Project Code: MQP SJB 4D17

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

Throughout the past few years, hikers have been more at risk at being injured or lost during their visit on the trails of both national and state parks. With the parks being located in an area where there are few to no cellular towers there is no way of communicating to park rangers or other emergency personnel. The goal of the Major Qualifying Project was to design a system that would make hiking safer by providing a way to indicate to park staff that an emergency had occurred. By providing the hikers with a handheld device that when activated will transmit important GPS data to park staff without the use of a designated cellular or wifi network it is ensured that safety is increased for all visitors.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Problem Statement

The National Park system encompasses 417 national parks in the United States. They span across more than 84 million acres in each state extended into the 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 stayed stagnant. While this shows that the crimes are not increasing this also shows that there is not preventative actions being taken. This can be seen in following table, which was derived from the Federal Agencies website and portrays two years of crimes from 2014 to 2016.

Table 1.1: 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

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.

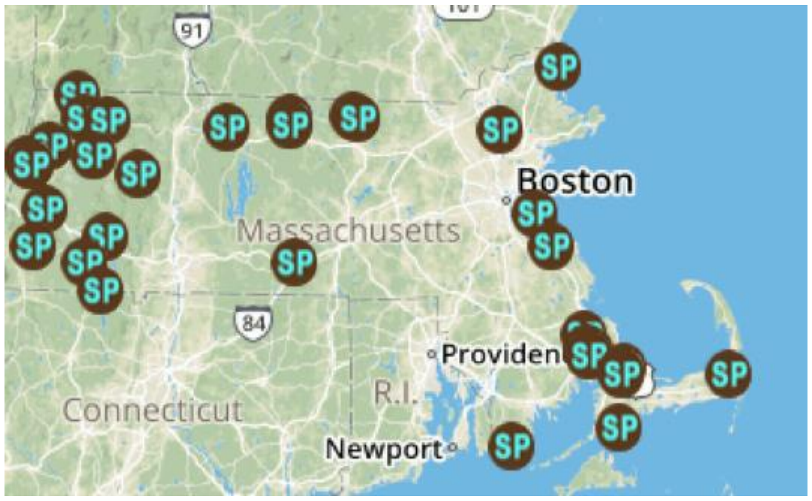


Figure 1.1: State Parks located in Massachusetts

Shown below is also the total crime in Forest Parks that occur compared to the crime that occurs in Massachusetts and on a National level.

Table 1.2: 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

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.

The Solution

As visitors of State and National Parks, the group has been made aware of safety tips, preventative course of actions and recommended training by websites and by the park service employees. No other safety measures are taken to ensure the safety of the visitors. It for this reason that 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.

From early research, it was found that the preventions taken for visitors of State and National Parks to prevent injury and crimes committed against them was less than desirable. The system designed is a solution based around the idea of providing visitors of State and National Parks with simple remote access to park service emergency personnel. The remote access device can be rented from the park and used to alert park service emergency personnel when an emergency situation arises with a press of a button. When 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.

The system can be broken down into the following parts: a Bracelet, a Beacon that retrieves and provides information to 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 Bracelet 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 Bracelets 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.

To meet the requirements of the system, numerous different technologies were necessary. The Bracelet contains GPS technology, and therefore also contains a low power microcontroller to interpret GPS data coming in as well as all of the data being sent out to the network. The Bracelet, the Beacon and the Base station use wireless communication technology to create a network for sending and receiving packets of information. In order to accomplish these goals, we looked into technologies such Bluetooth, Near Field Communication and alternative energy sources. Also researched were what other companies were making similar products. Through research it was identified that a product like this product would be new and useful since no other system would be as user-friendly, modular or inexpensive.

Chapter 2: Research

Current State of the Art

A number of other companies are working on similar solutions to this problem. Since there are no system designed primarily for keeping visitors of National and State Parks safe we had to look at systems for school campuses and systems that had the similar goal of enhancing the safety of the user.

The first system that will be looked at are based around existing emergency tower systems, similar to the towers located on the WPI campus.

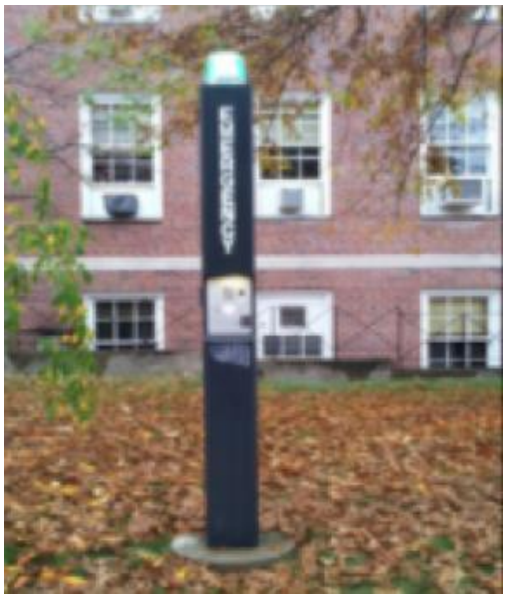


Figure 2.1: An Emergency Tower on WPI Campus

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 test purposes. Code Blue targets university, corporate and medical campuses as its markets. 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 of emergency cases but lacks the ability to let the students communicate with campus security. In the case of the visitors of the parks, they would be notified of wildfires, or possible dangers in the park but would not be able to communicate if they were in danger.

Figure 2.2: RAVENalert keychain

There is a patent for a disaster alert system that has been developed. This device includes a radio receiver and a processor programmed to monitor radio transmission from one or more central stations for disaster alerts directed to the location of the disaster alert device. The alert device will include an audio unit to alert personnel located at the site of the device to the precise nature of the disaster. The disaster alert devices are pre-programmed with information identifying the precise use location of the warning device. This use location information includes latitude and longitude of the use location and may also include other location information such as street address and zip code. Warnings are broadcasted from central stations identifying with latitude and longitude at-risk regions. To minimize required battery power the devices are programmed to listen for a warning for only very short periods of time such as one second each five minutes. The awake periods are preferably the same for all battery powered devices located in relatively large contiguous regions. The central stations that broadcast warnings are aware of the awake times, and the central stations are programmed to broadcast warnings to those devices during an awake period. Timing components in the disaster alert devices keep them synchronized with computers at the central stations. Each central station is equipped with a computer system with digital maps having latitude and longitude overlays so that at-risk regions can be specified. Disaster alert devices within the radio audience of the central station radio are awake during the broadcast and receive the header information. The header information is analyzed by the disaster alert devices and compared with their preprogrammed latitude and longitude positions. If they are outside the at risk region, they go back to sleep. If they are within the at risk region, they respond by recording the warning and instruction, sound an alarm, and audibly broadcast the warning and instructions. Mobile disaster alert devices incorporating a GPS device may be made available for mobile vehicle such as boats, cars and trucks. Each of these devices compare its actual latitude and longitude with the latitude and longitude information broadcast by the central station to determine if the device is in an at risk region. These mobile alert warning systems can also be incorporated in electronic devices that people typically carry around such as laptop computers and cell phones. These devices can get their GPS position from an incorporated GPS device or other sources. This is a similar concept to that of the RAVENalert. The user can receive where the disaster is occurring but the user cannot transmit their information.

Another device we looked at is called Life Alert, which is primarily used to help elderly citizens alert emergency personnel if they are in danger. The Life Alert device operates by using two separately encased and remotely positioned components. The first component comprises 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 usually communication with the land telephone line of the building in which 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 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.

Figure 2.3: Life Alert

System and Component Research

Communication

To implement this idea, there need to be a wireless network between the Bracelet, Beacon and Base station. For this project, different wireless network were researched. The first wireless network considered was Wifi.

Figure 2.4: Wifi

Some positives of Wifi are that it offers the use of a very secure network with a long range of transmission and even supports fast transmission speeds which allow more info to be transmitted at a time (250Mbs+). A con of using Wifi as the wireless network 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 this device being used where there isn’t easily accessible electricity this would be a problem. Another con is the applications use in the parks, we would need to construct a cell tower in various places of the parks which would interfere with normal wildlife activity and also have high costs. Lastly, Wifi being easily accessible in the parks would have an incentive for hikers to use their phones but would create noise pollution and would antagonize those who are trying to get away from the distractions that cellphones cause.

With Wifi ruled out we weighed the pros and cons of using Bluetooth technology, which is better suited for battery powered applications as they use 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 to this project and uses less power than other wireless technologies [4]. Although it is not as secure or as stable as Wifi it is a good substitute as it allows for peer to peer communication. The major downfall of Bluetooth was its max usable range of about 100 meters.

