Indoor Positioning System

EE 4820

Robert Jarvis
Arthur Mason
Kevin Thornhill
Bobby Zhang

Mentor:
Dr. Kemin Zhou

11/29/2011

Contents

1 Introduction

Wireless technology is used in our everyday lives. It can be used for tracking objects within an enclosed area. A popular wireless technology is RFID, which uses radio waves to exchange data between readers and electronic tags that are attached to any object. Xbee is another wireless technology that uses radio frequencies to transfer serial data. Most RF tracking systems only track whether the tagged object is in a specific area, for this project the RF system will locate the exact position of a tagged object indoors. Making a system convenient, easy to use and accurate are the general requirements. The device components, such as the readers and the electronic tags are small, which makes the system convenient. User friendliness will come from the RFID detection device, which will allow the user to scan a card to find the object they are looking for. Detecting the tag within 1 meter of the actual location is the accuracy is expected the indoor positioning system produce.

The Technical requirements consist of power, tracking, and time. The expected life time of the will be at least one year off battery power. The system should also be able to hold and track at least 10 tags. The last technical requirement is the system will provide real-time direction and distance to the user. The marketing requirements consist of an estimated cost of $1000, which includes four readers and two tags.

The final design review shows the progress of our work through the past year. Xbees and Arduinos are integrated together using trilateration shows how the system will find the location of the object. This review will also show how the RFID detection device is integrated into the system. Next, the battery power indicator is created to make the power last at least a year for each tag. The graphical user interface is also implemented into the system to show the user where the object is located. Tests and results for each of the systems are explained in detail through this review.

2 Hardware

2.1 Xbee Specifications:

The Xbee, or Zigbee, RF module is an embedded solution that provides wireless connectivity between devices. The modules use the Digimesh networking protocol for peer to peer networking using 2.4GHz or 900Mhz frequency bands. The Xbee is connected to an Arduino (ATmega microcontroller) that contains the programming on non-volatile memory. An Xbee mesh network can transfer data at a rate of 250kb/s at a range of up to 100 feet indoors. The Xbee module also has lower power consumption than the competing peer-to-peer communication technologies, such as Bluetooth and Wi-Fi. The Xbee has a consumption of 30mA while transmitting and 3µA at rest while the similar Bluetooth device consumes 40mA transmitting and 0.2mA while at rest. The reason for this difference in power consumption is the fact that Xbee system stays in sleep mode, like active RFID tags, most of the time. Bluetooth devices must always be transmitting or receiving and Wi-Fi devices are designed for high-power devices and not suitable for long-term battery life. The XBees in use at the conclusion of this project were using IEEE 802.15.4 (Series 1) networking protocol and also included an RPSMA connector for use with the Titanis antenna.

Figure 1: Xbee Module

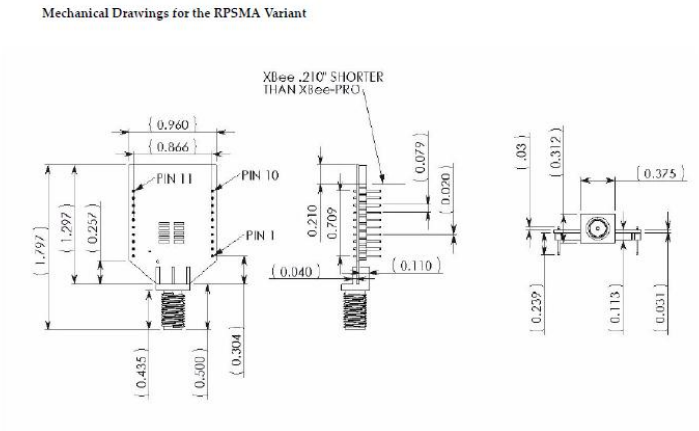


Figure 2: Mechanical Drawing of RPSMA XBee

2.2 Arduino Specifications

2.2.1 Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328. The board contains 14 digital input/output pins, 6 analog inputs, a 16 MHz oscillator, a USB connection for serial data communication, and a power jack. The Arduino Uno board will be used for the host computer connection and used for readers, or the locater nodes. For the Uno connected to the host computer, it will be able to communicate with the computer, which includes transferring data from the Arduino and uploading code to the Arduino, and powered using the USB connection. For the Uno used as locaters, power will supplied by an AC to DC converted. The board contains 31.5 KB of usable Flash memory, 2 KB of SRAM, 1 KB of EEPROM.

To simplify the connection to an Xbee module, the Arduino Uno will use an XBee shield to interface. The shield connects the serial pins (DIN and DOUT) of the Xbee to the serial pins (D0, D1) of the Arduino or to any digital pins. The board has an on-board regulator that takes 5V from the Arduino and regulates to 3.3VDC before being supplied to the XBee. The shield will also take care of level shifting on the DIN pin of the XBee. The shield also includes LEDs to indicate power and activity on the DIN, DOUT, RSSI, and DIO5 pins of the XBee.

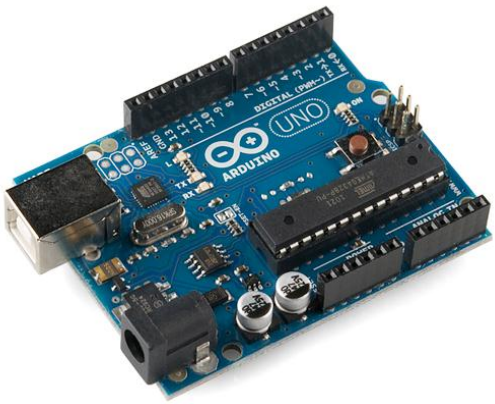


Figure 3: Arduino Uno

2.2.2 Arduino Fio

The Arduino Fio is a microcontroller board based on the ATmega328P. The board is designed to interface with an XBee module. It includes 14 digital input/output pins, 8 analog inputs, 8 MHz oscillator and an on-board resonator. The power is provided either by USB connected or by a Lithium Polymer battery. There is also an onboard battery charger to charge the battery over a USB connection. A user can upload code with an FTDI cable or wirelessly through a modified USB-to-Xbee adaptor such as XBee Explorer USB. The input voltage for operation is 3.35V – 12V. The input voltage to charge a lithium polymer battery is 3.7 – 7V.

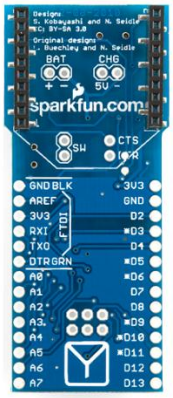


Figure 4: Arduino Fio

2.2.3 Arduino Mega

The Arduino Mega is a microcontroller board based on the ATmega1280. It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.
The ATmega1280 has 128 KB of flash memory for storing code (of which 4 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM. The ATmega1280 provides four hardware UARTs for TTL (5V) serial communication.

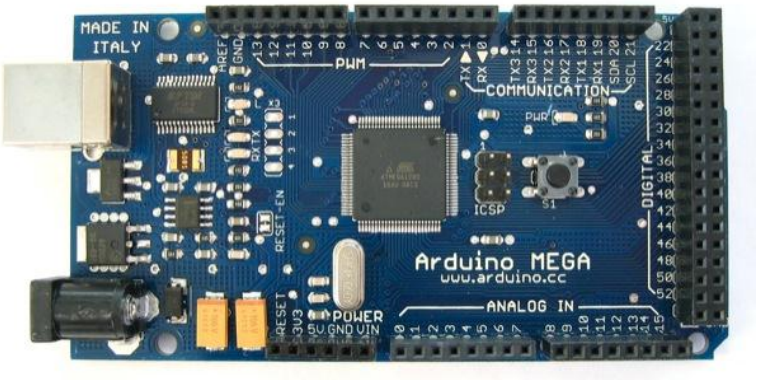


Figure 5: Arduino Mega

2.3 XBee Shield

The XBee shield allows for an Arduino board to communicate wirelessly using XBee modules, series 1 or series 2. The shield breaks out each of the XBee's pins to a through hole solder pad. Female headers can be soldered in this solder pad to simplify interfacing the Arduino with external hardware. The serial pins of the XBee are connected through an SPDT switch, which allows the user to select a connection between the DIN and DOUT of the XBee to either the UART pins, D0 and D1, or any digital pins, D2 and D3 by default. Power is supplied to the XBee by taking voltage from the 5V pin on the Arduino and regulating it to 3.3 VDC and fed into the XBee Vcc pin. The shield also takes care of level shifting on the DIN pin of the XBee. The shield also includes LEDs to indicate power and activity on the data pins, rssi pin and DIO5 pin of the XBee.

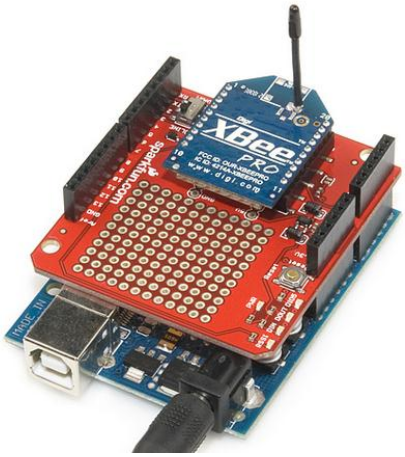


Figure 6: Stock XBee Shield Picture

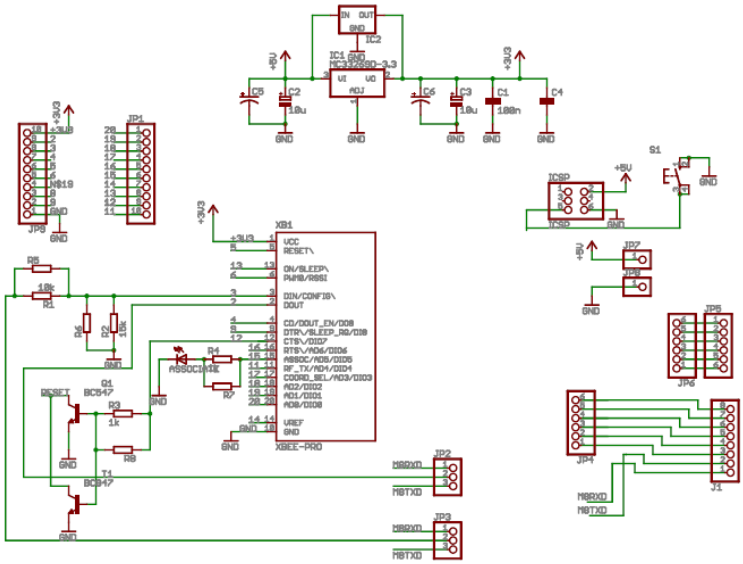


Figure 7: XBee Shield Eagle Schematic

3. Trilateration

3.1 Introduction to Trilateration

To locate an object that is in the field a system was created that uses radio frequency signals. The strength of the radio frequency signal is measured between the tagged object and the readers that are stationed in the field. Once the signal strength is gathered, it can be converted into a distance using the distance formula which will be explained in further detail later in this report. When the distances have been calculated they are plugged into a system on quadratic equations called trilateration. Using trilateration makes it is possible to find the tagged object on the XY plane and it will also allow us to find the z axis of the object as well. There are two types of trilateration, the first one is 2-D trilateration, which only allows you to find a tagged object on the XY plane and then there is 3D trilateration. 3D trilateration allows you to find a tagged object on the X, Y, and Z coordinate system.

3.2 Trilateration in 2D

For the indoor location system 2D trilateration is used to find a tagged object that is located on a surface, which will be on an XY plane. The location of three readers has to be known along with the distances between the readers and the unknown tagged object for 2D trilateration to work correctly. One can visualize this by looking at figure (x1), where the red dot in the center has an unknown location. The red dot will represent the location of the object that is being searched for. The reference nodes, also known as the readers, are labeled A_1, A_2, and A_3, the distances between the reference nodes and the tagged node are labeled d_1, d_2 and d_3. The intersection between all three nodes is the location of our unknown tag.

Each tag and reader will consist of a transceiver and an antenna. The tag transmits a signal and the signal strength, also known as RSSI (Received Signal Strength Indication) is measured between the tag and the stationed readers. The signal strength can be converted into distance, which gives us the three known distances that are needed for trilateration. The downside for using signal strength is that the calculated distance will not always have the exact distance between the reader and the tagged object. The reason for this is because the signal strength coming from the tag could be interfered with other signal frequencies, room temperature, humidity, construction within a building and metal interference. The equation converting signal strength to distance can be tuned for some of the parameters listed below, but there are some things that cannot be controlled, therefore with every distance that is calculated there will be a small error.

Once the distance is calculated, trilateration will be used to find the position of the tagged object. The way 2D trilateration works is shown below. In the following equations Xi and Yi represent the position of Ai (readers), where $i = 1,2,3$.

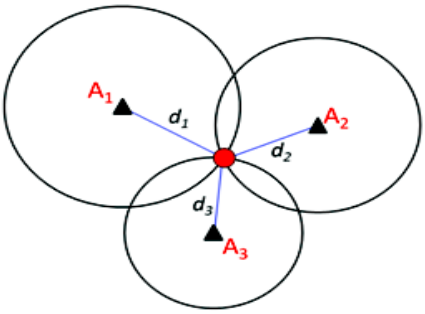


Figure 8: Trilateration in 2D

$$(x - x_1)^2 + (y - y_1)^2 = d_1^2 \qquad (Equation\ 1)$$

$$(x - x_2)^2 + (y - y_2)^2 = d_2^2 \qquad (Equation\ 2)$$

$$(x - x_3)^2 + (y - y_3)^2 = d_3^2 \qquad (Equation\ 3)$$

To simplify this system of quadratic equations, equation 3 will be substituted into equations 1 and 2, which will leave two linear equations.

$$2(x_2 - x_1)x + 2(y_2 - y_1)y = (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) \qquad (Equation\ 4)$$

$$2(x_3 - x_1)x + 2(y_3 - y_1)y = (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) \qquad (Equation\ 5)$$

The X and Y coordinates are found by solving equation 4 and equation 5 using Cramer's rule.

$$X = \frac{\begin{vmatrix} (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) & 2(y_2 - y_1) \\ (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) & 2(y_3 - y_1) \end{vmatrix}}{\begin{vmatrix} 2(x_2 - x_1) & 2(y_2 - y_1) \\ 2(x_3 - x_1) & 2(y_3 - y_1) \end{vmatrix}} \qquad (Equation\ 6)$$

$$Y = \frac{\begin{vmatrix} 2(x_2 - x_1) & (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) \\ 2(x_3 - x_1) & (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) \end{vmatrix}}{\begin{vmatrix} 2(x_2 - x_1) & 2(y_2 - y_1) \\ 2(x_3 - x_1) & 2(y_3 - y_1) \end{vmatrix}} \qquad (Equation\ 7)$$

Equations 6 and 7 will be solved using Matlab. The simulation will be shown in the section called Trilateration Simulation Results (Section 3.7).

3.3 Trilateration in 3D

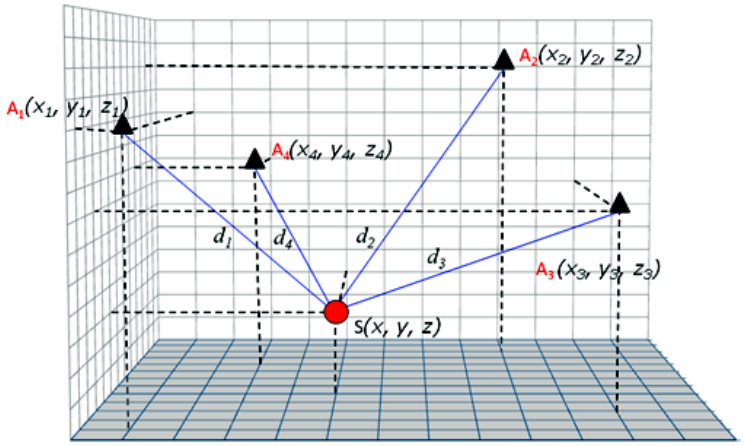


Figure 9: Trilateration in 3D

Originally, the indoor positioning system was supposed to find an object within a warehouse. With this in mind, it was taken into consideration that there are items elevated in high placed that can't be seen or reached. To do this, a 3D system is needed that will have a Z component along with the X and Y components. As you can see in figure 9, there is a fourth reader. This fourth reader gives us an extra component; therefore linear algebra can be used to find the height of the tagged object. The 3D trilateration quadratic equations are similar to the 2D trilateration quadratic equations. Also, because of the quadratic equations every reader can be at a different height. The equations for 3D trilateration are as follows:

$$(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = d_1^2 \qquad (Equation\ 8)$$

$$(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = d_2^2 \qquad (Equation\ 9)$$

$$(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = d_3^2 \qquad (Equation\ 10)$$

$$(x - x_4)^2 + (y - y_4)^2 + (z - z_4)^2 = d_4^2 \qquad (Equation\ 11)$$

The above equations can be simplified into 3 linear equations.

$$2(x_2 - x_1)x + 2(y_2 - y_1)y + 2(z_2 - z_1)z = (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) - (z_1^2 - z_2^2) \qquad (Eq\ 12)$$

$$2(x_3 - x_1)x + 2(y_3 - y_1)y + 2(z_3 - z_1)z = (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) - (z_1^2 - z_3^2) \qquad (Eq\ 13)$$

$$2(x_4 - x_1)x + 2(y_4 - y_1)y + 2(z_4 - z_1)z = (d_1^2 - d_4^2) - (x_1^2 - x_4^2) - (y_1^2 - y_4^2) - (z_1^2 - z_4^2) \qquad (Eq\ 14)$$

Now, the X,Y and Z components are found by solving equations 11,12, and 13 using Cramer's rule.

$$X = \frac{\begin{matrix}(d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) - (z_1^2 - z_2^2) & 2(y_2 - y_1) & 2(z_2 - z_1) \\ (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) - (z_1^2 - z_3^2) & 2(y_3 - y_1) & 2(z_3 - z_1) \\ (d_1^2 - d_4^2) - (x_1^2 - x_4^2) - (y_1^2 - y_4^2) - (z_1^2 - z_4^2) & 2(y_4 - y_1) & 2(z_4 - z_1)\end{matrix}}{\begin{matrix}2(x_2 - x_1) & 2(y_2 - y_1) & 2(z_2 - z_1) \\ 2(x_3 - x_1) & 2(y_3 - y_1) & 2(z_3 - z_1) \\ 2(x_4 - x_1) & 2(y_4 - y_1) & 2(z_4 - z_1)\end{matrix}} \qquad (Equation\ 15)$$

$$Y = \frac{\begin{matrix}2(x_2 - x_1) & (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) - (z_1^2 - z_2^2) & 2(z_2 - z_1) \\ 2(x_3 - x_1) & (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) - (z_1^2 - z_3^2) & 2(z_3 - z_1) \\ 2(x_4 - x_1) & (d_1^2 - d_4^2) - (x_1^2 - x_4^2) - (y_1^2 - y_4^2) - (z_1^2 - z_4^2) & 2(z_4 - z_1)\end{matrix}}{\begin{matrix}2(x_2 - x_1) & 2(y_2 - y_1) & 2(z_2 - z_1) \\ 2(x_3 - x_1) & 2(y_3 - y_1) & 2(z_3 - z_1) \\ 2(x_4 - x_1) & 2(y_4 - y_1) & 2(z_4 - z_1)\end{matrix}} \qquad (Equation\ 16)$$

$$Z = \frac{\begin{matrix}2(x_2 - x_1) & 2(y_2 - y_1) & (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) - (z_1^2 - z_2^2) \\ 2(x_3 - x_1) & 2(y_3 - y_1) & (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) - (z_1^2 - z_3^2) \\ 2(x_4 - x_1) & 2(y_4 - y_1) & (d_1^2 - d_4^2) - (x_1^2 - x_4^2) - (y_1^2 - y_4^2) - (z_1^2 - z_4^2)\end{matrix}}{\begin{matrix}2(x_2 - x_1) & 2(y_2 - y_1) & 2(z_2 - z_1) \\ 2(x_3 - x_1) & 2(y_3 - y_1) & 2(z_3 - z_1) \\ 2(x_4 - x_1) & 2(y_4 - y_1) & 2(z_4 - z_1)\end{matrix}} \qquad (Equation\ 17)$$

Both 2D and 3D trilateration have its pros and cons. Adding an extra reader to the system will increase the accuracy of the location of the tagged object in 2D. However in 3D trilataration, the fourth reader is needed for the extra unknown z-component. 3D trilateration will also be less accurate and more complex than 2D trilateration. The more complex the system is; the longer the computational time will be, which is another advantage that 2D trilateration has over 3D trilateration. But there is a way to execute a 3D system by expanding 2D trilateration and this system is called COLA, which will be explained in further detail in a section called COLA (Complexity Reduced 3D Trilateration Localization Approach) (Section 3.5).

3.4 Matlab code for Trilateration in 2D and 3D

The 2D trilateration Matlab code has to have two different Matlab files for it to work correctly. One file is the actual trilateration code, which is called two_tri.m and is located in Appendix B, the other Matlab file stores the distances between the node we are searching for and the known readers locations. The file that stores the reader coordinates and distances will be called Test2D.m and is located in Appendix B. The Test2D file has to have the known locations of the three readers and the node with the unknown location. The location of the reader will be manually put into the host computer by the user when the system is set up. However, the three distances will be fed to Matlab by the Arduino connected to the host computer. The Arduino is converting the RSSI signal into distance and sending it to Matlab through the serial port. Both Matlab files share information with each other, so, therefore once all the X and Y components of the readers as well as the distances are known, the X and Y component of the unknown node can be found using the two_tri.m file. When the program is done executing a screen will be displayed with three circles, one around each reader node, intersecting at the location of the unknown node.

The Matlab code for 3D Trilateration is very similar to the MATAB code for 2D trilateration. You can see the 3D trilateration code in Appendix B, and see how much larger and complex it is compared to the 2D trilateration code. This is why the 3D trilateration computational time is exponentially larger than the 2D trilateration computational time. Also you will notice in the simulations that there are circles around each reader node in 2D trilateration and where those circles intersect is where the unknown object is located, but for 3D trilateration there are no circles. That's because for 3D trilateration there has to be spheres instead of circles and since we were under time constraints we decided to put a dot where the unknown object is located. Other than the computational time and the extra component, the basic idea of the 3D trilateration code and the 2D trilateration code is the same. For 3D trilateration there are two Matlab files, one Matlab file stores the distances between the node that is being searched for along with the known reader nodes, and this Matlab file can be seen in Appendix B, the other Matlab file was mentioned before and it has the actual 3D trilateration code. The simulation for both the 2D and 3D trilateration are located in the section called Trilateration Simulation Results (Section 3.7.1 and 3.7.2).

3.5 COLA (Complexity Reduced 3D Trilateration Localization Approach):

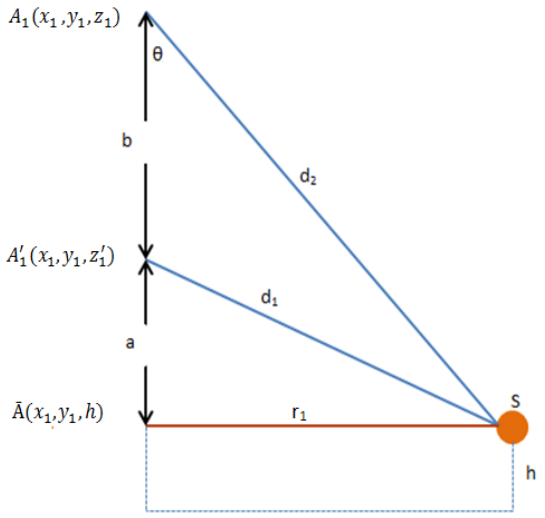


Figure 10: COLA 2D diagram

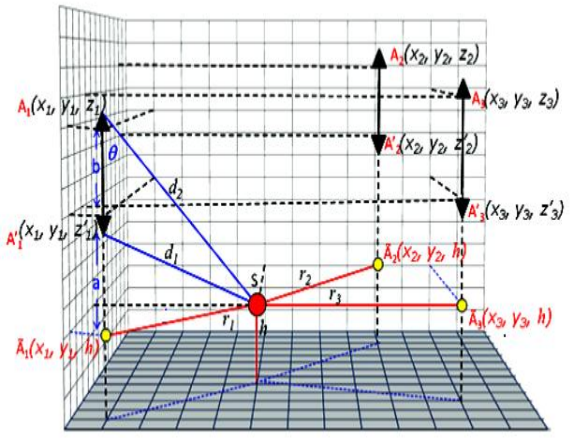


Figure 11: COLA 3D diagram

As mentioned before in the section 3.3**,** COLA can be used for 3D applications. COLA's computation time is much shorter than the traditional 3D trilateration, but COLA is much more expensive. The reason why it is more expensive is because three additional readers have to exist at the same XY axis as the original 3 readers, but have to be at an elevated location. This is the ideal method for tracking a tagged object indoors due to the fact that it is 60 percent more accurate than trilateration in 3D, the more readers, the more accurate the system will be. To find the height using COLA, complex algebra will be used and is shown below. To make this process easier to understand, take figure 10 and dissect it into 3 parts. To show how to find the height, the readers at $\bar{A}_i$ will be considered.

$$\cos\theta = \frac{a+b}{d_2} \qquad (Equation\ 18)$$

$$a = d_2\cos\theta - (z - z_i') \qquad (Equation\ 19)$$

$$d_2^2 = r_1^2 + (a + (z - z_i'))^2 \qquad (Equation\ 20)$$

$$d_2^2 = d_1^2 + 2(z_i - z_i')d_2\cos\theta - (z_i - z_i')^2 \qquad (Equation\ 21)$$

Using equation 20, cosθ can be solved for and gives the following equation:

$$\cos\theta = \frac{d_2^2 - d_1^2 + (z_i - z_i')^2}{2(z_i - z_i')d_2} \qquad (Equation\ 22)$$

The height equation is the following:

$$height = z_i - \left(a + (z_i - z_i')\right) \qquad (Equation\ 23)$$

where

$$\left(a + (z_i - z_i')\right) = (d_2\cos\theta - (z_i - z_i')) + (z_i - z_i') \qquad (Equation\ 24)$$

$$\left(a + (z_i - z_i')\right) = d_2\left[\frac{d_2^2 - d_1^2 + (z_i - z_i')^2}{2(z_i - z_i')d_2}\right] \qquad (Equation\ 25)$$

Substituting equation 23 into 22 gives us the height of the tagged object:

$$height = z_i - \left[\frac{d_2^2 - d_1^2 + (z_i - z_i')^2}{2(z_i - z_i')}\right] \qquad (Equation\ 26)$$

To find the distance(r_i), which is the distance between the tagged object and the XY coordinate of the readers is as follows,

$$r_i = \sqrt{d_2^2(1 - (\cos\theta)^2)} \qquad (Equation\ 27)$$

Substituting equation 20 in for d_2^2 and equation 21 in for $\cos\theta$ the distance can be calculated with the formula below:

$$r_i = \sqrt{\frac{-d_1^4 - d_2^4 + 2(z_i - z_i')^2 d_1^2 + 2(z_i - z_i')^2 d_1^2 + 2d_1^2 d_2^2 - (z_i - z_i')^4}{4(z_i - z_i')^4}} \qquad (Equation\ 28)$$

The last step of COLA is performing 2D trilateration using the distances (r) between the tagged object and the reference readers.

3.6 Matlab Code for COLA

The COLA Matlab code has two Matlab files. One Matlab file is used for finding distance between the known reader and the unknown tag, this Matlab file is called COLA.m and can be found in Appendix B**.** The other Matlab file is used for finding height of the unknown tag, this Matlab file is called Cola_Height.m and can be found in Appendix B. You can see that this code is less complex than the traditional 3D trilateration code and this means that the COLA computation time is much faster. The location of the unknown tag will be known so we can find the distances between the readers and the tagged object. But when the whole Indoor positioning system is set up the distances will be known from the RSSI signal strength conversion coming from the Arduino. Once the distances between the readers and the tags are known, the distance between the XY coordinate of the reader and the unknown tag will be known by using COLA.m. Once the distances between the readers XY coordinate and the unknown tags are known then the two_tri.m file will be run to find the XY coordinate of the unknown tag. The height of the object will be found using the Cola_Height.m file and it will take the data from the COLA.m file to solve for height. Once the height is found it will be stored as a variable. After the program has been completed, the location of the object will be displayed to the screen. The simulation results of COLA will be shown in the section called Trilateration Simulation Results, Section 3.7.3.

3.7 Trilateration Simulation Results

3.7.1 Simulation for 2D Trilateration

Location of Reader 1: [-2, 2]
Location of Reader 2: [2, 1]
Location of Reader 3: [-1, -2]
Location of Unknown Tag: [1, 1]

In figure 12 each red dot symbolized one reader and the location of the unknown tag is located where all three circles intersect.

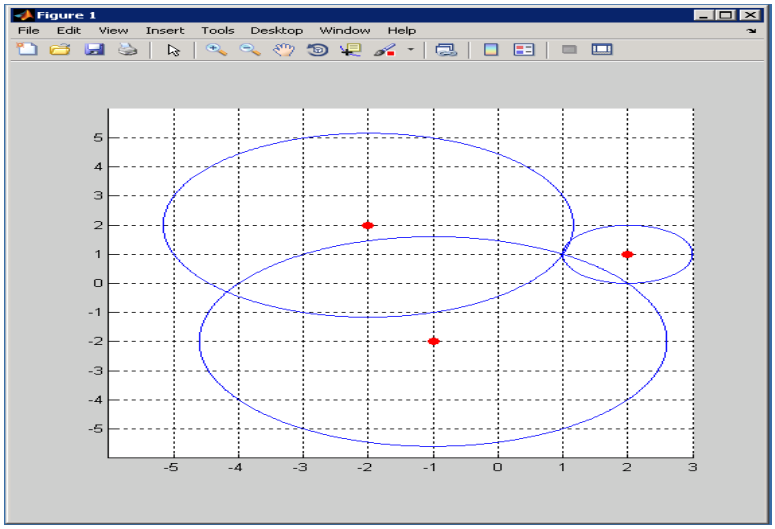


Figure 12: Simulation for 2D Trilateration

3.7.2 Simulation for 3D Trilateration

Location of Reader 1:[-2, 2, 2]
Location of Reader 2: [2, 1, -1]
Location of Reader 3: [-1, -2, 2]
Location of Reader 4: [3, 3, 3]
Location of Unknown Tag: [0, 0, 0]

For 3D trilateration, refer to figure 13. Each outer dot symbolized a reader. The location of the object is not shown with the intersection of spheres, but it is shown by a dot within the outer dots. Also, note that the readers are at different heights and the system still works correctly.

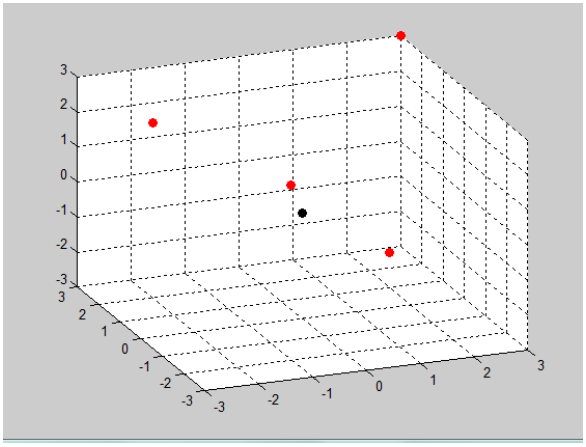


Figure 13: Simulation for 3D Trilateration

3.7.3 Simulation for COLA

Location of Lower Reader 1: [-2, 2, 2]
Location of Lower Reader 2: [1, 1, 2]
Location of Lower Reader 3: [-3, 0, 2]
Location of High Reader 1: [-2, 2, 4]
Location of High Reader 2: [1, 1, 4]
Location of High Reader 3: [-3, 0, 4]
Location of Unknown Tag: [4, 5, 1]

For the COLA simulation, refer to figure 14. Each red dot symbolizes a reader. Again, we are not able to use spheres to show the location of the unknown tag because this is in 3D; therefore, we show the location of the unknown tag with a black dot.

