COMMUNICATION SYSTEM OVER GNU RADIO AND OSSIE

Zizhi Cheng

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2011

APPROVED:

Shengli Fu, Major Professor
Hualiang Zhang, Committee Member
Kamesh Namuduri, Committee Member
Murali Varanasi, Chair of the Department of
Electrical Engineering
James D. Meernik, Acting Dean of the
Toulouse Graduate School

Cheng, Zizhi. Communication System over Gnu Radio and OSSIE. Master of Science (Electrical Engineering), December 2011, 172 pp., 4 tables, 55 illustrations, bibliography, 26 titles.

GNU Radio and OSSIE (Open-Source SCA (Software communication architecture) Implementation-Embedded) are two open source software toolkits for SDR (Software Defined Radio) developments, both of them can be supported by USRP (Universal Software Radio Peripheral).

In order to compare the performance of these two toolkits, an FM receiver over GNU Radio and OSSIE are tested in my thesis, test results are showed in Chapter 4 and Chapter 5. Results showed that the FM receiver over GNU Radio has better performance, due to the OSSIE is lack of synchronization between USRP interface and the modulation /demodulation components. Based on this, the SISO (Single Input Single Output) communication system over GNU Radio is designed to transmit and receive sound or image files between two USRP equipped with RFX2400 transceiver at 2.45G frequency.

Now, GNU Radio and OSSIE are widely used for academic research, but the future work based on GNU Radio and OSSIE can be designed to support MIMO, sensor network, and real time users etc.

ACKNOWLEDGEMENTS

I am heartily thankful to my major advisor, Dr.Shengli Fu, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. Also my appreciation is extended to all faculty and staff members of the electrical engineering department of University of North Texas.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Radio Basics

Radio systems have been used since the 19th century, a series of radio system products, telephony, audio, video, and data systems totally changed our life style and the medium of communication for obtaining and exchanging information. Based on these technologies, we officially entered a century of knowledge explosion.

Although radio systems are related to a great many of technic areas, the most basic secret of signal wireless transmission is electromagnetic radiation. The electromagnetic radiation is one kind of wave like energies when it travels through space. To put it simply, a system which can send modulated electromagnetic waves containing the message signal through free space with a certain frequency range is called a radio system, and the electromagnetic waves can be generated by an alternating current fed antenna. Electromagnetic radiation frequencies are called electromagnetic spectrum or radio spectrum. U.S. National Telecommunications and Information Administration (NTIA) office of spectrum management accounted radio spectrum in October 2003, a 3 KHz to 300 GHz radio allocations in United States is showed in Figure 1.1.[21]

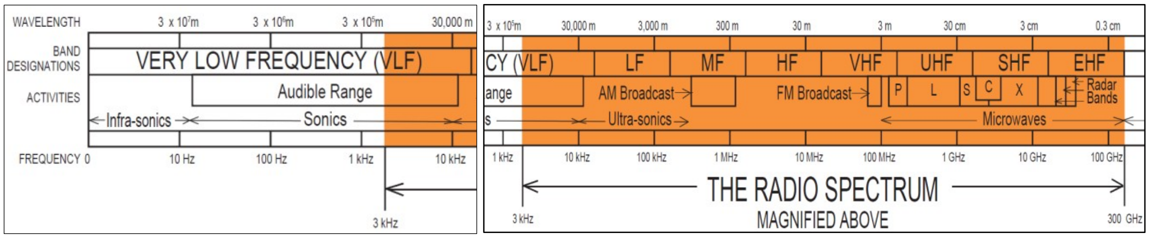


Figure 1.1 United States Frequency Allocations

United States uses medium wave frequency band for AM radio, its frequency range is from 520 KHz to 1610 KHz, and FM radio frequency range is from 87.8 MHz to 108 MHz. Within this range, channels 200 through 300 are covered, channels 200 to 220 (87.9 MHz to 91.9 MHz) are a reserved band for non-commercial educational (NCE) stations, and channels 221 to 300 (9201 MHz to 107.9 MHz) are for commercial and non-commercial stations.

In order to transmit the designed signal, modulation system is introduced. By using different modulation methods to change diversified properties (amplitude, frequency, pulse width, and phase,) of radiation waves, designed information signals can be carried. Generally, there are four types of modulation, digital modulation, analog modulation, digital baseband modulation, and pulse modulation. The digital modulation is normally used to transfer digital bit stream over an analog bandpass channel, and the analog modulation is used to transfer analog baseband or lowpass audio or TV signals over an analog bandpass channel. For analog signals, amplitude modulation (AM), frequency modulation (FM), and phase modulation (PM) are widely used, and AM and FM modulation are key technologies for radio systems. The fundamental digital modulation

methods include phase-shift keying (PSK), frequency-shift keying (FSK), amplitude-shift keying (ASK), quadrature amplitude modulation (QAM), etc. The PSK modulation is used for communication system project over GNU Radio and OSSIE (Open-Source SCA Implementation-Embedded) I introduce in Chapter 4 and Chapter 5.

Table 1.1 IEEE Wireless Standards

Name	Year	Frequency	Primary Use	Radio Tech
Wi-Fi (802.11 family)	802.11(1997)-802.11n(2009)	2.4, 5/3.7, 2.4/5GHz	Mobile internet	OFDM/MIMO
WiMAX (802.16 Family)	802.16(2001)-802.16m(2011)	2-11 GHz, 10-66 GHz	Mobile internet	MIMO-SOFDMA
LTE (3GPP Family)	1992-2011	1.4 MHz – 20MHz	General 4G	OFDMA/MIMO/SC-FDMA
iBurst (802.20 family)	2000	1.79GHz	Mobile Internet	HC-SDMA/TDD/MIMO
UMTS-TDD (UMTS/3GSM)	2002	1900MHz-2620MHz	Mobile internet	CDMA/TDD
WPAN (802.15.3)	2006	57GHz-64GHz	High-Rate wireless PAN	MB-OFDM
WRAN(802.22) Cognitive Radio	July 2011	6MHz	TV Broadcast Bands	OFDMA

In Table 1.1, IEEE 802.11 is the standards for WLAN (wireless local area network) computer communication in the 2.4, 3.6, and 5GHz frequency bands. In this family, 802.11 announced in 1997 was the first one, 802.11b was the first widely used standard followed by 802.11g and 802.11n, and other standards in this family are published as extensions or corrections for previous specifications. First WiFi (wireless fidelity)

technology was developed in 1997, it is a wireless technology for high speed internet and network connection. WiMAX (worldwide interoperability for microwave access) is implemented by IEEE 802.11 family, it can provide home and mobile internet access over a long distance. It is also widely used in different contries. LTE (long term ecolution) is a standard for maintaining 3GPP (3rd generation partnership project) projects and increasing wireless data networks capacity and speed. The iBurst (or HC-SDMA, high capacity spatial division multiple access) is a mobile broadband wireless access system developed by ArrayComm and be announced in April 2000. UMTS-TDD (universal mobile telecommunications system-time-division duplexing) is used to provide internet access. WPAN (wireless personal area network) is a personal area network for interconnecting devices around a person, which is announced in 2006. WRAN (wireless regional area network) is developed for using cognitive radio (CR) to share unused spectrums.

1.1.1. Quadrature Phase-Shift Keying (QPSK)

PSK modulation methods convey data by changing the carrier signals, and QPSK is one kind of PSK modulation methods. QPSK or 4-QAM uses four phases and four points to encode signals, two bits per symbol. The constellation diagram is shown in Figure 1.2[14].

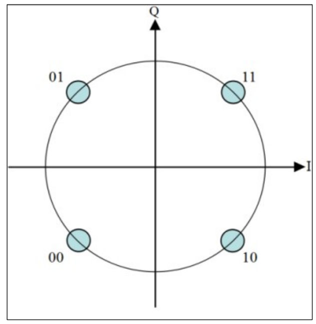


Figure 1.2 QPSK Constellation Diagram

For example, just like in Figure 1.2, it has four symbols (00, 01, 10, and 11) and four phases (π/4, 3π/4, 5π/4 and 7π/4), and the equation to write symbols in terms of sine and cosine waves is:

$$(t) = \sqrt{—}\cos\left(2\pi \quad t + (2n-1)\,—\right), \qquad n = 1,2,3,4. \quad (1)$$

The signal phase can be found by the following equations:

$$(t) = \sqrt{—}\cos\left(2\pi \quad\right) \qquad (2)$$

$$(t) = \sqrt{—}\sin\left(2\pi \quad\right) \qquad (3)$$

So four constellation points will be:

$$\left(\sqrt{\quad}, \sqrt{\quad}\right) \qquad (4)$$

Where (E_s = energy per symbol, f_c = carrier frequency, T_s = symbol duration)

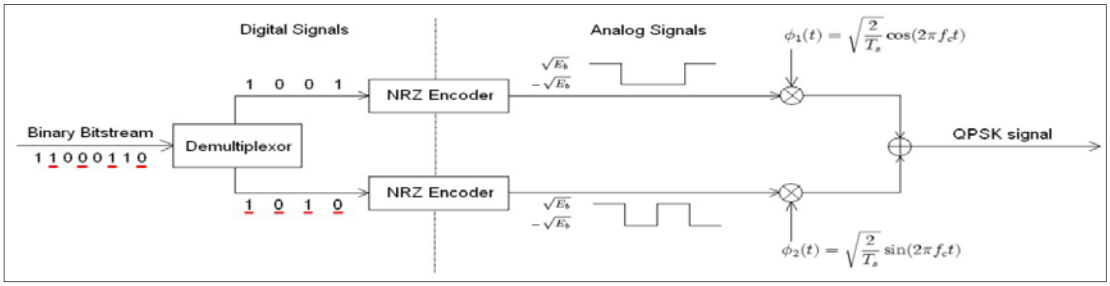


Figure 1.3 Conceptual Transmitter Structure for QPSK [14]

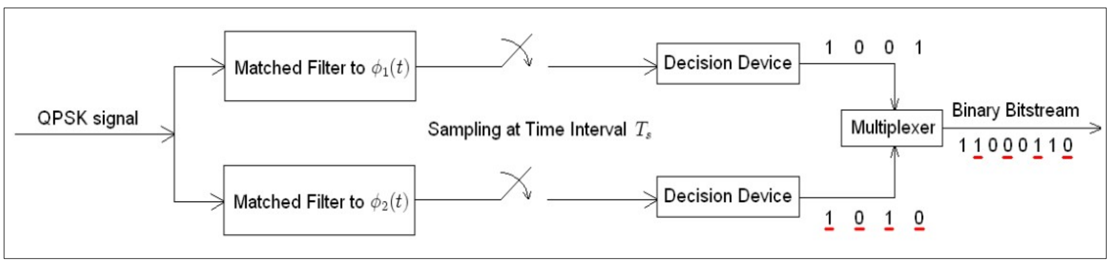


Figure 1.4 Receiver Structure for QPSK [14]

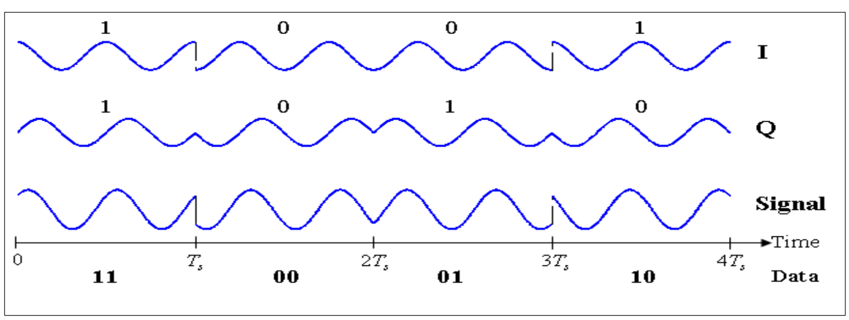


Figure 1.5 Timing Diagram for QPSK [14]

1.1.2. Differential Binary Phase-Shift Keying (DBPSK)

DBPSK differentially encode transmit signal compared with the previous signal, it can be represented as the following equation:

$$y[n] = y[n-1] \oplus x[n]$$

In this equation, x[n] is the bit input, y[n] is the current symbol, and y [n-1] is the previous symbol.

1.2 Software-Defined Radio

1.2.1 Advantages of Software-Defined Radio

By the exponential growth of ways and means of communication, people need to communicate by data, voice, video, broadcast, command, emergency response communications, and control communications, etc. The traditional technology is cost-effectively to modify radio devices, but SDR (software defined radio) technology brings a lot of flexibility and costs efficiency because it is a software-based approach to achieve variable communication system requirements. Software Defined Radio introduces software pieces instead of hardware components to treat signals in order to extract information.

Simlply speaking; any kind of device which can wirelessly transmit or receive signals within the radio frequency (RF) can be called radio, it exists in mobile phones, vehicles, and computers, etc. The traditional hardware-based radio devices can only be modified by physical intervention, higher production costs and less flexibility narrows its use in supporting multiple waveform standards. However; Software Defined Radio in

which some or all of the physical layer functions are software controlled allows multi-mode, multi-functional, or multi-band devices which makes it comparatively inexpensive and efficient. The approach is to handle different types of signals by loading the appropriate program.

Software defined radio's operates by modifiable software or firmware operating on programmable processing technologies, such as FPGA (field programmable gate arrays), DSP (digital signal processors), GPP (general purpose processors), SOC (programmable System on Chip), and other specific application programmable processors. The convenience gained by using these technologies is that it is easy to add new capabilities and wireless features to existing radio systems by programming without changing required new hardware expense.

1.2.2 The Structure of a Software Radio System

In the digital communication system, the fundamental diagram and basic elements can be illustrates by Figure 1.6. [17]

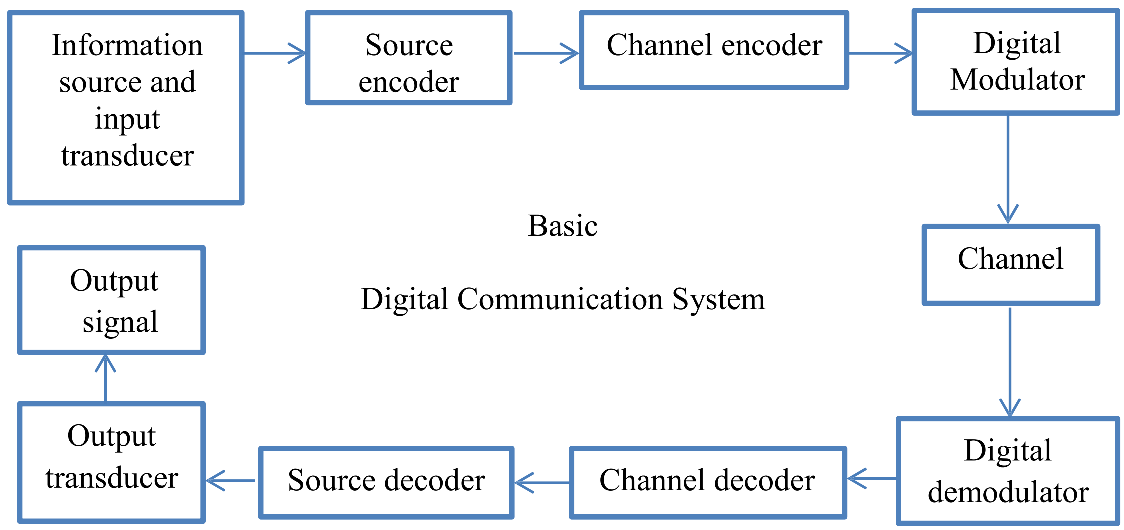


Figure 1.6 Basic Digital Communication System Diagram

The block diagram of a software radio system has a receive path and a transmit path, which are shown in Figure 1.2 and Figure 1.3, respectively.

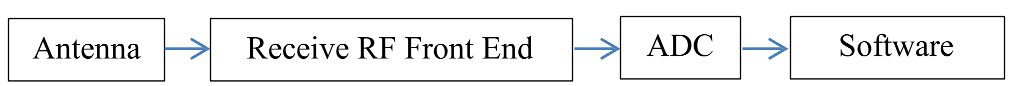


Figure 1.7 Software Radio Receive Path

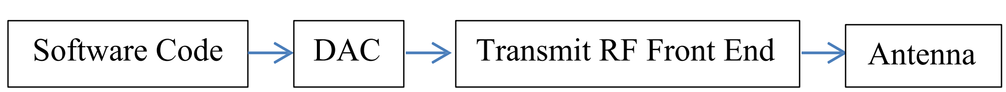


Figure 1.8 Software Radio Transmit Path

First, the analog-to-digital converter (ADC) and the digital-to-analog converter (DAC) are like the bridges between the continuous analog signals from physical world and discrete digital samples manipulated by software.

For the ADC, it has two primary characteristics: sampling rate that is the number of times per second of the analog signal measured by ADC; dynamic range that is the difference between the smallest and largest signal that can be distinguished, it's a function of the number of bits in the ADC's digital output and the design of the converter.

Secondly; the RF front-end can translate its input frequencies range to a lower frequencies range called intermediate frequency (IF). The reason we need to do this is because of the Nyquist sampling theorem.

From Nyquist sampling theory, we know that the ADC sampling frequency must be at least twice the maximum frequency of the signal so that the aliasing can be avoided. So there is a problem, if the ADC runs at 20 MHz, we cannot listen to broadcast FM radio at 92.1 MHz, however; with the RF front-end, we can translate signals occurring in 90-100 MHz range (RF) down to 0-10 MHz range (IF), then listening to the radio at 92.1 MHz with 20MHz ADC sampling rate is possible.

Compare with the basic digital communication system diagram, the communication system over GNU Radio can be designed as Figure 1.9.