Figure 2.5: Bluetooth

With both Wifi and Bluetooth ruled both having major cons in implementation and range respectively, a better suited technology was needed. The XBee’s use the Zigbee protocol which allows the creation of personal area networks that allows wireless communication between devices. The XBees were a halfway point between Wifi and Bluetooth. They were low power devices that could be put in a battery powered devices and had a longer usable range. Some of the higher level XBees had large ranges (15 and 40mi) that could even be increased with signal repeaters.

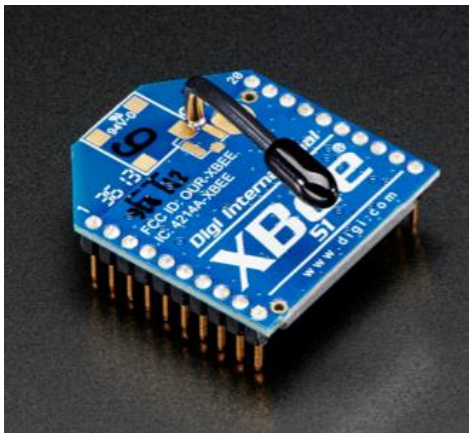


Figure 2.6: Xbee

The two pairs of XBees that were chosen were the Series 1 (100m) and Series 1 Pro (1Mi) they used little power (could be controlled by a micro controller) 1mW and 60mW during transmission respectively and still adhered to FCC laws when using high frequency applications.

Microcontrollers

When looking for microcontrollers the standard Arduino Uno was looked at. Arduino are used often and have a lot of examples and forums dedicated to it allowing for new users to program it. It seemed like a good choice but upon further investigation this controller uses quite a bit of power (45mA) so it would drain a battery fairly quickly. Lastly from the specifications it can be see that this controller can only support one universal asynchronous receiver transmitter (UART) device so we would only be able to use either the GPS or XBee with it not both.

Figure 2.7: Arduino Uno

To see if the different Arduinos varied enough to be useful we checked out the Arduino Mini. It 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 because of its small size additional equipment needs to be purchased to actually program the controller as it doesn't have a USB port. Also just as the Arduino Uno it lacks the capability to interface with more than one UART device making it essentially useless for our intended design.

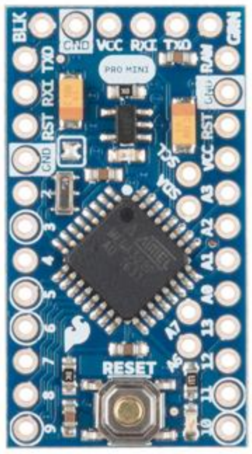


Figure 2.8: Arduino Pro Mini

The MSP430 is a low power microcontroller from TI. This controller was looked into because both students had previous experience with this device from an earlier ECE class. We knew we need at minimum two UART ports. The MSP430 has 4 ports that allow for 4 different peripherals to be controlled. Further looking at the specifications even in its most active powered mode it essentially used no power as it consumed a couple hundred micro amps (404 micro amps per MHz) which will allow our device to be active a lot longer. Also the controller had voltage out pins that could be used to power its peripherals which is a plus so the system will be self-contained.

Figure 2.9: MSP430

The MSP432 is a newer more advanced version of the MSP432 that has a 32bit processor. It has all the functions of the MSP432 but uses even less power (80 micro amps per MHz) although it is slightly more expensive. This controller was chosen because it still had all the functions necessary but was readily available from a friend and didn't require us to purchase anything else.

Figure 2.10: MSP432

Table 2.1: 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

GPS

The AdaFruit FeatherWing Ultimate GPS was the first GPS we researched which we ended up using. Since it has 66 channels it has high accuracy in determining where it is within a 3m radius. It even updates its position from 1 to 10 times a second although a lower refresh (1Hz) has better stability. 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 which it does. This specific device sends data in the NMEA 0183 format at 9600 BAUD (rate information is transferred). NMEA stands for National Marine Electronics Association and is used for electrical communication in marine electronics.

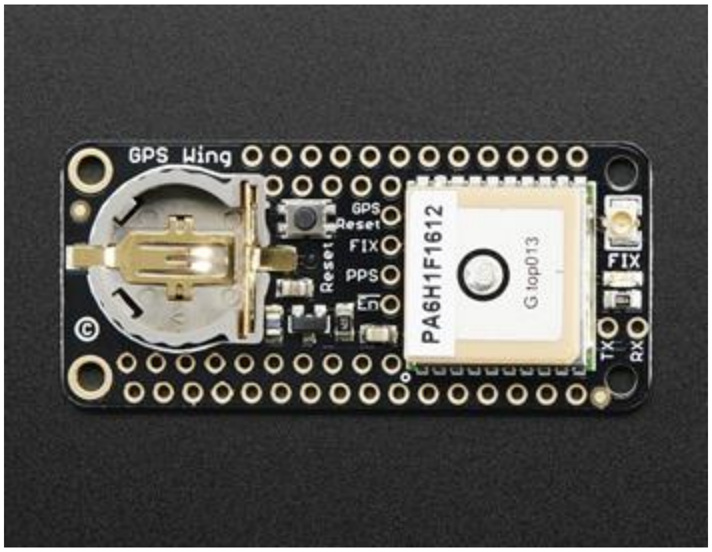


Figure 2.11: AdaFruit Ultimate GPS

The AdaFruit FeatherWing Ultimate GPS Breakout is essentially the same as the AdaFruit FeatherWing Ultimate GPS above but is much smaller in size and removes all the extra stuff leaving just the bare essentials 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.

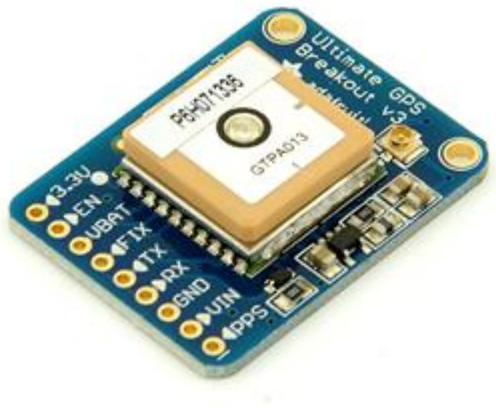


Figure 2.12: AdaFruit Ultimate GPS Breakout

Chapter 3: Design

When looking at previous design art we knew we would need at least three portions to our total design. For the user they would need some sort of device that they could use to send an alert (a watch, necklace etc.) we call this the trigger. Then the trigger would need to talk to an intermediate device which we'll call the beacon. This beacon needs to receive the data from the trigger and then relay it to the base station (the third device) where there would be people who could then act upon the trigger's notifications.

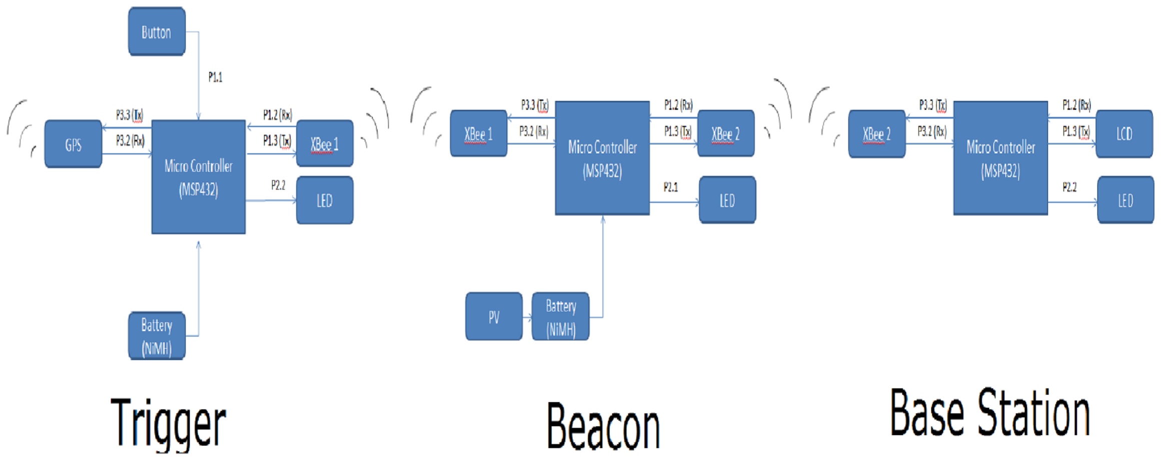


Figure 3.1: Full Device Block Diagram

Trigger

For the trigger it needed to be portable so that the hiker could take it with them this means that the trigger needs to be battery powered for an unknown amount of time. With this portion of the device using batteries we have to minimize the current draw until the hiker actively needs it.

