Wireless Mesh Networks for SCADA Systems

By Zachary Weisman

Advised by Dr. Hugh Smith

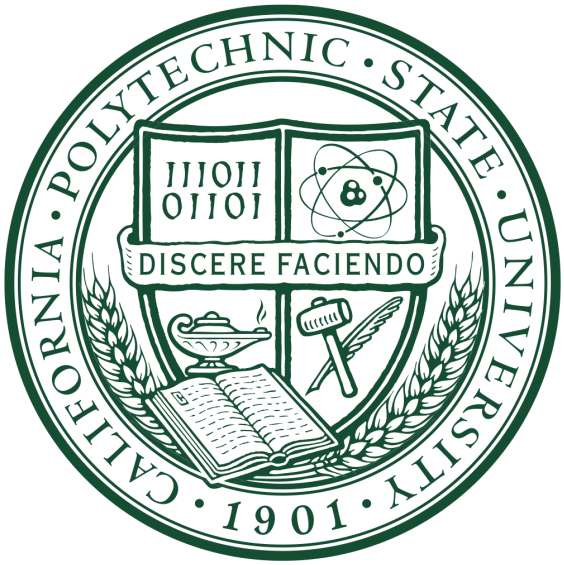


Table of Contents

Introduction

Project Overview

As the solar energy industry continues to grow, so does its need for efficient maintenance techniques. In the industry we use a technology known as Supervisory Control And Data Acquisition, or SCADA for short. SCADA is a technology that is used widely in many industries; from energy production and distribution to manufacturing; it is used to measure the health of large systems that would be difficult, or otherwise impossible to monitor by hand. As it stands, the current trend in SCADA monitoring for solar plants is to use wired solutions. A wired approach can be costly and time consuming, especially for larger multi-megawatt farms. The system would require many thousands of dollars in communication lines, thousands of dollars in repeater systems in order to propagate communications through such a large area, and all of this equipment would need to be buried, which is a huge manual labor undertaking.

My solution is a wireless network that uses a mesh topology provided by the Zigbee network specification. This network is able to provide an affordable, plug-and-play solution to collecting data on an energy production operation. Furthermore, the Zigbee specification provides a self-healing network, and also includes its own routing protocols; more information on that will follow. My work has been carried out under the funding and guidance, and as an intern for SunLink Corporation. SunLink is a long time manufacturer of large-scale ground-mount system (GMS) frames. More recently, SunLink has looked to diversify its portfolio into the business of bringing real-time data to its customers in order to bring them better operation and maintenance performance.

The development of the wireless SCADA system has been a massive undertaking, and is the culmination of years of study in both hardware, and software systems that resulted in about nine months of research and implementation. In this report I will detail all aspects of the project including the challenges faced in research, the implementation of security features, network performance, plant initialization, and the analysis of the hardware platform on which this system runs.

The User

The user of the system I have designed is a software developer. More specifically, I designed a reliable back end for the developer of the plant controller to simply send and receive information to and from the controller of each individual tracker station without having to think about the connection to each node.

SunLink contracted a team of expert software developers to work on the front facing side of the product called Vertex. Vertex is the product that site managers and engineering teams will directly interface with in order to see how their operation is doing, and push commands to it if they wish. These developers worked on a few different things: They were able to retrieve information from a plant controller, store data, and present all the information and options available to the customer in an easy to use GUI.

Project Definition

The need for this product is derived from an industry demand for real time data describing the efficiency of energy production installments. Especially with a solar tracker system, real time data can show us whether or not our tracker is indeed facing the sun at the most optimal angle, or if one of the trackers on the farm is not operating properly. We can even issue commands to force the trackers to any angle we want within its domain. Finally, with the integration of sensors like seismometers, weather stations, irradiance sensors, etc. we are able paint a better picture of why we get certain energy output, and are able to better protect the system from weather.

Objective

The objective of this project is to connect maintenance and site engineering crews at solar farms to data that will help them perform their jobs more efficiently. Efficiency means less time for the crew to be in the field performing service, and less downtime for the system, to allow more time for producing energy. My contribution to this goal is to provide a secure method for wirelessly moving data from each collection node to a solar plant controller that has an internet connection capable of reaching company servers that host this data.

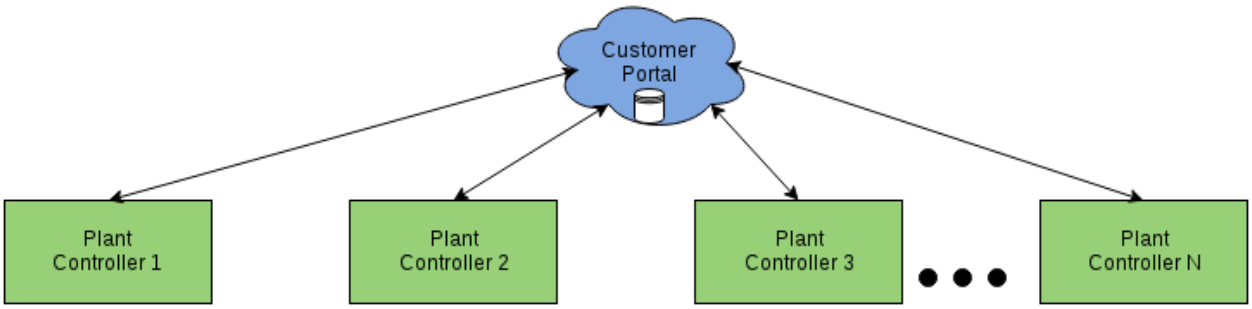


Figure 1. A high level look at a customer interface with N solar sites.

System Requirements

Being an integral part of an energy production system, this network component must meet the highest of reliability standards. To start, the overall tracker system is required to be able to work in all conceivable weather conditions. The most dangerous condition we can foresee is high winds - which carry the possibility of causing some dangerous resonance in the solar panel structure and blowing it apart. Another danger is freezing temperatures wreaking havoc on the hydraulic engine.

More important is the requirements for the network system. This component has three main requirements. One, that the network must be self healing in the event of node failure. Two, the network must reliably, and quickly push and pull data from the plant controller to each tracker node and vice-versa. Finally, we must have a secure network that no outside party is capable of decoding our data, or poisoning our network with false data.

Research

Before implementation of the final product even began, I went through many different iterations of trial implementation, and testing of various technologies to vet the most viable candidate. In this section, I will present many of the problems I discovered needed solving along the way, as well as general ideas that need covering in order to justify my design. During my research I learned a lot about things like 900 MHz and 2.4 GHz band communication, wireless collisions, and security. My solution to all of these problems will be discussed later on in this document.

I.) Frequency Bands

When deciding which wireless technology to use, the decision came down to two choices, either 900 MHz or 2.4 GHz bands. Since our application requires long range communications, I first decided on the 900 MHz option because most of the radios advertised higher transmit power and longer range. (2 options, 900 MHz - limited by line of sight, need active repeater or it passes through the ionosphere or 2.4 GHz) A simple internet search revealed a proliferation of inexpensive 900 MHz UART serial interface radios. One manufacturer stood out - Digi. Digi seemed to have a historical presence in the market, and had many different products to choose from within the ISM band with a wide set of features to choose from.

The ISM 900 MHz band is a radio frequency bandwidth which, according to the FCC (REFERENCE CHART) is an industrial, scientific, and medical radio band. This band is internationally reserved for industrial equipment communications, and occupies a 2 MHz wide band with +/- 0.13 MHz error in its domain. The ISM 2.4 GHz band is packed full of utilities including amateur use, radiolocation, and mobile communications with a +/- 0.50 MHz error per channel.

II.) PPP

At one point I used PPP (Point to Point Protocol) to essentially attempt to establish an ad-hoc TCP connection over the air. PPP is a layer two data link that is supported by an easily configured software suite available to most Linux distributions called PPPD (Point to Point Protocol Daemon). PPP has a fairly robust and secure link establishment protocol, which makes it ideal for our application.

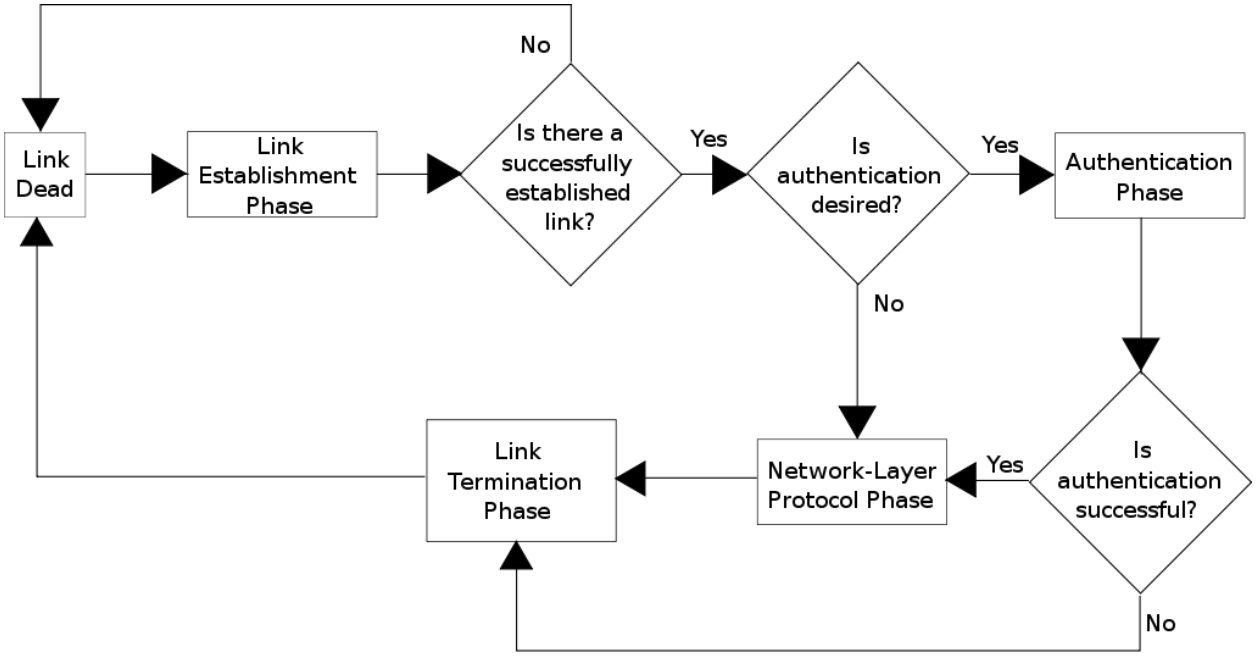


Figure 2. PPP link establishment and teardown diagram.

The nice part about PPP is that the daemon supports IP communication, as well as TCP control. This made it an ideal candidate because I could simply use unix socket API's for easy to implement C code. PPPD has the added benefit that it can act as an IP proxy for serial data coming on on the USB port. All you have to do is assign PPPD to a device (ie /dev/USB0) and it handles the forwarding of traffic, and does its own NAT. I tested PPPD out with a basic multi-user TCP chat program. I started PPPD in verbose mode on one raspberry pi so that it would print out every status message along the way, along with a chat server. Then from another raspberry pi, I handed PPPD the IP address I assigned to PPPD on the server, and connected from there. From this point I was able to do three things:

1.) Successfully ping the PPP gateway on the server node.
2.) Successfully ping www.google.com from the client.
3.) Send chat server status requests, and get a response.

From here I knew that PPP was going to make this project very easy, but when I tried to connect another client, PPPD successfully connected the new client, but I could in the terminal output that it got hung up in the process of establishment of the new link. This caused the first client to lose its connection, and the server completely locked up. At this point I determined that although PPP supports multilink, the daemon did not, and I could not find any other applications that did support multilink. The only way to proceed at this point would be to write some obnoxious bash scripts to continuously start and stop new links with PPPD with new clients, which is a less than ideal solution to the problem. Also, there were no development libraries available for PPP, so I decided to abandon the idea.

III.) 6LoWPAN

IPv6 Low Power Area Network (6LowPan) is a relatively new working group in the internet area of the IETF. This group has specified encapsulation and header compression for IPv6 packets sent over IEEE 802.15.4 networks. 6LoWPAN is an ideal technology for this application because it provides IP layer support, which would allow the use of unix socket API calls for easy software implementation, security addition with SSL, as well added reliability. The issue is that market for these radios right now is very expensive, and the current products do not quite have the range that I was looking for for this project.

IV.) Wireless Collision

My first iteration of prototyping began on the Digi XBee-PRO line of radios. In this iteration I was testing out the ability to send AT commands to the radio. AT commands are essentially firmware setting change requests. During this phase I set up some tests to examine the ability of the radios to receive messages under high traffic load. With a setup of four nodes sending to one single node, I let each node send a message as fast as they could. What happened was that the receiving node would only receive from one node while the other three nodes messages were lost.