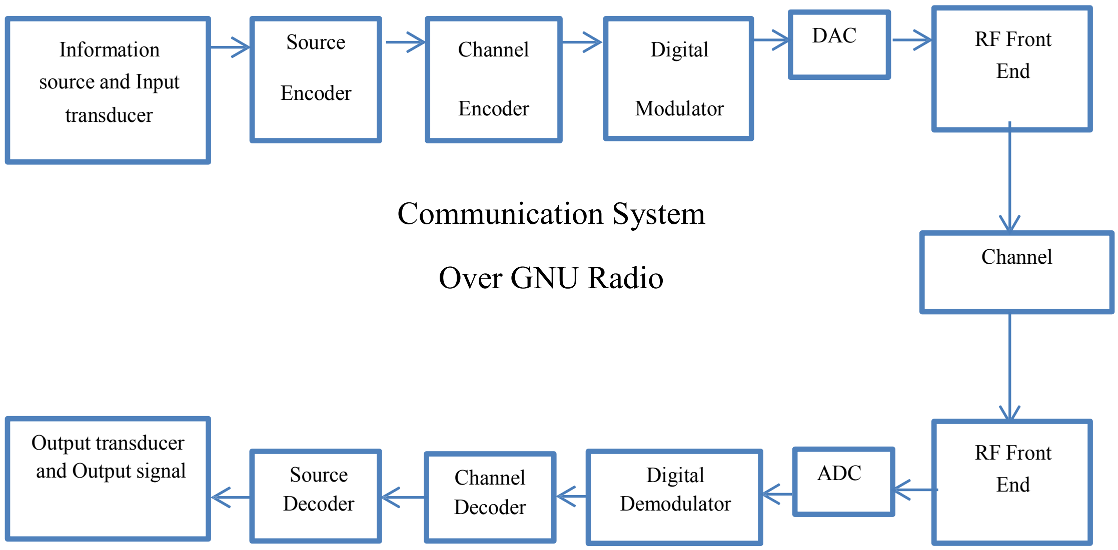


Figure 1.9 Communication System Over GNU Radio Diagram

1.3 Available Software Defined Radios

Because SDR is a convinent and inexpensive tool for communication system development, there are a great many groups and companies are working on it. It is very easy to find SDR hardware and toolkit in the market. Table 1.2 shows some available SDR hardware and software resources.

Table 1.2 Available Software Defined Radios

Name	Producer & Website	Frequency Range	Products and Price
USRP	Ettus Research LLC http://www.ettus.com/products	Up to 4 GHz	USRP™N200 ($1500-$1700), USRP™E100 ($1300-$1500), USRP™B100 ($650), USRP1 ($700), USRP2.
WARP	Rice University http://warp.rice.edu/ http://mangocomm.com/products	Up to 5 GHz	Hardware Products: WARP FPGA Board v1 ($3500), WARP FPGA Board v2 ($3500), Radio Board ($2000), Analog Board ($1500), and Clock Board ($750). Toolkits Products: MIMO Kit v1/v2 (v1: $8500; v2: $6500) SISO Kit v1/v2 ($6500)
FlexRadio	FlexRadio System http://www.flex-radio.com/	FLEX-1500: 0.01-54MHz FLEX-3000/5000: 0.01-65MHz	FLEX-1500 ($649.00), FLEX-3000 ($1699.00), FLEX-5000A ($2799.00), and VU5K
SDR-IQ	RFSPACE.Inc http://www.rfspace.com/RFSPACE/SDR-IQ.html	500Hz-30MHz	SDR-IQ set ($525)
DRB30	DRM Supporter http://www.nti-online.de/edirabox.htm	30kHz-30MHz	DRB 30 PC-controlled Shortwave Receiver (€ 299,00) USB Interface Adapter (€ 69,00)
QS1R	Software Radio Laboratory LLC. http://www.srl-llc.com/	Up to 130 MHz	QSiR Receiver ($799.99)

CHAPTER 2

GNU RADIO AND INSTALLATION

2.1 GNU Radio Hardware and Graphical User Interfaces

GNU Radio is a free software toolkit for SDR system and is a signal processing package which provides the processing blocks written in C++. By implementing Python program to build applications, graphical interfaces, and create a network or graph and connect signal processing blocks together, it can achieve a great many of communication system goals. It is released under GPL (GNU General Public License) version 3 license. GNU Radio is a convenient tool for developing wireless communication systems, and it is wildly used for both academic and commercial areas. Compared with traditional radio system design, GNU Radio is cheaper and more flexible. GNU Radio is implemented by USRP (Universal Software Radio Peripheral) mother-board and daughter-boards, and other hardware devices are needed for the GNU Radio communication projects, like Antenna, Attenuator, high-speed USB, RF cable, and USRP power wires.

2.1.1 Universal Software Radio Peripheral (USRP)

The Universal Software Radio Peripheral (USRP) offers a chance for engineers to be able to implement and design software radio system by a comparatively inexpensive hardware device and easy steps. The USRP plays like a digital baseband and IF (Intermediate frequency) section of a radio system, it enable general purpose computers

to be high bandwidth software radios. The host CPU will process all of the waveforms, like modulation and demodulation, and the USRP processes all of the high-speed general purpose operations like decimation and interpolation. Also a large community of developers and users created many hardware and software practical applications, so a lot of helpful information online can be found.

Besides the SISO-Proj and FM Receiver, the USRP can be implement many other applications such as RFID reader, Cellular base station, GPS receiver, Digital television decoder, Passive radar, and RF test equipment.

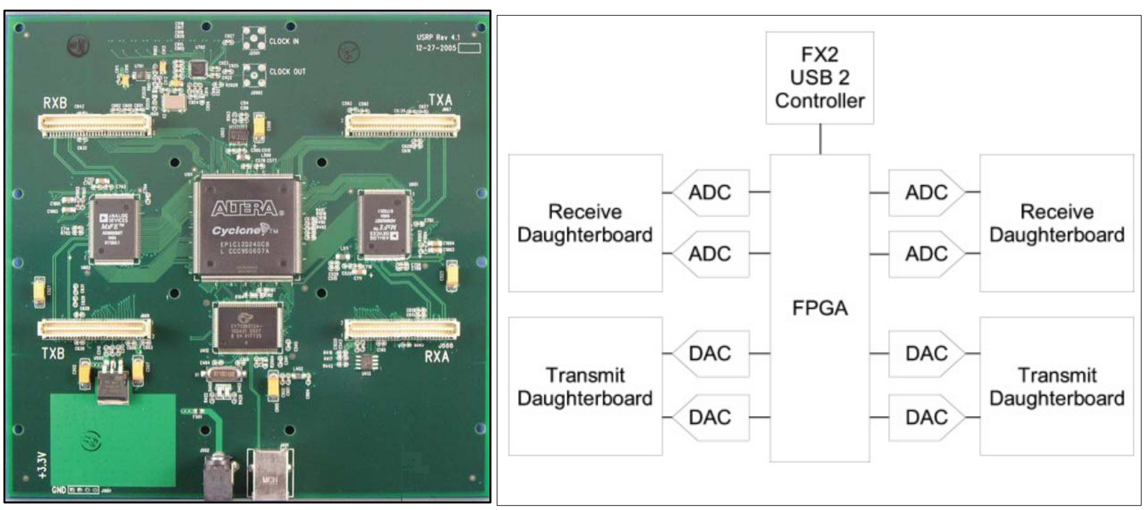


Figure 2.1 Universal Software Radio Peripheral (USRP) [9]

USRP family products and related products are offered by Ettus Research LLC, you can find and purchase them by the website http://www.ettus.com/order .

Table 2.1 USRP 1 Specifications [24]

Supported Operating System	Input	Output	Auxiliary I/Q
Linux Mac OS X Windows XP, Windows 2000, FreeBSD, NetBSD	Number of input channels: 4 (or 2 I/Q pairs) Sample rate: 64 Ms/s Resolution: 12 bits SFDR: 85 dB	Number of output channels: 4 (or 2 I/Q pairs) Sample rate: 128 Ms/s Resolution: 14 bits SFDR: 83 dB	High-speed digital I/O: 64 bits Analog input: 8 channels Analog output: 8 channels

And the features of USRP can be found in Appendix B. Because USRP mother board is equipped with USB 2.0, make sure your PC is also equipped with USB 2.0. There are 4 high-speed 12-bit ADCs, which can bandpass-sample signals up to 150 MHz. GNU Radio daughter-boards can enable USRP Mother-board to be a complete RF transceiver system and all available Daughter-boards are listed here,

- BasicRX: Receiver for use with external RF hardware
- BasicTX: 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 receiver
- WBX: 50 MHz to 2.2 GHz transceiver
- RFX400: 400-500 MHz transceiver
- RFX900: 750-1050 MHz transceiver
- RFX1200: 1150-1450 MHz transceiver
- RFX1800: 1.5-2.1 GHz transceiver
- RFX2400: 2.3-2.9 GHz transceiver
- XCVR2450: 2.4 GHz and 5 GHz dualband transceiver

More details about available Daughter-boards and other devices are showed in Appendix B. Equipping with all required hardware device, you will be able to design a two-way, high-bandwidth communications in variable frequency bands.

2.1.2 GNU Radio Graphical interfaces

GNU Radio applications graphical interfaces are built in Python, the wxPython is suggested to use in order to maximize cross-platform portability. And GNU Radio provides about 100 signal processing blocks implemented in C++, these signal processing blocks process infinite streams of short, float and complex data through their input and output ports. In the graph, some blocks have only input ports or output ports, and they are designed as data sources and sink. Python plays an important role in GNU radio, and it

creates signal flow graphs or a network to connect these C++ signal processing blocks together.

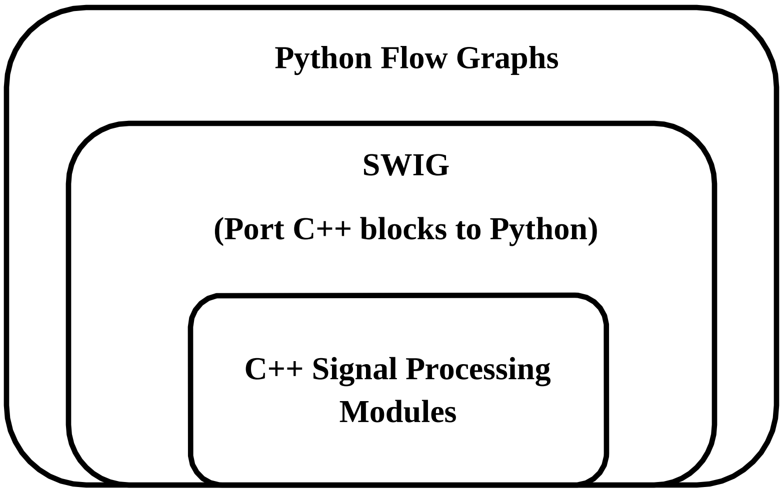


Figure 2.2 GNU Radio Software Architecture

2.2 GNU Radio Installation Guide for Ubuntu 10.04[1]

2.2.1 Install Dependencies

On the Ubuntu 10.04 desktop interface, open a terminal from “Applications”; enter the installation commands by the following instruction.

$ sudo apt-get -y install libfontconfig1-dev libxrender-dev libpulse-dev swig g++ automake autoconf libtool python-dev libfftw3-dev libcppunit-dev libboost-all-dev libusb-dev fort77 sdcc sdcc-libraries libsdl1.2-dev python-wxgtk2.8 git-core guile-1.8-dev libqt4-dev python-numpy ccache python-opengl libgsl0-dev python-cheetah python-

lxml doxygen qt4-dev-tools libqwt5-qt4-dev libqwtplot3d-qt4-dev pyqt4-dev-tools python-qwt5-qt4

2.2.2 Install GNU Radio (USRP is not required in this step)

$ git clone http://gnuradio.org/git/gnuradio.git   or

$ git clone git://gnuradio.org/gnuradio.git   (# download GNU Radio git master tree)

$ cd gnuradio   (# guide into gnuradio file direction)

$ ./bootstrap   (# bootstrap)

$ ./configure   (# configure)

$ make   (# make)

$ make check   (#)

$ sudo make install

In order to check the installation is successfully, try gnuradio-companion by typing

$ gnuradio-companion

If GNU Radio is installed successfully, the companion window will appear.

2.2.3 Configuring USRP support

In order to handle USRP by USB, you need to take the following script. The reason for this setting is that Ubuntu handles hot-plug devices by using udev. By this script,

Ubuntu will be configured to understand the next response, like if/when it detects the USRP on the USB.

sudo addgroup usrp

sudo usermod -G usrp -a <YOUR_USERNAME>

echo 'ACTION=="add", BUS=="usb", SYSFS{idVendor}=="fffe",

SYSFS{idProduct}=="0002", GROUP:="usrp", MODE:="0660"' > tmpfile

sudo chown root.root tmpfile

sudo mv tmpfile /etc/udev/rules.d/10-usrp.rules

where the <YOUR_USERNAME> is your user name, for example; if your name is "student", you can type

sudo usermod -G usrp -a student

For the "udev" to reload the rules, take the next script and reboot your computer.

sudo udevadm control --reload-rules

or

sudo /etc/init.d/udev stop

sudo /etc/init.d/udev start

or

sudo killall -HUP udevd

Now you can plug in your USRP to the computer by the USB and check if your USRP is being recognized or not.

Typing the following line in your terminal:

ls -lR /dev/bus/usb | grep usrp

If your USRP is being recognized, there will be a line shows up in your terminal like this:

crw-rw---- 1 root usrp 189, 514 Mar 24 09:46 003

and one line represent one USRP, if you have multiple USRPs plugged in, you will read multiple lines.

2.2.4 Installing in a Custom Directory

The GNU Radio will be installed to /usr/local direction generically, and if you want to install it to the location you choose, the solution is to configure the built to use the custom prefix, which will make the GNU Radio be installed to /opt/gnuradio

./boostrap

./configure --prefix=/opt/gnuradio

This next step is necessary because the custom installation directories can be able to find the python scripts, libraries, the headers, etc. First of all; open the .bashrc file by typing "gedit.bashrc" in the terminal and add the following script into this file.

GNU Radio installation

export PATH=$PATH:/opt/gnuradio/bin

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/gnuradio/lib

export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/opt/gnuradio/lib/pkgconfig

export PYTHONPATH=$PYTHONPATH:/opt/gnuradio/lib/python2.6/site-packages

And restart your computer.

The last step of this installation is

1) Make a copy from the current ld.so.conf file and save it in a temp folder:

cp /etc/ld.so.conf /tmp/ld.so.conf

2) Add /usr/local/lib path to it:

echo /usr/local/lib >> /tmp/ld.so.conf

3) Delete the original ld.so.conf file and put the modified file instead:

sudo mv /tmp/ld.so.conf /etc/ld.so.conf

4) Do ldconfig:

sudo ldconfig[3]

The GNU Radio is installed completely.

CHAPTER 3

OPEN-SOURCE SCA IMPLEMENTATION-EMBEDDED (OSSIE) AND INSTALLATION

3.1 Open-Source SCA Implementation-Embedded (OSSIE) Introduction

Based on the goals of supporting research and education in wireless communication by using Software Defined Radio (SDR), OSSIE is designed as an open source SDR development effort based at Wireless@VirginiaTech. This embedded OSSIE project provides an open source tools and framework to test and develop the SDR platforms and waveforms behavior. It is an excellent software to illustrate and analysis SDR's concepts and trade-offs for commercial and educational purposes.

The following links provide online help on OSSIE,

- http://ossie.wireless.vt.edu/trac/wiki, the online wiki
- listserv@listserv.vt.edu, the mailing lists
- http://ossie.wireless.vt.edu/trac/newticket, the Trac bug tracking tickets
- The IRC channel (you can start from IRCTutorial),
- http://ossie.wireless.vt.edu. OSSIE Home Page

There are two approaches for OSSIE installation:

1. Through Linux system with the source code;
2. Through OSSIE VMWare image on which has OSSIE pre-installed.

3.2 Install OSSIE from Source under Ubuntu 10.04[4]

3.2.1 Installing Dependencies on Ubuntu 10.04

Open a terminal, and install the following packages:

```
$ sudo aptitude install gcc build - essential
$ sudo aptitude -y install omniorb4 libomniorb4-dev omniidl4-python \
omniorb4-nameserver python-omniorb2 libgtk2.0-dev freeglut3-dev \
python-wxgtk2 .8 python -wxversion python-wxtools python-numpy \
python-numpy-ext python-numpy-dev python-profiler g++ automake \
libtool subversion python-dev fftw3-dev libcppunit-dev libboost-dev sdcc \
libusb-dev libasound2-dev libsdl1.2-dev guile-1.8 libqt3-mt-dev swig \
python-profiler automake1.9 python2 .6-dev sdcc-libraries guile-1.8-dev \
libqt4-dev ccache python-opengl libgsl0-dev python-lxml \
doxygen qt4-dev-tools libqwt5-qt4-dev libqwtplot3d-qt4-dev \
libboost-filesystem-dev libbo
```

3.2.1 Configure omniORB

```
$ cd omniORB-4.1.4
$ cp sample.cfg /etc/
$ mv /etc/sample.cfg /etc/omniORB.cfg
```

Depending on the omniORB dependency version, the omniORB file may be

/etc/omniORB.cfg or /etc/omniORB4.cfg. If you cannot find the file, just copy sample.cfg from omniORB-4.1.4 directory and rename it.

Open the file as root:

$ gksu nautilus

$ ./omniORB.cfg

And find the following line:

InitRef = NameService = corbaname :: my. host . name

Uncomment the line by deleting the pound or hash character "#" and change it to:

InitRef = NameService = corbaname ::127.0.0.1

3.2.2 Installing Portions of GNU Radio

OSSIE uses a small subset of GNU Radio to communicate with and configure the USRP 1, then follow the next step.

If you are using Ubuntu, and would like to use GNU Radio v3.1 and enter:

$ sudo aptitude install libusrp0 libusrp –dev

If you would like to use GNU Radio v3.2, and do not have Fedora 11 installed, then you must install from source:

$ wget ftp ://ftp.gnu.org/gnu/gnuradio/gnuradio -3.2.2. tar.gz

$ tar-xvf gnuradio-3.2.2.tar.gz

$ cd gnuradio-3.2.2/

```
$ ./configure
$ make
$ sudo make install
```

At this point, GNU Radio and its dependencies have been installed. Now setup the proper permissions for the USRP. As root, create a group which will have access to the USRP:

```
# /usr/sbin/groupadd usrp
```

Add users to the group which need access to the USRP:

```
# /usr/sbin/usermod-G usrp -a <USERNAME>
```

Now that users will have access to the USRP, read and write access to the device must be created. As root, create the file /etc/udev/rules.d/10-usrp.rules in a text editor:

```
$ vi /etc/udev /rules.d/10 -usrp.rules
```

Add the following text to the following text to the file:

```
ACTION =="add", BUS =="usb", SYSFS{idVendor}=="fffe", \
SYSFS{idProduct}=="0002", GROUP:="usrp", MODE:="0660"
```

The text above is displayed on two lines due to the constraints on page size; however the text must appear on a single line, without the backslash, in the file for the access to the

USRP to work properly. You may also add the following comment lines to the file for future reference:

rule to grant read/write access on USRP to group named usrp.

to use, install this file in /etc/udev/rules.d/ as

10 -usrp.rules

The USRP interface has now been created. As an optional test, connect the USRP to the computer and run the following command:

$ ls –lR /dev/bus/usb

The users root and usrp should now be listed under the user groups.