This led us to the conclusion that we should be able to control all the devices power consumption using a microcontroller as they turn on devices when necessary and turn them off afterwards. Using the controller (MSP432) will also let us control the data that enters and exits the trigger. In order to send the GPS coordinates of the hiker to the Beacon a GPS module and communication device (XBee) needed to be connected to the MSP432 using UART channels. These channels let the GPS transmit its data into the controller so they can then be transmitted to the XBee.

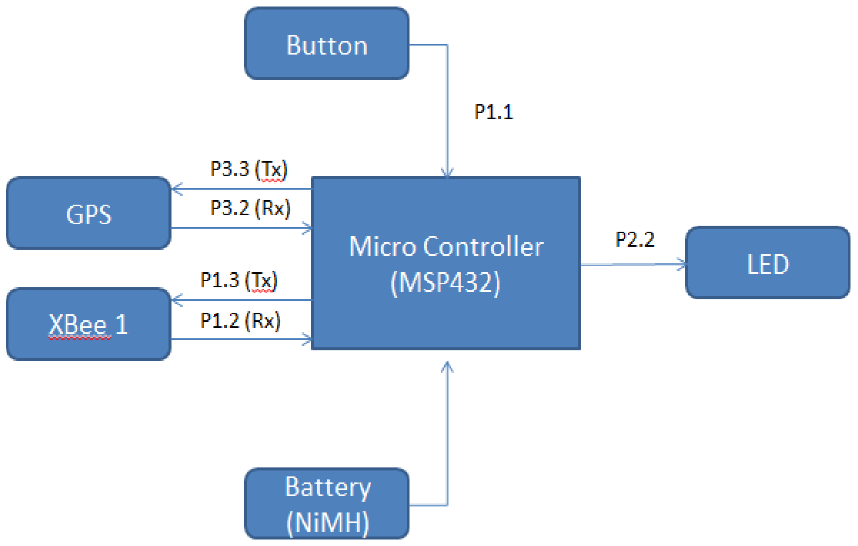


Figure 3.2: Trigger Block Diagram

Beacon

The beacon, our intermediate device that will receive coordinates from the trigger and then send them to the base station will need a power source as well. Since these are supposed to be scattered around the park they need to be self-sufficient and require little maintenance. To do this we decided to use a renewable resource and a battery to keep the beacons up and running. The easiest and most abundant resource we could use was solar energy so we would invest in a solar panel and a large battery. Following the same basic structure as the trigger the MSP432 should have two peripherals attached to it but instead of it being a GPS and an XBee it will be two XBees of different ranges. One XBee to communicate with the trigger the other to reach and communicate with the farther base station.

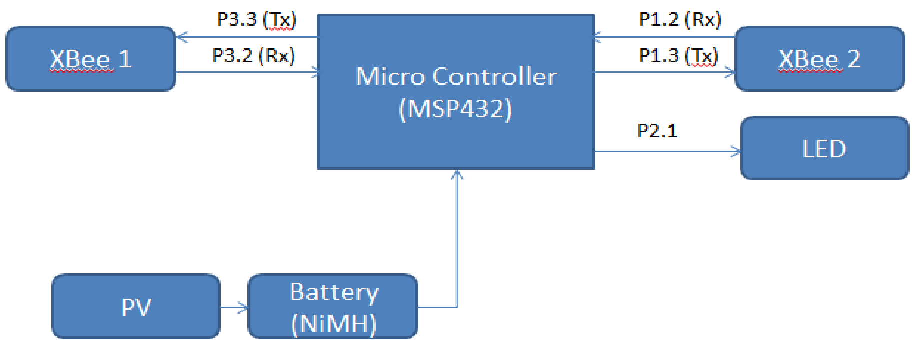


Figure 3.3: Beacon Block Diagram

Base Station

The last portion of our design is to receive the GPS data from the beacon (originally from the trigger) and then display it for park staff to see. Like the trigger and beacon before this part we will use a microcontroller as well, to control and distribute the incoming data. Although this time the MSP432 will be connected to an XBee (for communicating with the beacon) and an LCD from TI. We will ignore the power supply for this portion as it is assumed this part of the device will be stationed in place where it can use an outlet or something since it will be stationary.

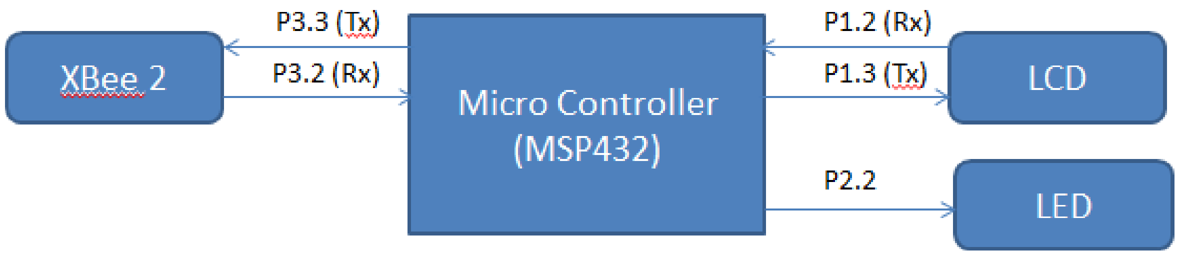


Figure 3.4: Base Station Block Diagram

Chapter 4: Methodology and Implementation

Our project's goal is to transmit accurate GPS data to a base station that will notify park staff that a hiker is experiencing an emergency at some location and needs assistance. This portion of the report demonstrates how we integrated our hardware and software components to create our final system.

Trigger

While going through the datasheet of the MSP432 we located the different UART ports that we could use. The MSP432 Launchpad has four UART ports but we only needed two so we 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 for the exchanging of information. For simplicity the peripheral devices (GPS and XBee) only required 3V to operate and had voltage regulators to help handle being powered so we were able to power them up with the MSP432 controller's voltage output pins. This would let us power only the MSP with batteries while everything else could run off of the controller.

With this general setup we could then start figuring out the necessary requirements for the trigger's battery. We assumed that all devices excluding the controller will be off until activated by the hiker but until then the MSP432 will be in low power mode waiting for the trigger's button input. With the MSP in low power mode it essentially usually uses no current (a few micro amps) so just about any battery will last for years at a time. When the hiker does press the input button the controller (0.625mA) will enter active mode then the GPS (25mA), XBee (45mA) and LED (20mA, just to show info is being sent) will power up and current will start to be drawn. The total max current drawn per hour will be 90.625mA.

Table 4.1: Trigger Power Table

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

As a security blanket we wanted the trigger to 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 trigger's location.

Beacon

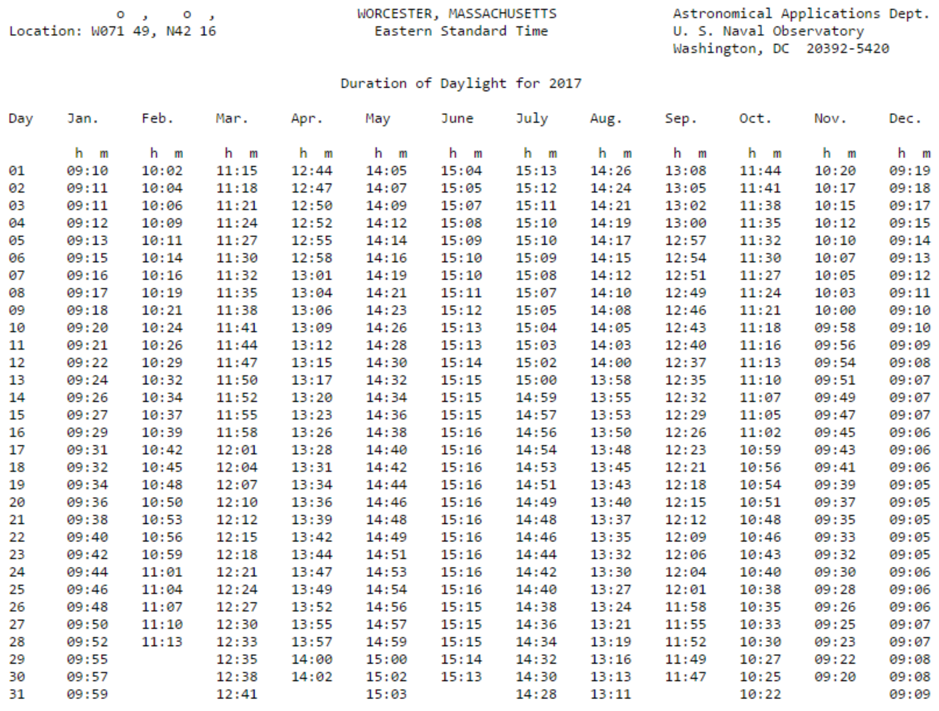
For our beacon we decided to use the same UART setup as the trigger using port 2's receive/transmit pins P2.2/P2.3 respectively as well as port 3's receive/transmit pins P3.2/P3.3 respectively. The two XBee peripherals would then be connected to these pins for the exchanging of information. For simplicity the peripheral devices (XBee1 and XBee2) only required 3V to operate and had voltage regulators to help handle being powered so we were able to power them up with the MSP432 controller's voltage output pins.

