while does not keep hikers safe, have a similar concept and were designed for keep school campuses safe.

The first system is based around existing emergency tower systems, similar to the towers located on the WPI campus.

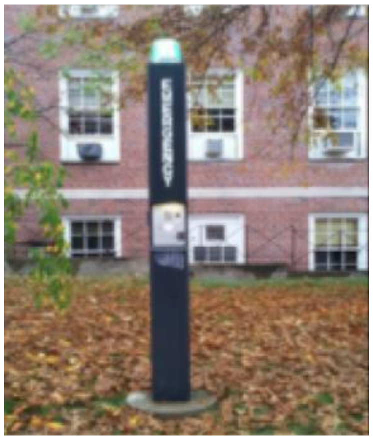


Figure 2.0: An Emergency Tower on WPI Campus

The emergency tower is a system developed by Code Blue, a company working on the development of the "Circle of Safety System," which provides remote access to the Code Blue emergency towers on some campuses [1]. This system includes a "panic button" pendant on the user which sends a signal to the nearest emergency tower from up to 200 feet away when pressed. When the emergency signal is received, the campus police can locate and respond to the call and also provides the police with identification of the user and links to his or her personal file. This system currently has 300 students using the device at the Butler University in Indianapolis for testing purposes. Code Blue targets universities, corporate and medical campuses as its market. Since the system adds cost to the university if adopted, a solution is to let individuals subscribe to the system at a rate of $75 for the pendant and $50 for the annual activation fee. The downside of this device is the user has to maintain the device themselves. For the parks, the visitors are only there for a limited timeframe and the park may not be visited frequently enough in order to make this affordable for the users and the park.

Another product on the market that is used for college campuses is called RAVENalert. This device sends emergency notifications to students with a brief account of any emergency incidents [2]. RAVENalert devices are the size of a keychain and uses wireless technology to send alerts to students via voice and vibration. This device serves its purpose of notifying the students in case of an emergency but lacks the ability of communicate between campus security and the students. In the case of park visitors, they would be notified of wildfires, or possible dangers in the park but would be unable to communicate if in eminent danger.

Figure 2.1: RAVENalert keychain

The last device we looked at is called Life Alert. Life Alert is primarily used to help elderly citizens alert emergency personnel if in danger. This device operates by using two separately encased and remotely positioned components. The first component comprises of a portable wearable device that contains an emergency button, and a radio transmitter. When the emergency button is activated by a user, a radio signal is sent to the second component that comprises a base unit. The base unit is in communication with the land telephone line of the building the device is used. The portable wearable unit typically comprises a small pendant-sized unit that is coupled to a lanyard or rope, and worn like a pendant around the neck of the user. The base device is often the size of a multi-line telephone base set, and is placed at a position in the house close to a telephone jack, so that it may connect through the phone jack into the land line circuitry of the house. To operate the unit, a user presses a button on the pendant/portable unit. The pendant then sends a signal to the base unit. The base unit has an automatic dialing feature and communicates a signal through the landline of the house to a help desk maintained by a company, such as Life

Figure 2.2: Life Alert

Alert or American Alarms. The normal protocol for dealing with such a call is that the call is received by the help desk operator, who then tries to communicate verbally with the user. This verbal communication is usually attempted through a “speaker phone” feature of the base unit. If the remote caregiver (here a help desk operator) can communicate with the user and establish that nothing is wrong with the user, or that a false signal has been sent, the caregiver can terminate the telephone call knowing that the user is in no emergency. On the other hand, if the user is capable of verbally communicating with the help desk so that the caregiver can determine the nature of the emergency, the help desk operator might be able to obtain enough information to contact the appropriate emergency responder, who may be a person such as the next of kin, a closely located friend, an ambulance, fireman or a police agency. In the case of the parks, a device like this would be deemed useless due to the lack of landlines around areas. Even though this is the case, the concept of wearing the device and alerting emergency personal was similar to our design concept for our device.

2.5 System and component research

In order to design a system that met design requirements and was efficient, components were researched that when used together would create a working system.

2.5.1 Wireless Network

One of the design requirements for our product was to establish a wireless network between the Tag, Beacon and Base station. For this project, three different wireless networks were researched.

The first wireless network considered was Wifi. One of the benefits of Wifi, is it offers the use of a secure network with a long range of transmission and supports faster transmission speeds allowing more info to be transmitted at a time (250Mbs+). A disadvantage of using Wifi is that it consumes a lot of power approximately 80mW to send data at a rate of 75 bytes per second [3] at close range. With our device being used in an area where electricity is non-accessible, this would cause a problem. Another disadvantage of Wifi is for it be accessible in the parks, cell towers would need to be accessible, adding high costs as well as interfering with normal wildlife activity. The easy accessibility, while would provide incentives for hikers to use would create noise pollution and antagonize those trying to rid themselves of cellphone use.

With Wifi eliminated as an option, the next wireless network considered was Bluetooth. This technology, is better suited for battery powered applications as it uses very little power, approximately 2mW to send data at a rate of 75 bytes per second [3] at close range. It is an inexpensive solution using less power than other wireless technologies [4] such as Wifi. Although it is not as secure or as stable as Wifi it is a good substitute as it allows peer to peer communication. A major disadvantage of Bluetooth eliminating it as an option is its max usable range of about 100 meters.

With Wifi and Bluetooth eliminated as an option, a better suited technology was needed. Zigbee is not as fast as Wi-Fi, since the data rate is 250KB/s in comparison to the data rates for Wi-Fi which is about 54Mb/s. The data transferred between the Tag and the Beacon is small enough that the data speed is more than sufficient. The Zigbee module can be configured as a mesh network. A mesh network uses a different approach to sending data throughout the network. In the mesh network, there is one coordinator that maintains the network by managing the associate devices and routers. In this project, the mesh network can be used. The one coordinator network would be the Emergency Personal headquarters while the routers will be used in the Beacons and the Tag would be the endpoints. An example of a mesh network can be seen in Figure 2.4.

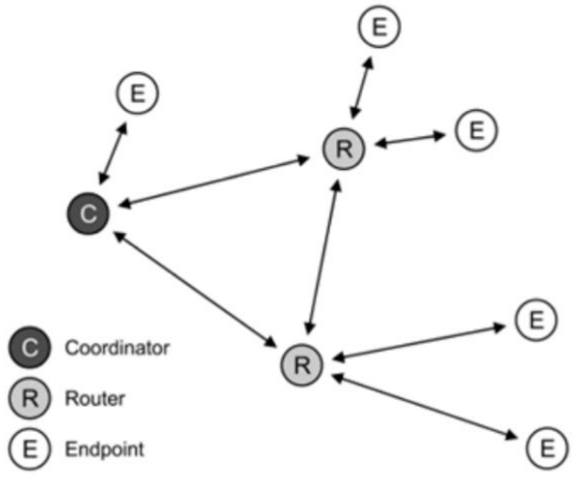


Figure 2.3: Mesh Network

With limited options the Xbee's were the best suited choice for our design. The two pairs of XBees that were chosen were the Series 1 (100m) and Series 1 Pro (1Mi). The Xbees chosen use little power, 1mW and 60mW during transmission respectively.

2.5.2 GPS

GPS provides a convenient and inexpensive solution to track location. Different types of GPS receivers were researched that would be compatible with the project design. Receivers get information from the GPS satellites orbiting around the earth and use triangulation to calculate the user's location. The GPS satellite provides longitude, latitude and longitude data. There are two many types of GPS receivers on the market including standalone systems and GPS modules. For this project, GPS modules were chosen since they can be embedded into a device. The output of a GPS module is a string of ASCII characters which can be converted into longitude, latitude and altitude data by using the NMEA protocol, the standard protocol for GPS systems. The GPS modules are small in size, have an update rate of about 5 to 10Hz, have an average power requirement of 30mA at 3.3V and an accuracy of up to 3m depending the number of channels of the GPS module. There is also a wide availability of software to convert GPS data into maps such as those used for Google Maps or Google Earth [13]. For example. A software called GPS Visualize can convert GPS data into a readable format that gives longitude and latitude information. The GPS data files can then be imported into this program and mapped into Google Maps. Due to this, the GPS module will be included in the tag. This is because the tag is required to provide accurate location data, have reasonable power consumption and update rate and software to easily recover the GPS data into the server.

2.5.3 Power Source

Since the Zigbee and GPS modules both require power to operate, the Tag must have a way of charging. Wireless power is a relatively new up and coming technology that has the potential to replaced existing wired technology. A reason to use wireless power is the elimination of the need for cables and ports allowing a sleeker, durable and portable design. Wireless power can be found in a range of different technology such toothbrushes, and smartphones.

There are several ways that allow a device to charge wirelessly, the most explored and popular of which is the Qi standard developed by the Wireless Power Consortium (WPC) [14]. This standard uses two magnetic coils to transfer power from a charging pad to a device [15]. While a charging pad is a more efficient method of wireless power, its range is lacking and can only send over distances of 5mm or less. When this was standard was updated in 2012, the transmitter was then able to send up to 5 watts of power 40mm [16]. The resonant frequency of the WPC can be selected between the ranges of 100KHz up to 205kHz. This technology tends to used either a half-bridge or a full bridge inverter in the transmitters circuit for the DC to AC conversion [17].

The Alliance for Wireless Power (A4WP) is a standardization organization that aims for "spatial freedom for charging of electrical devices." This standard organization allows several devices to be charge simultaneously and at greater distances [18]. This standard uses magnetic resonance [19]. This type of charging only transfers power at a particular resonant frequency. A power is used to induce current as well. To achieve power transfer in WPC and PMA, the transmitter will periodically search for the receiver. However, in A4WP this same technique cannot be utilized and amplitude, power, or current modulation might be used instead if circumstances were correct. Generally, Bluetooth or Zigbee are better options for A4WP communications [17].

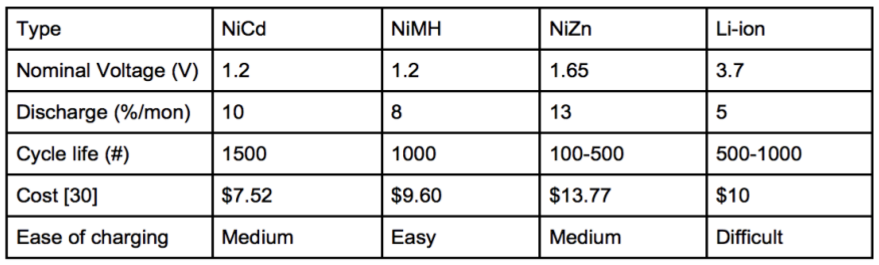
Inductive charging uses electromagnetic fields to transfer energy between the transmitter and receiver. Oscillating current at the primary coil produces varying electromagnetic fields which induces current in the secondary coil. The secondary coil is connected to the load to transfer energy [20]. In order to maximize the efficiency of the energy transfer using the coils, the resonant inductive coupling should be used [21]. At the resonant frequency, the system oscillates with the highest amplitude. In considering the efficiency of wireless charging, one should be aware that the coupling coefficient of the air is very small resulting in low efficiency. The coupling coefficient is the fraction of the electromagnetic flux transferred from one coil to another [22]. With normal inductive charging, the efficiency of the system will be low because of the small coupling coefficient of air. Therefore, for wireless charging for the tag resonant inductive charging would be the best technique. This would be ideal for this project since the tags would need to be charge using wireless charging.

The most important requirement for this project was, in particular the beacon, uses renewable energy. In order for the system to be expandable throughout the National and State parks, there cannot be heavy reliance of hardline power or data line. Hardline power is also not easily accessible throughout the State and National Parks make this not ideal for the design. Since the system should relatively small, solar was prioritized over other alternative renewable energy such as wind power. Wind Power is intermittent and unreliable causing it not to be suitable for a safety system.

Figure 2.4: Solar panel charging battery

Since the hours of sunlight are largely predictable, using solar power in this project would be easy. Solar power combined with an appropriately size batter bank can be relied on for day and night operation. It was decided that the best balance of desirable traits would be NiMH, which are the least expensive and most readily available, and they can easily be trickle charged. Additionally, their capacity is excellent, and they perform well under high drain conditions. This should allow them to last long enough for users to have overnight between charges and easily switch into emergency mode without depleting the battery quickly.

Table 2.1: Important Battery Characteristics



Type	NiCd	NiMH	NiZn	Li-ion
Nominal Voltage (V)	1.2	1.2	1.65	3.7
Discharge (%/mon)	10	8	13	5
Cycle life (#)	1500	1000	100-500	500-1000
Cost [30]	$7.52	$9.60	$13.77	$10
Ease of charging	Medium	Easy	Medium	Difficult

3. Design

This section provides the design process that was taken for our device that informs park service emergency personnel of the user's location at the press of a button. This section also provides information on how the modules were selected and the power estimate for each module.

3.1 Concept of operation

The device was designed to notify emergency personnel of a hiker in potential danger. In order for the device to do this in an efficient manner the device was designed to operate in two modes. When there are no hikers in danger the device will be in standby mode. This will allow all devices in the system to save power. In standby mode the Beacon will continue to look for incoming transmission from the Tag, but is otherwise inactive. When a Tag starts to transmit the emergency signal all devices will enter emergency mode. In emergency mode the Xbee and the GPS are activated in the Tag and will begin receiving position data from the satellites. This data will then be sent to the closest Beacon. Once the communication with a Beacon is established the GPS information is sent to the Base Station.