3.2.3 Install OSSIE

Download the latest tarball from http://ossie.wireless.vt.edu/download/tarballs/0.8.2/

Unpack ossie-0.8.2.tar.gz by typing:

$ wget http://ossie.wireless.vt.edu/download/tarballs/0.8.2/ossie-0.8.2.tar.gz

$ tar-xvf ossie-0.8.2.tar.gz

By default, the installation directory of the OSSIE platform is /sdr. In order to install new source code and binaries into this directory without root permissions, you need to create and change the ownership of /sdr.

sudo mkdir /sdr

chown -R username.username /sdr

Where username is your user name.

3.2.4 Using Autoconf and Updating System Libraries

$ cd ossie-0.8.2

$ ./configure --prefix=/sdr --libdir=/usr/local/lib/ \

--includedir=/usr/local/include/ --with-boost --with-boost-filesystem

$ make

$ sudo make install

Once the OSSIE libraries are installed, they need to be linked. As root edit the file /etc/ld.so.conf, adding the line

/usr/local/lib

Now run:

/sbin/ldconfig

3.2.5 Installation of OSSIE Eclipse Feature

Installation of the OSSIE Eclipse Feature (OEF) requires the installation of OSSIE, Java, and Eclipse. First, we start from Java installation.

Open /etc/apt/sources.list in an editor and add the following lines to the end of the file:

deb http://archive.canonical.com/ubuntu lucid partner

deb-src http://archive.canonical.com/ubuntu lucid partner

In a terminal, enter the following lines:

$ sudo apt-get update

$ sudo apt-get install sun-java6-jdk

Install the Eclipse IDE for Java Developers. Go to the Eclipse Download Center and download an Eclipse distribution for your platform. Eclipse is distributed as a tarball archive that you can unpack to location of your choice. Pick a location that is appropriate for your platform and simply unpack the contents. There is no self-installer, just unpack the distribution. Do not install Eclipse in a directory that has spaces anywhere in its full path name.

Installation of OEF is followed by next two lines.

$ cd /path/to/eclipse

$ ./eclipse

After Eclipse starts, on the toolbar select Help, Install New Software. In the new window, select the \Work with" textbox and enter the URL:

http://ossie.wireless.vt.edu/eclipse/, and select Add. Give a name, e.g., \OEF". The window will then add the URL, OSSIE, and OSSIE Waveform Developer Feature to the list of available software. Place a check in the box next to OSSIE Waveform Developer Feature and click "Next". Eclipse will show a window with more details, select Finish to complete the installation process. Allow Eclipse to restart when it prompts to do so. After it restarts, the OSSIE Eclipse Feature will be installed. Select the OSSIE perspective within Eclipse. On the toolbar, select Window, Open Perspective, Other. In the new window, select OSSIE which will then open the OSSIE perspective. On the toolbar, select File, New, OSSIE Waveform, or OSSIE Component to start developing. These same instructions used for installing OEF can be used later to update it to newer versions.

3.3 Using the OSSIE VMware Player Image

Except installing OSSIE from the source, you also can use the OSSIE VMware Player Image to run the OSSIE waveform. The username of OSSIE is "ossie", the password is "wireless", and the username and password may be asked to login to the OSSIE system or run the waveform.

3.3.1 Installing VMware Player

Download VMware Player at: http://www.vmware.com/download/player/, you need to log in your VMware account and download the version you need. If you are new, just simply create your own account by your email address and a new password. Then you can choose the version of VMware Player you want to install. Make this file executable

by right click this file → choose properties → Permissions → check mark make the file executable. This step is very important; otherwise you will be unable to install this bundle file. If you download VMware Player 3.1.4 for 64-bit Linux, you will find a file named VMware-Player-3.1.4-385536.x86_64.bundle where your download path is, type the following command into the terminal:

```
$ sudo bash VMware-Player-3.1.4-385536.x86_64.bundle
```

The VMware Player will be installed. After the installation, open VMware Player from Ubuntu main manual Applications -> System tools. For detailed instructions, you can read VMware Player User Manual from:

http://www.vmware.com/pdf/vmware_player200.pdf.

Download OSSIE VMware image at:

http://ossie.wireless.vt.edu/download/vmware/OSSIE-0.8.2-Ubuntu-10.04.2-VM.rar, extract this file to another direction, the purpose of this step is to save the original zip file as a backup in case you will need to start form a fresh install later, also it could save time and bandwidth. Once you completely installed VMware Player and unzipped OSSIE VMware image, boot OSSIE VMware image up by VMware Player, a window of update requirement will pops out, choose ok and start updating system. If you install everything correct, the main window should look as similar as Figure 3.1.

Also you can download VMware Images from

http://ossie.wireless.vt.edu/download/vmware/OSSIE-0.8.2-GR-3.3.0-GC-3.0-Ubuntu-

10.04.2.rar, and this version has more OSSIE open source SCA software radio frameworks/components and OSSIE Labs documents installed which I recommend. Just one more step after this you will be capable to run the demonstration waveforms and practice the labs.

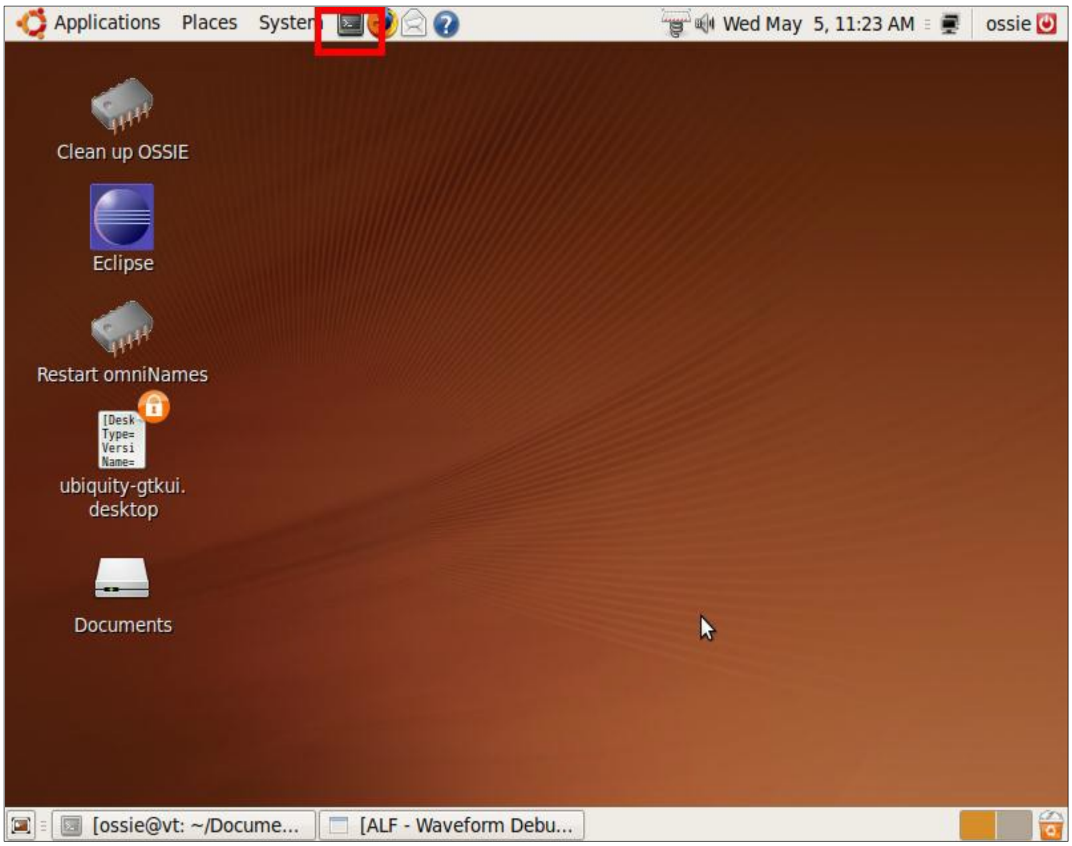


Figure 3.1 OSSIE Desktop Screen

3.3.2 Creating omniNames.sh

The naming service will automatically start when you boot the system if you install Omni ORB from RPM; otherwise you will need to start naming service manually by creating a file named omniNames.sh.

Boot OSSIE VMware image by VMware Player, open a terminal from the Applications → Accessories → terminal, and type:

```
$ sudo vi omniNames.sh
```

Press i from keyboard to be able to insert the code, and type the following code line by line into the document:

```
#!/bin/sh
killall omniNames
rm /sdr/logs/omninames*
omniNames –start –logdir /sdr/logs &
```

Press ESC and type ":wq" to save the file.

In order to set an appropriate permission for the file, type:

```
$ chmod 755 omniNames.sh
```

Where the first digit 7 enables the owner to read/write and execute permissions on this file; the second digit 5 enables read/write permissions for the group user; the third digit 5 enables read/write permissions for the guest user.

The next step is to make a directory by typing:

$ mkdir /sdr/logs

Then copy omniNames.sh file to /usr/local/bin by typing:

$ sudo cp omniNames.sh /usr/local/bin

Well, now you can check the naming service by typing the following line into the terminal,

$ sudo omniNames.sh

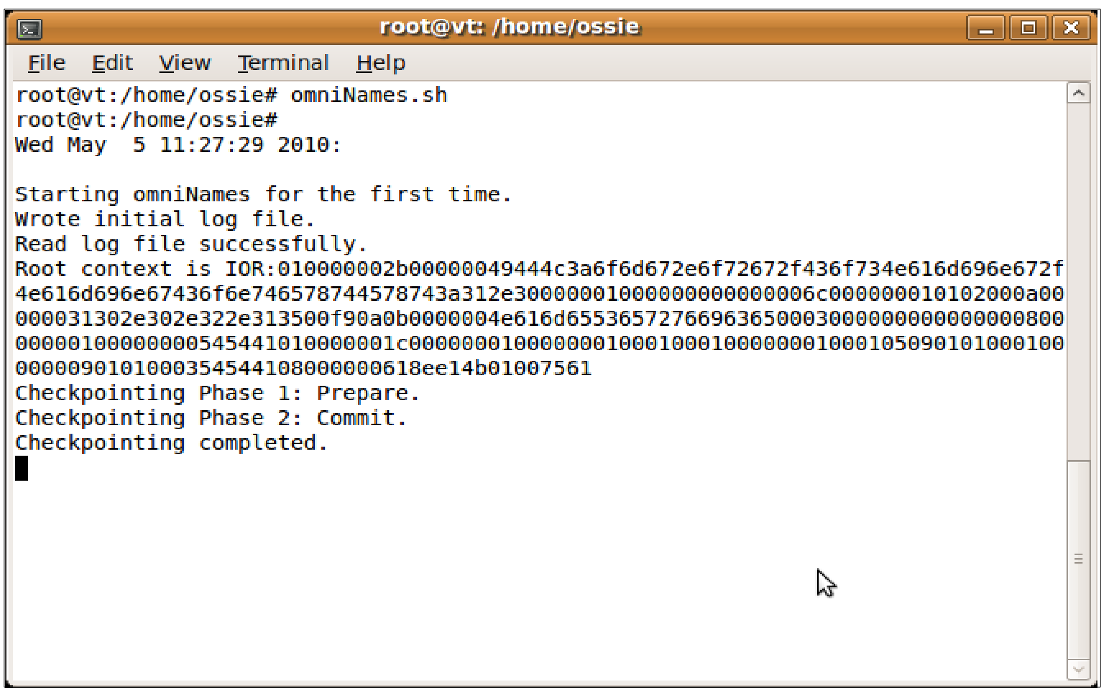


Figure 3.2 CORBA Naming Service Terminal

If the naming service is running correct, you should be able see the terminal looks like Figure 3.2.

Every time you run the omniNames.sh file, the system will start the CORBA naming service. The naming service is a standard service for CORBA applications, and it allows you to associate abstract names with CORBA objects and allows you to find those objects by the corresponding names. For this reason; start the naming service before every time you run a waveform is very important, otherwise your computer will be unable to find objects by the corresponding names, and failure to run the waveform.

CHAPTER 4

DEVELOPMENT OF SISO SYSTEM OVER GNU RADIO

4.1 GNU Radio Examples

4.1.1 Hello World Example

The Hello world example is a small test for GNU Radio and Python installation. The result is to show “Hello World” in a dialog window.

Open a terminal; enter the python interface by typing: *$ python*

When the python interface shows up, type the Hello World code

```
>>> from gnuradio import gr          # Importing gr modules from GNU radio library
>>> import wx                        # Importing wxPython GUI
>>> app=wx.PySimpleApp()             # Define a new wxPySimpleApp
>>> frame=wx.Frame(None,-1,"Hello World")   # Define a new wxFrame
>>> frame.Show(1)                    # Show this frame
True
>>> app.MainLoop()                   # Start this application’s mainloop
```

Then press “Enter”, and you will see a Hello World like Figure 4.1

Figure 4.1 Hello World

4.1.2 Dial Tone Example

After the installation of GNU Radio, a fold named “gnuradio-examples” is created. In this fold, “dial_tone.py” can be found in a sub-fold at “python” → “audio”, record the full path of this file.

The dial tone example, it generates two sine waves (350 Hz and 440 Hz) and send them to the sound card, one on the left channel, another one on the right channel. After run this example, you will hear a sound like the US dial tone from the sound card. Here is the dial tone example source code:

```
#!/usr/bin/env python from
gnuradio import gr from
gnuradio import audio
from gnuradio.eng_option import eng_option
from optparse import OptionParser
```

```
class my_top_block(gr.top_block):
 def __init__(self): gr.top_block.__
 init__(self)
  parser = OptionParser(option_class=eng_option)
  parser.add_option("-O", "--audio-output", type="string", default="",
          help="pcm output device name.  E.g., hw:0,0 or /dev/dsp")
  parser.add_option("-r", "--sample-rate", type="eng_float", default=48000,
          help="set sample rate to RATE (48000)")
  (options, args) = parser.parse_args ()
  if len(args) != 0:
    parser.print_help()
    raise SystemExit, 1
  sample_rate = int(options.sample_rate)
  ampl = 0.1
  src0 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 350, ampl)
  src1 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 440, ampl)
  dst = audio.sink (sample_rate, options.audio_output)
  self.connect (src0, (dst, 0))
  self.connect (src1, (dst, 1))
  if __name__ == '__main__':
 try:
  my_top_block().run()
```

```
  except KeyboardInterrupt:
    pass
```

First, we use “gr_sig_source_f” to generate two sine waves, scr0 and scr1, the “f” in “gr_sig_source_f” means these two signals are floating type. “audi.sink” is a receiver, and it can send the received signals to the sound card. “self.connect (src0, (dst, 0))” and “self.connect (src1, (dst, 1))” connect two sine wave source signals to the audio.sink. Close the terminal, the dial tone will be turned off.

Open a terminal, direct to the dial_tone.py file by the full path, then you will be able to run dial_tone.py.

For example:

```
$ cd Desktop/gnuradio-3.3.0/gnuradio-examples/python/audio
```

And run the dial tone example:

```
$ ./dial_tone.py
```

Then you will hear a US dial tone from the speaker.

To generate two sine waves in different frequencies, find the dial_tone.py and double click it, when a window pops up, press “Display”.

Find the following two lines in the file,

```
src0 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 350, ampl)
src1 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 440, ampl)
```

Change 350 and 440 to the frequencies you want to hear, and save the file and run it again.

If you change them to higher frequencies, you will hear a sharper sound, and lower tone with lower frequencies.

Stop this program by typing Ctrl-z or close the terminal.

4.2 SISO Communication System over GNU Radio

This project shows a single input single output communication system. In this project, signals will be sent from one USRP to another one through the air by DBPSK modulation.

4.2.1 Hardware and Software Requirements

Two PCs has Ubuntu 10.04 installed

Two USRP Mother-boards

Two USRP Daughter-boards: RFX2400: 2.3-2.9 GHz transceiver

Two Antennas: HG240RD-SM

Two SMA-Bulkheads and other connection device.

A speaker

4.2.2 System Design Setup and Running

Insert RFX2400 daughter-board to Mother-board RXA/TXA side. Connect antenna to the RFX2400 TX/RX port by using a SMA-Bulkhead, also plug in the USRP power wire and USB wire to the USRP and connect them to the power supply and computer one which is the transmit side. Repeat this for computer two which is the receive side and GNU Radio hardware on another side of the system. Also connect a speaker to the receiver side computer two. Before running the project, make sure the USRP is connected successfully. To make sure the USRP connection, open a terminal and type: *$ usrp_probe* If the connection is correct, a widow like Figure 4.2 will appear

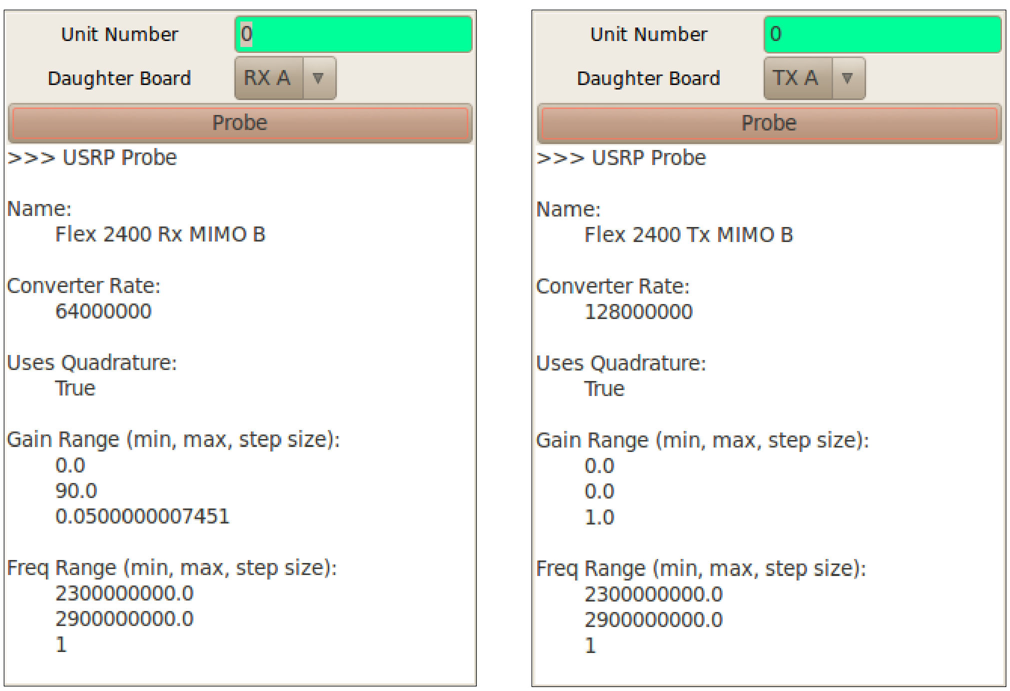


Figure 4.2 USRP Probe

This window shows the information of the daughter-board RX at A side, expand the small triangle at the right side of “Daughter Board” in the window, then can see RX at B side and TX at A or B side if you have a daughter-board connected to the mother-board side B. This window shows that the RFX2400 daughter-board is used and the frequency range is from 2.3GHz to 2.9GHz. In this project, a 2.45GHz transmit and receive frequency is designed. From the GNU Radio package, two files named benchmark_rx.py and benchmark_tx.py can be used to receive and transmit the signals. But other related files are required to run the benchmark_rx/tx.py files.

