Content Delivery in Ad Hoc Wireless Networks

Design Document

Dec 10-03

Client:

Iowa State University

Department of Electrical and Computer Engineering

Advisors:

Prof. Lei Ying

Ming Ouyang

Team Members:

Wyatt Brenneman

Taylor McKechnie

Prashanth Yanamandra

December 6, 2010

Table of Contents

List of Figures	4
List of Tables	5
List of Definitions	6
Executive Summary	7
Acknowledgement	8
Problem Statement	8
Operating Environment	9
Intended Users and Intended Uses	9
Assumptions	10
Limitations	10
Expected End Product and Deliverables	10
Functional Requirements	11
Non-Functional Requirements	11
Technology Considerations	12
Safety Requirements Considerations	13
Possible Risks and Risk Management	14
System Decomposition	14
Detailed Design	14
Details of Hardware	16
Software Design	17
Testing and Evaluation	18
Testing Phases	18
Evaluation	19
Test Results	19

Estimated Resources	21
Project Schedule	22
Task Breakdown	22
Team Information	24
Closing Summary	25
References	26
Appendix	27

List of Figures

Concept Sketch	8
Operating Environment	9
USRP Box	12
TelosB Sensor	12
System Decomposition	13
USRP Circuitry	15
USRP Block Diagram	16
Data Packet Sent From USRP	19
Data Received by USRP	19
Sensor Data Received by USRP	20
Sensor Data Relayed by USRP to Base Station Sensor	20
Light Sensor Graphical Data	21
Project Schedule	22

List of Tables

Intended Users and Intended Uses 9

Semester 1 Task Breakdown 22

Semester 2 Project Task Breakdown 23

List of Definitions

Wireless Ad-hoc Network: A decentralized wireless network which does not depend on preexisting infrastructure like routers but each node participates in data transmission.[1]

USRP/USRP2 (Universal Software Radio Peripheral): USRP/USRP2 is a general purpose RF hardware device designed by Ettus Research which provides a low-cost, readily but versatile radio functionality.

GNU Radio: GNU Radio is an open-source software development toolkit for the development of software-defined radios. It contains a variety of signal processing algorithms that we will use on the data collected by the USRP hardware.

P2P (Peer to Peer): A system where the users both supply and consume the data available on the network.

SD Card (Secure Data Card): A non-volatile memory card which would be used to store programming for the USRP2 in its standalone operation mode.

BER (Bit Error Rate): The number of erroneously received bits divided by the total number of transmitted bits. BER is unit-less and expressed as a percentage.

TinyOS: TinyOS is a free and open source component-based operating system that is written using the nesC programming language. [2]

Executive Summary

As the understanding of the world around evolves and grows, the control and distribution of information is key to the continual growth and progress of society. Previous exchanges of information required physically receiving the information from the source, receiving it from a third party, or from large network sources such as the internet. These previous methods were not conducive to real time sharing of information between individuals. Our project will allow users to stream data onto the network for others to view immediately and straight from the sources of the information.

A team of three Iowa State University College of Engineering students will work to create an ad Hoc wireless network which will allow the ability of users to stream information using P2P traffic. Our group will implement this network with several USRP and USRP2 radios, wireless sensors, and the GNU Radio protocol.

Acknowledgement:

Our group would like to thank Iowa State University for providing the equipment needed for the project including the USRPs, USRP accessories, and laptops used to complete this project. We would also like to thank our faculty advisor Professor Lei Ying and his PhD student Ming Ouyang for answering our questions about the project and helping troubleshoot the errors that we encountered.

Problem Statement

Wireless connectivity has become popular in our everyday life and the ability to exchange information is increasing just as fast. Examples of technologies implementing ad-hoc networks include Wi-Fi, Bluetooth and Zigbee.

Our goal is to establish communication between USRPs, USRP to Sensor and Sensor to Sensor using Zigbee protocol (IEEE 802.15.4). This protocol operates at 2.4GHz broadcasting frequency. We will be implementing a star network topology for our sensor networks. We also aim to print useful data obtained from the sensors

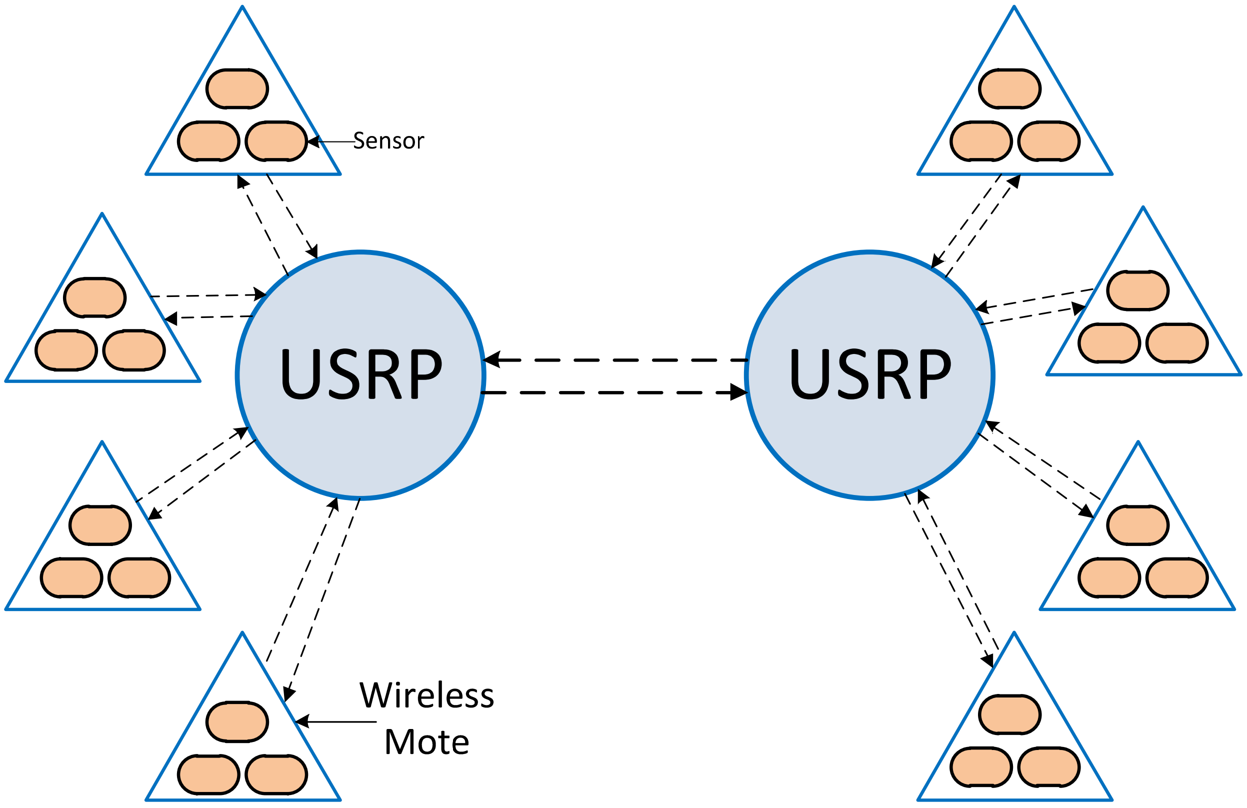


Figure: Concept Diagram

Operating Environment:

Operating Environment:

The eventual network envisioned for this project will be providing P2P streaming service for an individual building with the users accessing the network also inside the building. Because our system will be operating indoors to provide network coverage for the entire building we won't have to worry about external weather conditions affecting the equipment or the transmission of the data. Since our network will be operating indoors, the network will need to overcome the many stationary physical path obstacles as well as smaller moving path obstacles. Future projects may be implemented to provide an outdoor network and will need to provide physical protection from temperature, rain, dust, weather conditions, wildlife, and vandalism. Moving the network outside will also require compensating the data signal for adverse weather conditions to improve reliability.

