IMPLEMENTATION OF WIRELESS COMMUNICATIONS ON GNU RADIO

Simon M. Njoki

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

May 2012

APPROVED:

Kamesh Namuduri, Major Professor
Hyoung Kim, Committee Member
Miguel Acevedo, Committee Member
Murali Varanasi, Chair of the Department of Electrical Engineering
Costos Tsatsoulis, Dean of the College of Engineering
James D. Meernik, Acting Dean of the Toulouse Graduate School

Njoki, Simon M. Implementation of wireless communications on GNU Radio. Masters of Science (Electrical Engineering), May 2012, 44 pp., 13 figures, 5 tables, references, 35 titles.

This thesis investigates the design and implementation of wireless communication system over the GNU Radio. Wireless applications are on the rise with advent of new devices, therefore there is a need to transfer the hardware complexity to software. This development enables software radio function with minimum hardware dependency. The purpose of this thesis is to design a system that will transmit compressed data via Software Defined Radio (SDR). Some parameters such as modulation scheme, bit rate can be changed to achieve the desired quality of service. In this thesis GNU (GNU's not unix) radio is used while the hardware structure is Universal Software Radio Peripheral (USRP). In order to accomplish the goal, a compression technique called H264 (MPEG_4) encoding is applied for converting data into compressed format.

The encoder was implemented in C++ to get compressed data. After encoding, the transmitter reads the compressed data and starts modulation. After modulation, the transmitter put the packets into USRP and sends it to the receiver. Once packets are received they are demodulated and then decoded to recover the original data.

ACKNOWLEDGEMENTS

First, I would like to express my gratitude to Dr.Kamesh Namuduri, for introducing me to this interesting field of GNU (GNU's not Unix) radio and encouraging me to pursue my thesis on this topic and for his guidance throughout my stay in the university. I would like to thank Dr. Hyosung Kim and Dr. Miguel Acevedo for their support as committee members of my thesis. Lastly, I am grateful to my family and friends for their support over the years, and especially during the last several months of intense work on this thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

1.1 Overview

The demand for being connected has caused an exponential growth in wireless communications. However, hardware-based approach to traditional radio design imposes significant limitations. Software defined radio (SDR) technology [1] [2] brings flexibility, cost efficiency and power to drive communications forward, with benefits realized by service providers and product developers through end users [3].

The users can use relatively generic hardware, and customize it to their needs by choosing the software that fits specific application. One obvious benefit is that instead of building extra circuitry to handle different types of radio signals, one can just load an appropriate script. It would be so difficult to build a new circuitry any time you want to do the hardware upgrade. By reusing identical hardware platform for many terminals with different protocols, it is possible to reduce the time to market and development cost. SDR allows service providers to upgrade infrastructure without unreasonable cost [1].

The basic concept of SDR is to make radio functions hardware independent. Complex tasks like modulation, demodulation, encoding, decoding, filtering etc are implemented in software which eliminates the need for corresponding hardware. SDR leaves the hardware which is USRP (universal software radio peripheral) to take care of functions such as transmissions and reception of signal while GNU Radio does the entire complex signal processing on the general purpose processor. A combination of GNU Radio and USRP helps realize SDR. GNU Radio includes C++ classes that implement various signal processing functions.

While the signal processing blocks are implemented in C++, the main application is written in Python. SWIG (Simplified Wrapper and Interface Generator) works as glue between C++ classes and Python language. SWIG is a Linux package that converts the C++ classes into Python compatible classes [3]. GNU Radio framework is able to harness both languages. C++ provides compact code for signal processing block, while Python is provides flexibility and ease of programming. Using Python and C++ in combination, the signal processing units are implemented on general purpose processor by GNU Radio.

In order to transmit the desired signal at different rate, variable modulation schemes are introduced. Modulation provides more information capacity, compatibility with digital services, higher data security, and better quality communications.

Communication systems face several constraints such as limited bandwidth, limited power, and inherent noise level on the channels. This research investigates how different modulation schemes and bit rates will affect the signal quality. It evaluates two modulation schemes which are GMSK (gaussian mean shift keying) and BPSK (Binary Phase Shift Keying).

1.2 Motivation

SDR is a relatively new platform for research. The implementation of SDR platforms is quite challenging. This thesis is based on the original work on image communication using GNU Radio by Zhifeng Cheng [16]. In his work, Cheng used TCP/IP sockets to simulate wireless transmission as shown in figure 1.1. This thesis extends Cheng's approach by replacing sockets with USRP and adding BPSK and GMSK modules.

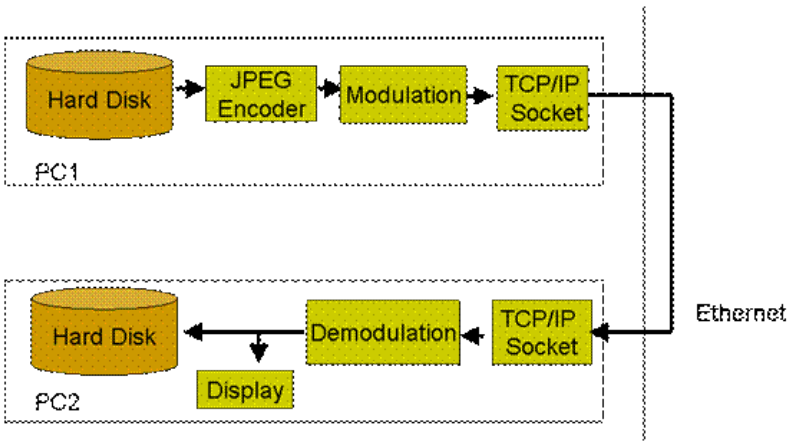

Figure 1.1. Architecture of TCP/IP

After close examination of potentially available platforms, Ettus Research's USRP SDR platform and Gnu Radio open-source software development toolkit were chosen.

1.3 Methodology

SDR and USRP require understanding knowledge of digital signal processing and digital communications. Several potential SDR platforms are analyzed. (USRP1) platform was chosen.

1.4 Major Issues

The major challenge is to select proper Unix/Linux platform. Our first choice, Suse 10.1 turned out to be very user unfriendly. It was very difficult to find all documentation to set up the USRP. The second option which is Ubuntu 10.10 worked out very well.

1.5 Performance Evaluation

The quality of a digital communication system is expressed in terms of accuracy of data received. It is generally measured as the fraction of the bits that are delivered with error. This

fraction is referred to as bit error probability or bit error rate (BER).

In this thesis, the performances of two modulation schemes, BPSK and GMSK schemes are compared while keeping the bit rate constant. A brief description of the two modulations is compared in the next subsection.

1.5.1 Binary Phase Shift Keying (BPSK)

In Binary Phase Shift Keying (BPSK) the information about the bit stream is contained in the changes of phase of the transmitted signal. With BPSK, the binary digits 1 and 0 maybe represented by analog levels $+\sqrt{E_b}$ and $-\sqrt{E_b}$ respectively. The system model is shown with figure 1.1 [1].