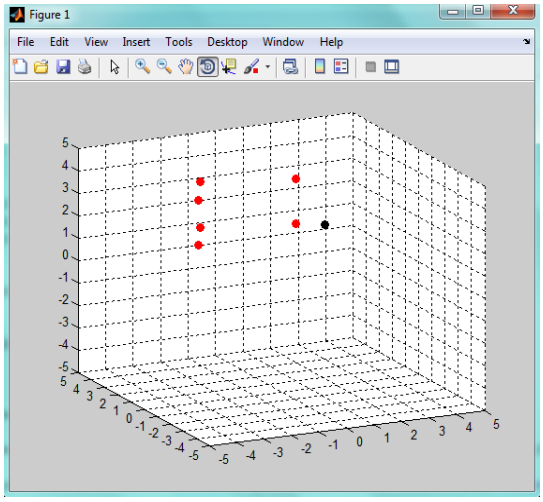


Figure 14: Simulation for COLA

3.8 Reader Stands for Trilateration

The reader stands were built originally for COLA, as you can see in figure 15, there is available space for a lower reader and a higher reader on each of the three stands. The original design was only about 2ft high. The reason for this design is because for COLA trilateration, as long as there are two different readers at two different elevations and have the same XY coordinate, height of the unknown tag can be found. With that in mind, the more compact the system is the more convenient it would be while we were demonstrating the indoor positioning system. Although it would be more convenient to have smaller stands, every time we tested the signal strength with the readers close to the ground the signal strength was inconsistent and weak. After doing some further research we realized that the ground is a good reflector of electromagnetic radiation. With the understanding that the earth could have possibly been reflecting our signal, it was decided to build the stands at approximately 5ft high. The height of these stands is where the lower readers will be. If COLA trilateration was to be tested, the stands would have to be extended to approximately 9 to 10 ft high for it to work properly.

Figure 15: All Three Trilateration Stands

Figure 16: Trilateration Stand

3.9 Trilateration Conclusion

Using trilateration was a success in this project and the actual testing and proofs will be shown in detail later in this report. Although trilateration does not give us the exact location of the object in the actual field, it does get the user in very close proximity. Throughout this semester we were not able to get the COLA working in the indoor positioning system due to time constraints, but the 2D trilateration is working better than expected. We hope another group can compound on our studies and hard work to make COLA a success for the indoor positioning system. As I mentioned before, COLA is the ideal way to locate an object. It's just a system that uses 2D trilateration, and finds the height using trigonometry. It also used six readers, which makes it much more accurate than the traditional 2D and 3D trilateration.

4 RFID Detection Device

4.1 Introduction to the RFID Detection Device

The detection device was created to make the Indoor positioning system much more user friendly. The project had to keep in mind that not all people using our system will have computer skills and may not have any type of technical background. Instead of a user typing in the object that they want to find, one could use an RFID reader with RFID cards to bring up the location of a tagged object. The basic idea behind the detection device is, for every item that is tagged in the field, it will have a RF card that is linked to it. Each RFID card has a unique 12 digit hex number. For example, if RFID card #1 = 4500B8E95541 we can link Box #1 to RFID card #1 by referencing Box #1 with the same unique 12 digit ID in our database. For this system to work correctly, it will need RFID cards, an ID-12 detection chip, and an Arduino Uno. The datasheet for the ID-12 chip is listed in Appendix A. There were several issues that came up while building the device. The first one that that will be discussed is figuring out the unique ID for each RF card, the second challenge to overcome was sending the unique ID to Matlab so the host computer would know which object the user is looking for. The project's final challenge was making the system where multiple unique ID's are stored in the host computer, so the user can just scan each card and find each item in the system.

4.2 RF Card 12 Digit Unique ID

Finding out the 12 digit unique ID was an unexpected challenge for this semester. It was assumed that the manufacturer would give us the ID for each card, but this didn't happen. To solve this problem, the ID-12 chip was connected to an Arduino as shown in figure 17. The Arduino Uno can transmit serial data using the TX pin and receive serial data using the RX pin, due to the built in serial communication library. Pin 9, also known as the data 0 pin of the ID-12 chip, will output at 9600 baud serial every time it reads a card being scanned. The data 0 pin is connected to the RX pin of the Arduino and the RF card data will be stored into the Arduino's 128 byte serial receive buffer. Once the 12 digit hex number is in the buffer, the next goal is to display it to the screen.

To program the Arduino to display to the screen, we used the serial library that is already built into the Arduino. There are some key functions that are used in the program and they are listed below.

- Serial.begin(), opens the serial port and sets the data rate at some value of bits per second, for the serial data transmission.

- Serial.read(), tells the Arduino to read the incoming serial data.
- Serial.avaliable(), checks to see if there is any data in the 128 byte serial buffer
- Serial.print(), prints the data to the screen as a human readable ASCII character

The code that was used to find the 12 digit unique ID for the RFID cards is listed in Appendix C. The code is written to open the serial port of the Arduino Uno and set the output rate at 9600 bps. Next, the code will check to see if there is data in the serial receive buffer. If nothing is returned at this point the RFID card was not detected and it will need to be rescanned. Otherwise, if it returns the number of bytes available, the unique ID number will be read and printed out to the screen.

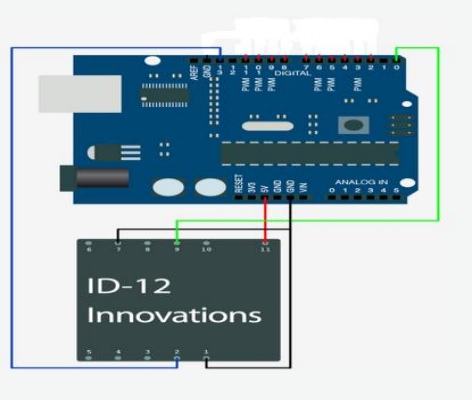


Figure 17: ID-12/Arduino circuit connection for Detection Device

4.3 Testing/Results for Finding 12 Digit Unique ID

The initial test with the circuit configuration in figure X10 and the original code, failed. At first, troubleshooting started with the ID-12 chip. To test the chip we connected a resistor and a LED light to PIN 5 of the Arduino Uno. A simple code was written to set pin 5 to a high voltage when a RFID card is detected. The experiment was tested and when the RFID card was scanned the LED would come on 50 percent of the time and stay off 50 percent of the time. This led to the conclusion of a bad connection within the breakout board or breadboard. With that in mind, the ID-12 chip was taken off the breakout board and soldered the PINS of the ID-12 chip directly to the wires that are connected to the Arduino Uno. Then we executed the test again and the LED light came on every time the RFID card was scanned. The conclusion from this test was that the ID-12 chip had good connections now; therefore, the original code was then uploaded to the Arduino. The original experiment was tested once again and when the RFID card was scanned the 12 digit unique ID was displayed on the screen as shown in figure 18. This allowed the group to move forward with the project knowing the ID number that will link the RFID card to the tag that is out in the field.

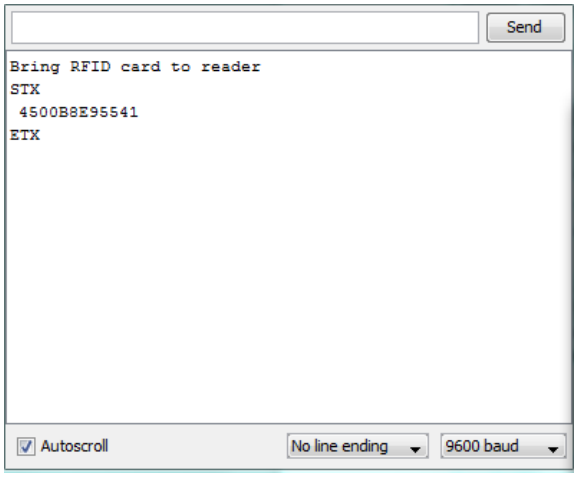


Figure 18: Display of Unique ID

4.5 Sending 12 Digit Unique ID to Host Computer

Since the ID for each RF card is known, a database is made within the Matlab program and this will allow us to run a comparison to know which tagged object the user is looking for. A 12 digit unique ID must be sent to Matlab from the Arduino Uno through the serial port for comparison of tags in the database. The newer versions of Matlab are capable of receiving and sending serial data using the communications port on a computer. For the 12 digit unique ID to be sent serially, two separate codes were written; an Arduino code and a Matlab code. Both codes are shown in Appendix C and B respectively.

For the Arduino code, additional code was added to the code that was written to display the ID number on to the screen. Instead of displaying it on to the screen, it is sent to the Matlab program. This is done by taking the ID number that is stored in the buffer and sent it through the communications port one byte at a time. The reason why it's sent one byte at a time is because that's what the Arduino Uno is programmed to do internally. For the Matlab code, Arduino COM port is opened as a file which allows the Matlab program to receive the data coming from the communications port. The Matlab code is written to receive the 12 digit ID one byte at a time since the Arduino is sending it at that rate. If both the Matlab program and the Arduino program are not sending and receiving the data at the same rate, it will not be effective. The proof that these two codes work together is shown below. In figure 18¸located in the previous section, one can see the unique ID (4500BE9285EC) and the 12 digit unique ID in Figure 19 are the same. The unique ID is sent to Matlab from the Arduino as a string of characters. Therefore when the unique ID's are programmed in to the Matlab database, they are stored as strings. This allows the user to write a string comparison code to find the tagged object that the user is looking for. Now that the unique ID's are sent to Matlab, the system is able to compare them and know which object the user is looking for.

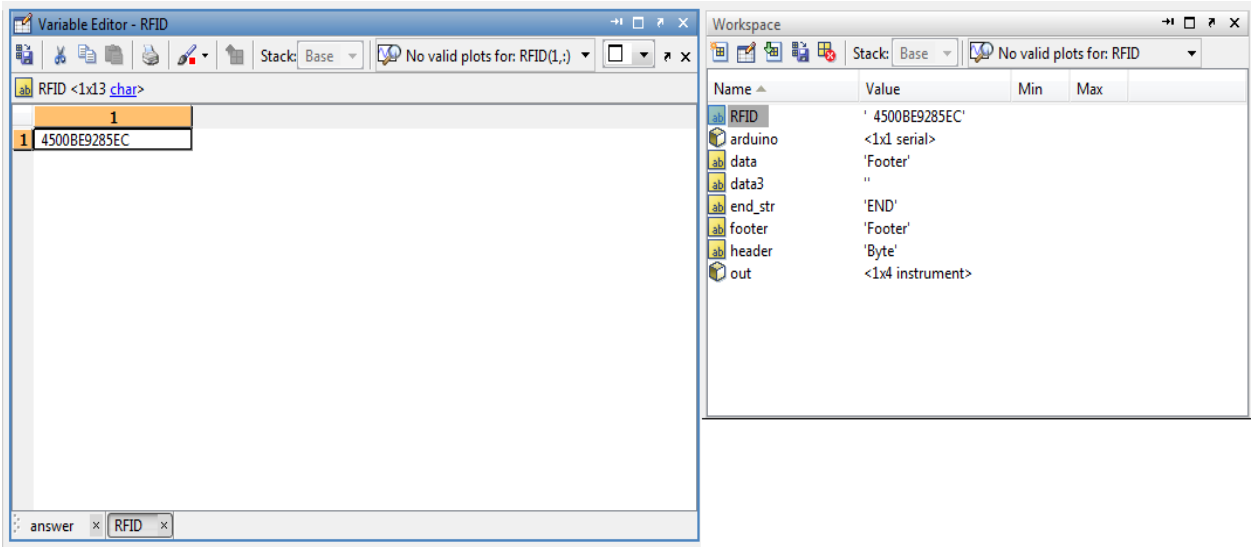


Figure 19: Display of Unique stored in Matlab

4.6 Detection Device Conclusion

The detection device is very simple to use. Once the user knows which object he/she wants to locate all they have to do is select the RFID card that is linked to that specific object. The card will be scanned by the ID-12 chip which is on the host computer box. After the card is scanned a graphical user interface will be displayed with the location of the object. As you can see in figure 20, the ID-12 chip is mounted on the outside of the host computer's box for easy access. The host computer box includes one of the readers, the host computer's Arduino (the Arduino MEGA) and the ID-12 detection device. Although, there were some difficulties with finding the unique ID numbers and sending that data to Matlab, the challenges were overcome and the system was successfully completed.

Figure 20: Host Computer Box

5 Batteries

5.1 Technical Specifications of batteries

The battery used in the design will be a lithium-ion polymer (Li-poly) battery due to its high density, long life cycles, and its ability to be thin. A typical Li-poly battery is essentially consisted of three parts or layers: the negative electrode, separator, and the positive electrode. The negative electrode consists of either $LiCoO_2$ or $LiMn_2O_4$. The typical reaction for the negative electrode is $Li_{1-x}CoO_2 + xLi^+ + xe^- \rightarrow LiCoO_2$. The separator is a conducting polymer electrolyte that performs Li+ conductions. The positive electrode is constructed with a Li or carbon-Li intercalation (the reversible inclusion of a molecule or groups between two other molecules or groups) compound. The typical reaction to the positive electrode is $Li_x \rightarrow C + xLi^+ + xe^-$. Once each of the layers is constructed, an aluminum or laminate casing is used to encase the full battery. Each of the layers is extremely thin and thus gives the battery its small and unique shapes. Figure 21 shows a cross-section of a Li-poly battery and how each of the constructed layers is built upon each other. Table 1 also shows the advantages and disadvantages of a lithium-ion polymer battery.

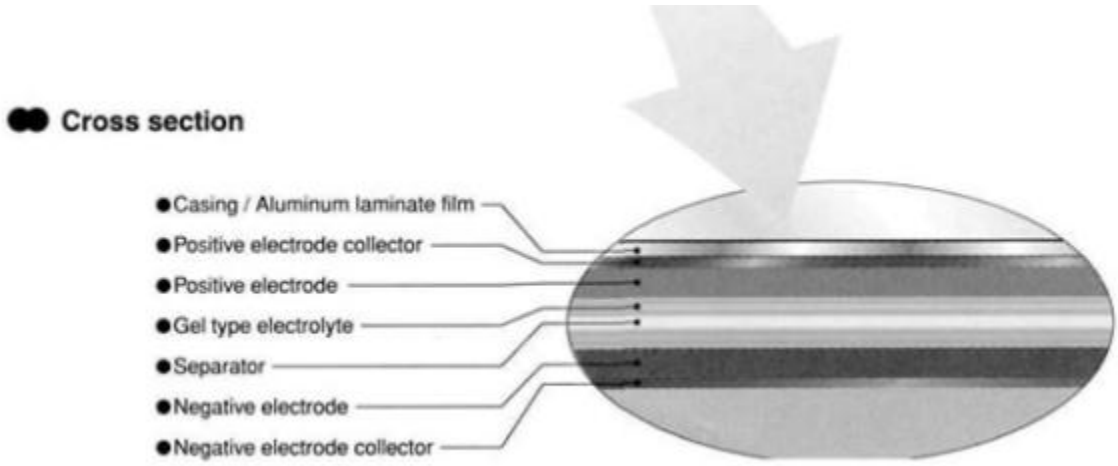


Figure 21: Cross Section of Lithium-ion polymer battery.

Table 1: Advantages and disadvantages of lithium-ion polymer battery.

Advantages	Disadvantages
High energy density.	Ageing on storage or use.
No memory effect.	Protection circuit recommended.
Low self-discharge.	Limited rate capability at low temperatures.
The design and size are easy and flexible.	Generally the highest cost but much cheaper recently.
Long life cycles (more than 1000 cycles).	Some safety issues (Flammable).
The lamination structure of electrode and electrolyte has high reliability for impact and vibration.	

The selected battery for this design is the Polymer Lithium Ion Battery SKU: PRT-08403 from http://www.sparkfun.com/products/8483. The battery has 2000mAh (milliamp hours) and is extremely light with a weight of 36g (1.27oz.). The average battery size is about four quarters in a 2x2 matrix configuration for reference with an actual dimension of 0.25x2.1x2.1" (5.8x54x54mm). The battery is extremely thin and lightweight and would provide the needed power for our design and the flexibility to be put in small confined spaces in order to design the most economic tags. Figure 22 shows a picture of the battery that will be used in the design.

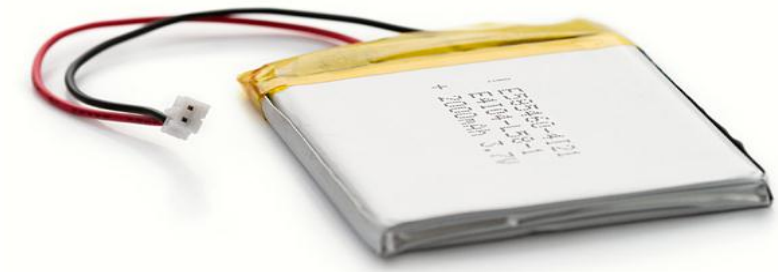


Figure 22: Lithium-ion polymer battery-2000mAh

5.2 Battery Connectors

The Arduino Fio/XBee combination will work correctly through the standard 2-pin JST connector, with 2mm spacing between the pins, from the battery to the Arduino Fio board. The high energy density cells are able to output 3.7v at 2000mAh with a discharge rate of $0.2C_5A$ discharge. The current tag system requires only around 3.3-3.4v so the battery should be able to sustain the voltage with a little bit of head room.

5.3 Battery Testing

According to the batteries' datasheet, at high temperatures of about 60^{o}C and low temperatures of 0^{o}C, the battery performed really well when the battery was initially charged and then discharged at a rate of $0.2C_5A$. At normal room temperatures and humidity, the battery had no distortion or electrolyte leakage for a discharge rate of $0.2C_5A$**.** Temperature shock was also tested with the battery from -20^{o}C for 2 hours to 50^{o}C for 2 hours repeating ten times and no electrolyte leakage was detected. This means that the battery is able to sustain both high, low, and shock temperatures without any problems. Since our position system is designed for areas such as warehouses, extreme temperatures should generally not be a problem. In case a situation does occur where extreme temperatures come into play, the battery should not have any major issues. The mechanical performance was also looked at when determining

the battery for our design. Vibrations were tested on a vibration table for 30 minutes and no influence to the batteries' electrical performance and appearance were detected. Collision performance was also looked upon in case backup batteries were needed to be encased in the tags. Multiple collisions were tested and no influence to the batteries' electrical performance and appearance were detected. Dropping the battery was also tested with random drops 10m in height onto concrete multiple times with no explosion or fire.

5.4 Battery Safety

Safety of the user must also be taken into consideration when generally dealing with lithium batteries since they can be very volatile and dangerous when re-charged incorrectly or overheated. The data sheet showed a multitude of tests conducted to determine the safety of the battery in case any of the situations should arise. The first situation was over charging the battery by sending a constant voltage of 4.8v with a constant current of $3C_5A$ till it declined to 0. There were no explosions or fire. The second situation was over discharge with a rate of $0.2C_5A$ continuously and no explosions or fires were observed. A short circuit situation was also tested to see if it would cause any problems but no problems were found. The next test was to add pressure to the battery of about 1kN and no fire or explosions were also observed. Finally, the battery was subjected to thermal shock by being put into an oven and the temperature was raised by 5^oC for every minute until it reached 120^oC and remained at that temperature for 60 minutes. By looking at these tests, the battery seems to be a perfect fit for any types of situations that may occur in a typical work area or warehouse. Therefore, it is safe to assume that the battery should fit our and the users safety needs when dealing with power.

5.5 Battery Life

Once the safety requirements of the batteries were met, the amount of discharge that the battery provides will need to be determined in order to determine the hours of battery life and the amount of stress on the battery. Discharge rates must be taken into account since high discharge rates could potentially shorted the lifespan and capacity of the battery. The battery must also not be discharged too low or over discharge. For li-poly batteries, the general range is to have the equipment cut-off at around 2.7-3.0v per cell. Anything lower than the given range of the voltage and the battery would be severely damaged and potentially put in a permanent sleep mode where even attempting to charge the battery would not bring it back to a useable condition. Therefore, most battery manufactures will ship batteries with around a 40% charge. In order to prevent to battery from reaching the state of no return, two preventative measures will be taken. The first will be discussed in the sections below by having a battery indicator ping the user that the battery has reached a certain percentage and must be replaced. The second measure is to shut the device off when a voltage of 2.7v is reached which can be accomplished through the Arduino Fio and programming code. The li-poly battery PRT-08403 for the indoor positioning system design has a recommended average discharge rate of $0.2C_5A$ and a maximum discharge rate of $2.0C_5A$ which is acceptable because the maximum discharge that will be needed is 52.1mA (45mA from the Arduino Fio and 7.1mA from the Xbee in transmitting mode) which should help to prolong the battery life as well as provide less stress for the battery.

Once the discharge rates were determined from the tag design, the amount of hours that the battery would be able to power the Arduino Fio and Xbee needed to be determined given the mAh that was provided with the battery. The battery is able to provide a 2000mAh, but in order to determine the amount of hours it would sustain; the amount of current will need to be determined. The best method to this approach is to average the amount high and low current that the circuit will draw when the Arduino Fio and the Xbee are in transmitting and non-transmitting mode. The Arduino Fio will use 45mA when transmitting at 8 MHz and <50uA in Power-down mode (Cyclic sleep). The Xbee will have an average current of 7.1mA when it is turned on and an average current of 1.1uA in sleep mode with WDT (Watch Dog Timer) disabled. Since our indoor positioning system uses a cyclic sleep method, each of the tags will only be transmitting for 1.932s then they will return to sleep or power-down mode. The hours that the batteries are able to supply will be based on our cyclic sleep method and the set time for each cycle. When the tags are not sleeping, they will be in idle mode in order to be ready to receive or send a signal. Therefore, we must also calculate the idle current which is also 1.1uA. The general equation for calculating current based on the cyclic method is:

$$\text{Current} = (52.1\text{mA}) * \frac{1.932s}{\text{Total Cycle Time (s)}} + (.0511\text{mA}) * \frac{\text{Total Cycle Time (s)} - \text{Total Sleep Time (s)} + 2.364}{\text{Total Cycle Time (s)}} + (50\text{mA}) * \frac{.432s}{\text{Total Cycle Time (s)}} + (.0511\text{mA}) * \frac{Total\ Sleep\ Time\ (s)}{\text{Total Cycle Time (s)}} \quad (Equation\ 28)$$

Once the total amount of current is found, the equation to determine hours based on mAh is:

$$\text{Hours of battery life} = \frac{2000\text{mAh}}{\text{Current}} \quad (Equation\ 29)$$

Example:

If we have a cyclic sleep cycle of 2 minutes with a total cycle of 10 minutes at a frequency of 4 MHz, the previous equations should help determine the amount of hours a 2000mAh battery will last.

$$\text{Current} = (52.1\text{mA}) * \frac{1.932s}{600s} + (.0511\text{mA}) * \frac{600s - 122.364s}{600s} + (50\text{mA}) * \frac{.432s}{600s} + (.0511\text{mA}) * \frac{120s}{600s} \quad (Equation\ 30)$$

$$\text{Current} = .255\text{mA}$$

$$\text{Hours of battery life} = \frac{2000\text{mAh}}{.210\text{mAh}} \quad (Equation\ 31)$$

Hours of battery life = 7853.59 hours (a little under one full year of battery life)

The life cycle of the battery will also need to be taken into consideration because if the battery can last one year but only charge two times before it is rendered useless, it will cause a massive economic and inconvenience to the consumer. It is also a bad idea to run the battery down past a certain percentage as that will reduce the life cycle of the battery. The best way to keep a high life cycle for the battery is to do a partial discharge. Unfortunately, this would limit the amount of battery hours we would be able to sustain. The threshold in which we would run the battery down would be 3.4v even though the batteries own charge would hold until 2.7v. This is due to the Xbee and Arduino Fio needing at least 3.35v in order to function correctly. Therefore, we also would not run the battery all the way down which would

increase the life cycle of the battery. For a partial discharge, the li-poly has an estimate of 800-1000 life cycle charges. It is believed that if the battery is kept at 3.35v as the end threshold; the battery should also be able to obtain a life cycle of around 800-1000 charges and be able to function close to one year.

5.6 Battery Storage

When storing the batteries for a long time, it is generally a good idea to store it with a partial charge so that even with a self-discharge; the battery is able to maintain a good capacity and life. The best environment for long time storage of the batteries would be a temperature of around 20^{o}C with a humidity of 45-85% and the battery charged to 40-60% in order to maintain the best quality and increase the life cycle of the battery.

5.7 Lithium-ion Polymer Recharging

Generally recharging a battery is done through a chemical reaction but in the li-poly batteries case it is different. It is mainly the flow of ion movement between anode and cathode. The li-poly battery is a "clean" system in that it only takes what it can absorb and anything more would put a high amount of stress on the battery**.** Figure 23 shows the charge stages of a lithium-ion battery in which our design will follow.

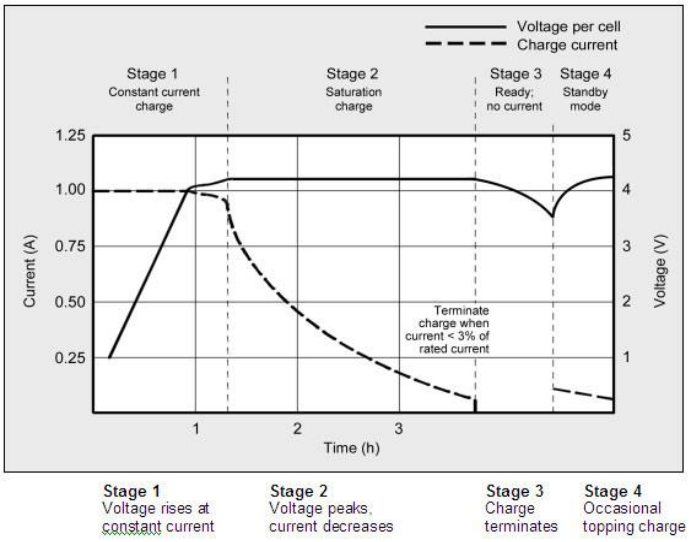


Figure 23: Charge stages of a lithium-ion battery.

If too much stress or an overcharge occurs on a li-poly battery; it could become unstable and catch on fire. Fortunately the battery was already tested with over charge enabled and no explosions or fires were detected. However, over charging could also cause damage to the cells which is why preventing over charge is a large issue even if the battery does not catch on fire. The design will use the MAX1555 integrated circuit as shown in figure 24.

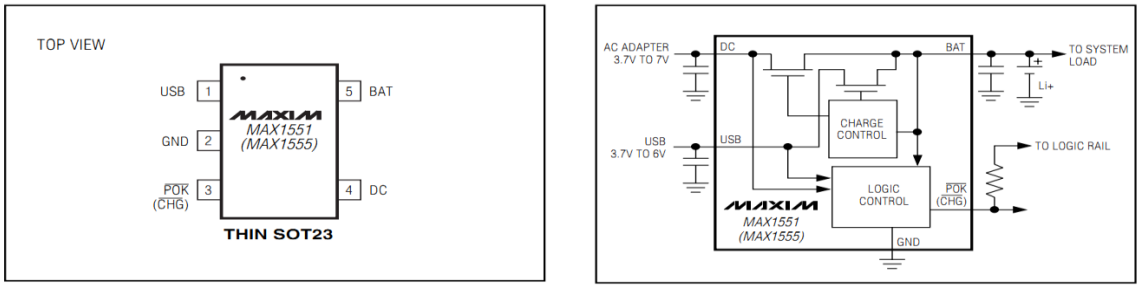


Figure 24: Pin configuration and typical operating circuit of MAX 1555

The design will able to charge through USB (pin 1) or DC Source (Pin 4) and the Arduino Fio already have this chip implemented into its design. The Arduino Fio is capable of using this integrated circuit in its system in order to charge the battery and it will automatically select between a USB or DC source if both are connected. The max current will be limited to 300mA using an external wall supply (3.7v to 7v) to prevent any problems that may occur. The PRT-08403 li-poly battery will charge at a constant current of 0.2C until the battery has reached 4.2vpc (the max voltage a cell can withstand before problems occur). It will hold the voltage at 4.2v until the charge of the current has dropped to around 10% of the initial charge rate (30mA) which is also known as trickle charging.

5.8 Results

In the end, the Spark fun Lithium-Ion Polymer battery gave the best results and showed through calculations that it was able to handle the load and abuse the system would put on the battery. It was also very thin and mobile so it would make the tag much easier to move around and have a smaller form factor. It would also be able to recharge very quickly using the Arduino Fio tags built in chip.

6 Power Supplies

The readers (Arduino Uno + Xbee) and the tags (Arduino Fio) will require different power sources for each. The readers will be plugged into the wall using an external source which will operate at 5v. The host reader will be powered and connected through USB connection which will help to power and transmit data between the host computer and host reader. Both types of powered options are shown in figure 25.

Figure 25: Arduino Uno reader. The red rectangle displays the USB power connection and the yellow rectangle displays the external wall powered connection.

The tags will be powered by lithium-ion polymer batteries connected through JST connectors to the Arduino Fio as shown in figure 26.

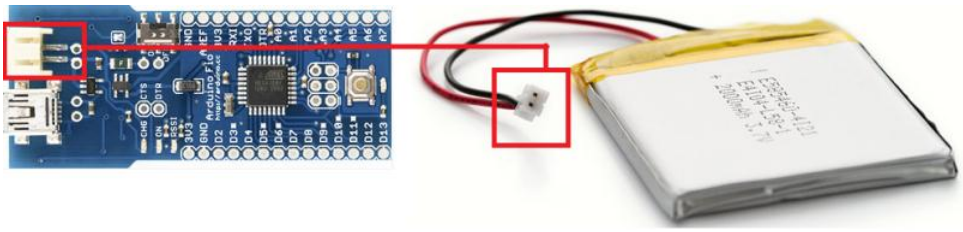


Figure 26: Arduino Fio and lithium-ion polymer battery JST connection.

Two batteries were initially ordered in order to test the products. The batteries have an average voltage of 3.7v with a maximum of 4.2v and a charge of 2000mAh. The cut-off voltage for the battery is 2.75v. Each of the batteries was tested with a multimeter to see if they were already charged and each battery had around 4v. If the battery ever dropped to the cut-off voltage, it would need to be recharged. The Arduino Fio already has an Integrated circuit which could recharge the battery. In order to determine when the battery needed to be recharged, a battery power indicator would need to be built in order to notify the user through the GUI that the battery would need to be changed.

7 Battery Power Indicator

The battery power indicator essentially uses a comparator and a voltage divider in order to determine when it would notify the user of low power. Instead of completely building another circuit, the Maxim 8212 Integrated circuit was used instead shown in figure 27.

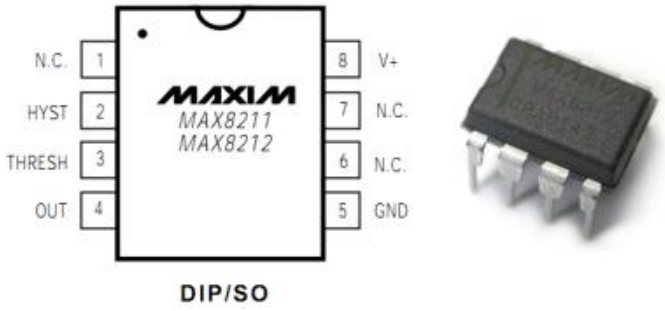


Figure 27: Maxim 8212 chip and pin layout.