Intended User	Intended Use
Non-technical unknowledgeable peripheral device user	Wirelessly broadcasting information to other users around the area who can see it in real time. Should not require any knowledge of how background network works, user only has to start transmitting by hitting a "broadcast" button and it should broadcast the data to all users from there until the user ends filming.
Administrator	Manage feeds on the network, limiting and/or cutting the connectivity of ones that become too much of a hindrance to the network.
Commercial user	Broadcast of meeting or demonstration from on-site area to an office building or safe-zone.
Educational user	Use of our project as an educational tool for wireless sensor communication networks.

Assumptions

- *Minimum user access:* The system works only when there is at least one user who is broadcasting and at least one user streaming the data.
- *Users within the range of sensors:* Users will be able to get best reception when they are in the range of about 100 feet from the sensory/ USRP network.
- *USRP and Laptop in same location:* The USRP does computations when it is connected to a laptop and will not function as a stand-alone system. The USRP2 does allow the use of an SD card instead of an external laptop for computation. However, using the SD card reduces the number and complexity of computations available to the USRP2.
- *Indoor Network:* The USRP and sensors will be placed indoors so physical protection from weather elements will not be needed. We also will have to compensate for signal degradation due to walls and reflections.

Limitations

- The equipment is not built to operate in extreme temperatures and should be kept indoors.

Expected End Product and Other Deliverables

The end product delivered will be

- *Communication between USRPs and sensors*
- The kinds of communication that will be available are USRP to USRP, Sensor to Sensor, USPR to Sensor, Sensor to USRP, and Sensor to USRP to Sensor.

Other Deliverables include

- *Project Plan Document*

The project plan document highlighting the proposed approach plan will be provided at the completion of the project

- *Design Document*

The design document highlighting the design approach considered for the project will be provided at the completion of the project.

- *Design Review Presentation*

The presentation made for the design review will be provided at the completion of the project

- *Project Poster*

The poster designed highlighting our achievements for the project will be provided at the completion of the project

- *User Guide*

A user manual will be provided to the user upon completion of the project in order to act as a reference on how each element of the system works. This document will include a software guide as well as instruction sets that assist with training and reference material.

Functional Requirements

- *Streaming of Wireless Sensor Data throughout Network*
    - Our system will allow wireless sensor motes to stream sensor data to the USRP backbone of our system. The data will be propagated to other wireless motes through the USRP backbone.
- *Simultaneous Streaming*
    - The fully functional system must be capable of supporting 10 simultaneous streams at any instant of time. The wireless motes require a unique ID # when programmed so sensor data can be tracked to the specific sensors.
- *Utilize the Zigbee 802.15.4 Protocol*
    - Our network will be operating in the 2.4 – 2.4835 GHz band, which is the worldwide band for Zigbee. We will be utilizing channel 16 of this band, which is at 2.48 GHz center frequency.
- *Broadcasting range*
    - The system must be able to provide good reception for users in a close range

Non- Functional Requirements

- *Physical Dimensions*

- The dimensions of the sensors need will need to be small enough to be implemented in classrooms and labs without needing to change any layout of the room. The dimension of the USRP is fixed.

- *Equipment Protection*
    - The equipment is placed indoors at all times and does not require weather protection.
- *Power Requirements*
    - The USRP and the USRP2 need an AC to DC converter. The sensors will need 2 AA batteries to supply power.

Technology Considerations

- *Hardware Considerations*
    - *USRP*: We choose to use the USRP because it has been specifically designed to use the GNU Radio software. By utilizing a software defined radio, much more flexibility is given to our system by taking the much of the system design away from hardware and given to software design.

        - *Daughterboard:* The USRP daughterboard that we have chosen to use is the RFX2400 Transceiver. This will allow us to transmit and receive with only one antenna and also operates within the worldwide Zigbee 802.15.4 band.
    - *Sensors:* The sensors used in our system are Crossbow TelosB sensors. We choose these because they include voltage, temperature/humidity, visible light spectrum, and visible light to infrared sensors. These sensors can also be easily programmed using TinyOS and are powered by only two AA batteries.

- *Software Considerations*
    - *GNU Radio:* GNU Radio is an open-source software development toolkit which enables a variety of signal processing on the data collected by the USRP hardware. GNU radio is a

software interface to the USRP. There are also many projects using GNU Radio and USRPs and we can utilize these projects as a resource for learning.

  - *TinyOS:* We will be using TinyOS for programming the wireless motes because it utilizes nesC which is very similar to C syntax and allows component blocks to be tied together easily.

Safety Considerations

- *Excessive exposure to radio frequency*
  - Being exposed to excessive radio frequency can lead to health hazards
- *No interference*
  - Equipment used for this project will not interfere with other radio devices which include wireless 2 way radios.

All the technology used here is purely for educational purposes and not for jamming other equipment or for destructive purposes.

Risks and Mitigation Plans

The project is designed with precision and quality; however, there are certain risks, which could be encountered during the course of project lifetime. The possible risks are indicated below:

- *Device/Sensor failure*
  - Sensors are prone to fail with increasing time and usage. To reduce the effect of sensor failing on the system, our team is planning to overlap sensors and make a mesh network. Taking this approach, we will be able to use other sensors to route the data temporarily, until the sensor is repaired/ replaced. The sensors we plan to use are commonly available and will be easy to order and replace.
- *Data Transmission Overlap*
  - Data being sent across a wireless medium is will end up travelling in multiple paths to the receiver. This can cause deconstructive interference from phase shifting and congestion in the system. We will implement a way to recognize the desired signal and reduce the chance of propagating duplicate signals.

System Decomposition

The USRP sends and receives information from the video sensor wirelessly at 2.4GHz. The data packets received from the video sensor are processed by the FPGA in the USRP. The processed data is then sent to the laptop computer through a USB cable or a gigabit cord.