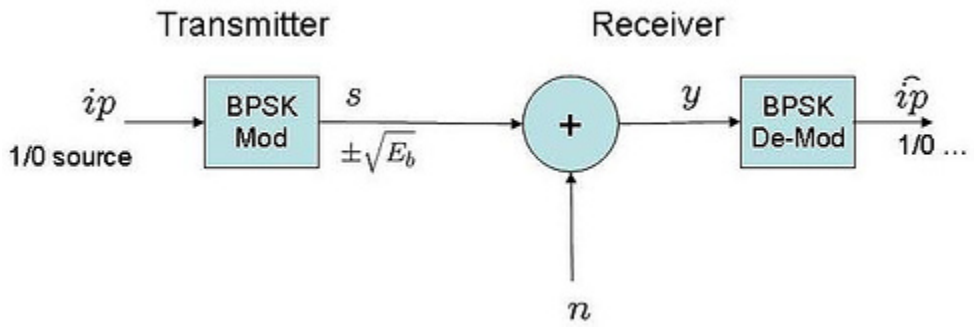

Figure 1.2. BPSK model

In this model the received is given by the equations below:

- y=s1+n when the bit 1 is transmitted
- y=s0+n when the bit zero is transmitted

The conditional probability distribution functions (PDFs) of y, x for the two instances are given by:

- $p(y|s_0) = \frac{1}{\sqrt{\pi N_0}} e^{\frac{-(y+\sqrt{E_b})^2}{N_0}}$
- $p(y|s_1) = \frac{1}{\sqrt{\pi N_0}} e^{\frac{-(y-\sqrt{E_b})^2}{N_0}}$ [1]

Assuming (s0, s1) are equally probable and threshold zero makes the decision we find that if the received signal is greater than 0, the receiver assumes 1 is transmitted and if the received signal is less than or equal to zero, then the receiver assumes 0 was transmitted i.e.

- y>0 => s1 and
- y ≤ 0 =>s0

1.5.2 GMSK System Model

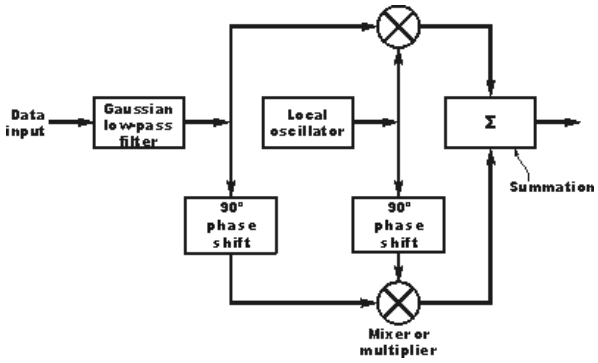

Figure 1.3. GMSK model

GMSK can be viewed as either frequency or phase modulation. The phase of a carrier is advanced or retarded up to 90 degrees over the course of a bit period depending on data pattern. The rate of change of phase is controlled by the Gaussian filter. The net result of this is that depending on the bandwidth time product (BT), the phase change may fall short of 90 degrees. This has an impact on the BER although the advantage of this scheme is the improved bandwidth.

1.6 Choice of Modulation

There are factors that determine the choice of a modulation scheme in wireless applications. Performance of a wireless system is dependent on the efficiency of the modulation scheme in use. The goal of modulation technique is not only to transport a message signal through a radio channel, but also to achieve this with the best quality, power efficiency, and least amount of bandwidth.

CHAPTER 2

SOFTWARE DEFINED RADIO

2.1 Software Defined Radio (SDR) Design

In order to demonstrate the ability of SDR, we created different systems using various modulations schemes with constant bit rate. These systems were debugged and tested using USRP boards which will be discussed in later chapter. Figure 2.1 shows SDR transmitter while Figure 2.2 shows the SDR receiver.

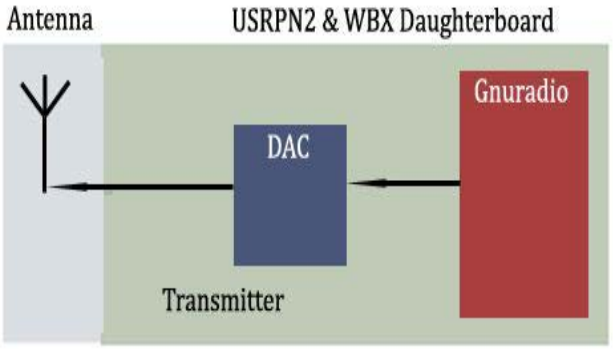

Figure 2.1. Transmitter

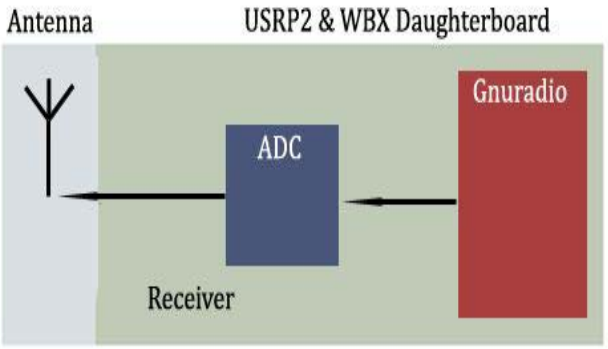

Figure 2.2. Receiver

The system on either side has an radio frequency (RF) front end, each consisting a daughterboard and antenna, along with digital to analog and analog to digital converters and field programmable gate array (FPGA) that is loaded with software from GNU Radio.

2.2 GNU Radio

GNU Radio is an open source software kit tool signal processing package which performs encoding and decoding. It provides the signal processing run time and processing blocks to implement software defined radio [4]. GNU Radio allows programmers to implement SDR

applications on all kinds of PC operating systems, like Linux, Windows, UNIX and Mac OS. In this research Ubuntu Linux operating system was used.

In GNU Radio all applications are written in Python programming language while critical portions such as signal processing blocks are implemented in C++. GNU Radio take this advantages of C++ to realize highly optimized signal processing code as a well as the user friendly language Python to construct applications [5].

Simplified Wrapper and Interface Generator (SWIG) which comes between Python and C++ is used as glue between them. The users can design their own blocks using C++ and install those blocks to the library after generating the Python code by SWIG. Graphs are constructed and run in Python. For example, "*Hello World* " in GNU Radio generates two sine waves and outputs them to the sound card, one on the left channel, one on the right channel.