This proved a lesson in wireless collisions. Because the radios I had featured no intelligent firmware set or API, colliding messages had no way of knowing that they were not being received, which is obviously an issue for a system of many nodes trying to compete to contact a single node. I had essentially discovered that these were half-duplex radios.

Finally, I tried using the 2.4 GHz radios that had a much broader feature set, including support for the Zigbee protocol, and an active API to help. After a little bit of testing, I found that the Zigbee modules supported message receipts, which notified the sender of whether or not a message was successfully received by its intended recipient. This did not completely solve the issue of collision avoidance, detection, or correction, but it did provide a basis for handling the loss.

V.) Zigbee Protocol

The zigbee specification provides three very important tools for our application. First, it provides a self-healing mesh network using ADOV (Ad hoc On-Demand Distance Vector) Routing, specified in RFC 3561 (http://www.rfwireless-world.com/Tutorials/Zigbee_tutorial.html).

Second, it provides us with a protocol that lets us know the status of our messages, and a convenient packet structure that follows. In this application, we are concerned with three different message types (although zigbee specifies many more). We have the transmit request, transmit status, and the receive packet types.

TX Request

Byte Order	**Description**
0	Start Delimiter (Always 0x7E)
1-2	Length
3	Frame Type (0x10)
4	Frame ID (Sequence Number)
5-12	64-Bit Destination Address
13 - 14	16-Bit Destination Address
15	Broadcast Radius (0x00)
16	Options (0x00)
17 - (17 + (Length - 14))	Message Data
Last Byte	Checksum

Table 1. TX Request Structure

TX Status

Byte Order	**Description**
0	Start Delimiter (Always 0x7E)
1-2	Length
3	Frame Type (0x8B)
4	Frame ID (Sequence Number)
5-6	16-Bit Destination Address
7	Transmit Retry Count
8	Delivery Status
9	Discovery Status

10	Checksum

Table 2. TX Status Structure

Receive Packet

Byte Order	**Description**
0	Start Delimiter (Always 0x7E)
1-2	Length
3	Frame Type (0x90)
4-11	64-Bit Source Address
12-13	16-Bit Source Address
14	Receive Options
15 - (15+(Length-12))	Received Message
Last Byte	Checksum

Table 3. Receive Packet Structure

Finally, Zigbee provides us with a protocol that notifies us whether or not our message was received. This is handy in implementation because all we have to do after transmitting is wait for either an ACK or a NACK message.

This protocol also specifies three types of nodes. We have Coordinators, Routers, and End Points (Nodes). The structure is simple, each network needs exactly one Coordinator in order to set up the network, store node information, and route messages between nodes. A single zigbee network doesn't need to have any routers, as the coordinator can handle the job of routing messages, but if a widely expandable network is desired, routers are necessary.

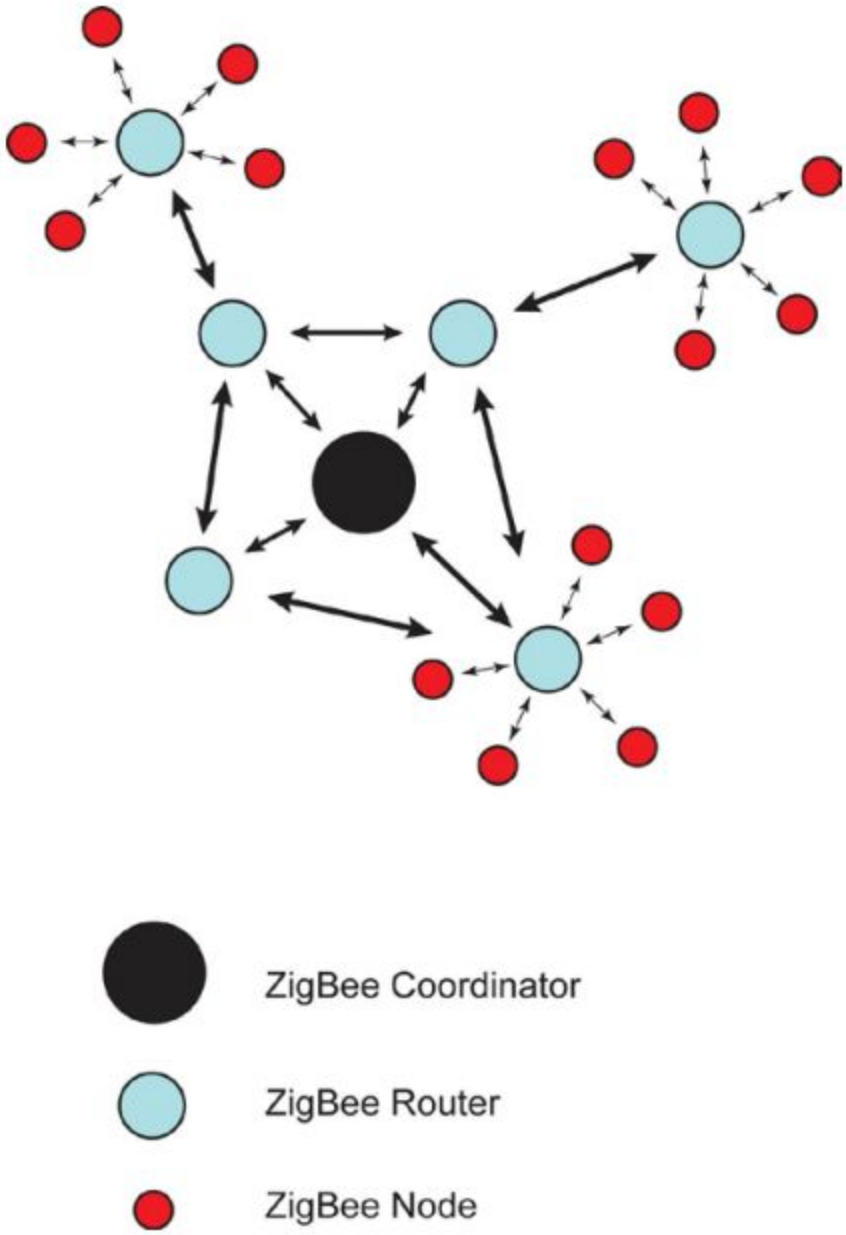


Figure 3. Zigbee Network Diagram. Retrieved from www.l-com.com

VI.) Security

When it came to security, I had to get a little creative with what kind of attacks I could imagine would happen. The first, and fairly obvious attack was a data poisoning attack: an attack where a user could join the network with a Zigbee, or any other kind of dynamic bandwidth radio like a WiFi Pineapple and feed false data into the system. On

the same note, a user that is able to join the network could also simply record data that the plant operator might want to stay private.

Design

Platform

My communications platform only consisted of a few parts:

1.) Raspberry Pi 2 Model B - $39.95
2.) Digi XBee Zigbee Pro S2C 2.4 GHz Radio - $28.50
3.) Sparkfun XBee Explorer USB Dongle - $24.95

In order to kick off the explanation of this project, let's talk about the core of the system: The tracker. A single tracker at its core is a large hydraulic engine capable of driving up to 62 rows of 60 solar modules each on a single axis. The purpose of this hydraulic engine is to point the solar panels towards the sun as it the Earth rotates through the day.

Tilt Range (East/West)	+/- 52.5°
Power Output	Up to 1.2 MWp DC
Wind Load	150 mph with no snow load, 120 with up to 50 psf snow load
Nominal Input Voltage	480 VAC (3 phase)
Input Voltage Range	208 - 480 VAC (3 phase)
Max. Power Draw	4.2 kW
Parasitic Load	< 0.03%

Table 4. Tracker Specs

Each tracker includes a combiner box. The combiner box houses all of the power regulation electronics, as well as another component more relevant to our discussion - the programmable logic controller (PLC). In our combiner box we use an IDEC MicroSmart Pentra, which is essentially the brain of the tracker. The IDEC PLC features a USB programming port, as well as an ethernet port for Modbus-TCP communication.

PLC

Modbus is a serial communication protocol is widely used throughout industry to transmit signals from instrumentation and control devices back to a main controller or data gathering system. In our situation, we are reading sensor data, as well as pushing angle update information when necessary. To the programmer, reading Modbus TCP data, the PLC is essentially just a register bank. One of SunLink's electrical engineers programmed the PLC to keep the following information on hand:

1.) Update Interval
2.) Longitude
3.) Latitude
4.) Time Zone
5.) Module width (width of the individual solar panel)
6.) Row spacing (Distance between each row of panels per tracker)
7.) Current tracker tilt angle
8.) East and West angle limits of tilt
9.) Wind speed upper and lower threshold for stow mode
10.) Tilt bands

In addition, certain registers are reserved for holding fault bits, which are set in the event of a hardware fault, and some for future sensor data. Each register is 16 bits, so each 32-bit float is stored in two sequential registers.

The PLC is undoubtedly the most important part of the tracker system. The tracker can operate without any of the wireless communication, as it is pre-programmed to send tilt-angle updates to the hydraulic engine throughout the day via the Solar Position Algorithm developed by NREL. NREL's algorithm specifies how to determine the position of the sun given a set of coordinates, date, and a real time clock.

Raspberry Pi

As the main computing core of the combiner box, I chose the Raspberry Pi Model 2 B. The Raspberry Pi is perhaps one of the world's greatest full featured, low-cost ARM Linux prototyping device. This device was primarily chosen because of its ease of use as a prototype platform. It was simple to boot into a full Debian installation and jump right into testing. In addition, the Raspberry Pi 2 has a few hardware specs that are of particular interest to us.

1.) 900 MHz quad-core ARM Cortex-A7 CPU
2.) 1 GB RAM
3.) 40 GPIO pins
4.) Ethernet Port

Recently, Raspbian released a special “Lite” version of their kernel which removed all of the GUI support, making for a much lighter weight operating system. It does not run any X server daemons or include any other unnecessary graphics related processes that could interrupt normal operation of an industrial system. Although a microcontroller would have sufficed for the implementation of this project, I decided on the Raspberry Pi because the Linux distribution allowed for simple installation of any development library that I needed, and because of its many interfaces, including the GPIO, USB, and ethernet ports.

Zigbee Radio

The Digi XBee Zigbee PRO RF module is a 20-pin radio module that comes in two different flavors. One comes with a simple wire antenna, and the other comes with an RPSMA (Reverse Polarity SMA) connector; the main difference is that the RPSMA version has slightly higher gain than a wire antenna. RPSMA antennas are usually one-quarter wave dipole - although there are not any performance sheets available, wire antennas are usually less.

Performance information on this radio module can be found in Appendix C. The main bit of information we are concerned with is the outdoor transmission range, and data rate - both of which are satisfactory for this application.

Sparkfun XBee Explorer USB Dongle

The USB dongle was an unplanned component, but very important nonetheless. It allowed for very simple integration of the radio onto the raspberry pi through the USB port rather than having to use the UART interface. The main component of interest on the Explorer dongle is the FT 231X USB to Serial Converter chip. The FT 231X is a full-speed USB 2.0 bridge to handle handshake to UART interfaces. This allows us to essentially ignore the UART connection all together in software.