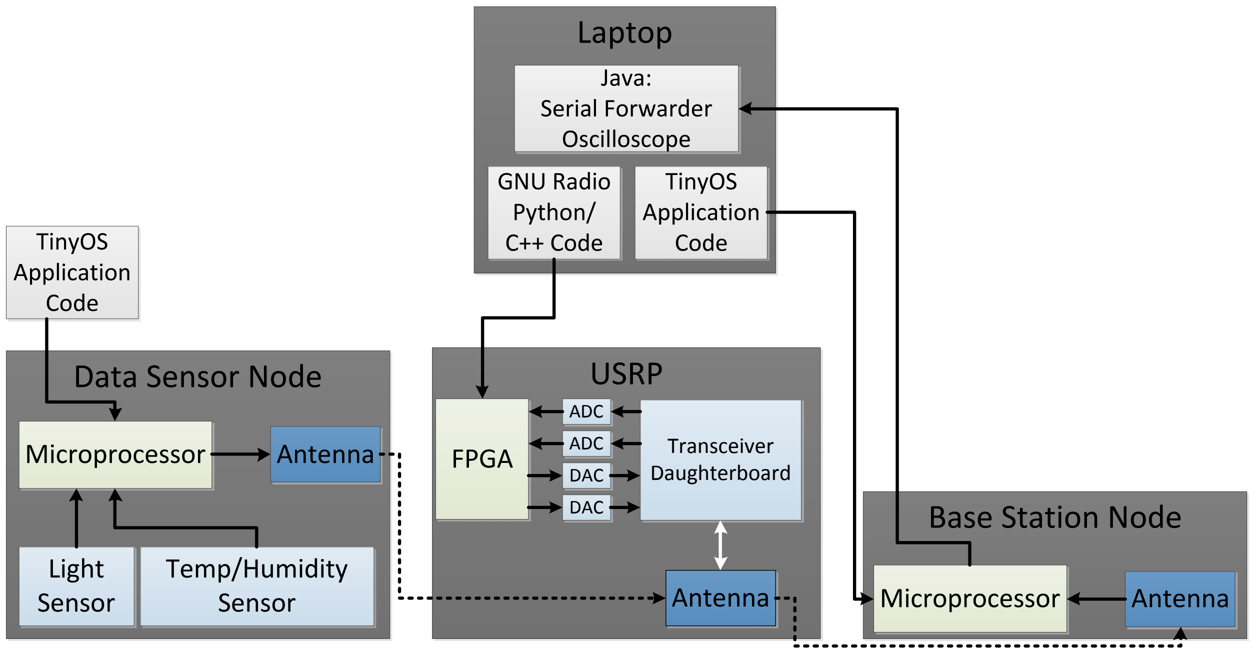


Figure: System Decomposition

Detailed Design Overview

USRP

For our project we will be using the USRP as well as the newer USRP2 to provide the hardware interface of our radio system. The main reason for using the USRP/USRP2 is that they were designed with the GNU Radio software in mind and created to maximize the capabilities of GNU Radio. The USRP/USRP2 design is also open source, which means that all of the tools we will be using to create our final product are open source. This fact allows us forgo typical software and licensing fee that could be attributed to a typical project of our nature.

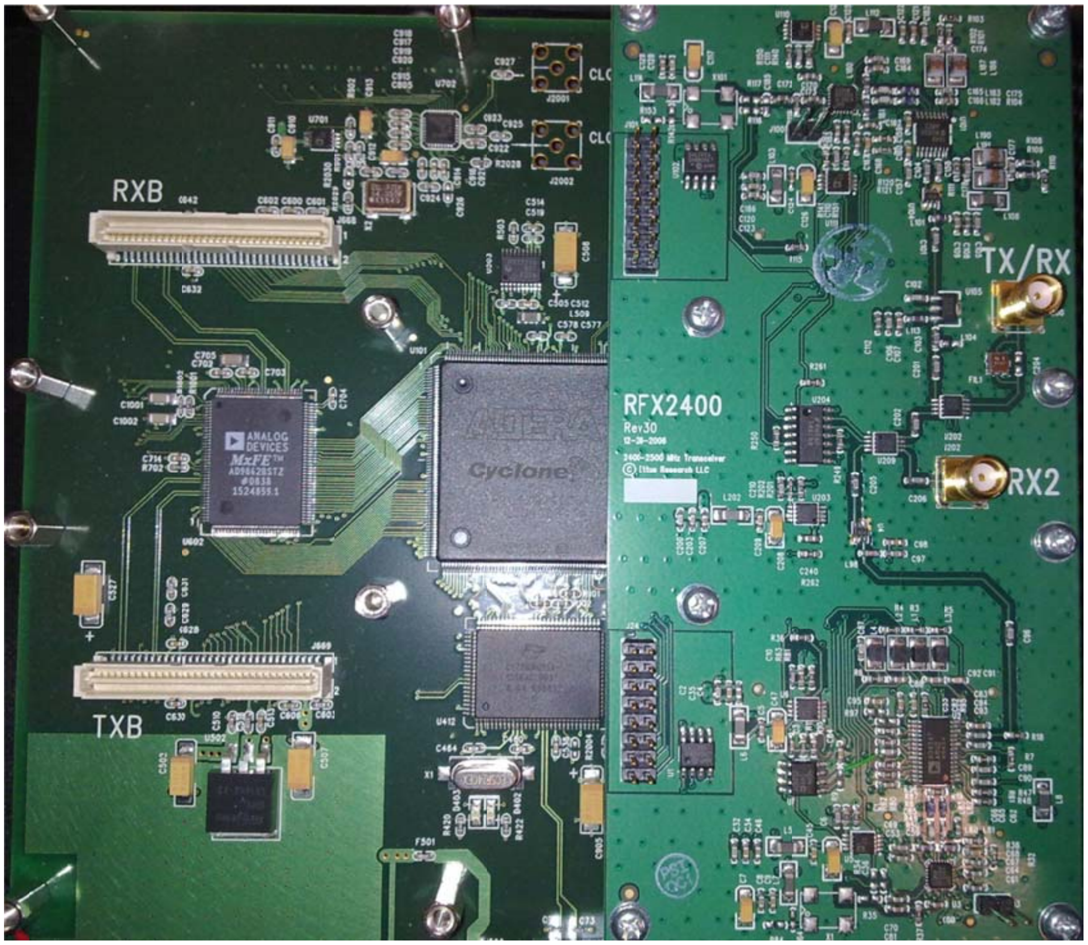


Figure: Trasnsceiver Daughter board used in USRP

Details of Hardware

For both our USRP and USRP2 modules we will be using the same antennas and daughterboards.

Antenna: We will be using the VERT2450 antenna on all of our USRPs.

- Dual band 2.400 – 2.800 GHz and 4.900 – 5.900 GHz.

Daughterboard: We will be using the RFX2400 Transceiver daughterboard in all of our USRPs.

- Fully controllable through radio software or FPGA
- Transmitting power of 50mW (17 dBm)
- Able to transmit and receive through one antenna and board connection
- 30 MHz transmit and receive bandwidth
- Frequency range of 2.300 – 2.900 GHz

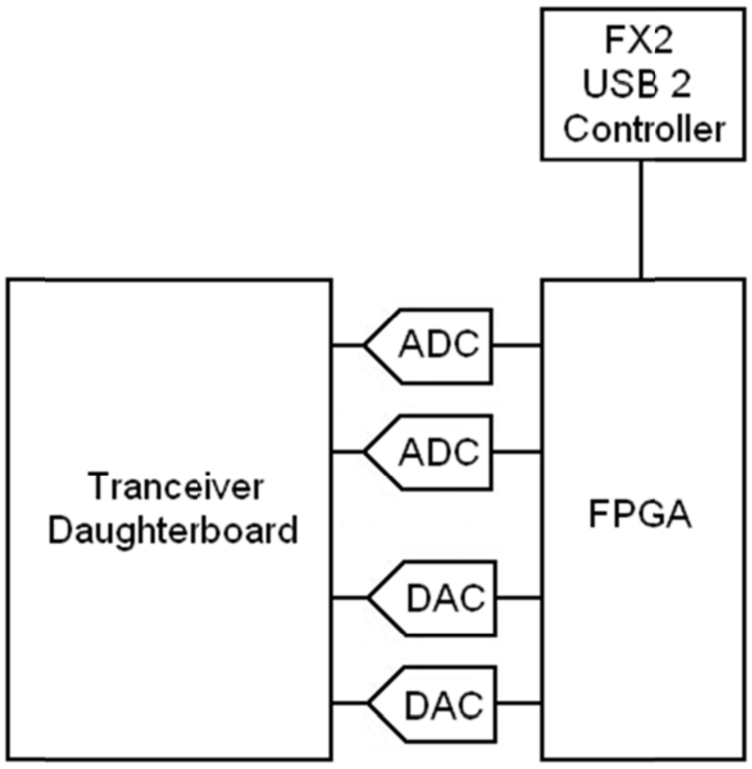
The first generation USRP utilizes 4 high-speed ADCs and DACs which are connected to an Altera Cyclone EP1C12 FPGA. The USRP also uses USB2 to interface with laptop.

Analog to Digital Converters

- 64 Ms/s at 12-bit

Digital to Analog Converters

- 128 Ms/s at 14-bits



The second generation USRP2 also utilizes 4 high-speed ADCs and DACs, which are then connected to a Xilinx Spartan 3-2000 FPGA which then connects to a Gigabit Ethernet interface.