In order to run the benchmark files, a SISO-Proj folder that consists of the following files is needed on both transmit and receive side computers:

-- benchmark_rx.py

-- benchmark_tx.py

-- dbpsk.py / dbpsk.pyc

-- generic_usrp.py / generic_usrp.pyc

-- pick_bitrate.py / pick_bitrate.pyc

-- psk.py / psk.pyc

-- receive_path.py / receive_path.pyc

-- transmit_path.py / transmit_path.pyc

-- usrp_options.py / usrp_options.pyc

-- usrp_receive_path.py / usrp_receive_path.pyc

-- usrp_transmit_path.py / usrp_transmit_path.pyc

-- Files to transmit

Checking the missing file by type the following line into the terminal to run the file:

$ ./benchmark_rx.py

The ImportError line will show you the missing file. All files can be found in GNU Radio library.

Gstreamer Player is needed to play the received mp3 music in this project, to install it, open a terminal enter the following lines to install gstreamer tools

$ sudo apt-get install gstreamer-tools

After install the Gstreamer Player, the received mp3 file can be played in another terminal when the file is being sent. Or by writing python code for the Gstreamer Player and add it to benchmark_rx.py, the mp3 file will automatically be played when the file is completely received.

The Gstreamer Player Python Code is:

```
def on_tag(bus, msg):
        taglist = msg.parse_tag()
        print 'on_tag:'
        for key in taglist.keys():
            print '\t%s = %s' % (key, taglist[key])
    #our stream to play
    music_stream_uri = 'file:///home/zizhi/Desktop/SISO-Proj/received.dat'
    player = gst.element_factory_make("playbin", "player")
```

```
    player.set_property('uri', music_stream_uri)
    #start playing
    player.set_state(gst.STATE_PLAYING)
    #listen for tags on the message bus; tag event might be called more than once
    bus = player.get_bus()
    bus.enable_sync_message_emission()
    bus.add_signal_watch()
    bus.connect('message::tag', on_tag)
    #wait and let the music play
    raw_input('Press enter to stop playing...')
```

Adding this coding into benchmark_rx.py between

```
    # Stop rb flow graph
    raw_input()
    dest_file.close()
    tb.stop()
      -----------------------------          Adding Code              --------------------------
if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

After connection all of hardware and installed all needed software, on the computer two, open a terminal, direct to the SISO-Proj folder, and enter

$ ./benchmark_rx.py -f 2.45G -w 0 -u 1 -m dbpsk -r 500k -R A

When the following lines appear in the terminal, GNU Radio is ready to receive signals.

```
>>> gr_fir_ccf: using SSE

Requested RX Bitrate: 500k

Actual Bitrate: 500k

Warning: Failed to enable realtime scheduling.

Ready to receive packets
```

Open a terminal on computer one, and navigate to the SISO-Proj folder and enter

$ ./benchmark_tx.py -f 2.45G -w 0 -u 1 -m dbpsk --from-file music.mp3 -r 500k

When the following lines appear in the terminal, the GNU Radio starts to transmit signals

```
>>> gr_fir_ccf: using SSE

Requested TX Bitrate: 500k Actual Bitrate: 500k

Warning: failed to enable realtime scheduling

.........................................................................................................................

..................................................................................
```

Meanwhile, open another terminal on the computer two, enter

```
$ gst-launch playbin num-buffers=10000 uri=file:/home/zizhi/Desktop/SISO-Proj/received.dat
```

Then the mp3 music will be heard from the speaker.

4.3 FM Receiver over GNU Radio

Compare with the traditional FM receivers that are built entirely using hardware fabricated in a plant, the open source signal processing software package GNU Radio is a more convenient and inexpensive tool to build the FM Receiver. All that's needed is a PC running on the Ubuntu 10.04 and UHD properly intalled version of GNU Radio, then it is possible to build a FM receiver.

4.2.1 Build and Run FM Receiver

Connect BasicRx daughter-board to the RXA port on the mother-board, the BasicTx daughter-board to the TXA port on the mother-board. Also connect a HG2407RD-SM Antenna to the RX-A port on BasicRX board. Plug in the power wire and connect it to the power supply, connect USRP to the computer by USB. The hardware connection is complete.

Open file usrp_wfm_rcv.py, find line

```
usrp_decim = 200
```

Change 200 to 150, this value is related to the sample rate, only the appropriate sample rate will make the received signal distortion free. Here set *usrp_decim = 150* will make the sample to be 42666 and it is close to the sound card requirement 44100 sample rate.

```
adc_rate = self.u.adc_rate()          # 64 MS/s
usrp_decim = 150
self.u.set_decim_rate(usrp_decim)
usrp_rate = adc_rate / usrp_decim
```

Open a terminal, direct to the path of usrp_wfm_rcv.py, enter

```
$ ./usrp_wfm_rcv.py --freq 200M
```

The terminal will show the following information:

```
Using RX d'board A: Basic Rx
>>> gr_fir_ccf :  using SSE
>>> gr_fir_fff :  using SSE
audio_alsa_sink[hw: 0,0] : unable to support sampling rate 42666
   card requested 44100 instead.
FYI: No Powermate or Contour Knob found
aUaUaUaUaUaUaUaUaUaUaUaUaUaUaUaUaUaUaUaUaUaUaU
```

And a FM Receiver window will pops up, which looks like Figure 4.3.

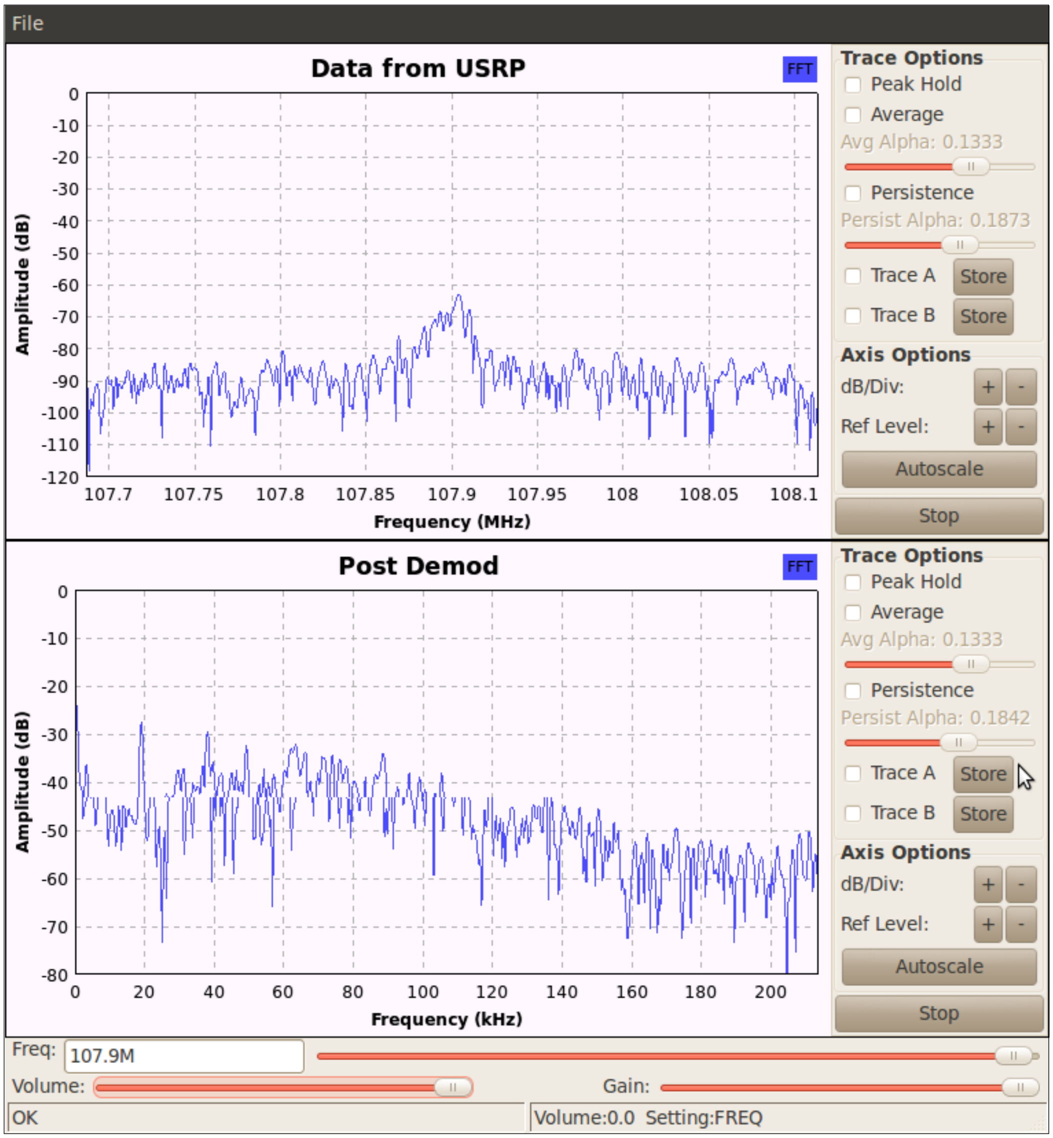


Figure 4.3 FM Receiver Window Screen

Change the frequency by adjusting the bar by the side of "Freq" to receive signals from

CHAPTER 5

DEMONSTRATION AND COMMUNICATION SYSTEMS OVER OSSIE

5.1 OSSIE Waveform Demonstration

For the purpose of understanding OSSIE, OSSIE developers offer OSSIE Labs for OSSIE 0.6.1 version up to OSSIE 0.8.2 version, and the GNU Radio Labs at http://ossie.wireless.vt.edu/download/labs/. Since OSSIE 0.8.2 the lasted version, it is used here, and a combination OSSIE 0.8.2 Lab will be explained in detail.

OSSIE Labs were developed with the assistance of Philip Balister, Jacob DePriest, Jeff Reed, and Max Robert of the Mobile and Portable Radio Research Group, Wireless@Virginia Tech (http://www.mprg.org, http://wireless.vt.edu) and uses the OSSIE open source SCA software radio framework and components (http://ossie.wireless.vt.edu).

5.1.1 QPSK Demo Introduction [5]

This demonstration uses OSSIE 0.8.2 and the OSSIE Eclipse Feature (OEF), running on the VMware image using Ubuntu 10.04. It shows how to build and operate QPSK transmission demonstration by using components TxDemo, ChannelDemo, and RxDemo, it also consists of all basic knowledge about assembling and running waveforms. The system diagram is showed in Figure 5.1.

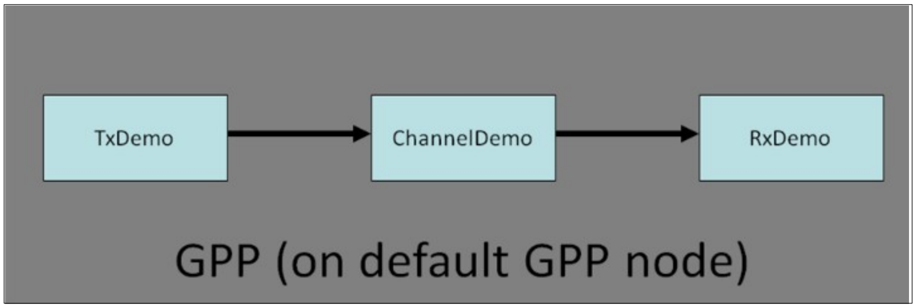


Figure 5.1 QPSK Demo Diagram

5.1.1.1 Getting Started

Boot your computer on which has Ubuntu 10.04, navigate to “Applications” -> “system tools” -> “VMware image” -> “run a virtual machine”. If require to login, then use username “ossie” and password “wireless” to login.

Open a terminal by navigating to “Applications” -> “Accessories” -> “Terminal”, or click on the terminal shortcut in the toolbar at the top of the desktop. And type the following line into the terminal to stare the naming service:

$ sudo omniNames

If the naming service starts correct, the terminal should look like Figure 3.2, and keep this window open.

Or you can test if the CORBA naming service is already running by type in a terminal:

$ ps –e | grep omniNames

And if it is running, the omniNames should be listed.

5.1.1.2 Create and Build the Waveform

Double click Eclipse on the desktop to open Eclipse. When the Eclipse interface pops up, navigate to “File” -> “New” and select “Other…” -> expand “OSSIE” folder -> “OSSIE Waveform” and click “Next”.

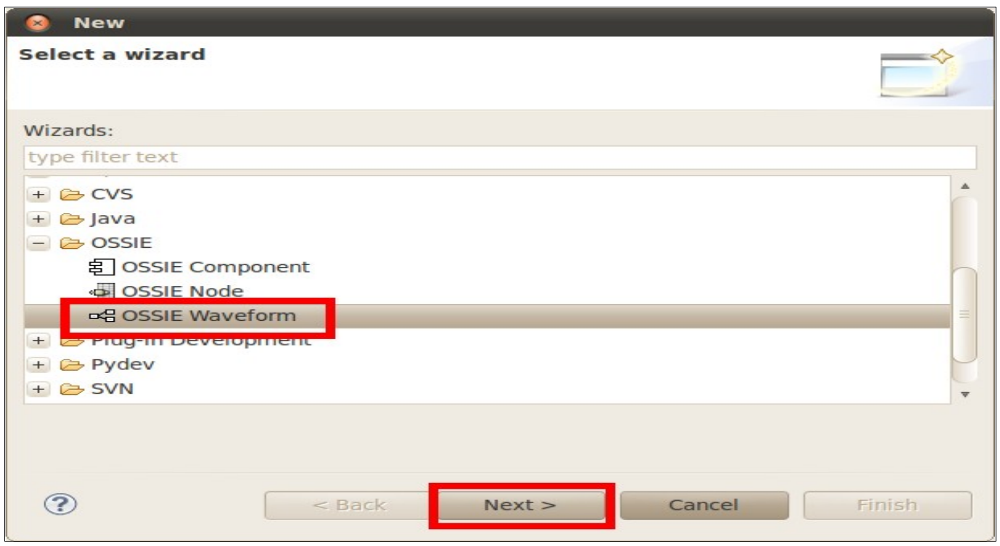


Figure 5.2 Create OSSIE Waveform Screen

A wizard will appear and here a waveform project name has to be given, and click “finish”. An editor will pop up and it look like Figure 5.3.

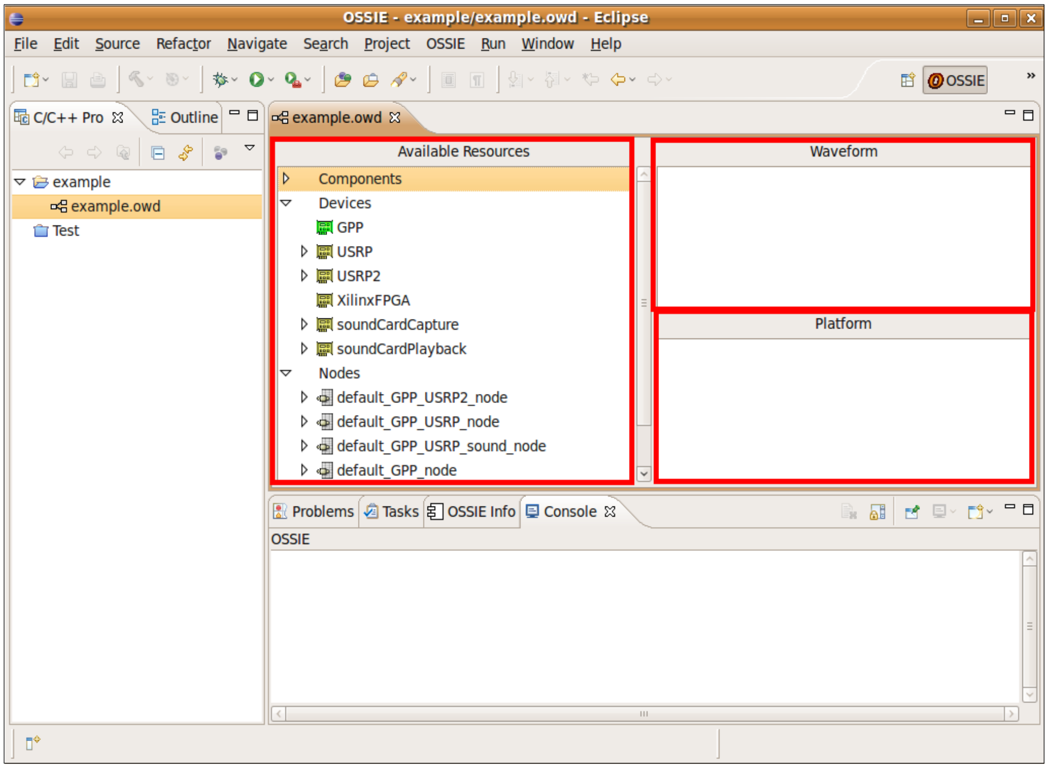


Figure 5.3 OSSIE-Eclipse Waveform Window

This editor has 3 panels:

(a) Available Resources: This panel contains all of the available components, devices, and nodes installed on the system. And all details of those components and devices can be found in 4.2.6, or online source at

http://ossie.wireless.vt.edu/trac/wiki/WaveformDevelopmentGuide

(b) Waveform: This panel is where to drag components to add them to the waveform.

(c) Platform: This panel is where to drag nodes and devices to add them to the platform.

Click on the triangle on the left side of “Components” to expand the list of available components. Adding components to the design by double left clicking on the component you need, or clicking on the component for the design and hold down the left mouse button with the mouse then drag the component to the waveform panel. Adding nodes to the Platform by double clicking on the node and it will appear in the Platform panel.

In this Lab, the communication system consists of three components: transmitter demo, channel demo, and receiver demo. So double click on TxDemo, ChannelDemo, and RxDemo, and find the default_GPP_node under the “Nodes” in the Available Resources panel, double click on it. Expand the default_GPP_node in the Platform panel to show the GPP1 device assigned to it. To deploy TxDemo to the GPP1, drag TxDemo from the waveform panel onto GPP1 in the platform panel. Repeat the same process for ChannelDemo and RxDemo. Now, the editor should be like Figure 5.4.

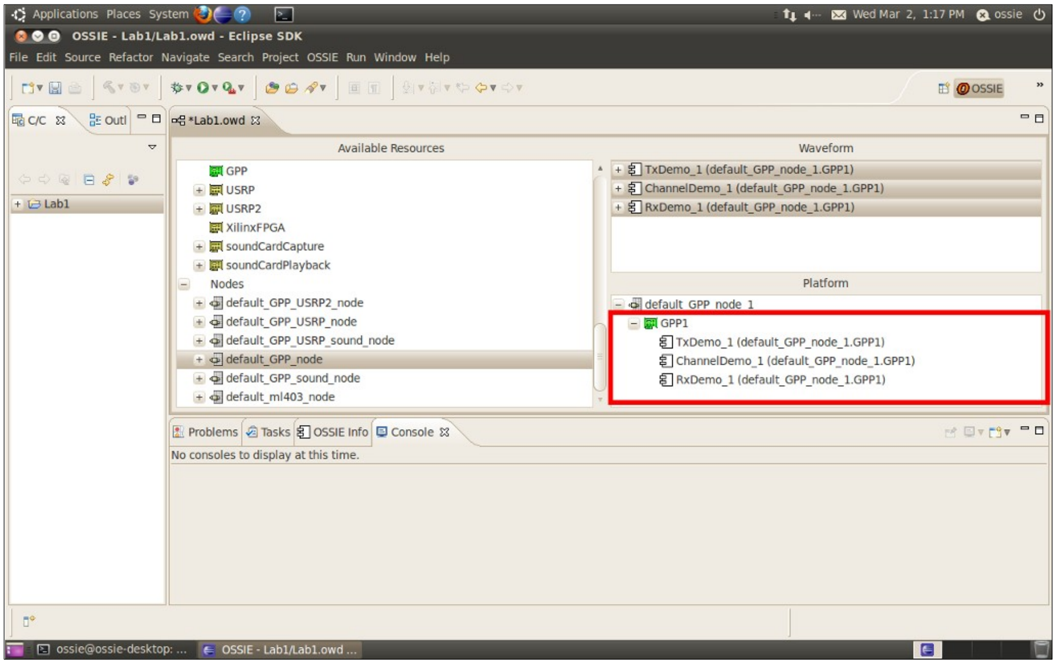


Figure 5.4 QPSK Demo-Eclipse Window

To make sure those three components are deployed correctly, expand the GPP1 device in the platform panel and you should see all of them. Also the three components in the waveform panel should all now show “default_GPP_node_1.GPP1”.

Now, assign the transmitter component to be the task of Assembly Controller, simply right click on the TxDemo component in the waveform penal, and select “Set Assembly Controller”. (Note: when you run your waveform, it starts from the start () function which is called assembly controller. The assembly controller has the ability to start and stop other components.) The TxDemo in Waveform panel should look like Figure 5.5.

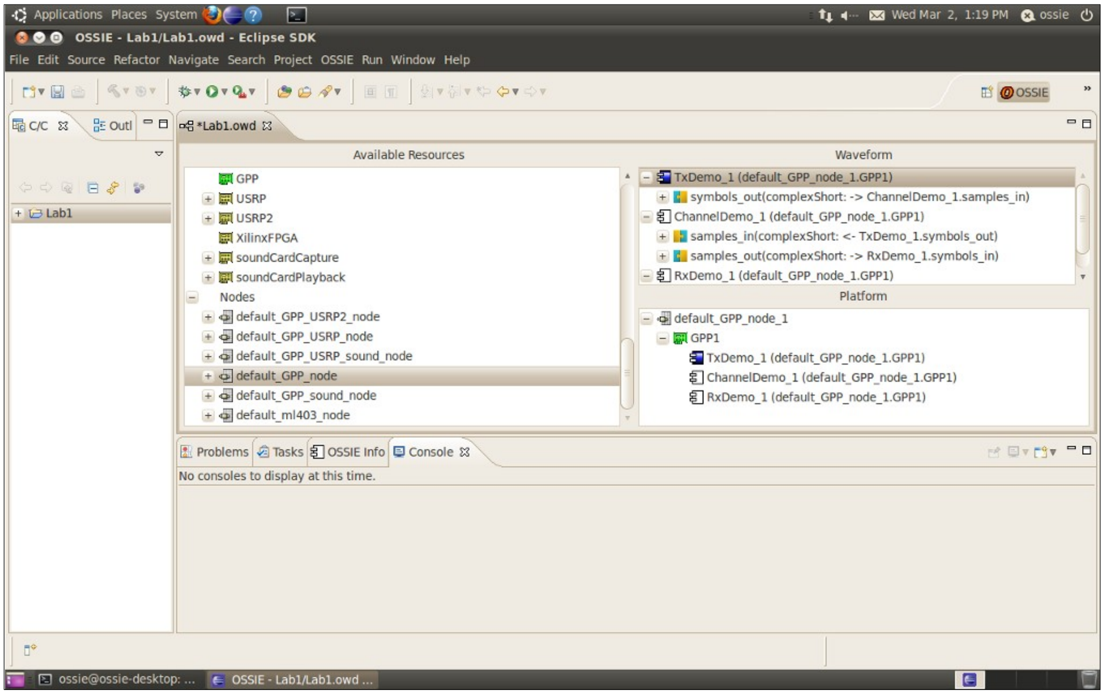


Figure 5.5 QPSK Demo-Eclipse-Assembly Controller Setting

Expand TxDemo, ChannelDemo, and RxDemo and connect them. To do this, two connections have to be made. First of all; drag “symbols_out” port on the TxDemo to the “samples_in” port on the ChannelDemo. Secondly; drag “symbols_out” port on ChannelDemo to the “sympols_in” port on the RxDemo. (Note: If the connections are complete, one port must be an output (“uses”, orange puzzle piece) and other must be an input (“provides”, blue puzzle piece)).

Save this waveform: select “File” -> “Save” or press “CTRL” and “S”. OEF will generate corresponding xml files and deploy them to the path /sdr/dom/waveforms/<your project name>.

5.1.1.3 Run the Waveform

Check if the CORBA naming service is running by typing

$ ps –e | grep omniNames

And the omniNames should be listed.

Or restart the naming service by typing

$ omniNames.sh

And the terminal should be like Figure 3.2.

Next, run the Node Booter in the terminal or in Eclipse.

(a) Running the Node Booter in terminal

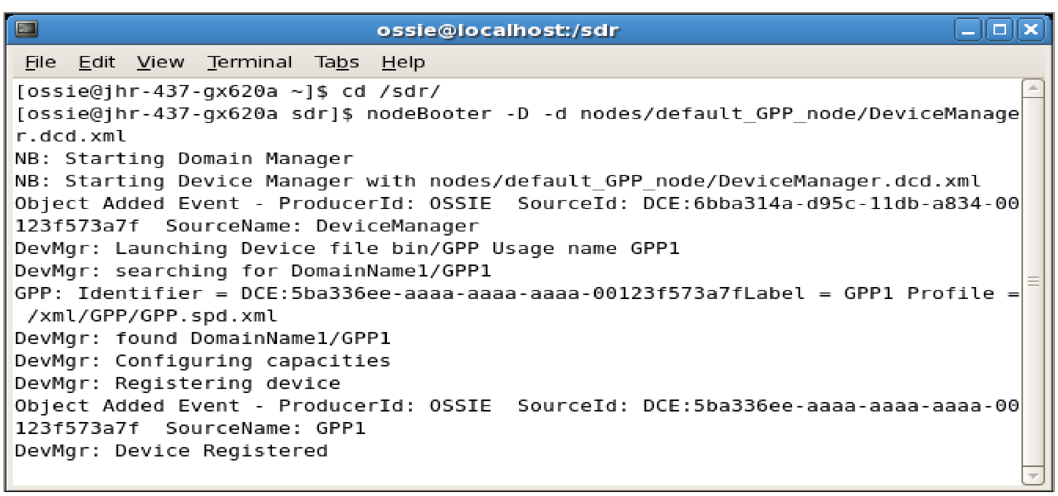


Figure 5.6 Running NodeBooter in Terminal

Open a terminal and type

$ cd /sdr/

$ nodeBooter -D -d dev/ nodes/default_GPP_node/DeviceManager.dcd.xml

If other nodes are needed for other waveform projects, you will just need to change the node name. (For example: $ nodeBooter -D -d dev/nodes /<your node name>/DeviceManager .dcd.xml)

Open a new terminal and load your waveform by typing:

$ sudo C_wavLoader

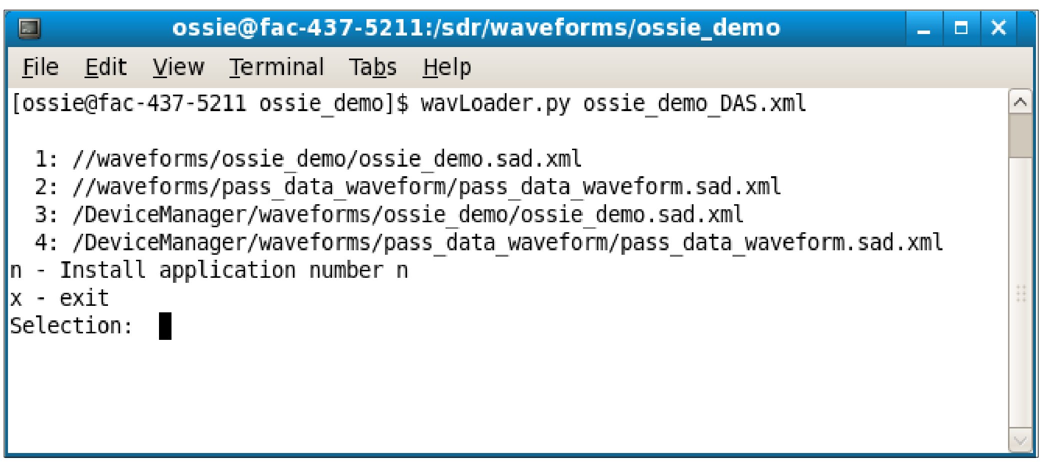


Figure 5.7 Loading Waveform Terminal-1

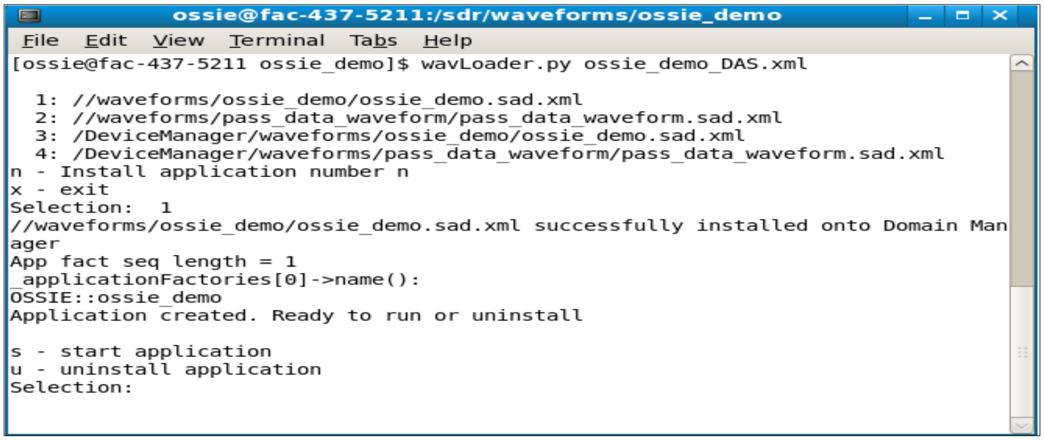


Figure 5.8 Starting Waveform Terminal

Your waveform is loaded now, and it is a file named <your project name>.sad.xml. Type the number of your waveform in the list after “Selection” and enter “s”.

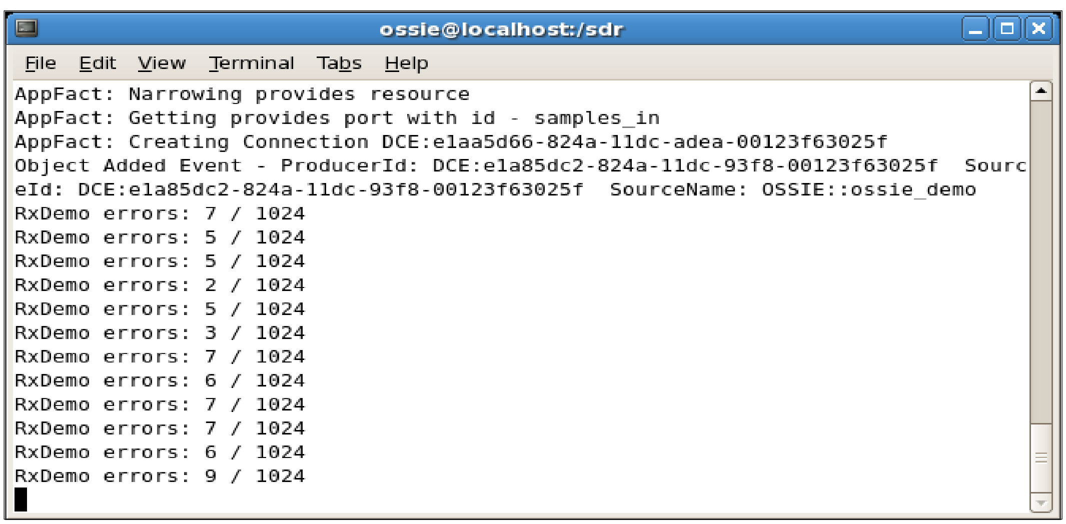


Figure 5.9 QPSK Demo-Output

The receiver output information will show in the terminal like Figure 5.9

Enter “x” into the terminal, you can stop the waveform.

(b) Running the NodeBooter in Eclipse

In Eclipse, choose “OSSIE” -> “Run Nodebooter” (If the system asks for the root password, enter “wireless”). Leave the defaults and click on “OK”.

(Note: If you need to use the other nodes for your waveform project, press “Browse” button by the side of “Start Domain Manager”, navigate to /sdr/dev/nodes/<the node name> and choose the file named “DeviceManager.dcd.xml”. Leave the defaults in Figure 5.10 and click “OK”).

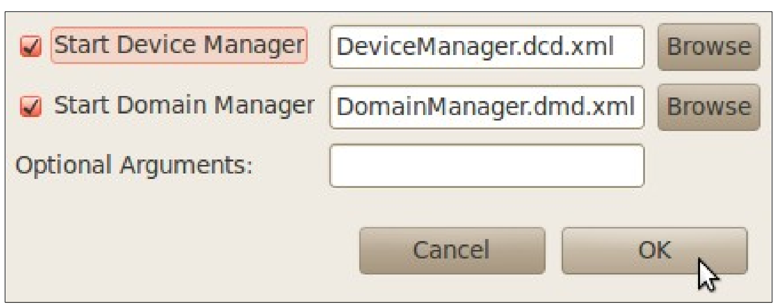


Figure 5.10 NodeBooter Node Setting

You will see the “Object Added Events” for Device Manager (SourceName: Device Manager) and APP device (SourceName: GPP1) in the console within Eclipse.

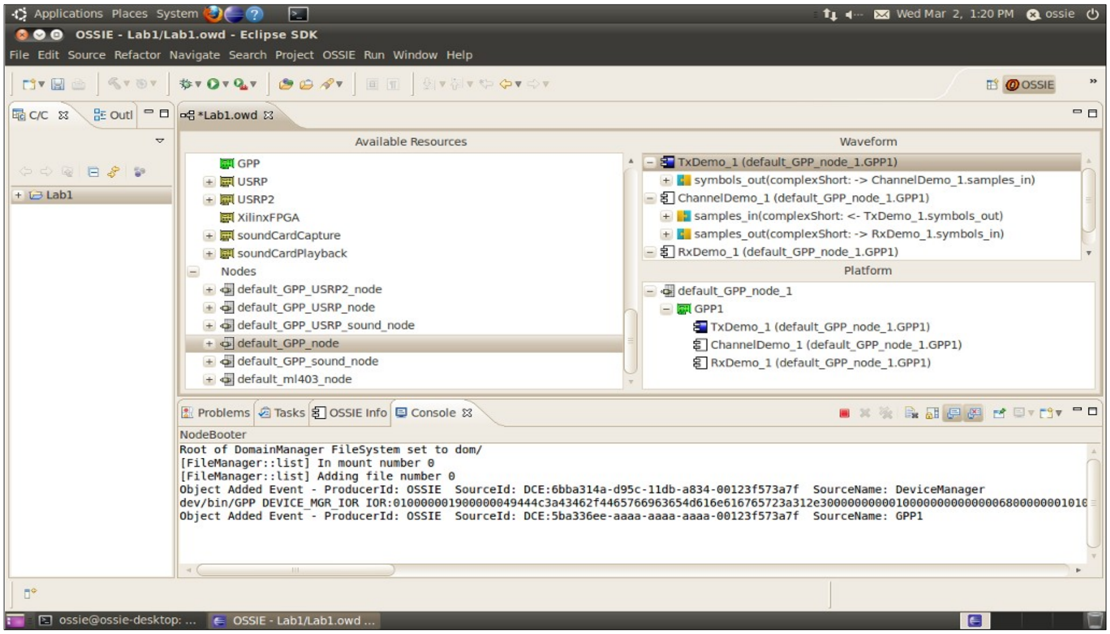


Figure 5.11 NodeBooter – Eclipse Window

Open a new terminal and load your waveform by typing: $ sudo C_wavLoader

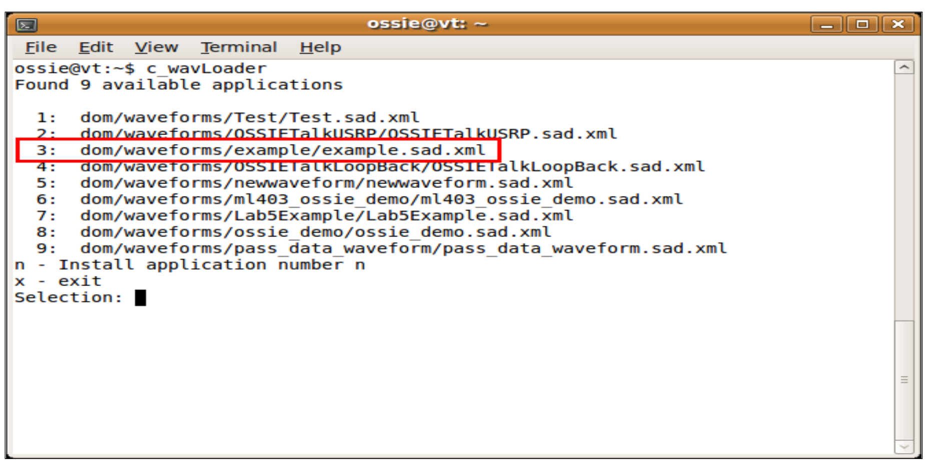


Figure 5.12 Loading Waveform Terminal-2

The waveform is loaded now, and it is a file named <your project name>.sad.xml. Type the waveform number and youe will have two choices, “s – start application” and “u – uninstall application”. Enter “s”, in the console, you will see lines of the form: “RXDemo errors: X/1024”. This is the output of the receiver, and with X number of bit errors out of 1024 bits. This output information will be stopped by pressing the rad stop button in Eclipse.

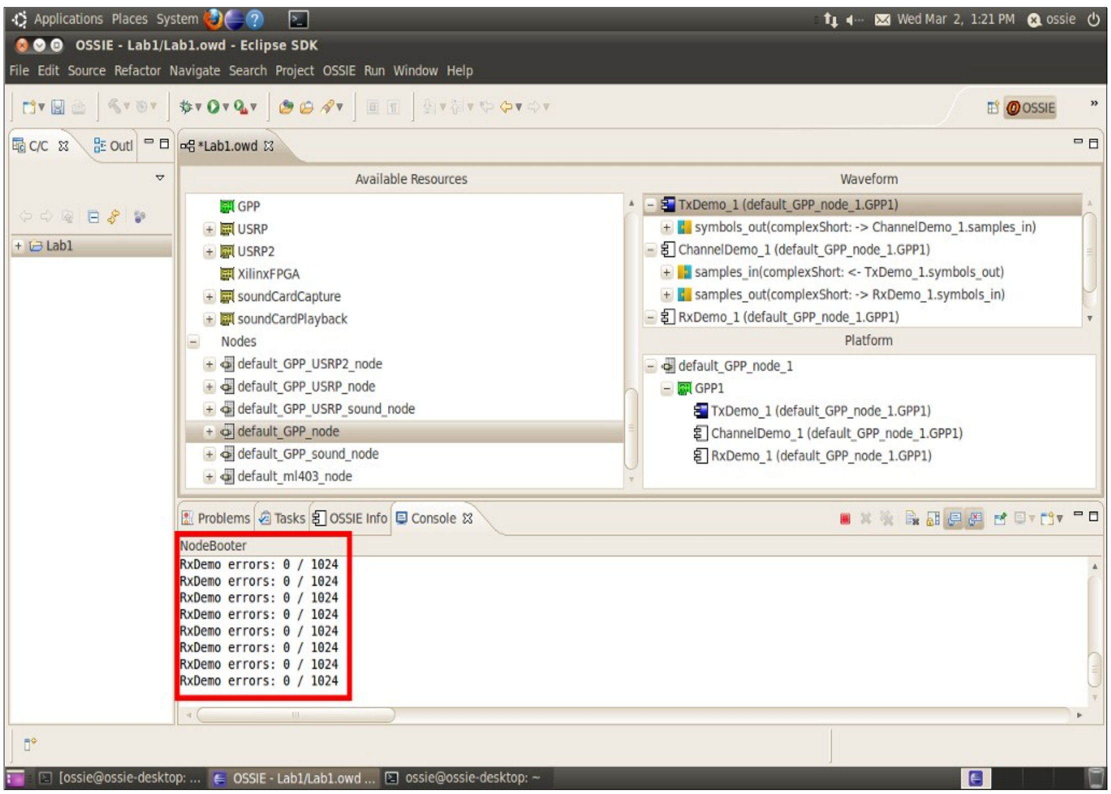


Figure 5.13 QPSK Waveform Output-Eclipse Window

5.1.1.4 ALF Graphical Debugging Environment

ALF, from which you can install and start waveforms, provides a graphical user interface for waveform. After you run the NodeBooter and start the naming service, in Eclipse, click on “OSSIE” -> “ALF”, double click on the waveform you want to run in the Waveform Application panel, then you will see your waveform appears in the Application panel. Right click on your waveform in the Application panel to display the waveform’s components in the main window.