7.1 Battery Power Indicator Revision 1

By using the Maxim IC, the comparator was complete and the only thing that needed to be determined was the values for the voltage divider and the hysteresis. In order to use the correct values for resistors, we needed to determine all the voltage usage and cut-off points of the Arduino Fio and the lithium-ion polymer battery. The Arduino Fio could operate from 1.8v to 5.5v but since our battery cut-off voltage was 2.75v, it was decided that an estimate of 2.9v would be perfect. This leaves a little breathing room

in case the battery was not immediately taken and charged. For the voltage divider and hysteresis, we chose R_1=1MΩ and used the equations:

$$R_2=R_1 \times \frac{(V_l-V_{th})}{V_{th}} = R_1 \times \frac{(V_l-1.15v)}{1.15v}$$ Equation 32: Determine R_2

$$R_3=R_2 \times \frac{(V_u-V_l)}{1.15V}$$ Equation 33: Determine Hysteresis

Thus, R_2=1.44559MΩ, and R_3=43kΩ. Once the values were determined the circuit was built using a breadboard in case any changed needed to be made as show in Figure 28.

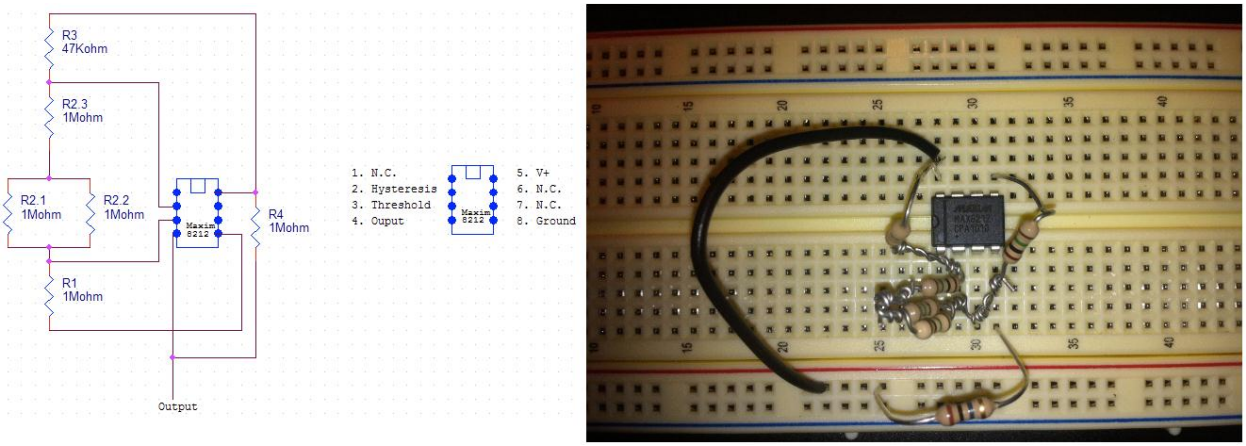


Figure 28: First iteration schematic and built battery indicator on breadboard

In order to test the battery power indicator, a variable voltage supply was used simulating the lithium-ion polymer batter. The voltage was set to 4v and gradually decreased until we hit a point in which the output gave a high. Ideally, it would have been the calculated 2.9v. In reality, the point at which it gives a high is 2.907v which is still extremely close to our ideal point. Then, the voltage supply was gradually increased and once the voltage hit 2.951v it went back to low. By subtracting these two values 2.951v-2.907v we notice that our hysteresis is .044v. This means that if the battery drained to 2.907v and somehow received extra voltage to bump it to 2.922v it would still stay high instead of toggling between low and high constantly causing false alarms and stress on the circuit.

7.2 Results

There were some issues when building this circuit. Originally an LED was put at the output so it would be able to notify the user visually that the battery was low. It was realized that our Maxim 8212 chip was inverted. In other words, the LED lit up when a low value was given and off when a high value was given. This would cause too much of a drain on the battery if the LED was constantly on. In order to solve this problem, an inverter was created using BS170 mosfet transistor. Unfortunately, under many tests and circuit redesigns, the voltage needed to fire the LED was too low and the transistor was unable to provide the voltage. Thus, the idea was finally scrapped and an LED will be routed and attached to the Arduino Fio instead.

7.3 Batter Power Indicator Revision 2

The previous battery indicator needed to be revised since the team had underestimated the amount of voltage that the Arduino Fio and XBee system used. Also, the previous battery indicator needed to be adjusted into a much simpler version in order to be attached to the Arduino Fio for the device to be more mobile and the resistors were adjusted to give a more precise value for our threshold. By using the Maxim IC, the comparator was complete and the only thing that needed to be determined was the values for the voltage divider and the hysteresis. In order to use the correct values for resistors, we needed to determine all the voltage usage and cut-off points of the Arduino Fio and the lithium-ion polymer battery. The Arduino Fio could operate at 3.3v but since our battery cut-off voltage was 2.4v, it was decided that an estimate of 3.4v would be perfect. For the voltage divider and hysteresis, we chose R1=1MΩ and used the previous equations 31 and equation 32 to recalculate the resistor values. Thus, R_2=1.86957MΩ, and R_3=81kΩ. Once the values were determined the circuit was built using a breadboard in case any changes needed to be made as show in Figure 29. The battery indicator circuit was reduced dramatically in size and the circuit works just as well as the previous iteration.

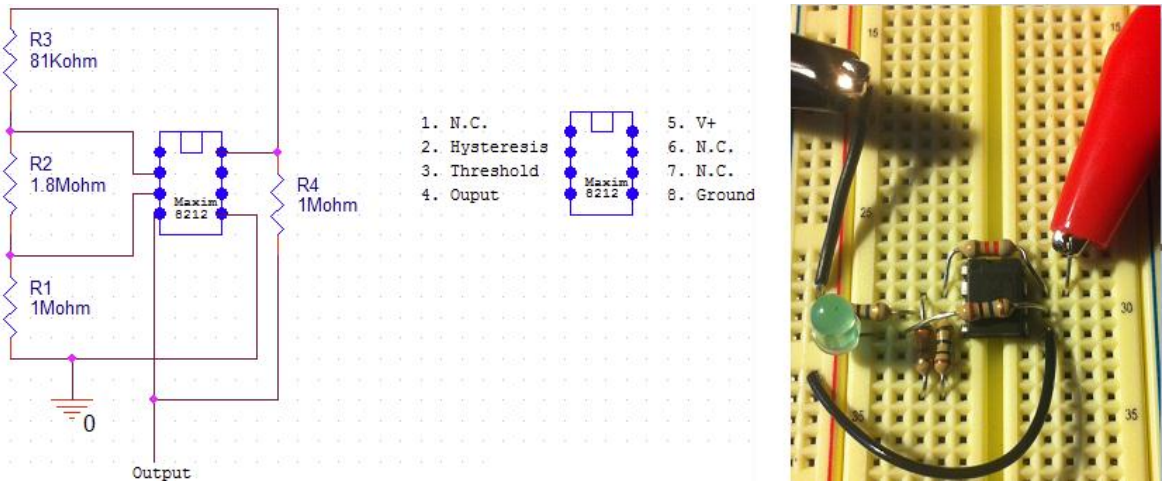


Figure 29: iteration 2 revised schematic and built battery indicator on breadboard

7.4 Battery Power Indicator Testing

In order to test the battery power indicator, a variable voltage supply was used simulating the lithium-ion polymer batter. The voltage was set to 4v and gradually decreased until we hit a point in which the output gave a high. Ideally, it would have been the calculated 3.3v. In reality, the point at which it gives a high is 3.308v which is still extremely close to our ideal point. Then, the voltage supply was gradually increased and once the voltage hit 3.35v it went back to low. By subtracting these two values 3.35v-3.308v we notice that our hysteresis is .05v. This means that if the battery drained to 3.308v and somehow received extra voltage to bump it to 3.31v it would still stay high instead of toggling between low and high constantly causing false alarms and stress on the circuit.

The power requirements for this design are one of the most important aspects due to the design. The design must be able to incorporate a battery that is small enough to be lightweight and portable, to be able to last an estimate of one full year before needing to recharge, and to be as cost-effective as

possible. Other components that would benefit the power design requirements are being able to notify the user when the battery is low and the ability to recharge the battery when needed.

7.5 Results

The second revision battery indicated that was built fit every specification the team had for the indicator. It was very small and mobile once attached to the Arduino Fio and the voltage requirement was spot on and hit 3.3v with a hysteresis to prevent false alarms. Although, one issue was still not resolved and that dealt with the battery indicator being inverted. This means that when the battery had power then the light would stay on. This would drain the battery a lot. A mosfet was added during the testing to see if it would reverse the situation but in the end, there was no LED that would power on with a voltage that low. Thus, this shows that a proof of concept and it could work perfectly if we had better components.

8 XBee Programming

8.1 Introduction to XBee Programming

Using factory default settings and the XBee shield interface to the Arduino, the Arduino can be programmed to send and receive data between two XBee modules using the built-in Serial library. A programmer can use serial.print() to send data and serial.available() , serial.read() to detect and read data. The default factory settings also determine what ISM band the modules operates with, the destination address (default is set to broadcast to all available nodes in range), and the channel the module is in.

8.2 Testing and Verification of XBees

After the XBees were received from www.sparkfun.com, two modules were inserted in XBee shields interfaced with Arduinos. One module was programmed to continuously broadcast a byte of information. Another module was programmed to check the serial input buffer for any serial output from the XBee, read this data into a variable, and output the variable to the serial monitor on the desktop.

8.2.1 Results

The result from this test was every XBee module was verified for functionality upon arrival in the mail.

8.3 AT Command Programming

To change the settings of an XBee, you must put the XBee into AT command mode. This is done by sending the 3-character command sequence “+++” to the XBee module and observing the guard times before and after the command characters. The Guard Times parameters have a default value of 0x3E8 or 1 second.

Once the AT command mode sequence has been issued, the module responds by sending an “OK\r” out of the DOUT pin. Once this response is received, AT commands can be sent to the XBee through the DIN pin.

There are two separate ways of sending AT commands to the XBee module. The first way is to send an AT command such as “ATDL” with no value. The module will respond to this command with the current

destination low address. The second way is to send an AT command with a value such as "ATDL0000FFFF". This command will change the value of the 32 bit register holding the low part (32 bits of 64 bit address) of the destination address to 0x0000FFFF. The module will respond with "OK" if the command upon successful execution of the command or respond with "ERROR".

Commands will not take effect until the AC (Apply Changes) command is issued or the WR (Write) command is issued. The WR command will write the new value to non-volatile memory on the XBee module so that the value will remain on the XBee after the XBee is powered down. To exit command mode, the "ATCN" command must be sent.

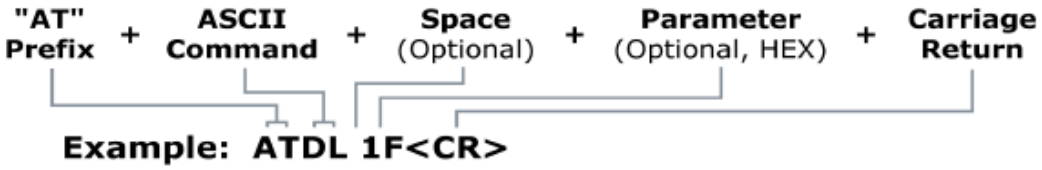


Figure 30: AT Command Syntax

8.3.1 AT Command Test

Using the AT command code in Appendix [x], AT commands were sent to an XBee module and 64 bit destination address was changed from broadcast (DL = 0x0000FFFF) to the 64 bit address of another XBee module. Using the X-CTU software provided by Digi International, the non-volatile memory on the XBee can be read and displayed. An XBee module was plugged into an XBee Explorer and connected to X-CTU. An XBee explorer is a USB to serial base unit for the XBee line. Using a mini USB cable, the explorer can be attached directly to a computer for use with the X-CTU software. The memory settings were read into X-CTU as seen in figure 32 and confirmed to be correctly saved on the non-volatile memory of the XBee. Data was then sent between two modules, similar to the test in 8.2. The sender was the XBee with the updated destination address of the receiver. A third XBee was interfaced with an Arduino to ensure that only the receiver received data from the sender. Since every XBee has a unique 64-bit address, the only module that should receive information is the receiver.

Figure 31: USB Explorer

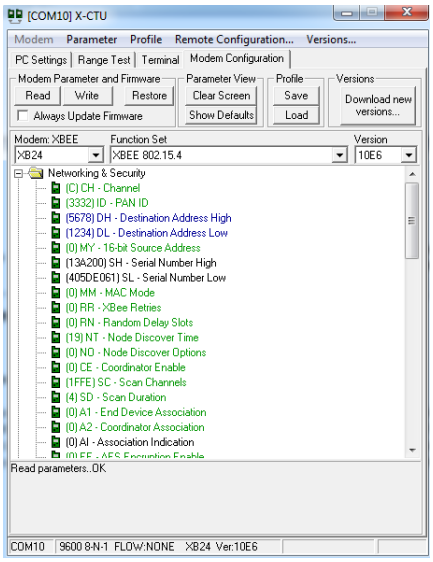


Figure 32: X-CTU Screenshot

8.3.2 Results

The result from this test was that AT commands were successfully saved into the memory of the XBee. By using the X-CTU software and XBee explorer, the non-volatile memory was checked for correctness. Also, the receiver successfully received information from the sender. While this was happening, the third XBee received no information. This is the correct result because the sender is no longer broadcasting the data to all available modules, only sending it to the 64 bit destination address.

8.4 API Programming

API operation is an alternative to transparent (AT) operation. The frame-based API extends the level to which a host application can interact with the networking capabilities. When in API mode, all data entering and leaving the module's UART is contained in frames that define operations or events within the module. A host application, an Arduino for example, can send data frames to the module that contain address and payload information instead of using command mode to modify addresses. The module will send data frames to the application containing status packets; as well as source, and payload. The reasons for the use of API programming in the positioning system is: the transmission of data to multiple destinations without entering Command Mode, to be able to identify the source address of each received packet, and to retrieve the RSSI information from memory. The following data frames are using the Digimesh networking scheme and modules.

8.4.1 API Data Frames

8.4.1.1 AT Command

The AT Command is same command used in Section 8.3. The frame as seen in figure 33 is what the host application needs to use to query or set module parameters on the XBee. The API command applies changes after executing the command.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x04	
	Frame-specific Data	Frame Type	3	0x08	
		Frame ID	4	0x52 (R)	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		AT Command	5	0x4E (N)	Command Name - Two ASCII characters that identify the AT Command.
			6	0x48 (H)	
		Parameter Value (optional)			If present, indicates the requested parameter value to set the given register. If no characters present, register is queried.
	Checksum		8	0x0F	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Figure 33: AT Command Digimesh API Frame

8.4.1.2 AT Command Response

In response an AT Command frame, an XBee will respond with an AT Command Response frame. This frame will indicate if the settings were set successfully or will respond with register data if memory was requested to be read. An example would be “DB” being sent as an AT Command and the module responding with “0x4C” in the Command Data field.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x05	
	Frame-specific Data	Frame Type	3	0x88	
		Frame ID	4	0x01	Identifies the UART data frame being reported. Note: If Frame ID = 0 in AT Command Mode, no AT Command Response will be given.
		AT Command	5	‘B’ = 0x42	Command Name - Two ASCII characters that identify the AT Command.
			6	‘D’ = 0x44	
		Command Status	7	0x00	0 = OK 1 = ERROR 2 = Invalid Command 3 = Invalid Parameter
		Command Data			Register data in binary format. If the register was set, then this field is not returned, as in this example.
	Checksum		8	0xF0	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Figure 34: AT Command Response Digimesh API Frame

8.4.1.3 Transmit Request

The Transmit Request API frame will cause the XBee module to send data as an RF packet to the specified destination. A host application specifies a destination by the 64-bit address field in the data frame. To send a broadcast signal, you can set the 64 bit address to 0x000000000000FFFF. In series 1 modules, you can also use a 16 bit address transmit request. Digimesh protocol only supports 64 bit address transmit request frames.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x16	
	Frame-specific Data	Frame Type	3	0x10	
		Frame ID	4	0x01	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following address is also supported: 0x000000000000FFFF - Broadcast address
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x0A	
			11	0x01	
			LSB 12	0x27	
		Reserved	13	0xFF	Set to 0xFFFE.
			14	0xFE	
		Broadcast Radius	15	0x00	Sets maximum number of hops a broadcast transmission can occur. If set to 0, the broadcast radius will be set to the maximum hops value.
		Transmit Options	16	0x00	Bitfield: bit 0: Disable ACK bit 1: Don't attempt route Discovery. All other bits must be set to 0.
		RF Data	17	0x54	Data that is sent to the destination device
			18	0x78	
			19	0x44	
			20	0x61	
			21	0x74	
			22	0x61	
			23	0x30	
			24	0x41	
	Checksum		25	0x13	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Figure 35: Transmit Request Digimesh API Frame

8.4.1.4 Receive Packet

When an XBee received an RF packet, the data frame shown in figure 36 is sent out of the UART to the host application.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x12	
	Frame-specific Data	Frame Type	3	0x90	
		Frame ID	4	0x00	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		64-bit Source Address	MSB 5	0x13	64-bit address of sender
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		Reserved	12	0xFF	Reserved
			13	0xFE	
		Receive Options	14	0x01	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet
		Received Data	15	0x52	Received RF data
			16	0x78	
			17	0x44	
			18	0x61	
			19	0x74	
			20	0x61	
	Checksum		21	0x11	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Figure 36: Receive Packet Digimesh API Frame

8.4.2 Arduino-XBee API Library

A user-made open source library that was made for API mode operation was found (http://code.google.com/p/xbee-arduino/) and researched. It supports both Series 1 and Series 2 XBee modules with support for a majority of packet types (Transmit Request, etc) supported. To use this library, the XBees must be set to API mode 2 (escaped characters enabled).

8.4.2.1 API Library Test

Since the Digimesh modules have the specifications as the Series 1 modules, it was expected that the library would work with the Digimesh modules. An Arduino program was written to send an API AT Command to a module and retrieve data from memory similar to the test done in section 8.3.1. Another program was written to do a sender/receiver test similar to test done in section 8.2.

8.4.2.2 Results

The AT Command test was a success. The Arduino successfully sent an AT Command data frame and successfully received an AT Command Response data frame back and read the data from memory. The transmission test was a failure. Using example code from the library's maker for the sender and receiver, the receiver never received any RF data and did not connect with the sender XBee. The conclusion from this test is that the library's code for series 1 does not work with the Digimesh data frames.

8.4.3 API Library Programming

From the results of the API Library Test, research was done to troubleshoot the transmission failure. The result of that research indicated that the Series 1 module and Digimesh modules have different Transmit Request and Receive Packet data frames. The user-made library did not support these Digimesh data frames.

8.4.3.1 Series 1 vs Digimesh Data Frames

Comparing the Series 1 Transmit Request data frame in figure 37 to the Digimesh Transmit Request in Section 8.4.1.3, the reserved bytes (0xFFE) and broadcast radius byte was inserted into the Digimesh data frame.

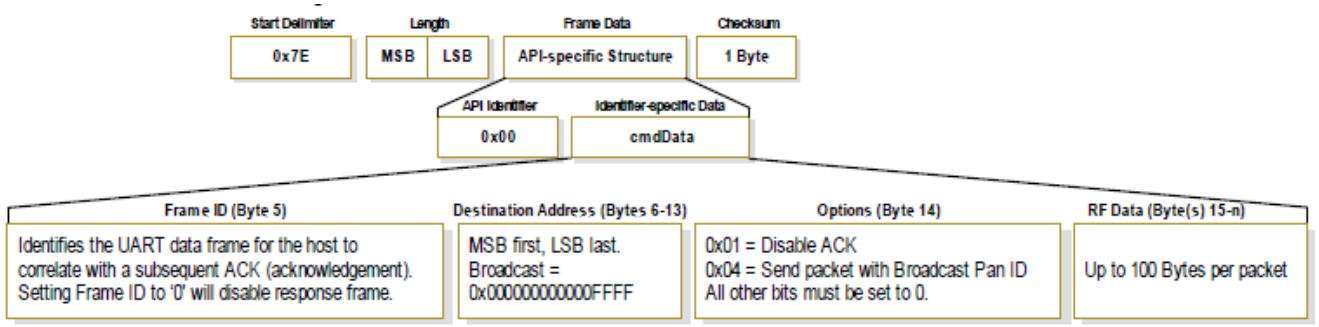


Figure 37: Series 1 Transmit Request

Comparing the Series 1 Receive Packet in figure 38 to the Digimesh Receive packet in Section 8.4.1.4, the RSSI byte was removed in the Digimesh data frame and the reserved bytes (0xFFFE) were inserted.

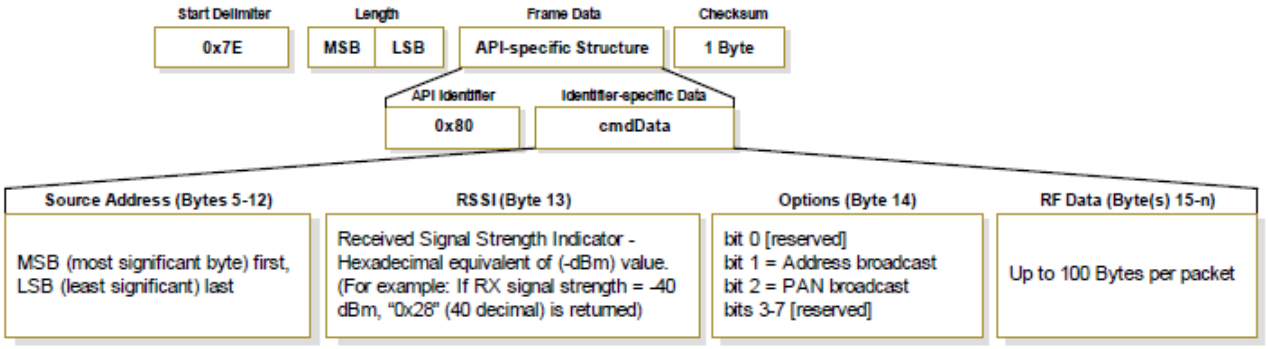


Figure 38: Series 1 Receive Packet

Using this information, the library was updated with the DMTxRequest and DMRxResponse functions (seen in Appendix D).

8.4.3.2 API Library Test 2

A second transmission test was done using the updated API library. The programs used for the sender and receiver were similar to programs used in section 8.4.2.1 except for the use of DMTxRequest instead of Tx64Request and DMRxResponse instead of Rx64Response.

8.4.3.3 Results

The XBees correctly transmitted data between the sender and receiver modules. Data was read from the receive packet and outputted to the serial monitor on the host computer. With API transmission working, distance testing can begin.

9 Tag/Reader Distance Tests

9.1 Tag/Reader Testing Introduction

In order to make sure the system worked correctly, a distance test was needed. This involved placing a tag at a set distance away from the reader. And through computation and programming, obtaining a hexadecimal value that will then be converted into meters in order to determine the distance. This testing was necessary seeing how this is the main reason for our system; calculate the distance between an object in the field in relation to readers throughout a given room or area.

9.1.1 Distance Formula

From previous testing, it was concluded that

$$d = 10^{\left(\frac{P_0 - F_m - P_r - 10*n* \log_{10}(f) + 30*n - 32.44}{10*n}\right)} \ (Equation\ 33)$$

was the most accurate distance formula available. The variables in the formula are as follows:

$$F_m = Fade\ Margin = 10$$

$$N = Path - Loss\ Exponent = Variable(tuned)$$

$$P_r = RSSI\ Received = RSSI\ Signal\ Read\ by\ Arduino$$

$$P_0 = RSSI\ Received\ at\ 0\ meters = 23$$

$$f = frequency(MHz) = 2400$$

Using this formula, the next tests were performed where a distance was calculated between a reader and tag using RSSI data.

9.2 Test 1

Testing of the system began inside one of the rooms located in the ERAD building. Every meter was marked off using a tape measure and a yardstick. Also, blue scotch tape was used to designate each meter. Over the course of a few weeks, the system was tested at least 10 times. Figure 39 shows the results of the first test that involves the one tag and only one reader:

Figure 39: Varying distance testing between tag and reader 1

9.2.1 Results

Test 1 was measured at a distance of 7.5 meters. Test 2 at 1m, test 3 at 9m, and test 4 at approximately 5m. As shown from figure 39, the results are non consistent, yet some are closer to their actual approximated measurable values. It can be seen that test 3 produced the most accurate representation of distance comparatively to how far apart the tag and the reader were. Test 4 also produced similar results.

The system was tested at different lengths to also see how accurate the readings would be depending on the distance between the tag and the reader. This is one of the main reasons to why the distances vary each test. Another reason is to test the theory that at further distances, the readings would become more accurate rather than at closer distances. For this test, that theory was proven to be true.

9.3 Second Distance Test

This test was conducted outside in the hallway of the ERAD building next to the main lab. This area was chosen because of the open space as well as the possibility of less interference from metallic objects such as lockers, wires, and metal stands. For this test, five distances were measured. Figure 40 displays the results of each test:

Figure 40: Varying distance testing between tag and reader 2

9.3.1 Results

From the results in Figure 40, the conclusion is that the most accurate reading for each testing sequence is when the devices are both facing away from each other. The one exception is when the distance is stretched further than 9 meters such as that shown for test 3. However, once the reader was facing away from the tag, which was facing towards the reader, the resulting distance shot up dramatically leading to an inaccurate reading.

Throughout the testing process, it was observed that the system was not omni directional. This is noticeable from there being empty results for the "Facing", "Readers Facing, Tags Away", and "Readers Away, Tags Facing" results. These results were either very small, large, or too unstable to be read in as actual results. In order to make sure of the results that were received, it was decided to do another testing indoors which involved other methods.

9.4 Third Distance Test

For this testing series, our goal was to see if the devices would either read better or worse if they were placed on their sides or held in the air. This was also attempted in order to obtain a better range from the antennas as well as possibly increasing the accuracy of their radiation spans. Figure 41 shows the results of the testing:

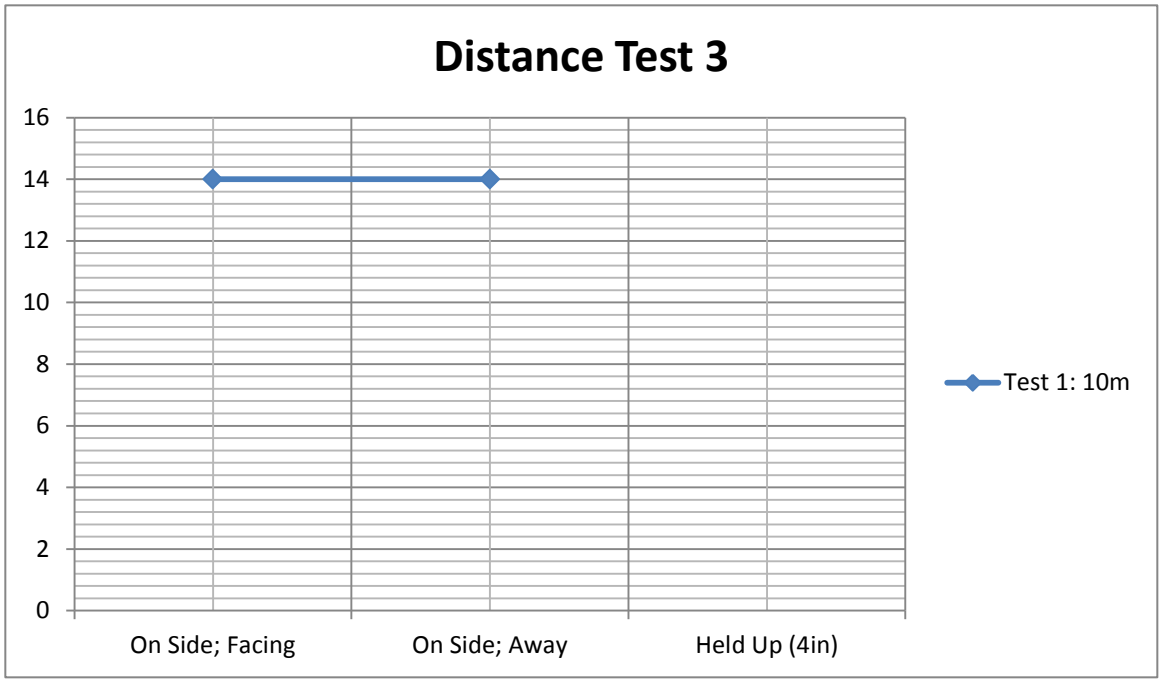


Figure 41: Single distance testing between tag and reader

9.4.1 Results

When the devices were placed on their sides, facing towards each other and facing away, they both produced the same results of 14 meters. These results are obviously off by 4 meters for both sequences. Therefore, placing the devices on their sides has a detrimental effect on the results of the system. Lastly, the devices were held upward by at least 4 inches. However, numerous problems occurred. The values tend to jump around rather than stay consistent. There was also an issue in which the signals strength was not strong enough. Lastly, there could have been added interference from our hands holding the devices upward. Consequently, holding the devices upward produced nothing as far as distance was concerned for the project.

9.5 Conclusion

The conclusion from these tests was that the chip antennas that came with the Digimesh XBee 2.4 GHz modules were not suitable for use in a location system. From the distance tests in Section 9, it was clear that the antennas were not omni-directional, which is a requirement for a location system. This is a requirement because the system will not know the orientation of the tag. Research was done (see Section 10) and a decision was made to switch XBee modules and antennas to equipment provided by the Cameron group.

10 Antenna Comparisons

In order to provide a better signal and receive all the data/packets from each of the Arduino Uno readers we had use antennas designed for 2.4GHz applications. The team noticed a big difference between the onboard chip antennas of the XBee and the dedicated external antennas that the Cameron group helped purchase and fund. The main differences between the two antennas are the range and the

radiation pattern. The ideal or standard radiation pattern would be perfect oval shapes in all directions but there are many environmental affects that could distort the radiation pattern.

10.1 XBee Antennas

Initially, the chip antennas were already embedded into the Digimesh XBee modules.

Figure 42: XBee with an integrated chip antenna show in the red box

Antenna location is very important for optimal performance. In general, the antennas radiate and receive best perpendicular to the direction in which they are pointing. Therefore, a vertical antenna radiation pattern is strongest across the horizon. Metal objected near the antenna may also distort the radiation pattern of the XBee antenna. Metal objects between the readers and the tags will also block the radiation path or reduce the transmission distance so all the readers and tags should be positioned away from them if possible. When testing the antenna in a room. The team originally overlooked ventilation ducts, metal poles and studs, and even concrete due to the metal reinforcement usually embedded inside them. Thus, the readers should not be placed inside a metal enclosure and make sure that the embedded XBee is placed at the edge of the host PCB where it is mounted. Due to all these issues that were overlooked when testing and the radiation pattern was already distorted to begin with as seen below in figure 43:

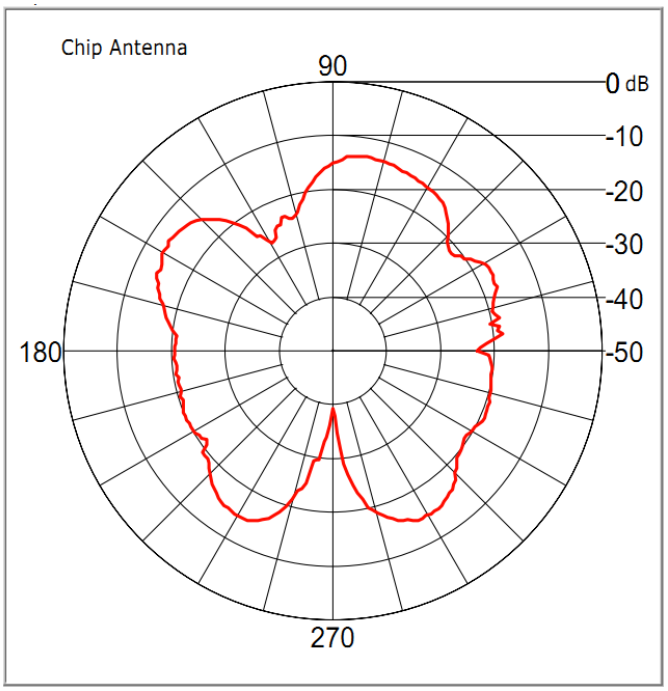


Figure 43: Chip antenna radiation pattern from XBee

By looking at the figure, the team decided that using these antennas were not going to give us the precise and accurate measurements that we needed in order to make our indoor location system work. During the many days testing our setup, we were never able to get any decent readings that were even close to our percent error that the team had estimated.