Analog to Digital Converters

- 100 Ms/s at 14-bit

Digital to Analog Converters

- 400 Ms/s at 16-bits

Software Design

- *USRP Software*
  - The software packages that were used for the USRP software are GNU Radio and UCLA Zigbee PHY. The UCLA Zigbee PHY project was created to extend GNU Radio to interoperate with Chipcon CC1000 and CC2420 radios. We choose to use this because it is an open source project and the TelosB mote uses a CC2420 radio for its wireless transmission. This allowed us to focus on writing an application for our sensor network without having to write all the physical layer coding needed for Zigbee.

    Our python script utilizes two of the example codes provided by the UCLA Zigbee PHY project. The project provides a basic transmission script and a basic receive script for sending and receiving packets formatted for the CC2420 chip. Since we want our program to be able to transmit and receive messages simultaneously we needed to combine the two scripts effectively.

    The original receive script would send the same message every time, but since we are forwarding the messages that are received we needed to parse the incoming data into its Frame Control Field (FCF), packet number, address info, payload, and Cyclical Redundancy Check (CRC). We then took the packet number, address info, and payload and passed it to the send_pkt() function. This function is part of the UCLA Zigbee PHY library, which generates an FCF and CRC for packet information and then sends the data to the USRP. One of the largest problems with combining the two separate scripts was that both scripts were written assuming no other program would be accessing the daughterboard. This meant that initially that the USRP could only receive or only transmit.

    The solution for this problem was to create a new top level class where both the receive path and transmit path are activated and then combined. Since this became the top level class, this forced the receive and transmit classes to become second level blocks and changing some of the class initialization functions.

    After connecting both transmit and receive paths together we were then able to transmit and receive at the same time, but we needed to add a check that the packet received was not just transmitted. This prevents the USRP from creating an infinite loop where it will continually send the same message since it is picking up the message it has just transmitted.

- *TinyOS Software*
  - To test the functionality of our system of receiving and sending packets from the TelosB motes we used a basic example application with a few modifications. We choose to use the Oscilloscope and BaseStation applications provided by TinyOS.

The Oscilloscope application takes sampled sensor data and transmits it wirelessly using the CC2420 chip. The data packet contains 10 readings 16-bit sensor data as well as mote id. The default sensor for the Oscilloscope program was a voltage sensor that measured the supply voltage of the microprocessor. While this data can be useful to determine if the batteries may need to be replaced in a mote, the data is nearly constant and is not useful for a demonstration. To make this application interactive for our demonstration we choose the sensor that it would be reading to the Hamamatsu s1087 visible light sensor. This allows us to see drastic changes in the sensor readings by affecting the amount of light that reaches the sensor to verify the real-time transmission of the data.

Testing and Evaluation

Testing will be performed in various stages during the course of the project. The initial stage of testing includes testing the system by sending data packets from one USRP to another USRP. We will then move to testing communications between the USRPs and the sensors.

Testing Phases

As was stated earlier the first stage is testing communication between the USRP devices. This was done by sending various packets from one USRP to another and checking them to ensure they were the same packet. We had some user error here when putting in the frequency to transmit and receive on.

The second stage was testing the sensor to sensor communication. In this phase of testing, a sensor transmitted either light, voltage or temperature readings as data packets to the base station (a sensor programmed to be a receiver connected to the laptop). The base station printed out the information received from the transmitter sensor using the Java Oscilloscope application.

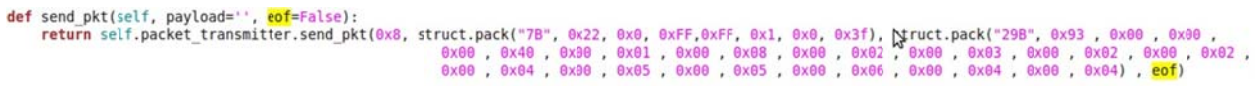
The third stage was testing the sensor to USRP communication. Data packets were sent back and forth between the sensor and the USRP. Data sent from the sensor to the USRP was printed to the screen and then using another java application included with TinyOS called Listen we printed the data received by the base station sensor. In this stage we had to ensure the received payload was parsed correctly and preserved their formatting to prevent any packets failing which the packet would be dropped.

The final stage of testing was sending a packet from a senor to a USPR and then to another sensor. In this stage we had to ensure that the packet was successfully received and transmitted. To do this we had to ensure the correct parts of the packet were stripped away when it was received so that the payload (packet data) was left unchanged. When moving from receiving to transmission we ran in to an error of the USRP receiving the packet it was transmitting in a continuous loop.

Evaluation

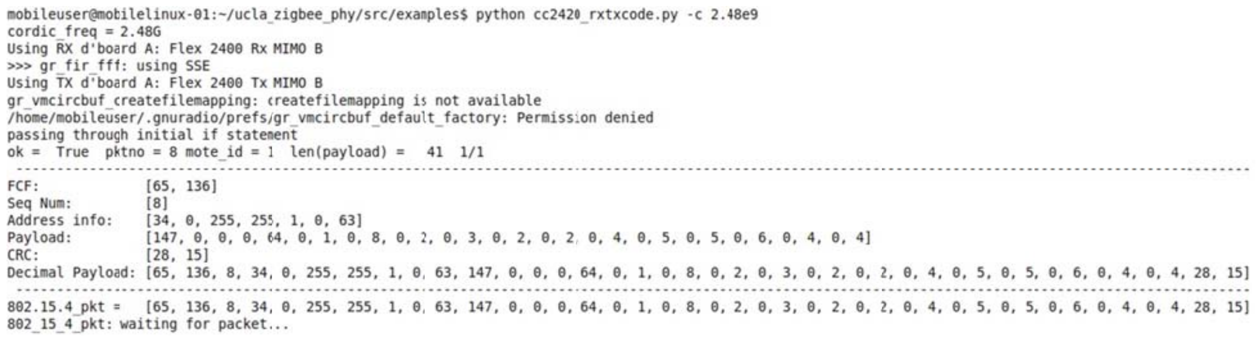
Data packets will be transmitted from a TelosB sensor to an USRP. The USRP will print out the received sensor data. It will simultaneously transmit the received data to another USRP. The second USRP will print the received data. We will then compare the received data with the transmitted data to check for error free transmission. The USRP will then transmit the data to the sensors, which will confirm the data received by a light blink. This multi-hop transmission will check full system functionality correctness.

Test Results



```
def send_pkt(self, payload='', eof=False):
    return self.packet_transmitter.send_pkt(0x8, struct.pack("7B", 0x22, 0x0, 0xFF,0xFF, 0x1, 0x0, 0x3f), struct.pack("29B", 0x93 , 0x00 , 0x00 ,
                                             0x00 , 0x40 , 0x00 , 0x01 , 0x00 , 0x08 , 0x00 , 0x02 , 0x00 , 0x03 , 0x00 , 0x02 , 0x00 , 0x02 ,
                                             0x00 , 0x04 , 0x00 , 0x05 , 0x00 , 0x05 , 0x00 , 0x06 , 0x00 , 0x04 , 0x00 , 0x04) , eof)
```