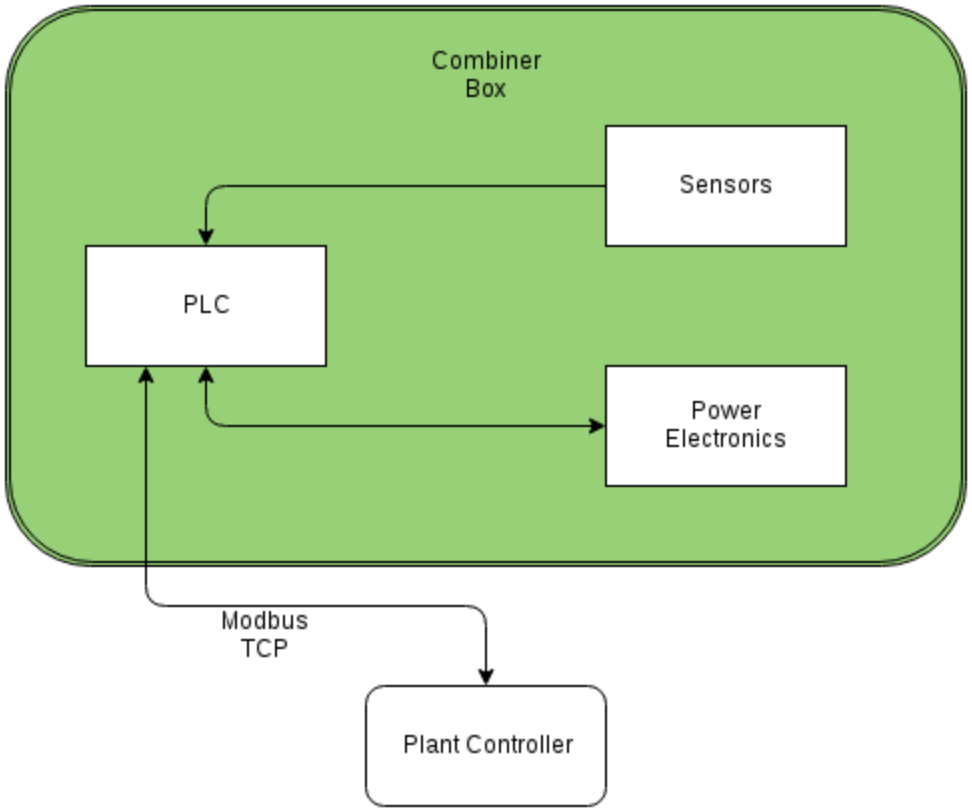


Figure 4. System diagram with wired connection

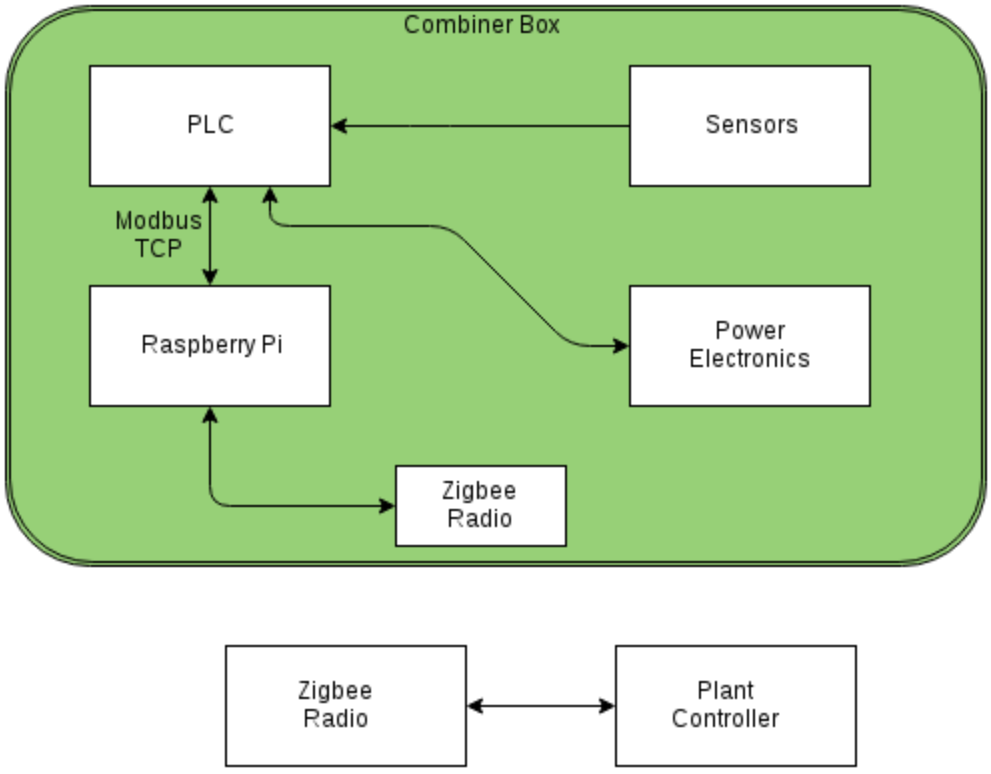


Figure 5. System diagram with new wireless system

Implementation

I.) PLC

The first step in implementation was to figure out how to pull data off of the PLC using the Raspberry Pi. Fortunately, there was a package available in the Debian repositories called libmodbus. Libmodbus provides a few very easy to use functions to connect to a modbus device that is already assigned an IP address.

```
modbus_t* modbus_new_tcp(char* ipAddr, uint16_t port); // Creates a reference to a //modbus
tcp object.
int modbus_connect(modbus_t* mb); // Attempts to make connection to modbus device
void modbus_free(modbus_t* mb); // Frees the modbus object
void modbus_read_registers(modbus_t* mb, int reg, int num_regs, void* dest); // Reads //
num_regs registers starting at register reg into the array pointed to by dest.
void modbus_write_registers(modbus_t* mb, int reg, int num_regs, void* src); // Writes
//num_regs registers starting at register reg from array src
void modbus_close(modbus_t* mb); // Closes modbus TCP connection
```

II.) I/O

One of the biggest challenges I faced getting started was being able to communicate with the radios. My primary goal was to get the radio to respond with “OK” when I sent it “+++” over UART. The “+++” is the method by which a programmer enters the radio into AT mode - the mode which allows the user to change the radio's settings.

The Raspberry Pi 2 comes with two sets of TX/RX UART pins, but only one set of RTS/CTS pins. Unfortunately the flow control pins are not enabled by default on the Raspberry Pi, and I needed them to enable communication with the radio. After a couple weeks of searching through forums, I could not find a solution to enabling those pins without having to explore the kernel source.

At this point I decided to buy the Explorer USB dongle, because I knew that all of the flow control could be handled through the USB ports. After plugging everything up, I used a utility called Minicom to test the connection. Minicom is a serial interface terminal that allows you to type characters directly into your serial connection. Initially, I tested the following commands in order:

Command	Result
“+++” (enter AT mode)	“OK\n”
“ATSH\r” (Request high 4 bytes of MAC)	“13A200\n”
“ATSL\r” (Request low 4 bytes of MAC)	“40D99BB4\n”
“ATCN\r” (Exit AT mode)	“OK\n”

Table 5. Primary AT Commands for Firmware Configuration

Once I was confident in how to use the AT interface, I started looking for C libraries for serial interfacing. A popular choice among Raspberry Pi users is wiringSerial. This library lets you easily establish a serial connection, and communicate using a few simple functions.

```
int serialOpen(const char* path, int baud); // Returns a file descriptor
void serialPutchar (int fd, unsigned char c); // Sends a single byte
int serialDataAvail (int fd); // Returns number of bytes available
int serialGetchar(int fd); // Reads character from input buffer
void serialFlush(int fd); // Flushes all waiting data
```

III.) Firmware Configuration

The next step was to determine what firmware settings were required to create our network. As I mentioned before, our zigbee network needed three things: a coordinator, a router, and some endpoints. In order to configure these, there are few parameters that need to be set, and in order to do this, we need to send AT commands to the radio using the commands listed in Appendix E.

First we have the PAN (Personal Area Network) ID. All devices on a single network must have the same PAN ID, or else they will not be seen by the network; this is done using the ATID command. Next, we need to enable the coordinator radio using ATCE. This should be set to 1 for the coordinator, 0 for all other radios. Also, channel verification should be on using ATJV, and that concludes the network configuration portion.

Next we should configure our serial interfacing options like baud rate, CTS, RTS, and API mode. To do this we need ATBD, ATD7, ATD6, and ATAP. ATD7 and ATD6 are both enabled. These two flags enable RTS and CTS. ATAP defines that we are

enabling API mode 1 which essentially puts the radio into zigbee specification mode, rather than raw byte broadcasting.

Now for our application we need to enable AES encryption, so we use ATEE to enable it, and then use ATKY to enter the 128-bit encryption key in hex format. The key should be the same across all devices on the network.

Finally, we should differentiate a router and an end device by setting the sleep mode. For coordinators and routers, sleep mode (ATSM) should be set to 0, or no sleep. End devices should be set to anything but 0, but for this application I used mode 4, which is cyclic sleep mode. All other should be left to default.

IV.) Security

Finally, security was really the meat and potatoes of this project. Once I figured out how to setup the network, the real challenge was to make sure nobody could break in. Security implementation ranged all the way from the network start up process, to encrypting and decrypting every message. So let's start with the startup process.

On startup, both the plant controller and each node expect a file that specifies the approved hardware addresses that are approved on the network. Both will start, read this file, and populate a list that associates both the 64-bit and 16-bit with an index. In addition, the plant controller will need to be prepared prior to deployment with the public RSA keys for every node in the network in the same working directory as the binary. When it starts up, and reads the configuration file, it will also read a string name that indicates the prefix for the public key file so have to read its own private RSA key and keep a reference to it at all times in an OpenSSL RSA pointer. The following is an example of a valid configuration file:

approved_macs.cfg

```
2
END_DEVICE_1 0 19 162 0 64 217 155 180 31 208
END_DEVICE_2 0 19 162 0 64 217 155 158 234 37
```

The contents of that directory would look like:

ls -l

```
plant_controller
node
END_DEVICE_1.pem
END_DEVICE_2.pem
private.pem
approved_macs.cfg
```

Similarly, each node will keep a copy of a similar configuration file that specifies the plant controllers hardware addresses. Now, to decompose the configuration file, there are ten integers. The first eight numbers are scanned, and casted as uint8_t's to fit into an array that represents the 64-bit hardware address, and the last two numbers are scanned and casted into an array that represent the 16-bit hardware address.

This whole scheme is part of an effort to avoid having to use manual network discovery by waiting for nodes to contact the plant controller and then register their hardware address. In such a case, it would be easy for a malicious user to register their device as an approved device if they happened to come onto the network at the same time as all the other nodes.

At this point, our network provisioning process is protected so long as no malicious users are able to gain access to whatever process is used to distribute this information to every Raspberry Pi before the plant is even set up. Our next line of defense is our hardware AES encryption, this is another task that should be taken care of before deployment.

As an added layer, I also used the OpenSSL API to implement RSA encryption and decryption. Before each message is sent, the payload portion of the zigbee transmit request is encrypted with the intended recipient's public key. Upon reception, the message payload is decrypted using that node's private key before being analyzed.

In order to protect against playback attacks, I needed a way to keep track of when a message was sent. At the beginning of each connection, the plant controller will

generate a random 32-bit int with which the plant controller and that node will start their sequence counting from. Each message response should increment the counter by one, so that both nodes know what number to expect. Another important thing to do is to authenticate the actual data being sent. To do this, I needed a reference point to compare the decrypted data to. What I did was create my own message digest that consists of both the senders 16-bit hardware address, and the expected randomly seeded sequence number. In order to validate the message, I decrypt the message, and calculate a new checksum for the message itself (so two total checksums for the entire packet). The Zigbee checksum is simple to calculate. It is only one byte, and all one has to do is sum all the bytes in the message, and only take the least significant eight bits.

Spoofing a hardware address is easy because that part of the message is completely transparent, but since this message digest is protected by AES 128 and RSA encryption, it would be very difficult to crack.

Here is the basic process for receiving a message:

1.) Calculate checksum, reject packet if incorrect.
2.) Read 64-bit and 16-bit source address. Retrieve private key and decrypt message.
3.) Calculate checksum of encapsulated message with message digest.
    a.) If authentic, accept and read message
    b.) If not, discard message

Here is my basic message format that is encapsulated inside the Zigbee message data

Byte Order	**Description**
0-1	16-Bit Source Address
3-4	Sequence Number
5-6	Type
6-7	Length (Of data portion only)
8-(8+length-1)	Data
8+length	Checksum

Table 6. Custom Packet Structure

The initialization process is fairly straightforward, and involves only four messages. The plant controller will send the node a message with sequence number of zero, and a message type of 0x0001, to which the node responds with a similar message. Next, the plant controller sends the node a randomly generated sequence start number. The node would respond with an incremented sequence number, thus the process would be complete. If at any point during this process the node does not respond, then a flag is raised for that node on the plant controller, notifying the user.

Integration and Testing

The testing phase consisted mainly of using software provided by the radio manufacturer. I used a total of seven radios in order to test my final design. This set up consisted of one radio in coordinator mode, two configured as routers, four configured as endpoints to mock trackers and one endpoint that mocked a plant controller.

1.) Testing Firmware configuration using XCTU
2.) Testing encryption.
3.) Testing overall network.

In the early stages of network testing, I used XCTU to send simple “Hello, World” messages between nodes to ensure that I had configured the firmware settings on each radio properly. XCTU is Digi’s proprietary platform testing application that allows the user to easily change firmware settings, build networks, and send messages from all within a GUI.