10.2 Titanis Antennas

The new antenna is the Titanis 2.4 GHz Swivel SMA Antenna by Antenova. The Titanis antenna is also designed for 2.4 GHz applications and meets the FCC regulation with high efficiency. The Texas Instruments' location system used for these antennas and these antennas were not in use. With the backing of the Cameron group, new XBees were purchased and these antennas were put into use. A model of the antenna can be seen below in figure 44:

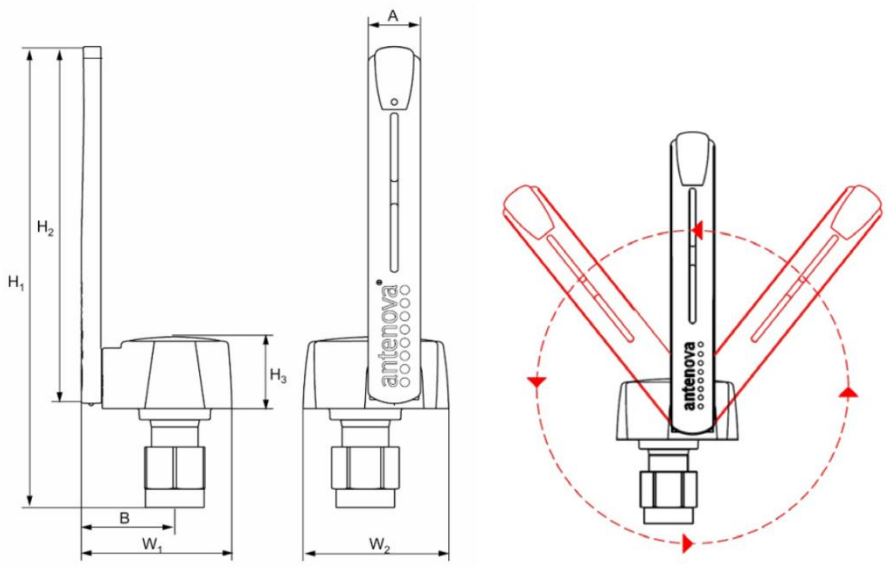


Figure 44: Titanis antenna used to attach to the XBee

The Titanis antenna is very configurable and can swivel in any direction in order to provide the best radiation pattern. The radiation patters for all the planes can be seen below in figure 45:

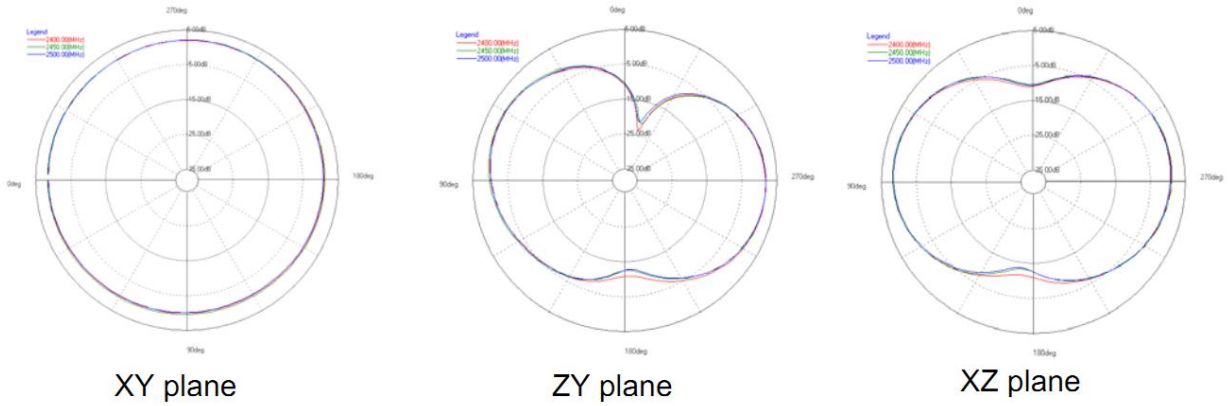


Figure 45: Titanis antenna radiation pattern from different planes

With a dedicated antenna instead on the onboard antenna, the radiation patterns are much rounder and stronger. Although the metal and concrete interference still apply, the Titans cover a larger area than the regular XBee antennas and give us a much better and accurate reading.

10.3 Results

In conclusion, it was decided that the Titanis antenna was much more accurate and gave a much wider range for our indoor positioning system. Due to flooding at Digi's factory causing a shortage of Digimesh modules, the decision to switch to 802.15.4 XBee modules for their RPSMA connectors was made.

11 Distance Tests with New Antennas

11.1 Test 1

Due to the bad radiation from the old antennas, new antennas were obtained from the Cameron group. As mentioned, these antennas provided a better radiation allowing for a stronger signal between devices. Also, due to the better radiation, testing the system at further distances was now possible.

Problems were encountered during the initial testing period. It was discovered that doing the distance testing indoors with the new antennas caused for inaccurate results. This was mainly due to the new antenna's sensitivity to metal interference and WiFi interference, which operates on same ISM band. Therefore, the testing was taken outside in front of CEBA. Table 2 displays the results:

Table 2: Distance testing with new antennas 1

Actual Distance	Calculated Distance
10m	10.5m
6m	6m
19.1m	19m
21m	25m

11.1.1 Results

From these four testing sequences, it can be seen that the results were more accurate due to the new antennas. None of the results match exactly the distance to which they were placed, but the degree of error remained so small that it was not a big concern.

11.2 Test 2

Due to the fact that cars were passing by while the system was being testing in front of CEBA, testing was postponed and resumed on the parade grounds. As testing commenced, it was noticed that the measurements would vary off by at least 2 meters and at most 6. The environment was considerably more humid and higher temperature than test 1. Figure 46 displays and shows the results:

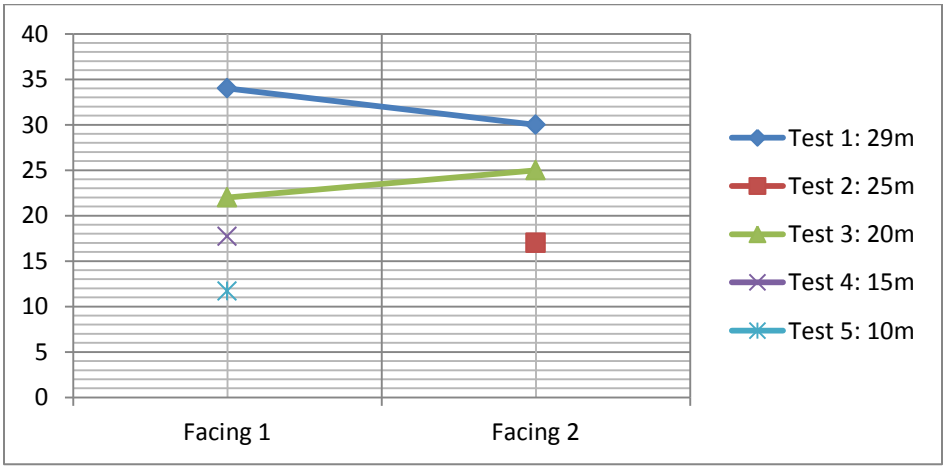


Figure 46: Distance testing with new antennas 2

11.2.1 Results

The calculated distances were off by at least 2 meters to 6 meters. The changes in environment attenuated the signal strength and made the signal strength weaker than Test 1 for the same distances apart. The conclusion from this test was the distance formula needs to be tuned to the environment every time we test.

11.3 Conclusion

From the distances tests, it was concluded that the Titanis antennas would work with a location system. The antennas are omni-directional on the XY plane. The lesson learned from these tests was that the distance formula path-loss exponent needs to be tuned to the environment. Factors that can affect RSSI are: temperature, humidity, elevation, and interference from objects.

12 2-D Trilateration Location Tests

Every test below was conducted using the Arduino Tag, Reader, Coordinator, and Mega codes in Appendix C and the Location System Matlab Code in Appendix B. The coordinator is reader 2, which is connected to the Arduino Mega. Each of these tests is performed the same way: the coordinator will send a signal to the readers and reset the RSSI array, coordinator will send a signal to tag, tag will send 150 packets to each reader and delay 1 second between every 50 packet burst cycle, and RSSI measurements will be sent to coordinator and subsequently sent to Matlab through the Arduino Mega.

12.1 Test 1

This test was located at the parade grounds on university grounds. A 10 meter x 10 meter coordinate system was measured out and marked. After measuring out 10 meters and tuning the distance formula to the environment, the system was set up for 2-D trilateration. Reader 1 was placed at [10, 0], reader 2 was placed at [0, 10], and reader 3 was placed at [-10, 0]. A tag was placed at [0, 4] and the system was started from the host computer with the Matlab program in Appendix [x] running. This test was done at ground level and the antennas aimed orthogonally to the ground.

12.1.1 Results

The result of this test was inconclusive. The signal strength of signal strength of reader 3 was too low for the actual distance of the reader to the tag. The conclusion from this test is that the water system and pipes underground could be attenuated the signal from the tag to reader 3. The water meter for the parade grounds was directly in between the tag and reader 3.

12.2 Test 2

Test 1 was repeated except every reader and tag was raised into the air 5 feet, thus still 2-D trilateration. The location of the tag and readers did not change. The orientation of the antennas did not change. This was done to eliminate any external attenuation of the signal from being ground level.

12.2.1 Results

This test was a success. As you can see in figure 47, the tag was shown at [1, 4]. This is within 1 meter of the actual location of the tag and considered a success for our system. Comparing the actual result to the simulated result, the actual result is relatively close to the simulated, perfect result. The reason for this error is the error in the distance measurement of each reader. The error in the distance measurement is ~.5meters for each reader. These errors compound with each other when the trilateration algorithm is performed and provides an error of 1 meter in the tag location.

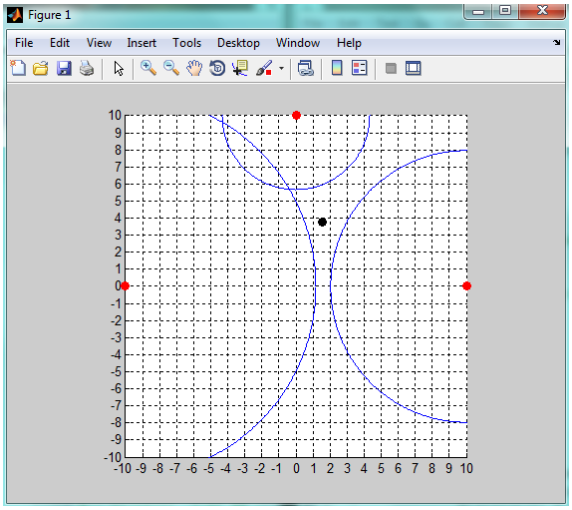


Figure 47: Test 2 Result

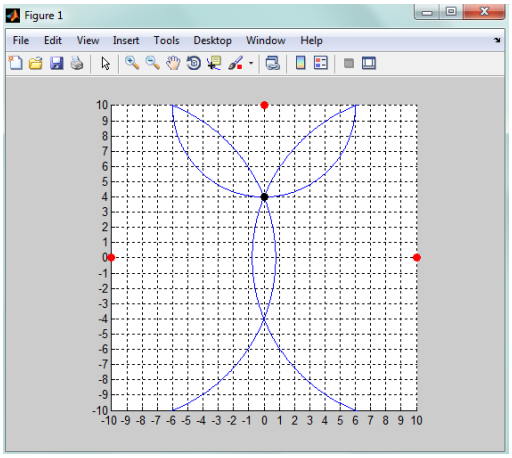


Figure 48: Test 2 Simulated Result

12.3 Test 3

Test 3 was conducted in the parking lot of the electrical engineering building. The temperature was 75 degrees Fahrenheit and light humidity. Reader 1 was set up at [0, 10], reader 2 at [-10,0], reader 3 at [10,0], and the tag was placed at the [0,0]. The readers were strapped to the reader stands facing towards the origin while the tag was held up at the vertical level of the readers with its antenna orthogonal to the ground. The distance formula was not yet tuned to the environment and the system was tested using the same distance formula as test 2. For this test, the RFID reader was integrated into the system. After the location process, the RFID card linked to the tag in the field would be scanned and the plot would appear.

12.3.1 Results

The distance formula not being tuned affected the results greatly. Since there was less humidity during the test than test 2, the signal strengths of each reader was smaller. Therefore, the distances were calculated to be shorter than they actually are (as seen in figure 49). The small error in the calculated location of the tag is an unexpected result since the distance formula calculated smaller distances than the actual distances. The RFID integration was a success since the plot below only appeared after the RFID card was scanned and its ID sent to the Matlab program.

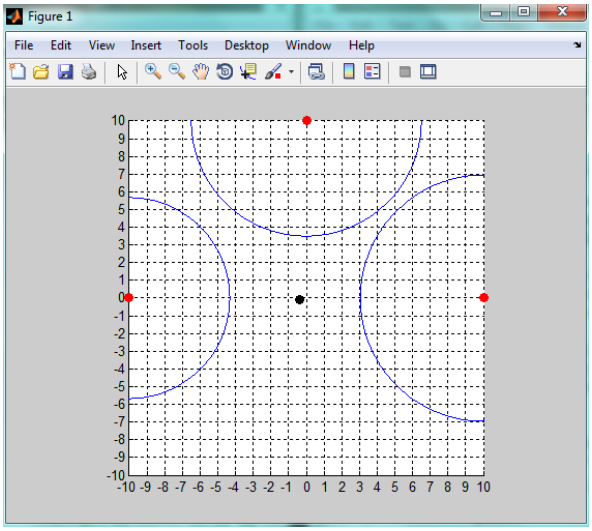


Figure 49: Test 3 Results

12.4 Test 4

Test 3 was repeated with the distance formula tuned to the environment.

12.4.1 Results

This test was a success. Once the distance formula was tuned to the environment, the distances calculated were relatively close. Reader 1 and 2 calculated a distance of 12 meters between itself and the tag instead of 10 meters. This caused the error in the calculated position of the tag of 1 meter, which is acceptable for this test. Comparing the two figures below, one can see the accuracy of the actual results against the simulated results.

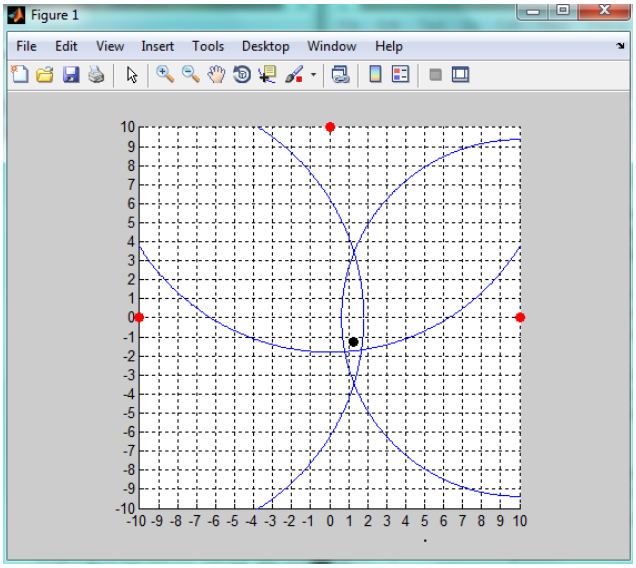


Figure 50: Test 4 Results

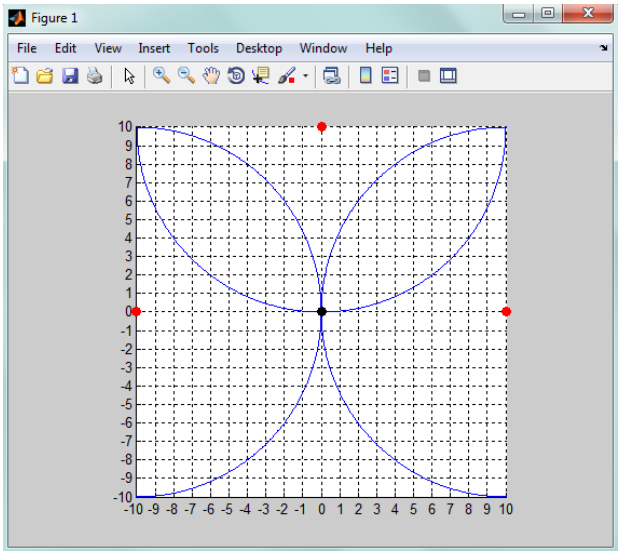


Figure 51: Test 4 Simulated Results

12.5 Test 5

In this test, the tag was moved to [2, 4] and the system was once again activated. The tag orientation is the same as the previous 2 tests.

12.5.1 Results

The result was this test was once again a success. The tag was calculated to be at [1, 3], which gives an error of 1 meter from the actual location. This was caused by reader 2's RSSI reading having a 1 meter error in its calculation.

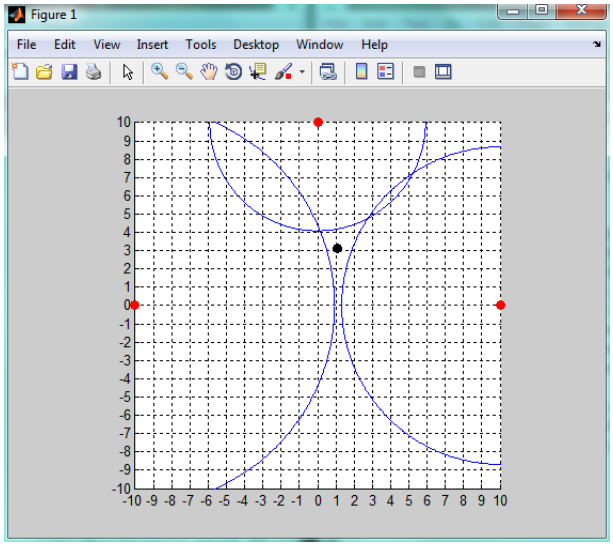


Figure 52: Test 5 Results

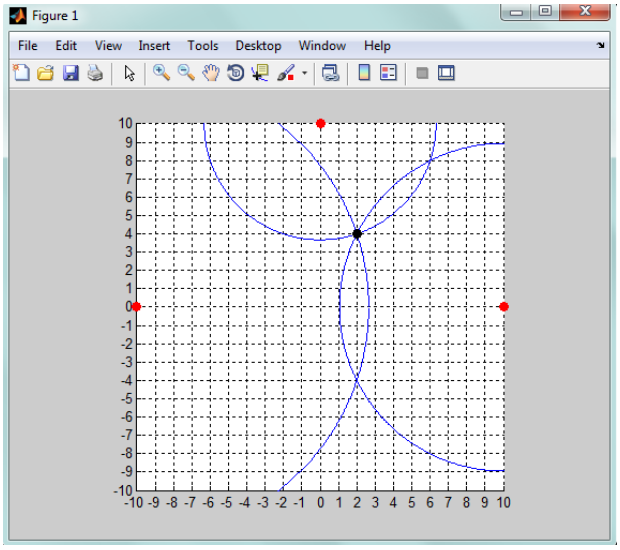


Figure 53: Test 5 Simulated Results

12.6 Indoor Test 1

This test was conducted at Zoar Baptist Church's basketball indoor gymnasium. Reader 1 was located at [0, 10], reader 2 at [-10, 2], and reader 3 at [10, 2]. The tag was located at [0, 5] and had the same orientation as previous tests 3 through 5. The indoor environment included metal bleachers surrounding the basketball court with metal basketball goals. A distance test was performed with a distance of 10 meters between tag and reader. Then, the distance formula was tuned to be correct for the indoor environment. Once the distance formula was tuned, the system was set up and tested.

12.6.1 Results

These indoor tests were inconclusive. The system provided the correct location of the tag about one-fourth of the time that the system was activated. For the rest of the tests done indoors, the tag was calculated to be anywhere from 3-5 meters away from its actual location. This could be caused by the interference of the metal bleachers and goals within the gymnasium. In figure 54 below, the best result of the group of tests indoors is shown.

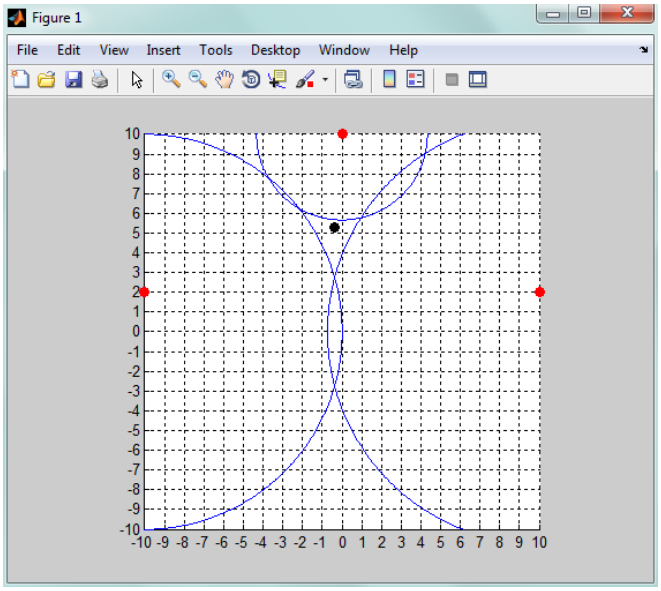


Figure 54: Indoor Test Results

13 Graphical User interface

13.1 Introduction to GUI

It was decided that having one would not only allow the opportunity to easily utilize and use the system, but also the end user once the system is up and operational. Ideally, the GUI should be able to operate on any working computer that has the capabilities of handling the software.

13.2 GUI layout

The GUI was given a basic layout that would be easy to use and view. For viewing purposes, the GUI was fitted with a gridded type layout along its X and Y-axes. A "Height" plane was also inserted into the GUI once it was decided to try 3-D trilateration. By this, the GUI's gridded screen layout became a 3-D representation of the field where the system will be set up and operational.

13.2.1 Layout design

Alongside a gridded system, the figure was also enabled to rotate in 3-D. This was to allow the user to move the figure in any direction they want in order for them to better view the area and where the object was in relation. One problem encountered was that during rotation, one would not be able to notice which side or face was which. Therefore, the axes were colored and labeled accordingly. The X-axis colored as red and the Y-axis colored as blue. The "Height" or Z-axis was left the same, black, in order to have a base of color and not to make the figure too colorful or pigmented. Figure 55 shows the resulting layout of the figure that is used in the GUI:

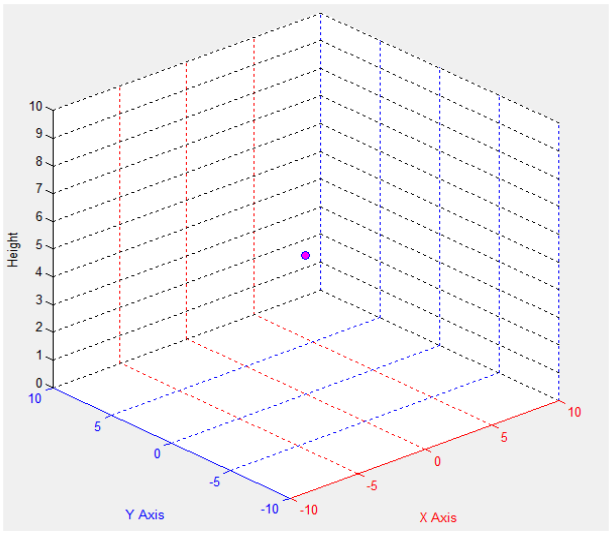


Figure 55: 3-D Figure

Each grid was labeled numerically in order to give the user an idea of distance. The numeric system is depicted in meters because the system is set to read in as meters. Because the system is meant to work indoor, 10 meters distance given for each axis, including the height or Z-axis. A dot was included into the layout to depict where, in the given, area the object was. Once the system begins to run, it will send the

information as to where the object is in the field. Once that information is sent into the GUI, the GUI's code will plot that objects location on the figure within the gridded field. In order to distinguish the dot, it was given a simple magenta color and blue outline. This was also done to allow it to stand out to the user. As the figure is rotated, the object will remain in its same relative space without moving or disappearing.

There was also a need to have a layout of the room and use it within the 3-D figure. To do this, a built in Matlab rectangle function was used. By using this function, rectangular objects would be able to depict tables, desk, and chairs. These rectangular objects would represent these pieces of furniture throughout a given room. The only issue with using this function is that it worked only in 2-D models, therefore not 3-D. Figure X shows the 2-D figure in which the rectangular pieces were used:

13.2.2 Information layout

Along with the rotation figure, a small information gathering area was added on its left. This area contained the spaces for the X, Y, and "Height"-axis. Once the system is online and triggered to begin working, the readers will find the tag in the field. Once found, their coordinates, X, Y, and "height", will be sent through to the GUI via MATLAB. The actual numeric information or each axis will be placed in its designated area. Information sent into these fields will be read in using a particular function that will be discussed later. Appendix X shows the code used in order to make this happen

13.2.3 Results

The end result is the rotatable figure being represented on the left and the information field containing the axis fields on the right. The layout was made simple in order to not over complicate anything or confuse the user with clutter. The main goal was to make sure the basics were in line and working before adding detail as far as its presentation was concerned.

13.3 GUI Built in Functions

The GUI comes equipped with preset functions in order to allow the programmer to program what they want to have happen once the GUI is called on or made into an .exe file. The user also has the ability to add functions to the GUI as such were added when the GUI was given X, Y, and Z-axis's. There's no set limit on the amount of functions that can be added to the interface in the making of a GUI.

13.3.1 Opening function

The main function in which was used to program the GUI was the opening function as shown in Code x:

```
function GUI_RFID_OpeningFcn(hObject, eventdata, handles, varargin)
```

Figure 56: Opening function for Matlab GUI

This function is the first function that gets called upon when the GUI opens. Therefore, everything that needed to happen with the figure and the information needed to go in this area. Within this function, the GUI was given its layout as well as the section where the information were to be placed. Appendix X shows the code that was used.

In order to read in the information from the Arduino's and Xbees, separate functions were included into the opening function. Those functions controlled the information flowing into the GUI as well as the distance configuration that measured the distances between the readers and the tag. These were placed in the opening function because the system is needed to work automatically; once the GUI was called on.

If the program were not configured to work this way, a push button command would be needed in order to enable the indoor system to operate. This became problematic, mainly because the system runs off of battery power. Meaning the system would have to remain on while draining power until called upon or activated by the push button. The push button method would drain the battery power causing the system to only operate for approximately days at a time. Hence, the automatic method was decided on.

13.3.2 Additional Built-in Functions

The GUI can have functions added to it by the programmer in the initial stage of its building based on what they need. For this project, the GUI needed sections to read in information for the X, Y, and "Height"-axis. There was also a need for the points on the figure to be plotted. Because of this, six additional functions were created as shown in Code X:

```
function X_plot_Callback(hObject, eventdata, handles)
function X_plot_CreateFcn(hObject, eventdata, handles)
function Y_plot_Callback(hObject, eventdata, handles)
function Y_plot_CreateFcn(hObject, eventdata, handles)
function Height_plot_Callback(hObject, eventdata, handles)
function Height_plot_CreateFcn(hObject, eventdata, handles)
```

Figure 57: Functions for different elements of Matlab GUI

Each function is designed to handle the specifics of what they were made for. For instance, the function *X_plot* is designed to handle how the object is plotted along the x-axis. For this portion, the group's main task was being able to have the built in *X_plot* function read in a function that was built that will bring in the information from the Arduinos and Xbees.

13.4 Implementation into Location System

After 2D trilateration testing was completed, the only remaining step before the end of the semester was to implement the Matlab GUI into the existing location system. This is done by calling the GUI as a function and passing variables to the GUI as parameters. The GUI has a parameter named "varargin" which is an array which holds a variable number of input arguments. The variables passed from a Matlab program to the GUI will be held in this array. Using this knowledge, the calculated x-coordinate of the tag, y-coordinate of the tag, reader coordinate matrix, and distance matrix variables were passed to the GUI. The GUI will then plot and output information that is user-friendly. The code for this implementation can be seen in Appendix B (Matlab GUI) and Appendix C (Location System).

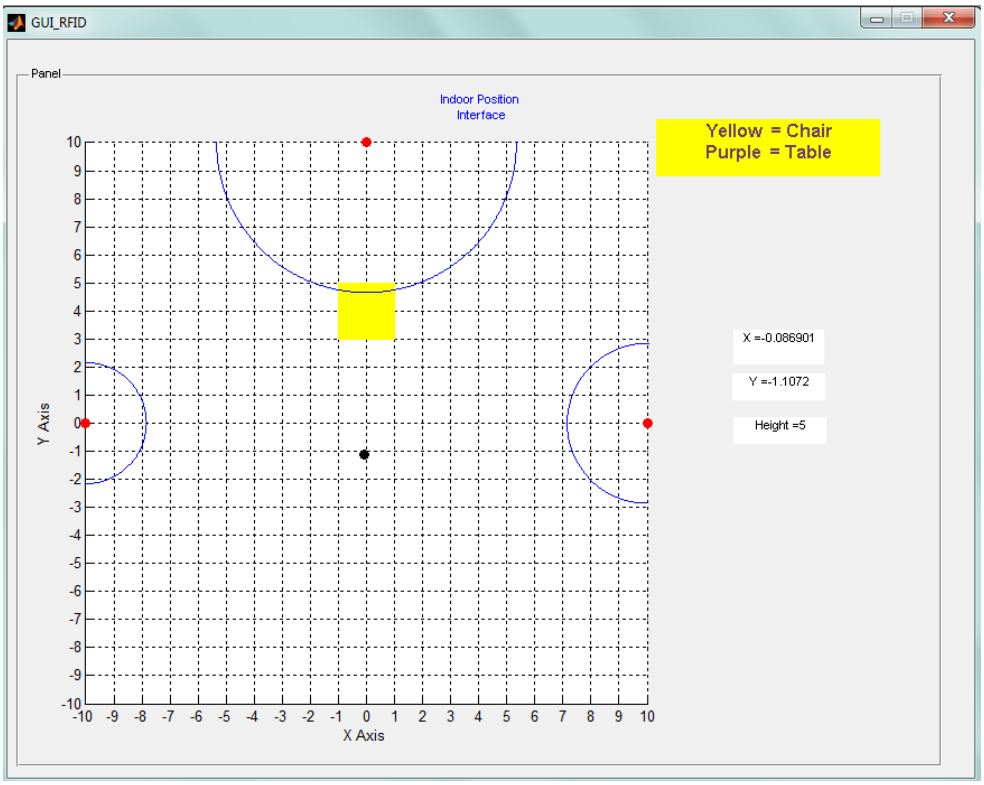


Figure 58: Example Implementation of Matlab GUI with Location System (not actual test)

14 Conclusion

Overall, the group considers our positioning system a success even though the system is inconsistent indoors. The goal of the project changed from indoor to outdoor during the year with the partnership with the Cameron group. Without their funds, this project so far would not have made any progress during the semester.