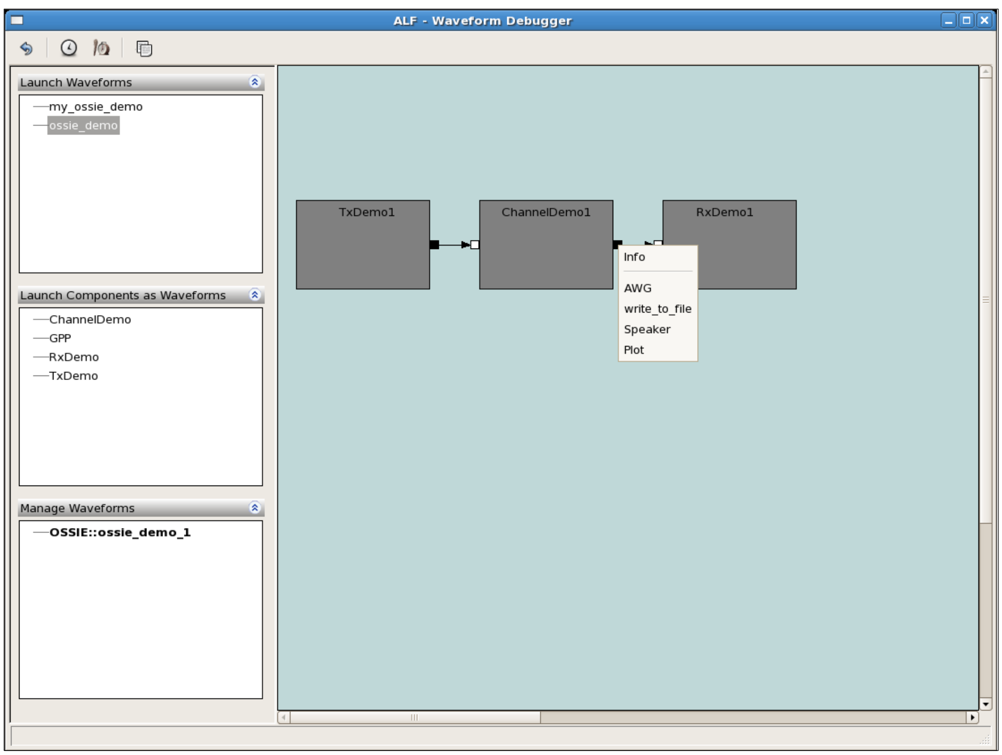


Figure 5.14 ALF Showing Waveform Running

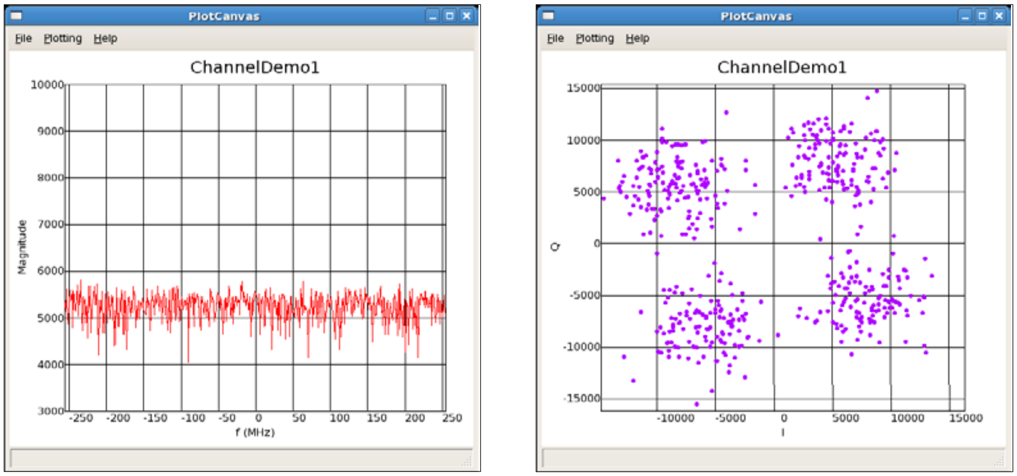


Figure 5.15 Plot Tool Showing Spectrum and I/Q

Right click on a black rectangle on the right side of ChannelDemo component in the main window, click on "Plot" option to plot output data for this waveform. PlotCanvas window shows the spectrum plotting first, if you want to see I/Q plot, click on "Plotting" on the PlotCanvas window and choose "I/Q", then the I/Q plotting window will appear.

5.1.1.5 OSSIE WaveDash

By right clicking the components in the Waveform panel, all components variables can be changed for the design. Ossie WaveDash tool also can be used to achieve this goal. After running ALF and plot waveform, click on OSSIE from the Eclipse window; choose "Wavedash", the Ossie WaveDash window will appear.

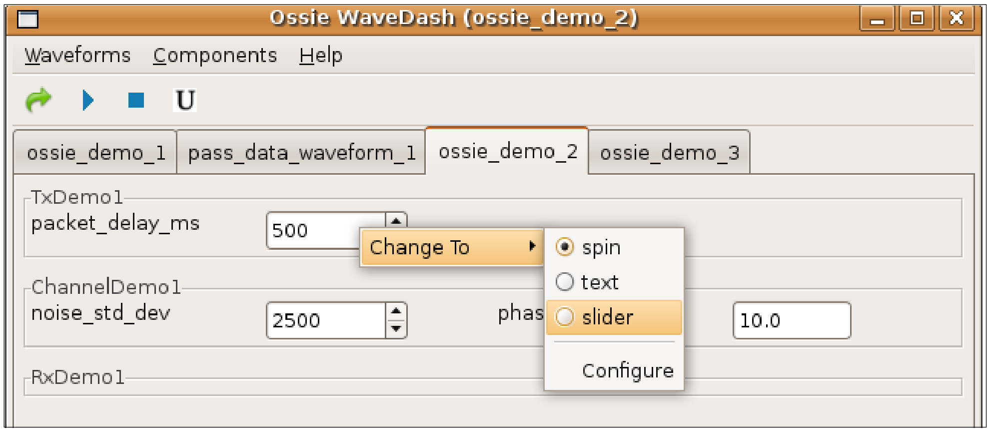


Figure 5.16 QPSK Demo - Ossie WaveDash Window

In this window, you can see all of the components' variables which can be set. Right click on the variable text box; there are three choices to change the value setting, spin, text, and slider. By choosing "Configure", you can set minimum and maximum values for the variable value slider. In this QPSK Demo, if you change noise_std_dev value, obvious the waveform output in Eclipse, you can see if you increase the noise, the bit error rate will be increased.

5.2 FM Receiver over OSSIE GNU Radio

This FM Receiver Demo is a project which receives FM signals from GNU Radio antenna and use OSSIE software components to process the signal and play it from the sound card.

5.2.1 Build and Run FM Receiver

OSSIE 0.8.2 and OSSIE Eclipse Feature (OEF) are needed for this FM receiver. In order to build FM receiver over OSSIE GNU Radio, six software components and three hardware devices are needed.

OSSIE Eclipse Software Components and variable setting:

-- USRP_Commander

rx_freq: 162475000   tx_freq: 0

tx_interp: 512   rx_decim: 256

rx_size: 8192   rx_gain: 0

rx_gain_max: 1   tx_start: 0

rx_start: 1

-- amplifier

I_gain: 1   Q_gain: 1

-- WFMDemod

-- AutomaticGainControl

energy_lo: 4000   energy_hi: 4000

k_attack: 0.002   k_release: 0.0005

g_min: 0.01   g_max: 1000

rssi_pass: 0

-- Decimator-1

DecimateBy: 10

-- Decimator-2

DecimateBy: 1

OSSIE Eclipse Hardware Devices:

-- GPP

-- USRP

-- Sound card

GNU Radio Hardware Devices:

-- USRP Mother-board

-- USRP Daughter-board: BasicRX/TX: 1 to 250 MHz IF Transmitter and Receiver

-- Antenna: HG240RD-SM

-- SMA-Bulkheads and other connection devices.

-- A speaker

Open a new OSSIE waveform in Eclipse, name it FM Receiver. Double click on these six software components and default_GPP_USRP_sound_node in the Available Resources panel, they will appear in the Waveform and Platform panel. Based on the design, set all components variables correct. Deploy all of the components to the default_GPP_USRP_sound_node node and the GPP device by dragging them on to the GPP device instance in the platform panel under default GPP_USRP_sound_node. Expand all components and device and connect them together. To do this, connect the following pairs by dragging one component port into another component port. If the connection is successful, the puzzle at the left side of the component port will be completed, you can see the blue one connects the yellow one and form a complete rectangle:

(a) USRP_Commander: RX_Control (Uses) -> USRP1: RX_Control

(b) amplifier: datain -> USRP1: RX_data

(c) amplifier: dataOut -> Decimator_01: inData

(d) Decimator_01: outData -> AutomaticGainControl: data_in

(e) AutomaticGainControl: data_out -> WFMDemod: dataIn

(f) WFMDemod: dataOut -> Decimator_02: inData

(g) Decimator_02: outData -> soundCardPlayback: soundOut

Here, the USRP_Commander should be the assembly controller, to do this, right click on the USRP_Commander in the Waveform layout panel, and select “Set Assembly Controller”. If this action is active, the rectangle at the left side of USRP_Commander component will become to be blue in both Waveform and Platform layout panels.

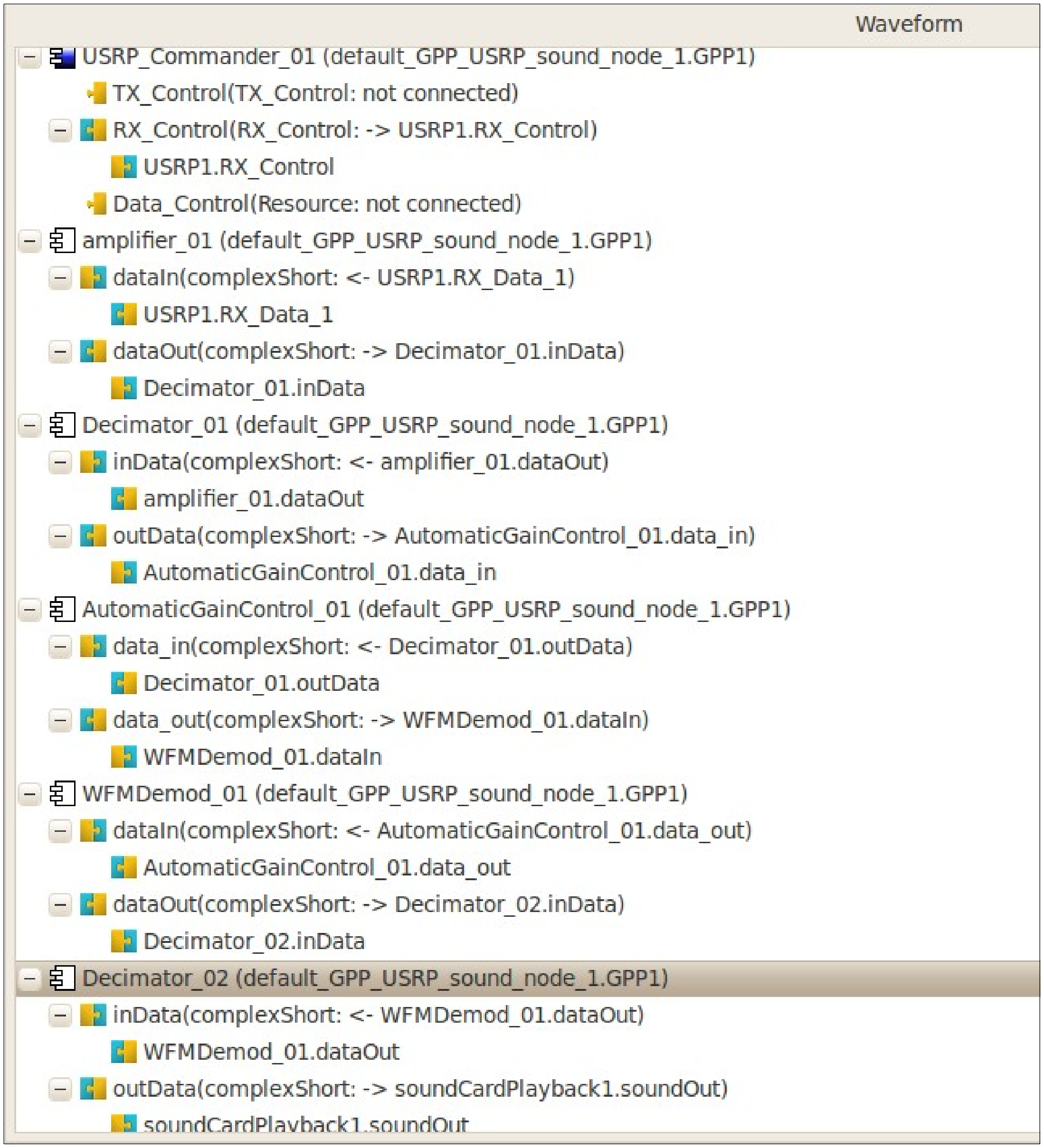


Figure 5.17 OSSIE FM Receiver Waveform Panel

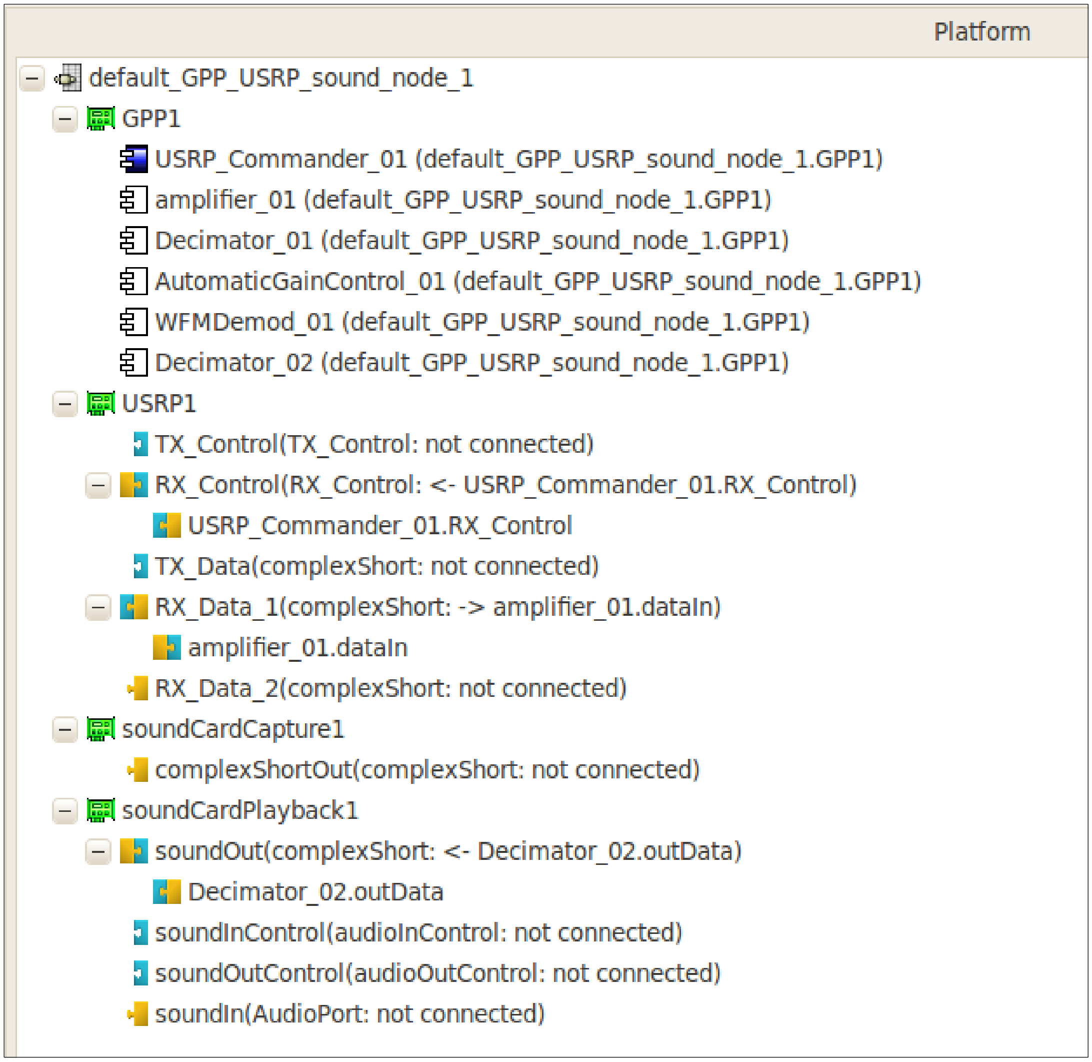


Figure 5.18 OSSIE FM Receiver Platform Panel

Press Crtl-S to save this waveform.

Navigate to /sdr/dev/xml/soundCardPlayback/, open soundCardPlayback.prf.xml by typing:

$ gedit soundCardPlayback.prf.xml

Find the following line:

<value> XXXX<value>

Change XXXX to 25000 because this is setting the sample rate to 25 KHz in the sound card.

FM Receiver only needs to receive signals, so plug Basic RX daughter board into the RX-A slot on the USRP main board and connect USRP to the computer using the USB cable.

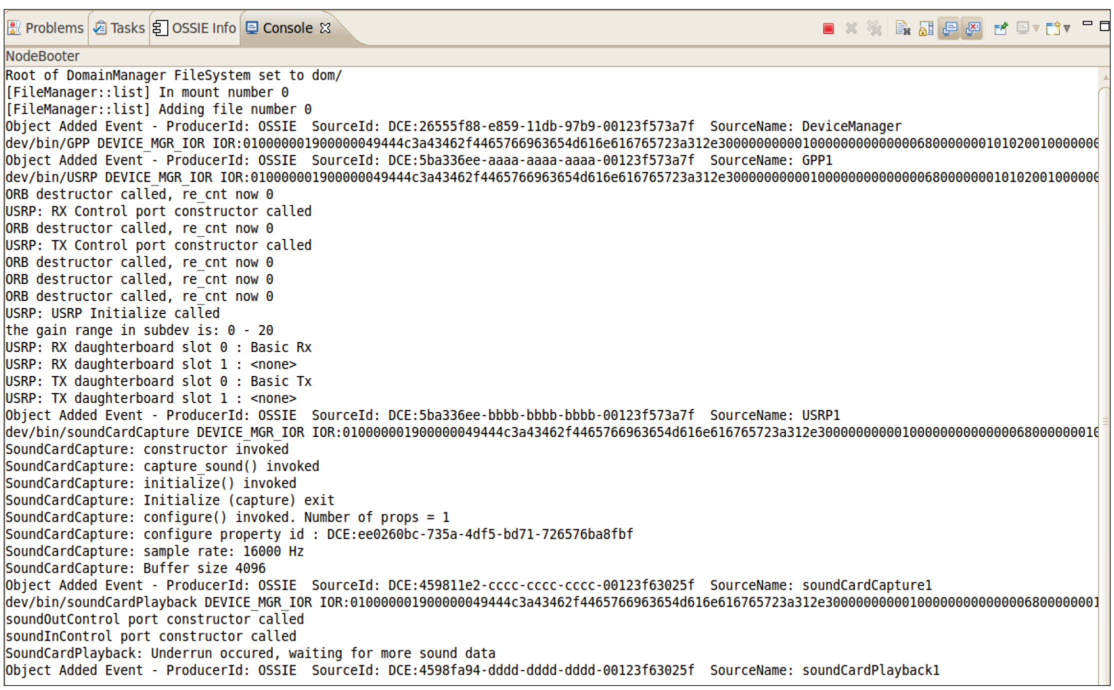


Figure 5.19 OSSIE FM Receiver NodeBooter

Run NodeBooter in Eclipse, click “OSSIE” in the Eclipse main window and choose “NodeBooter”. Select default_GPP_USRP_sound_node node by navigating to /sdr/dev/nodes/ default_GPP_USRP_sound_node from DeviceManager browse, choose DeviceManager.dcd.xml and press “OK”. The running NodeBooter in Eclipse should look like Figure 5.19.

Open ALF for FM Receiver by clicking on “OSSIE” in Eclipse and choose ALF, display FM Receiver waveform, the waveform components diagram appears and the FM radio sound will be heard from the speaker.

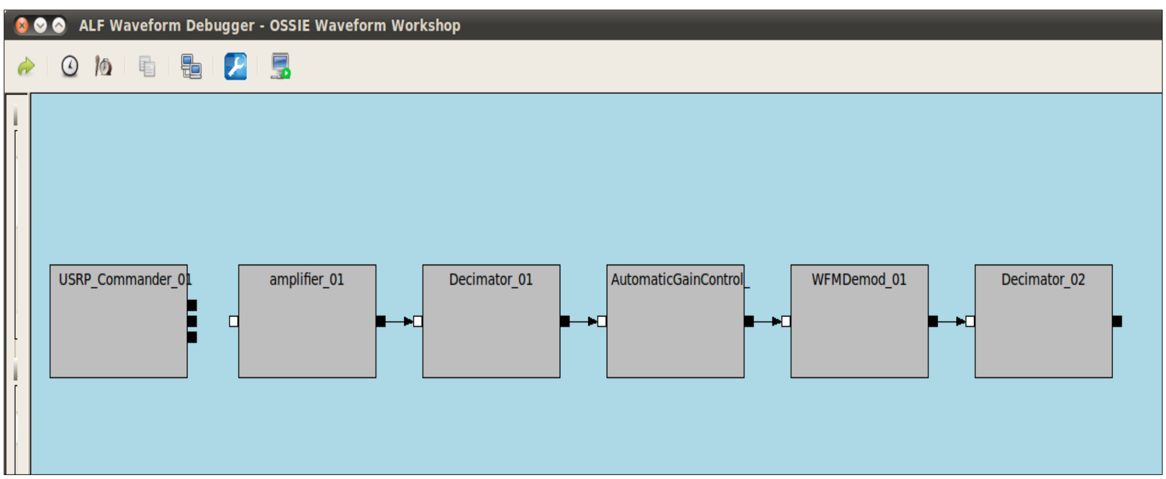


Figure 5.20 OSSIE FM Receiver ALF Waveform Debugger

Open WaveDash and change receive frequencies, and then you can enjoy the FM radio from different frequencies setting.

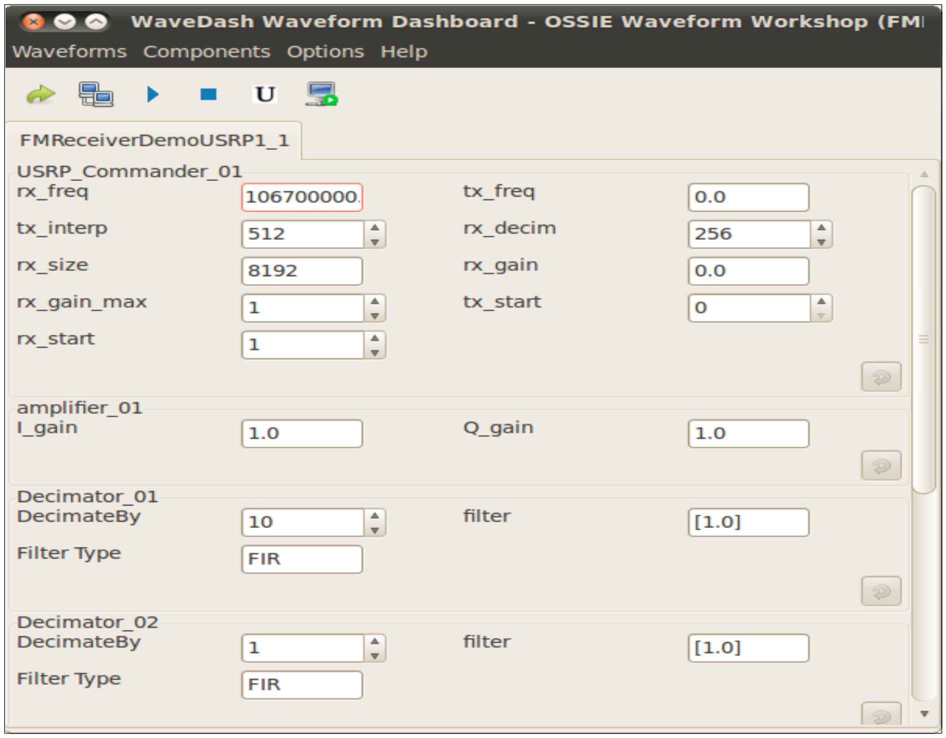


Figure 5.21 OSSIE FM Receiver WaveDash Waveform Dashboard

CHAPTER 6

CONCLUSION AND FUTURE WORK

Software Defined Radio, a radio design revolution, is an exciting field, and its flexibility provides all users with more convenience to achieve pretty much all applications a traditional radio can do. The goal of this thesis project is to implement the communication system in both GNU Radio and OSSIE to achieve SDR applications. By the comparison in this thesis, GNU Radio and OSSIE have several characteristics in common. Firstly; both of them are free toolkits for Software Defined Radio application development. Secondly, both of them have graphical tools to develop waveforms by connecting their own components. Thirdly, both of them have frameworks which provide a high level interface for waveforms development. Based on these similar features of GNU Radio and OSSIE, developing waveforms by using these two toolkits of SDR are alike. The developer need to understand the SDR framework working principles, then will be able to design a communication system by the similar strategy.

In this thesis project, the FM broadcasting signals are successfully received by the FM reveiver over GNU Radio and OSSIE, however; the signals received by GNU Radio is very clear through many FM broadcasting channels, the received signals from the OSSIE FM revceiver have more distortions and only through a few channels. This result shows that the GNU Radio signal processing blocks has better capability to process FM

signals. Unfortunatly, the OSSIE device interface for the USRP could not be used to realize the same waveform: the lack of synchronization between the USRP interface and the demodulator/modulaton component was considered to be the reason for the unexpected behavior of the waveforms. [12]

Except the SISO Communication System and FM Receiver I built in this thesis, there are many applications can be reached by SDR GNU Radio and OSSIE, such as software GPS, Web Cam transceiver, Multiple input multiple output (MIMO) processing, and sensor network, etc. A deep understanding and developing of software radio requires knowledge related to many domains. As the future work, the deep research and development of GNU Radio and OSSIE components will be applied.

APPENDIX A

UBUNTU 10.04 INSTALLATION GUIDE

Ubuntu 10.04 is used for both GNU Radio communication system and OSSIE GNU Radio system in this case, so the Ubuntu 10.04 installation will be introduced in details. Ubuntu free download can be found at http://www.ubuntu.com/download, and from this website, choose “Try it from a CD or USB stick”. There have two Ubuntu version options, select “Ubuntu 10.04 LTS – Long-term support” and 32-bit or 64-bit based on your computer hardware, then click the big orange button on the right and start download. This Ubuntu Download page introduces three tutorials, first of all; it shows how to burn Ubuntu image to a CD:

Download and install Infra Recorder from http://infrarecorder.org/ -> Insert a blank CD in the drive -> select “Do Nothing” or “Cancel” when an auto-run dialog bod pops up -> open Infra Recorder -> click “Actions” button -> click “Burn Image” -> select the Ubuntu CD image file you just downloaded -> click “Open” -> click “Ok” in the dialog box.

Secondly; it shows the option that you can try out Ubuntu before you install it after you have a bootable Ubuntu CD, by this approach, you can run Ubuntu on your computer without affecting your current system:

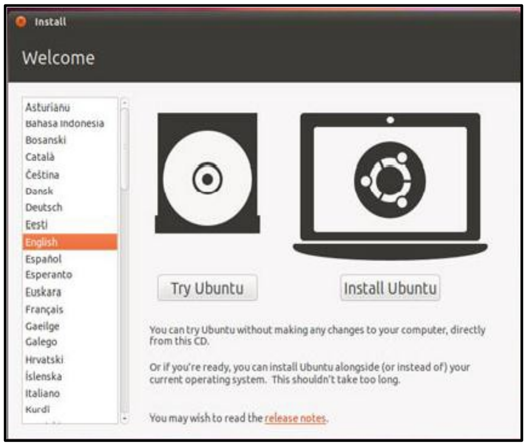


Figure A.1 Ubuntu Welcome Screen

Figure A.2 Ubuntu Desktop Screen

Insert the bootable Ubuntu CD into the CD/DVD-drive and restart the computer -> choose the language you preferred when the welcome screen (Figure A.1) appears -> click “Try Ubuntu” button.

When the Ubuntu desktop screen (Figure A.2) appears, you can try out all Ubuntu functions, if you want to install it now, just need to click “Install Ubuntu 11.04” option. Thirdly; for the convenience, you may want to install Ubuntu directly from the CD. Make sure your computer has enough space for Ubuntu -> insert the CD into CD/DVD drive and restart the computer -> choose the language you preferred and click “Install Ubuntu” -> mark “Download updates while installing (if your computer is connected to the internet)” and “Install this third-party software” -> click “Forward” -> mark “Install Ubuntu 10.04 with your Current System” or “Install Ubuntu by the entire disk” -> click “Forward” -> click “Install Now” -> setting your location -> click “Forward” -> setting your keyboard layout (optional) -> click “Forward” -> create your use-name and

password -> click “Forward” -> wait for installation -> when the “Installation Complete” window appears, click “Restart Now”, then you can enjoy the convenience of Ubuntu.[2]

APPENDIX B

AVAILABLE DAUGHTER-BOARDS AND OTHER DEVICES

B.1. BasicRX/BasicTX: Reviver/Transmitter for use with external RF hardware

(1 to 250 MHz IF Transmitter and Receiver)

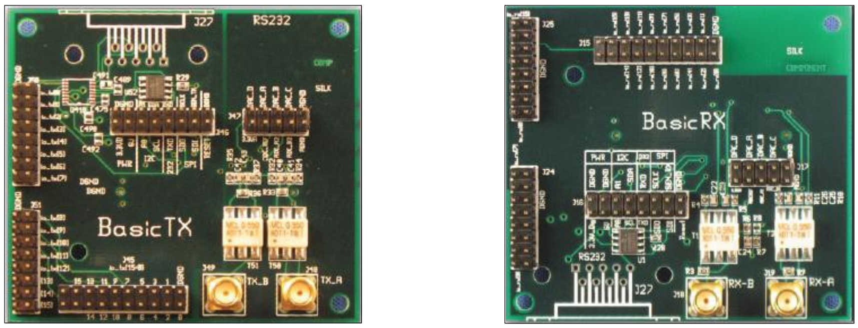


Figure B.1 Basic TX/RX

B.2. LFRX: DC to 30 MHz receiver/Transmitter (DC to 30 MHz Transmitter and

Reviewer)

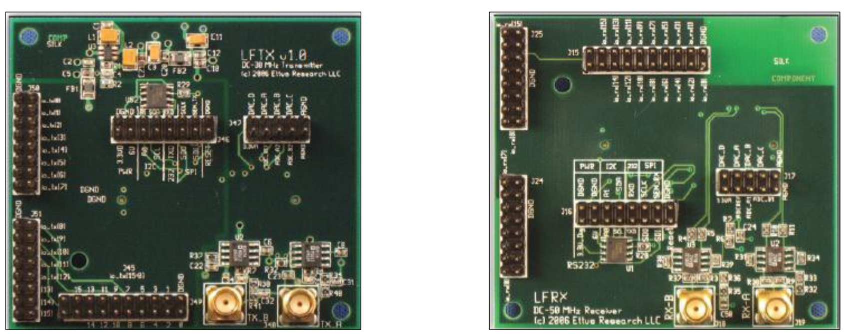


Figure B.2 LFTX/LFRX

B.3. TVRX: 50 to 860 MHz receiver (Dual 50 MHz to 860 MHz Receiver)

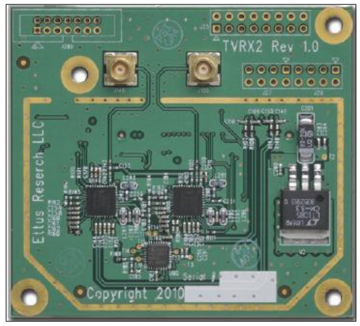


Figure B.3 TVRX2

B.4. DBSRX: 800 MHz to 2.4 GHz receiver (800 MHz to 2.4 GHz Receiver)

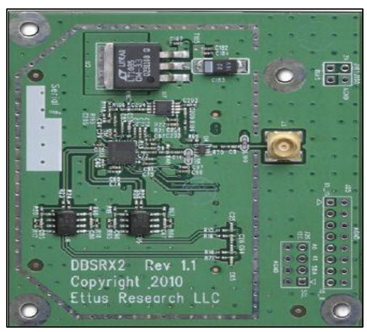
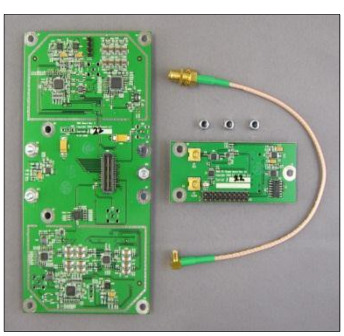


Figure B.4 DBSRX2

B.5. WBX: 50 MHz to 2.2 GHz transceiver



• Frequency Range: 50 MHz to 2.2 GHz

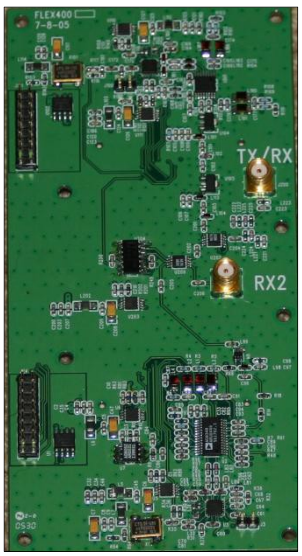
• Transmit Power: 30 to 100 mW typical

• Dual synthesizers for independent TX and RX frequencies

Figure B.5 WBX

The frequency range of the WBX covers many bands of interest, including white spaces, broadcast television, public safety, land-mobile communications, and low-power unlicensed devices.

B.6 RFX400: 400-500 MHz transceiver



• Transceiver.

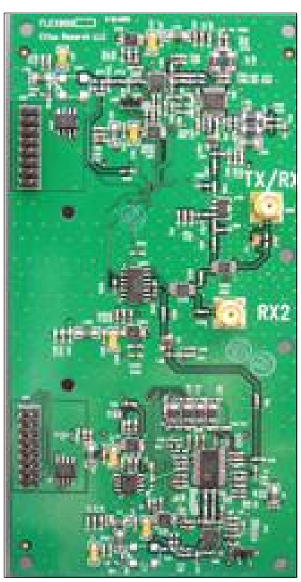
•100+mW output.

•45dB AGC.

•Can be changed to cover 200 MHz up to 800 MHz with a hardware mod.

Figure B.6 RFX400

B.7 RFX900: 750-1050 MHz transceiver



• Frequency Range: 750 to 1050 MHz

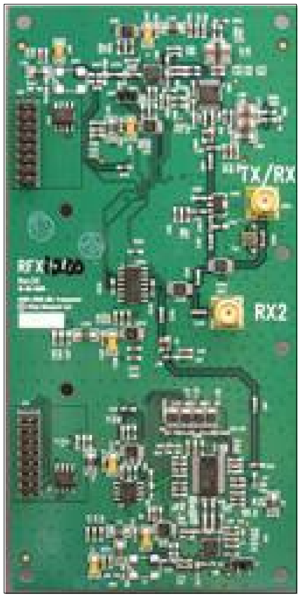
• Transmit Power: 200 mW typical

• Dual synthesizers for independent TX and RX frequencies

Features coverage of cellular, paging, two-way radio, and 902 to 928 MHz ISM band.

Figure B.7 RFX900

B.8 RFX1200: 1150-1450 MHz transceiver



• Frequency Range: 1150 to 1450 MHz

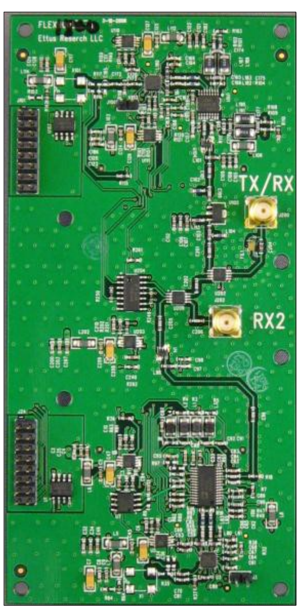
• Transmit Power: 200 mW typical

• Dual synthesizers for independent TX and RX frequencies

Features coverage of navigation, satellite, and amateur bands.

Figure B.8 RFX1200

B.9 RFX1800: 1.5-2.1 GHz transceiver



• Frequency Range: 1.5 to 2.1 GHz

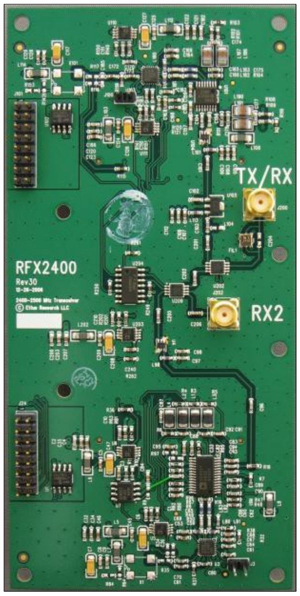
• Transmit Power: 100 mW typical

• Dual synthesizers for independent TX and RX frequencies

Features coverage of DECT, US-DECT, and PCS (including unlicensed) frequencies.

Figure B.9 RFX1800

B.10 RFX2400: 2.3-2.9 GHz transceiver



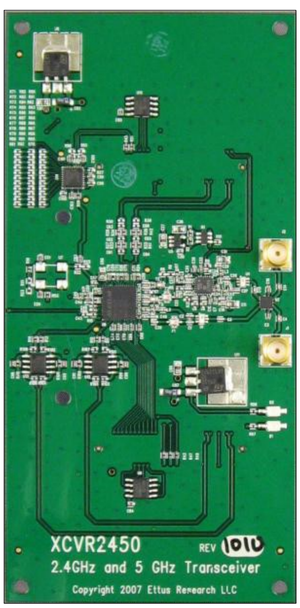
• Frequency Range: 2.3 to 2.9 GHz

• Transmit Power: 50 mW typical

The RFX2400 has a band-pass filter around the 2400 to 2483 MHz ISM band on the TXRX port, while the RX2 port is unfiltered allowing for coverage of the entire frequency range without attenuation. Features coverage of the 2.4 GHz ISM band allowing applications using most of the communications standards in this ISM band.

Figure B.10 RFX2400

B.11 XCVR2450: 2.4 GHz and 5 GHz dual-band transceiver



• Frequency Range: 2.4 to 2.5 GHz, and 4.9 to 5.9 GHz

• Transmit Power: 100 mW typical

• Single synthesizer shared between TX and RX

The XCVR2450 covers both the ISM band at 2.4 GHz and the entire 4.9 to 5.9 GHz band, including the public safety, UNII, ISM, and Japanese wireless bands.

Figure B.11 XCVR2450

B.12 RF Cables Available (SMA-Bulkhead)

SMA-M to SMA-F bulkhead connector for most daughter-boards.

Figure B.12 RF Cables Available

B.13 Antennas Available

(a) VERT400

VERT400 144 MHz, 400 MHz, and 1200 MHz Tri-band 7-inch omnidirectional vertical antenna.

Works with WBX, RFX400, RFX1200.

Figure B.13.1 VERT400

(b) VERT900

VERT900 824 to 960 MHz, 1710 to 1990 MHz Quad-band Cellular/PCS and ISM Band omnidirectional vertical antenna, 3dBi Gain.

Works with WBX, RFX900, RFX1800.

Figure B.13.2 VERT900

(c) VERT2450

VERT2450 Dual Band 2.4 to 2.48 GHz and 4.9 to 5.9 GHz omnidirectional vertical antenna, 3dBi Gain, Ideal for RFX2400 and XCVR2450.

Figure B.13.3 VERT2450

(d) HG240RD-SM

B.14 USRP Mother Board

Figure B.14

- Four 64 MS/s 12-bit analog to digital converters
- Four 128 MS/s 14-bit digital to analog converters
- Four digital down-converters with programmable decimation rates
- Two digital up-converters with programmable interpolation rates
- High-speed USB 2.0 interface (480 Mb/s)
- Capable of processing signals up to 16 MHz wide
- Modular architecture supports wide variety of RF daughter-boards
- Auxiliary analog and digital I/O support complex radio controls such as RSSI and AGC
- Fully coherent multi-channel systems (MIMO capable)

APPENDIX C

SOURCE CODES FOR SISO COMMUNICATION SYSTEM OVER GNURADIO [3]

C.1 Source Code 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 gnuradio import usrp
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser
```

```
import random
import struct
import sys
import string
import time
import pygst
pygst.require("0.10")
import gst

# 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 = OptionParser (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.dat", 'w+a')

# 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()



def on_tag(bus, msg):
    taglist = msg.parse_tag()
    print 'on_tag:'
    for key in taglist.keys():
        print '\t%s = %s' % (key, taglist[key])

#our stream to play
music_stream_uri = 'file:///home/UNT/zc0029/Desktop/one2one/received.dat'
player = gst.element_factory_make("playbin", "player")
player.set_property('uri', music_stream_uri)
#start playing
player.set_state(gst.STATE_PLAYING)

#listen for tags on the message bus; tag event might be called more than once
bus = player.get_bus()
bus.enable_sync_message_emission()
```

```
    bus.add_signal_watch()
    bus.connect('message::tag', on_tag)
    #wait and let the music play
    raw_input('Press enter to stop playing...')

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

C.2 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 OptionParser

import random, time, struct, sys

# from current dir
import usrp_transmit_path
import bpsk
#import os
#print os.getpid()
#raw_input('Attach and press enter')

class my_top_block(gr.top_block):
    def __init__(self, modulator, options):
        gr.top_block.__init__(self)

        self.txpath = usrp_transmit_path.usrp_transmit_path(modulator, options)

        self.connect(self.txpath)
```

```
# /////////////////////////////////////////////////////////////////////////////
#                                   main
# /////////////////////////////////////////////////////////////////////////////

def main():

    def send_pkt(payload='', eof=False):
        return tb.txpath.send_pkt(payload, eof)

    def rx_callback(ok, payload):
        print "ok = %r, payload = '%s'" % (ok, payload)

    mods = modulation_utils.type_1_mods()

    parser = OptionParser(option_class=eng_option, conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")

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

    parser.add_option("-s", "--size", type="eng_float", default=1500,
                      help="set packet size [default=%default]")
    parser.add_option("-M", "--megabytes", type="eng_float", default=1.0,
                      help="set megabytes to transmit [default=%default]")
    parser.add_option("","--discontinuous", action="store_true", default=False,
                      help="enable discontinous transmission (bursts of 5 packets)")
```

```
parser.add_option("","--from-file", default=None,
                  help="use file for packet contents")

usrp_transmit_path.add_options(parser, expert_grp)

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

(options, args) = parser.parse_args ()

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

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

if options.from_file is not None:
    source_file = open(options.from_file, 'r')

# build the graph
tb = my_top_block(mods[options.modulation], options)

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