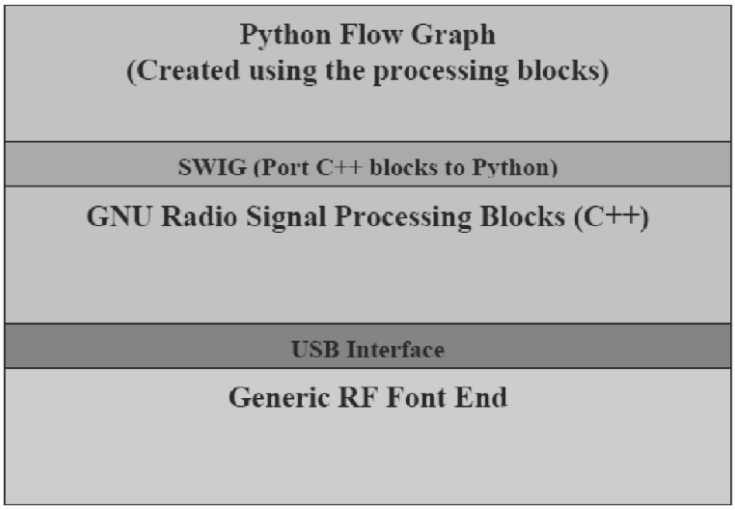

Figure 2.3. GNU Radio block diagram

2.3 Python

Python is dynamic object-oriented programming language that has capability to

interface with other languages. It is simple to use and offers much more structure and support for large programs than a shell script could offer. Python is available on almost all operating systems including Windows, Mac OS, and UNIX operating systems such Linux. Basically, in GNU Radio platform all the signal blocks that are written in C++ are connected by Python. Figure 2.4 is a diagram of dial tone generator, and Figure 2.5 is a simple Python code for this dial tone generator.

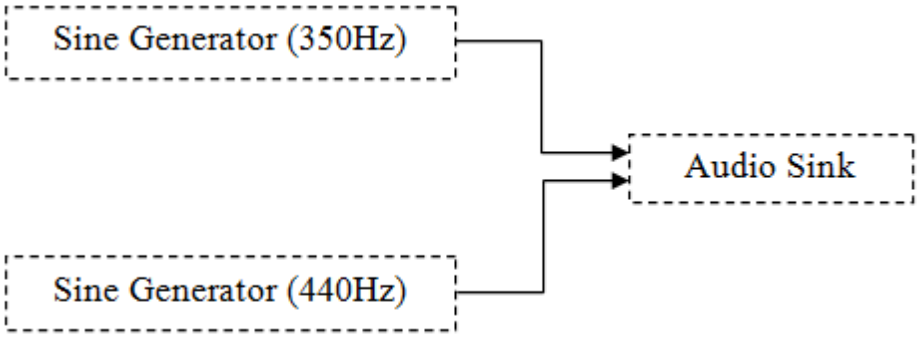

Figure 2.4. Block processing of dial toner generator

```
1 #!/usr/bin/env python

2 from gnuradio import gr
3 from gnuradio import audio
4 def build_graph ():
5       sampling_freq = 48000
6       ampl = 0.1
7       fg = gr.flow_graph ()
8       src0 = gr.sig_source_f (sampling_freq, gr.GR_SIN_WAVE, 350, ampl)
9       src1 = gr.sig_source_f (sampling_freq, gr.GR_SIN_WAVE, 440, ampl)
10      dst = audio.sink (sampling_freq)
11      fg.connect ((src0, 0), (dst, 0))
12      fg.connect ((src1, 0), (dst, 1))

13      return fg
14      if __name__ == '__main__':
15      fg = build_graph ()
16      fg.start ()
17      raw_input ('Press Enter to quit: ')
18      fg.stop ()
```

Figure 2.5. Python programming of dial tone generator

In the program, above every command line is unique and very important in the running of the Python dial tone generator. In first command line it shows the python location file which is important to run python file directly. In C++/C programming, we use directive *# include* and this is same as using *import* in Python programming. In this command line two modules are imported from GNU Radio-package, *audio* and *gr* which are very important in running GNU Radio application. Lines 4-12 define another class of block called *my_top_block* and this comes from another class called *gr.top_block*. It helps in constructing the flow graph especially when one wants to represent the signal in GNU Radio Companion (GRC). In line 4, we see that in order to define function, command *def* is used. In our program, we have variables *sample rate* and *amp1* and these are for sampling rate and signal generator. In reference to line 13, two signals sources *src0* and *src1* are generated. *Src0* is a sine wave is with sampling rate of 3200, amplitude of 0.1, and reference frequency of 350Hz.

Src1 is like *src0*, but with frequency of 440 Hz. We use prefix to show the type of float output and accept floating numbers samples in the range of –1 and +1 which is *f of* the block *gr.sig_source_f*. In line 10, signal sink is defined as *audio.sink ()* and this plays back any samples fed into it. The instructions of the blocks and these are shown in lines 8 and 9 run the program self.connect (block1, block2, block3) [5]. Python script is executed with command chmod+x file.py.

2.4 USRP

USRP is the hardware platform for SDR which is composed of programmable field gate array (FPGA), analog to digital converter/ digital to analog converter (ADC/DAC) and universal

bus controller (USB). It is designed to interface analog signal with the software. It takes an analog signal and interfaces it with SDR libraries, such as gnuradio, Simulink within Matlab, Labview and UHD (Universal Hardware Driver) [3].

USRP is basically a motherboard with FPGA as well as USB microcontroller. It has a daughterboard which has both transmitter (TX) and receiver (RX). The daughterboard contains transformers with impedance match, the 200 Ohms input of the mixed signal to analog devices AD9862 front end. The two onboard analog devices AD9862 capture the data; do decimation and interpolation tasks and filtering. Altera FPGA outputs stream of data into Cypress FX2 microcontroller. The microcontroller accesses the interface between FPGA and universal USB 2.0 so that we can transfer data into PC. In addition to two analog signal input or output for basic TX and RX boards we also have access to various auxiliary digitizers as part of mixed signal front end.

2.4.1 Applications of USRP

- An APC025 compatible transmitter/receiver and decoder
- RFID reader
- Testing equipment
- A cellular GSM base station
- A GPS receiver
- An FM radio transmitter
- A digital television (ATSC) decoder
- Passive order

- Synthetic aperture radar
- An amateur radio
- A teaching aid
- Digital broadcasting (DAB/DAB+/DMB) transmitter
- Mobile WIMAX receiver with USRP N2x0

Figure 2.6 is a picture of USRP board in a casing while figure 2.7 is block diagram of URSP system [3].