With this setup we could still keep most of the beacon powered down having only the controller and XBee1 (needs to actively listen for the trigger's transmission) on. 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 trigger as XBee2 uses five times more current to transmit (since it transmits a farther distance), also XBee1 needs to remain on constantly waiting for a transmission. This device being run 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 13.3 hours if we use a 5Ah battery. Although if no transmission is activated and only the controller and XBee1 are powered the device can run for 98.8 hours without a charge.

Table 4.2: Beacon Power Table

Device	Voltage	Current Draw
MSP432	3V	0.625mA
XBee1	3.3V	50mA
XBee2	3.3V	305mA
LED	1.8V	20mA

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 that keeps track of the amount of daylight each day of the year has shown that 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 use daily (assuming the panel is not covered) especially from May to September which is when most people visit parks and hiking trails.



```
          o  ,     o  ,                  WORCESTER, MASSACHUSETTS           Astronomical Applications Dept.
Location: W071 49, N42 16                  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 4.1: Daylight Hours 2017 (Worcester, Ma)

Base Station

For our base station we decided to use the same UART setup as the trigger and beacon 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 XBee and LCD peripherals would then be connected to these pins for the exchanging of information. For simplicity the peripheral devices (XBee2 and LCD) only required 3V to operate and had voltage regulators to help handle being powered so we were able to power them up with the MSP432 controller’s voltage output pins.

Since we are assuming the last stage of our project will be stationary and have a dedicated power source we can ignore the power requirements. All this portion of the device has to do is listen for incoming data and when it receives something transmit it to the MSP432 microcontroller to be sent to the LCD where park staff can see the location of the person in distress.

Setup

For our complete system at first we thought we should have the beacons deployed along the hiking trails much like how street lights are spaced along busy streets as shown below in Figure 4.2. This would reduce the amount of beacons needed and we could space them in a manner where the hiker’s trigger is always in range of at least one beacon. The beacons would then be in range of the base station to communicate there is an emergency.

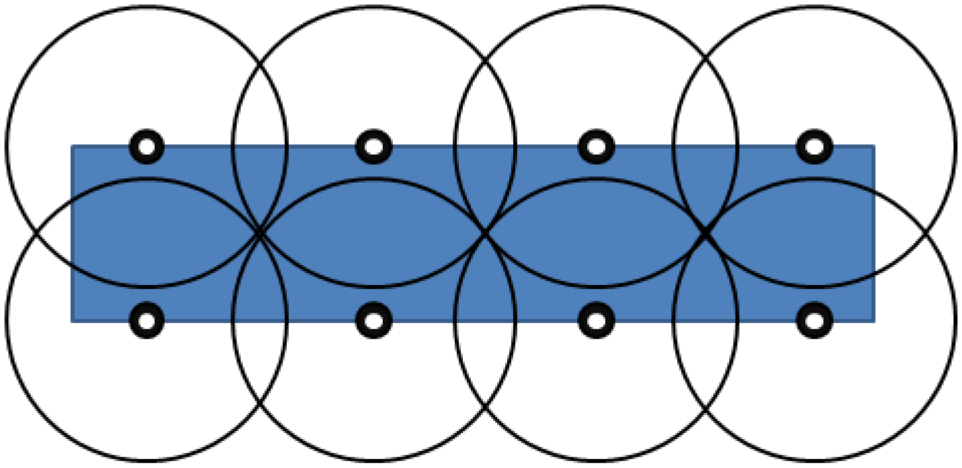


Figure 4.2: Trail Setup

The black outlined circles indicate the range (100m) where the trigger’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 we were getting further into the project we started to notice 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 because there was no light. They would not be in a position to activate their trigger and have the signal be received by a beacon, a major flaw in how we were planning to implement the system.

To combat these holes we came up with a solution shown in Figure 4.3 below. This lets us encompass the entire park in beacons 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 way would cost more initially but would lower cost for actually getting the hikers help as the park staff would know exactly where to go and start their search. It would also increase the chances of finding the hikers safely, instead of waiting for someone to notice a hiker has been gone for days and starting a blind search reducing the hiker's survival.