3.2 Design outline

This system can be broken down into the following parts: the tag, the beacon and the base station. Each device had its own requirements that helped create a simple system that would signal emergency personnel to the location where the portable device was activated.

The Tag needed to locate the user and send the information to the closest beacon when an emergency occurred. Since the hiker would be undergoing physical activity with this device it was imperative that the device could be portable. With these specifications the device had to be small, portable, and lightweight. If any of these requirements were not fulfilled the device would be undesirable for the hiker.

The device was split into three parts: input hardware, microcontroller/software architecture, and output hardware. The input hardware consisted of a way for the user to interface with the microcontroller and it also had a way for the location of the user to be identified. The microcontroller/software architecture portion utilized software to detect an input and an output for the location of the user. The output hardware obtained the location from the microcontroller and transmitted this information to the beacon. As illustrated in the block diagram shown in Figure 3.0, the Tag begins at the user interface level which includes the emergency. When the emergency button is pressed, the emergency state is activated in the microcontroller which then communicates with the GPS receiver and Xbee. The Tag when in standby mode will use minimal power to not drain the charge from the rechargeable batteries.

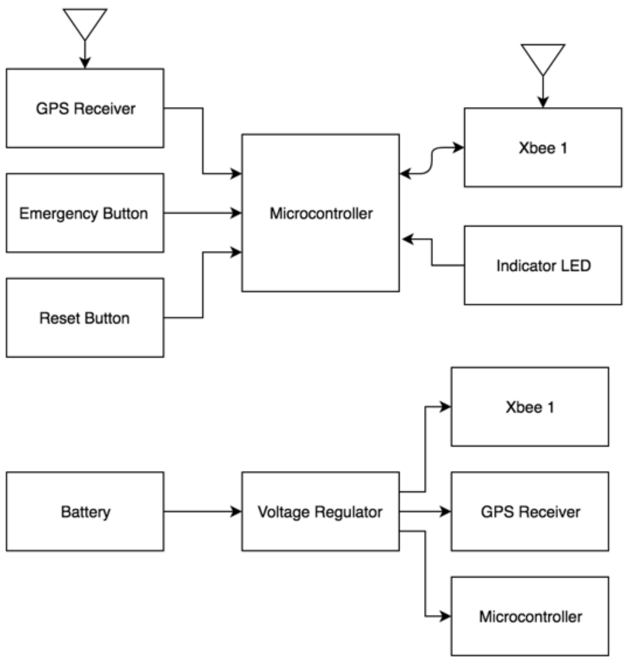


Figure 3.0: Block Diagram for Tag

The software architecture for the Tag can be seen in Figure 3.1. The Tag is in low power mode continuously until the user button is pressed. When the hiker presses the emergency button the device will switch to high power mode turning on both the GPS and the XBee1signal. The GPS will transmit the user’s location from the satellites directly to the beacon via the Xbee1. When the beacon receives the GPS information it will send a signal back to the Tag. The Tags indicator LED will turn on notifying the hiker that the beacon has received the signal. The GPS will continue to check if the user’s location has changed. If the location of the user changes the GPS packet will be retransmitted. The Tag will return to low power mode once the reset button has been pressed signaling the hiker has been reached by the emergency personnel.

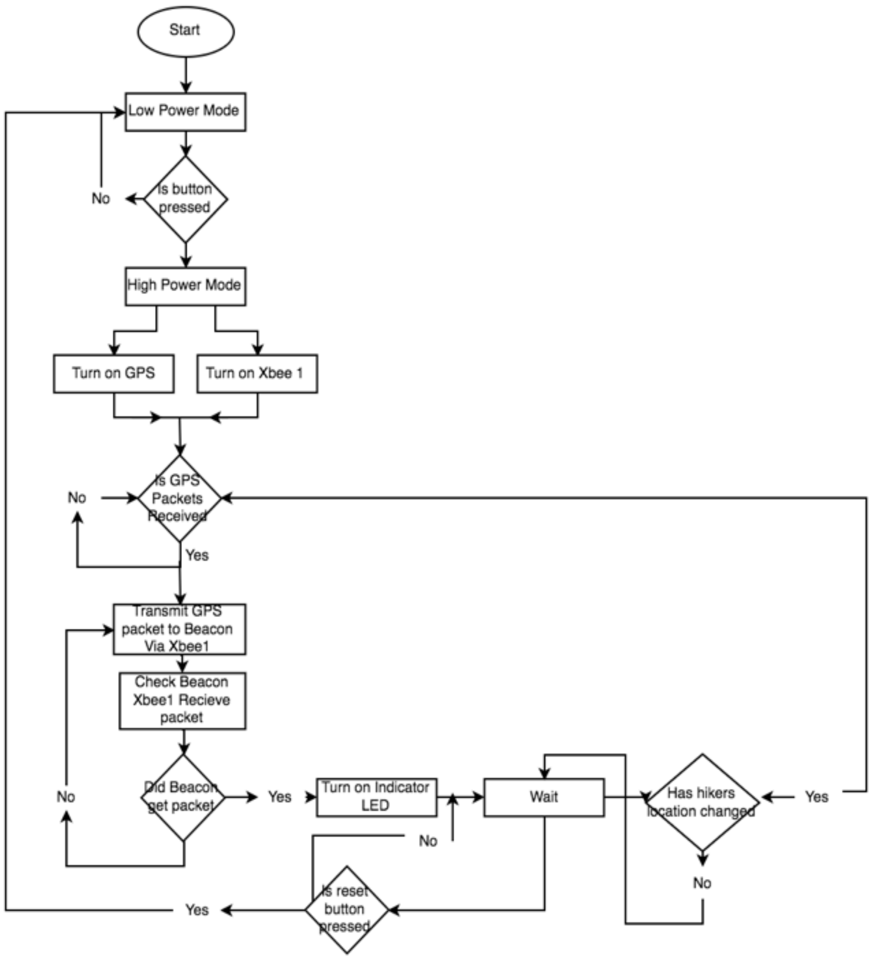


Figure 3.1: Software Flowchart for the Tag

The Beacon is the device in the system that relays the emergency signal, and GPS coordinates to the park safety headquarters. The first objective of the device is that it needed to remain on indefinitely. The second objective of the device is that it was required to receive the location being transmitted from the tag. The third objective of the device was it was required to transmit the location to the base station. With the objectives in mind the beacons would be installed along the trail within a 2mile radius and would be outside for the life of the system. The Beacon nearest to an activated Tag will receive the data and relay it to the nearest Beacon, taking the shortest path to the Bay Station. The devices are weatherproof, powered by battery and reusable energy. The system is capable to receive and transmit a signal at any given moment. If one of these requirements were not considered or met during the design of the beacon, the beacon would fail to transmit/receive a signal and the system would no longer be cohesive resolution.

Like the tag the beacon was split into three parts in order to design the system: input hardware, microcontroller/software architecture and output hardware. The input hardware consisted of a way for the beacon to receive the signal that encompassed the hikers GPS coordinates. The received signal was then communicated to the microcontroller. The microcontroller then communicated this signal to the output hardware that would then transmit the signal to the base station.

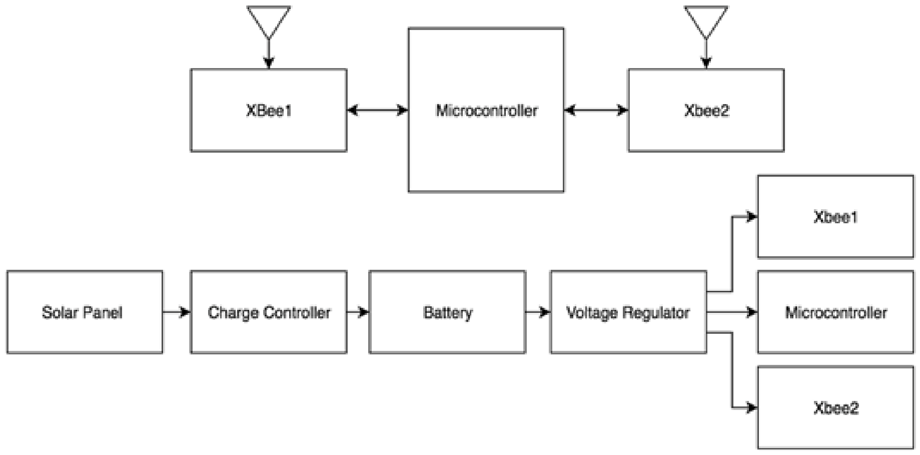


Figure 3.2: Block Diagram for Beacon

The software architecture for the beacon can be seen in the Figure 3.3. The beacon will stay in low power mode looking for the signal from the tag where the emergency button has been pressed. When this has occurred the Xbee1 will receive the GPS packet from the tag. The beacon will transmit the GPS information using the XBee 2 to the base station either directly or by transmitting the information through the beacons until the distance is small enough to transmit the packet to the base station. The beacon will then check if the tag reset button was pressed. If the reset button was not pressed the beacon will return to the waiting stage where it will check if another GPS packet has been received with the individual’s location. When the reset button has been pressed the beacon will return to low power mode for the next transmission.

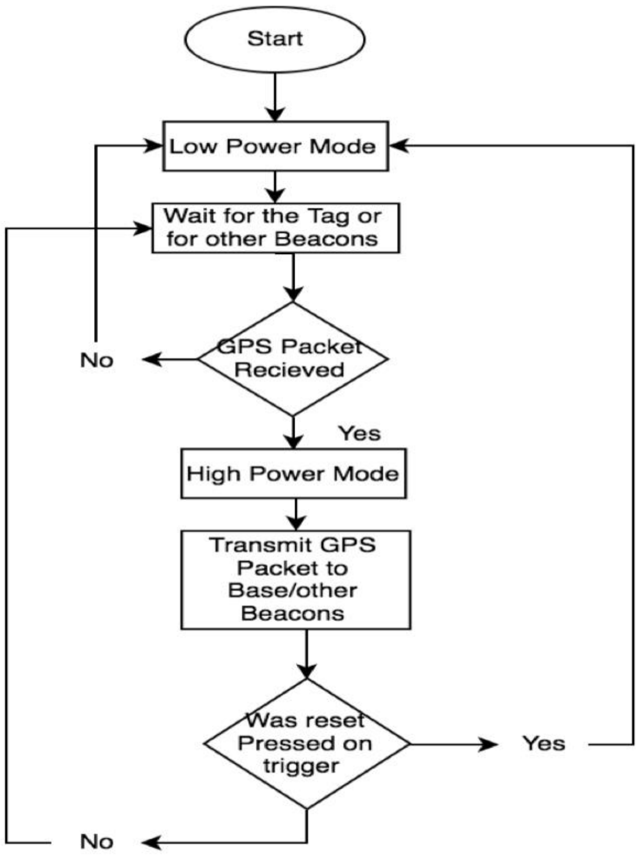

Start
Low Power Mode
Wait for the Tag or for other Beacons
GPS Packet Recieved
No
Yes
High Power Mode
Transmit GPS Packet to Base/other Beacons
Was reset Pressed on trigger
Yes
No

Figure 3.3: Software Flowchart for the Beacon

The Base Station is the end point for the transmission of the emergency signal and GPS coordinates. The first objective of the device is that it needed to receive the location/signal transmitted. The second objective required was to display the signal on the emergency personnel's computer. This device was simple to design. The input hardware consisted of the Xbee2 for the base station to receive the location of the hiker. The received signal was then communicated to the microcontroller. The microcontroller was connected to the computer where the location/signal was transmitted and displayed. The base station will continue to display the GPS location of the user until the reset button of the trigger is pressed.

These three devices will work together to create a system that was designed to address the problem while providing all customers with a way to protect their hikers and resolve needs not met by the competition. In this current state the project provides the base features of the desired system. It provides the GPS coordinates of the portable device to be sent along a small network to a simple user interface but lacks the robustness needed for a consumer product.

3.3 Module Selection

In order for the devices to work together to create a system that would signal emergency personnel to the location where the portable device was activated, modules needed to be selected. This section explains the research and thought process that was used in order to selected modules that would work together to create an efficient system.

3.3.1 Wireless Communication

Due to the research of a Zigbee network, a mesh network was selected to create the relay network for this application. The series that allows us to create a mesh network is the Xbee ZB series. We decided to choose the Series 1 and the Series 1 Pro. The Xbees chosen use little power, 1mW and 60mW during transmission respectively. The main differences between these two models is their range. The Series 1 has a range of 300ft while the Series 1 Pro has a range of 1Mi. The Xbee Series 1 was chosen as the communication between the Trigger and the Beacon which is why the Xbee with less range was chosen. The Xbee Series 1 Pro was chosen as the communication between the Beacon and the Base Station which is why the Xbee with more range was chosen.

The next step would be to determine which antenna package would be used. For the Xbee ZB modules, there are four different antenna packages. The four antenna packages that can be used with the Xbee ZB module are the wire antenna, the U.FL antenna, the PCB antenna and the RPSMA antenna. For the Tag and the Beacon, the wire antenna would be more than enough for their transmissions to each other.