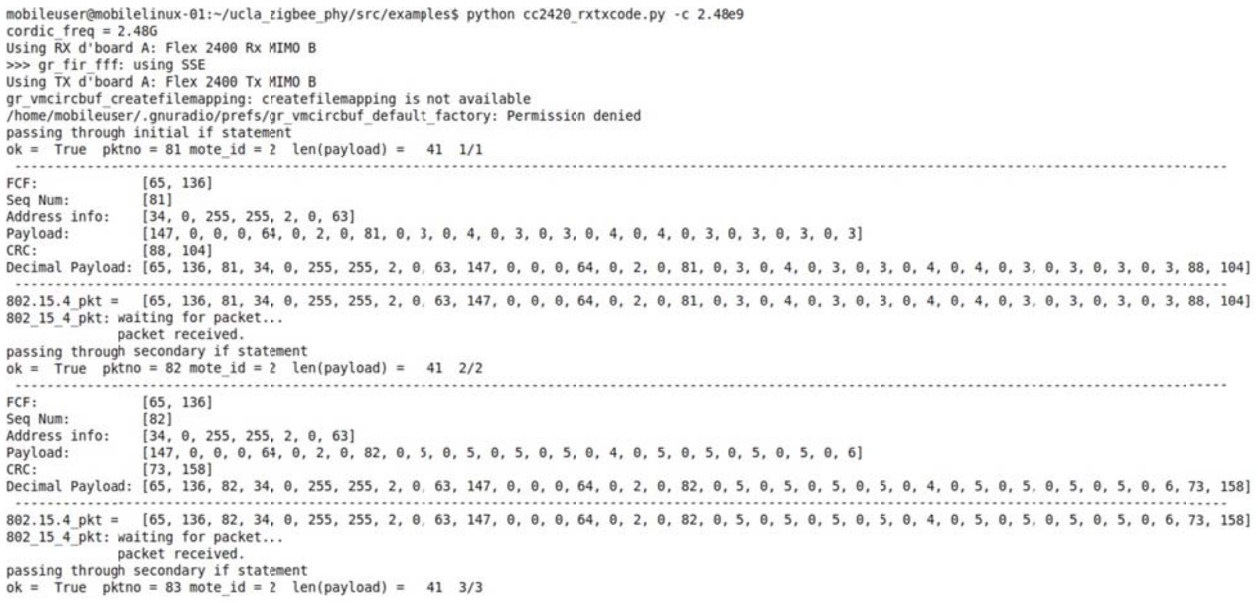
Figure: Data Packet Sent From USRP



```
mobileuser@mobilelinux-01:~/ucla_zigbee_phy/src/examples$ python cc2420_rxtxcode.py -c 2.48e9
cordic_freq = 2.48G
Using RX d'board A: Flex 2400 Rx MIMO B
>>> gr_fir_fff: using SSE
Using TX d'board A: Flex 2400 Tx MIMO B
gr_vmcircbuf_createfilemapping: createfilemapping is not available
/home/mobileuser/.gnuradio/prefs/gr_vmcircbuf_default_factory: Permission denied
passing through initial if statement
ok =  True  pktno = 8 mote_id = 1  len(payload) =   41  1/1
---------------------------------------------------------------------------------------------------------------------------------------
FCF:            [65, 136]
Seq Num:        [8]
Address info:   [34, 0, 255, 255, 1, 0, 63]
Payload:        [147, 0, 0, 0, 64, 0, 1, 0, 8, 0, 2, 0, 3, 0, 2, 0, 2, 0, 4, 0, 5, 0, 5, 0, 6, 0, 4, 0, 4]
CRC:            [28, 15]
Decimal Payload: [65, 136, 8, 34, 0, 255, 255, 1, 0, 63, 147, 0, 0, 0, 64, 0, 1, 0, 8, 0, 2, 0, 3, 0, 2, 0, 2, 0, 4, 0, 5, 0, 5, 0, 6, 0, 4, 0, 4, 28, 15]
---------------------------------------------------------------------------------------------------------------------------------------
802.15.4_pkt =   [65, 136, 8, 34, 0, 255, 255, 1, 0, 63, 147, 0, 0, 0, 64, 0, 1, 0, 8, 0, 2, 0, 3, 0, 2, 0, 2, 0, 4, 0, 5, 0, 5, 0, 6, 0, 4, 0, 4, 28, 15]
802_15_4_pkt: waiting for packet...
```

Figure: Data Received by USRP

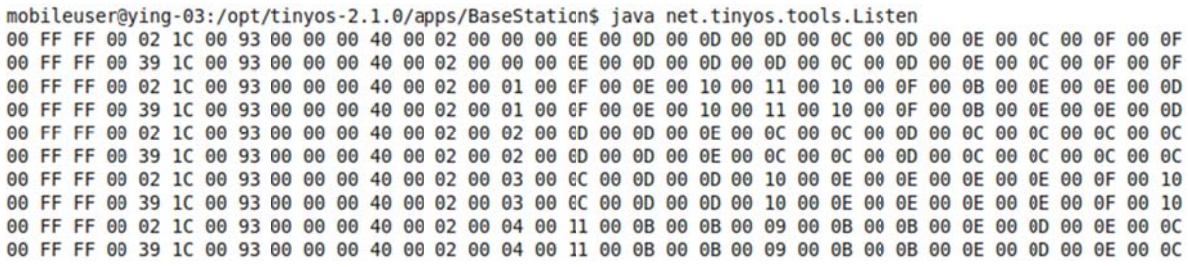
The above figures show our hardcoded message from the transmitted USRP to the second USRP. The FCF is generated by an underlying python script ieee802_15_4_pkt.py and we have modified it to give the FCF needed by the TelosB mote. The CRC is also generated by the same underlying script and is based on the data sent and acts as an error check for received packets.



```
mobileuser@mobilelinux-01:~/ucla_zigbee_phy/src/examples$ python cc2420_rxtxcode.py -c 2.48e9
cordic_freq = 2.48G
Using RX d'board A: Flex 2400 Rx MIMO B
>>> gr_fir_fff: using SSE
Using TX d'board A: Flex 2400 Tx MIMO B
gr_vmcircbuf_createfilemapping: createfilemapping is not available
/home/mobileuser/.gnuradio/prefs/gr_vmcircbuf_default_factory: Permission denied
passing through initial if statement
ok =  True  pktno = 81 mote_id = 2  len(payload) =   41  1/1
 ----------------------------------------------------------------------------------------------------------------------
FCF:            [65, 136]
Seq Num:        [81]
Address info:   [34, 0, 255, 255, 2, 0, 63]
Payload:        [147, 0, 0, 0, 64, 0, 2, 0, 81, 0, 3, 0, 4, 0, 3, 0, 3, 0, 4, 0, 4, 0, 3, 0, 3, 0, 3, 0, 3]
CRC:            [88, 104]
Decimal Payload: [65, 136, 81, 34, 0, 255, 255, 2, 0, 63, 147, 0, 0, 0, 64, 0, 2, 0, 81, 0, 3, 0, 4, 0, 3, 0, 3, 0, 4, 0, 4, 0, 3, 0, 3, 0, 3, 0, 3, 88, 104]
 ----------------------------------------------------------------------------------------------------------------------
802.15.4_pkt =  [65, 136, 81, 34, 0, 255, 255, 2, 0, 63, 147, 0, 0, 0, 64, 0, 2, 0, 81, 0, 3, 0, 4, 0, 3, 0, 3, 0, 4, 0, 4, 0, 3, 0, 3, 0, 3, 0, 3, 88, 104]
802_15_4_pkt: waiting for packet...
              packet received.
passing through secondary if statement
ok =  True  pktno = 82 mote_id = 2  len(payload) =   41  2/2
 ----------------------------------------------------------------------------------------------------------------------
FCF:            [65, 136]
Seq Num:        [82]
Address info:   [34, 0, 255, 255, 2, 0, 63]
Payload:        [147, 0, 0, 0, 64, 0, 2, 0, 82, 0, 5, 0, 5, 0, 5, 0, 5, 0, 4, 0, 5, 0, 5, 0, 5, 0, 5, 0, 6]
CRC:            [73, 158]
Decimal Payload: [65, 136, 82, 34, 0, 255, 255, 2, 0, 63, 147, 0, 0, 0, 64, 0, 2, 0, 82, 0, 5, 0, 5, 0, 5, 0, 5, 0, 4, 0, 5, 0, 5, 0, 5, 0, 5, 0, 6, 73, 158]
 ----------------------------------------------------------------------------------------------------------------------
802.15.4_pkt =  [65, 136, 82, 34, 0, 255, 255, 2, 0, 63, 147, 0, 0, 0, 64, 0, 2, 0, 82, 0, 5, 0, 5, 0, 5, 0, 5, 0, 4, 0, 5, 0, 5, 0, 5, 0, 5, 0, 6, 73, 158]
802_15_4_pkt: waiting for packet...
              packet received.
passing through secondary if statement
ok =  True  pktno = 83 mote_id = 2  len(payload) =   41  3/3
```