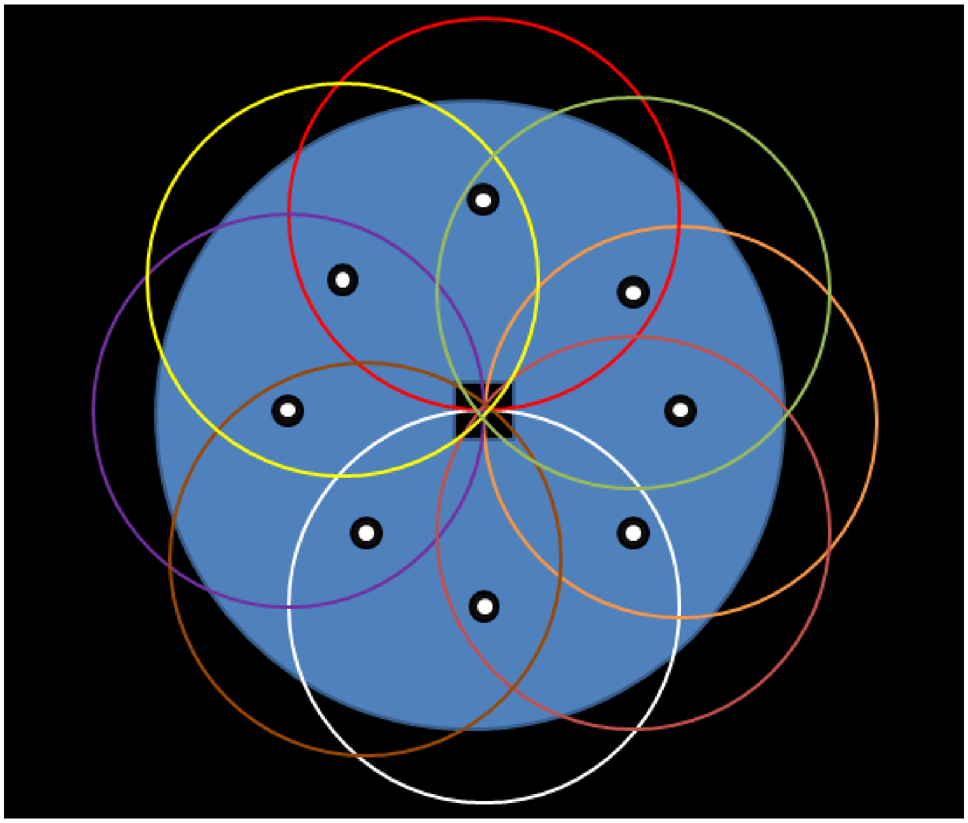


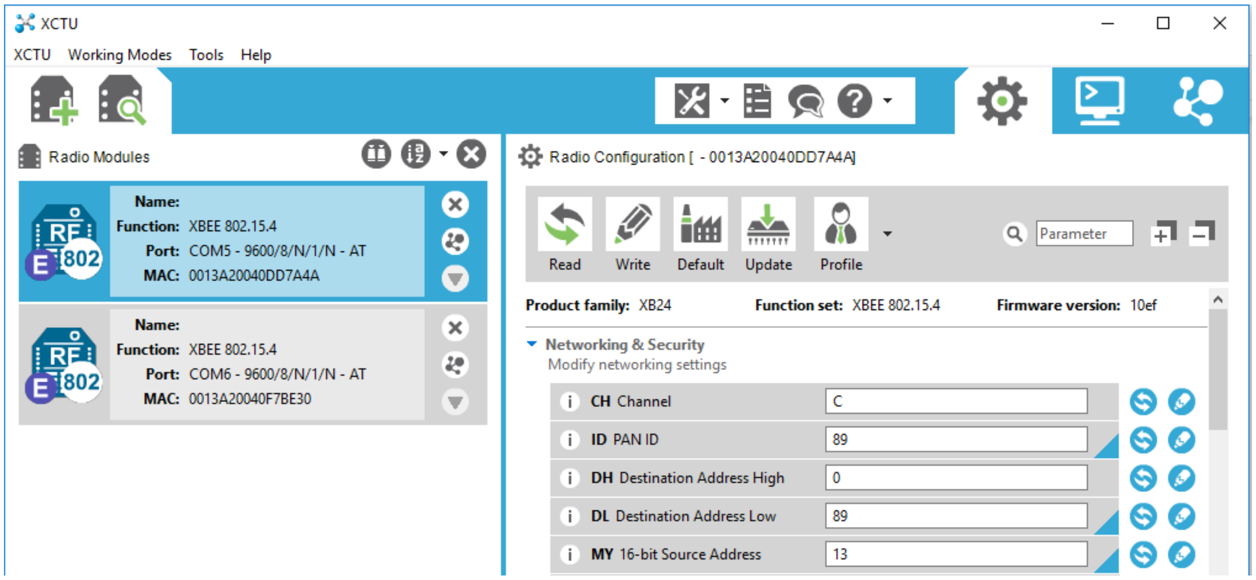
Figure 4.3: Park Setup

The black outlined circles indicate the range (100m) where the trigger'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.

Chapter 5: Testing and Results

Once we had all of our hardware we had to run some tests on our equipment to ensure that all our equipment was working.

We started with the XBees first as one pair had come from a friend who had all the necessary pieces (USB and Breadboard docks). To make sure the pair of Xbee1s could talk each other we downloaded Digikey's XCTU program. This let us configure each XBee so they could communicate. We wanted to make the transmission process easy so we made it so that one Xbee1 would act only as a transmitter while the other XBee1 acted only as a receiver. We were able to do this by changing the address of the receiving XBee1 to 89, then we had the transmitting XBee1 start sending its low byte to address 89 (receiving XBee1). The different XBee1 setups are shown in Figures 5.1 and 5.2 below. The channel and PAN ID are the same for both XBee1s so that they are in the same network allowing them to communicate. Although the channel and ID are the same they are unique so that there is almost no chance that anyone else using XBees will be operating in our network let alone send to the proper addresses causing interference.


XCTU
XCTU Working Modes Tools Help
Radio Modules
Read Write Default Update Profile
Parameter
Networking & Security
Modify networking settings

Radio Configuration [- 0013A20040DD7A4A]

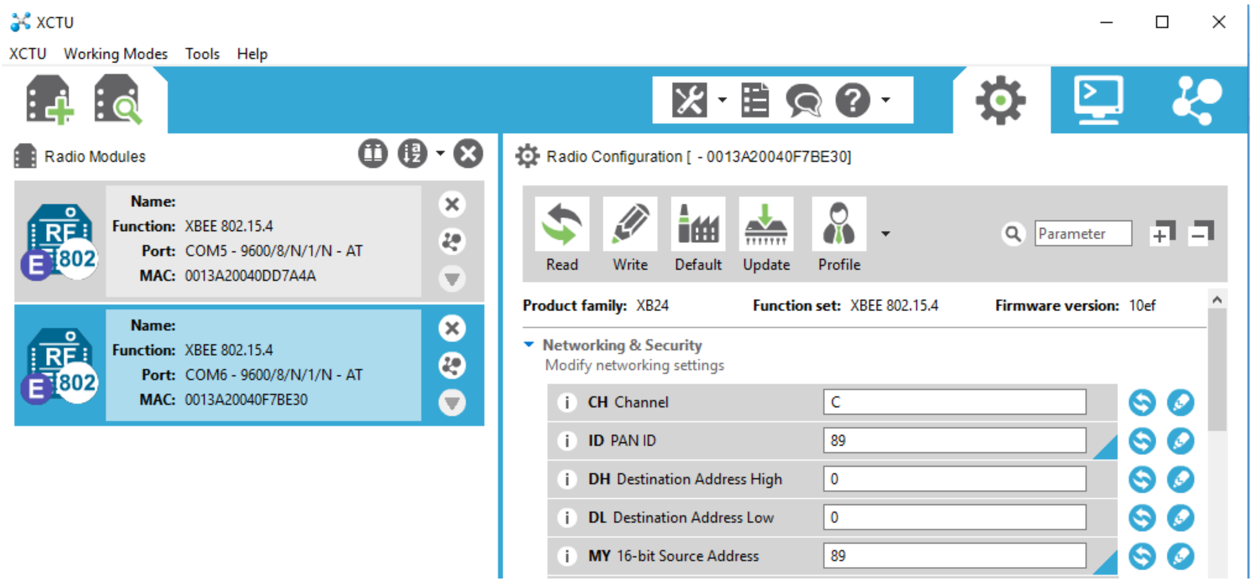
Name:	
Function:	XBEE 802.15.4
Port:	COM5 - 9600/8/N/1/N - AT
MAC:	0013A20040DD7A4A

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 5.1: Tx XBee1 Setup


XCTU
XCTU Working Modes Tools Help
Radio Modules
Radio Configuration [- 0013A20040F7BE30]
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

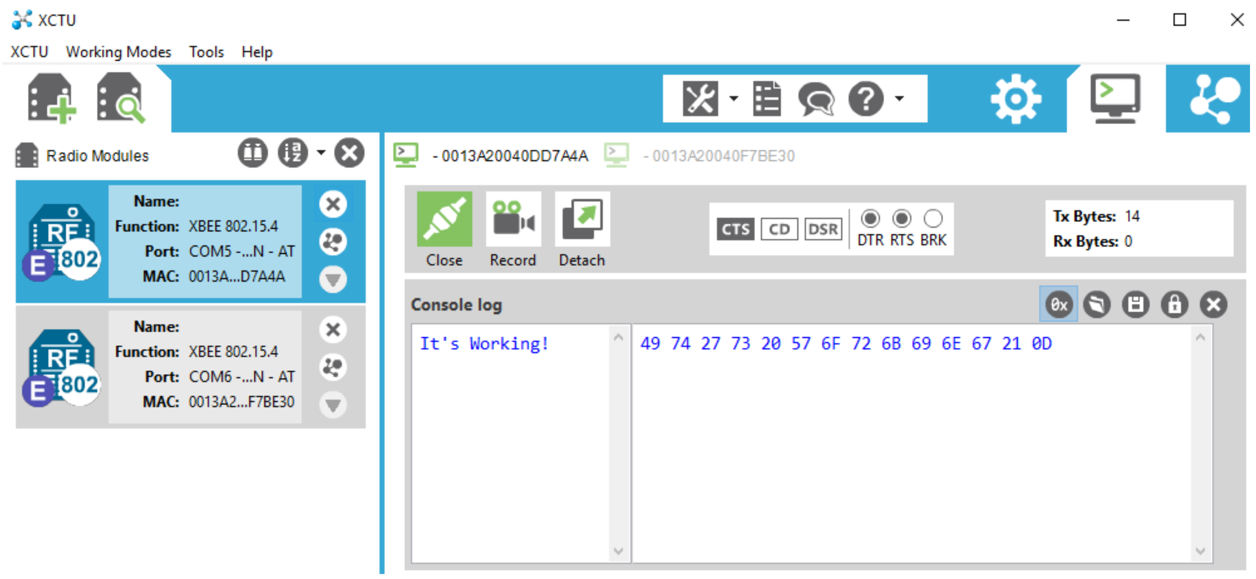
Product family: XB24 Function set: XBEE 802.15.4 Firmware version: 10ef

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

Figure 5.2: Rx XBee1 Setup

With both XBee1s setup we could then test the connection by switching the XCTU program from setup mode to console mode. In this mode we could write in the console log and see what was being 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 (Figure 5.3). We then clicked over to our receiving Xbee1 (Figure 5.4) to see if the statement was received which it was as “It’s Working!” was written in red. Just to check if the receiving XBee1 could send to our 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 how we liked, we moved onto configuring and testing the longer ranged pair of XBee2s. We went through the same process as before with the XBee1s but changed the PAN ID to 13 as precaution even though the XBee2s and XBee1s shouldn’t be able to communicate anyways based on the implemented zigbee protocol.



Name:
Function: XBEE 802.15.4
Port: COM5 -...N - AT
MAC: 0013A...D7A4A

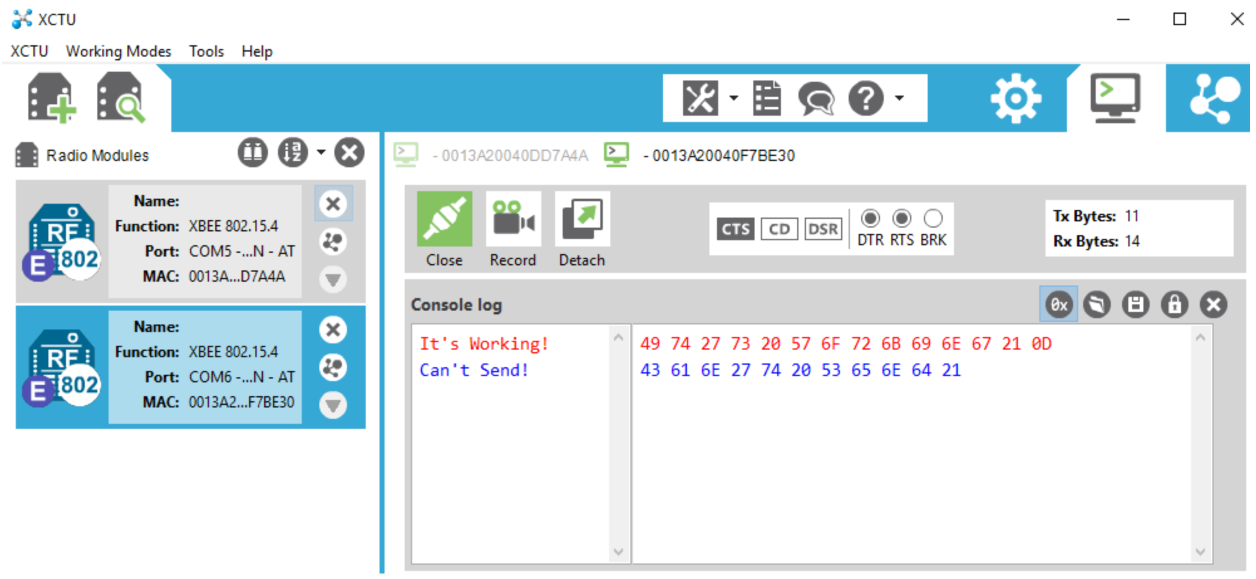
Name:
Function: XBEE 802.15.4
Port: COM6 -...N - AT
MAC: 0013A2...F7BE30

- 0013A20040DD7A4A - 0013A20040F7BE30

Tx Bytes: 14
Rx Bytes: 0