When the system is setup, calibrated, and tested outdoors, the accuracy of the location of the tag is within 1 meter, which is within our accuracy requirement. The system is completely user friendly by being a one click operation in Matlab and easy to use RFID card system. The tag is small and light weight for easy attachment to objects to be tracked. The system is easy to update and can easily handle up to 10 tags. The group met the marketing requirement by keeping the cost under 1000 dollars, as seen in Section 15. The system also meets the technical requirement of real time location since it can find the location of a tag in the field within 10 seconds of startup. These 10 seconds can be easily sped up through editing tag and coordinator code of the Arduino, which are currently slowed down for debugging.

The last technical requirement involves the battery life of the system. This is the one requirement of the project that did not get completed during the semester. The reason for this is the switch of hardware from Digimesh XBee modules to 802.15.4 (Series 1) modules. This switch of hardware types had to be made during the semester due to the flooding of a Digi factory in Thailand during the semester. The flood caused a shortage of modules with the RPSMA connector and the hardware switch had to be made to series 1 because they were currently in stock. If the group waited for the Digimesh RPSMA modules, the project would not have been completed. The series 1 modules do not have the synchronous sleep mode that was proposed to save the battery for the Project Design Review. In the future, the group would like a wakeup circuit calibrated for 2.4 GHz attached to the tag and used for battery life.

For the project moving forward, the group would like to see 3D trilateration for an outdoor environment. This would involve adding another reader and stand and testing how the RSSI is affected by the tag being higher/lower than the reader. Also, the Matlab GUI can replace the RFID card system and become even more user friendly by having a drop down menu listing all the tags to find. Another future goal of this project is to merge this project with the Cameron group's augmented reality using a SQL database to send coordinates of tags to their system.

Overall, this XBee location system is considered a reliable, working prototype for a location system used in an outdoor environment.

15 Budget

Below is a table detailing the group's budget during the semester. The 802.15.4 XBees, Titanis Antennas, and adapters were provided by the Cameron group funds, thus costing our group nothing. This collaboration was done because the Cameron group wanted to use our location system for their augmented reality. The antennas were left over pieces from the Texas Instruments location system used in the last year's Cameron group. Also, an Arduino Mega and 2 Arduino Unos that were previously owned by members of the group were used but not included in the budget.

Table 3: Budget for Parts Ordered

Product	Individual Price	Number Ordered	Total Price
Lithium Ion Polymer Batteries	$16.95	2	$33.90
Arduino Uno	$29.95	3	$89.85
Arduino Fio	$25.00	1	$25.00
Arduino Fio Cable	$20.00	1	$20.00
Maxim 8212 Chip	$3.75	2	$7.50
ID-12 Chip	$29.95	1	$29.95
RFID Card	$1.95	2	$3.90
DC Power Supplies	$1.90	6	$11.40
Project Enclosure Box	$5.95	1	$5.95
Digimesh XBee 2.4 GHz	$21.00	4	$84.00
Xbee Shield	$24.95	4	$99.80
802.15.4 Xbee 2.4 GHz(RPSMA)	$21.00	8	$168.00
Titanis Antenna	$30.00	8	$240.00
RPSMA to SMA Adapter	$6.99	8	$55.92
Total with Cameron support:			$408.25
Total:			$872.17

16 References:

"How to calculate battery run-time." *Notes for Design Engineers: How to calculate how much Battery capacity you need*. Power Stream, 28 Apr 2011. Web. 2 May 2011. <http://www.powerstream.com/battery-capacity-calculations.htm>.

"How to Prolong Lithium-based Batteries." *Battery University*. Isidor Buchmann, 02 May 2011. Web. 2 May 2011. <http://batteryuniversity.com/learn/article/how_to_prolong_lithium_based_batteries>.

"Lithium Polymer Charging/Discharging & Safety Information." *MaxAmps*. Traxxas, 02 May 2011. Web. 2 May 2011. <http://www.maxamps.com/lipo-care.php>.

N/A. September 23, 2008. MAX8211, MAX8212: Microprocessor Voltage Monitors with Programmable Voltage Detection. Maxim-ic. April 19, 2011 www.maxim-ic.com/datasheet/index.mvp/id/1273

Pistoia, G. *Batteries for Portable Devices*. 1st ed. Rome, Italy: Elsevier, 2005. 1-309. Print.

Shih, Chia-Yen, and Jose Marron. "COLA: Complexity-Reduced Trilateration Approach for 3D Localization In Wireless Sensor Networks." 2010 Fourth International Conference on Sensor Technologies and Applications. (2010): 1-32. Print.

XBee & XBee-PRO OEM RF Module Antenna Considerations. 2005. MaxStream. Sept. 2005 <ftp1.digi.com/support/images/XST-AN019a_XBeeAntennas.pdf>

Titanis 2.4 GHz Swivel SMA Antenna. 2011. Antenova. 11 Jan. 2011 <datasheet.elcodis.com/pdf/18/70/187086/b4844-01.pdf>

Appendix A: Datasheets

Titanis Antenna

antenova®

Titanis 2.4 GHz Swivel SMA Antenna

Part No. B4844 / B6090 Product Specification

1 Features

- Designed for 2.4 GHz applications: Bluetooth®, Wi-Fi® (802.11a/b/g/n), ZigBee®, etc. as well as 2.3 GHz WiMAX™, 2.5 GHz WiMAX™ and WiBro.
- Antenna with a SMA male connector
- Also available as SMA reverse thread to meet FCC regulations, part 15
- High efficiency
- Supplied in bulk

2 Description

Titanis is intended for use with all 2.4 GHz applications. The antenna is fitted with a SMA male connector and a blade made of flexible material that can be rotated 360 degree.

No external matching network required.

3 Applications

- Development tools
- Test equipment
- Access points, routers, etc
- Printers

Integrated Antenna and RF Solutions

4 Part number

Titanis Standard SMA – male: B4844

Titanis Reverse thread SMA – male: B6090

5 General data

Product name	Titanis 2.4 GHz
Part Number	B4844 (Standard SMA – male)
	B6090 (Reverse thread SMA – male)
Frequency	2.4 – 2.5 GHz
Polarization	Linear
Operating temperature	-40 ºC to +85 ºC
Impedance	50 Ω
Weight	7.1 g
Antenna type	[1]Swivel external
Dimensions	20 x 19.5 x 62.5 [mm]

[1]The blade of the antenna is the only part that swivels. DO NOT twist the plastic housing of the antenna blade. The housing is NOT designed to twist or turn and any attempt to do so will likely result in permanent damage to the antenna and its performance and will not be covered by warranty. Installation and removal of the antenna should only be done by turning the metal SMA connector.

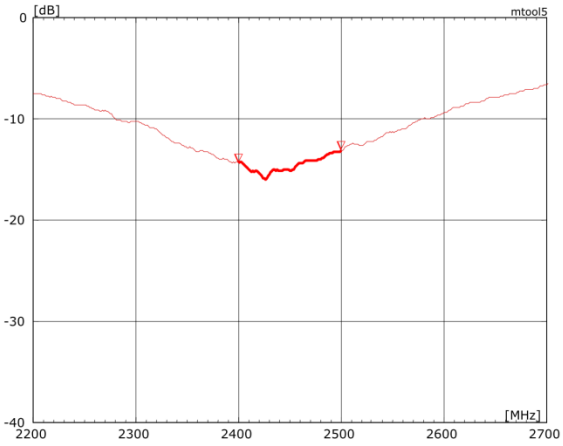
Titanis is not suitable for outdoor use or applications.

6 Electrical characteristics

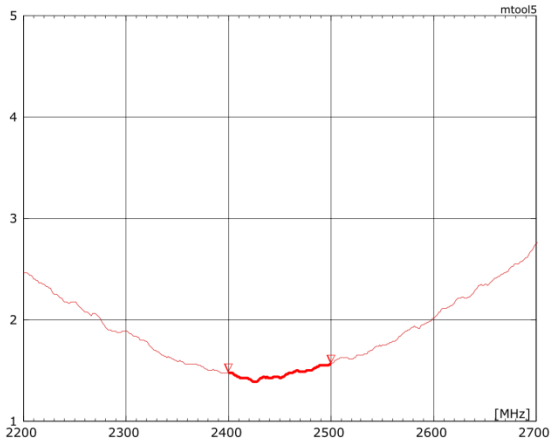
	Typical performance	Conditions
Peak gain	2.2 dBi	Data given for the 2.4 – 2.5 GHz frequency range
Average gain	-1.0 dBi	
Average efficiency	80%	
Maximum Return Loss	-13 dB	
Maximum VSWR	1.6:1	

7 Electrical performance

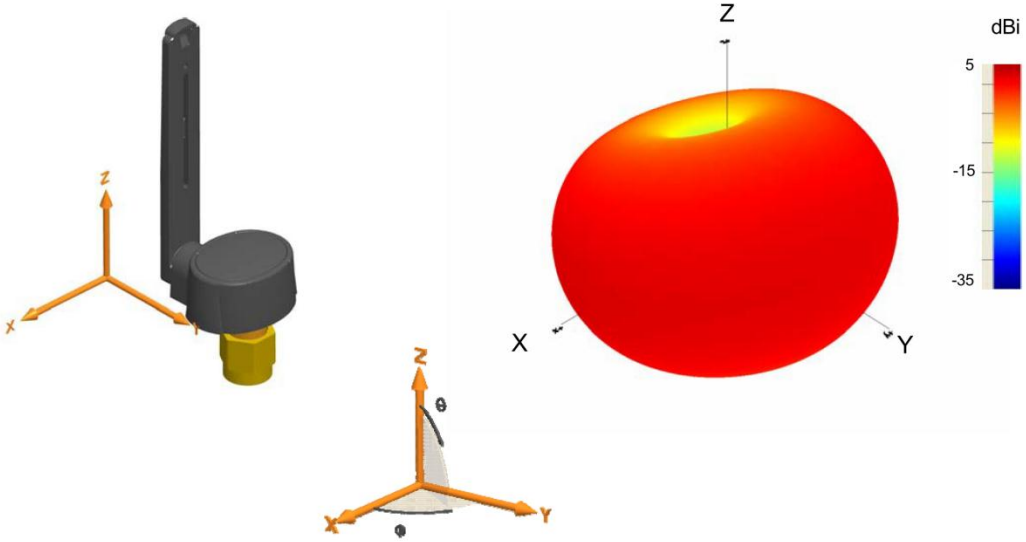
7-1 Return Loss

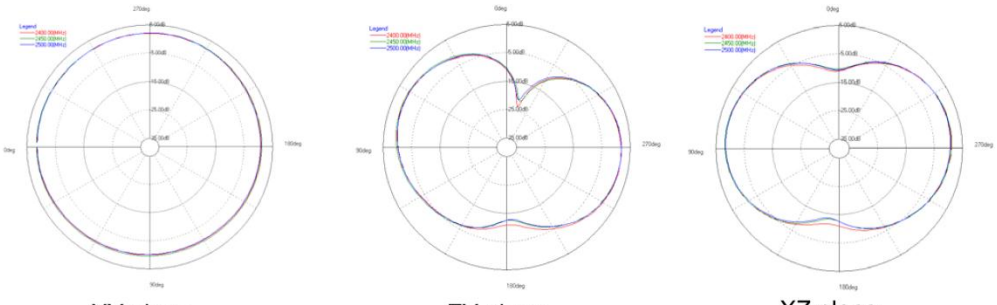


7-2 VSWR



7-3 Antenna patterns

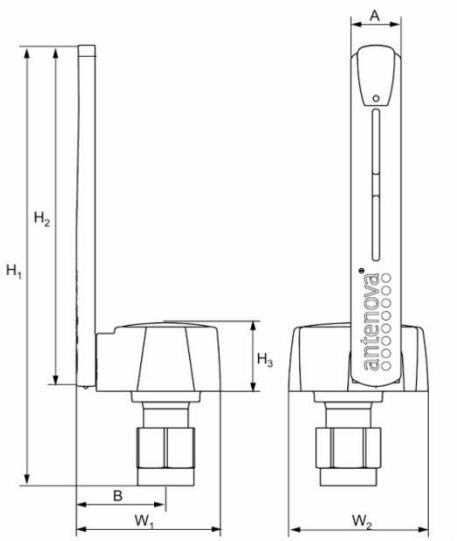


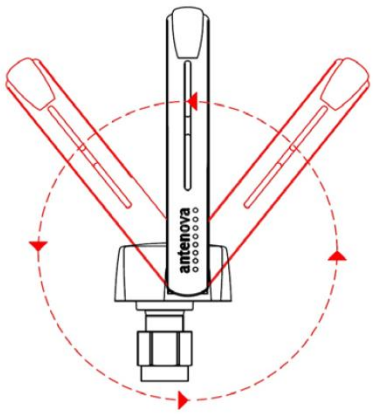


XY plane ZY plane XZ plane

Patterns show combined polarisations

8 Antenna dimensions





A	B	H1	H2	H3	W1	W2
		Height	Height	Height	Width	Width
7 ± 0.2	12.5 ± 0.5	62.5 ± 0.5	48.3 ± 0.5	9.5 ± 0.5	20 ± 0.3	19.5 ± 0.3

Dimensions in mm

Warning: The blade of the antenna is the only part that swivels. DO NOT twist the plastic housing of the antenna blade. The housing is NOT designed to twist or turn and any attempt to do so will likely result in permanent damage to the antenna and its performance and is not covered by warranty. Installation and removal of the antenna should only be done by turning the metal SMA connector.

9 Hazardous material regulation conformance

The antenna has been tested to conform to RoHS requirements. A certificate of conformance is available from Antenova's website.

10 Packaging

10-1 Optimal storage conditions

Temperature	-10ºC to 40ºC
Humidity	Less than 75% RH
Shelf Life	48 Months
Storage place	Away from corrosive gas and direct sunlight

10-2 Packaging information

The antennas are delivered in bulk, enclosed in plastic bags.

10-3 Bag label information

Dimensions in mm

www.antenova.com

Corporate Headquarters

Antenova Ltd.
Far Field House
Albert Road
Stow-cum-Quy
Cambridge
CB25 9AR

Tel: +44 1223 810600
Fax: +44 1223 810650
Email: info@antenova.com

North America Headquarters

Antenova Ltd.
Rogers Business Park
2541 Technology Drive Suite 403
Elgin, IL 60124

Tel: +1 (847) 551 9710
Fax +1 (847) 551 9719
Email: info@antenova.com

Asia Headquarters

Antenova Asia Ltd.
4F, No. 324, Sec. 1, Nei-Hu Road
Nei-Hu District
Taipei 11493
Taiwan, ROC

Tel: +886 (0) 2 8797 8630
Fax: +886 (0) 2 8797 6890
Email: info@antenova.com

ISOQAR QUALITY ASSURED ISO 9001 UKAS QUALITY MANAGEMENT 026
Certificate No: 4598

Integrated Antenna and RF Solutions

585460

LI-POLYMER BATTERY PACKS

Specification

Type:585460 2000mAh

Prepared/Date	Auditing/Date	Approved/Date
WANG MAR 16, 2006	LI MAR 16, 2006	XIONG MAR 16, 2006

UNIONFORTUNE PRODUCT SPECIFICATION	Doc. No.	2006.3.16
	Edition No.	2.0
	Sheet	1/5

1 Scope

This product specification describes UNIONFORTUNE polymer lithium-ion battery. Please using the test methods that recommend in this specification. If you have any opinions or advices about the test items and methods, please contact us. Please read the cautions recommended in the specifications first, take the credibility measure of the cell's using.

2 Product Type, Model and Dimension

2.1 Type Polymer lithium-ion battery

2.2 Model 585460

2.3 Cell Dimension(Max, Thickness×Width×Length mm^3) 5.8×54×60

Pack Dimension(Max, Thickness×Width×Length mm^3) None

3 Specification

Item		Specifications	Remark
Nominal Capacity		2000 mAh	0.2C_5A discharge
Nominal Voltage		3.7V	Average Voltage at 0.2C_5A discharge
Charge Current		Standard 0.2 C_5A Max 1C_5A	Working temperature 0 40
Charge cut-off Voltage		4.20±0.03V	
Standard Discharge Current		0.2C_5A	Working temperature -20 60
Max Discharge Current		2.0C_5A	Working temperature 0 60
Discharge cut-off Voltage		2.75 V	
Cell Voltage		3.7-3.9 V	When leave factory
Impedance		≤ 300 mΩ	AC 1KHz after 50% charge
Weight		Approx: 37g	
Storage temperature	≤1month	-20 45	Best 20±5 for long-time storage
	≤3month	0 30	
	≤6month	20±5	
Storage humidity		65±20% RH	

4 General Performance

Definition of Standard charging method At 20±5 charging the cell initially with constant current 0.2C_5A till voltage 4.2V, then with constant voltage 4.2V till current declines to 0.05C_5A.

Item		Test Methods	Performance
4.1	0.2C Capacity	After standard charging, laying the battery 0.5h, then discharging at 0.2C_5A to voltage 2.75V, recording the discharging time.	≥300min
4.2	1C Capacity	After standard charging, laying the battery 0.5h, then discharging at 1C_5A to voltage 2.75V, recording the discharging time.	≥51min
4.3	Cycle Life	Constant current 1C_5A charge to 4.2V, then constant voltage charge to current declines to 0.05C_5A, stay 5min constant current 1C_5A discharge to 2.75V stay 5min. Repeat above steps till continuously discharging time less than 36min.	≥300times
4.4	Capability of keeping electricity	20±5 , After standard charging, laying the battery 28days, discharging at 0.2C_5A to voltage 2.75V, recording the discharging time.	≥240min

5 Environment Performance

Item		Test Methods	Performance
5.1	High temperature	After standard charging, laying the battery 4h at 60 , then discharging at 0.2C_5A to voltage 2.75V, recording the discharging time.	≥270min
5.2	Low temperature	After standard charging, laying the battery 4h at 0.2C_5A, then discharging at 0.2C_5A to voltage 2.75V, recording the discharging time.	≥210min
5.3	Constant humidity and temperature	After standard charging, laying the battery 48h at 40±2 , RH 93±2%. Recording 0.2C_5A discharging time	No distortion No electrolytes leakage ≥270 min
5.4	Temperature shock	After standard charging, battery stored at -20 for 2 hours, then stored at 50 for 2 hours. Repeat 10 times.	No electrolytes leakage

6 Mechanical Performance

Item		Test Methods	Performance
6.1	Vibration	After standard charging, put battery on the vibration table. 30 min experiment from X,Y,Z axis. Scan rate: 1 oct/min; Frequency 10-30Hz, Swing 0.38mm; Frequency 30-55Hz, Swing 0.19mm.	No influence to batteries' electrical performance and appearance.
6.2	Collision	After vibration test, batteries were laying on the vibration table about X, Y, Z axis. Max frequency acceleration: 100m/s^2; collision times per minutes: 40~80; frequency keeping time 16ms; all collision times 1000±10.	No influence to batteries' electrical performance and appearance.
6.3	Drop	Random drop the battery from 10m height onto concrete one times.	No explosion or fire

7 Safety Test

Test conditions The following tests must be measured at flowing air and safety protection conditions. All batteries must standard charge and lay 24h.

Item		Test Methods	Performance
7.1	Over charge	At 20±5 , charging batteries with constant current 3C_5A to voltage 4.8V, then with constant voltage 4.8V till current decline to 0. Stop test till batteries' temperature 10 lower than max temperature.	No explosion or fire
7.2	Over discharge	At 20±5 , discharge battery with 0.2C_5A continuously 12.5h.	No explosion or fire
7.3	Short-circuit	At 20±5 , connect batteries' anode and cathode by wire which impedance less than 50mΩ, keep 6h.	No explosion or fire
7.4	Extrusion	At 20±5 , put the battery in two parallel steal broad, add pressure 13kN.	No explosion or fire
7.5	Thermal shock	Put the battery in the oven. The temperature of the oven is to be raised at 5±1 per minute to a temperature of 130±2 and remains 60 minutes.	No explosion or fire

8 Cautions

1. **Cautions of batteries' operation**

The batteries must be careful of proceed the operation for it's soft package.

Aluminum packing materials

The aluminum packing material was easily damaged by the sharp edge part, such as nickel-tabs.

forbid to use the sharp part touching the battery;

should cleaning working condition, avoiding the sharp edge part existence;

forbid to pierce the battery with nail and other sharp items;

the battery was forbidden with metal, such as necklace, hairpin etc in transportation and storage.

Sealed edge

Sealing edge is very easily damaged and don't bend it.

The Al interlayer of package has good electric performance. It's forbidden to connect with exterior component for preventing short-circuits.

Folding edge

The folding edge is formed in batteries' processes and passed all hermetic tests, don't open or deform it. The Al interlayer of package has good electric performance. It's forbidden to connect with exterior component for preventing short-circuits.

Tabs

The batteries' tabs are not so stubborn especially for aluminum tabs. Don't bend tabs.

Mechanical shock

Don't fall, hit, bent the batteries' body.

Short-circuit

Short-circuit is strictly prohibited. It should damage batteries badly.

2. **Standard Test Environment for polymer lithium-ion batteries**

Environment temperature: 20±5

Humidity: 45-85%

3. Cautions of charge & discharge

charge

Charging current should be lower than values that recommend below. Higher current and voltage charging may cause damage to cell electrical, mechanical, safety performance and could lead heat generation or leakage.

Batteries charger should charging with constant current and constant voltage mode;

Charging current should be lower than (or equal to)1C_5A;

Temperature 0 40 is preferred when charging;

Charging voltage must be lower than 4.25V.

discharge

Discharging current must be lower than (or equal to)2C_5A;

Temperature 0 60 is preferred when discharging;

Discharging voltage must not be lower than 2.75V.

over-discharge

It should be noted that the cell would be at an over-discharge state by its self-discharge. In order to prevent over-discharge, the cell shall be charged periodically to keeping voltage between 3.6-3.9V. Over-discharge may cause loss of cell performance. It should be noted that the cell would not discharge till voltage lower than 2.5V.

4.Storage of polymer lithium-ion batteries

The environment of long-time storage:

Temperature: 20±5 ;

Humidity: 45-85%;

Batteries were 40 60% charged.

5.Transportation of polymer lithium-ion batteries

The batteries should transportation with 10 50% charged states.

6.Others

Please note cautions below to prevent cells' leakage, heat generation and explosion.

Prohibition of disassembly cells;

Prohibition of cells immersion into liquid such as water or seawater;

Prohibition of dumping cells into fire;

Prohibition of using damaged cells. The cells with a smell of electrolyte or leakage must be placed away from fire to avoid firing.

In case of electrolyte leakage contact with skin, eye, physicians shall flush the electrolyte immediately with fresh water and medical advise is to be sought.

9. Notice of Designing Battery Pack

9.1 Pack design

Battery pack should have sufficient strength and battery should be protected from mechanical shock. No sharp edge components should be inside the pack contain the battery.

9.2 PCM design

The overcharge threshold voltage should not be exceed 4.25V.

The over-discharge threshold voltage should not be lower than 2.75V.

The PCM should have short protection function built inside.

9.3 Tab connection

Ultrasonic welding or spot welding is recommended to connect battery with PCM or other parts.

If apply manual solder method to connect tab with PCM, the notice below is very important to ensure battery performance.

The electric iron should be temperature controlled and ESD safe;

Soldering temperature should not exceed 350 ;

Soldering time should not be longer than 3s, keep battery tab cold down before next soldering;

Soldering times should not exceed 5 times;

Directly heat cell body is strictly prohibited, battery may be damaged by heat above approx. 100 .

9.4 Cell fixing

The battery should be fixed to the battery pack by its large surface area. No cell movement in the battery pack should be allowed.

9.5 Cells replacement

The cell replacement should be done by professional people.

Prohibit short-circuit between cells' Al package and exterior component.

10. Cell Drawing:

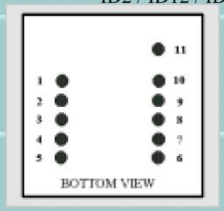
ID-12 RFID Detection Device

ID SERIES DATASHEET MAR 01, 2005

ID-2/ID-12 Brief Data

The ID2. ID12 and ID20 are similar to the obsolete ID0, ID10 and ID15 MK(ii) series devices, but they have extra pins that allow Magnetic Emulation output to be included in the functionality. The ID-12 and ID-20 come with internal antennas, and have read ranges of 12+ cm and 16+ cm, respectively. With an external antenna, the ID-2 can deliver read ranges of up to 25 cm. All three readers support ASCII, Wiegand26 and Magnetic ABA Track2 data formats.

ID2 / ID12 / ID20 PIN-OUT



1. GND
2. RES (Reset Bar)
3. ANT (Antenna)
4. ANT (Antenna)
5. CP
6. Future
7. +/- (Format Selector)
8. D1 (Data Pin 1)
9. D0 (Data Pin 0)
10. LED (LED / Beeper)
11. +5V

Operational and Physical Characteristics

Parameters	ID-2	ID-12	ID-20
Read Range	N/A (no internal antenna)	12+ cm	16+ cm
Dimensions	21 mm x 19 mm x 6 mm	26 mm x 25 mm x 7 mm	40 mm x 40 mm x 9 mm
Frequency	125 kHz	125 kHz	125 kHz
Card Format	EM 4001 or compatible	EM 4001 or compatible	EM 4001 or compatible
Encoding	Manchester 64-bit, modulus 64	Manchester 64-bit, modulus 64	Manchester 64-bit, modulus 64
Power Requirement	5 VDC @ 13mA nominal	5 VDC @ 30mA nominal	5 VDC @ 65mA nominal
I/O Output Current	+/-200mA PK	-	-
Voltage Supply Range	+4.6V through +5.4V	+4.6V through +5.4V	+4.6V through +5.4V

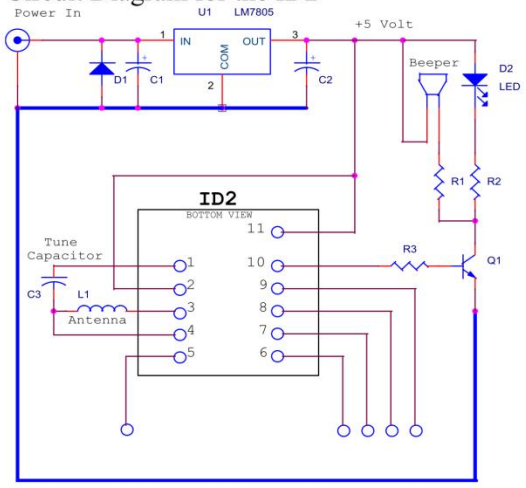
Pin Description & Output Data Formats

Pin No.	Description	ASCII	Magnet Emulation	Wiegand26
Pin 1	Zero Volts and Tuning Capacitor Ground	GND 0V	GND 0V	GND 0V
Pin 2	Strap to +5V	Reset Bar	Reset Bar	Reset Bar
Pin 3	To External Antenna and Tuning Capacitor	Antenna	Antenna	Antenna
Pin 4	To External Antenna	Antenna	Antenna	Antenna
Pin 5	Card Present	No function	Card Present *	No function

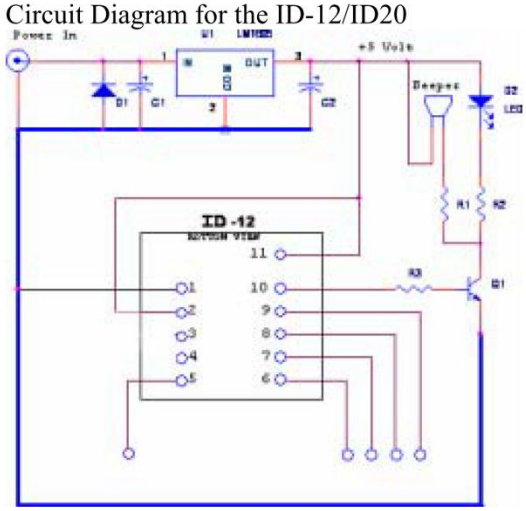
Pin 6	Future	Future	Future	Future
Pin 7	Format Selector (+/-)	Strap to GND	Strap to Pin 10	Strap to +5V
Pin 8	Data 1	CMOS	Clock *	One Output *
Pin 9	Data 0	TTL Data (inverted)	Data *	Zero Output *
Pin 10	3.1 kHz Logic	Beeper / LED	Beeper / LED	Beeper / LED
Pin 11	DC Voltage Supply	+5V	+5V	+5V

* Requires 4K7 Pull-up resistor to +5V

Circuit Diagram for the ID2

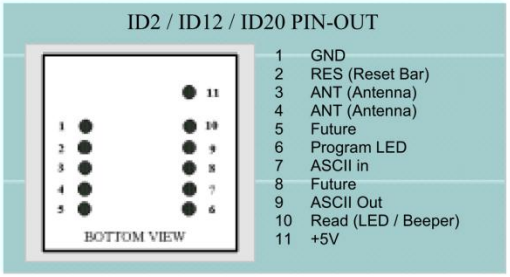


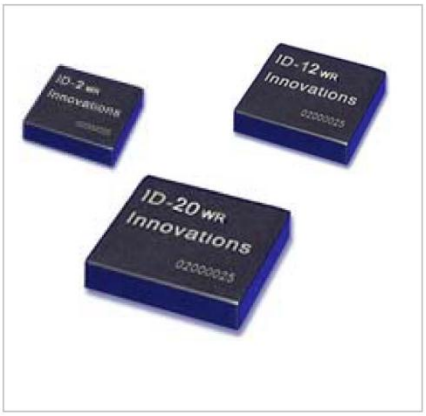
Circuit Diagram for the ID-12/ID20



ID-2RW/ID-12RW Brief Data

The ID2-RW, ID12-RW and ID15-RW are a new series of Read/Write modules for the Temec Q5 tag. It has full functionality including password. They contain built-in algorithms to assist customers programming the popular Sokymat Unique type tag. Password protection is allowed. Control is via a host computer using a simple terminal program such as hyper terminal or Qmodem.





Operational and Physical Characteristics

Parameters	ID-2RW	ID-12RW	ID-20RW
Read Range	N/A (no internal antenna)	12+ cm (Unique Format)	15+ cm (Unique Format)
Dimensions	21 mm x 19 mm x 6 mm	26 mm x 25 mm x 7 mm	40 mm x 40 mm x 9 mm
Frequency	125 kHz	125 kHz	125 kHz
Card Format	Temec Q5555	Temec Q5555	Temec Q5555
Read Encoding	Manchester modulus 64	Manchester modulus 64	Manchester modulus 64
Power Requirement	5 VDC @ 13mA nominal	5 VDC @ 30mA nominal	5 VDC @ 50mA nominal
I/O Output Current	+/-200mA PK	-	-
Voltage Supply Range	+4.6V through +5.4V	+4.6V through +5.4V	+4.6V through +5.4V
Coil Detail	L = 0.6mH - 1.5mH, Q = 15-30	-	-

Description

A simple terminal program such as Qmodem or Hyper-terminal can be used to send commands to the module. The blocks are individually programmable. The command interface is simple to use and easily understood. The programmer also has two types of internal reader. One of these is provided to read Sokymat 'Unique' type tag configuration. The module does not require a MAX232 type chip interface. The module does **not** need an RS232 interface such as a MAX232 IC. The input pin7 goes to the computer through a 4k7 resistor and the output goes to the computer through a 100R resistor.

DATA FORMATS

Output Data Structure – ASCII

STX (02h)	DATA (10 ASCII)	CHECK SUM (2 ASCII)	CR	LF	ETX (03h)

[The 1byte (2 ASCII characters) Check sum is the "Exclusive OR" of the 5 hex bytes (10 ASCII) Data characters.]