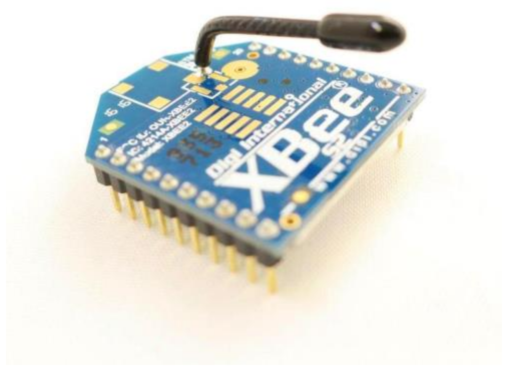


Figure 3.4: Xbee ZB Module

3.3.2 GPS

We considered price, size, update rate, power requirment, number of channels, antenna types, accuracy and compatibility for choosing a GPS module. The number of channels for the GPS was important since the number of channels affected the accuracy of the position. The type of interface was also important since the GPS needed to be compatible with the prototype. Due to this factor we specifically looked for GPS modules with UART and SPI interfaces in order to be most compatible with the microcontroller chosen for this project. With this in mind, there are many GPS modules that met the requirements of our system. The two that we had considered were the AdaFruit FeatherWing Ultimate GPS and the AdaFruit FeatherWing Ultimate GPS Breakout.

The AdaFruit FeatherWing Ultimate GPS, with 66 channels the GPS has high accuracy in determining where it is within a 3m radius. It updates its position from 1 to 10 times a second. Further looking into the GPS' datasheet it can be used with the MSP432 as it is operated at 3.3V. Many GPS' communicate differently using I2C, SPI, UART etc. For our design the GPS needed to use a UART connection.

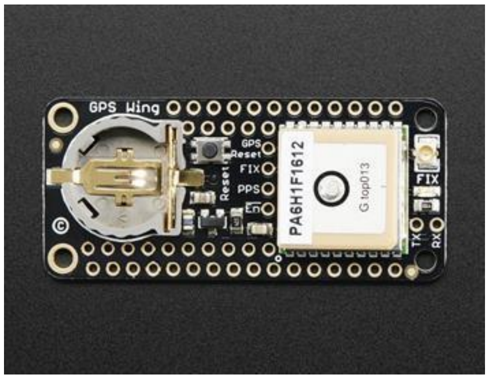


Figure 3.5: AdaFruit Ultimate GPS

The AdaFruit FeatherWing Ultimate GPS Breakout is essentially the same as the AdaFruit FeatherWing Ultimate GPS but is smaller in size and only contains the essentials needed to run the device. This would have been a better choice (for simplicity) to use as we only need to use four pins to use the GPS with the MSP432 but due to accessibility the AdaFruit Ultimate GPS was chosen.

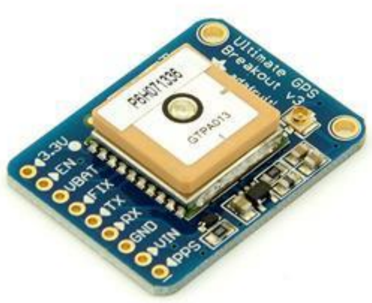


Figure 3.6: AdaFruit Ultimate GPS Breakout

3.3.3 Microcontroller

Given the concept of the Beacon and the Tag, both devices are well suited as embedded computing applications. A microcontroller can be serve the needs of these devices, given their simplicity of implementation and low level hardware. In particular, when the data is being sent at high speeds through a relay network, the microcontroller will be beneficial to help route the information. In order to be able to transmit the data in our design a microcontroller will need to be used. We looked into the Arduino Uno, Arduino Uno Mini, the MSP430, and the MSP432.

The Arduino Uno is often used for different types of applications because of its robust and easy to use design. With this in mind, the Arduino would not be ideal for our application due to its over consumption of power (45mA) causing the battery to drain fairly quickly. The Arduino can also only support one universal asynchronous receiver transmitter device (UART). This would not be beneficial for our device due to the use of more than one UART, the GPS and the Xbee.

Figure 3.7: Arduino Uno

The Arduino Mini was also considered. The Arduino Mini uses the same processor as the Arduino Uno but uses far less power (4.74mA) and space making it ideal for battery powered applications. Unfortunately, with its smaller size additional equipment is needed to program the controller due to no USB port. Like the Arduino Uno, the Mini lacks the capability to interface with more than one UART device making it useless for our intended design.

Figure 3.8: Arduino Pro Mini

The MSP430 is a low power microcontroller from TI. This lower power microcontroller is used frequently in WPI Electrical Engineering courses. Due to its frequent use this device was considered. With a design requirement of a minimum of two UART ports the MSP430 is highly suited having 4 ports that allows 4 different peripherals to be controlled. The MSP430 also meets the design requirement of lower power consumption with it essentially using no power as it consumes 404 micro amps per MHz. This would allow the device to be active a lot longer.

Figure 3.9: MSP430

The MSP432 is a newer more advanced version of the MSP432 that has a 32bit processor. It has the functions of the MSP432 but consumes less power about 80 micro amps per MHz. This controller was chosen due to its necessary functions and its accessibility to us.

Figure 3.10: MSP432

Below is a table of the operating voltages, the low power mode and the UART for the microcontrollers compared for this design.

Table 3.0: Microcontroller Pros and Cons

Microcontroller	Operating Voltage	Low Power Mode	UART
Arduino Uno	5V	Yes	1
Arduino Pro Mini	3.3V	Yes	1

MSP430	3.3V	Yes	4
MSP432	3.3V	Yes	4

3.3.4 Power Estimate

A breakdown of the voltage and current draw of each chosen module to be used in the prototype can be seen in Table 3.1. The modules will be in low power mode for majority of the time.

Table 3.1: Power Consumption Consideration

Device	Voltage	Current Draw
MSP432	3V	0.625mA
XBee1	3.3V	45mA
XBee2	3.3V	305mA
GPS	3.3V	25mA
LED	1.8V	20mA

An in detail breakdown of the three main components, the Tag, Beacon and Base Station is given below.

The different UART ports that could be used for the Tag was located in MSP432 datasheet. The MSP432 Launchpad has four UART ports. The Tag only needed two so it was decided to use port 2's receive/transmit pins P2.2/P2.3 respectively as well as port 3's receive/transmit pins P3.2/P3.3 respectively. The GPS and XBee peripherals would then be connected to these pins to exchange information. For simplicity the peripheral devices (GPS and XBee) only required 3V to operate and had voltage regulators to help the distribution of the power. We were able to power the GPS and XBee with the MSP432 controller's voltage output pins. This would allow the MSP to be powered only with batteries while everything else would be powered off of the controller.

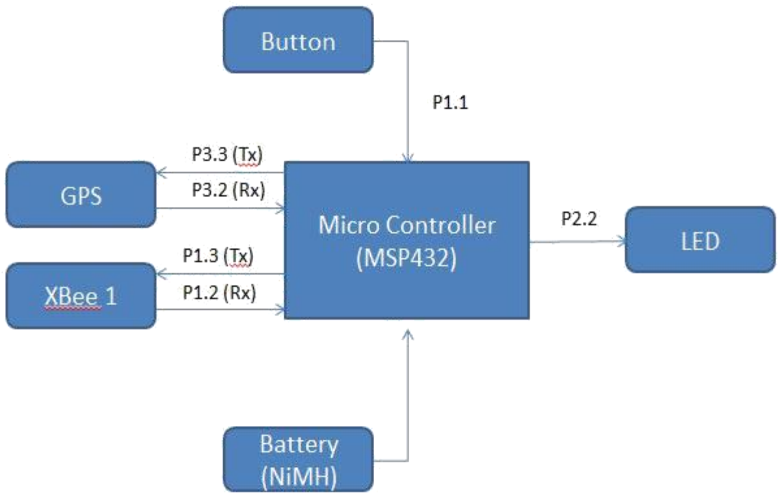


Figure 3.11: Tag UART Ports

For identifying Tags battery requirements, we assumed that all devices excluding the controller would be off until activated by the hiker. Until activated the MSP432 would be in low power mode waiting for the Tag's button input. With the MSP in low power mode it uses minimal current (a few micro amps). When the hiker does press the input button the controller (0.625mA) will enter active mode. The GPS (25mA), XBee (45mA) and LED (20mA, just to show info is being sent) will initialize and current will start to be drawn. The total max current drawn per hour will be 90.625mA.

As a security blanket, the Tag will be able to last at least a day at max current draw. Multiplying 90.625 by 24 hours means we need a power source with a capacity of 2,175mAh. We then found high capacity batteries on Amazon, they were AA batteries at 1.2V and 2800mAh each. This means we could run the device at full transmitting power for 30.89 hours on three batteries in series (the controller runs on 3V) which should be more than enough for a park ranger to receive the emergency notification and send help to the Tag's location.

For power circuit the batteries voltage of 3.6V needed to be stepped down to 3V which was quickly accomplished using the LM317 (represented as the LT317A regulator if Figure 5.11) adjustable voltage regulator. This regulator can output a voltage between 1.25V and 37V. In order to output 3V we needed to use the right resistor ratio which is found by rearranging the equation Vout = 1.25 * (1 + R2/R1) into R2 = R1 * (Vout/1.25 – 1). We chose R1 to be 250 ohms which yielded an R2 of 350 ohms. This provided a steady 3V output that will be used to power the microcontroller and its devices for approximately 30.89 hours at full transmitting power.

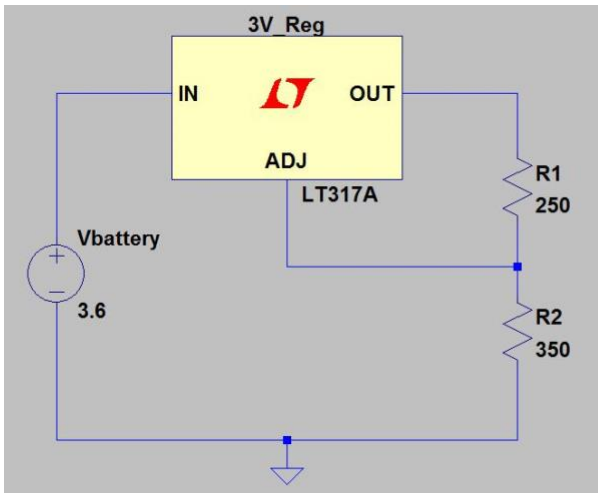
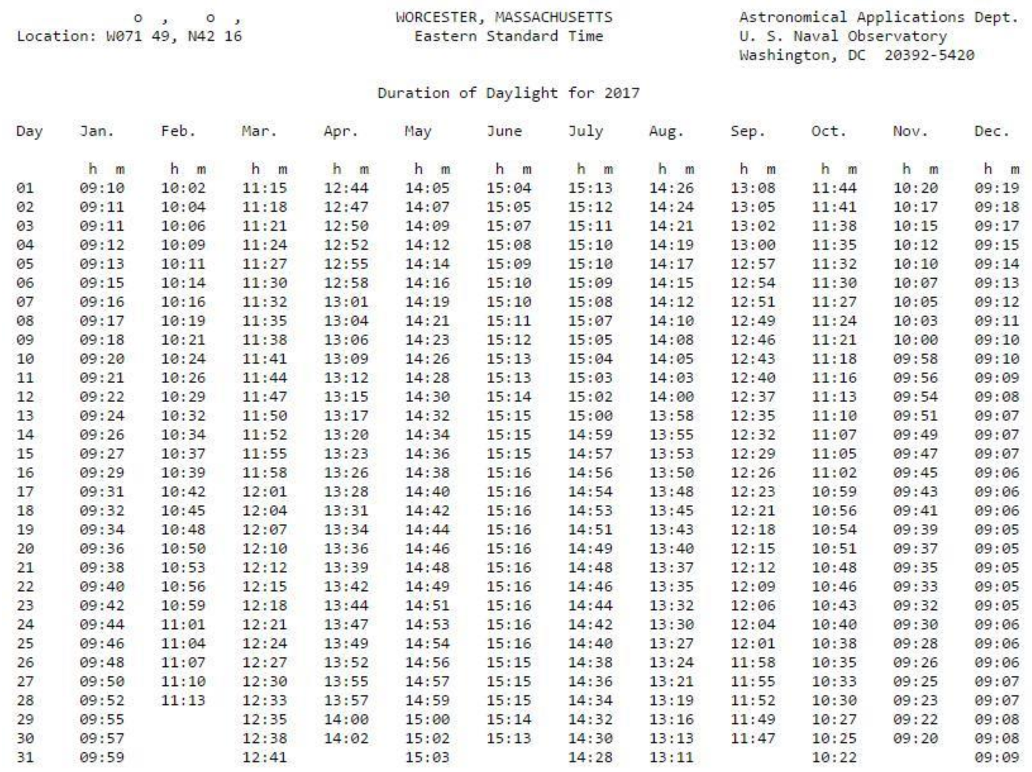


Figure 3.12: Tag's Power Circuit

For the beacon it was decided to use the same UART setup as the Tag using port 2's receive/transmit pins P2.2/P2.3 respectively as well as port 3's receive/transmit pins P3.2/P3.3 respectively. The two XBee peripherals would then be connected to these pins for the exchange of information. For simplicity the peripheral devices (XBee1 and XBee2) only required 3V to operate. The Xbee's had voltage regulators to allow the MSP432 controller's voltage output pins to supply power.