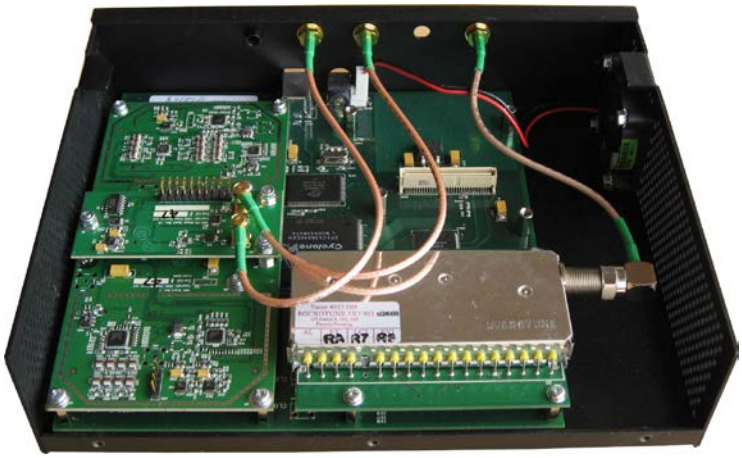

Figure 2.6. USRP board in case

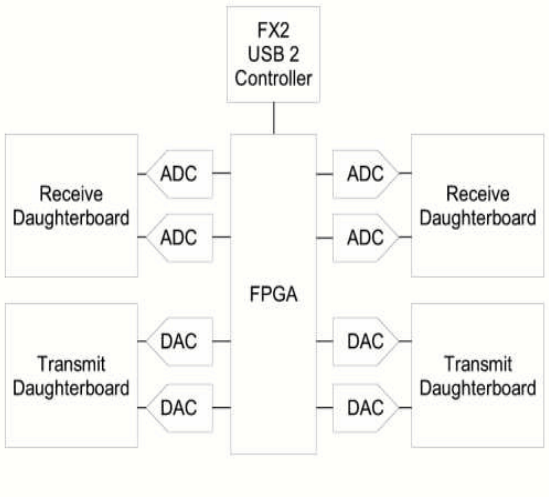

Figure 2.7. Block diagram of USRP system

2.5 Daughterboard

Ettus research offers many daughterboards with differing features. The daughterboard are easily installed and available for any project. In this research, two transceivers (XCVR) 2450 daughterboards are used. XCVR 2450 covers ISM band, 2.4 GHz, and entire 4.9 to 5.9 GHz band including the public safety, UNII, ISM, and Japanese wireless bands [3]. They have transmit power of 100mW and single synthesizer shared between TX and RX.

Daughterboards make it possible to use USRP in different frequency spectrums. This is because there are physical RF components needed to receive different frequency spectrum. On the motherboard there are four slots where one can plug up to 2 TX and 2 RX daughterboard. There are two slots for 2 TX daughterboard, labeled TXA and TXB, and 2 corresponding RX daughterboards, RXA and RXB. Each daughterboard slot has access to 2 of the 4-high speed AD/DA converters. It is also possible to use transceiver daughterboard to enable USRP to send and receive simultaneously [3]. Below is a list of various daughterboards that are available [3].

- Basic RX: Receiver for use with external RF hardware
- Basic TX: Transmitter for use with external RF hardware
- LFRX: DC to 30 MHz receiver
- LFTX: DC to 30 MHz transmitter
- TVRX: 50 to 860 MHz receiver
- DBSRX: 800 MHz to 2.4 GHz
- WBX: 50 MHz to 2.2 GHz transceiver
- RFX400: 400-500 MHz transceiver
- RFX900 : 750-1050 MHz transceiver

- RFX1200: 1150-1450 MHz transceiver
- RFX2400: 2.3-2.9 GHz transceiver
- XCVR2450: 2.4 GHz and 5 GHz dual band transceiver

2.6 Analog –Digital –Converter (ADC)

ADC converter converts analog signal to digital. In USRP board there are four 12- bit high-speed A to D with sampling arte of 64 million samples per second. In reality, it could digitize a band with wide width of 32 MHz [8]. The only problem is, it is not possible to receive signals with bandwidth larger than 32 MHz therefore the 32 MHz is calculated based on Nyquist rate [3].

2.7 Digital –Analog –Converter (DAC)

The DAC converts a digital constructed signal to analog. On transmitter side, there are four high-speed 14-bit D to A converters. The DAC clock frequency is 128 million samples per second, and Nyquist rate is 64 MHz [7].

2.8 Field Programmable Gate Array (FPGA)

The FPGA is the heart of the USRP. The FPGA being used in this research is Cyclone FPGA manufactured by Cypress. Using a good USB controller, the USRP can sustain 32 MB/s across the USB [10]. Figure 2.8 shows how digital down converter can be implemented.

16-bit data to host usb

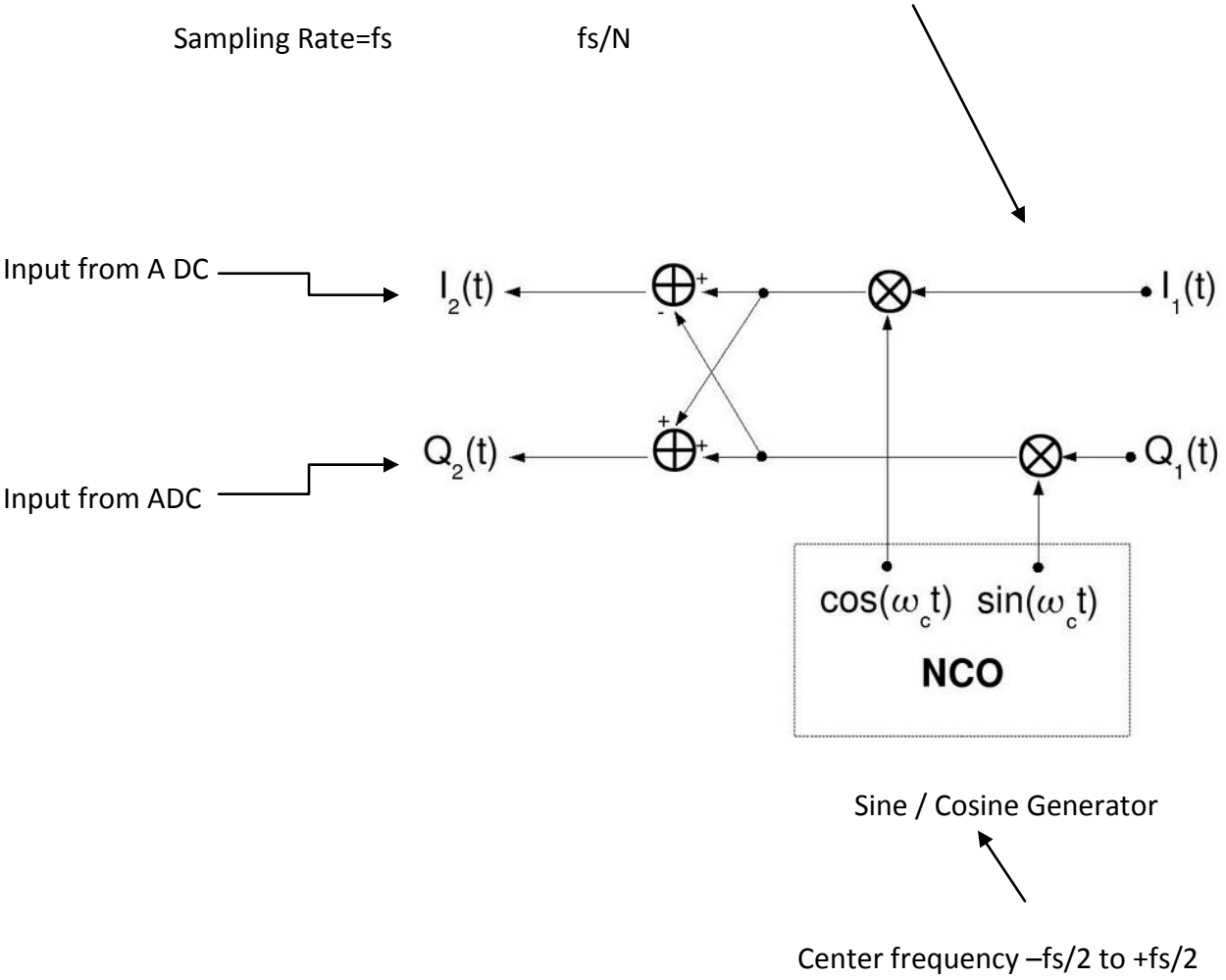

Figure 2.8. Digital down converter block diagram

2.9 Universal Serial Bus (USB) 2.0 Controller

USB is used to connect the USRP to the computer. USRP connects to the computer via a high speed USB2 interface only. It cannot work with USB 1.1. The data throughput is 32 MB/s. The USB connection has an impact on the performance [10].