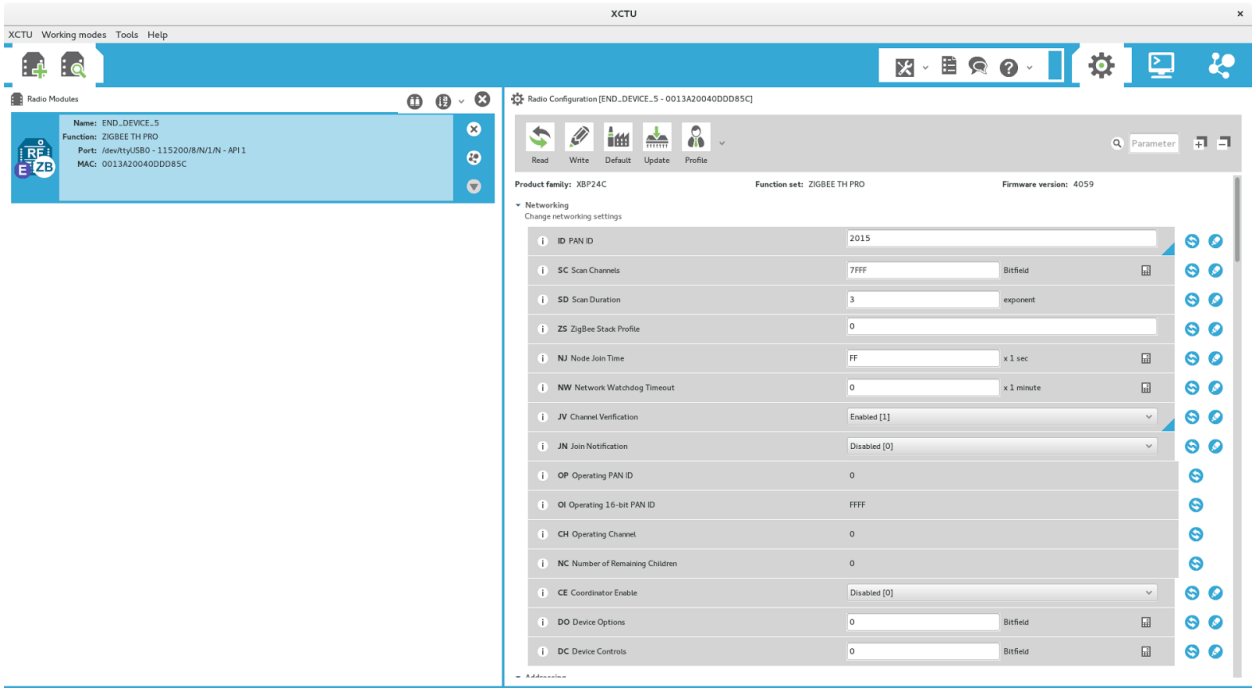


Figure 6. XCTU Firmware Configuration Page

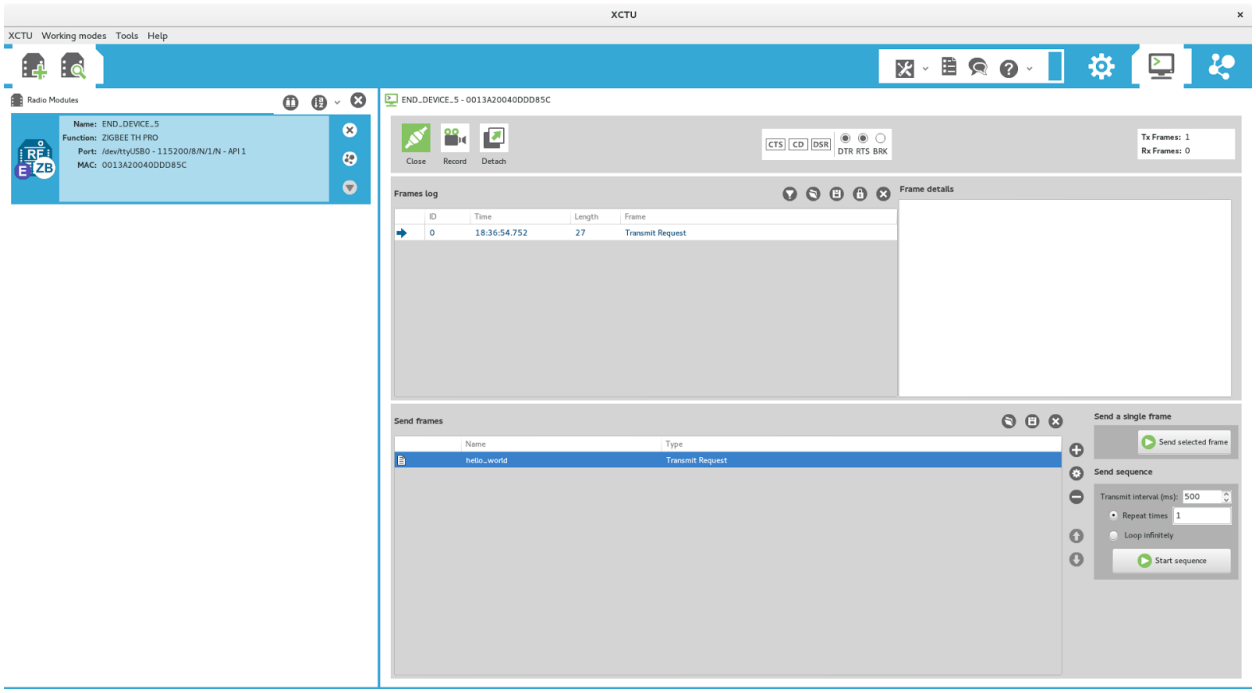


Figure 7.Frame Builder Page

In figure 7, we see the page that enables us to build whatever kind of zigbee frames we want. For our application, all we had to do was choose whether we wanted to send a transmit request, transmit status, or receive packet type. After that all you need to do is fill out the destination address information, and put in a message. XCTU allows

you to set up a sequence of packets in any interval you want, in case you want to test sequencing, or network saturation.

The encryption testing was fairly easy. I wrote two functions, one for encrypting a message with a public key, and another for decrypting with a private key. Testing simply involved building a few message arrays, encrypting and then decrypting them, and verifying that the message was still the same.

```
int pub_encrypt(uint8_t* targetbuf, uint8_t* srcbuf, uint16_t size, RSA* pubkey);
int private_decrypt(uint8_t* targetbuf, uint8_t* srcbuf, uint16_t size, RSA* privatekey);
```

After the communication and security protocol was established, I used three different configurations to ensure the network behaved as expected. The first I used only a coordinator, the second with a single router, and the third with two routers in which I forced the messages to go through both routers to ensure end to end functionality. The zigbee self-healing feature makes each end device automatically elect a new router depending on signal strength (RSSI between 0 and 255). So, to test make sure messages were passing through the desired routers, I used some RF shielding material to reduce the signal strength between nodes.

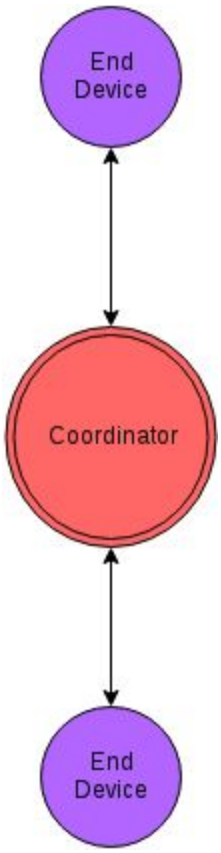


Figure 8. Test setup with only a single coordinator as a router

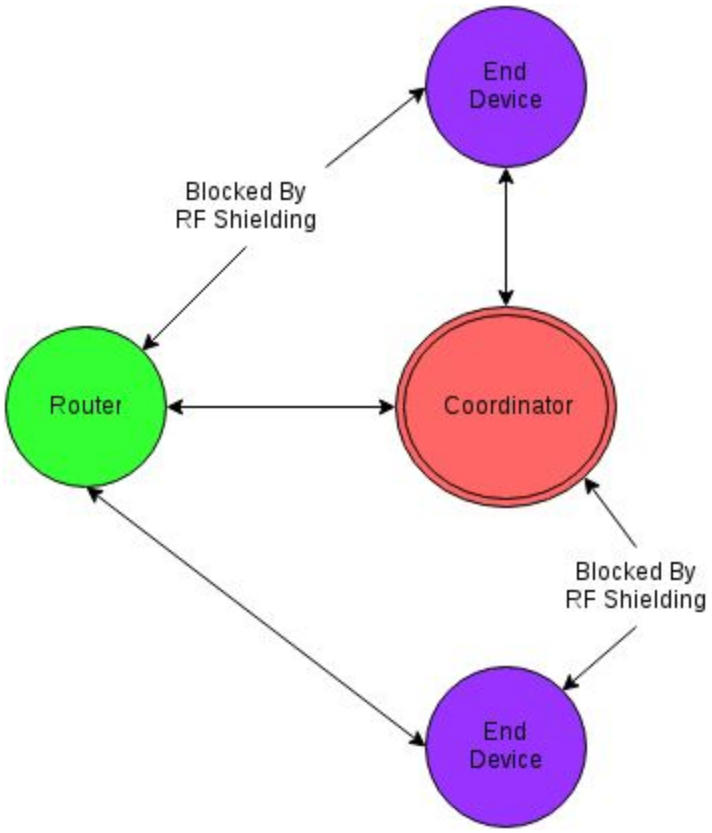


Figure 9. Single Router

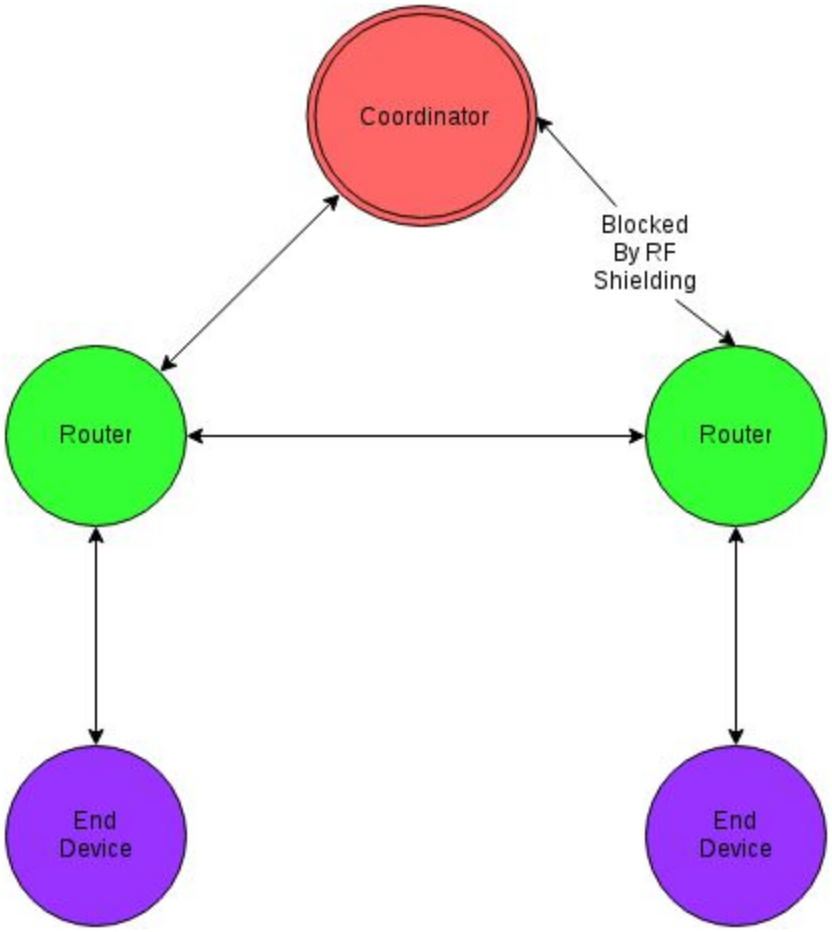


Figure 10. Two router setup

References

1.) Bhondekar, Amol P., and Harmanpreet Kuar. "Routing Protocols in Zigbee Based Networks: A Survery." *www.researchgate.net*. 29 Apr. 2015. Web. 12 Mar. 2016. <https://www.researchgate.net/publication/275637579_Routing_Protocols_in_Zigbee_Based_networks_A_Survey>.

2.) OpenSSL. "Crypto Library." *OpenSSL Cryptography and SSL/TLS Toolkit*. 03 May 2016. Web. <https://www.openssl.org/docs/man1.0.2/crypto/>.

3.) FTDI Chip. "FT231X USB to Full Handshake UART IC Datasheet Version 1.2." *www.ftdichip.com*. 15 Feb. 2013. Web. <http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT231X.pdf>.

4.) Texas Instruments. "Overview for 6LoWPAN." *Www.ti.com*. Web. <http://www.ti.com/lsds/ti/wireless_connectivity/6lowpan/overview.page>.