With this setup, most of the beacon is powered down with only the controller and XBee1 actively listening for the Tag's transmission. When a transmission is received, XBee2 can then be activated to transmit to the base station and the LED can flash as confirmation. This portion of our device uses more power than the Tag as XBee2 uses five times more current to transmit (since it transmits a farther distance). XBee1 needs to remain on since it is constantly waiting for a transmission. This device running at total max current draw (375.625mA) which consists of XBee1 (50mA), XBee2 (300mA), MSP (0.625mA) and an LED (20mA) can be run for a total of 13.3 hours if using a 5Ah battery. On low power mode the device can run for 98.8 hours without a charge.

We chose to use a 12V 5Ah Lead Acid battery for this portion of our project. Lead Acid batteries have a high capacity and are easily recharged making it a good choice for our project. The capacity will allow the beacon to run for approximately two weeks without a recharge. To recharge this battery a panel with a higher voltage was needed so we went with a 10W solar panel with an open circuit voltage of 22.41V and a short circuit current of 610mA. At the panel's max power point the voltage is 17.9V and the current is 560mA well above the requirements for recharging the chosen battery. The current from the panel can then refill an empty battery in under 9 hours of daylight. According to the U.S. Navy Observatory each day has more than 9 hours of sun each day (for Worcester, Ma). This means the 5Ah can fully recharge any energy the battery will consume daily.



```
           o  ,    o  ,
Location: W071 49, N42 16        WORCESTER, MASSACHUSETTS        Astronomical Applications Dept.
                                  Eastern Standard Time          U. S. Naval Observatory
                                                                 Washington, DC  20392-5420

                              Duration of Daylight for 2017
```

Day	Jan.	Feb.	Mar.	Apr.	May	June	July	Aug.	Sep.	Oct.	Nov.	Dec.
	h m	h m	h m	h m	h m	h m	h m	h m	h m	h m	h m	h m
01	09:10	10:02	11:15	12:44	14:05	15:04	15:13	14:26	13:08	11:44	10:20	09:19
02	09:11	10:04	11:18	12:47	14:07	15:05	15:12	14:24	13:05	11:41	10:17	09:18
03	09:11	10:06	11:21	12:50	14:09	15:07	15:11	14:21	13:02	11:38	10:15	09:17
04	09:12	10:09	11:24	12:52	14:12	15:08	15:10	14:19	13:00	11:35	10:12	09:15
05	09:13	10:11	11:27	12:55	14:14	15:09	15:10	14:17	12:57	11:32	10:10	09:14
06	09:15	10:14	11:30	12:58	14:16	15:10	15:09	14:15	12:54	11:30	10:07	09:13
07	09:16	10:16	11:32	13:01	14:19	15:10	15:08	14:12	12:51	11:27	10:05	09:12
08	09:17	10:19	11:35	13:04	14:21	15:11	15:07	14:10	12:49	11:24	10:03	09:11
09	09:18	10:21	11:38	13:06	14:23	15:12	15:05	14:08	12:46	11:21	10:00	09:10
10	09:20	10:24	11:41	13:09	14:26	15:13	15:04	14:05	12:43	11:18	09:58	09:10
11	09:21	10:26	11:44	13:12	14:28	15:13	15:03	14:03	12:40	11:16	09:56	09:09
12	09:22	10:29	11:47	13:15	14:30	15:14	15:02	14:00	12:37	11:13	09:54	09:08
13	09:24	10:32	11:50	13:17	14:32	15:15	15:00	13:58	12:35	11:10	09:51	09:07
14	09:26	10:34	11:52	13:20	14:34	15:15	14:59	13:55	12:32	11:07	09:49	09:07
15	09:27	10:37	11:55	13:23	14:36	15:15	14:57	13:53	12:29	11:05	09:47	09:07
16	09:29	10:39	11:58	13:26	14:38	15:16	14:56	13:50	12:26	11:02	09:45	09:06
17	09:31	10:42	12:01	13:28	14:40	15:16	14:54	13:48	12:23	10:59	09:43	09:06
18	09:32	10:45	12:04	13:31	14:42	15:16	14:53	13:45	12:21	10:56	09:41	09:06
19	09:34	10:48	12:07	13:34	14:44	15:16	14:51	13:43	12:18	10:54	09:39	09:05
20	09:36	10:50	12:10	13:36	14:46	15:16	14:49	13:40	12:15	10:51	09:37	09:05
21	09:38	10:53	12:12	13:39	14:48	15:16	14:48	13:37	12:12	10:48	09:35	09:05
22	09:40	10:56	12:15	13:42	14:49	15:16	14:46	13:35	12:09	10:46	09:33	09:05
23	09:42	10:59	12:18	13:44	14:51	15:16	14:44	13:32	12:06	10:43	09:32	09:05
24	09:44	11:01	12:21	13:47	14:53	15:16	14:42	13:30	12:04	10:40	09:30	09:06
25	09:46	11:04	12:24	13:49	14:54	15:16	14:40	13:27	12:01	10:38	09:28	09:06
26	09:48	11:07	12:27	13:52	14:56	15:15	14:38	13:24	11:58	10:35	09:26	09:06
27	09:50	11:10	12:30	13:55	14:57	15:15	14:36	13:21	11:55	10:33	09:25	09:07
28	09:52	11:13	12:33	13:57	14:59	15:15	14:34	13:19	11:52	10:30	09:23	09:07
29	09:55		12:35	14:00	15:00	15:14	14:32	13:16	11:49	10:27	09:22	09:08
30	09:57		12:38	14:02	15:02	15:13	14:30	13:13	11:47	10:25	09:20	09:08
31	09:59		12:41		15:03		14:28	13:11		10:22		09:09

Figure 3.13: Daylight Hours 2017 (Worcester, MA)

The beacon's power circuit was a little more complicated than the Tag's circuit as a solar panel is needed to recharge the battery. The first step was to filter and lower the voltage the solar panel was producing. This is due to the original high voltage waveform causing harm to the battery. In the circuit is a 0.1 microfarad filter capacitor reducing any ripple voltage altering the output to essentially DC. This is then put through a LM7815 (represented by a LT317A regulator in Figure 5.12) fixed voltage regulator which outputs a steady 15V. This regulator was chosen as it was close to the charging voltage for the 12V battery. The diode (D1) in place serves two purposes. First, it blocks the battery from passing voltage through the voltage regulator if the battery has the higher potential. Secondly the diode drops the 15V down to about 14.3V which is what is used to charge the battery. With the renewable energy portion of the circuit designed the battery voltage was needed to be dropped down to a usable level. Like the Tag the LM317 adjustable voltage regulator was used. The same ratio of resistors to get a 3V output for the microcontroller and its peripheral devices was also used. This power circuit will allow the beacon to run without charge for 13.3 hours at full power and 98.8 hours if only the beacon is listening. The solar panel at max power will charge the battery in just under 9 hours in sunlight fully replenishing the battery everyday allowing to the device to function solely on battery power when needed.

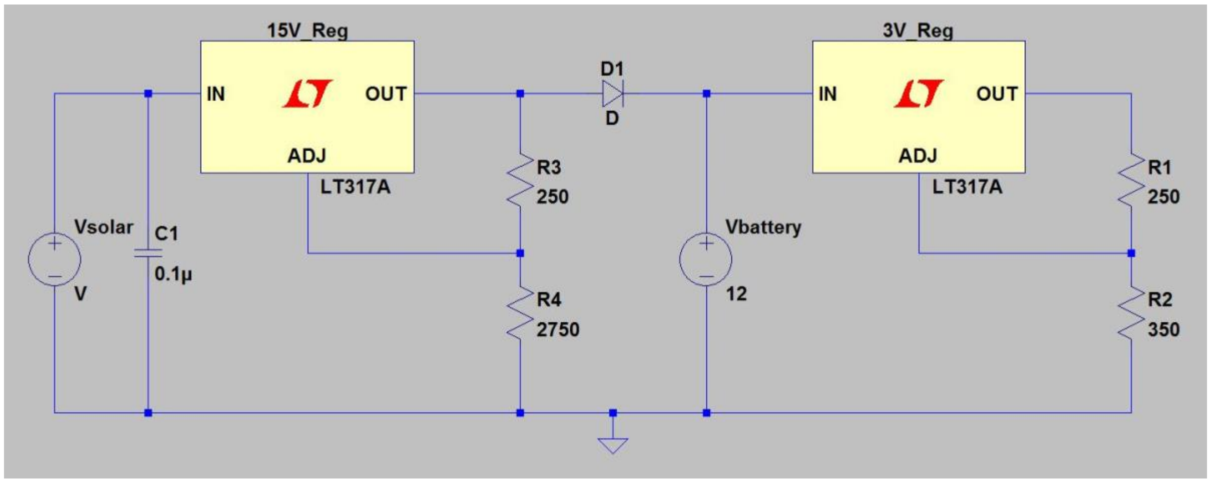


Figure 3.14: Beacon's Power Circuit

4. Methodology and Implementation

This section explains the process that was used to test individual modules. This section also provides explanation on how modules were tested together in order to emulate different aspects of the main device. The testing procedures for the devices in the system were described.

4.1 Setup

For the complete system, we first thought the beacons should be deployed along the hiking trails much like how street lights are spaced along busy streets as shown below in Figure 4.1. This would reduce the amount of beacons needed. The beacons would be spaced in a manner where the hiker's Tag is always in range of at least one beacon. The beacons would then be in range of the base station to communicate if there is an emergency.

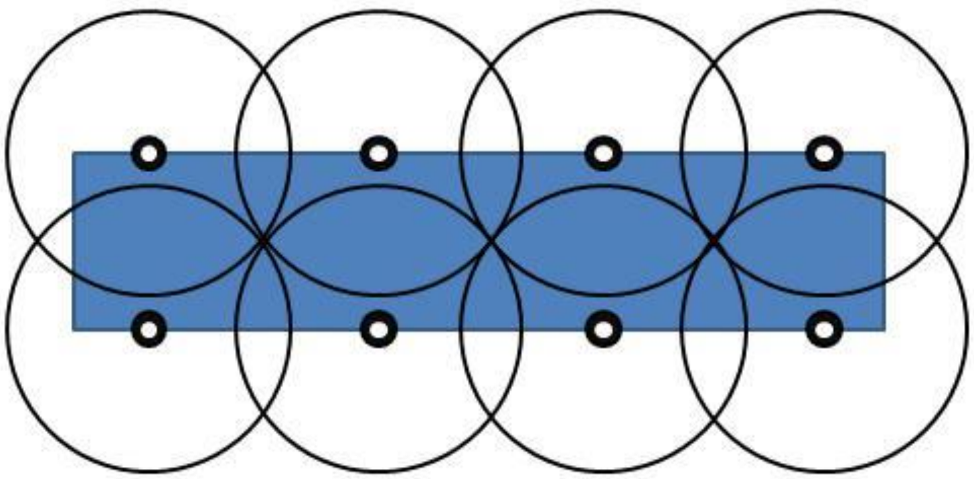


Figure 4.0: Trail Setup

The black outlined circles indicates the range (100m) where the Tag's XBee1 can communicate with the beacon's XBee1 while the blue rectangle indicates the hiking trail. The Beacons (white circles) will then need to be within 1Mi of the base station.

As the project progressed we noticed that placing beacons just along the trail might have some gaps in providing safety to the hiker. For instance, if the hiker was walking on a trail near a cliff and fell off or if a hiker just accidentally walked off of a trail's path they would not be in a position to activate their Tag. The signal would then not be received by a beacon. This was a major flaw in how we were planning to implement the system.

To solve the major flaw described above, the solution shown in Figure 4.2 was discovered. This encompasses the entire park. The beacons are spaced out in a way that the hiker is always within range of a beacon. This will ensure that the hikers in the park are always able to communicate with park staff as the outermost beacons will be within the usable range of the base station. For our project, we are imagining a smaller park although the project could be adapted for bigger national parks. This would cost more initially but would lower costs for long term. It would increase the chances of finding the hikers safely.