Figure: Sensor Data Received by the USRP

For this test condition we sent data from the TelosB mote and received the data on the USRP. Since the mote is wireless and unconnected to a computer we cannot verify the exact data. However, we can see that the packet number is increasing by one and the mote id is constant.   The payload data […,0,3,0,4,0,3,…,0,3,0,3,…] are the sensor readings. Each reading is a 16-bit integer and we covered the light sensor for this test so the values received are reasonable for this experiment.



```
mobileuser@ying-03:/opt/tinyos-2.1.0/apps/BaseStation$ java net.tinyos.tools.Listen
00 FF FF 00 02 1C 00 93 00 00 00 40 00 02 00 00 00 0E 00 0D 00 0D 00 0D 00 0C 00 0D 00 0E 00 0C 00 0F 00 0F
00 FF FF 00 39 1C 00 93 00 00 00 40 00 02 00 00 00 0E 00 0D 00 0D 00 0D 00 0C 00 0D 00 0E 00 0C 00 0F 00 0F
00 FF FF 00 02 1C 00 93 00 00 00 40 00 02 00 01 00 0F 00 0E 00 10 00 11 00 10 00 0F 00 0B 00 0E 00 0E 00 0D
00 FF FF 00 39 1C 00 93 00 00 00 40 00 02 00 01 00 0F 00 0E 00 10 00 11 00 10 00 0F 00 0B 00 0E 00 0E 00 0D
00 FF FF 00 02 1C 00 93 00 00 00 40 00 02 00 02 00 0D 00 0D 00 0E 00 0C 00 0C 00 0D 00 0C 00 0C 00 0C 00 0C
00 FF FF 00 39 1C 00 93 00 00 00 40 00 02 00 02 00 0D 00 0D 00 0E 00 0C 00 0C 00 0D 00 0C 00 0C 00 0C 00 0C
00 FF FF 00 02 1C 00 93 00 00 00 40 00 02 00 03 00 0C 00 0D 00 0D 00 10 00 0E 00 0E 00 0E 00 0E 00 0F 00 10
00 FF FF 00 39 1C 00 93 00 00 00 40 00 02 00 03 00 0C 00 0D 00 0D 00 10 00 0E 00 0E 00 0E 00 0E 00 0F 00 10
00 FF FF 00 02 1C 00 93 00 00 00 40 00 02 00 04 00 11 00 0B 00 0B 00 09 00 0B 00 0B 00 0E 00 0D 00 0E 00 0C
00 FF FF 00 39 1C 00 93 00 00 00 40 00 02 00 04 00 11 00 0B 00 0B 00 09 00 0B 00 0B 00 0E 00 0D 00 0E 00 0C
```

Figure: Sensor Data Relayed by USRP to Base Station Sensor

In this test, the BaseStation application from TinyOS will drop any duplicate packets so to ensure that the USRP was properly forwarding the received packet we modified the mote id. This results in the BaseStation app picking up two packets: one directly from the sensor and one from the USRP. The fifth byte denotes the mote id change and as one can see the rest of the data is the exact same for all other bytes which shows that the packet is forwarded correctly.

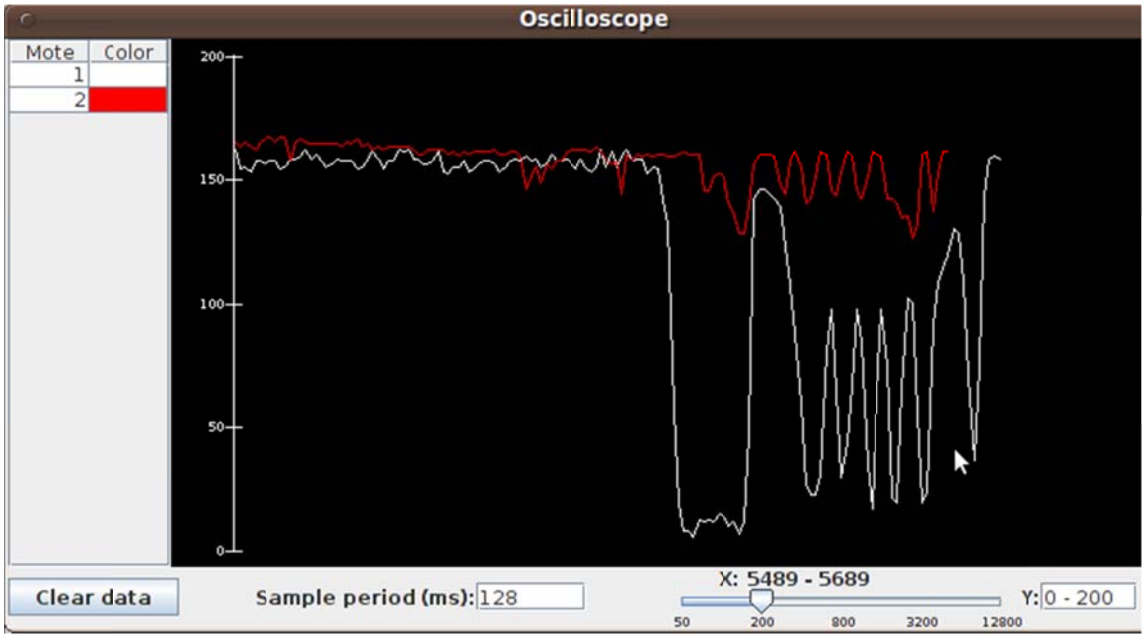
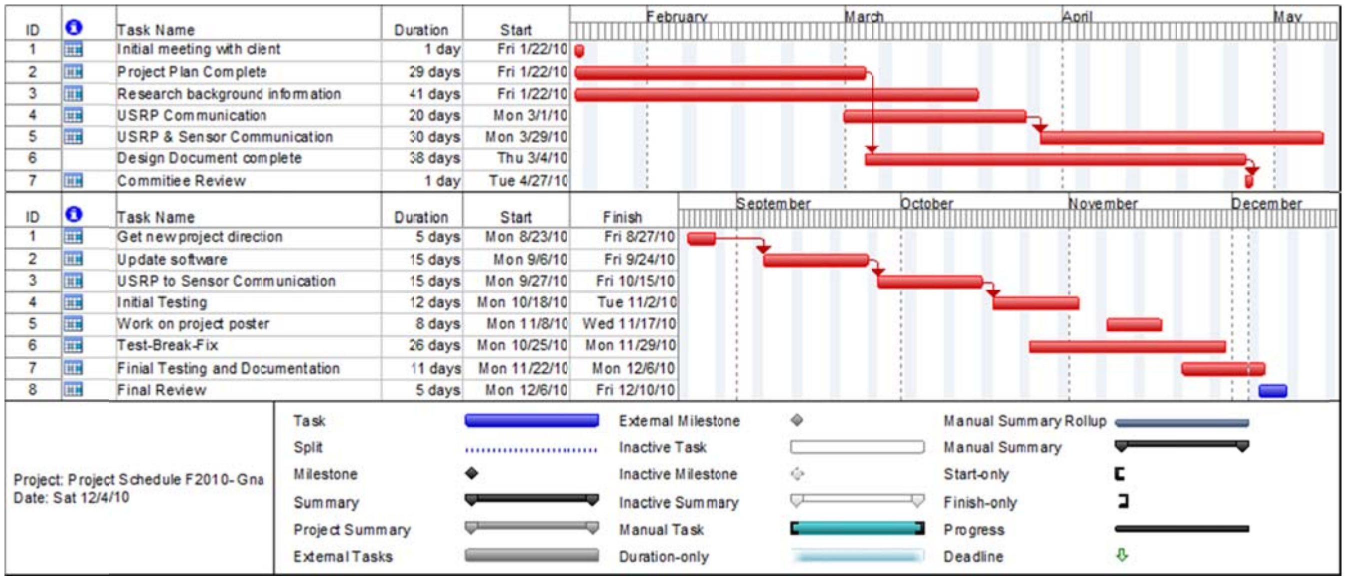