CHAPTER 3

SOFTWARE DEFINED RADIO PERIPHERAL AND GNU RADIO SETUP

3.1 USRP Specifications

Universal software radio peripheral (USRP) is very important platform for SDR. USRP provides the hardware platform for the GNU Radio project. It was first released in 2004, and was connected to a computer with a small field programmable gate array (FPGA). The FPGA was not only used primarily for routing the information but also allowed some limited signal processing [3]. The USRP could realistically support about 3MHz of bandwidth due primarily to the performance restrictions of the USB interface. The second version was released in September 2008 and utilizes gigabit Ethernet to support 25 MHz of bandwidth [6]. The radio frequency performance of the USRP is limited and is more directed toward experimentation rather than matching any communications standard [6]. General purpose processors (GPPs) are less effective at physical layer processing but excel at the higher layers and are more accessible to the general software designer. Table 3.1 shows specifications of USRP and Table 3.2 shows the types of USRP [3].

Table 3.1. Specification of USRP

Description	Info
Year Release	2005
RF Bandwidth (MHz)	5
Frequency Range	2.3-2.9
Processing partition	Off-board
Processor architecture	GPP
Connectivity	USB

Table 3.2. Types of USRP

	USRP1	USRP2
Manufacturer	Ettus Research	Ettus Research
ADCs	64 MS/s 12 bit	100 MS/s 14 bit
DACs	128 MS/s 14 bit	400 MS/s 16 bit
Mixer	Programmable decimation and interpolation factors	
Maximum Bandwidth	16 MHz	50MHz
PC connection	USB 2.0 (32 MB/s half duplex)	Gigabit Ethernet (1000 Mbits/s)
RF range	DC- 5.2 GHz defined through daughter boards	

3.2 Setup Procedure

The whole process consists of setting up two Linux computers. In this research two dual core Linux machines were used. Dual core processor machines were chosen because of their ability to handle large amount of data. USRP boards are connected to the computers through USB 2.0. The first PC acts as the transmitter while the other PC as the receiver. Ubuntu Maverick 10.10 version was chosen as the development platform. Ubuntu is the most user friendly Linux operating system with ease of installation of GNU radio package. The steps involved in the installing of GNU Radio are explained in the next subtopic.

3.3 Installing GNU Radio

The systems (Maverick 10.10) used for the experiments in this research needed binary packages to be installed on them. This is because source packages are needed to be complied to install GNU Radio. Most of installations were done on terminal command window and the rest through Synaptic Package Manager. Shown below is the list of development tools required for compilation:

- g++ (GNU C++ compiler)
- g++ -4.4 (Installed w/g++)
- subversion (Subversion/svn)
- make (Already installed)
- autoconf (Autoconf)
- automake (Installed w/autoconf)
- libtool (Generic library support script)
- sdcc (Small Device C compiler)
- sdcc –libraries (Installed w/sdcc)
- guile-1.8-dev (Guile 1.8 Development files)
- ccache (Ccache (caches the output of C/C++ compilation)

3.4 Installing Synaptic Package Manager (Libraries Needed for Runtime and Compilation)

- python-dev (Header files)
- libfftw3 (FFTW library)
- libcppunit-dev (CppUnit 1.12.1)
- libusb-dev (Userspace USB programming library development files)
- wx-common (Wx Widgets Cross-platform C++ GUI toolkit)
- python-wxgtk2.8 (Widgets Cross-platform C++ GUI toolkit)
- python-numpy (Numpy)
- alsa-base (ALSA driver) (already installed)
- libasound2 (Already installed)

- libasound2-dev (Shared library for ALSA applications (development files))
- qt4-dev-tools (Qt 4 development tools)
- libqt4-dev (Installed w/qt4-dev-tools)
- pyqt4-dev-tools (Development tools for PyQt4)
- libsdl 1.2-dev (Simple DirectMedia Layer (SDL))
- libgs10-dev (GNU Scientific Library (GSL))

3.4.1 Installing Additional Synaptic Packages

- swig 1.3 (SWIG 1.3)
- swig (Installed w/swig 1.3)
- libqwt5-qt4-dev (Qwt library 5.2.0-1)
- libqwtplot3d-qt4 (QWT QwtPlot3D for qt4)
- python-scipy (Scientific tools for Python)
- python-matplotlib (Matplolib)
- python-tk (Already installed)
- doxygen (Doxygen)
- fort77 (Invoke f2c like a real compiler)
- spu-gcc (SPU cross-compiler (preprocessor and C compiler)
- git-core (Git)
- python-opengl (PyOpenGL)
- python-cheetah (Cheetah)
- python-lxml (Lxml)

3.4.2 Installing Boost

Boost is a very important aspect of GNU radio. It needs to be installed for GNU radio to function properly. Boost is a collection of all C++ libraries, a pre-required package for installation of GNU radio. Boost provides powerful extensions to C++ from many aspects such as algorithm implementation, math/numeric, input/output etc [2]. The following steps are followed to download boost:

- Download boost_1_37_0.tar.bz from the source
- Unpack it somewhere and cd into the resulting directory: cd boost_1_37_0. Use prefix BOOST_PREFIX=/opt/boost_1_37_0/include/boost-1_37
- Enter command ./configure-prefix=BOOST_PREFIX-with libraries=thread, date_time, program options to configure
- Enter the command make to compile the package
- Enter the command sudo make install to install the package

3.4.3 Installing GNU Radio

- #Install GNU Radio form git
- git clone: http//gnuradio.org/git/gnuradio.git
- cd gnuradio
- ./bootstrap
- ./configure
- make
- make check

- make install

3.4.4 Configuring USRP

- add group usrp
- usermod-G usrp-a smn0064
- echo 'ACTION==”add”, BUS==”usb”,SYSFS{idVendor}==”ffe”, SYSFS{idproduct}==”0002”,GROUP
- sudo chown root.root tmpfile
- mv tmpfile /etc/udev/rules.d/10-usrp.rules

3.4.5 Testing USRP Throughput

Once USRP is made available on Ubuntu, now we need to verify that GNU radio works with USRP. USRP configuration was verified using the command ls-1R /dev/bus/usb |grep usrp. The response crw-rw------1 root usrp 189,10 2011-10-11 17:26: 006 was confirmation that USRP was communicating with computer via USB.

Maximum throughput of USRP was verified by typing the following command on terminal window: cd~/usr/share/gnuradio/examples/usrp./usrp_benchmark_usb.py.