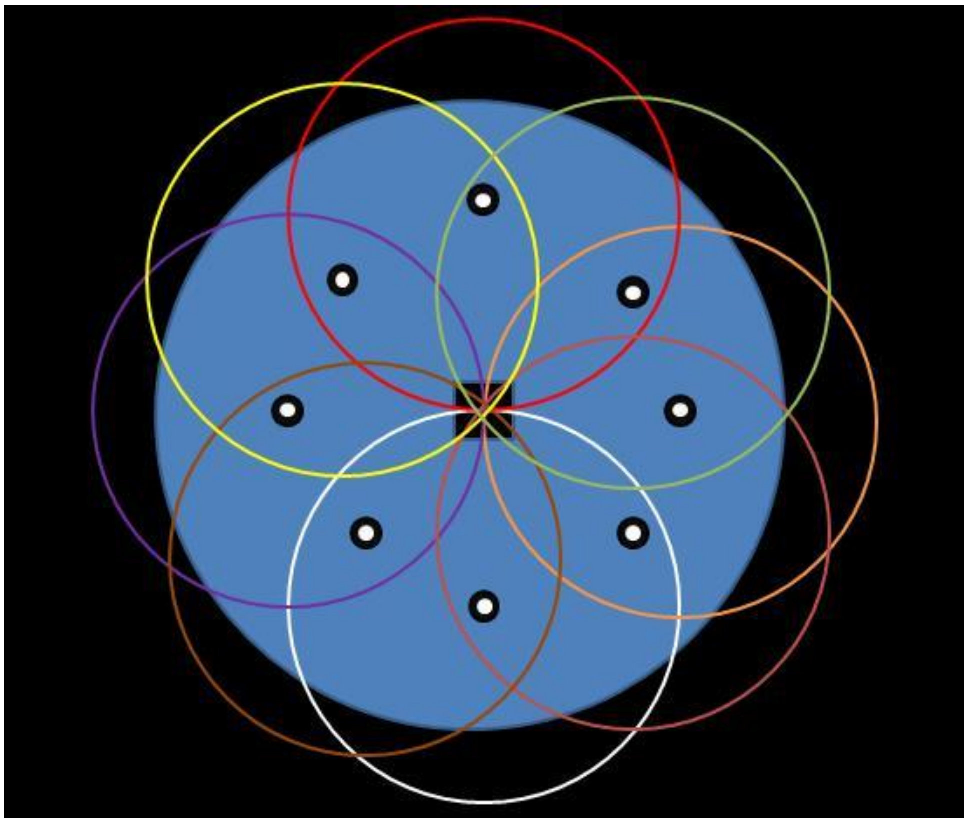


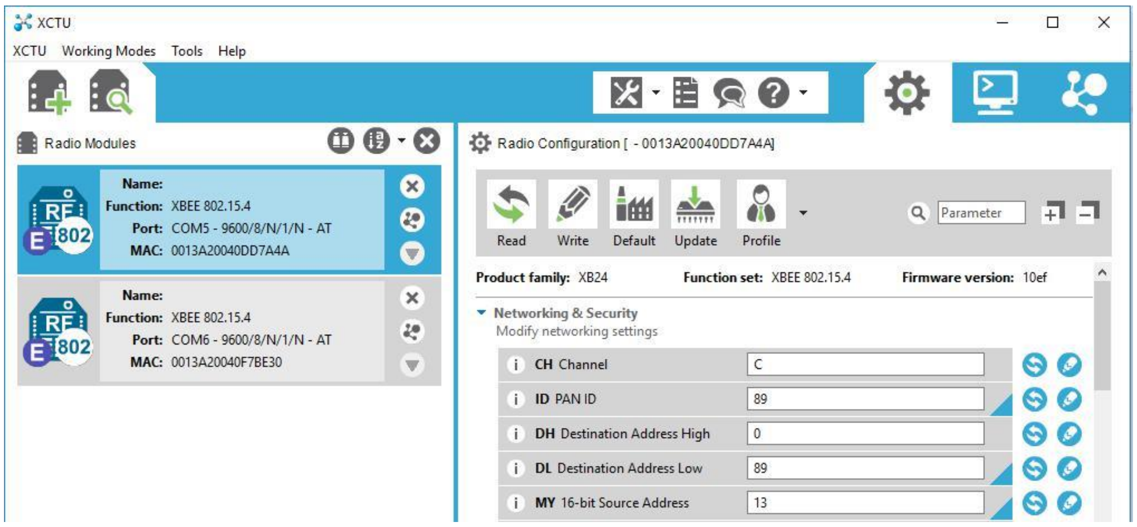
Figure 4.1: Park Setup

The black outlined circles indicate the range (100m) where the Tag's XBee1 can communicate with the beacon's (white circle) XBee1. While the multicolored circle's display the beacon's XBee2 range which intersects with the base station (black box in the center). The blue circle centered on the base station is the max range its XBee2 has.

4.2 Xbee Communication Testing

To determine that the Xbee's were working correctly, an Xbee network had to be created for the devices to connect. Using the X-CTU software, it was determined that the Xbee modules were configured correctly and found in the network. If an XBee device was not found in the network, than a possible problem could arise when using the Xbee with the microcontroller. A possible problem is the device receiving data correctly from the microcontroller but the Xbee not being configured correctly in the network. The first step was to configure all four Xbee's in API mode. This was done by placing the Xbee into the USB breakout board and loading the correct firmware on the Xbee. The next step was to make sure all Xbees in the network had the same PAN ID and SC value. This ensured that all of the XBee's would be in the same network. This ensured that the devices would be able to communicate with each other.

In order to make the transmission process easy, one Xbee1 would act only as a transmitter while the other XBee1 would act as a receiver. We were able to do this by changing the address of the receiving XBee1 to 89. Then the transmitting XBee1 would send its low byte to address 89 (receiving XBee1). The different XBee1 setups are shown in Figures 4.2 and 4.3 below. The channel and PAN ID are the same for both XBee1s allowing them to be in the same network and communicate. Although the channel and ID are the same they are unique. This allows a unique network to be created. This will help avoid interference from other Xbee users.


XCTU
XCTU Working Modes Tools Help
Radio Modules
Radio Configuration [- 0013A20040DD7A4A]
Read
Write
Default
Update
Profile
Parameter
Networking & Security
Modify networking settings

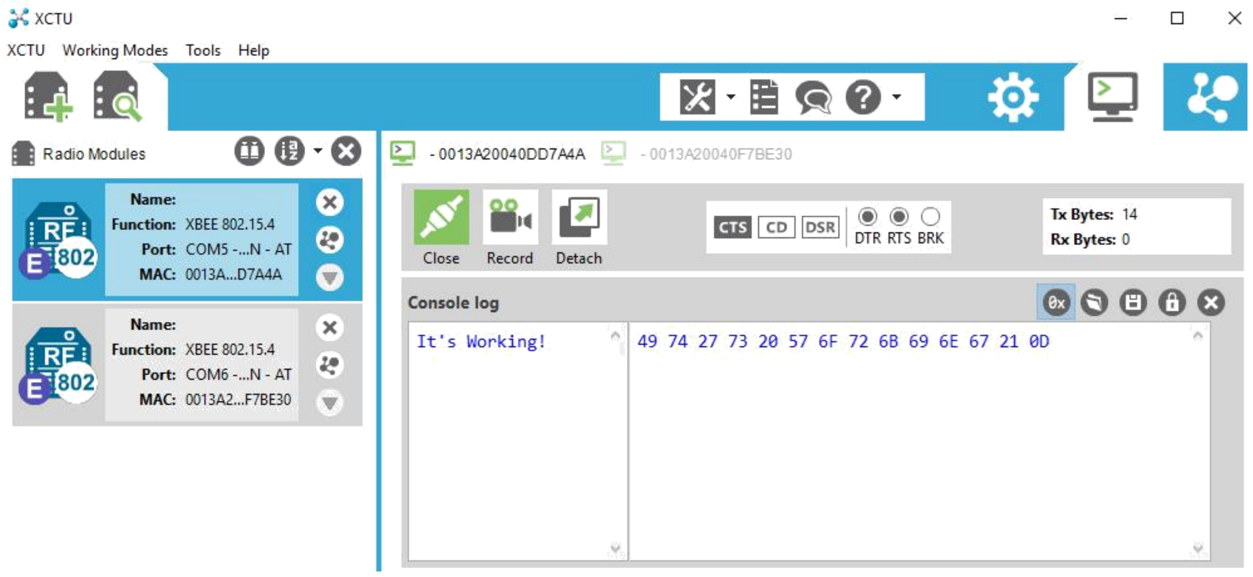
Name:
Function: XBEE 802.15.4
Port: COM5 - 9600/8/N/1/N - AT
MAC: 0013A20040DD7A4A

Name:
Function: XBEE 802.15.4
Port: COM6 - 9600/8/N/1/N - AT
MAC: 0013A20040F7BE30

Product family: XB24 Function set: XBEE 802.15.4 Firmware version: 10ef

CH Channel	C
ID PAN ID	89
DH Destination Address High	0
DL Destination Address Low	89
MY 16-bit Source Address	13

Figure 4.2: Tx XBee1 Setup



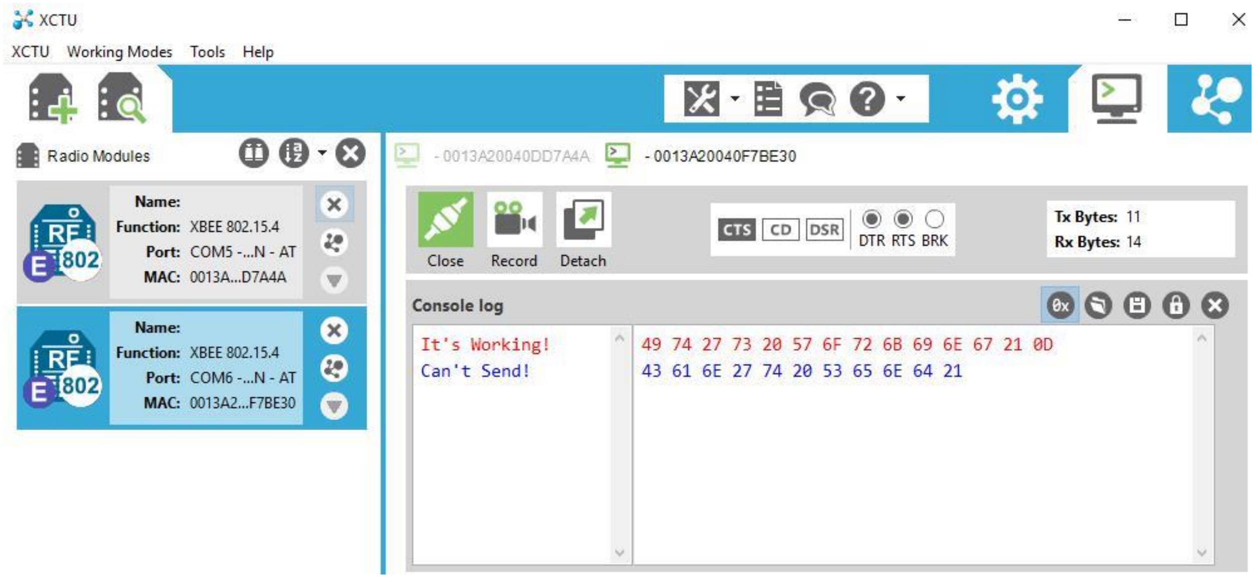
```
It's Working!
```

```
49 74 27 73 20 57 6F 72 6B 69 6E 67 21 0D
```

Figure 4.3: Rx XBee1 Setup

With both XBee1s set up, we then tested the connection by switching the XCTU program from setup mode to console mode. This mode was used to write and observed what was transmitted and received by each XBee1. The statements being transmitted are shown in blue and the statements being received are shown in red. For our designated transmitting XBee1 we started by writing “It’s Working!” in the console log. We then clicked over to our receiving Xbee1 (Figure 4.4) to see if the statement was received. To check if the receiving XBee1 could send to the transmitting XBee1 we wrote “Can’t Send!” in the console. The transmitting XBee1 didn’t receive the message as was expected because the receiving XBee1 sends to a different address.

With the pair of XBee1s configured, we moved on to configuring and testing the longer ranged pair of XBee2s. We repeated the same process as before with the XBee1s but changed the PAN ID to 13 as a precaution. This was strictly precautionary due to the zigbee protocol not allowing communication between the Xbee1 and the Xbee2.

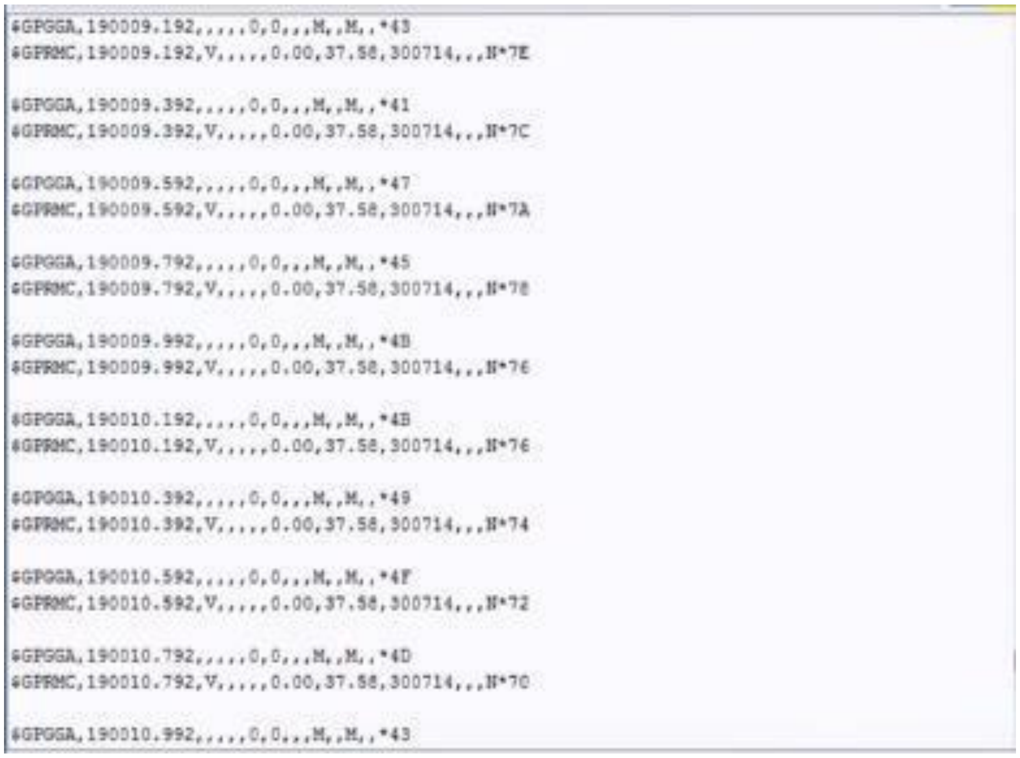


```
It's Working!    49 74 27 73 20 57 6F 72 6B 69 6E 67 21 0D
Can't Send!      43 61 6E 27 74 20 53 65 6E 64 21
```