Figure: Light Sensor Graphical Data

Estimated Resources and Cost

Resource	Number of Units	Cost per Unit	Total
USRP (Base Kit)	2	$700	$1,400
USRP2 (Base Kit)	3	$1400	$4,200
RFX 2400 Transceiver daughterboard	5	$275	$1,375
VERT 2450 Antenna	5	$35	$175
Sensors	20	$139	$2,780
Laptops	5	$800	$4,000
Work Hours	450	$20/Hour	$9,000
SD Cards	3	$20	$60
Total Cost			$22,990

Project Schedule



ID	Task Name	Duration	Start
1	Initial meeting with client	1 day	Fri 1/22/10
2	Project Plan Complete	29 days	Fri 1/22/10
3	Research background information	41 days	Fri 1/22/10
4	USRP Communication	20 days	Mon 3/1/10
5	USRP & Sensor Communication	30 days	Mon 3/29/10
6	Design Document complete	38 days	Thu 3/4/10
7	Commitiee Review	1 day	Tue 4/27/10

ID	Task Name	Duration	Start	Finish
1	Get new project direcion	5 days	Mon 8/23/10	Fri 8/27/10
2	Update software	15 days	Mon 9/6/10	Fri 9/24/10
3	USRP to Sensor Communication	15 days	Mon 9/27/10	Fri 10/15/10
4	Initial Testing	12 days	Mon 10/18/10	Tue 11/2/10
5	Work on project poster	8 days	Mon 11/8/10	Wed 11/17/10
6	Test-Break-Fix	26 days	Mon 10/25/10	Mon 11/29/10
7	Finial Testing and Documentation	11 days	Mon 11/22/10	Mon 12/6/10
8	Final Review	5 days	Mon 12/6/10	Fri 12/10/10

Project: Project Schedule F2010-Gna
Date: Sat 12/4/10

Task Breakdown

Semester 1#

	Research	Hardware Implementation	Software Design	Documentation	Total
Wyatt Brenneman	35	5	10	30	80
Taylor McKechnie	40	0	10	25	75
Prashanth Yanamandra	35	5	10	30	80
Total	105	10	30	90	215

Semester 2

	Transmission	Receiving	Signal Processing	Testing	Documentation	Total
Wyatt Brenneman	20	20	5	30	30	105
Taylor McKechnie	10	10	10	10	30	70
Prashanth Yanamandra	5	5	5	5	40	60
Total	35	35	20	45	100	235

#

Client/Advisors Information

Professor Lei Ying
3219 Coover Hall
Ames, IA 50011
leiying@iastate.edu
515 294-5353

Ming Ouyang
2215 Coover Hall
Ames, IA 50011
mouyang@iastate.edu
515 294-2664

Project Team Information

Wyatt Brenneman
232 Walnut Ave Unit 16
Ames, IA 50010
wgbn01@iastate.edu
319 981-7003

Taylor McKechnie
225 Hyland Ave Apt 12
Ames, IA 50014
thrp1st@iastate.edu
402 926-1033

Prashanth Yanamandra
143 University Village Apt F
Ames, IA 50010
psy@iastate.edu
321 704-6321

Closing Summary

In a world where the need of fast and reliable exchange of information is growing our network will showcase one possibility of how this can be achieved. Our network will rely on the flexibility of P2P sharing and combine it with the immediate gratification of streaming data. By streaming the data the users will not be required to wait to view the information they need as well and continue to propagate the data through the network.

References

[1] Crossbow Wireless Modules Portfolio – Crossbow Technology. Retrieved March 1, 2010. From the World Wide Web : < http://www.xbow.com/Products/productdetails.aspx?sid=156>.

[2] TinyOS." *Wikipedia, the free encyclopedia*. Retrieved December 4, 2010 from the World Wide Web: <http://en.wikipedia.org/wiki/TinyOS>.

[2] Ettus Research LLC. Retrieved March 1, 2010. From the World Wide Web : <http://www.ettus.com/>.

[3] "GNU Radio". Retrieved March 1, 2010 from the World Wide Web. <http://gnuradio.org/trac>.

[4] "GNU Radio." *Wikipedia, the free encyclopedia*. Retrieved March 1, 2010 from the World Wide Web: <http://en.wikipedia.org/wiki/GNU_Radio>.

[5] ISU Electrical and Computer Engineering: ECpE Building Addition and Coover Hall Renovations Project. Retrieved March 1, 2010 from the World Wide Web: <http://www.ece.iastate.edu/typo3temp/pics/6552a907c0.gif>.

[6] Jackey, John. FPGA for MRFM Cantilever Control. Retrieved April 26, 2010 from the World Wide Web :< http://www.research.cornell.edu/KIC/events/MRFM2006/pdfs/Jacky%20talk/jackytalk.html>.

[7] "Peer-to-peer." *Wikipedia, the free encyclopedia*. Retrieved March 1, 2010 from the World Wide Web: <http://en.wikipedia.org/wiki/Peer-to-peer>.

[8] "Universal Software Radio Peripheral." *Wikipedia, the free encyclopedia*. Retrieved March 1, 2010 from the World Wide Web: <http://en.wikipedia.org/wiki/Universal_Software_Radio_Peripheral>.