```
It's Working!        49 74 27 73 20 57 6F 72 6B 69 6E 67 21 0D
```

Figure 5.3: Tx XBee1 Test



Name:
Function: XBEE 802.15.4
Port: COM5 -...N - AT
MAC: 0013A...D7A4A

Name:
Function: XBEE 802.15.4
Port: COM6 -...N - AT
MAC: 0013A2...F7BE30

- 0013A20040DD7A4A - 0013A20040F7BE30

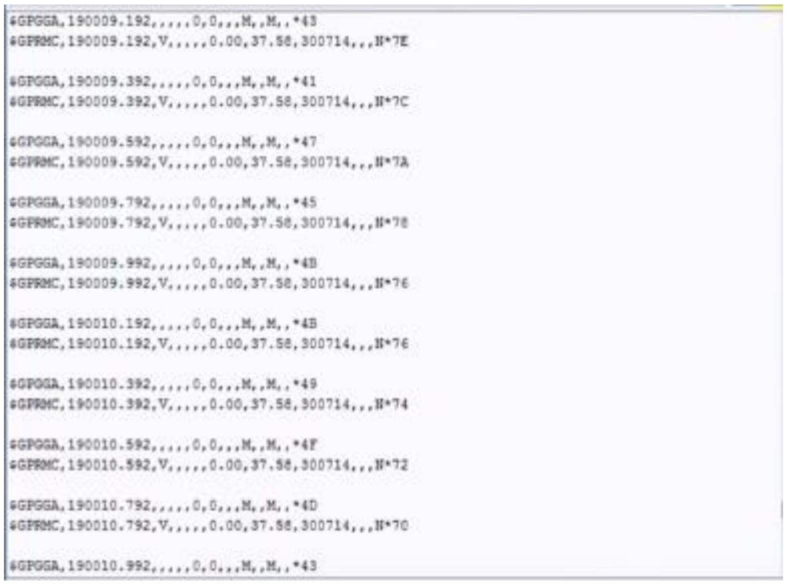
Tx Bytes: 11
Rx Bytes: 14

```
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 5.4: Rx XBee1 Test

With both pairs of XBees functioning as expected we progressed onto testing the three microcontrollers in order to do this we downloaded the Energia Integrated Development Environment (IDE) and all the associated drivers for our version of the MSP432. With the IDE fully updated we downloaded the simple blink LED program to each controller to test if they all worked, which they did. The program made an RGB LED flash red once a second. To test some more sophisticated parts of the microcontroller we ran a program that let us send data using UART to the computer so we knew we had all the functions we needed from the different MSP432s.

For testing our GPS the AdaFruit people had created a driver library for our GPS so all we had to do was download and install the library into Energia. With this library we were able to connect the GPS to the UART pins in port 3. The GPS starts sending data automatically once it is powered up so we just sent the data coming into the computer's serial port to see the raw GPS data. The NMEA 0183 format sends a lot of raw data so we edited the program (Appendix A "GPSTest") using the AdaFruit library so we could parse through the necessary info and send only what was necessary. In Figure 5.5 below the parsed GPS data is shown, the initial test occurred indoors so this is a general position. However if we were outside and got a fix on a satellite then specific coordinates 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 5.5: Serial Monitor Results

With a working GPS we then turned to seeing if we could print to our LCD. First we used an LCD example (Appendix A "LCD Example") already created by Energia to confirm we 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.

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 couldn't see what was being sent and/or received. So 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 which meant we then couldn't know if the data from the trigger would even make it to the beacon.

This grew to be a problem so we deceided to switch to Code Composer Studio (CCS) where it was slightly more difficult to use but gave us more features to use. This meant we had to start from scratch and learn how to setup all the pins we wanted to use manually. This led us to download CCS version 6 and all 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. 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 trigger'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 was a bunch of garbage 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.

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 because the pins were too close together (2mm) and our two XBee breakout boards were being used by the XBee1s. We thought we may have to purchase two new breakout boards which would have taken days to get so we got a little creative and figured we could use the XBee USB board since they had a place to solder on pins. After going through the schematic of the XBee USB boards we figured we could get away with it and soldered pins the boards. We had trouble soldering such a small device since we didn't want to ruin any electrical components on the boards as we only had the two. Luckily Mr. Appleyard in the ECE shop was able to solder the pins on for us. We then replaced the XBee1s in the system with XBee 2s then made sure the hardcoded string was still sent perfectly, which it was.

Knowing the wireless components for our devices worked would make any problems that could occur easier to debug. 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 the Explorer wasn't getting any power or just the LED stopped working. We ruled out the LED just 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 they 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 now powers up correctly which 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 we wrote "Serial.print(Serial1.read());" instead of "Serial.write(Serial1.read());" which was quickly corrected. During this test were still using the hardcoded strings so we still knew what data was being sent.

The hardcoded string that we sent made it successfully from the trigger 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 just expected a general string of GPS data with no coordinates shown because the GPS can't find a satellite signal if not directly near a window while inside of Atwater Kent. Our expectations were confirmed as the necessary strings from the GPS would still be transmitted to the serial monitor.