Figure 4.4: Rx XBee1 Test

4.3 GPS Testing

For testing the AdaFruit GPS a driver library was created. For the GPS we downloaded and installed the library into Energia. With this library we were able to connect the GPS to the UART pins in port 3. The GPS started to send data automatically once powered up. The data was sent to the computer's serial port to see the raw GPS data. The NMEA 0183 format sends a lot of raw data so the program was edit (Appendix A "GPSTest") using the AdaFruit library to send only what was necessary. In Figure 4.6, the parsed GPS data is shown, the initial test occurred indoors so this is a general position. However, if we were located outside the specific coordinates from the satellite would be displayed.



```
$GPGGA,190009.192,,,,,0,0,,,M,,M,,*43
$GPRMC,190009.192,V,,,,,0.00,37.58,300714,,,N*7E

$GPGGA,190009.392,,,,,0,0,,,M,,M,,*41
$GPRMC,190009.392,V,,,,,0.00,37.58,300714,,,N*7C

$GPGGA,190009.592,,,,,0,0,,,M,,M,,*47
$GPRMC,190009.592,V,,,,,0.00,37.58,300714,,,N*7A

$GPGGA,190009.792,,,,,0,0,,,M,,M,,*45
$GPRMC,190009.792,V,,,,,0.00,37.58,300714,,,N*78

$GPGGA,190009.992,,,,,0,0,,,M,,M,,*4B
$GPRMC,190009.992,V,,,,,0.00,37.58,300714,,,N*76

$GPGGA,190010.192,,,,,0,0,,,M,,M,,*4B
$GPRMC,190010.192,V,,,,,0.00,37.58,300714,,,N*76

$GPGGA,190010.392,,,,,0,0,,,M,,M,,*49
$GPRMC,190010.392,V,,,,,0.00,37.58,300714,,,N*74

$GPGGA,190010.592,,,,,0,0,,,M,,M,,*4F
$GPRMC,190010.592,V,,,,,0.00,37.58,300714,,,N*72

$GPGGA,190010.792,,,,,0,0,,,M,,M,,*4D
$GPRMC,190010.792,V,,,,,0.00,37.58,300714,,,N*70

$GPGGA,190010.992,,,,,0,0,,,M,,M,,*43
```

Figure 4.5: Serial Monitor Results

4.4 LCD Display Testing

With a working GPS, we then wanted to see if we could print to the LCD. First we used an LCD example (Appendix A "LCD Example") already created by Energia to confirm LCD functionality. Then we edited the file (Appendix A "LCDTest") to take input from UART pins and display them on the LCD as if the GPS data was being received from the XBee. This program didn't work quite as expected, the LCD would display single characters at a time then overwrite them, but not the entire sentence at once. We also noticed when printing to the LCD that there would be too many characters from GPS string to be nicely displayed. With this information it was determined that it is better to display the information on the actual computer screen, this would reduce cost and be easier to implement.

4.5 Coding Problem

During the process of trying to program all the different MSP432s we realized that Energia didn't map out all of the different physical UART pins. The only UART ports I could use were port 3 and the computer's port. This wasn't a problem yet but it meant we couldn't use the serial monitor so we could not see what was being sent and/or received. When the XBee was connected to the computer's port as a substitute, there was no way to be sure the correct GPS data was being written to the XBee. This meant we did not know if the data from the Tag would make it to the beacon.

This grew to be a problem. It was decided to switch to Code Composer Studio (CCS) where it was slightly more difficult to use but provided us with more useful features. This meant that the pins how to be setup manually. This led us to download CCS version 6 and all of the necessary drivers to use the controllers. We ran a blink LED program again to make sure we had everything setup and running properly first.

Next, a program (Appendix B "CCS UART Test") was edited to test the transmission of data from one port to another. For this program, we had to configure certain pins to UART mode and setup whether they were input or outputs as well as create interrupts to handle receiving and transmitting data. For the most part it worked initially but we noticed inside the MSP432 buffers the data would appear in the Tx register but then wouldn't be sent to the Rx register we've assumed we might be missing one line of code to receive the data that is being transmitted but as we were debugging we realized that the port was transmitting but we didn't connect the Tx pin to an Rx pin. When the Tx pin was connected to an Rx pin we could then see the data in the Rx buffer. This offered us valuable information in using the computer's port for UART communication as we could now test transmission in Energia using pins 1.2 (Rx) and 1.3 (Tx) then sending the data to a second microcontroller and viewing the data in the serial monitor. Now with a way to view the transferring of data with Energia we started using Energia again.

5. Testing and Results

This section explains the process that was used to test each device separately (Tag, Beacon, Base Station). This section also provides explanation on how the devices were tested together in order to test the system as a whole.

5.1 Tag to Beacon

The testing of the Tag was conducted. In order to make sure we were sending and receiving the proper data we setup our hardware without the GPS connected so we could maintain control of the data by sending a hardcoded string. This hardcoded string was sent from the Tag's microcontroller to the XBee1 (Appendix B "UART Tx Test") and wirelessly transmitted to the beacon's XBee1 this data was then transferred to the beacon's computer port to be displayed in the serial monitor (Appendix B "UART Rx Test"). The first line in the serial monitor unfortunately random characters. We then decided to retry and give the XBees time to power up first. This greatly improved the data received and showed a repeating series of numerical characters. We went back into the code and noticed that we were using the line "Serial.print(Serial1.read());" to display to the monitor which was throwing off our results by encoding the string into an unreadable format. Simply switching from "Serial.print(Serial1.read());" to "Serial.write(Serial1.read());" displayed our hardcoded string perfectly not only confirming proper communication between both XBee1s but that we could use the computer's UART pins for other devices.

5.2 Beacon to Base Station

With the pair of XBee1s communicating properly as part of the system we needed to then test the pair of XBee2s. Upon going to test the XBee2s it was noticed that we couldn't use this pair of XBees with a breadboard due to the pins being too close together (2mm). We used the XBee USB boards since they had a place to solder on pins. After going through the schematic of the XBee USB boards we soldered pins to the boards. We then replaced the

XBee1s in the system with the XBee2s then made sure the hardcoded string was still sent perfectly, which it was.

Although when our hardware was changed to use both pairs of XBees we saw that one XBee Explorer's power LED was not lighting this led us to believe either the Explorer wasn't getting any power or that the LED was not working. We ruled out the possibility of the LED not working by plugging a USB into the Explorer port and seeing the power LED light up. Knowing that the Explorer wasn't getting power we tried using a different breadboard, adjusting the soldered pins and checking the individual XBees to see if it still worked as expected. The XBees worked like normal but nothing we did powered up the Explorer, after a few hours of trying to figure out how to fix it we noticed that one of the connections was a little skewed so we had to place the Explorer in the breadboard slightly crooked and it powered up perfectly.

The XBee Explorer powering up correctly allowed us to test both pairs of XBees together. Instead of writing to the serial monitor we removed the MSP432 jumper and attached the Tx pin to the XBee2 to wirelessly transmit to the base station XBee2 and then to the serial monitor. We originally got the same numerical characters from the first test but since we have seen this problem before, we knew that the problem was "Serial.print(Serial1.read());" instead of "Serial.write(Serial1.read())." During this test we still were using the hardcoded strings.

The hardcoded string that was sent made it successfully from the Tag to the base station's serial monitor. We then exchanged the hardcoded string for the actual GPS module. Since this initial testing was indoors we expected a general string of GPS data with no coordinates shown. Our expectations were confirmed as the necessary strings from the GPS would still be transmitted to the serial monitor.

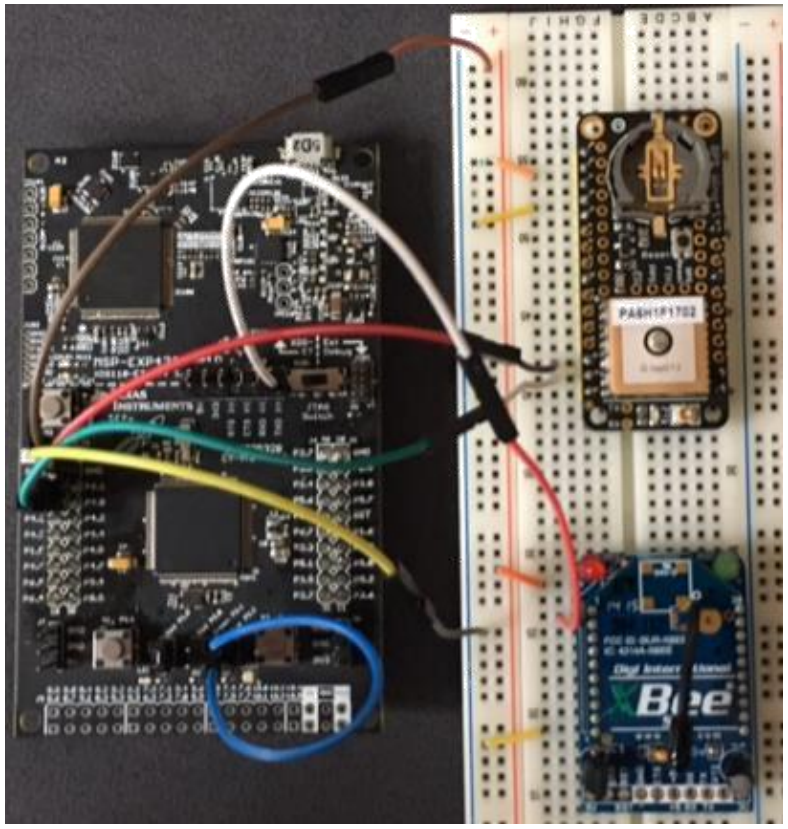


Figure 5.0: Fully Assembled Tag Hardware

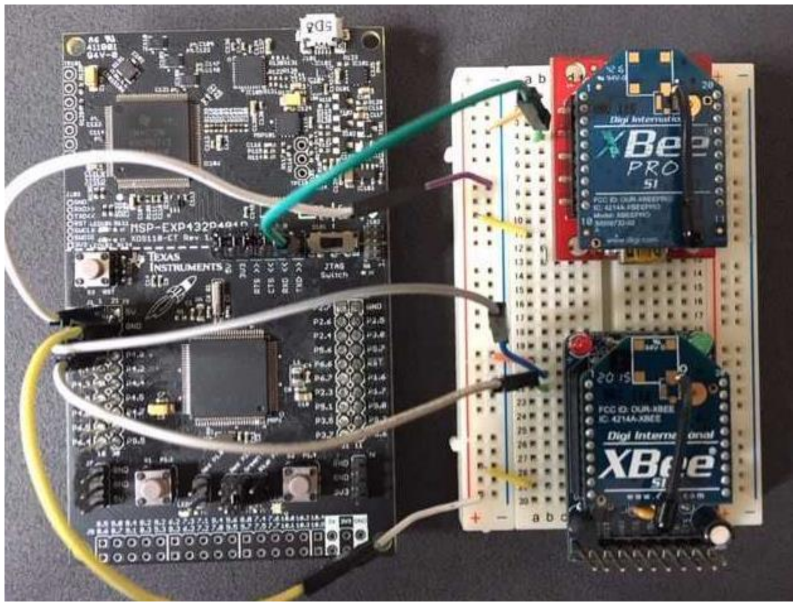


Figure 5.1: Fully Assembled Beacon Hardware

Lastly with the core of the project assembled and tested we made the GPS data on the serial monitor a little bit more user friendly so the park staff would be able to easily see what the longitude, latitude and altitude of the Tag are. The complete Tag code (Appendix C "Final Tag Code") will only print the relevant info (longitude, latitude and altitude) if the GPS has a satellite fix, if the GPS does not have a fix it will print the two necessary NMEA strings in the format of the hardcoded strings shown above previously. To show data transmission for all three different portions of our device we use LEDs so we can have confirmation of when data is being sent and/or received.