5.) Montenegro, G., Microsoft Corporation, N. Kushalnagar, Intel Corp, J. Hui, D. Culler, and Arch Rock Corp. "Transmission of IPv6 Packets over IEEE 802.15.4 Networks." *Tools.ietf.org*. Sept. 2007. Web. <https://tools.ietf.org/html/rfc4944>.

6.) Perkins, C., Nokia Research Center, E. Belding-Royer, University of California, Santa Barbara, S. Das, and University of Cincinnati. "Ad Hoc On-Demand Distance Vector (AODV) Routing."*Www.ietf.org*. July 2003. Web. <http://www.ietf.org/rfc/rfc3561.txt>.

7.) McNeil, Peter. "Industrial Wireless Mesh Network Architectures." *Www.l-com.com*. Web. <https://www.l-com.com/multimedia/whitepapers/wp_Wireless-Industrial-Mesh-Networks.pdf>.

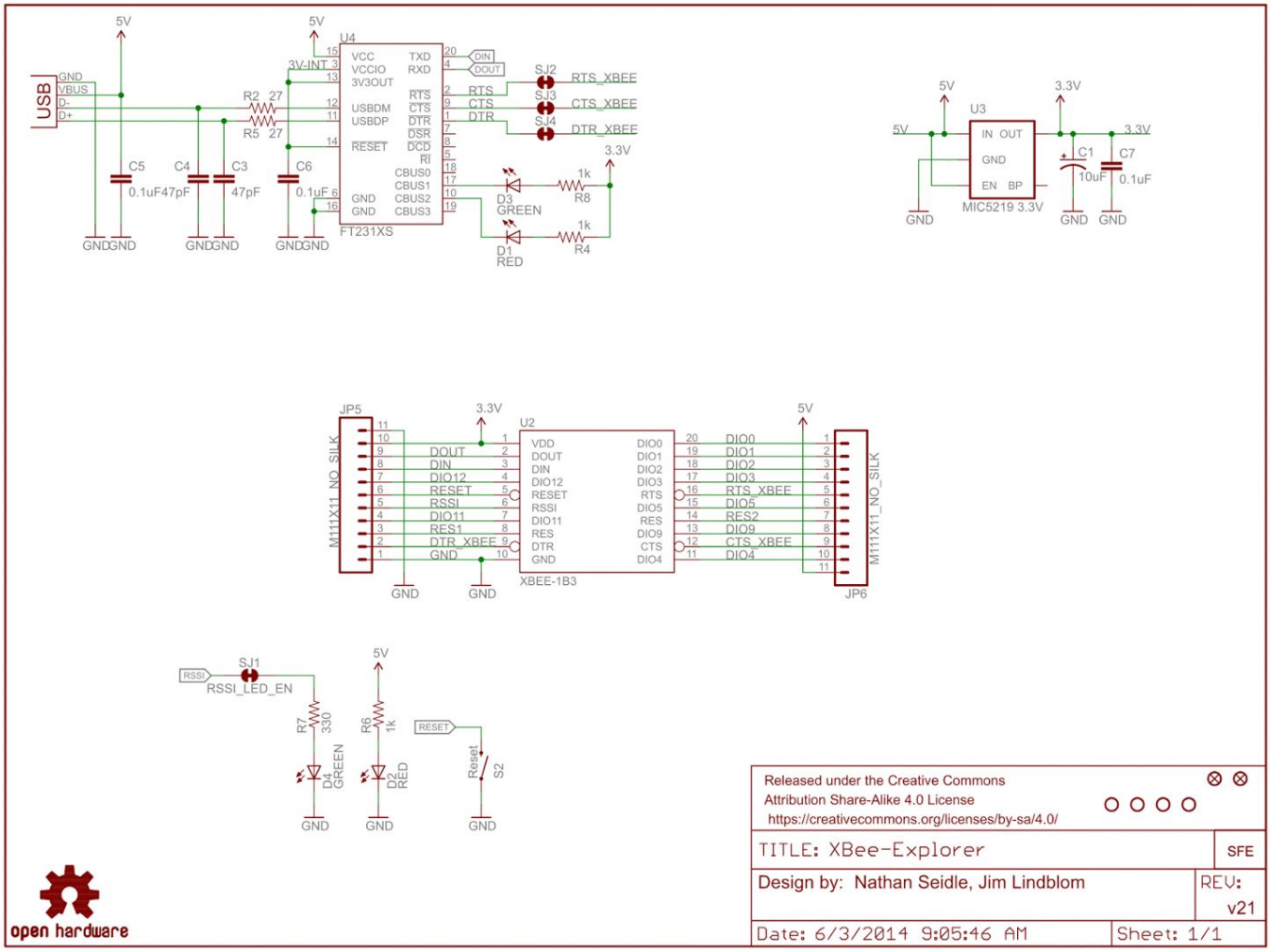
8.) Piyare, Rajeev, and Seeong-ro Lee. "Performance Analysis of XBee ZB Module Based Wireless Sensor Networks." *Http://www.ijser.org/*. Apr. 2013. Web. <http://www.ijser.org/researchpaper/Performance-Analysis-of-XBee-ZB-Module-Based-Wireless-Sensor-Networks.pdf>.

9.) Reda, Ibrahim, and Afshin Andreas. "Solar Position Algorithm for Solar Radiation Applications."*Www.nrel.gov*. Jan. 2008. Web. <http://www.nrel.gov/docs/fy08osti/34302.pdf>.

10.) Libmodbus.org. "Libmodbus Documentation Page." *Libmodbus*. Web. <http://libmodbus.org/documentation/>.

Appendix A: XBee Explorer Dongle Pinout Schematic

Retrieved from https://cdn.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Explorer-v21b.pdf



Appendix B: The following entry is an excerpt from the XBee Zigbee Pro radio user's manual.

Retrieved from http://ftp1.digi.com/support/documentation/90002002.pdf

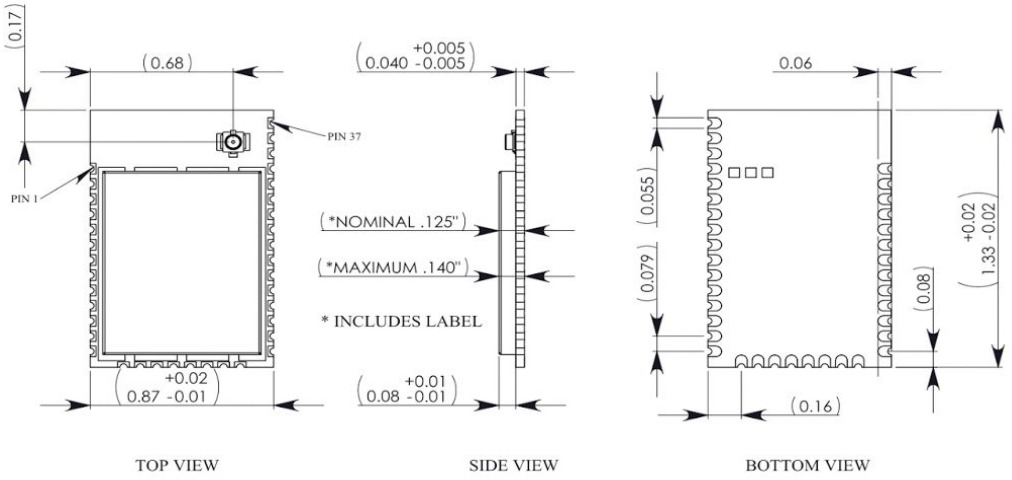
Hardware specifications for the programmable variant

If the module has the programmable secondary processor, add the following table values to the specifications listed on page 8. For example, if the secondary processor is running at 20 MHz and the primary processor is in receive mode then the new current value will be $I_{total} = I_{r2} + I_{rx} = 14\ mA + 9\ mA = 23\ mA$, where I_{r2} is the runtime current of the secondary processor and I_{rx} is the receive current of the primary.

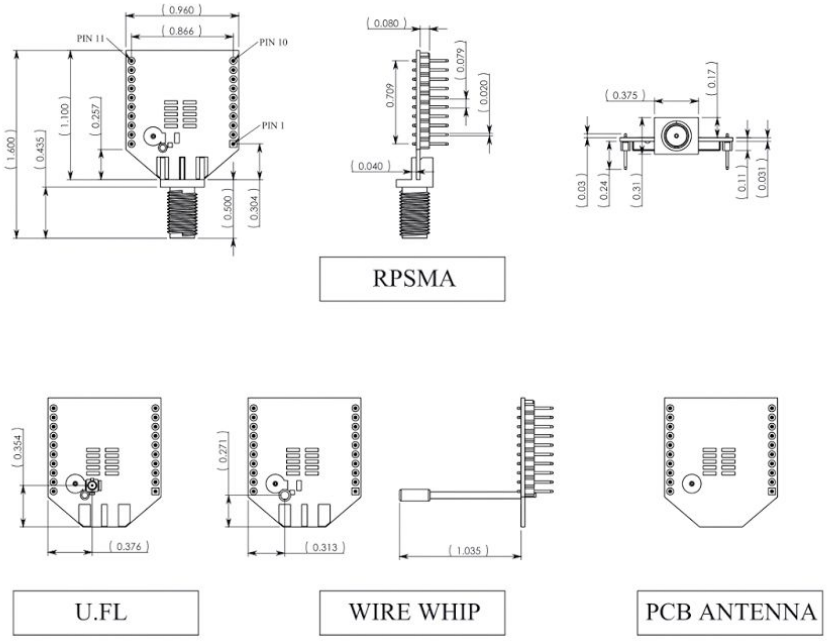
Optional Secondary Processor Specification	These numbers add to specifications (Add to RX, TX, and sleep currents depending on mode of operation)
Runtime current for 32k running at 20MHz	+14mA
Runtime current for 32k running at 1MHz	+1mA
Sleep current	+0.5µA typical
For additional specifications see Freescale Datasheet and Manual	MC9S08QE32
Minimum Reset low pulse time for EM357	+26µS
VREF Range	1.8VDC to VCC

Mechanical drawings

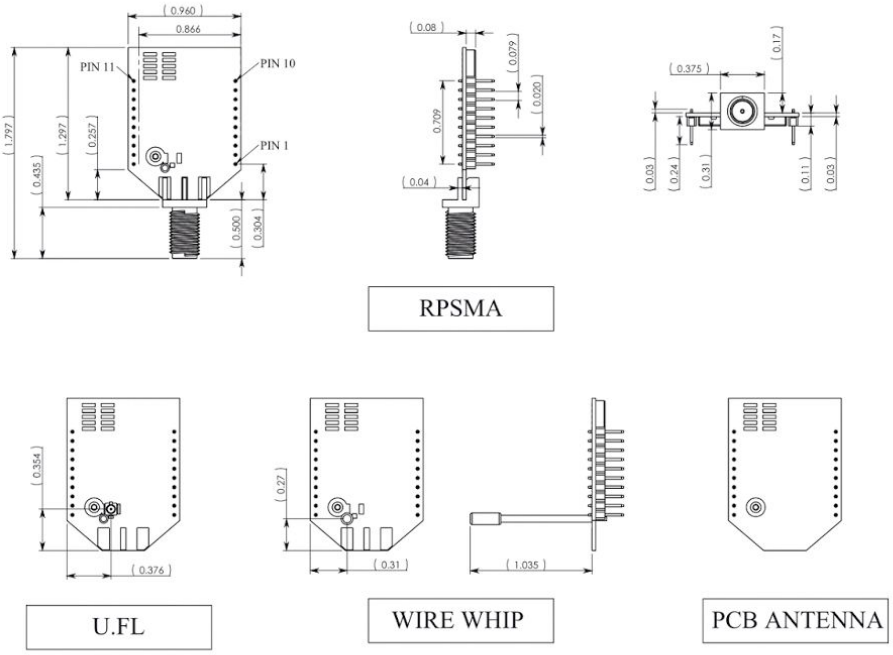
The following mechanical drawings of the XBee/XBee-PRO ZB RF Modules show all dimensions in inches. The first drawing shows the SMT model (antenna options not shown).



The drawings below show the XBee TH module.


0.960
0.866
PIN 11
PIN 10
PIN 1
1.600
0.435
1.100
0.257
0.500
0.304
0.080
0.079
0.020
0.709
0.040
0.375
0.17
0.03
0.24
0.31
0.11
0.031
RPSMA
0.354
0.376
0.271
0.313
1.035
U.FL
WIRE WHIP
PCB ANTENNA

The drawings below show the XBee-PRO TH model.