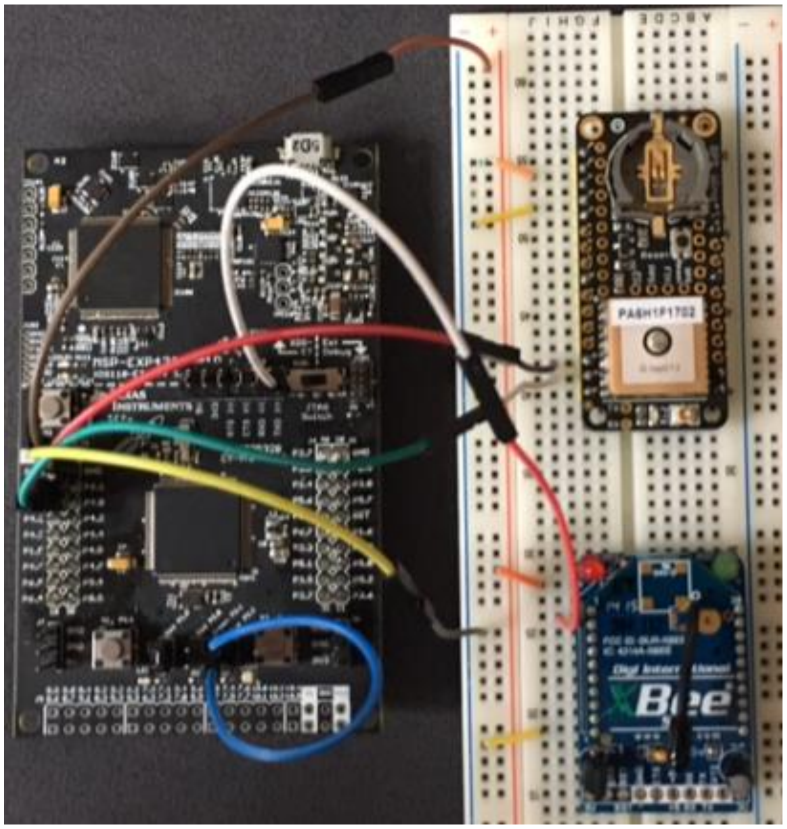


Figure 5.6: Fully Assembled Trigger Hardware

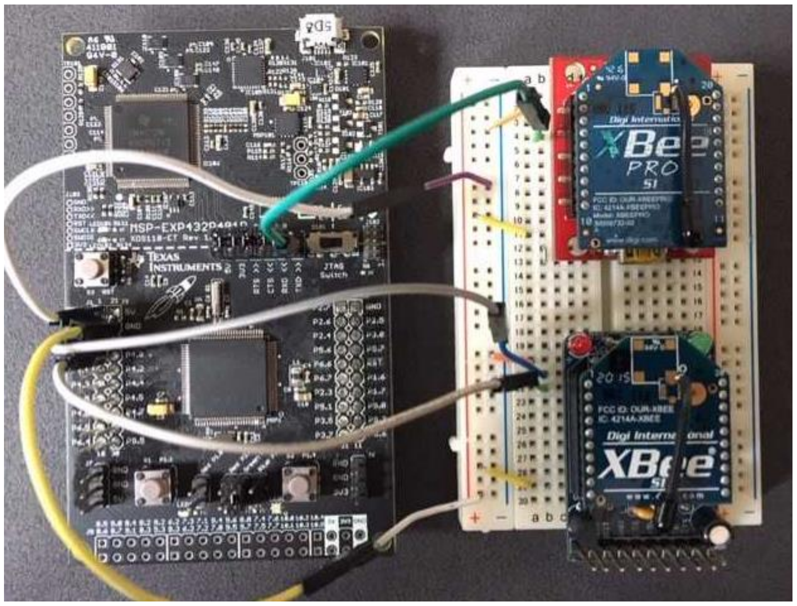


Figure 5.7: Fully Assembled Beacon Hardware

Lastly with the core of our 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 trigger are. The complete trigger code (Appendix C "Final Trigger 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.

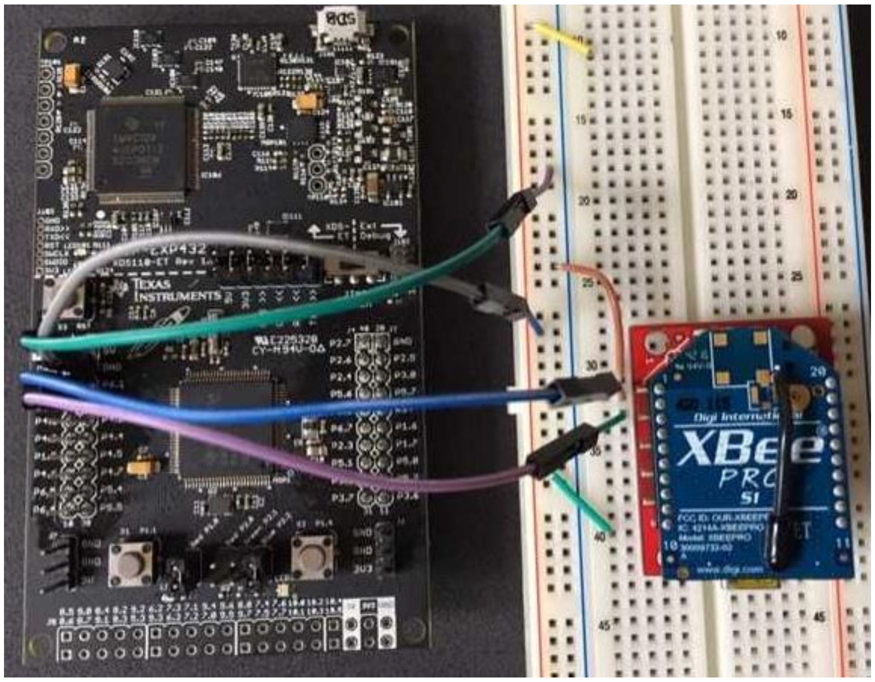
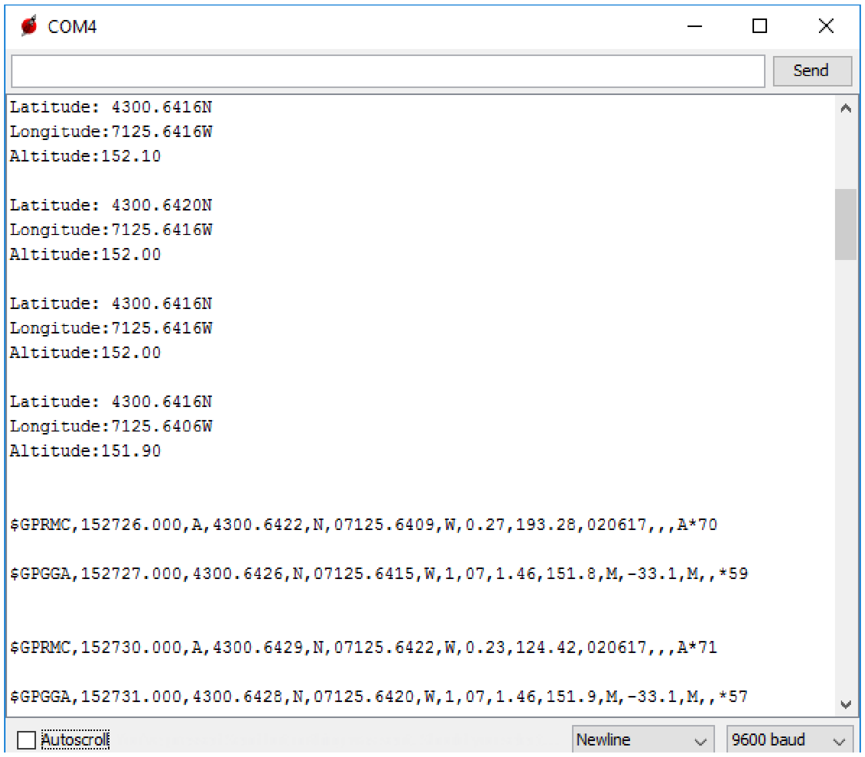