- Testing 2MB/sec……usb_throughput = 2M
- ntotal =1000000
- nright =999988
- runlength =999988
- errors = 12

- OK
- Output cut……
- Testing 32MB/sec……..usb_throughput =32 M
- ntotal =16000000
- nright =15998313
- runlength =15998313
- errors =1687
- OK
- Max USB/USRP throughput = 32MB/sec

3.4.6 FM Receiver

In GNU radio, we have functional block that is almost similar to the Simulink in Matlab called GNU Radio Companion (GRC) which can be used to transmit and receive the signal. GRC comes with all predefined blocks for signal sources, sinks and modulations. The steps given below show how to build the FM receiver using usrp_siggen.py signal generator:

i. cd ~/Desktop/gnuradio examples/usrp/python ./usrp_oscope.py –f 2.45 G

ii. cd ~/Desktop/gnuradio examples/usrp/python ./usrp_oscope.py -2.45 G –d 256 –R B

iii. cd ~/Desktop/gnuradio examples/usrp/python ./usrp_fft - 2.45 G – 16 –R

iv. cd ~/Desktop/gnuradio examples/usrp/python ./usrp_fm_tx_gui.py –f 2.45 G –T

v. cd ~/Desktop/gnuradio examples/usrp/python ./usrp_fm_rx_gui.py –f 2.45G –R B

The command in line *i* will generate a sine signal, and transmit it via daughterboard on slot A of the USRP to the motherboard, while the command on line *ii* was used to observe the signal in time domain, for instance, at center frequency of 2.45 G with decimation rate of 256 from the daughterboard on slot B of the USRP motherboard. The command on line *iii* the usrp_fft.py was used to observe the signal in frequency domain. Spectrum analyzer was used to observe the signal at center frequency of 2.45 GHz with 4 MHz bandwidth from the daughterboard on slot B of the USRP motherboard. The command on line *iv* was used to set up FM transmission and lastly the command on line *v* was used to show the maximum frequency transmitted which matched with frequency on daughterboard.

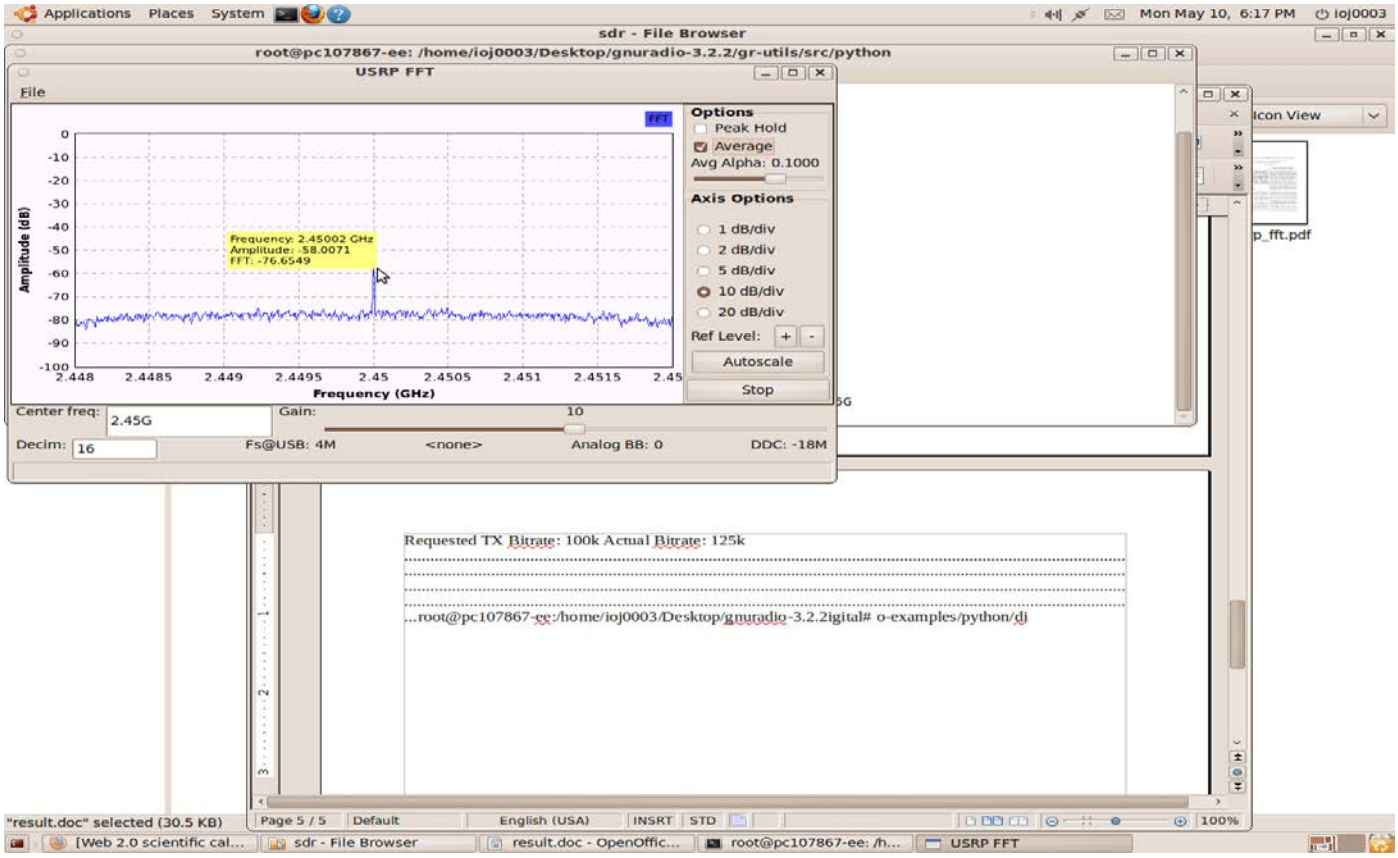

Figure 3.1. Maximum Frequency Transmitted

CHAPTER 4

EXPERIMENTS

4.1 Goal of Experiments

To increase the performance of the data transmission in SDR system, the packet error rate experiment is implemented. 172 data packets are transmitted using the BPSK /GMSK modulation and 500kbps is used as shown in Table 4.1. Numbers of pktno and n_right data packet are observed on the screen of receiver. The distance between transmitter and receiver is varied in order to measure the *pktno* and *n_right* data packets. Our objective is to transmit compressed data using the modules in GNU radio. We first read data file from the hard disk, and then sent it to the encoder implemented in C++. After encoding, the transmitter reads the data file and starts modulation. After modulation, the transmitter puts the packets into USRP and sends it to the receiver. We add modulation and demodulation in the transmitter and receiver in order to implement a practical wireless communication system. The primary purpose of this experiment is to evaluate the key performance metrics of the wireless channel using packet error rate which is the ratio of the number of corrupted packets over the number of transmitted packets.

When transferring the data from one USRP to another through loop-cable benchmark_tx.py and benchmark_rx.py, GNU radio package has been used as frame work. During this experiment both GMSK and BPSK were used and the results are compared.

4.2 Commands to Activate Receiver and Transmitter

The benchmark_tx transmits the data through USRP on the communication channel.

Benchmark_rx application is always in listening mode and just listens to the incoming data through USRP. These two Python codes take various command line arguments such as demodulation/modulation scheme, and bit rate. The default modulation scheme is GMSK at 500kbps.