Output Data Structure – Wiegand26

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
P	E	E	E	E	E	E	E	E	E	E	E	E	O	O	O	O	O	O	O	O	O	O	O	O	P
Even parity (E)													Odd parity (O)												

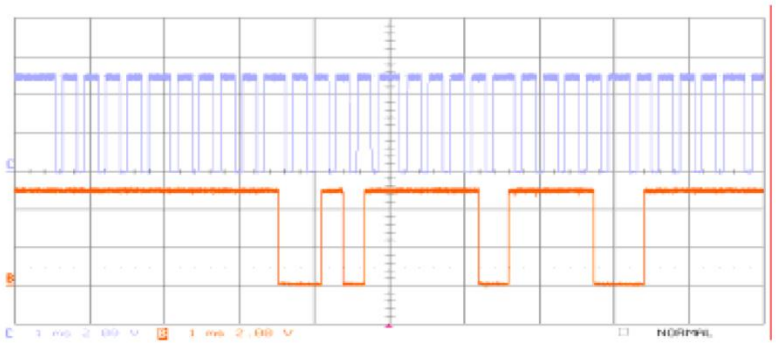
P = Parity start bit and stop bit

Output Data Magnetic ABA Track2

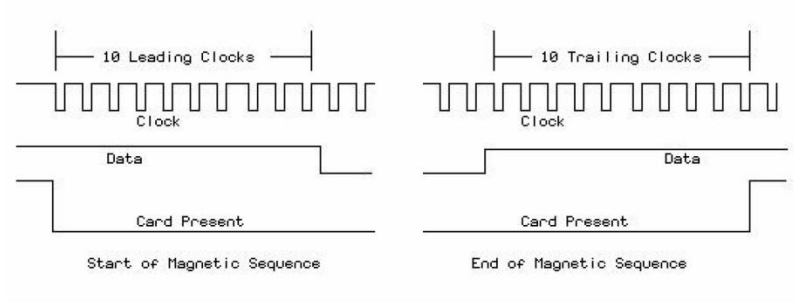
10 Leading Zeros	SS	Data	ES	LCR	10 Ending Zeros

[SS is the Start Character of 11010, ES is the end character of 11111, LRC is the Longitudinal Redundancy Check.]

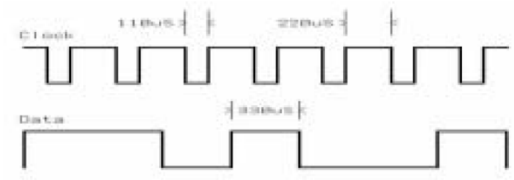
Magnetic Emulation Waveforms



Start and End Sequences For Magnetic Timing

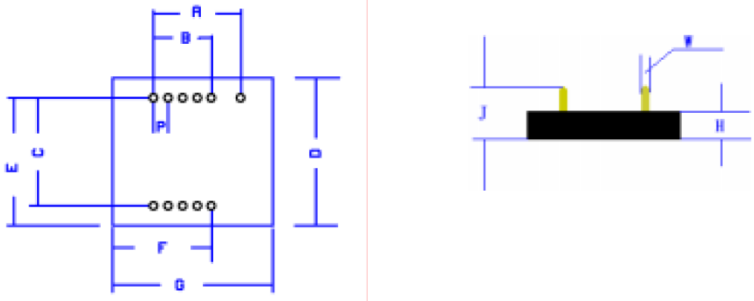


DATA TIMINGS FOR MAGNETIC EMULATION



The magnetic Emulation Sequence starts with the Card Present Line going active (down). There next follows 10 clocks with Zero '0' data. At the end of the 10 leading clocks the start character (11010) is sent and this is followed by the data. At the end of the data the end character is sent followed by the LCR. Finally 10 trailing clocks are sent and the card present line is raised.
The data bit duration is approximately 330uS. The approximate clock duration is 110uS. Because of the symmetry data can be clocked off either the rising or falling edge of the clock.

Dimensions (Top View) (mm)



	ID-0/ID-2wr			ID-10/ID-12wr			ID-15/ID-20wr		
	Nom.	Min.	Max.	Nom.	Min.	Max.	Nom.	Min.	Max
A	12.0	11.6	12.4	12.0	11.6	12.4	12.0	11.6	12.4
B	8.0	7.6	8.4	8.0	7.6	8.4	8.0	7.6	8.4
C	15.0	14.6	15.4	15.0	14.6	15.4	15.0	14.6	15.4
D	20.5	20.0	21.5	25.3	24.9	25.9	40.3	40.0	41.0
E	18.5	18.0	19.2	20.3	19.8	20.9	27.8	27.5	28.5
F	14.0	13.0	14.8	16.3	15.8	16.9	22.2	21.9	23.1
G	22.0	21.6	22.4	26.4	26.1	27.1	38.5	38.2	39.2
P	2.0	1.8	2.2	2.0	1.8	2.2	2.0	1.8	2.2
H	5.92	5.85	6.6	6.0	5.8	6.6	6.8	6.7	7.0
J	9.85	9.0	10.5	9.9	9.40	10.5	9.85	9.4	10.6
W	0.66	0.62	0.67	0.66	0.62	0.67	0.66	0.62	0.67

Note – measurements do not include any burring of edges.
NOTICE - Innovated Devices reserve the right to change these specifications without prior notice.

Designing Coils for ID2

The recommended Inductance is 1.08mH to be used with an internal tuning capacitor of 1n5. In general the bigger the antenna the better, provided the reader is generating enough field strength to excite the tag. The ID-2 is relatively low power so a maximum coil size of 15x15cm is recommended if it is intended to read ISO cards. If the reader is intended to read glass tags the maximum coil size should be smaller, say 10x10cm.

There is a science to determine the exact size of an antenna but there are so many variables that in general it is best to get a general idea after which a degree of 'Try it and see' is unavoidable.

If the reader is located in a position where there is a lot of heavy interference then less range cannot be avoided. In this situation the coil should be made smaller to increase the field strength and coupling.

It is difficult to give actual examples of coils for hand wounding because the closeness and tightness of the winding will significantly change the inductance. A professionally wound coil will have much more inductance than a similar hand wound coil.

For those who want a starting point into practical antenna winding it was found that 63 turns on a 120mm diameter former gave an inductance of 1.08mH. For those contemplating adding an additional tuning capacitor it was found that 50 turns on a 120mm diameter former gave 700uH. The wire diameter is not important.
Anybody who wishes to be more theoretical we recommend a trip to the Microchip Website where we found an application sheet for Loop Antennas.
http://ww1.microchip.com/downloads/en/AppNotes/00831b.pdf

The Tuning Capacitor

It is recommended that the internal 1n% capacitor is used for tuning, however a capacitor may be also be added externally. The combined capacitance should not exceed 2n7. Do not forget that the choice of tuning capacitor can also substantially affect the quality of your system. The Id12 is basically an ID2 with an internal antenna. The loss in an ID12 series antenna is required to be fairly high to limit the series current. A low Q will hide a lot of the shortcomings of the capacitor, but for quality and reliability and repeatability the following capacitors are recommend.

Polypropylene	Good Readily available. Ensure AC voltage at 125kHz is sufficient.
COG/NPO	Excellent. Best Choice
Silver Mica	Excellent but expensive
Polycarbonate	Good Readily available. Ensure AC voltage at 125kHz is sufficient.

Voltage Working.

A capacitor capable of withstanding the RMS voltage at 125KHz MUST be chosen. The working voltage will depend on the coil design. I suggest the designer start with rugged 1n5 Polypropylene 630v capacitor to do his experiments and the come down to a suitable size/value. The capacitor manufacturer will supply information on their capacitors. Do not simply go by the DC voltage. This means little. A tolerance of 2% is preferable. A tolerance of 5% is acceptable.

Fine Tuning

We recommend using an oscilloscope for fine-tuning. Connect the oscilloscope to observe the 125KHz AC voltage across the coil. Get a sizeable piece of ferrite and bring it up to the antenna loop. If the voltage increases then you need more inductance (or more capacitance). If the voltage decreases as you bring the ferrite up to the antenna then the inductance is too great. If you have no ferrite then a piece of aluminum

sheet may be used for testing in a slightly different way. Opposing currents will flow in the aluminum and it will act as a negative inductance. If the 125kH AC voltage increases as the aluminum sheet approaches the antenna then the inductance is too high. Note it may be possible that the voltage will first maximize then decrease. This simply means that you are near optimum tuning. If you are using ferrite then the coil is a little under value and if you are using an aluminum sheet then the coil is a over under value.

ID Innovations
Advanced Digital Reader Technology
----Better by Design

19-0539; Rev 3; 1/95

MAXIM

Microprocessor Voltage Monitors with Programmable Voltage Detection

General Description

Maxim's MAX8211 and MAX8212 are CMOS micropower voltage detectors that warn microprocessors (µPs) of power failures. Each contains a comparator, a 1.5V bandgap reference, and an open-drain N-channel output driver. Two external resistors are used in conjunction with the internal reference to set the trip voltage to the desired level. A hysteresis output is also included, allowing the user to apply positive feedback for noise-free output switching.

The MAX8211 provides a 7mA current-limited output sink whenever the voltage applied to the threshold pin is less than the 1.5V internal reference. In the MAX8212, a voltage greater than 1.5V at the threshold pin turns the output stage on (no current limit).

The CMOS MAX8211/MAX8212 are plug-in replacements for the bipolar ICL8211/ICL8212 in applications where the maximum supply voltage is less than 16.5V. They offer several performance advantages, including reduced supply current, a more tightly controlled bandgap reference, and more available current from the hysteresis output.

Applications

µP Voltage Monitoring
Undervoltage Detection
Overvoltage Detection
Battery-Backup Switching
Power-Supply Fault Monitoring
Low-Battery Detection

Features

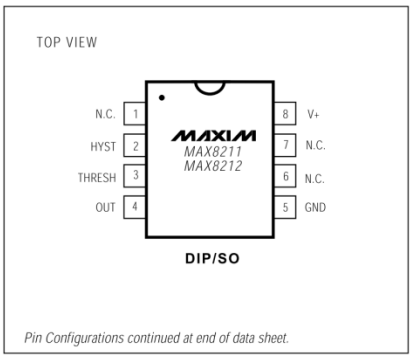
- **µP Power-Fail Warning**
- **Improved 2nd Source for ICL8211/ICL8212**
- **Low-Power CMOS Design**
- **5µA Quiescent Current**
- **On-Board Hysteresis Output**
- **±40mV Threshold Accuracy (±3.5%)**
- **2.0V to 16.5V Supply-Voltage Range**
- **Define Output Current Limit (MAX8211)**
- **High Output Current Capability (MAX8212)**

Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX8211CPA	0°C to +70°C	8 Plastic DIP
MAX8211CSA	0°C to +70°C	8 SO
MAX8211CUA	0°C to +70°C	8 µMAX
MAX8211CTY	0°C to +70°C	8 TO-99
MAX8211EPA	-40°C to +85°C	8 Plastic DIP
MAX8211ESA	-40°C to +85°C	8 SO
MAX8211EJA	-40°C to +85°C	8 CERDIP
MAX8211ETY	-40°C to +85°C	8 TO-99
MAX8211MJA	-55°C to +125°C	8 CERDIP
MAX8211MTV	-55°C to +125°C	8 TO-99

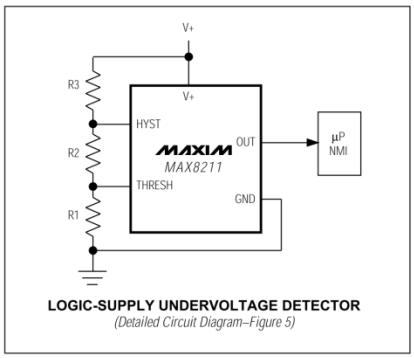
Ordering Information continued on last page.
** Contact factory for dice specifications.*

Pin Configuration



Pin Configurations continued at end of data sheet.

Typical Operating Circuit



LOGIC-SUPPLY UNDERVOLTAGE DETECTOR
(Detailed Circuit Diagram–Figure 5)

Microprocessor Voltage Monitors with Programmable Voltage Detection

ABSOLUTE MAXIMUM RATINGS

Supply Voltage .....-0.5V to +18V
Output Voltage .....-0.5V to +18V
Hysteresis.....+0.5V to -18V with respect to (V+ + 0.5V)
Threshold Input Voltage .....-0.5V to (V+ + 0.5V)
Current into Any Terminal .....±50mA
Continuous Power Dissipation (T_A = +70°C)
Plastic DIP (derate 9.09mW/°C above +70°C) 727mW
SO (derate 5.88mW/°C above +70°C).....471mW
CERDIP (derate 8.00mW/°C above +70°C).....640mW
TO-99 (derate 6.67mW/°C above +70°C).....533mW
Operating Temperature Ranges
MAX821_C_ _ .....0°C to +70°C
MAX821_E_ _.....-40°C to +85°C
MAX821_M_ _.....-55°C to +125°C
Storage Temperature Range .....-65°C to +150°C
Lead Temperature (soldering, 10sec) .....+300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

(V+ = 5V, T_A = +25°C, unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS		MAX8211 MIN	MAX8211 TYP	MAX8211 MAX	MAX8212 MIN	MAX8212 TYP	MAX8212 MAX	UNITS
Supply Current	I+	2V ≤ V+ ≤ 16.5V, GND ≤ V_{TH} ≤ V+	T_A = +25°C		5	15		5	15	µA
			T_A = T_{MIN} to T_{MAX}			20			20	
Threshold Trip Voltage	V_{TH}	T_A = +25°C	V+ = 16.5V, I_{OUT} = 4mA	1.11		1.19	1.11		1.19	V
			V+ = 2V, I_{OUT} = 500µA							
		T_A = T_{MIN} to T_{MAX}	V+ = 16.5V, I_{OUT} = 3mA	1.05		1.25	1.05		1.25	
			V+ = 2.2V, I_{OUT} = 500µA							
Threshold Voltage Disparity between Output and Hysteresis Output	V_{THP}	I_{OUT} = 4mA, I_{HYST} = 1mA			±0.1			±0.1		mV
Guaranteed Operating Supply Voltage Range	V_{SUPP}	T_A = +25°C		2.0		16.5	2.0		16.5	V
		T_A = T_{MIN} to T_{MAX}		2.2		16.5	2.2		16.5	
Typical Operating Supply Voltage Range	V_{SUPP}			1.5		16.5	1.5		16.5	V
Threshold Voltage Temperature Coefficient	$\Delta V_{TH}/\Delta T$	See Figure 4			-200			-200		ppm/°C
Variation of Threshold Voltage with Supply Voltage	ΔV_{TH}	V+ = 4.5V to 5.5V			1.0			0.2		mV
Threshold Input Current	I_{TH}	0V ≤ V_{TH} ≤ V+, T_A = +25°C			0.01	10		0.01	10	nA
		T_A = T_{MIN} to T_{MAX}				20			20	
Output Leakage Current	I_{LOUT}	T_A = T_{MIN} to T_{MAX}, C/E temp. ranges	V_{OUT} = 16.5V, V_{TH} = 1.0V						10	µA
			V_{OUT} = 16.5V, V_{TH} = 1.3V			10				
			V_{OUT} = 5V, V_{TH} = 1.0V						1	
			V_{OUT} = 5V, V_{TH} = 1.3V			1				
		T_A = T_{MIN} to T_{MAX}, M temp. range	V_{OUT} = 16.5V, V_{TH} = 0.9V						30	
			V_{OUT} = 16.5V, V_{TH} = 1.3V			30				
			V_{OUT} = 5V, V_{TH} = 0.9V						10	
			V_{OUT} = 5V, V_{TH} = 1.3V			10				

ELECTRICAL CHARACTERISTICS (continued)

(V+ = 5V, T_A = +25°C, unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS		MAX8211 MIN	MAX8211 TYP	MAX8211 MAX	MAX8212 MIN	MAX8212 TYP	MAX8212 MAX	UNITS
Output Saturation Voltage	V_{OL}	I_{OUT} = 2mA, V_{TH} = 1.0V			0.17	0.4				V
		I_{OUT} = 2mA, V_{TH} = 1.3V						0.17	0.4	
Maximum Available Output Current	I_{OH}	C temp. range, V_{OUT} = 5V	V_{TH} = 1.0V (Note 1)	4	7.0					mA
			V_{TH} = 1.3V (Note 2)				12	35		
Hysteresis Leakage Current	I_{LHYS}	T_A = T_{MIN} to T_{MAX}, C/E temp. ranges, V+ = 16.5V, V_{TH} = 1.0V, V_{HYST} = -16.5V with respect to V+				0.1			0.1	µA
		T_A = T_{MIN} to T_{MAX}, M temp. range, V+ = 16.5V, V_{TH} = 0.9V, V_{HYST} = -16.5V with respect to V+				3			3	
Hysteresis Saturation Voltage	V_{HYS} (MAX)	I_{HYST} = 0.5mA, V_{TH} = 1.3V, measured with respect to V+			-0.1	-0.2		-0.1	-0.2	V
Maximum Available Hysteresis Current	V_{HYS} (MAX)	V_{TH} = 1.3V, V_{HYS} = 0V		2	10		2	10		mA

Note 1: The maximum output current of the MAX8211 is limited by design to 30mA under any operating condition. The output voltage may be sustained at any voltage up to +16.5V as long as the maximum power dissipation of the device is not exceeded.

Note 2: The maximum output current of the MAX8212 is not defined, and systems using the MAX8212 must therefore ensure that the output current does not exceed 50mA and that the maximum power dissipation of the device is not exceeded.

Detailed Description

As shown in the block diagrams of Figures 1 and 2, the MAX8211 and MAX8212 each contain a 1.15V reference, a comparator, an open-drain N-channel output transistor, and an open-drain P-channel hysteresis output. The MAX8211 output N-channel turns on when the voltage applied to the THRESH pin is less than the internal reference (1.15V). The sink current is limited to 7mA (typical), allowing direct drive of an LED without a series resistor. The MAX8212 output turns on when the voltage applied to THRESH is greater than the internal reference. It is not current limited, and will typically sink 35mA.

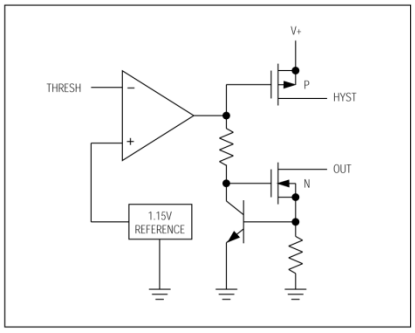


Figure 1. MAX8211 Block Diagram

Compatibility with ICL8211/ICL8212

The CMOS MAX8211/MAX8212 are plug-in replacements for the bipolar ICL8211/ICL8212 in most applications. The use of CMOS technology has several advantages. The quiescent supply current is much less than in the bipolar parts. Higher-value resistors can also be used

Microprocessor Voltage Monitors with Programmable Voltage Detection

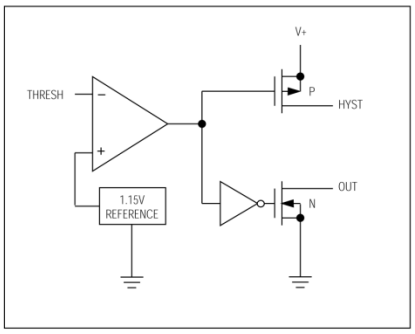


Figure 2. MAX8212 Block Diagram

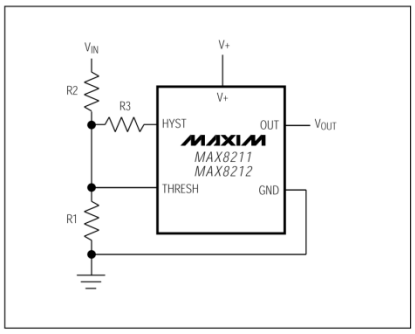


Figure 3. Basic Overvoltage/Undervoltage Circuit

in the networks that set up the trip voltage, since the comparator input (THRESH pin) is a low-leakage MOSFET transistor. This further reduces system current drain. The tolerance of the internal reference has also been significantly improved, allowing for more precise voltage detection without the use of potentiometers.

The available current from the HYST output has been increased from 21µA to 10mA, making the hysteresis feature easier to use. The disparity between the HYST output and the voltage required at THRESH to switch the OUT pin has also been reduced in the MAX8211 from 8mV to 0.1mV to eliminate output "chatter" or oscillation.

Most voltage detection circuits operate with supplies of 15V or less; in these applications, the MAX8211/MAX8212 will replace ICL8211/ICL8212s with the performance advantages described above. However, note that the CMOS parts have an absolute maximum supply-voltage rating of 18V, and should never be used in applications where this rating could be exceeded. Exercise caution when replacing ICL8211/ICL8212s in closed-loop applications such as programmable zeners. Although neither the ICL8211/ICL8212 nor the MAX8211/MAX8212 are internally compensated, the CMOS parts have higher gain and may not be stable for the external compensation-capacitor values used in lower-gain ICL8211/ICL8212 circuits.

Applications Information

Basic Voltage Detectors

Figure 3 shows the basic circuit for both undervoltage detection (MAX8211) and overvoltage detection (MAX8212). For applications where no hysteresis is needed, R3 should be omitted. The ratio of R1 to R2 is then chosen such that, for the desired trip voltage at V_{IN}, 1.15V is applied to the THRESH pin. Since the comparator inputs are very low-leakage MOSFET transistors, the MAX8211/MAX8212 can use much higher resistors values in the attenuator network than can the bipolar ICL8211/ICL8212. See Table 1 for switching delays.

Table 1. Switching Delays

TYPICAL DELAYS	MAX8211	MAX8212
$t_{(on)}$	40µs	250µs
$t_{(off)}$	1.5ms	3ms

Voltage Detectors with Hysteresis

To ensure noise-free output switching, hysteresis is frequently used in voltage detectors. For both the MAX8211 and MAX8212 the HYST output is on for threshold voltages greater than 1.15V. R3 (Figure 3) controls the amount of current (positive feedback) supplied from the HYST output to the mid-point of the resistor divider, and hence the magnitude of the hysteresis, or dead-band.

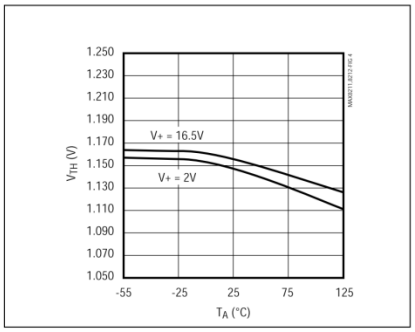


Figure 4. MAX8211/MAX8212 Threshold Trip Voltage vs. Ambient Temperature

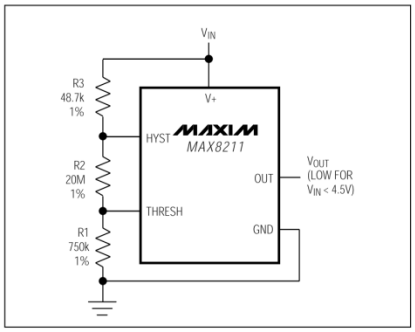


Figure 5. MAX8211 Logic-Supply Low-Voltage Detector

Calculate resistor values for Figure 3 as follows:

1) Choose a value for R1. Typical values are in the 10kΩ to 10MΩ range.
2) Calculate R2 for the desired upper trip point V_U using the formula:

$$R2 = R1 \times \frac{(V_U - V_{TH})}{V_{TH}} = R1 \times \frac{(V_U - 1.15V)}{1.15V}$$

3) Calculate R3 for the desired amount of hysteresis, where V_L is the lower trip point:

$$R3 = R2 \times \frac{(V+ - V_{TH})}{(V_U - V_L)} = R2 \times \frac{(V+ - 1.15V)}{(V_U - V_L)}$$

or, if $V^+ = V_{IN}$:

$$R3 = R2 \times \frac{(V_L - V_{TH})}{(V_U - V_L)} = R2 \times \frac{(V_L - 1.15V)}{(V_U - V_L)}$$

Figure 5 shows an alternate circuit, suitable only when the voltage being detected is also the power-supply voltage for the MAX8211 or MAX8212.

Calculate resistor values for Figure 5 as follows:

1) Choose a value for R1. Typical values are in the 10kΩ to 10MΩ range.
2) Calculate R2:

$$R2 = R1 \times \frac{(V_L - V_{TH})}{V_{TH}} = R1 \times \frac{(V_L - 1.15V)}{1.15V}$$

3) Calculate R3:

$$R3 = R1 \times \frac{(V_U - V_L)}{1.15V}$$

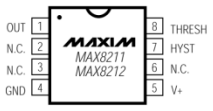
Low-Voltage Detector for Logic Supply

The circuit of Figure 5 will detect when a 5.0V (nominal) supply goes below 4.5V, which is the V_{MIN} normally specified in logic systems. The selected resistor values ensure that false undervoltage alarms will not be generated, even with worst-case threshold trip values and resistor tolerances. R3 provides approximately 75mV of hysteresis.

Microprocessor Voltage Monitors with Programmable Voltage Detection

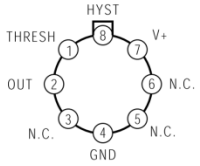
Pin Configurations (continued)

TOP VIEW



µMAX

TOP VIEW



TO-99*

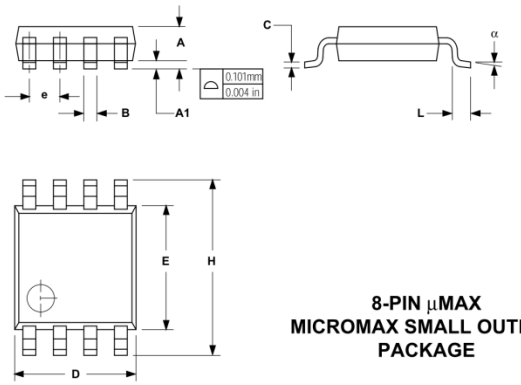
* CASE IS CONNECTED TO PIN 7 ON TV PACKAGE.
CASE IS CONNECTED TO PIN 4 ON TY PACKAGE.

Ordering Information (continued)

PART	TEMP. RANGE	PIN-PACKAGE
MAX8212CPA	0°C to +70°C	8 Plastic DIP
MAX8212CSA	0°C to +70°C	8 SO
MAX8212CUA	0°C to +70°C	8 µMAX
MAX8212CTY	0°C to +70°C	8 TO-99
MAX8212EPA	-40°C to +85°C	8 Plastic DIP
MAX8212ESA	-40°C to +85°C	8 SO
MAX8212EJA	-40°C to +85°C	8 CERDIP
MAX8212ETY	-40°C to +85°C	8 TO-99
MAX8212MJA	-55°C to +125°C	8 CERDIP
MAX8212MTV	-55°C to +125°C	8 TO-99

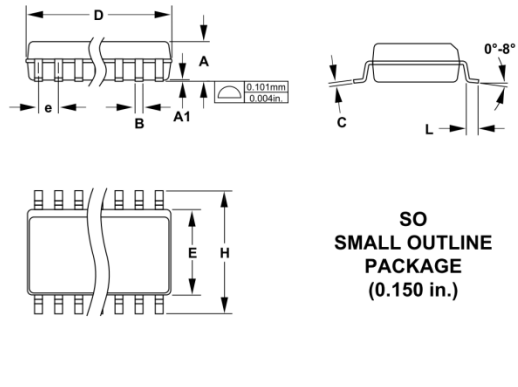
** Contact factory for dice specifications.*

Package Information



DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.036	0.044	0.91	1.11
A1	0.004	0.008	0.10	0.20
B	0.010	0.014	0.25	0.36
C	0.005	0.007	0.13	0.18
D	0.116	0.120	2.95	3.05
E	0.116	0.120	2.95	3.05
e	0.0256		0.65	
H	0.188	0.198	4.78	5.03
L	0.016	0.026	0.41	0.66
α	0°	6°	0°	6°

8-PIN μMAX
MICROMAX SMALL OUTLINE
PACKAGE



DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.053	0.069	1.35	1.75
A1	0.004	0.010	0.10	0.25
B	0.014	0.019	0.35	0.49
C	0.007	0.010	0.19	0.25
E	0.150	0.157	3.80	4.00
e	0.050		1.27	
H	0.228	0.244	5.80	6.20
L	0.016	0.050	0.40	1.27

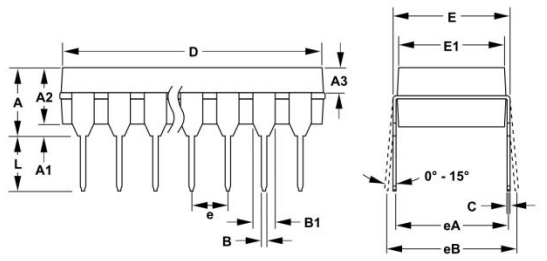
SO
SMALL OUTLINE
PACKAGE
(0.150 in.)

DIM	PINS	INCHES		MILLIMETERS	
		MIN	MAX	MIN	MAX
D	8	0.189	0.197	4.80	5.00
D	14	0.337	0.344	8.55	8.75
D	16	0.386	0.394	9.80	10.00

21-0041A

Microprocessor Voltage Monitors with Programmable Voltage Detection

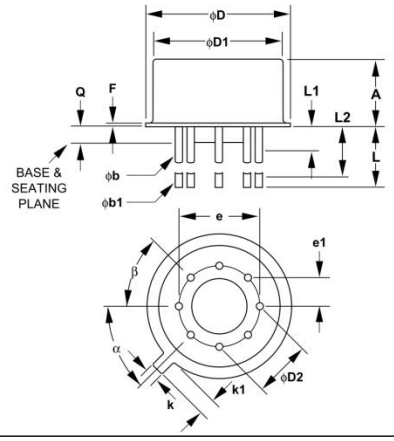
Package Information (continued)



Plastic DIP
PLASTIC
DUAL-IN-LINE
PACKAGE
(0.300 in.)

DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	–	0.200	–	5.08
A1	0.015	–	0.38	–
A2	0.125	0.175	3.18	4.45
A3	0.055	0.080	1.40	2.03
B	0.016	0.022	0.41	0.56
B1	0.045	0.065	1.14	1.65
C	0.008	0.012	0.20	0.30
D1	0.005	0.080	0.13	2.03
E	0.300	0.325	7.62	8.26
E1	0.240	0.310	6.10	7.87
e	0.100	–	2.54	–
eA	0.300	–	7.62	–
eB	–	0.400	–	10.16
L	0.115	0.150	2.92	3.81

DIM	PINS	INCHES		MILLIMETERS	
		MIN	MAX	MIN	MAX
D	8	0.348	0.390	8.84	9.91
D	14	0.735	0.765	18.67	19.43
D	16	0.745	0.765	18.92	19.43
D	18	0.885	0.915	22.48	23.24
D	20	1.015	1.045	25.78	26.54
D	24	1.14	1.265	28.96	32.13



DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.165	0.185	4.19	4.70
φb	0.016	0.019	0.41	0.48
φb1	0.016	0.021	0.41	0.53
φD	0.335	0.375	8.51	9.40
φD1	0.305	0.335	7.75	8.51
φD2	0.110	0.160	2.79	4.06
e	0.200 BSC		5.08 BSC	
e1	0.100 BSC		2.54 BSC	
F	–	0.040	–	1.02
k	0.027	0.034	0.69	0.86
k1	0.027	0.045	0.69	1.14
L	0.500	0.750	12.70	19.05
L1	–	0.050	–	1.27
L2	0.250	–	6.35	–
Q	0.010	0.045	0.25	1.14
α	45° BSC		45° BSC	
β	45° BSC		45° BSC	

21-0022A

8-PIN
TO-99 METAL CAN
PACKAGE

XBee & XBee-PRO OEM RF Module Antenna Considerations

Contents

Application Note

XST-AN019a

September 2005

Website: www.maxstream.net
Email: rf-xperts@maxstream.net

Abstract

This document presents practical information regarding the performance of the XBee and XBee-PRO RF Modules. The focus will be on the attributes of the different antenna options that are available to the modules. This information is intended to assist the designer in selecting the most appropriate module/antenna combination for their application. Indoor and outdoor systems will be covered.