Figure 5.8: Fully Assembled Base Station Hardware

Finally with full and complete code the hardware and software were ready to be tested out in the real world. We attached the trigger 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 trigger 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 experiment with how long it took the trigger 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
```

[ ] Autoscroll | Newline | 9600 baud

Figure 5.9: Base Station Results (Top Format: With Fix, Bottom Format: Without Fix)

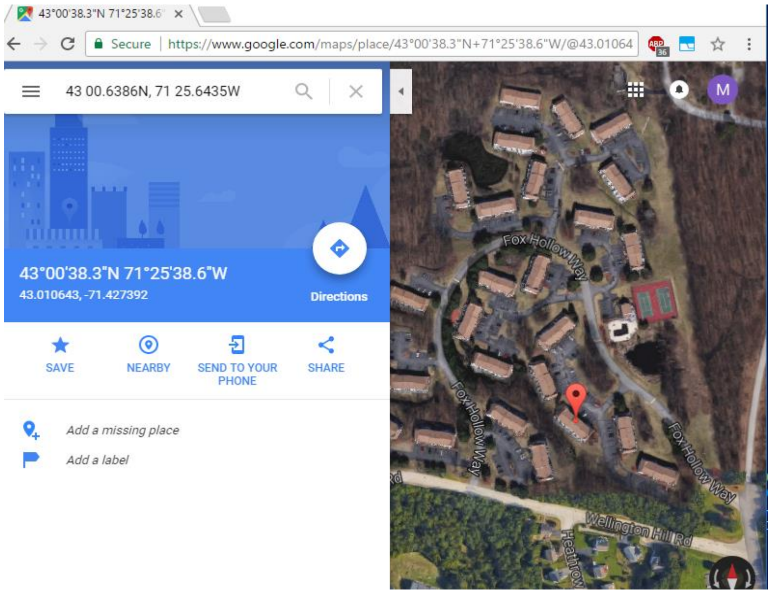


Figure 5.10: Trigger's GPS Coordinates in Google Maps

This device runs properly on micro USB power from an outlet or computer which can limit mobility of the device if you don't have a battery pack or small laptop you can easily walk around with. So, we designed power circuits for both the trigger 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 fine as 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.

For our trigger portion the power circuit was fairly straightforward we just needed to drop our batteries 3.6V 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 just need to use the right resistor ratio which can be 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 yields an R2 of 350 ohms giving us a steady 3V output that can be used to power the microcontroller and its devices for approximately 30.89 hours at full transmitting power.

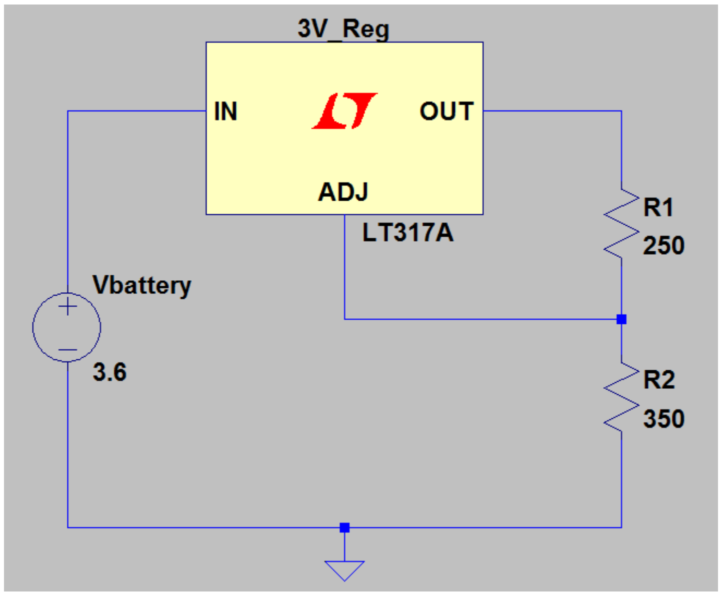


Figure 5.11: Trigger's Power Circuit

The intermediate portion of our device the beacon's power circuit was a little more complicated than the trigger's circuit as we needed to use a solar panel to recharge a battery. The first step was to filter and lower the voltage the solar panel was producing as the original waveform could hurt the battery by having to high of a voltage. In the circuit is a 0.1 microfarad filter capacitor which will reduce any ripple voltage so its output is essentially DC this is put through the 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 we are using to charge the battery. With the renewable energy portion of the circuit designed all that was needed now was to drop the battery voltage down to a usable level (so we don’t damage any electronic components) like we did with the trigger so we used another LM317 adjustable voltage regulator and the same ratio of resistors to get a 3V output for the microcontroller and its peripheral devices. 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 assuming it is not being blocked so the battery will be fully replenished everyday allowing to the device to function solely on battery power during the night.

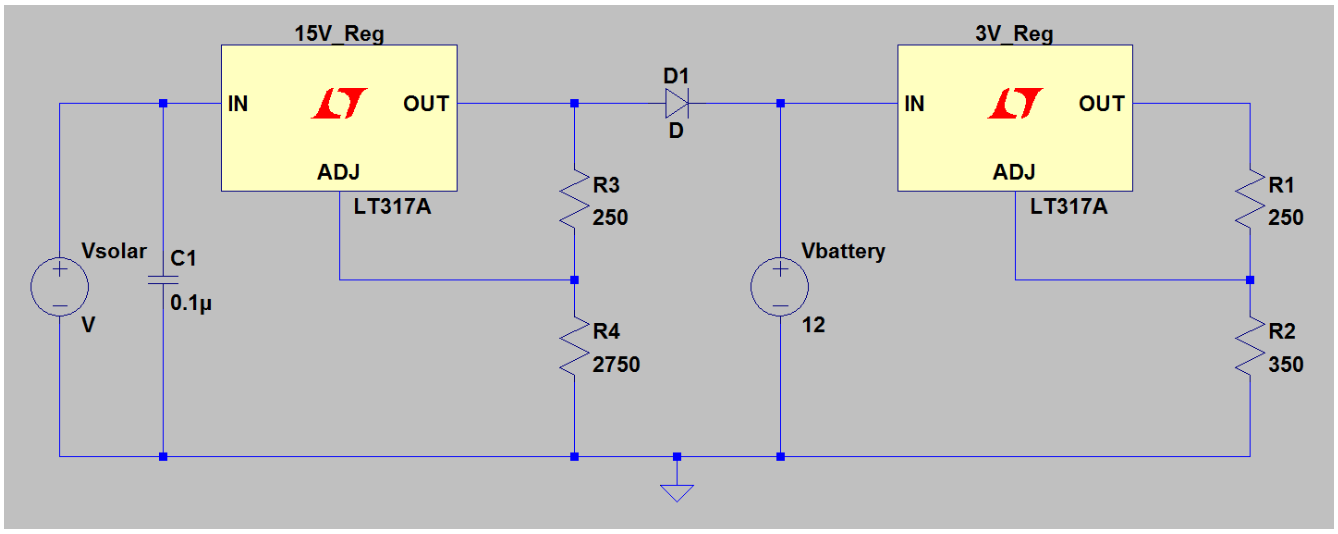


Figure 5.12: Beacon’s Power Circuit

Chapter 6: Conclusions

Overall our goal was to be able to improve hiker safety by creating a device that can send the GPS coordinates of the hiker to park staff so they can get help to that specific location. This goal was met and is able successfully receive GPS data from the AdaFruit GPS module then have them parsed for the important data and then sent from the trigger all the way to the base station and displayed on a computer screen in an easy to understand format for the park staff that can easily be entered into Google Maps to show the physical location of the trigger.

The project took about a month longer than the anticipated 7 weeks of D-Term as we faced some challenges when doing research for parts and learning how to use unfamiliar coding environments. As well as preparing for life after school while working on this MQP.

Without this device if a hiker goes missing or get hurt they have to rely on someone noticing they are gone and waiting for help or try and find their way to help. This reduces their chances at getting help in a timely manner and will increase cost and the amount of man hours required to find the missing or hurt hiker as the park staff will have an extremely broad area to search through with no guarantee they're even in the right place. This why our device is need 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 there are still some aspects of the device that can be improved. If in future works these recommendations are expanded on more this device could become more reliable and actually be tested in an actual park environment.

Chapter 7: Future Works

While going through the process of creating this hiking safety device we came up with a few suggestions that could be used to help further the development of this MQP.

First when we were initially trying to figure out the scale of our project we wanted to focus mostly on national parks because a lot of people go missing in them. So it would be nice to have a cost analysis on implementing a large scale hiking safety system in such a vast area to see if a national park would even want to spend the money to place the beacons throughout the park as well as give every visitor a trigger.

Next since we would want to experiment with such a large area it would be better if using longer ranged XBees would be beneficial for the transmission. Almost going hand in hand with the longer ranged XBees would be to see if the beacon could be constructed into a mesh network so if a person with a trigger is really far away one beacon could pick up the distress signal and pass it to another beacon that is closer to the base station and keep passing it off until the data gets to the base station itself instead of the two step transmission we did. That would allow for more efficient transmissions and theoretically increase the transmission range to as far as needed.

Lastly they could see if using XBees really are the best choice for communication. So they should look into implementing the system using Wifi, Bluetooth or etc. which could end up being a better choice for added reliability and large parks.

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>.

Appendices

Appendix A

GPSTest

```
#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());
}
```

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);
}
```

LCDTest

```
#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 B

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);
}
```

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);
}
```

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 C

Final Trigger 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());
}
```

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());
  }
}
```

Final Base Station Code

```
#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());
  }
}
```