The commands below show how the transmitter and receiver on USRP were configured to transmit and receive the data file. The receiver was set with the following command:

```
Cd ~/Desktop/03$./benchmark_rx.py –f 2.45G –w 0 –u 1-m GMSK - tiff_1  -r
500k
```

On hitting enter it goes into listening mode and displays the following:

```
>> gr_fir _fff: using   SSE

Requested RX Bitrate: 500k

Actual Bitrate: 500k

Warning: Failed to enable real-time scheduling

Ready to receive packets
```

On transmitter side the following command is used to set up the transmitter:

```
Cd ~/Desktop/03$./benchmark_tx.py –f 2.45G –w 0 –u 1-m bpsk – from file
tennis_2. tiff_1 -r 500k
```

This displays the following result

```
>> gr_fir_fff: using SSE

Requested TX Bitrate: 500k Actual Bitrate: 500k

Warning: failed to enable real-time scheduling
```

Immediately after starting the transmitter the receiver plays the role of the server to listen to the incoming packets and puts the incoming packets into the next block to restore the input data to the original data. At transmitter for instance, 172 data packets are assembled and transmitted at 100/150/300/500 kbps rates.

When the transmitter and receiver are closer, most of the received data packets show *True* reception on the terminal screen as shown in Table 4.1. The *pktno* in Table 4.1 shows the number of transmitted data packets. Data packets are dissembled and displayed as *n_right* data packets. The *n_right* data packet indicates actual reception data packets at the receiver. Sometimes, the reception of data packets was delayed and this is called propagation delay. This situation depends on the distance and sensitivity of the receiver to receive the data packets. As shown in Table 4.1 *n_right* value is lagging with *n_rcvd* as long as it is producing the *True* transmission. When the distance is too far, some of those data packets will be lost into free space. When the *False* transmission is detected at the receiver, *n_right* data packets are not counted as actual data packets. The *n_right* data packets show more lagging with the *n_rcvd* packet. This is because, the propagation delay will increase when the distance between the transmitter and receiver is very far. As per documentation, 2.45 GHz frequency was chosen. There is higher degree of packet loss and packet corruption in wireless media. The base code of benchmark_tx.py and benchmark_rx.py is modified to include protection against packet loss and packet corruption along with duplicate packets.

While the throughput is mainly limited by the speed of USB2 interface physical, manmade factors have significant effects on Packet Error Rate (PER) with GMSK and BPSK modulation scheme. In Tables 4.1 and 4.2 we see that the choice of modulation has a significant effect on packet error. The results are achieved by changing the modulation scheme while keeping bit rate constant. Using the formula 1-(n_right/pkno)*100, the result of packet errors is tabulated.

Table 4.1. Experiment showing the number of received packets using BPSK

```
ok = True pktno = 149 n_rcvd = 150 n_right = 150
ok = True pktno = 150 n_rcvd = 151 n_right = 151
ok = True pktno = 151 n_rcvd = 152 n_right = 152
ok = True pktno = 152 n_rcvd = 153 n_right = 153
ok = True pktno = 153 n_rcvd = 154 n_right = 154
ok = True pktno = 154 n_rcvd = 155 n_right = 155
ok = True pktno = 155 n_rcvd = 156 n_right = 156
ok = True pktno = 156 n_rcvd = 157 n_right = 157
ok = True pktno = 157 n_rcvd = 158 n_right = 158
ok = True pktno = 158 n_rcvd = 159 n_right = 159
ok = True pktno = 159 n_rcvd = 160 n_right = 160
ok = True pktno = 160 n_rcvd = 161 n_right = 161
ok = True pktno = 161 n_rcvd = 162 n_right = 162
ok = True pktno = 162 n_rcvd = 163 n_right = 163
ok = True pktno = 163 n_rcvd = 164 n_right = 164
ok = True pktno = 164 n_rcvd = 165 n_right = 165
ok = True pktno = 165 n_rcvd = 166 n_right = 166
ok = True pktno = 166 n_rcvd = 167 n_right = 167
ok = True pktno = 167 n_rcvd = 168 n_right = 168
ok = True pktno = 168 n_rcvd = 169 n_right = 169
ok = True pktno = 169 n_rcvd = 170 n_right = 170
ok = True pktno = 170 n_rcvd = 171 n_right = 171
ok = True pktno = 171 n_rcvd = 172 n_right = 172
```

Table 4.2. Experiment showing the number of received packets using GMSK

```
ok = True  pktno = 149 n_rcvd = 150 n_right = 150
ok = True  pktno = 150 n_rcvd = 151 n_right = 151
ok = True  pktno = 151 n_rcvd = 152 n_right = 152
ok = True  pktno = 152 n_rcvd = 153 n_right = 153
ok = True  pktno = 153 n_rcvd = 154 n_right = 154
ok = True  pktno = 154 n_rcvd = 155 n_right = 155
ok = True  pktno = 155 n_rcvd = 156 n_right = 156
ok = True  pktno = 156 n_rcvd = 157 n_right = 157
ok = True  pktno = 157 n_rcvd = 158 n_right = 158
ok = True  pktno = 158 n_rcvd = 159 n_right = 159
ok = True  pktno = 159 n_rcvd = 160 n_right = 160
ok = True  pktno = 160 n_rcvd = 161 n_right = 161
ok = True  pktno = 161 n_rcvd = 162 n_right = 162
ok = True  pktno = 162 n_rcvd = 163 n_right = 163
ok = False pktno = 163 n_rcvd = 164 n_right = 161
ok = False pktno = 164 n_rcvd = 165 n_right = 159
ok = False pktno = 165 n_rcvd = 166 n_right = 157
ok = False pktno = 166 n_rcvd = 167 n_right = 155
ok = False pktno = 167 n_rcvd = 168 n_right = 153
ok = False pktno = 168 n_rcvd = 169 n_right = 152
ok = False pktno = 169 n_rcvd = 170 n_right = 150
ok = False pktno = 170 n_rcvd = 171 n_right = 147
ok = False pktno = 171 n_rcvd = 172 n_right = 145
```

4.3 Analysis

As shown in Table 4.2 BPSK achieves much higher rate of performance in terms of packets received correctly. Although GMSK has advantage of reducing sideband power which in turns reduce out of band interference between signal carriers in adjacent frequency channels, the Gaussian filter increases the modulation memory in the system and causes intersymbol interference. This makes it more difficult to discriminate between different transmitted data values and requiring more complex channel equalization algorithms such as adaptive equalizer at the receiver. GMSK has high spectral efficiency, but it needs a higher power level than BPSK, in order to transmit the same amount of data reliably. This is why GMSK is performing poorly compared to BPSK.

CHAPTER 5

CONCLUSIONS

From the experiments we can conclude that we successfully implemented a video communication system using GNU radio and USRP. The system can be utilized to conduct a variety of experiments related to video communications.