Appendix

The appendix contains the code designed for the transmitting and receiving functions of the USRP

```
#!/usr/bin/env python
#
# This program was written for a senior design project at Iowa State University
# under the supervision of Professor Lei Ying, and Ming Ouyang.
#
# Receives and Forwards 802.15.4 RADIO packets from TelosB Wireless sensor motes
#
# This code is a modified combination of cc2420_rxtest.py and cc2420txtext.py
# from the UCLA Zigbee PHY project examples.  The cc2420_rxtest.py and
# cc2420_txtest.py were created By: : Thomas Schmid, Leslie Choong, Mikhail Tadjikov,
# and Sanna Leidelof.
#
# Written and Modified by: Wyatt Brenneman and Taylor Mckechnie

from gnuradio import gr, eng_notation
from gnuradio import usrp
from gnuradio.ucla_blks import ieee802_15_4_pkt
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import math,struct, sys, random
def pick_subdevice(u):
    """
    The user didn't specify a subdevice on the command line.
    If there's a daughterboard on A, select A.
    If there's a daughterboard on B, select B.
    Otherwise, select A.
    """
    if u.db(0, 0).dbid() >= 0:       # dbid is < 0 if there's no d'board or a problem
        return (0, 0)
    if u.db(1, 0).dbid() >= 0:
        return (1, 0)
    return (0, 0)
class stats(object):
    def __init__(self):
        self.npkts = 0
        self.nright = 0
#/////////////////////////////////////////////////////////////////////////////////////////////
#                                   Combine Transmit and Receive Paths
#/////////////////////////////////////////////////////////////////////////////////////////////
class path_block(gr.top_block):
    i = 0
    mote_id_sent = 0
    packet_num_sent = 0
    def __init__(self):
            gr.top_block.__init__(self)
                    def rx_callback(ok, payload):
```

```
                    payload2 = map(hex, map(ord, payload))
                    payloaddec = map(ord, payload)
                    packet_num = payload[2]
                    mote_id = payload[7]
                    #app.start()
                    if path_block.i==0:                    #Allows for first received packet to be transmitted
                              print "passing through initial if statement"
                          st.npkts += 1
                          if ok:
                             st.nright += 1
                          print "ok = %5r  pktno = %d mote_id = %d  len(payload) = %4d  %d/%d" % (ok,
ord(payload[2]), ord(payload[7]), len(payload),
                                                                                   st.nright, st.npkts)

                         print " ------------------------------------------------------------------------------------------
------------------------------------------------------------"
                              print "FCF:            " + str(map(ord, payload[0:2]))
                              print "Seq Num:        " + str(map(ord, payload[2]))
                              print "Address info:    " + str(map(ord, payload[3:10]))
                              print "Payload:        " + str(map(ord, payload[10:39]))
                              print "CRC:            " + str(map(ord, payload[39:41]))
                              print "Decimal Payload: " + str(payloaddec)
                              self.txpath.send_pkt(payload,False)
                              print "802_15_4_pkt: waiting for packet..."
                              path_block.i=1
                    elif (path_block.packet_num_sent != packet_num and path_block.mote_id_sent ==
mote_id) or path_block.mote_id != mote_id:
                              print"            packet received."
                              print "passing through secondary if statement"                #Will not allow
transmission if received packet was just transmitted
                                                                                                #This prevents an
infinite transmission loop
                          st.npkts += 1
                          if ok:
                             st.nright += 1

                          (pktno,) = struct.unpack('!H', payload[0:2])
                          print "ok = %5r  pktno = %d mote_id = %d  len(payload) = %4d  %d/%d" % (ok,
ord(payload[2]), ord(payload[7]), len(payload),
                                                                                   st.nright, st.npkts)

                          print " ------------------------------------------------------------------------------------------
------------------------------------------------------------"
                              print "FCF:            " + str(map(ord, payload[0:2]))
                              print "Seq Num:        " + str(map(ord, payload[2]))
                              print "Address info:    " + str(map(ord, payload[3:10]))
                              print "Payload:        " + str(map(ord, payload[10:39]))
                              print "CRC:            " + str(map(ord, payload[39:41]))
                              print "Decimal Payload: " + str(payloaddec)
                              self.txpath.send_pkt(payload,False)
                              print "802_15_4_pkt: waiting for packet..."
                    path_block.packet_num_sent = packet_num
                    path_block.mote_id_sent = mote_id
                    sys.stdout.flush()

            parser = OptionParser (option_class=eng_option)
            parser.add_option("-R", "--rx-subdev-spec", type="subdev", default=None,
                    help="select USRP Rx side A or B (default=first one with a daughterboard)")
            parser.add_option("-T", "--tx-subdev-spec", type="subdev", default=None,
                    help="select USRP Tx side A or B (default=first one with a daughterboard)")
            parser.add_option ("-c", "--cordic-freq", type="eng_float", default=2480000000,
                     help="set rx cordic frequency to FREQ", metavar="FREQ")
            parser.add_option ("-r", "--data-rate", type="eng_float", default=2000000)
            parser.add_option ("-f", "--filename", type="string",
                     default="rx.dat", help="write data to FILENAME")
            parser.add_option ("-g", "--gain", type="eng_float", default=0,
                     help="set Rx PGA gain in dB [0,20]")

            (options, args) = parser.parse_args ()

            st = stats()
            i = 0
            mote_id_sent = 'null'
            packet_num_sent = 'null'

            self.rxpath = oqpsk_rx_graph(options, rx_callback)
            self.txpath = transmit_path(options)
            self.connect(self.txpath)
            self.connect(self.rxpath)

#//////////////////////////////////////////////////////////////////////////////////////////
#                                                   Recieve Block
#//////////////////////////////////////////////////////////////////////////////////////////
class oqpsk_rx_graph (gr.hier_block2):
    def __init__(self, options, rx_callback):
            gr.hier_block2.__init__(self, "oqpsk_rx_graph",
                                                   gr.io_signature(0, 0, 0), # Input signature
                                                   gr.io_signature(0, 0, 0)) # Output signature

        print "cordic_freq = %s" % (eng_notation.num_to_str (options.cordic_freq))

        self.data_rate = options.data_rate
        self.samples_per_symbol = 2
        self.usrp_decim = int (64e6 / self.samples_per_symbol / self.data_rate)
        self.fs = self.data_rate * self.samples_per_symbol
        payload_size = 128             # bytes

                u = usrp.source_c (0, self.usrp_decim)
        if options.rx_subdev_spec is None:
            options.rx_subdev_spec = pick_subdevice(u)
        u.set_mux(usrp.determine_rx_mux_value(u, options.rx_subdev_spec))
        subdev = usrp.selected_subdev(u, options.rx_subdev_spec)
        print "Using RX d'board %s" % (subdev.side_and_name(),)

        u.tune(0, subdev, options.cordic_freq)
        u.set_pga(0, options.gain)
        u.set_pga(1, options.gain)

        self.u = u

        self.packet_receiver =
ieee802_15_4_pkt.ieee802_15_4_demod_pkts(self,callback=rx_callback,sps=self.samples_per_symbol,s
ymbol_rate=self.data_rate,
                                                                threshold=-1)

        self.squelch = gr.pwr_squelch_cc(50, 1, 0, True)
        self.connect(self.u, self.squelch, self.packet_receiver)


#////////////////////////////////////////////////////////////////////////////////////////////
#                                               Transmit Block
#////////////////////////////////////////////////////////////////////////////////////////////
class transmit_path(gr.hier_block2):
    def __init__(self, options):
            gr.hier_block2.__init__(self, "transmit_path",
                                            gr.io_signature(0, 0, 0), # Input signature
                                            gr.io_signature(0, 0, 0)) # Output signature

        self.normal_gain = 8000
        self.u = usrp.sink_c()
        dac_rate = self.u.dac_rate();
        self._data_rate = 2000000
        self._spb = 2
        self._interp = int(128e6 / self._spb / self._data_rate)
        self.fs = 128e6 / self._interp
        self.u.set_interp_rate(self._interp)

        # determine the daughterboard subdevice we're using
        if options.tx_subdev_spec is None:
            options.tx_subdev_spec = usrp.pick_tx_subdevice(self.u)
        self.u.set_mux(usrp.determine_tx_mux_value(self.u, options.tx_subdev_spec))
        self.subdev = usrp.selected_subdev(self.u, options.tx_subdev_spec)
        print "Using TX d'board %s" % (self.subdev.side_and_name(),)

        self.u.tune(0, self.subdev, options.cordic_freq)
        self.u.set_pga(0, options.gain)
        self.u.set_pga(1, options.gain)

        # transmitter
        self.packet_transmitter = ieee802_15_4_pkt.ieee802_15_4_mod_pkts(self, spb=self._spb,
msgq_limit=2)
        self.gain = gr.multiply_const_cc (self.normal_gain)
        self.connect(self.packet_transmitter, self.gain, self.u)
        self.set_gain(self.subdev.gain_range()[1])  # set max Tx gain
        self.set_auto_tr(True)                      # enable Auto Transmit/Receive switching

    def set_gain(self, gain):
        self.gain = gain
        self.subdev.set_gain(gain)

    def set_auto_tr(self, enable):
        return self.subdev.set_auto_tr(enable)

    def send_pkt(self, payload, eof=False):
            print " ---------------------------------------------------------------------------------------------------------------
----------------------------------------------"

        return self.packet_transmitter.send_pkt(ord(payload[2]),payload[3:10] ,payload[10:39], eof)



def main ():

    app = path_block()
    app.start()
    app.wait()
if __name__ == '__main__':
    main ()
```