Pin signals for the surface mount module

Pin #	Name	Direction	Default State	Description
1	GND	-	-	Ground
2	VCC	-	-	Power Supply
3	DOUT / DIO13	Both	Output	UART Data Out / GPIO
4	DIN / $\overline{\text{CONFIG}}$ / DIO14	Both	Input	UART Data In / GPIO
5	DIO12	Both		GPIO
6	$\overline{\text{RESET}}$	Input		Module Reset
7	RSSI PWM / DIO10	Both	Output	RX Signal Strength Indicator / GPIO
8	PWM1 / DIO11	Both	Disabled	Pulse Width Modulator / GPIO
9	[reserved]	-	Disabled	Do Not Connect
10	$\overline{\text{DTR}}$ / SLEEP_RQ / DIO8	Both	Input	Pin Sleep Control Line / GPIO
11	GND	-	-	Ground

Pin #	Name	Direction	Default State	Description
12	SPI_$\overline{\text{ATTN}}$ / $\overline{\text{BOOTMODE}}$ / DIO19	Output	Output	Serial Peripheral Interface Attention Do not tie low on reset
13	GND	-	-	Ground
14	SPI_CLK / DIO18	Input	Input	Serial Peripheral Interface Clock / GPIO
15	SPI_$\overline{\text{SSEL}}$ / DIO 17	Input	Input	Serial Peripheral Interface not Select / GPIO
16	SPI_MOSI / DIO16	Input	Input	Serial Peripheral Interface Data In / GPIO
17	SPI_MISO / DIO15	Output	Output	Serial Peripheral Interface Data Out / GPIO
18	[reserved]*	-	Disabled	Do Not Connect
19	[reserved]*	-	Disabled	Do Not Connect
20	[reserved]*	-	Disabled	Do Not Connect
21	[reserved]*	-	Disabled	Do Not Connect
22	GND	-	-	Ground
23	[reserved]	-	Disabled	Do Not Connect
24	DIO4	Both	Disabled	GPIO
25	$\overline{\text{CTS}}$ / DIO7	Both	Output	Clear to Send Flow Control / GPIO
26	ON / $\overline{\text{SLEEP}}$ / DIO9	Both	Output	Module Status Indicator / GPIO
27	VREF	Input	-	Not used for EM357. Used for programmable secondary processor. For compatibility with other XBee modules, we recommend connecting this pin to the voltage reference if Analog Sampling is desired. Otherwise, connect to GND.
28	ASSOCIATE / DIO5	Both	Output	Associate Indicator / GPIO
29	$\overline{\text{RTS}}$ / DIO6	Both	Input	Request to Send Flow Control / GPIO
30	AD3 / DIO3	Both	Disabled	Analog Input / GPIO
31	AD2 / DIO2	Both	Disabled	Analog Input / GPIO
32	AD1 / DIO1	Both	Disabled	Analog Input / GPIO
33	AD0 / DIO0	Both	Input	Analog Input / GPIO / Commissioning Button
34	[reserved]	-	Disabled	Do Not Connect
35	GND	-	-	Ground
36	RF	Both	-	RF IO for RF Pad Variant

Pin #	Name	Direction	Default State	Description
37	[reserved]	-	Disabled	Do Not Connect

Signal Direction is specified with respect to the module

See Design notes for SMT RF pad modules on page 25 for details on pin connections

* Refer to the Writing Custom Firmware section for instructions on using these pins if JTAG functions are needed

Pin signals for the through-hole module

Pin #	Name	Direction	Default State	Description
1	VCC	-	-	Power Supply
2	DOUT / DIO13	Both	Output	UART Data Out
3	DIN / $\overline{\text{CONFIG}}$ / DIO14	Both	Input	UART Data In
4	DIO12 / SPI_MISO	Both	Disabled	GPIO/ SPI slave out
5	$\overline{\text{RESET}}$	Input	Input	Module Reset
6	RSSI PWM / PWMO DIO10	Both	Output	RX signal strength indicator / GPIO
7	PWM1 / DIO11	Both	Disabled	GPIO
8	[reserved]	-	-	Do Not Connect
9	$\overline{\text{DTR}}$ / SLEEP_RQ / DIO8	Both	Input	Pin Sleep Control Line / GPIO
10	GND	-	-	Ground
11	SPI_MOSI / DIO4	Both	Disabled	GPIO/ SPI slave in
12	$\overline{\text{CTS}}$ / DIO7	Both	Output	Clear-to-Send Flow Control / GPIO
13	ON_$\overline{\text{SLEEP}}$ / DIO9	Both	Output	Module Status Indicator / GPIO
14	VREF	-	-	Not connected
15	ASSOCIATE / DIO5	Both	Output	Associate Indicator / GPIO
16	$\overline{\text{RTS}}$ / DIO6	Both	Input	Request to Send Flow Control / GPIO
17	AD3 / DIO3 / SPI_$\overline{\text{SSEL}}$	Both	Disabled	Analog Input / GPIO / SPI Slave Select
18	AD2 / DIO2 / SPI_CLK	Both	Disabled	Analog Input / GPIO / SPI Clock
19	AD1 / DIO1 / SPI_$\overline{\text{ATTN}}$	Both	Disabled	Analog Input / GPIO / SPI Attention
20	AD0 / DIO0 / CB	Both	Disabled	Analog Input / GPIO / Commissioning Button

EM357 pin mappings

The following table shows how the EM357 pins are used on the XBee.

Note Some lines may not go to the external XBee pins in the programmable secondary processor version.

EM357 Pin #	EM357 Pin Name	XBee (SMT) Pad #	XBee (TH) Pin #	Other Usage
12	RST	6	5	Programming
18	PA7	8	7	
19	PB3	29	16	Used for UART
20	PB4	25	12	Used for UART
21	PA0 / SC2MOSI	16	11	Used for SPI
22	PA1 / SC2MISO	17	4	Used for SPI
24	PA2 / SC2SCLK	14	18	Used for SPI
25	PA3 / SC2$\overline{\text{SSEL}}$	15	17	Used for SPI
26	PA4 / PTI_EN	32	19	OTA packet tracing
27	PA5 / PTI_DATA / $\overline{\text{BOOTMODE}}$	12	NA	OTA packet tracing, force embedded serial bootloader, and SPI attention line
29	PA6	7	6	
30	PB1 / SC1TXD	3	2	Used for UART
31	PB2 / SC1RXD	4	3	Used for UART
33	PC2 / JTDO / SWO	26	13	JTAG (see Writing Custom Firmware section)
34	PC3 / JTDI	28	15	JTAG (see Writing Custom Firmware section)
35	PC4 / JTMS / SWDIO	5	4	JTAG (see Writing Custom Firmware section)
36	PB0	10	9	
38	PC1 / ADC3	30	17	
41	PB7 / ADC2	31	18	
42	PB6 / ADC1	33	20	
43	PB5 / ADC0			Temperature sensor on PRO version

Appendix C: Performance Specifications of Digi XBee Zigbee PRO

Retrieved from http://ftp1.digi.com/support/documentation/90002002.pdf

- ATDO has HIGH_RAM_CONCENTRATOR and NO_ACK_IO_SAMPLING options added.
- 4040 - Binding and Multicasting transmissions are supported.
- AT&X command added to clear binding and group tables.
- Added Tx options 0x04 (indirect addressing) and 0x08 (multicast addressing).
- A 5 second break will reset the XBee. Then it will boot with default baud settings into command mode.
- BD range increased from 0-7 to 0-0x0A, and nonstandard baud rates are permitted, but not guaranteed.
- NI, DN, ND string parameters support upper and lower case.
- TxOption 0x01 disables retries and route repair. RxOption 0x01 indicates the transmitter disabled retries.
- 4050 - FR returns 0x00 modem status code instead of 0x01.
- S2C TH and S2C TH PRO supported.
- DC10 - verbose joining mode option.
- Self addressed fragmentable messages now return the self-addressed Tx Status code (0x23) instead of simply success (0x00).

With 4x5A, 7x5A:

- S2D SMT supported (HV=0x33).
- Configuration changes will delay the start of network formation/joining for 5 seconds.
- Verbose Join messages will be blocked while command mode is enabled.
- The UART remains enabled if SPI is blocked.
- DC80 enables a reset after 60 seconds of no joinable beacon responses received.

Note WR your configuration settings before enabling this option. Verbose Join will show "Reset for DC80" just before the reset takes place. If API mode is enabled, a modem status message with a status code of 0x1F will be sent just before the reset takes place.

- Under ZigBee, if all radios in a network have EE and EO set to 0x01, then the network key will be sent in the clear (unencrypted) at association time.

Specifications

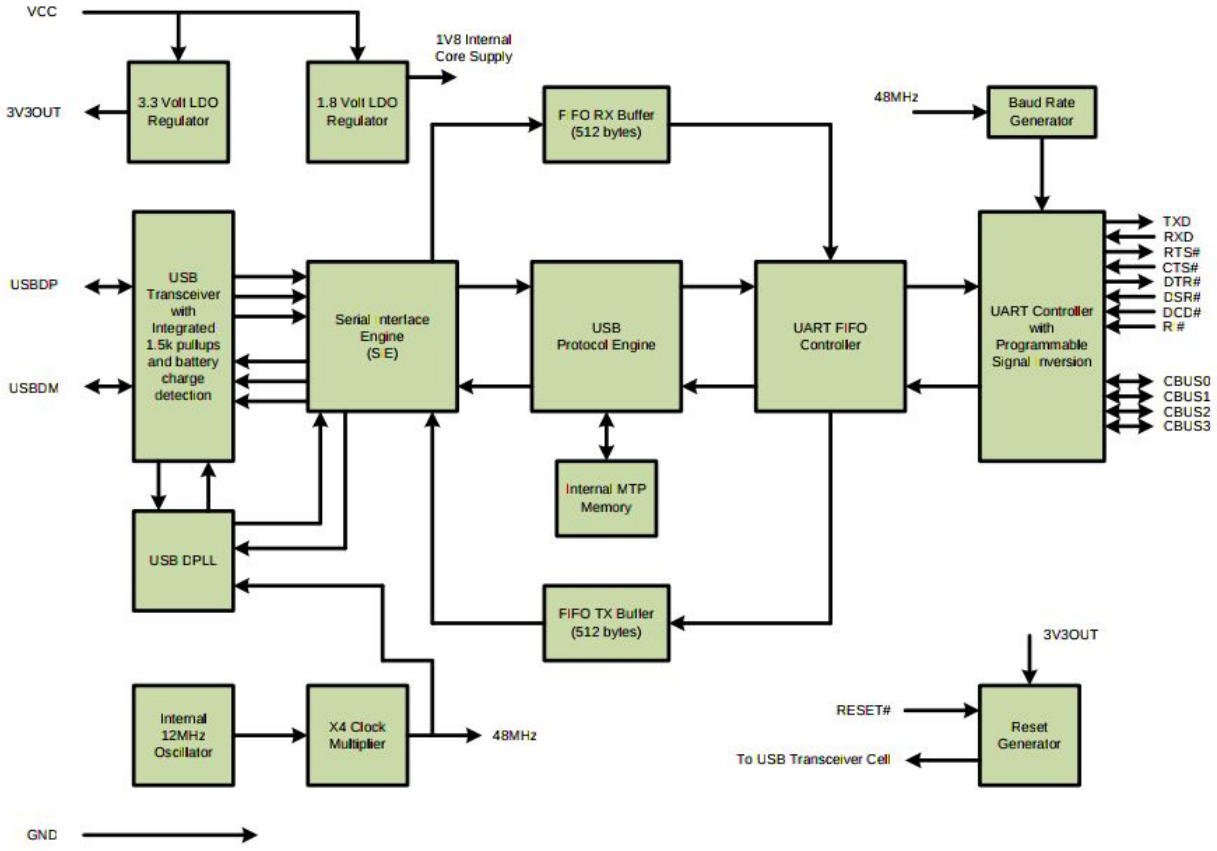
Hardware specifications

The following table provides the specifications for the module.