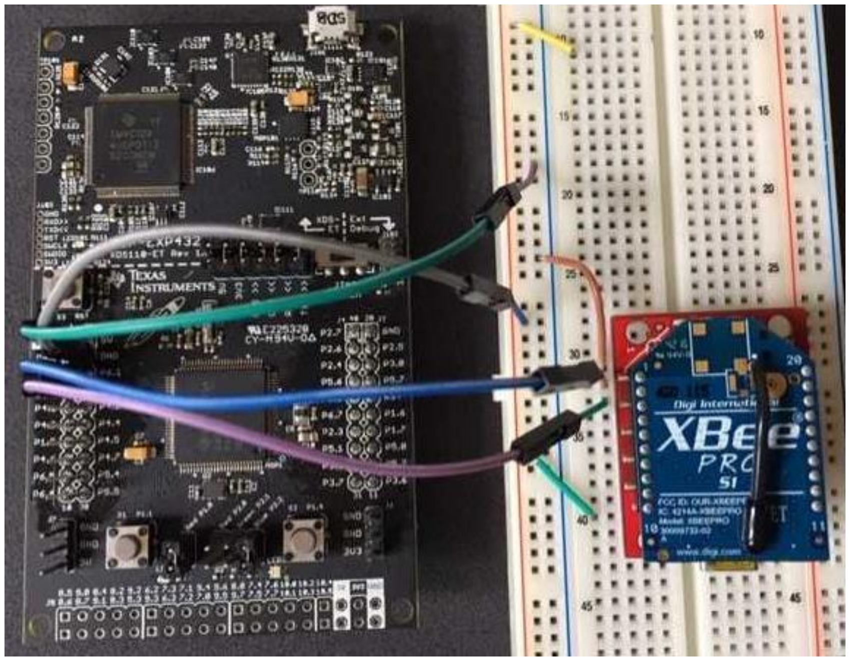
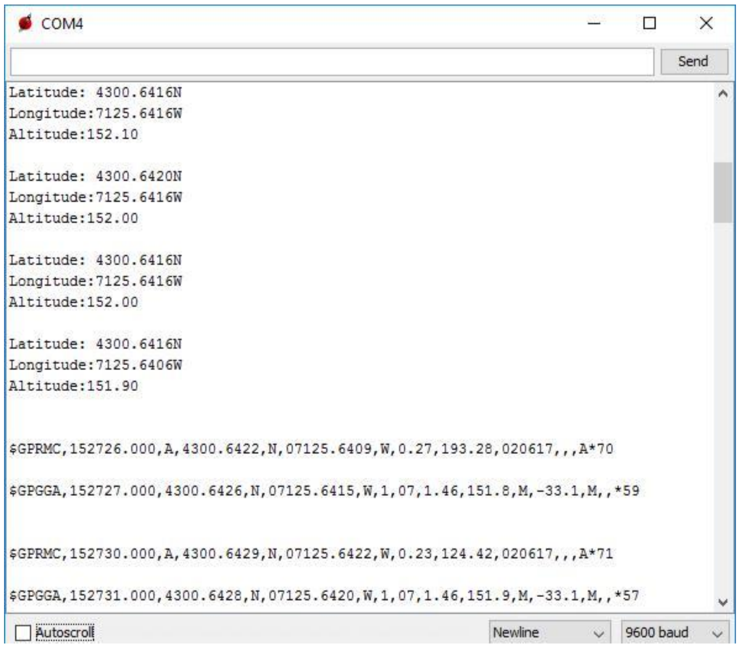


Figure 5.2: Fully Assembled Base Station Hardware

Finally with full and complete code the hardware and software were ready to be tested out in the real world. We attached the Tag to a laptop as a power source and roamed around a neighborhood in Manchester, NH. The beacon was using an outside outlet for power while the base station was also connected to a laptop inside of a house. We proceeded to walk around the neighborhood with just the Tag for a few minutes collecting GPS data every 10 seconds. Accuracy testing for the Safety Sensor was done by taking a sample of the data collected and entering it into Google Maps in a specified format. When the coordinates were entered, Google Maps quickly zoomed onto our location showing our device truly worked as anticipated. Furthermore, we did an experiment with how long it took the Tag to send relevant information as the GPS needs time to power up and find where it is. From a cold start and sending instantly it took between 55 (NMEA format) and 65 (fixed format) seconds to send good info. Although if the GPS already had power (a warm start), we saw it start sending good data in approximately 25 seconds for the fixed format GPS data.



```
Latitude: 4300.6416N
Longitude:7125.6416W
Altitude:152.10

Latitude: 4300.6420N
Longitude:7125.6416W
Altitude:152.00

Latitude: 4300.6416N
Longitude:7125.6416W
Altitude:152.00

Latitude: 4300.6416N
Longitude:7125.6406W
Altitude:151.90


$GPRMC,152726.000,A,4300.6422,N,07125.6409,W,0.27,193.28,020617,,,A*70

$GPGGA,152727.000,4300.6426,N,07125.6415,W,1,07,1.46,151.8,M,-33.1,M,,*59


$GPRMC,152730.000,A,4300.6429,N,07125.6422,W,0.23,124.42,020617,,,A*71

$GPGGA,152731.000,4300.6428,N,07125.6420,W,1,07,1.46,151.9,M,-33.1,M,,*57
```

Figure 5.3: Base Station Results (Top Format: With Fix, Bottom Format: Without Fix)

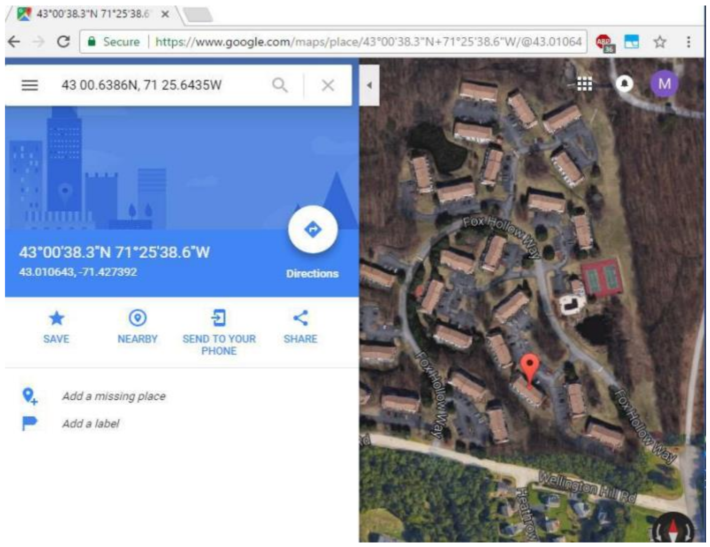


Figure 5.4: Tag's GPS Coordinates in Google Maps

This device runs properly on micro USB power from an outlet or computer which can limit mobility of the device. So, we designed power circuits for both the Tag and beacon (not needed for the base station) to represent how they would be used in the real world. Upon further inspection of the MSP432’s datasheet we saw that the microcontroller itself can handle 4.17V max but the pins can take 3.7Vmax. This is adequate since our three batteries in series only produce 3.6V. We also noted that the pin we use to power the GPS and XBee peripherals is physically connected to the pin we are going to put the battery power on meaning any voltage we use will be put through our peripheral devices. Our devices aren’t able to handle the 3.6V that we will be using as they run on a voltage range of 2.8V to 3.4V so our external voltage needs to be reduced to a usable level (3V or 3.3V) or risk being damaged.

6. Conclusions

Overall our goal was to improve hiker safety by creating a device that can send their GPS coordinates to park staff in case of an emergency. We were able to successfully receive GPS data from the AdaFruit GPS module and send it from the Tag through the Beacon to the Base Station. We were able to display the GPS coordinates on the computer which then can be entered into Google Maps to show the physical location of the Tag. During this project we faced some tough challenges but were able to overcome them.

This device provides a level of safety for a hiker that has yet to be seen. If a hiker has gone missing or was injured this device will notify emergency personal of this occurrence. Without the device there is a reduce chance of getting the help needed in a timely manner and will increase cost and the amount of man hours required to find the missing or hurt hiker. Our device is needed to help assist both the hiker and park staff in having a safe and friendly visit to hiking trails and parks.

Although we have a fully completed system that accomplishes all the goals we set in the beginning of the project the system we had was never integrated and can be improved.

7. Future Works

At the completion of our MQP, we identified ideas that could be used to help further the development of the device.

In order for the device to be used by hikers the device would need to be made portable. In the final stages of our project the Tag portion of the system was not portable and remained on bread boards. In order for this to occur printed circuit boards would need to be created. A casing would need to be designed that would not only enclose the Tag and protect the contents but be user friendly. The casing for the Beacon would need to be designed to withstand the unpredictable weather conditions since the Beacon will stay outside for the life of the system.

Another suggestion for the project was to create multiple Beacons in order to determine how the Beacons would communicate in order to relay the message back to the Base Station.

The last suggestion four our project would be to use the Xbees that are created for longer data transmissions to see if Xbee's are the device to use for long distance data transmission or if they should only be used for short distance data transmission.

References

[1] Martha Waggoner. “Researchers Tackle Campus Safety Devices.” Web. 19 Sep 2013. .<http://www.ecu.edu/cs-admin/news/041305campussafetu.cfm>

[2] ”RAVENAlert Keychain - Campus Emergency Alerts”. Web. 19 Sep 2013. <http://www.intelliguardsystems.com/students-parents.php>

[3] Vogler, Elise. "Bluetooth vs. Wi-Fi Power." *It Still Works*. N.p., n.d. Web. <http://itstillworks.com/bluetooth-vs-wifi-power-consumption-17630.html>.

[4] Gislason, Drew. Zigbee Wireless Networking. Newnes, 2008. Books24x7. Web. Oct. 19, 2013.<http://common.books24x7.com.ezproxy.wpi.edu/toc.aspx?bookid=32185>

[5] http://www.areavibes.com/springfield-ma/forest+park/crime/

[6] permut1979disaster, Disaster alert system, Permut, A.A. and Permut, A.R. and Permut, R.M., <https://www.google.com/patents/US4155042>, 1979, may 15, Google Patents, US Patent 4,155,042

[7] Daylight or Darkness Duration Table for One Year. U.S. Navy Observatory, n.d. Web. 23 May 2017.

[8] Texas Instruments Incorporated. "MSP432P401R Datasheet." (n.d.): n. pag. TI, Mar. 2017. Web. <http://www.ti.com/lit/ds/symlink/msp432p401m.pdf>.

[9] Digi. "XBee Datasheet." (n.d.): n. pag. SparkFun, 2009. Web. <https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf>.

[10] Lady Ada. "AdaFruit Ultimate GPS FeatherWing Datasheet." (n.d.): n. pag. AdaFruit, 16 May 2017. Web. <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps-featherwing.pdf>.

[11] "Pros vs Cons of Wifi and Bluetooth." Quora, 2015. Web. <https://www.quora.com/What-are-the-pros-and-cons-of-WiFi-Direct-versus-Bluetooth-Classic>

[12] "Arduino Board Specifications." Arduino, 2017. Web. <https://www.arduino.cc/en/Products/Compare>.

[13] GPS Visualizer: Do-It-Yourself Mapping. Web. 10 Oct 2013. http://www.gpsvisualizer.com/

[14] Berg, Andrew. "Wireless Charging and a Tale of Two Standards." Wireless Week. Advantage Business Media, 03 Jul 2013. Web. 6 Oct. 2013.

[15] Higginbotham, Stacey. "10 Things to Know About Wireless Power." Gigaom. Wordpress.com, 04 Oct 2009. Web. 19 Oct. 2013.

[16] Pollicino, Joe. "WPC updates Qi standard, increases inductive charging distance to 40mm." Engadget. AOL Inc., 20 Apr 2012. Web. 16 Oct. 2013.

[17] Bodo's Power Systems. Wireless Power By IDT. 2013. 3-4. eBook.

[18] About Us. Power Matters Alliance. Power Matters Alliance, Inc. Web. 16 Oct. 2013.

[19] Who are we. Alliance for Wireless Power. Alliance for Wireless Power, 2012. Web. 16 Oct. 2013.

[20] Rouse, Margaret. "Resonance Charging." Search Mobile Computing. 2008. .

Appendices

Appendix A: GPS Test

```
#include <Adafruit_GPS.h>

#define GPSSerial Serial1

Adafruit_GPS GPS(&GPSSerial);

String NMEA1;
String NMEA2;
char c;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("If you're reading this...the code may work");
  GPS.begin(9600); GPS.sendCommand("$PGCMD,33,0*6D");
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  pinMode(77, OUTPUT);
  delay(1000);

}
void loop() {
  // put your main code here, to run repeatedly:
  readGPS();
  delay(20);
}

void readGPS() {
  clearGPS();
  while(!GPS.newNMEAreceived()) {
        c=GPS.read();
  }

  GPS.parse(GPS.lastNMEA());
  NMEA1=GPS.lastNMEA();

  while(!GPS.newNMEAreceived()) {
        c=GPS.read();
  }

  GPS.parse(GPS.lastNMEA());
  NMEA2=GPS.lastNMEA();

  Serial.println(NMEA1);
  Serial.println(NMEA2);
  Serial.println("");

  digitalWrite(77, HIGH);
  delay(100);
  digitalWrite(77, LOW);

}

void clearGPS() {

  while(!GPS.newNMEAreceived()) {
        c=GPS.read();
  }

  GPS.parse(GPS.lastNMEA());

  while(!GPS.newNMEAreceived()) {
        c=GPS.read();
  }

  GPS.parse(GPS.lastNMEA());
}
```