XBee and XBee-PRO Product Comparison

XBee and XBee-PRO OEM RF Modules are small, high-performance, low-cost, wireless data transceivers. Both operate in the 2.4 GHz ISM band and because they have agency approvals (FCC, ETSI approvals pending), both can be operated without a station license. The XBee and XBee-PRO are pin-compatible with one another, though the XBee-PRO is slightly longer than the XBee. Both modules are available with a whip antenna, a low-profile chip antenna or a U.FL connector (to which an external antenna can be connected). The XBee transmits up to 1 mW of power, while the XBee-PRO transmits up to 60 mW of power. In addition to transmitting more power, the XBee-PRO is capable of receiving weaker signals than is the XBee; which means the XBee-PRO has better receiver sensitivity. Because the XBee-PRO is both more sensitive and transmits more power, it can send and receive data over longer distances than the XBee.

Link Quality Evaluation

In an effort to provide some practical information to the reader, as it relates to the distance of various XBee/XBee-PRO wireless links, a series of range tests (both indoor and outdoor) were performed. The indoor range tests were carried out in an office building and in a large warehouse (containing aisles of storage shelving). The outdoor range tests were completed near a business park (interspersed with multilevel buildings, young trees, parking areas and bordering a residential area).

Link distance, or range, was determined by measuring packet delivery from a transmitter to a receiver. The transmitter resided in a fixed location, while the receiver was moved to a number of different locations. Receiver locations were chosen such that the distance between the transmitter and receiver could be gradually increased until the link quality began to suffer. Most of the outdoor receiver locations were within visual line-of-sight of the transmitter (refer to application note "XST-AN010a" for more information regarding line-of-sight conditions). Some of the outdoor receiver locations were not within RF line-of-sight of the transmitter; however, they were within visual line-of-sight of the transmitter.

The transmitter was programmed to transmit packets containing a number that was incremented from one packet to the next (1, 2, 3 and so on…). The receiver (and associated lap-top PC) was configured to quantify successful packet delivery as a percentage of the total number of packets sent. The transmitter and receiver were alike, meaning that the module-type and antenna-type were identical. The modules were mounted to host interface boards. 99% successful packet delivery was chosen as a benchmark for comparison purposes. The receiver did not acknowledge to the transmitter when a packet arrived successfully. Furthermore, the transmitter sent each packet only once.

The table below summarizes the results of the evaluation. The distances presented in Table 1 represent what a user might expect to achieve in his application*. Activation of retries on the modules will improve packet delivery reliability in the presence of interference at the expense of overall data throughput (effective data rate).

Figure 1. Wireless link performance for various module/antenna/environment scenarios [Benchmark of 99% Packet Throughput]

Module	Antenna Type	Outdoor Distance (Visual Line-of-Sight)	Indoor Distance (Office Building)	Indoor Distance (Warehouse)
XBee	Chip	470 ft. (143 m)	80 ft. (24 m)	-
	Whip	845 ft. (258 m)	80 ft. (24 m)	84 ft. (26 m)
XBee-PRO	Chip	1690 ft. (515 m)	140 ft. (43 m)	-
	Whip	4382 ft. (1335 m)	140 ft. (43 m)	355 ft. (108 m)

A dipole antenna was also tested. The dipole and whip antennas perform similarly.

The radiation pattern for the whip antenna is similar to that of a dipole. That is to say, it is shaped like a donut. Thus, the performance of a module using a whip antenna, is relatively insensitive to its orientation in the plane that is perpendicular to the whip antenna. On the other hand, the radiation pattern of the chip antenna is not as uniform as that of the whip antenna. Therefore, certain orientations will achieve better performance than others. As our evaluation was performed, the orientation was selected to achieve the best performance. Because the radiation pattern will be affected by the antenna's immediate surroundings, MaxStream recommends range testing be performed with the module installed in its final assembly.

Observations

After reviewing Table 1, we can make several important observations.

- The whip antenna has a range advantage over the chip antenna, but only outdoors.
- The XBee-PRO can achieve more range than the XBee.
- The XBee-PRO and XBee both achieve more range outdoors than they do indoors.

* *Actual performance depends on many factors in the environment. Consequently, individual results may vary. Factors include: antenna orientation; antenna height, proximity of antenna to other objects such as an enclosure, PCB, or other mounting structures; trees; rain; snow; sleet; hail; bushes; shrubbery; flocks of birds; swarms of bees; moving vans; parked cars, trucks and vans; cars, trucks and vans in motion; intentional or unintentional interferers; etc. Longer distances may be possible with reduced throughput. Obstructions in the propagation path will affect performance. Other wireless networks or systems may affect performance.*

Discussion

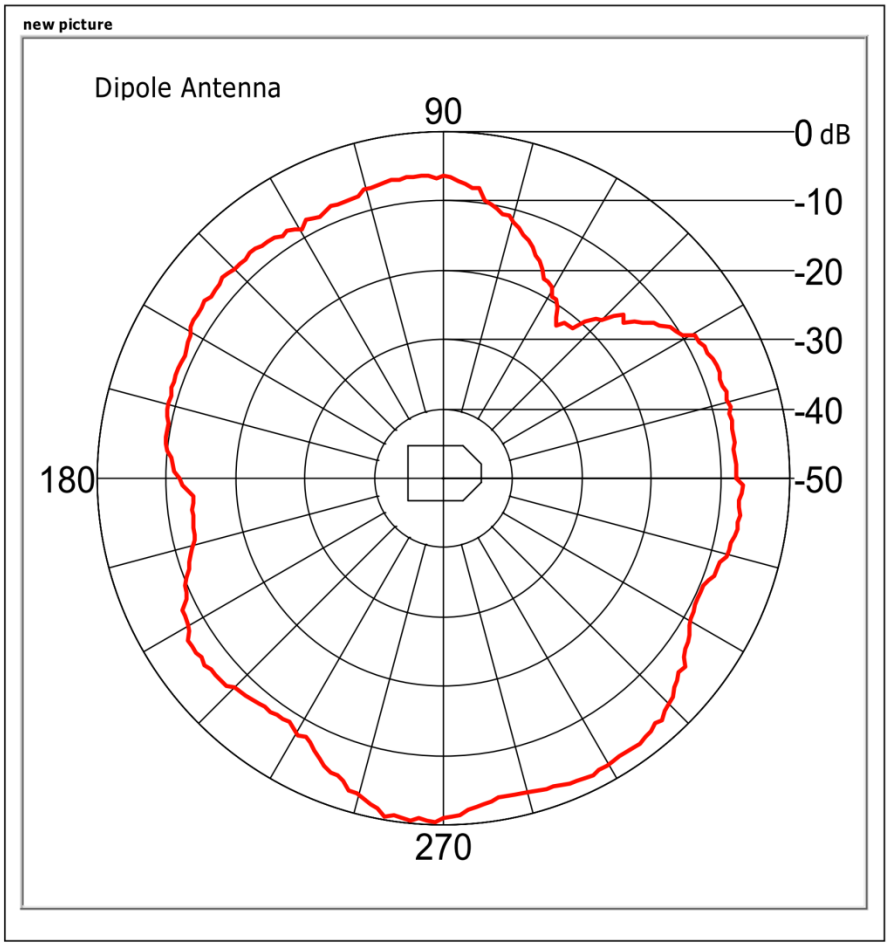
The whip antenna on the XBee module affords additional range in outdoor applications. However, it also occupies more space. If more range is required, and space is a constraint, then the XBee-PRO with a chip antenna may be more appropriate. On the other hand, if more range is a requirement and cost, not space, is the constraint, then the XBee with a whip antenna may be the best choice.

It should also be clear that the XBee-PRO can achieve superior range when compared to the XBee. Thus, if the application requires more range than the XBee can provide, then the XBee-PRO with a whip or a chip antenna could be used. Again, the chip antenna is best for tight spaces, while the whip antenna achieves more range.

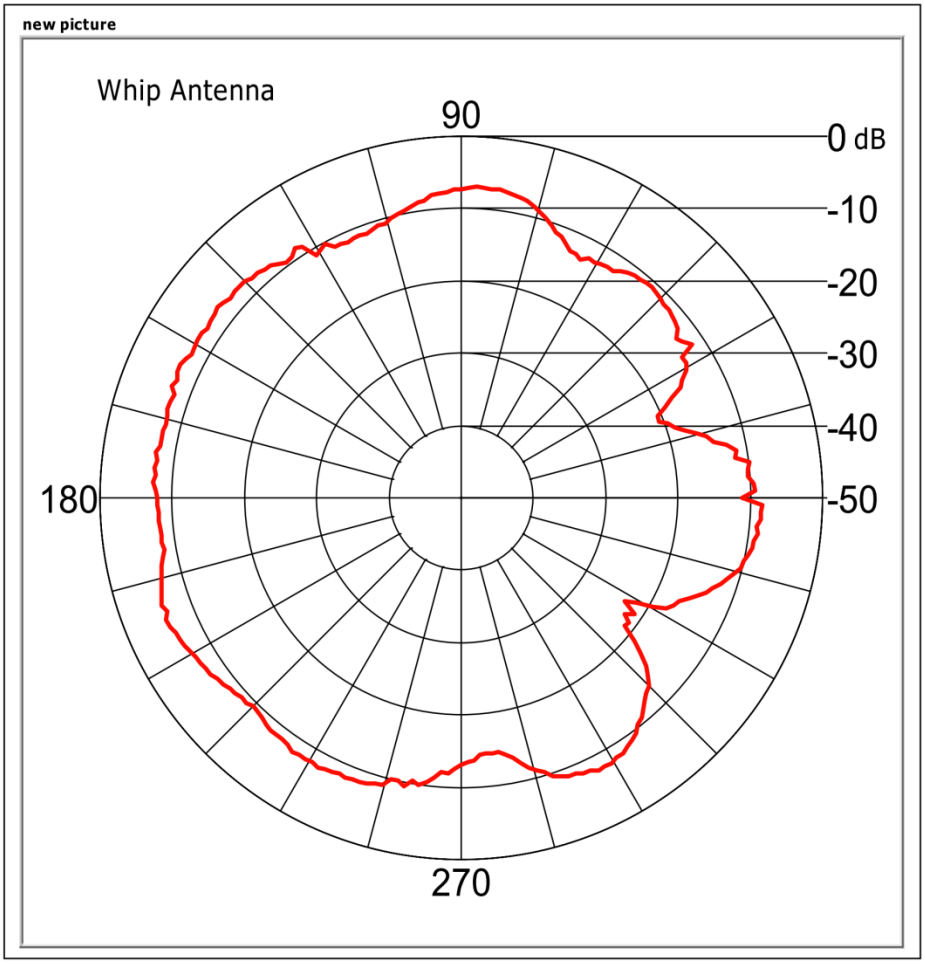
More Information

The information presented above has been given to help the reader understand the basic performance of the XBee and XBee-PRO wireless transceiver modules under various operating conditions. More information and resources are available by visiting www.maxstream.net. Antenna radiation patterns are available for both the chip and whip antennas and can be found in Appendix A. More detailed information associated with the link quality evaluation is also available. Thank you for considering MaxStream for your wireless data needs.

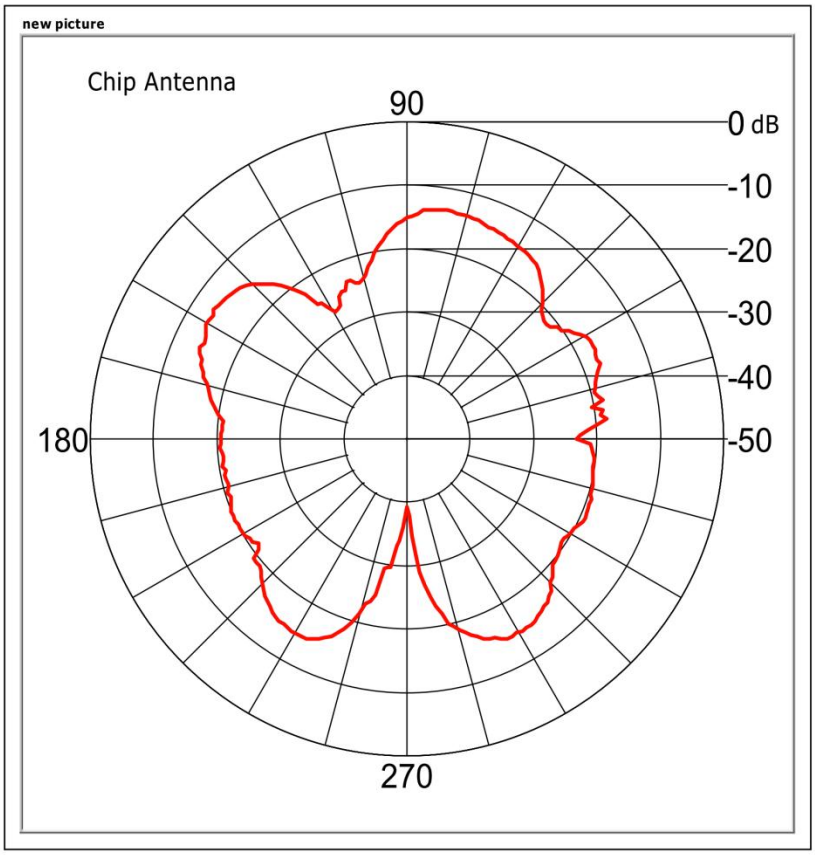
Appendix A



Radiation pattern of a dipole antenna connected to an XBee-PRO. The pattern is normalized to its peak. The chamfered end of the XBee-PRO points toward zero degrees as shown in the figure.



Radiation pattern of a whip antenna connected to an XBee-PRO. The pattern is normalized to the peak of the dipole antenna on the preceeding page, permitting an easy comparison.



Radiation pattern of a chip antenna connected to an XBee-PRO. The pattern is normalized to the peak of the dipole antenna on the first page of this appendix, again, permitting an easy comparison.

Appendix B: Matlab Code

two_tri.m

```
function [ x, y ] = two_tri(x1, x2, x3, y1, y2, y3, d1, d2, d3)
%2-D trilateration function
%   Inputs: Readers 1-3 (x,y) and Distances to Tag
%
% x_n1 = (d1^2 - d2^2)-(x1^2 - x2^2)-(y1^2 - y2^2)*2*(y3 - y1)
% x_n2 = 2*(y2 - y1)*(d1^2 - d3^2) - (x1^2 - x3^2) - (y1^2 - y3^2)
% x_d = 2*(x2 - x1)*2*(y3 - y1) - 2*(y2 - y1)*2*(x3-x1)
% x = (x_n1 - x_n2)/x_d
% y = 2;
%
x_n11 = (d1^2 - d2^2) - (x1^2 - x2^2) - (y1^2 - y2^2);
x_n21 = (d1^2 - d3^2) - (x1^2 - x3^2) - (y1^2 - y3^2);
x_n12 = 2*(y2-y1);
x_n22 = 2*(y3-y1);

d11 = 2*(x2-x1);
d21 = 2*(x3-x1);
d12 = 2*(y2-y1);
d22 = 2*(y3-y1);

x_n = [x_n11, x_n12; x_n21, x_n22]
d = [d11, d12; d21, d22]

x = x_n/d
x = det(x)

y_n11 = d11
y_n21 = d21
y_n12 = x_n11
y_n22 = x_n21

y_n = [y_n11, y_n12; y_n21, y_n22]
y = y_n/d
y = det(y)

end
```

Test2D.m

```
%prevents rounding when displaying fractions
format long

%Reader 2-D Coordinates [x, y]
Reader = [-2, 2; 2, 1; -1, -2];

%plot reader locations
scatter(getcolumn(Reader(1:3,1:2),1),getcolumn(Reader(1:3,1:2),2), 'MarkerEdgeColor',
[1 0 0], 'MarkerFaceColor', [1 0 0]);figure(gcf)
%x = 0;0;0;
%y = 0;0;0;
%x = getcolumn(Reader(1:3,1:2),1)
%y = getcolumn(Reader(1:3,1:2),2)
%e = [1;1;1]
hold on
%errorbar(x,y, e, 'og', 'Marker', '+');
%axis([-5 5 -5 5]) %set axis for 2-D graphs
set(gca, 'XTick', -5:1:5);
set(gca, 'YTick', -5:1:5);
```

```
grid on;

%distances to Tag from Reader(i)
%Distance = [2.82842715;2.236067977;2.236067977];  %(0, 0) tag
Distance = [3.16227766; 1; 3.60555127]; %(1, 1) tag

%error circles
circle([-2, 2],Distance(1), 1000,'-');figure(gcf)
%http://www.mathworks.com/matlabcentral/fileexchange/2876-draw-a-circle
circle([2, 1],Distance(2),1000, '-');figure(gcf)
circle([-1, -2],Distance(3), 1000, '-');figure(gcf)
pause;

x = 0;
y = 0;

%Reader(row, col)
[x, y] = two_tri(Reader(1,1), Reader(2, 1), Reader(3,1), Reader(1, 2), Reader(2, 2),
Reader(3, 2), Distance(1), Distance(2), Distance(3));

%plot tag
scatter(x,y, 'MarkerEdgeColor', [0 0 0], 'MarkerFaceColor', [0 0 0]);figure(gcf)
circle([x, y],.5,1000,'-');figure(gcf)
pause;
clf;
```

three_tri.m

```
function [x, y, z ] = three_tri( x1, y1, z1, d1, x2, y2, z2, d2, x3, y3, z3, d3, x4,
y4, z4, d4 )
%3-D trilateration. 4 anchor nodes
%   x1 = x coordinate of reader 1
%   y1 = y coordinate of reader 1
%   z1 = z coordinate of reader 1
%   d1 = distance from tag to reader 1
%   Function signature is done this way to make the functions below easier
%   to type and understand and debug. Harder to call the function but
%   easier to edit/understand the equations below

%x_numerator elements
x_n11 = (d1^2-d2^2) - (x1^2-x2^2) - (y1^2-y2^2) - (z1^2-z2^2); %sigma
x_n21 = (d1^2-d3^2) - (x1^2-x3^2) - (y1^2-y3^2) - (z1^2-z3^2); %beta
x_n31 = (d1^2-d4^2) - (x1^2-x4^2) - (y1^2-y4^2) - (z1^2-z4^2); %phi
x_n12 = 2*(y2-y1);
x_n22 = 2*(y3-y1);
x_n32 = 2*(y4-y1);
x_n13 = 2*(z2-z1);
x_n23 = 2*(z3-z1);
x_n33 = 2*(z4-z1);

%all the individual elements in M(COLA ieee document)
d11 = 2*(x2-x1);
d21 = 2*(x3-x1);
d31 = 2*(x4-x1);
d12 = 2*(y2-y1);
d22 = 2*(y3-y1);
d32 = 2*(y4-y1);
d13 = 2*(z2-z1);
d23 = 2*(z3-z1);
d33 = 2*(z4-z1);

%bringing M together into [3, 3] matrix
d = [d11, d12, d13; d21, d22, d23; d31, d32, d33];
```

```
%bringing numerator together for x
x_n = [x_n11, x_n12, x_n13; x_n21, x_n22, x_n23; x_n31, x_n32, x_n33];

%finding x by dividing matrix operation and then determinant
x = x_n / d;
x = det(x);

%individual y elements
y_n11 = 2*(x2-x1);
y_n21 = 2*(x3-x1);
y_n31 = 2*(x4-x1);
y_n12 = x_n11; %sigma
y_n22 = x_n21; %beta
y_n32 = x_n31; %phi
y_n13 = 2*(z2-z1);
y_n23 = 2*(z3-z1);
y_n33 = 2*(z4-z1);

%bringing numerator together for y
y_n = [y_n11, y_n12, y_n13; y_n21, y_n22, y_n23; y_n31, y_n32, y_n33];

%finding y by dividing matrix operation and then determinant
y = y_n / d;
y = det(y);

%individual z elements
z_n11 = 2*(x2-x1);
z_n21 = 2*(x3-x1);
z_n31 = 2*(x4-x1);
z_n12 = 2*(y2-y1);
z_n22 = 2*(y3-y1);
z_n32 = 2*(y4-y1);
z_n13 = x_n11; %sigma
z_n23 = x_n21; %beta
z_n33 = x_n31; %phi

%bringing z numerator together
z_n = [z_n11, z_n12, z_n13; z_n21, z_n22, z_n23; z_n31, z_n32, z_n33];

%finding z by dividing matrix operation and then determinant
z = z_n / d;
z = det(z);

end
```

Test3D.m

```
%prevents rounding when displaying fractions
format long

%Reader 3-D Coordinates [x, y, z]
Reader = [-2, 2, 2; 2, 1, -2; -1, -2, 2; 3, 3, 3];

%plot reader locations
scatter3(getcolumn(Reader(1:4,1:3),1),getcolumn(Reader(1:4,1:3),2),
getcolumn(Reader(1:4,1:3),3), 'MarkerEdgeColor', [1 0 0], 'MarkerFaceColor', [1 0
0]);figure(gcf)
%x = 0;0;0;
%y = 0;0;0;
%x = getcolumn(Reader(1:3,1:2),1)
%y = getcolumn(Reader(1:3,1:2),2)
```

```
%e = [1;1;1]
hold on
%errorbar(x,y, e, 'og', 'Marker', '+');
axis([-3 3 -3 3 -3 3]) %set axis for 2-D graphs
set(gca, 'XTick', -3:1:3);
set(gca, 'YTick', -3:1:3);
set(gca, 'ZTick', -3:1:3);
grid on;

%distances to Tag from Reader(i)
%Distance = [2.82842715;2.236067977;2.236067977];  %(0, 0) tag
Distance = [3.464101615; 3; 3; 5.196152423]; %(0, 0 , 0) tag

%initialize variables
x = 0;
y = 0;
z = 0;

[x, y, z] = three_tri(Reader(1, 1), Reader(1, 2), Reader(1, 3), Distance(1), Reader(2,
1), Reader(2, 2), Reader(2, 3), Distance(2), Reader(3, 1), Reader(3, 2), Reader(3, 3),
Distance(3), Reader(4, 1), Reader(4, 2), Reader(4, 3), Distance(4));

scatter3(x, y, z, 'MarkerEdgeColor', [1 1 0], 'MarkerFaceColor', [1 1 0]);figure(gcf)
```

COLA.m

```
%prevents formatting of decimals
format long

%super-node lower level nodes
%3-D coordinates [x, y, z]
lowReader = [-2, 2, 2; 1, 1, 2; -3, 0 , 2]; %all z-values must be the same

%x, y values must be same as lowReader values.
%Z values must be > lowReader values and equal to each other
highReader = [-2, 2, 4; 1, 1, 4; -3, 0, 4];

%tag x y z for testing purposes
tag = [4; 5; 1];

low_d1 = sqrt((lowReader(1, 1)- tag(1))^2 + (lowReader(1, 2)- tag(2))^2 +
(lowReader(1, 3) - tag(3))^2);
low_d2 = sqrt((lowReader(2, 1)- tag(1))^2 + (lowReader(2, 2)- tag(2))^2 +
(lowReader(2, 3)- tag(3))^2);
low_d3 = sqrt((lowReader(3, 1)- tag(1))^2 + (lowReader(3, 2)- tag(2))^2 +
(lowReader(3, 3)- tag(3))^2);

%find distances for tag (0, 0, 0)
high_d1 = sqrt((highReader(1,1)- tag(1))^2 + (highReader(1, 2)- tag(2))^2 +
(highReader(1, 3)- tag(3))^2);
high_d2 = sqrt((highReader(2,1)- tag(1))^2 + (highReader(2, 2)- tag(2))^2 +
(highReader(2, 3)- tag(3))^2);
high_d3 = sqrt((highReader(3,1)- tag(1))^2 + (highReader(3, 2)- tag(2))^2 +
(highReader(3, 3)- tag(3))^2);

lowDistances = [low_d1; low_d2; low_d3];
highDistances = [high_d1; high_d2; high_d3];

%height of tag
z = colaHeight(lowReader, highReader, lowDistances, highDistances)
%array for 2-D trilateration
distances = [0; 0; 0];
```

```
distances = findDistance(lowReader, highReader, lowDistances, highDistances);

[x, y] = two_tri(lowReader(1, 1), lowReader(2, 1), lowReader(3, 1), lowReader(1, 2),
lowReader(2, 2), lowReader(3, 2), distances(1), distances(2), distances(3))
```

COLA_height.m

```
function [ tagHeight ] = colaHeight(lowReader, highReader, lowDistances,
highDistances)
%COLAHEIGHT Find height of tag
%   Using supernode distances, use trig to determine height of tag

%heightDifference = difference in height of readers in supernode
%same for every node
heightDifference = highReader(1, 3) - lowReader(1, 3);

height = [0; 0; 0];

%see cola page 28 to get formula for h
for i = 1:3
    height(i) = highReader(i, 3) - ((highDistances(i)^2 - lowDistances(i)^2 +
heightDifference^2)/(2*heightDifference));
end

% DID NOT WORK: Mistake in formula derivation. See cola page 28 where
% cos(theta is found. multiple each side by d2 to get the height of tag +
% height of lower reader node.
% for i = 1:3
%     %numerator and denominator for cosine law
%     num = heightDifference^2 + lowDistances(i)^2 - highDistances(i)^2;
%     den = 2*lowDistances(i)*heightDifference;
%     theta = acos(num/den);
%     phi = 180 - theta;
%     x = cos(phi) * lowDistances(i);
%     height(i) = lowReader(i, 3) - x
% end

%set x = 0 for below
x = 0;
%find average height found by the 3 super-nodes for more accurate results
for i = 1:3
    x = x + height(i);
end
x = x/3;
tagHeight = x;
```

MATLAB GUI

```
function varargout = GUI_RFID(varargin)
% GUI_RFID MATLAB code for GUI_RFID.fig
%      GUI_RFID, by itself, creates a new GUI_RFID or raises the existing
%      singleton*.
%
%      H = GUI_RFID returns the handle to a new GUI_RFID or the handle to
%      the existing singleton*.
%
%      GUI_RFID('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI_RFID.M with the given input arguments.
%
```

```
%      GUI_RFID('Property','Value',...) creates a new GUI_RFID or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before GUI_RFID_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to GUI_RFID_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI_RFID

% Last Modified by GUIDE v2.5 19-Sep-2011 15:33:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_RFID_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_RFID_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before GUI_RFID is made visible.
function GUI_RFID_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_RFID (see VARARGIN)


x=varargin{1}
y=varargin{2}
z=5;

%plot3(x,y,z, 'o', 'MarkerFaceColor', 'm');figure(gcf);
scatter(getcolumn(varargin{3}(1:3,1:2),1),getcolumn(varargin{3}(1:3,1:2),2),
'MarkerEdgeColor', [1 0 0], 'MarkerFaceColor', [1 0 0]);figure(gcf)
set(gca, 'Xcolor', [0 0 0]);
set(gca, 'Ycolor', [0 0 0]);
axis([-10 10 -10 10]) %set axis for 2-D graphs
set(gca, 'XTick', -10:1:10);
```

```
set(gca, 'YTick', -10:1:10);
grid on;
hold on;
%set(gca, 'Zcolor', [0 0 0]);

xlabel('X Axis'),
ylabel('Y Axis'),
%zlabel('Height'),

rectangle('Position', [-1,3,2,2], 'FaceColor', [1,1,0], 'EdgeColor',
[1,1,0]);


circle([0, 10],varargin{4}(1), 1000,'-');figure(gcf)
%http://www.mathworks.com/matlabcentral/fileexchange/2876-draw-a-circle
circle([-10, 0],varargin{4}(2),1000, '-');figure(gcf)
circle([10, 0],varargin{4}(3), 1000, '-');figure(gcf)


scatter(x,y, 'MarkerEdgeColor', [0 0 0], 'MarkerFaceColor', [0 0
0]);figure(gcf)

% axis manual;
% axis([-10 10 -10 10 0 10]);
% grid on;
% rotate3d on;

stringx = num2str(x);
stringy = num2str(y);
stringz = num2str(z);
str_x = 'X = ';
str_y = 'Y = ';
str_z = 'Height = ';
output_x = strcat(str_x, stringx);
output_y = strcat(str_y, stringy);
output_z = strcat(str_z, stringz);
set(handles.X_plot, 'String', output_x);
set(handles.Y_plot, 'String', output_y);
set(handles.Height_plot, 'String', output_z);

% Choose default command line output for GUI_RFID
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI_RFID wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = GUI_RFID_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function X_plot_Callback(hObject, eventdata, handles)
% hObject    handle to X_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of X_plot as text
%        str2double(get(hObject,'String')) returns contents of X_plot as a
double


% --- Executes during object creation, after setting all properties.
function X_plot_CreateFcn(hObject, eventdata, handles)
% hObject    handle to X_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Y_plot_Callback(hObject, eventdata, handles)
% hObject    handle to Y_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Y_plot as text
%        str2double(get(hObject,'String')) returns contents of Y_plot as a
double

% --- Executes during object creation, after setting all properties.
function Y_plot_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Y_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in Find_pushbutton1.
function Find_pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to Find_pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

function Height_plot_Callback(hObject, eventdata, handles)
% hObject    handle to Height_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Height_plot as text
%        str2double(get(hObject,'String')) returns contents of Height_plot as
a double

% --- Executes during object creation, after setting all properties.

function Height_plot_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Height_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Matlab Distance Formula

```
function [ distance ] = rssiDistance( rssi )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
%COLA formula
%P_o = 17; %RSSI at 0 meters
P_o = 23;
f = 2400; %frequency (2.4 GHz = 2400 MHz)

%n = 3.5; %path-loss exponent
n = 3.3;
f_m = 6; %fade margin

distance = 10^((P_o - f_m - rssi - (10*n*log10(f)) + (30*n) - 32.44)/(10*n));
```

RFID Matlab Serial Input

```
%Taking in Serial Data from Arduino

clear all;
Arduino = serial('COM9'); %define serial
Arduino.BaudRate = 9600; %baud rate
```

```
end_str = 'END';
header = 'Byte';
footer = 'Footer';
RFID = '';

fopen(Arduino); %opens up serial port for reading


out = instrfind('Port', 'COM9'); %Port and COM3 are the objects



while(1)
    try
data = fscanf(Arduino, '%s'); %takes in data as a string and returns it to data
   if (strcmp(data, footer)) %if Serial takes in the string "End" close the serial
port and exit loop
        fclose(Arduino);
        delete(Arduino);
        break; %terminate loop
   else
        if(strcmp(data, header))
            data3 = fscanf(Arduino, '%s');
            RFID = strcat(RFID, data3);
        end
   end
    catch exception
        fclose(Arduino);
        delete(Arduino);
        break;
    end
end
```

Appendix C: Arduino Code

Printing RFID to Screen

```
//#include <NewSoftSerial.h>

int resetPin = 13;
const int serIn = 2;
const int serOut = 3;
//NewSoftSerial port(serIn, serOut);
char incomingByte;


void setup()
{
 Serial.begin(9600);
 pinMode(resetPin, OUTPUT);
 digitalWrite(resetPin, HIGH);
 //port.begin(9600);
 Serial.println("Bring RFID card to reader");
}

void loop()
{
 boolean reading = false;
 incomingByte= 0;
 if (Serial.available())
 {
   incomingByte = Serial.read();
   if(incomingByte == 2)
   {
    Serial.println("STX");
   }
   if(incomingByte == 3)
   {
    Serial.println("ETX");
   }
   else
   {
   Serial.print(incomingByte, BYTE);
   }
 }