APPENDIX A

SOURCE CODES FOR RECEIVER

A.1 Source for benchmark_rx.py

```
#! /usr/bin/env python
#
# Copyright 2005,2006,2007,2009 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#
```

```
from gnuradio import gr, gru, modulation_utils

from new radio import usrp

from gnu radio import eng_notation

from gnuradio.eng_option import eng_option

from optparse import OptionParser


import random

import struct

import sys

import string

import time


# from current dir

import usrp_receive_path

import bpsk

#import os

#print os.getpid()

#raw_input('Attach and press enter: ')


class my_top_block (gr.top_block):

   def __init__ (self, demodulator, rx_callback, options):
```

```
        gr.top_block.__init__ (self)

        # Set up receive path
        self.rxpath = usrp_receive_path. Usrp_receive_path (demodulator, rx_callback, options)

        self.connect (self.rxpath)

# ////////////////////////////////////////////////////////////////////////
#                             main
# ////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main ():
    global n_rcvd, n_right, dest_file

    n_rcvd = 0
    n_right = 0

    def rx_callback (ok, payload):
        global n_rcvd, n_right, dest_file
        (pktno,) = struct.unpack ('! H', payload [0:2])
```

```
        data = payload [2:]
    n_rcvd += 1
    if ok:
        n_right += 1

        if pktno > 0:                # Do not write first dummy packet (pktno #0)
            dest_file.write (data)
            dest_file.flush ()

        payload = struct.pack ('! H', n_rcvd & 0xffff)

        # Print Data
    print "ok = %5s pktno = %4d n_rcvd = %4d n_right = %4d" % (
        ok, pktno, n_rcvd, n_right)

demods = modulation_utils.type_1_demods ()

# Create Options Parser:
parser = Option Parser (option class=eng_option, conflict handler="resolve")
expert_grp = parser.add_option_group ("Expert")

parser.add_option ("-m", "--modulation", type="choice", choices=demods.keys (),
```

```
            default='gmsk',
            help="Select modulation from: %s [default=%%default]"
                % (', '.join (demods.keys ()),))

usrp_receive_path.add_options (parser, expert_grp)

for mod in demods.values ():
  mod.add_options (expert_grp)

(options, args) = parser.parse_args ()

if len (args)! = 0:
    parser.print_help (sys.stderr)
    sys.exit (1)

if options.rx_freq is None:
    sys.stderr.write ("You must specify -f FREQ or --freq FREQ\n")
    parser.print_help (sys.stderr)
    sys.exit (1)

dest_file = open ("received_picture.jpg", 'wb')
```

```
    # build the graph
    tb = my_top_block (demods [options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling ()
    if r! = gr.RT_OK:
        print "Warning: Failed to enable real-time scheduling."

    # start flow graph
    tb.start ()
    print "Ready to receive packets"

    # Stop rb flow graph
    raw input ()
    dest_file.close ()
    tb.stop ()
if __name__ == '__main__':
    try:
        Main ()
    except Keyboard Interrupt:
        pass
```

APPENDIX B

SOURCE CODES FOR THE TRANSMITTER

B.1 Source code for benchmark_tx.py

```
#! /usr/bin/env python
#
# Copyright 2005,2006,2007,2009 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
```

```
#


from gnuradio import gr, gru, modulation_utils

from gnuradio import usrp

from gnuradio import eng_notation

from gnuradio.eng_option import eng_option

from optparse import Option Parser


import random

import struct

import sys

import string

import time


# from current dir

import usrp_receive_path

import bpsk

#import os

#print os.getpid ()

#raw_input ('Attach and press enter: ')


class my_top_block (gr.top_block):
```

```
    def __init__ (self, demodulator, rx_callback, options):
        gr.top_block.__init__ (self)

        # Set up receive path
        self.rxpath = usrp_receive_path. Usrp_receive_path (demodulator, rx_callback, options)

        self.connect (self.rxpath)

# ////////////////////////////////////////////////////////////////////////
#                             main
# ////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main ():
    global n_rcvd, n_right, dest_file

    n_rcvd = 0
    n_right = 0

    def rx_callback (ok, payload):
        global n_rcvd, n_right, dest_file
```

```
    (pktno,) = struct.unpack ('! H', payload [0:2])
        data = payload [2:]
    n_rcvd += 1
    if ok:
        n_right += 1

        if pktno > 0:                # Do not write first dummy packet (pktno #0)
            dest_file.write (data)
            dest_file.flush ()

        payload = struct.pack ('! H', n_rcvd & 0xffff)

        # Print Data
    print "ok = %5s pktno = %4d n_rcvd = %4d n_right = %4d" % (
        ok, pktno, n_rcvd, n_right)

demods = modulation_utils.type_1_demods ()

# Create Options Parser:
parser = Option Parser (option_class=eng_option, conflict_handler="resolve")
expert_grp = parser.add_option_group ("Expert")
```

```
parser.add_option ("-m", "--modulation", type="choice", choices=demods.keys (),
                   default='gmsk',
                   help="Select modulation from: %s [default=%%default]"
                       % (', '.join (demods.keys ())),))

usrp_receive_path.add_options (parser, expert_grp)

for mod in demods.values ():
    mod.add_options (expert_grp)

(options, args) = parser.parse_args ()

if len (args)! = 0:
    parser.print_help (sys.stderr)
    sys.exit (1)

if options.rx_freq is None:
    sys.stderr.write ("You must specify -f FREQ or --freq FREQ\n")
    parser.print_help (sys.stderr)
    sys.exit (1)
```

```
    dest_file = open ("received_picture.jpg", 'wb')
    # build the graph
    tb = my_top_block (demods [options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling ()
    if r! = gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    # start flow graph
    tb.start ()
    print "Ready to receive packets"

    # Stop rb flow graph
    raw_input ()
    dest_file.close ()
    tb.stop ()

if __name__ == '__main__':
    try:
        Main ()
    except Keyboard Interrupt:
        pass
```

REFERENCE LIST

[1] J.G Proakis. *Digital Communications*. McGraw-Hill Professional, 4th ed., 2000.

[2] B.Slar. *Digital Communications*. *Fundamentals Principles and Applications*. Prentice Hall, 2nd ed., January 2001.

[3] Ettus Research LLC. URL http://www.ettus.com.

[4] Gnu Radio Organization. URL http://www.GnuRadio.org.

[5] Qin Chen. *Wireless H.264 Video TestBed Using Gnu Radio System Architecture and Setup*. Dept of Electrical & Computer Engineering. University of Florida, 2008.

[6] Ronan Farrell, Magdalena Sanchez, and Gerry Corley. *Software Defined Radio Demonstration An example and Future International Journal of Digital Multimedia Broadcasting 10, 3-10*. United States of America. Hindawi Publishing Corporation 2009.

[7] Naveen Manicka. *Gnu Radio Tested. Master Thesis* .University of Deleware, 2007.

[8] Alex Verduin. *Wireless Protocols analysis approach* .Master Thesis. Universiteit Van Amsterdam, 2008.

[9] Theodore S.Rappaport. *Wireless Communications Principles and Practice*. Person Education, 2nd ed., 2009.

[10] Arch C.Luther. *Principle of Digital and Audio and Video.* United States of America. Arctech House, 1999.

[11] *Python v2.7.1 documentation*. URL http://docs.python.org/index.html.October 2011.

[12] Eric Blossom. *How to Write Signal Processing Block* .GNU Radio, July 2006. URL http:// gnuradio.org/redmine/wiki/gnuradio.

[13] Wesley J.Chun. *Core Python Programming*. United States of America: Prentice Hall PTR.2001