Appendix B: LCD Example

```
//
//  Sharp BoosterPackLCD SPI
//  Example for library for Sharp BoosterPack LCD with hardware SPI
//
//
//  Author : Stefan Schauer
//  Date  : Jan 29, 2014
//  Version: 1.00
//  File  :  LCD_SharpBoosterPack_SPI_main.ino
//
//  Version:  1.01 : added support for CC3200
//
//  Based on the LCD5110 Library
//  Created by Rei VILO on 28/05/12
//  Copyright (c) 2012 http://embeddedcomputing.weebly.com
//  Licence CC = BY SA NC
//
//  Edited 2015-07-11 by ReiVilo
//  Added setOrientation(), setReverse() and flushReverse()
//

// Include application, user and local libraries
#include "SPI.h"
#include "OneMsTaskTimer.h"
#include "LCD_SharpBoosterPack_SPI.h"

// Variables
LCD_SharpBoosterPack_SPI myScreen;
uint8_t myOrientation = 0;
uint16_t myCount = 0;

// Add setup code
void setup() {
   Serial.begin(9600);

   myScreen.begin();
   myScreen.clearBuffer();
```

```
    myScreen.setFont(1);
    myScreen.text(10, 10, "Hello!");
    myScreen.flush();

    for (uint8_t i=0; i<20; i++) delay(100);
    myScreen.reverseFlush();
    for (uint8_t i=0; i<20; i++) delay(100);

    myScreen.clear();

    for (uint8_t i=0; i<4; i++)
    {
        myScreen.setOrientation(i);
        myScreen.text(10, 10, String(i));
        myScreen.flush();
    }
    for (uint8_t i=0; i<20; i++) delay(100);

    Serial.print("myCount = ");
}

// Add loop
code void loop()
{
    myCount++; Serial.print(-
    myCount, DEC); if (myCount
    > 16)
    {
        myOrientation++;
 //     if (myOrientation > 4) myOrientation = 0;
        myOrientation %= 4;
        myScreen.setOrientation(myOrientation);
        myCount = 0;
        Serial.println();
        Serial.print("**  myOrientation  =  ");
        Serial.println(myOrientation, DEC);
        Serial.print("myCount = ");
    }
    myScreen.clearBuffer();
    myScreen.setFont(0);
    myScreen.text(myCount, 10, "ABCDE", LCDWrapNone);
    for (uint8_t i=10; i<LCD_HORIZONTAL_MAX-10; i++) {
        myScreen.setXY(i,20,1);
    }

    myScreen.text(10,30,String(myCount,10));

    for (uint8_t i=0; i<=20; i++) {
        myScreen.setXY(50+i,30,1);
        //    }
        //    for (uint8_t i=0; i<=20; i++) {
        myScreen.setXY(50,30+i,1);
        //    }
        //    for (uint8_t i=0; i<=20; i++) {
        myScreen.setXY(50+i,50,1);
        //    }
        //    for (uint8_t i=0; i<=20; i++) {
        myScreen.setXY(70,30+i,1);
    }

    myScreen.setFont(1);
    myScreen.setCharXY(10, 40);
    myScreen.print("ABC");
    myScreen.setFont(0);
    myScreen.setCharXY(60, 60);
    myScreen.print(0x7F, HEX);
    myScreen.print(0x81, HEX);
    myScreen.setCharXY(10, 60);
    myScreen.println("Break!");
    myScreen.print("ABC\nabc");
    myScreen.flush();

    for (uint8_t i=0; i<2; i++) delay(100);
}
```

Appendix C: LCD Test

```
#include "SPI.h"
#include "OneMsTaskTimer.h"
#include "LCD_SharpBoosterPack_SPI.h"

// Variables
LCD_SharpBoosterPack_SPI myScreen;
char c;

// Add setup code
void setup() {
            Serial.begin(9600);
            myScreen.begin();
   // myScreen.clearBuffer();
            myScreen.setFont(0);
            myScreen.text(10, 10, "Eh!");
            myScreen.print("Ah!");
            myScreen.flush();
            delay(500);
            myScreen.clear();
}

// Add loop
code void loop()
{
  //Serial.println("From the Loop");
  printScreen();
  myScreen.setCharXY(10, 30);
  delay(1000); //myScreen.clear();

}

void printScreen() {
  myScreen.clearBuffer();
  myScreen.setFont(0);
  Serial.println("From the Function");
  if (Serial.available()) {
  delay(1);
  c=Serial.read();
  //myScreen.text(10,40,c);

  myScreen.print(c);
  myScreen.flush();


  digitalWrite(77, HIGH);
  delay(100);
  digitalWrite(77, LOW);
  }
}
```

Appendix D: CCS UART Test

```
/* DriverLib Includes */
#include "driverlib.h"

/* Standard Includes */
#include <stdint.h>

#include <stdbool.h>

/* UART Configuration Parameter. These are the configuration parameters to
 * make the eUSCI A UART module to operate with a 9600 baud rate. These
 * values were calculated using the online calculator that TI provides
 * at:
 *http://software-
dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html
 */

char info = 'g';
const eUSCI_UART_Config uartConfig =
{
	EUSCI_A_UART_CLOCKSOURCE_SMCLK,          // SMCLK Clock Source
	78,                                  // BRDIV = 78
	2,                                   // UCxBRF = 2
	0,                                   // UCxBRS = 0
	EUSCI_A_UART_NO_PARITY,                   // No Parity
	EUSCI_A_UART_LSB_FIRST,                  // LSB First
	EUSCI_A_UART_ONE_STOP_BIT,               // One stop bit
	EUSCI_A_UART_MODE,                     // UART mode
	EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION // Oversampling
};

void flash_once(void)
{
 MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
 //_delay_cycles(1000);
 //MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
```

```
}

void flash_one(void)
{
 MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);
 //_delay_cycles(1000);
 //MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
}

void flash_done(void)
{
 MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);
 //_delay_cycles(1000);
 //MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
}

int main(void)
{
    /* Halting WDT */
    MAP_WDT_A_holdTimer();

    // Selecting P1.2 and P1.3 in UART mode
    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1,
            GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);
    //Set LED 1.0 as output
    //MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    //MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);


    // Selecting P2.2 and P2.3 in UART mode
    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P2,
            GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);
    //Set LED 2.0 as output
    //MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);
    //MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);


    // Selecting P3.2 and P3.3 in UART mode
    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P3,
            GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);
    //Set LED 2.1 as output
    //MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);
    //MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);

    /* Setting DCO to 12MHz */
    CS_setDCOCenteredFrequency(CS_DCO_FREQUENCY_12);

    // Configuring UART Module0
    MAP_UART_initModule(EUSCI_A0_BASE, &uartConfig);

    // Configuring UART Module1
    MAP_UART_initModule(EUSCI_A1_BASE, &uartConfig);

    // Configuring UART Module2
    MAP_UART_initModule(EUSCI_A2_BASE, &uartConfig);

    // Enable UART module0
    MAP_UART_enableModule(EUSCI_A0_BASE);

    // Enable UART module1
    MAP_UART_enableModule(EUSCI_A1_BASE);

    // Enable UART module2
    MAP_UART_enableModule(EUSCI_A2_BASE);

    // Enabling interrupts0
    MAP_UART_enableInterrupt(EUSCI_A0_BASE,
EUSCI_A_UART_RECEIVE_INTERRUPT);
    MAP_Interrupt_enableInterrupt(INT_EUSCIA0);
    //MAP_Interrupt_enableSleepOnIsrExit();
    MAP_Interrupt_enableMaster();

    // Enabling interrupts1
    MAP_UART_enableInterrupt(EUSCI_A1_BASE,
EUSCI_A_UART_RECEIVE_INTERRUPT);
    MAP_Interrupt_enableInterrupt(INT_EUSCIA1);
    //MAP_Interrupt_enableSleepOnIsrExit();
    MAP_Interrupt_enableMaster();

    // Enabling interrupts2
    MAP_UART_enableInterrupt(EUSCI_A2_BASE,
EUSCI_A_UART_RECEIVE_INTERRUPT);
    MAP_Interrupt_enableInterrupt(INT_EUSCIA2);
   // MAP_Interrupt_enableSleepOnIsrExit();
    MAP_Interrupt_enableMaster();

    while(1)
    {
            UART_transmitData(EUSCI_A0_BASE, info);
            //UART_transmitData(EUSCI_A1_BASE, info);
            //UART_transmitData(EUSCI_A2_BASE, info);
            //MAP_PCM_gotoLPM0();
    }
}

// EUSCI A0 UART ISR - Echoes data back to PC host
void EUSCIA0_IRQHandler(void)
{
    uint32_t status = MAP_UART_getEnabledInterruptStatus(EUSCI_A0_BASE);

    flash_once();

    MAP_UART_clearInterruptFlag(EUSCI_A0_BASE, status);

    if(status & EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG)
    {
            flash_done();

        MAP_UART_transmitData(EUSCI_A0_BASE,
MAP_UART_receiveData(EUSCI_A0_BASE));
        //MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
    }

    //MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

}
```

```
// EUSCI A1 UART ISR - Echoes data from P2 to PC
host void EUSCIA1_IRQHandler(void)
   uint32_t status = MAP_UART_getEnabledInterruptStatus(EUSCI_A1_BASE);

   flash_one();

   MAP_UART_clearInterruptFlag(EUSCI_A1_BASE, status);

   if(status & EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG)
   {
         flash_done();
      MAP_UART_transmitData(EUSCI_A1_BASE,
MAP_UART_receiveData(EUSCI_A1_BASE));
      //MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);
   }

   //MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
}

// EUSCI A2 UART ISR - Echoes data P3 to P2
void EUSCIA2_IRQHandler(void)
{
   uint32_t status = MAP_UART_getEnabledInterruptStatus(EUSCI_A2_BASE);

   flash_once();

   MAP_UART_clearInterruptFlag(EUSCI_A2_BASE, status);

   if(status & EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG)
   {
         flash_done();
      MAP_UART_transmitData(EUSCI_A2_BASE,
MAP_UART_receiveData(EUSCI_A2_BASE));
      //MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);
   }

   //MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
}
```

Appendix E: UART Tx Test

```
void setup() {
  Serial.begin(9600);
  pinMode(77, OUTPUT);
  delay(1000);
}

void loop() {
Serial.println("$GPGGA,092750.000,5321.6802,N,00630.3372,W,1,8,1.03,61.7,M,55.2,M,,*76"
);
Serial.println("$GPRMC,092750.000,A,5321.6802,N,00630.3372,W,0.02,31.66,280511,,,A*43"
);
Serial.println("");

digitalWrite(77, HIGH);
delay(100);
digitalWrite(77, LOW);
delay(1000);
}
```

Appendix F: UART Rx Test

```
 void setup() {
  Serial.begin(115200);
  Serial1.begin(9600);
  pinMode(76, OUTPUT);
  delay(1000);
}

void loop() {
digitalWrite(76, LOW);
if (Serial1.available() > 0) {
  digitalWrite(76, HIGH);
  Serial.write(Serial1.read);
```

Appendix G: Final Tag Code

```
#include <Adafruit_GPS.h>

#define GPSSerial Serial1

Adafruit_GPS GPS(&GPSSerial);

char c;
int look = 0;
String NMEA1;
String NMEA2;
int buttonState = 0;
const int buttonPin = PUSH1;

void setup() {
  // put your setup code here, to run once: Serial.begin(9600);

  GPS.begin(9600); GPS.sendCommand("$PGCMD,33,0*6D");

  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);

  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);

  pinMode(77, OUTPUT);

  pinMode(78, OUTPUT);
  pinMode(buttonPin,
  INPUT_PULLUP); delay(1000);

}

void loop() {
  // put your main code here, to run repeatedly:
  while (look == 0) {
    buttonState = digitalRead(buttonPin);
    if (buttonState == LOW) {
      look = 1;
      digitalWrite(78, HIGH);
    }
  }
     readGPS();
     delay(10000);
}

void readGPS() {

  clearGPS();

  while(!GPS.newNMEAreceived()) {
    c=GPS.read();
  }

  GPS.parse(GPS.lastNMEA());
  NMEA1=GPS.lastNMEA();

  while(!GPS.newNMEAreceived()) {
    c=GPS.read();
  }

  GPS.parse(GPS.lastNMEA());
  NMEA2=GPS.lastNMEA();


  if (GPS.fix == 1) {
    Serial.print("Latitude: ");
    Serial.print(GPS.latitude, 4);
    Serial.println(GPS.lat);
    Serial.print("Longitude:");
    Serial.print(GPS.longitude, 4);
    Serial.println(GPS.lon);
    Serial.print("Altitude:");
    Serial.println(GPS.altitude);
    Serial.println("");

  digitalWrite(77, HIGH);
  delay(100);
  digitalWrite(77, LOW);
  }
  else {
    Serial.println(NMEA1);
    Serial.println(NMEA2);
    Serial.println("");

  digitalWrite(77, HIGH);
  delay(100);
  digitalWrite(77, LOW);
  }
}

void clearGPS() {

  while(!GPS.newNMEAreceived()) {
    c=GPS.read();
  }

  GPS.parse(GPS.lastNMEA());

  while(!GPS.newNMEAreceived()) {
    c=GPS.read();
  }

  GPS.parse(GPS.lastNMEA());

  while(!GPS.newNMEAreceived()) {
    c=GPS.read();
  }

  GPS.parse(GPS.lastNMEA());
}
```

Appendix H: Final Beacon Code

```
#define LED GREEN_LED

void setup() {
Serial.begin(9600);
Serial1.begin(9600);
pinMode(LED, OUTPUT);
delay(1000);
}

void loop() {
  digitalWrite(LED,LOW);
  readXBee1();
}

void readXBee1() {
  if (Serial1.available() > 0) {
    digitalWrite(LED,HIGH);
    Serial.write(Serial1.read());
  }
}
```

Appendix I: Final Base Station

```
#define LED1 BLUE_LED

void setup() {
Serial.begin(9600);
Serial1.begin(9600);
pinMode(78, OUTPUT);
pinMode(LED1, OUTPUT);
delay(1000);
}

void loop() {
  digitalWrite(78, HIGH);
  digitalWrite(LED1, LOW);
  readXBee2();
}
```

```
void readXBee2() {
  if (Serial1.available() > 0) {
    digitalWrite(LED1,HIGH);
    Serial.write(Serial1.read);
```