Specification	XBee ZigBee S2C	XBee-PRO ZigBee S2C
Performance		
Indoor/Urban Range	Up to 200 ft. (60 m)	Up to 300 ft. (90 m)
Outdoor RF line-of-sight Range	Up to 4000 ft. (1200 m)	Up to 2 miles (3200 m)
Transmit Power Output (maximum)	6.3mW (+8dBm), Boost mode 3.1mW (+5dBm), Normal mode Channel 26 max power is +3dBm	63mW (+18 dBm)
RF Data Rate	250,000 b/s	

Appendix D: FT 231X Block Diagram

Retreived from http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT231X.pdf



Appendix E: Digi XBee Pro Zigbee AT Command Reference

Retrieved from http://ftp1.digi.com/support/documentation/90002002.pdf

Command reference tables

Addressing commands

AT Command	Name and Description	Parameter Range	Default
DH	**Destination Address High**.Set/Get the upper 32 bits of the 64-bit destination address. When combined with DL, it defines the 64-bit destination address for data transmission. Special definitions for DH and DL include 0x000000000000FFFF (broadcast) and 0x0000000000000000 (coordinator).	0 - 0xFFFFFFFF	0
DL	**Destination Address Low**. Set/Get the lower 32 bits of the 64-bit destination address. When combined with DH, it defines the 64-bit destination address for data transmissions. Special definitions for DH and DL include 0x000000000000FFFF (broadcast) and 0x0000000000000000 (coordinator).	0 - 0xFFFFFFFF	0xFFFF(Coordinator) 0 (Router/End Device)
MY	**16-bit Network Address**. Read the 16-bit network address of the module. A value of 0xFFFE means the module has not joined a ZigBee network	0 - 0xFFFE [read-only]	0xFFFE
MP	**16-bit Parent Network Address**. Read the 16-bit network address of the module's parent. A value of 0xFFFE means the module does not have a parent.	0 - 0xFFFE [read-only]	0xFFFE
NC	**Number of Remaining Children**. Read the number of end device children that can join the device. If NC returns 0, then the device cannot allow any more end device children to join.	0 - MAX_CHILDREN (maximum varies)	read-only
SH	**Serial Number High**. Read the high 32 bits of the module's unique 64-bit address.	0 - 0xFFFFFFFF [read-only]	factory-set
SL	**Serial Number Low**. Read the low 32 bits of the module's unique 64-bit address.	0 - 0xFFFFFFFF [read-only]	factory-set

AT Command	Name and Description	Parameter Range	Default
NI	**Node Identifier.** Stores a string identifier. The register only accepts printable ASCII data. In AT Command Mode, a string can not start with a space. A carriage return ends the command. Command will automatically end when maximum bytes for the string have been entered. This string is returned as part of the ND (Node Discover) command. This identifier is also used with the DN (Destination Node) command. In AT command mode, an ASCII comma (0x2C) cannot be used in the NI string	20-Byte printable ASCII string	ASCII space character (0x20)
SE	**Source Endpoint**. Set/read the ZigBee application layer source endpoint value. This value will be used as the source endpoint for all data transmissions. SE is only used in transparent mode.The default value 0xE8 (Data endpoint) is the Digi data endpoint	0 - 0xFF	0xE8
DE	**Destination Endpoint**. Set/read ZigBee application layer destination ID value. This value will be used as the destination endpoint all data transmissions. DE is only used in transparent mode.The default value (0xE8) is the Digi data endpoint.	0 - 0xFF	0xE8
CI	**Cluster Identifier**. Set/read ZigBee application layer cluster ID value. This value will be used as the cluster ID for all data transmissions. CI is only used in transparent mode.The default value0x11 (Transparent data cluster ID).	0 - 0xFFFF	0x11
TO	**Transmit Options**. Set/read ZigBee application layer source transmit options value. This value will be used as the transmit options for all data transmissions in transparent mode.	0 - 0xFF Unused bits must be set to 0. These bits may be logically ORed together: 0x01 - Disable retries and route repair. 0x20 - Enable APS Encryption (if EE=1). Note that this decreases the maximum RF payload by 4 bytes below the value reported by NP. 0x40 - Use the extended timeout for this destination.	0x00

AT Command	Name and Description	Parameter Range	Default
NP	Maximum RF Payload Bytes. This value returns the maximum number of RF payload bytes that can be sent in a unicast transmission. If APS encryption is used (API transmit option bit enabled), the maximum payload size is reduced by 9 bytes. If source routing is used (AR < 0xFF), the maximum payload size is reduced further. Note: NP returns a hexadecimal value. (e.g. if NP returns 0x54, this is equivalent to 84 bytes)	0 - 0xFFFF	[read-only]
DD	**Device Type Identifier**. Stores a device type value. This value can be used to differentiate different XBee-based devices. Digi reserves the range 0 - 0xFFFFFF. For the XBee ZB SMT module, the device type is 0xA0000.	0 - 0xFFFFFFFF	0xA0000
CR	**Conflict Report**. The number of PAN id conflict reports that must be received by the network manager within one minute to trigger a PAN ID change. A corrupt beacon can cause a report of a false PAN id conflict. A higher value reduces the chance of a spurious PAN ID change. Starting with revision 4050, setting CR to 0 will instead set the threshold value to the default configuration value (3).	1-0x3F	3

Networking commands

AT Command	Name and Description	Parameter Range	Default
CH	**Operating Channel**. Read the channel number used for transmitting and receiving between RF modules. Uses 802.15.4 channel numbers. A value of 0 means the device has not joined a PAN and is not operating on any channel.	XBee 0, 0x0B - 0x1A XBee-PRO 0, 0x0B - 0x19 (Channels 11-25)	[read-only]
CE	**Coordinator Enable**. Set/read whether module is a coordinator.	0 - Not a coordinator 1 - Coordinator (SM must be 0 in order to set CE to 1.)	0
ID	**Extended PAN ID**. Set/read the 64-bit extended PAN ID. If set to 0, the coordinator will select a random extended PAN ID, and the router / end device will join any extended PAN ID. Changes to ID should be written to non-volatile memory using the WR command to preserve the ID setting if a power cycle occurs.	0 - 0xFFFFFFFFFFFFFFFF	0
OP	**Operating Extended PAN ID**. Read the 64-bit extended PAN ID. The OP value reflects the operating extended PAN ID that the module is running on. If ID > 0, OP will equal ID.	0x01 - 0xFFFFFFFFFFFFFFFF	[read-only]

AT Command	Name and Description	Parameter Range	Default
NH	**Maximum Unicast Hops**. Set / read the maximum hops limit. This limit sets the maximum broadcast hops value (BH) and determines the unicast timeout. The timeout is computed as (50 * NH) + 100 ms. The default unicast timeout of 1.6 seconds (NH=0x1E) is enough time for data and the acknowledgment to traverse about 8 hops.	0 - 0xFF	0x1E
BH	**Broadcast Hops.** Set/Read the maximum number of hops for each broadcast data transmission. Setting this to 0 will use the maximum number of hops.	0 - 0x1E	0
OI	**Operating 16-bit PAN ID**. Read the 16-bit PAN ID. The OI value reflects the actual 16-bit PAN ID the module is running on.	0 - 0xFFFF	[read-only]
ND	**Node Discovery**. Broadcast a ND command to the network. If an optional node identifier string parameter is given, then only those devices with a matching NI string should respond without a random offset delay. If no node identifier string parameter is given, then all devices should respond with a random offset delay. The NT setting determines the range of the random offset delay. The NO setting sets options for the Node Discovery. Warning: if the NT setting is small relative to the number of devices in the network, responses may be lost due to channel congestion. Regardless of the NT setting, because the random offset only mitigates against transmission collisions, getting responses from all devices in the network is not guaranteed.	20-byte printable ASCII string	ASCII space character (0x20)
NT	**Node Discovery Timeout**. Set/Read the node discovery timeout. When the network discovery (ND) command is issued, the NT value is included in the transmission to provide all remote devices with a response timeout. Remote devices wait a random time, less than NT, before sending their response.	0x20 - 0xFF [x 100 msec]	0x3C (60d)
NO	**Network Discovery options.** Set/Read the options value for the network discovery command. The options bitfield value can change the behavior of the ND (network discovery) command and/or change what optional values are returned in any received ND responses or API node identification frames. Options include: 0x01 = Append DD value (to ND responses or API node identification frames) 002 = Local device sends ND response frame when ND is issued.	0 - 0x03 [bitfield]	0

AT Command	Name and Description	Parameter Range	Default
SC	**Scan Channels**. Set/Read the list of channels to scan. ***Coordinator*** - Bit field list of channels to choose from prior to starting network. ***Router/End Device*** - Bit field list of channels that will be scanned to find a Coordinator/Router to join. Changes to SC should be written using WR command to preserve the SC setting if a power cycle occurs. Bit (Channel): 0 (0x0B) 4 (0x0F) 8 (0x13)12 (0x17) 1 (0x0C) 5 (0x10) 9 (0x14) 13 (0x18) 2 (0x0D) 6 (0x11) 10 (0x15)14 (0x19) 3 (0x0E) 7 (0x12)11 (0x16)15 (0x1A) Note: Note the following when setting SC to 0xFFFF. On the XBee modules, Channel 26 is not allowed to transmit at more than 3 dBm. If Channel 26 is present in the search mask (SC), then active search (beaconing) for network formation by a Coordinator will be limited to no more than 3 dBm on all channels. Other communication by a Coordinator/Router/EndDevice, or active search for network joining (association) by Routers and End Devices will be limited to no more than 3 dBm on Channel 26 - the transmit power on other channels will be controlled by PL and PM configuration settings. For the XBee-PRO SMT module, Channel 26 is not allowed to transmit at more than 6 dBm. For the XBee-PRO TH module, Channel 26 is not allowed to transmit at more than 2 dBm.	1 - 0xFFFF [bitfield]	7FFF

AT Command	Name and Description	Parameter Range	Default
SD	**Scan Duration**. Set/Read the scan duration exponent. Changes to SD should be written using WR command. Note: If channel 26 (0x8000) is enabled in the search channel mask (SC), transmit power on all channels will be capped at 3 dBm during network formation or joining. ***Coordinator*** - Duration of the Active and Energy Scans (on each channel) that are used to determine an acceptable channel and Pan ID for the Coordinator to startup on. ***Router / End Device*** - Duration of Active Scan (on each channel) used to locate an available Coordinator / Router to join during Association. Scan Time is measured as:(# Channels to Scan) * (2 ^ SD) * 15.36ms - The number of channels to scan is determined by the SC parameter. The XBee can scan up to 16 channels (SC = 0xFFFF). Sample Scan Duration times (13 channel scan): If SD = 0, time = 0.200 sec SD = 2, time = 0.799 sec SD = 4, time = 3.190 sec SD = 6, time = 12.780 sec Note: SD influences the time the MAC listens for beacons or runs an energy scan on a given channel. The SD time is not a good estimate of the router/end device joining time requirements. ZigBee joining adds additional overhead including beacon processing on each channel, sending a join request, etc. that extend the actual joining time.	0 - 7 [exponent]	3
ZS	**ZigBee Stack Profile**. Set / read the ZigBee stack profile value. This must be set the same on all devices that should join the same network.	0 - 2	0
NJ	**Node Join Time**. Set/Read the time that a Coordinator/Router allows nodes to join. This value can be changed at run time without requiring a Coordinator or Router to restart. The time starts once the Coordinator or Router has started. The timer is reset on power-cycle or when NJ changes. For an end device to enable rejoining, NJ should be set less than 0xFF on the device that will join. If NJ < 0xFF, the device assumes the network is not allowing joining and first tries to join a network using rejoining. If multiple rejoining attempts fail, or if NJ=0xFF, the device will attempt to join using association.	0 - 0xFF [x 1 sec]	0xFF (always allows joining)
JV	**Channel Verification**. Set/Read the channel verification parameter. If JV=1, a router or end device will verify the coordinator is on its operating channel when joining or coming up from a power cycle. If a coordinator is not detected, the router or end device will leave its current channel and attempt to join a new PAN. If JV=0, the router or end device will continue operating on its current channel even if a coordinator is not detected.	0 - Channel verification disabled 1 - Channel verification enabled	0

AT Command	Name and Description	Parameter Range	Default
NW	**Network Watchdog Timeout**. Set/read the network watchdog timeout value. If NW is set > 0, the router will monitor communication from the coordinator (or data collector) and leave the network if it cannot communicate with the coordinator for 3 NW periods. The timer is reset each time data is received from or sent to a coordinator, or if a many-to-one broadcast is received.	0 - 0x64FF [x 1 minute] (up to over 17 days)	0 (disabled)
JN	**Join Notification**. Set / read the join notification setting. If enabled, the module will transmit a broadcast node identification packet on power up and when joining. This action blinks the Associate LED rapidly on all devices that receive the transmission, and sends an API frame out the serial port of API devices. This feature should be disabled for large networks to prevent excessive broadcasts.	0 - 1	0
AR	**Aggregate Routing Notification**. Set/read the periodic time for broadcasting aggregate route messages. If used, these messages enable many-to-one routing to the broadcasting device. Set AR to 0x00 to send only one broadcast, to 0xFF to disable broadcasts, or to other values for periodic broadcasts in 10 second units.	0 - 0xFF (x10 sec)	0xFF (disabled)