}

void resetReader()
{
 digitalWrite(resetPin, LOW);
 delay(150);
 digitalWrite(resetPin, HIGH);
}
```

Sending RFID to Matlab

```
//#include <NewSoftSerial.h>

int resetPin = 13;
const int serIn = 2;
const int serOut = 3;
```

```
//NewSoftSerial port(serIn, serOut);
char incomingByte;


void setup()
{
 Serial.begin(9600);
 pinMode(resetPin, OUTPUT);
 digitalWrite(resetPin, HIGH);
 //port.begin(9600);
 Serial.println("Bring RFID card to reader");
}

void loop()
{
 boolean reading = false;
 incomingByte= 0;
 if (Serial.available())
 {
   incomingByte = Serial.read();
   if(incomingByte == 2)
   {
     Serial.println("Header");
   }
   if(incomingByte == 3)
   {
     Serial.println("Footer");
   }
   else
   {
   Serial.print(incomingByte, BYTE);
   Serial.print(incomingByte);
   }
 }

}
void resetReader()
{
 digitalWrite(resetPin, LOW);
 delay(150);
 digitalWrite(resetPin, HIGH);
}
```

AT Command

```
String xb1_DH= "ATDH0013A200\r";
String xb1_DL= "ATDL40715D91\r";
String xb2_DH= "ATDH0013A200\r";
String xb2_DL= "ATDL40715D34\r";
String api = "ATAP0\r"; //sets API mode to 1
boolean configured = false;

boolean configureRadio() {
 Serial.print("+++");

 String ok_response = "OK\r";
 String addressLow = String("");
 String addressHigh = String("");
```

```
String response = String("");
while (response.length() < ok_response.length())
{
 if(Serial.available() > 0)
 {
  response += (char) Serial.read();
 }
}

 if (response.equals(ok_response))
 {
  Serial.print("ATSH\r");
  delay(200);
    while(Serial.available() > 0)
    {
      addressHigh += (char) Serial.read();
    }
  Serial.print("ATSL\r");
  delay(200);
  while(Serial.available() > 0)
  {
   addressLow += (char) Serial.read();
  }
  Serial.print(xb1_DH);
  delay(10);
  Serial.print(xb1_DL);
  delay(10);
  Serial.print(api);
  delay(10);
  Serial.flush(); //flushes the reponses from changing addresses
  Serial.print("ATDH\r");
  delay(20); //minimum delay for waiting for 8 bytes
   while(Serial.available() > 0)
    {
      addressHigh += (char) Serial.read();
    }
  Serial.print("ATDL\r");
  delay(20); //minimum delay for waiting for 8 bytes
  while(Serial.available() > 0)
  {
   addressLow += (char) Serial.read();
  }
            Serial.print("ATWR\r");
            delay(10);
  Serial.print("ATCN\r");
  Serial.println("");
  Serial.print("Coordinator = ");
  Serial.println(addressHigh);
  Serial.print("Address low = ");
  Serial.println(addressLow);
  Serial.println(response);
  return true;
 }
 else {
  return false;
 }
}
```

```
void setup()
{
 Serial.begin(9600);
 configured = configureRadio(); //AT commands
}

void loop()
{
 if(configured) {
  Serial.print("Hello!");
  delay(3000);
 }
 else {
  delay(30000);
  configureRadio();
 }
}
```

Series 1 API Sender Test Code

```
#include <XBee.h>

XBee xbee = XBee();

unsigned long start = millis();

// allocate two bytes for to hold a 10-bit analog reading
uint8_t payload[] = { 0, 0 };

// with Series 1 you can use either 16-bit or 64-bit addressing

// 16-bit addressing: Enter address of remote XBee, typically the coordinator
//Tx16Request tx = Tx16Request(0x1874, payload, sizeof(payload));

// 64-bit addressing: This is the SH + SL address of remote XBee
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200, 0x4008b490);
// unless you have MY on the receiving radio set to FFFF, this will be received as a RX16 packet
Tx64Request tx = Tx64Request(addr64, payload, sizeof(payload));

TxStatusResponse txStatus = TxStatusResponse();

int pin5 = 0;

int statusLed = 11;
int errorLed = 12;

void flashLed(int pin, int times, int wait) {

  for (int i = 0; i < times; i++) {
    digitalWrite(pin, HIGH);
    delay(wait);
    digitalWrite(pin, LOW);

    if (i + 1 < times) {
      delay(wait);
    }
  }
}
```

```
void setup() {
  pinMode(statusLed, OUTPUT);
  pinMode(errorLed, OUTPUT);

  xbee.begin(9600);
}

void loop() {

   // start transmitting after a startup delay.  Note: this will rollover to 0 eventually so not best way to handle

     xbee.send(tx);

     // flash TX indicator
     flashLed(statusLed, 1, 100);
   }

   // after sending a tx request, we expect a status response
   // wait up to 5 seconds for the status response
   if (xbee.readPacket(5000)) {
      // got a response!

      // should be a znet tx status
            if (xbee.getResponse().getApiId() == TX_STATUS_RESPONSE) {
              xbee.getResponse().getZBTxStatusResponse(txStatus);

              // get the delivery status, the fifth byte
        if (txStatus.getStatus() == SUCCESS) {
            // success.  time to celebrate
            flashLed(statusLed, 5, 50);
        } else {
            // the remote XBee did not receive our packet. is it powered on?
            flashLed(errorLed, 3, 500);
        }
      }
   } else {
     // local XBee did not provide a timely TX Status Response -- should not happen
     flashLed(errorLed, 2, 50);
   }

   delay(1000);
}
```

Series 1 API Receiver Test Code

```
#include <XBee.h>

XBee xbee = XBee();
XBeeResponse response = XBeeResponse();
// create reusable response objects for responses we expect to handle
Rx16Response rx16 = Rx16Response();
Rx64Response rx64 = Rx64Response();

int statusLed = 11;
int errorLed = 12;
int dataLed = 10;
```

```
uint8_t option = 0;
uint8_t data = 0;

void flashLed(int pin, int times, int wait) {

   for (int i = 0; i < times; i++) {
     digitalWrite(pin, HIGH);
     delay(wait);
     digitalWrite(pin, LOW);

     if (i + 1 < times) {
       delay(wait);
     }
   }
}

void setup() {
  pinMode(statusLed, OUTPUT);
  pinMode(errorLed, OUTPUT);
  pinMode(dataLed,  OUTPUT);

  // start serial
  xbee.begin(9600);

  flashLed(statusLed, 3, 50);
}

// continuously reads packets, looking for RX16 or RX64
void loop() {

   xbee.readPacket();

   if (xbee.getResponse().isAvailable()) {
     // got something

     if (xbee.getResponse().getApiId() == RX_16_RESPONSE || xbee.getResponse().getApiId() == RX_64_RESPONSE) {
       // got a rx packet

       if (xbee.getResponse().getApiId() == RX_16_RESPONSE) {
            xbee.getResponse().getRx16Response(rx16);
             option = rx16.getOption();
             data = rx16.getData(0);
       } else {
            xbee.getResponse().getRx64Response(rx64);
             option = rx64.getOption();
             data = rx64.getData(0);
       }

       // TODO check option, rssi bytes
       flashLed(statusLed, 1, 10);

       // set dataLed PWM to value of the first byte in the data
       analogWrite(dataLed, data);
     } else {
            // not something we were expecting
       flashLed(errorLed, 1, 25);
     }
   }
```

```
}
```

Location System Tag Final Code

```
#include <XBee.h>

XBee xbee = XBee();

uint8_t payload[] = {102, 0, 2, 66, 47};
unsigned long start = millis();

boolean started;
XBeeAddress64 tag = XBeeAddress64(0x0013A200, 0x407BABC0); //Arduino 2009
XBeeAddress64 reader1 = XBeeAddress64(0x0013A200, 0x40715D34); //Arduino UNO
XBeeAddress64 reader2 = XBeeAddress64(0x0013A200, 0x4077939A);
XBeeAddress64 reader3 = XBeeAddress64(0x0013A200, 0x407BACD9);

TxStatusResponse txStatus = TxStatusResponse();

//DMTxRequest rd1 = DMTxRequest(reader1, payload, sizeof(payload));
//DMTxRequest rd2 = DMTxRequest(reader2, payload, sizeof(payload));
//DMTxRequest rd3 = DMTxRequest(reader3, payload, sizeof(payload));

//Tx64Request rd1 = Tx64Request(reader1, payload, sizeof(payload));
//Tx64Request rd2 = Tx64Request(reader2, payload, sizeof(payload));
//Tx64Request rd3 = Tx64Request(reader3, payload, sizeof(payload));
Tx16Request rd1 = Tx16Request(0x0001, payload, sizeof(payload));
Tx16Request rd2 = Tx16Request(0x0002, payload, sizeof(payload));
Tx16Request rd3 = Tx16Request(0x0003, payload, sizeof(payload));

Rx16Response rx16 = Rx16Response();

uint16_t received_16 = 0;

int count = 0;

void setup() {
 Serial.begin(9600);
 xbee.begin(9600);
 delay(5000);
}

void loop() {
 xbee.readPacket();

 if (xbee.getResponse().isAvailable())
 {
  if (xbee.getResponse().getApiId() == RX_16_RESPONSE)
  {
  xbee.getResponse().getRx16Response(rx16);
  received_16 = rx16.getRemoteAddress16();
  if(received_16 == 2)
  {
  for(int j = 0; j < 9; j++)
  {
  if (count == 0)
  {
    for(int i = 0; i <= 49; i++)
```

```
    {
      xbee.send(rd1);
      if (xbee.readPacket(5000))
    {
     Serial.println("Reader1");
    }
      delay(8);
    }
    count++;
   }
   else if (count == 1) {
    for(int i = 0; i <= 49; i++)
    {
    xbee.send(rd2);
    if (xbee.readPacket(5000))
    {
     Serial.println("Reader2");
    }
    delay(8);
    }
    count++;
   }
   else
   {
    for(int i = 0; i <= 49; i++)
    {
    xbee.send(rd3);
    if (xbee.readPacket(5000))
    {
     Serial.println("Reader3");
    }
    delay(8);
    }
    count = 0;
   }

//  if (xbee.readPacket(5000))
//  {
//    if (xbee.getResponse().getApiId() == TX_STATUS_RESPONSE) //
//    {
//      xbee.getResponse().getZBTxStatusResponse(txStatus); //get acknowledge response
//      if (txStatus.getStatus() == SUCCESS) { //txResponse received
//       // Serial.println("Acknowledge packet received");
//      } else {
//       Serial.println("Acknowledge packet NOT received");
//      }
//    }
//  } else {
//    Serial.println("Acknowledge packet did not get here in 5 seconds");
//  }
  delay(1000);
    }
  }
    }
  }
}
```

Location System Reader Final Code

```
#include <XBee.h>
#include <NewSoftSerial.h>

XBee xbee = XBee(); //make instance of xbee

NewSoftSerial nss(2,3);
uint8_t rssi[149];

uint8_t payload[] = {0};
XBeeAddress64 tag = XBeeAddress64(0x0013A200, 0x407BABC0);
XBeeAddress64 reader1 = XBeeAddress64(0x0013A200, 0x40715D34);
XBeeAddress64 reader2 = XBeeAddress64(0x0013A200, 0x4077939A);
XBeeAddress64 reader3 = XBeeAddress64(0x0013A200, 0x407BACD9);
XBeeAddress64 received;
XBeeResponse response = XBeeResponse();

uint16_t received_16;

//DMTxRequest coord = DMTxRequest(reader2, payload, sizeof(payload));
//Tx64Request coord = Tx64Request(reader2, payload, sizeof(payload));
//DMTxRequest respond = DMTxRequest(tag, payload, sizeof(payload));

//Tx64Request respond = Tx64Request(tag, payload, sizeof(payload));
Tx16Request coord = Tx16Request(0x0002, payload, sizeof(payload));

Rx16Response rx16 = Rx16Response();
Rx64Response rx64 = Rx64Response();

DMRxResponse rxDM = DMRxResponse();
int count;

uint8_t dbCmd[] = {'D', 'B'};

AtCommandRequest atRequest = AtCommandRequest(dbCmd);

AtCommandResponse atResponse = AtCommandResponse();

int incomingByte;

uint8_t option = 0;
uint8_t data = 0;
uint8_t length = 0;
void setup()
{
  xbee.begin(9600);
  //Serial.begin(9600);
  delay(5000);
  nss.begin(9600);
  count = 0;
}

void loop()
{
  xbee.readPacket();

  if (xbee.getResponse().isAvailable()) { //received packet
```

```
   if (xbee.getResponse().getApiId() == RX_16_RESPONSE || xbee.getResponse().getApiId() == RX_64_RESPONSE ||
xbee.getResponse().getApiId() == RX_RESPONSE ) {
     if (xbee.getResponse().getApiId() == RX_16_RESPONSE) {
       xbee.getResponse().getRx16Response(rx16);
       received_16 = rx16.getRemoteAddress16();
       //data = rx16.getApiId();
       data = rx16.getData(0);
       Serial.println("16 Response");
             if(received_16 > 6)
             {
          rssi[count] = rx16.getRssi();
          delay(7);
          Serial.print("RSSI = ");
          Serial.println(rssi[count], HEX);
          count++;
          Serial.print("COUNT = ");
          Serial.println(count);
       }
       if(received_16 == 2)
       {
        Serial.println("RESET");
        count = 0;
       }
     } else if (xbee.getResponse().getApiId() == RX_64_RESPONSE) {
                          Serial.println("64 Response");
       xbee.getResponse().getRx64Response(rx64);
                          received = rx64.getRemoteAddress64();
                          if(received.getMsb() == tag.getMsb() && received.getLsb() == tag.getLsb())
                          {
                                    rssi[count] = rx64.getRssi();
                                    count++;
                                    delay(7);
                                    Serial.print("COUNT = ");
                                    Serial.println(count);
                          }
     } else if (xbee.getResponse().getApiId() == RX_RESPONSE) {
       xbee.getResponse().getDMRxResponse(rxDM);
//       for(int i = 0; i < rxDM.getDataLength(); i++)
//       {
//       data = rxDM.getData(i);
//       Serial.print("Data = ");
//       Serial.println(data, DEC);
//       }
       received = rxDM.getRemoteAddress64();
       if(received.getMsb() == tag.getMsb() && received.getLsb() == tag.getLsb())
       {
         rssi[count] = sendAtCommand();
         count++;
         delay(7);
         Serial.println("COUNT = ");
         Serial.println(count);
         //get DB and send to coord
       }
     }


   }
  }
```

```
 if(count == 149)
 {
  payload[0] = gaussian(rssi);
  Serial.print("Gaussian Avg = ");
  Serial.print(payload[0], DEC);
  xbee.send(coord);
  count = 0;
 }
}

byte sendAtCommand() {
 xbee.send(atRequest);

 //wait for response packet
 if(xbee.readPacket(5000)) {
  if(xbee.getResponse().getApiId() == AT_COMMAND_RESPONSE) {
   xbee.getResponse().getAtCommandResponse(atResponse);

   if (atResponse.isOk()) {
    Serial.print("Command [");
    Serial.print(atResponse.getCommand()[0]);
    Serial.print(atResponse.getCommand()[1]);
    Serial.println("] was successful!");

    if (atResponse.getValueLength() > 0) {
     Serial.print("Command value length is ");
     Serial.println(atResponse.getValueLength(), DEC);

     Serial.print("Command value: ");


     for (int i = 0; i < atResponse.getValueLength(); i++) {
      Serial.println(atResponse.getValue()[i], DEC);
      return atResponse.getValue()[i];
     }
     Serial.println("");
       }
     }
    else {
     Serial.print("Command return error code: ");
     Serial.println(atResponse.getStatus(), HEX);
    }
   } else {
    Serial.print("Expected AT response but got ");
   Serial.print(xbee.getResponse().getApiId(), HEX);
   }
 } else {
  if (xbee.getResponse().isError()) {
   Serial.print("Error reading packet. Error code: ");
   Serial.println(xbee.getResponse().getErrorCode());
  } else {
    Serial.println("No Response from radio");
  }
 }
}

uint8_t gaussian(uint8_t rssi[])
{
```

```
 double rssi_sum = 0; //summation
 for(int i = 0; i <= 149; i++)
 {
  rssi_sum += rssi[i];
 }

 int rssi_avg = rssi_sum / 150; //find mean of all 50 values
 return rssi_avg;

}
```

Location System Coordinator Final Code

```
#include <XBee.h>
#include <NewSoftSerial.h>

XBee xbee = XBee(); //make instance of xbee

NewSoftSerial nss(6,7); // (RX, TX)

uint8_t payload[] = {0};
XBeeAddress64 tag = XBeeAddress64(0x0013A200, 0x40715D91);
XBeeAddress64 reader1 = XBeeAddress64(0x0013A200, 0x40715D34);
XBeeAddress64 reader2 = XBeeAddress64(0x0013A200, 0x4077939A);
XBeeAddress64 reader3 = XBeeAddress64(0x0013A200, 0x407793A7);
XBeeAddress64 received;
XBeeResponse response = XBeeResponse();

Tx16Request tag1 = Tx16Request(0x007, payload, sizeof(payload));
Tx16Request reset_r1 = Tx16Request(0x0001, payload, sizeof(payload));
Tx16Request reset_r3 = Tx16Request(0x0003, payload, sizeof(payload));

DMTxRequest respond = DMTxRequest(tag, payload, sizeof(payload));

Rx16Response rx16 = Rx16Response();
Rx64Response rx64 = Rx64Response();

DMRxResponse rxDM = DMRxResponse();

uint8_t dbCmd[] = {'D', 'B'};
int count;
uint8_t rssi[149];

AtCommandRequest atRequest = AtCommandRequest(dbCmd);

AtCommandResponse atResponse = AtCommandResponse();

int incomingByte;

uint8_t option = 0;
uint8_t data = 0;
uint8_t length = 0;

uint16_t received_16 = 0;

void setup()
{
 xbee.begin(9600);
```

```
 //Serial.begin(9600);
 xbee.send(reset_r1);
 xbee.send(reset_r3);
 delay(5000);
 xbee.send(tag1);
 nss.begin(9600);
 count = 0;
}

void loop()
{
 xbee.readPacket();

 if (xbee.getResponse().isAvailable()) { //received packet
  if (xbee.getResponse().getApiId() == RX_16_RESPONSE || xbee.getResponse().getApiId() == RX_64_RESPONSE ||
xbee.getResponse().getApiId() == RX_RESPONSE ) {
    if (xbee.getResponse().getApiId() == RX_16_RESPONSE) {
     xbee.getResponse().getRx16Response(rx16);
     received_16 = rx16.getRemoteAddress16();
     if (received_16 > 6)
    {
        rssi[count] = rx16.getRssi();
        count++;
        delay(7);
        Serial.print("COUNT = ");
        Serial.println(count);
    }
     else if(received_16 == 1)
     {
        nss.print(1, HEX);
        nss.print(rx16.getData(0), HEX);
        Serial.println("READER1");
     }
     else if(received_16 == 3)
     {
        nss.print(3, HEX);
        nss.print(rx16.getData(0), HEX);
        Serial.println("READER3");
     }

     Serial.println("16 Response");
    } else if (xbee.getResponse().getApiId() == RX_64_RESPONSE) {
     xbee.getResponse().getRx64Response(rx64);
     received = rxDM.getRemoteAddress64();
     if (received.getMsb() == tag.getMsb() && received.getLsb() == tag.getLsb())
    {
        rssi[count] = rx64.getRssi();
        count++;
        delay(7);
        Serial.print("COUNT = ");
        Serial.println(count);
    }
     else if(received.getMsb() == reader1.getMsb() && received.getLsb() == reader1.getLsb())
     {
        nss.print(1, HEX);
        nss.print(rxDM.getData(0), HEX);
        Serial.println("READER1");
     }
```

```
      else if(received.getMsb() == reader3.getMsb() && received.getLsb() == reader3.getLsb())
      {
         nss.print(3, HEX);
         nss.print(rxDM.getData(0), HEX);
         Serial.println("READER3");
      }
      Serial.println("64 Response");
     } else if (xbee.getResponse().getApiId() == RX_RESPONSE) {
      xbee.getResponse().getDMRxResponse(rxDM);
      received = rxDM.getRemoteAddress64();
      if (received.getMsb() == tag.getMsb() && received.getLsb() == tag.getLsb())
     {
         rssi[count] = sendAtCommand();
         count++;
         delay(7);
         xbee.send(respond);
         Serial.print("COUNT = ");
         Serial.println(count);
     }
      else if(received.getMsb() == reader1.getMsb() && received.getLsb() == reader1.getLsb())
      {
         nss.print(1, HEX);
         nss.print(rxDM.getData(0), HEX);
         Serial.println("READER1");
      }
      else if(received.getMsb() == reader3.getMsb() && received.getLsb() == reader3.getLsb())
      {
         nss.print(3, HEX);
         nss.print(rxDM.getData(0), HEX);
         Serial.println("READER3");
      }
    }
   }
}
  if(count == 149)
  {
    payload[0] = gaussian(rssi);
    nss.print(2, HEX);
    nss.print(payload[0], HEX);
    Serial.print("Gaussian Avg = ");
    Serial.println(payload[0], HEX);
    count = 0;
    Serial.flush();
  }
}

byte sendAtCommand() {
  xbee.send(atRequest);

  //wait for response packet
  if(xbee.readPacket(5000)) {
    if(xbee.getResponse().getApiId() == AT_COMMAND_RESPONSE) {
      xbee.getResponse().getAtCommandResponse(atResponse);

     if (atResponse.isOk()) {
       Serial.print("Command [");
       Serial.print(atResponse.getCommand()[0]);
       Serial.print(atResponse.getCommand()[1]);
```

```
      Serial.println("] was successful!");

     if (atResponse.getValueLength() > 0) {
      Serial.print("Command value length is ");
      Serial.println(atResponse.getValueLength(), DEC);

      Serial.print("Command value: ");


      for (int i = 0; i < atResponse.getValueLength(); i++) {
       Serial.print(atResponse.getValue()[i], DEC);
       return atResponse.getValue()[i];

      }
      Serial.println("");
        }
      }
     else {
      Serial.print("Command return error code: ");
      Serial.println(atResponse.getStatus(), HEX);
     }
    } else {
     Serial.print("Expected AT response but got ");
    Serial.print(xbee.getResponse().getApiId(), HEX);
    }
  } else {
   if (xbee.getResponse().isError()) {
    Serial.print("Error reading packet. Error code: ");
    Serial.println(xbee.getResponse().getErrorCode());
   } else {
     Serial.println("No Response from radio");
   }
  }
}

uint8_t gaussian(uint8_t rssi[])
{
  double rssi_sum = 0; //summation
  for(int i = 0; i <= 149; i++)
  {
    rssi_sum += rssi[i];
  }

  int rssi_avg = rssi_sum / 150; //find mean of all 50 values
  return rssi_avg;
}
```

Location System Arduino Mega Final Code

```
uint8_t incomingbyte = 0;
uint8_t incomingbyte2 = 0;
uint8_t incomingbyte3 = 0;
int incomingByte;
int rssi;
char tag1[13] = "4500BE9285EC";
boolean reading = false;
int index;
char tagString[13];
```

```
char tagString1[13];
void setup()
{
 Serial3.begin(9600);
 Serial.begin(9600);
 Serial1.begin(9600);
 pinMode(3, OUTPUT); //power for RFID reader
 digitalWrite(3, HIGH);
 rssi = 0;
 index = 0;
}

void loop()
{
 incomingbyte = 0;
 incomingbyte2 = 0;
 incomingbyte3 = 0;
 incomingByte = 0;

 if(Serial1.available())
 {
    incomingByte = Serial1.read();
  if(incomingByte == 2)
  {
   reading = true;
   Serial.println("Header");
  }
  if(incomingByte == 3)
  {
   reading = false;
   for(int i = 0; i < 12; i++)
   {
    tagString1[i] = tagString[i+1];
    Serial.println("Byte");
    Serial.println(tagString1[i]);
   }
  Serial.println("Footer");
  }
  else if (reading && incomingByte != 10 && incomingByte != 13)
  {
    tagString[index] = incomingByte;
    index++;
  }
 }

 if(Serial3.available())
 {
  incomingbyte = Serial3.read();
  if (incomingbyte < 58)
  {
  incomingbyte = incomingbyte - 48;
  }
  else
  {
   incomingbyte = incomingbyte - 55;
  }
  switch (incomingbyte)
  {
```

```
case 1:
Serial.println("READER1");
incomingbyte2 = Serial3.read();
if (incomingbyte2 < 58)
{
incomingbyte2 = incomingbyte2 - 48;
}
else
{
  incomingbyte2 = incomingbyte2 - 55;
}
 incomingbyte3 = Serial3.read();
if (incomingbyte3 < 58)
{
incomingbyte3 = incomingbyte3 - 48;
}
else
{
  incomingbyte3 = incomingbyte3 - 55;
}
rssi = (incomingbyte2 << 4) | incomingbyte3;
Serial.println(rssi);
rssi = 0;
break;

case 2:
Serial.println("READER2");
 incomingbyte2 = Serial3.read();
if (incomingbyte2 < 58)
{
incomingbyte2 = incomingbyte2 - 48;
}
else
{
  incomingbyte2 = incomingbyte2 - 55;
}
 incomingbyte3 = Serial3.read();
if (incomingbyte3 < 58)
{
incomingbyte3 = incomingbyte3 - 48;
}
else
{
  incomingbyte3 = incomingbyte3 - 55;
}
rssi = (incomingbyte2 << 4) | incomingbyte3;
Serial.println(rssi);
rssi = 0;
break;

case 3:
Serial.println("READER3");
incomingbyte2 = Serial3.read();
if (incomingbyte2 < 58)
{
incomingbyte2 = incomingbyte2 - 48;
}
else
```

```
    {
      incomingbyte2 = incomingbyte2 - 55;
    }
     incomingbyte3 = Serial3.read();
    if (incomingbyte3 < 58)
    {
    incomingbyte3 = incomingbyte3 - 48;
    }
    else
    {
      incomingbyte3 = incomingbyte3 - 55;
    }
    rssi = (incomingbyte2 << 4) | incomingbyte3;
    Serial.println(rssi);
    rssi = 0;
    break;

    default:
    break;
   }
  }
}
boolean checkTag(char tag[], char tagString[])
{
  for(int i = 0; i < 12; i++)
  {
    if(tag[i] != tagString[i])
    {
      Serial.print("i = ");
      Serial.println(i);
      return false;
    }
  }
  return true;
}
```

Appendix D: XBee API Library

Code that was added into the XBee-API Library (Section 8.4.2) to support Digimesh data frames for transmitting and receiving.

Header File

```
class DMTxRequest : public PayloadRequest {
public:
	/*
	Creates tx request with choice of optioon and not the default frameID
	*/
	DMTxRequest(XBeeAddress64 &addr64, uint8_t option, uint8_t *payload, uint8_t payloadLength, uint8_t frameId,
uint8_t radius);
	/*
	Creates tx request with default (ack enabled) ptioon and the default frameID
	*/
	DMTxRequest(XBeeAddress64 &addr64, uint8_t *payload, uint8_t payloadLength);

	DMTxRequest();

	XBeeAddress64& getAddress64();
	void setAddress64(XBeeAddress64& addr64);
	uint8_t getOption();
	void setRadius(uint8_t radius);
	uint8_t getRadius();
	void setOption(uint8_t option);
	uint8_t getFrameData(uint8_t pos);
	uint8_t getFrameDataLength();

private:
	XBeeAddress64 _addr64;
	uint8_t _option;
	uint8_t _radius;
};

class DMRxResponse : public RxResponse {
public:
	DMRxResponse();
	uint8_t getRssiOffset();
	XBeeAddress64& getRemoteAddress64();
private:
	XBeeAddress64 _remoteAddress;
};
```

.CPP File

```
//Digimesh TX request with input of option/frameID/radius
DMTxRequest::DMTxRequest(XBeeAddress64 &addr64, uint8_t option, uint8_t *data, uint8_t dataLength, uint8_t frameId,
uint8_t radius) : PayloadRequest(TX_REQUEST, frameId, data, dataLength, radius)
{
	_addr64 = addr64;
	_option = option;
	_radius = radius;
}

//default option/frameID/radius for DM_TXrequest
```

```
DMTxRequest::DMTxRequest(XBeeAddress64 &addr64, uint8_t *data, uint8_t dataLength) : PayloadRequest(TX_REQUEST,
DEFAULT_FRAME_ID, data, dataLength, RADIUS)
{
	_addr64 = addr64;
	_option = ACK_OPTION;
	_radius = RADIUS;
}

//default constructor
DMTxRequest::DMTxRequest() : PayloadRequest(TX_REQUEST, DEFAULT_FRAME_ID, NULL, 0) {

}
//outputs correct frame for TX request
uint8_t DMTxRequest::getFrameData(uint8_t pos) {
	if (pos == 0) {
		return (_addr64.getMsb() >> 24) & 0xff;
	} else if (pos == 1) {
		return (_addr64.getMsb() >> 16) & 0xff;
	} else if (pos == 2) {
		return (_addr64.getMsb() >> 8) & 0xff;
	} else if (pos == 3) {
		return _addr64.getMsb() & 0xff;
	} else if (pos == 4) {
		return (_addr64.getLsb() >> 24) & 0xff;
	} else if (pos == 5) {
		return (_addr64.getLsb() >> 16) & 0xff;
	} else if (pos == 6) {
		return (_addr64.getLsb() >> 8) & 0xff;
	} else if (pos == 7) {
		return _addr64.getLsb() & 0xff;
	} else if (pos == 8) {
		return RESERVED_1;
	} else if (pos == 9) {
		return RESERVED_2;
	} else if (pos == 10) {
		return _radius;
	} else if (pos == 11) {
		return _option;
	} else {
		return getPayload()[pos - TX_API_LENGTH];
	}
}
XBeeAddress64& DMTxRequest::getAddress64() {
	return _addr64;
}

void DMTxRequest::setAddress64(XBeeAddress64& addr64) {
	_addr64 = addr64;
}

uint8_t DMTxRequest::getOption() {
	return _option;
}

void DMTxRequest::setRadius(uint8_t radius) {
	_radius = radius;
}
```

```
uint8_t DMTxRequest::getRadius() {
	return _radius;
}

void DMTxRequest::setOption(uint8_t option) {
	_option = option;
}

uint8_t DMTxRequest::getFrameDataLength() {
	return TX_API_LENGTH + getPayloadLength();
}

//constructor
DMRxResponse::DMRxResponse() : RxResponse() {
	_remoteAddress = XBeeAddress64();
}

XBeeAddress64& DMRxResponse::getRemoteAddress64() {
	return _remoteAddress;
}

uint8_t DMRxResponse::getRssiOffset() {
	return DM_RSSI_OFFSET;
	}

//framedata starts at 1 to account for frame id coming after api id
void XBeeResponse::getDMRxResponse(XBeeResponse &dmRxResponse) {
	DMRxResponse* dmRx = static_cast<DMRxResponse*>(&dmRxResponse);

	dmRx->setFrameData(getFrameData());
	setCommon(dmRxResponse);

	dmRx->getRemoteAddress64().setMsb((uint32_t(getFrameData()[1]) << 24) + (uint32_t(getFrameData()[2]) << 16) +
(uint16_t(getFrameData()[3]) << 8) + getFrameData()[4]);
	dmRx->getRemoteAddress64().setLsb((uint32_t(getFrameData()[5]) << 24) + (uint32_t(getFrameData()[6]) << 16) +
(uint16_t(getFrameData()[7]) << 8) + getFrameData()[8]);
}
```