Security commands

AT Command	Name and Description	Parameter Range	Default
EE	**Encryption Enable**. Set/Read the encryption enable setting.	0 - Encryption disabled 1 - Encryption enabled	0
EO	**Encryption Options.** Configure options for encryption when EE=1. Unused option bits should be set to 0. Options include: 0x01 - (ZigBee) Send the network key in the clear (unencrypted) over-the-air during a join 0x02 - (Smart Energy) Enable as a trust center (Coordinator only) 0x08 - (Smart Energy) Authenticate during joining (End Device and Router only)	0 - 0xFF	
NK	**Network Encryption Key**. Set the 128-bit AES network encryption key. This command is write-only; NK cannot be read. If set to 0 (default), the module will select a random network key.	128-bit value	0
KY	**Link Key**. Set the 128-bit AES link key. This command is write only; KY cannot be read. Setting KY to 0 will cause the coordinator to transmit the network key in the clear to joining devices, and will cause joining devices to acquire the network key in the clear when joining.	128-bit value	0

RF interfacing commands

AT Command	Name and Description	Parameter Range	Default
PL	**Power Level**. Select/Read the power level at which the RF module transmits conducted power. For XBee-PRO (S2B) Power Level 4 is calibrated and the other power levels are approximate. Calibration occurs every 15 seconds based on radio characteristics determined at manufacturing time, the ambient temperature, and how far off the voltage is from the typical 3.3 V. If the input voltage is too high, the module will reset. For the regular XBee, when operating on channel 26, no PL/PM selection will allow greater than +3 dBm output.	XBee (boost mode disabled) 0 = -5 dBm 1 = -1 dBm 2 = +1 dBm 3 = +3 dBm 4 = +5 dBm XBee-PRO (Boost mode enabled) 4 =+18 dBM 3 = +16 dBm (approx.) 2 = +14 dBm (approx.) 1 = +12 dBm (approx.) 0 = 0 dBm (approx.)	4
PM	**Power Mode** (XBee only). Set/read the power mode of the device. Enabling boost mode will improve the receive sensitivity by 2dB and increase the transmit power by 3dB Note: This command is disabled on the XBee-PRO. It is forced on by the software to provide the extra sensitivity. Boost mode imposes a slight increase in current draw. See section 1.2 for details.	0-1, 0= -Boost mode disabled, 1= Boost mode enabled.	1
DB	**Received Signal Strength**. This command reports the received signal strength of the last received RF data packet or APS acknowledgment. The DB command only indicates the signal strength of the last hop. It does not provide an accurate quality measurement for a multihop link. DB can be set to 0 to clear it. The DB command value is measured in -dBm. For example if DB returns 0x50, then the RSSI of the last packet received was -80dBm.	0 - 0xFF Observed range for XBee-PRO: 0x1A - 0x58 For XBee: 0x 1A - 0x5C	
PP	**Peak Power.** Read the dBm output when maximum power is selected (PL4).	0x0-0x12	[read only]

Serial interfacing (I/O) commands

AT Command	Name and Description	Parameter Range	Default
AP	**API Enable**. Enable API Mode. This command is ignored when using SPI. API mode 1 is always used.	0 = API-disabled (operate in transparent mode) 1 = API-enabled 2 = API-enabled (w/escaped control characters)	1
AO	**API Options**. Configure options for API. Current options select the type of receive API frame to send out the UART for received RF data packets.	0 - Default receive API indicators enabled 1 - Explicit Rx data indicator API frame enabled (0x91) 3 - enable ZDO passthrough of ZDO requests to the serial port which are not supported by the stack, as well as Simple_Desc_req, Active_EP_req, and Match_Desc_req.	0
BD	**Interface Data Rate.** Set/Read the serial interface data rate for communication between the module serial port and host. Any value above 0x0A will be interpreted as an actual baud rate. The modules support standard baud rates from 1200 to 115200 baud. Non-standard baud rates are permitted but their performance is not guaranteed.	0 - 0x0A 0 = 1200 b/s 1 = 2400 2 = 4800 3 = 9600 4 = 19200 5 = 38400 6 = 57600 7 = 115200 8 = 230400 9 = 460800 A = 921600	3
NB	**Serial Parity**. Set/Read the serial parity setting on the UART.	0 = No parity 1 = Even parity 2 = Odd parity 3 = Mark parity	0
SB	**Stop Bits**. Set/read the number of stop bits for the UART. (Two stop bits are not supported if mark parity is enabled.)	0 = 1 stop bit 1 = 2 stop bits	0
RO	**Packetization Timeout**. Set/Read number of character times of inter-character silence required before packetization. Set (RO=0) to transmit characters as they arrive instead of buffering them into one RF packet The RO command is only supported when operating in transparent mode.	0 - 0xFF [x character times]	3

AT Command	Name and Description	Parameter Range	Default
D7	**DIO7 Configuration**. Select/Read options for the DIO7 line of the RF module.	0 = Unmonitored digital input 1 = CTS Flow Control 3 = Digital input 4 = Digital output, low 5 = Digital output, high 6 = RS-485 transmit enable (low enable) 7 = RS-485 transmit enable (high enable)	1
D6	**DIO6 Configuration**. Configure options for the DIO6 line of the RF module.	0 = Unmonitored digital input 1 = RTS flow control 3 = Digital input 4 = Digital output, low 5 = Digital output, high	0

I/O commands

AT Command	Name and Description	Parameter Range	Default
IR	**I/O Sample Rate**. Set/Read the I/O sample rate to enable periodic sampling. For periodic sampling to be enabled, IR must be set to a non-zero value, and at least one module pin must have analog or digital I/O functionality enabled (see D0-D9, P0-P4 commands). The sample rate is measured in milliseconds.	0, 0x32:0xFFFF (ms)	0
IC	**I/O Digital Change Detection**. Set/Read the digital I/O pins to monitor for changes in the I/O state. IC works with the individual pin configuration commands (D0-D9, P0-P4). If a pin is enabled as a digital input/output, the IC command can be used to force an immediate I/O sample transmission when the DIO state changes. IC is a bitmask that can be used to enable or disable edge detection on individual channels. Unused bits should be set to 0. Bit (IO pin): 0 (DIO0) 4 (DIO4) 8 (DIO8) 1 (DIO1) 5 (DIO5) 9 (DIO9) 2 (DIO2) 6 (DIO6) 10 (DIO10) 3 (DIO3) 7 (DIO7) 11 (DIO11)	: 0 - 0xFFFF	0
P0	**PWM0 Configuration**. Select/Read function for PWM0.	0 = Unmonitored digital input 1 = RSSI PWM 3 - Digital input, monitored 4 - Digital output, default low 5 - Digital output, default high	1

F.) Analysis of Senior Project Design

Analysis of Senior Project Design

Please provide the following information regarding your Senior Project and submit to your advisor along with your final report. Attach additional sheets for your responses to the questions below.

Project Title: Wireless Mesh Networks for SCADA Systems

Quarter / Year Submitted: Sp/2016

Student: (Print Name) Zachary Weisman **(Sign)** [signature]

Advisor: (Print Name) Dr. Hugh Smith **(Initial)** [initial] **Date:** 6/7/16

- **Summary of Functional Requirements**

Describe the overall capabilities of functions of your project or design. Describe what your project does. (Do *not* describe how you designed it.)

- **Primary Constraints**

Describe significant challenges or difficulties associated with your project or implementation. For example, what were limiting factors or other issues that impacted your approach? What made your project difficult? What parameters or specifications limited your options or directed your approach?

- **Economic**
  - Original estimated cost of component parts (as of the start of your project)
  - Actual final cost of component parts (at the end of your project)
  - *Attach a final bill of materials for all components*
  - Additional equipment costs (any equipment needed for development?)
  - Original estimated development time (as of the start of your project)
  - Actual development time (at the end of your project)
- **If manufactured on a commercial basis:**
  - Estimated number of devices to be sold per year
  - Estimated manufacturing cost for each device
  - Estimated purchase price for each device
  - Estimated profit per year
  - Estimated cost for user to operate device, per unit time (specify time interval)
- **Environmental**

Describe any environmental impact associated with manufacturing or use.

- **Manufacturability**

Describe any issues or challenges associated with manufacturing.

- **Sustainability**
  - Describe any issues or challenges associated with maintaining the completed device or system.
  - Describe how the project impacts the sustainable use of resources.
  - Describe any upgrades that would improve the design of the project.
  - Describe any issues or challenges associated with upgrading the design.
- **Ethical**

Describe ethical implications relating to the design, manufacture, use or misuse of the project.

- **Health and Safety**

Describe any health and safety concerns associated with design, manufacture or use.

- **Social and Political**

Describe any social and political concerns associated with design, manufacture or use.

- **Development**

Describe any new tools or techniques used for either development or analysis that you learned independently during the course of your project.

1.) Summary of Functional Requirements

The requirement of this project is simple, although implementation was more complicated. This project is meant to be a backbone network on which a single computer on a solar farm can request status information about the power production of solar panels distributed throughout the farm, wirelessly.

2.) Primary Constraints

Constraints for this project mainly revolved around two situations, one of which being the event of power loss. This possibility led me down the path of writing start up scripts to handle node provisioning without need for a human to intervene. The next situation that needed handling was the possibility of a malicious entity within range of the radios. Features needed to be implemented to prevent playback, and MitM attacks.

3.) Economic Analysis

At the beginning of this project there were only two components estimated:
i.) Raspberry Pi Model 2 - $39.95
ii.) Digi XBee PRO 900 Module - $28.00

By the end of the project, implementation had changed, this hardware needs also changed. I used different radio's along with an extra interface component, which brought the total number of materials per node to three:
i.) Raspberry Pi Model 2 - $39.95
ii.) Digi XBee Zigbee Pro Module - $28.50
iii.) SparkFun XBee Explorer Dongle - $24.95

The total bill of materials for development and testing is as follows:

Description	**Price**	**Quantity**	**Cost**
Raspberry Pi Model 2	$39.95	5	$199.75
Digi XBee Pro 900 Module	$28.00	5	$140.00
Digi XBee Zigbee Pro	$28.50	10	$285.00
Sparkfun Explorer Dongle	$24.95	10	$249.50
Total			**$874.25**

4.) Manufacturing Analysis

At the time of writing this document, the final product that this project is to be integrated into is still under development. There is no clear way of projecting either the number of devices sold per year, profit gained from this component, or purchase price. Those values are all still to be set by market demand. The only clear number we can ascertain is the operating cost. The goal for this project was to create an autonomous self healing network, and that's what was achieved, so ideally there would be no operating cost.

5.) Environmental Impact

The only environmental impact I can see with this project is the manufacturing of the components themselves. Primarily, during the PCB process, many of the etching chemicals are often dumped into the environment surrounding the manufacturing site, this harming local ecosystem. Another environmental impact can be attributed to shipping. Many of these components are manufactured in China, and shipped to the United States on large container ships, which have been shown to release massive amounts of the most harmful or emissions of most fossil-fuel based transportation methods.

6.) Manufacturability

Manufacturing in the case of my project is almost a moot point. All components are purchased pre-assembled, and the final assembly of the fixture (plugging in two components) can be handled by construction workers during site assembly.

7.) Sustainability

The end product is designed to be plug-and-play, so there is no issue with sustainability in terms of maintenance. Unfortunately, software update practices have not been established, so in the event that software does need to be changed on each node, it would be a very daunting, and certainly disruptive undertaking to the operation. Upgrading the overall design of the communication would theoretically be fairly straightforward, because it's integration with the solar tracker is fairly modular.

8.) Ethical Problems

There are only a handful of ethical problems related to this project. Manufacturing ethics, honesty in terms of data integrity, and honesty on the part of SunLink to ensure prompt notification of any security breaches. Obviously the current state of manufacturing ethics is a hot topic. The decision to source components from manufacturers with certain policies regarding their employees and the environment is in the hands of the customer of this product. Data integrity - whether or not SunLink is supplying real data to their customers about their operation's power production is the responsibility of executives at SunLink. Finally, should there be a breach in security, obviously that would affect the reputation of SunLink in the industry, so their decision to notify customers of such an event is also in question.

9.) Health and Safety

The only health or safety risks associated with this project are those aforementioned risks inherent in the manufacturing of components.

10.) Social And Political

Socially and politically, there are many narratives being discussed right now about the ethics of data privacy, one that is not appropriate for discussion in this document, but is worth noting. Politically, another conversation is the integration of more solar energy technologies into the mainstream power grid. More specifically in California there is the debate about whether or not to continue providing subsidies to solar companies in order to reduce entry costs into the market for new customers.

11.) Development

Development of this project relied heavily on techniques learned in various embedded development courses, as well as integrated circuit design and analysis courses through my college career. Software testing was aided by the use of free, proprietary software called X-CTU provided by Digi. Overall, I didn't pick up any new techniques for development or testing; my Cal Poly education adequately prepared me to handle this project.