```
tb.start()                    # start flow graph

# generate and send packets
nbytes = int(1e6 * options.megabytes)
n = 0
pktno = 0
pkt_size = int(options.size)

# Send dummy first packet (pktno #0)
data = (pkt_size - 2) * chr(pktno & 0xff)
payload = struct.pack('!H', pktno & 0xffff) + data
send_pkt(payload)
n += len(payload)
sys.stderr.write('.')              # Outputs "........" oncreen
pktno += 1

while n < nbytes:
   if options.from_file is None:
      data = (pkt_size - 2) * chr(pktno & 0xff)
   else:
      data = source_file.read(pkt_size - 2)
      if data == '':
         break;
   payload = struct.pack('!H', pktno & 0xffff) + data
   send_pkt(payload)
   n += len(payload)
   sys.stderr.write('.')                    # Outputs "........" oncreen
   if options.discontinuous and pktno % 5 == 4:
      time.sleep(1)
```

```
        pktno += 1

    time.sleep(5)                        # Let the last packet finish sending
    send_pkt(eof=True)

    tb.wait()                            # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

C.3 Source Code for dbpsk.py

```
#
# Copyright 2005,2006,2007 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.
#

# See gnuradio-examples/python/digital for examples

"""
differential BPSK modulation and demodulation.
"""

from gnuradio import gr, gru, modulation_utils
from math import pi, sqrt
import psk
import cmath
from pprint import pprint

# default values (used in __init__ and add_options)
_def_samples_per_symbol = 2
_def_excess_bw = 0.35
_def_gray_code = True        # does not matter for BPSK
_def_verbose = False
_def_log = False

_def_costas_alpha = 0.1
_def_gain_mu = None
```

```
_def_mu = 0.5
_def_omega_relative_limit = 0.005


# /////////////////////////////////////////////////////////////////////////////
#                         BPSK modulator
# /////////////////////////////////////////////////////////////////////////////

class bpsk_mod(gr.hier_block2):

    def __init__(self,
                 samples_per_symbol=_def_samples_per_symbol,
                 excess_bw=_def_excess_bw,
                 gray_code=_def_gray_code,
                 verbose=_def_verbose,
                 log=_def_log):
        """
        Hierarchical block for RRC-filtered differential BPSK modulation.

        The input is a byte stream (unsigned char) and the
        output is the complex modulated signal at baseband.

        @param samples_per_symbol: samples per baud >= 2
        @type samples_per_symbol: integer
        @param excess_bw: Root-raised cosine filter excess bandwidth
        @type excess_bw: float
        @param gray_code: Tell modulator to Gray code the bits
        @type gray_code: bool
        @param verbose: Print information about modulator?
```

```
        @type verbose: bool
        @param log: Log modulation data to files?
        @type log: bool
        """

        gr.hier_block2.__init__(self, "bpsk_mod",
                                gr.io_signature(1, 1, gr.sizeof_char),       # Input signature
                                gr.io_signature(1, 1, gr.sizeof_gr_complex)) # Output
signature

        self._samples_per_symbol = samples_per_symbol
        self._excess_bw = excess_bw
        self._gray_code = gray_code

        if not isinstance(self._samples_per_symbol, int) or self._samples_per_symbol < 2:
            raise TypeError, ("sbp must be an integer >= 2, is %d" %
self._samples_per_symbol)

            ntaps = 11 * self._samples_per_symbol

        arity = pow(2,self.bits_per_symbol()) # arity = 2^1 = 2

        # turn bytes into k-bit vectors
        self.bytes2chunks = gr.packed_to_unpacked_bb(self.bits_per_symbol(),
gr.GR_MSB_FIRST)

        if self._gray_code:
            self.symbol_mapper = gr.map_bb(psk.binary_to_gray[arity])          # [0, 1]
        else:
```

```
        self.symbol_mapper = gr.map_bb(psk.binary_to_ungray[arity])      # [0, 1]

      self.diffenc = gr.diff_encoder_bb(arity)

      self.chunks2symbols = gr.chunks_to_symbols_bc(psk.constellation[arity])     # [-
1+0j, 1+0j]

      # pulse shaping filter
        self.rrc_taps = gr.firdes.root_raised_cosine(
          self._samples_per_symbol,   # gain (samples_per_symbol since we're
                          # interpolating by samples_per_symbol)
          self._samples_per_symbol,   # sampling rate
          1.0,                 # symbol rate
          self._excess_bw,           # excess bandwidth (roll-off factor)
        ntaps)
        self.rrc_filter = gr.interp_fir_filter_ccf(self._samples_per_symbol,
                                    self.rrc_taps)

        # Connect
      self.connect(self, self.bytes2chunks, self.symbol_mapper, self.diffenc,
              self.chunks2symbols, self.rrc_filter, self)

      if verbose:
        self._print_verbage()

      if log:
        self._setup_logging()

    def samples_per_symbol(self):
```

```
        return self._samples_per_symbol

    def bits_per_symbol(self=None):   # static method that's also callable on an instance
        return 1
    bits_per_symbol = staticmethod(bits_per_symbol)      # make it a static method.
RTFM

    def add_options(parser):
        """
        Adds BPSK modulation-specific options to the standard parser
        """
        parser.add_option("", "--excess-bw", type="float", default=_def_excess_bw,
                          help="set RRC excess bandwith factor [default=%default]")
        parser.add_option("", "--no-gray-code", dest="gray_code",
                          action="store_false", default=True,
                          help="disable gray coding on modulated bits (PSK)")
    add_options=staticmethod(add_options)

    def extract_kwargs_from_options(options):
        """
        Given command line options, create dictionary suitable for passing to __init__
        """
        return modulation_utils.extract_kwargs_from_options(bpsk_mod.__init__,
                                                            ('self',), options)
    extract_kwargs_from_options=staticmethod(extract_kwargs_from_options)


    def _print_verbage(self):
        print "\nModulator:"
```

```
        print "bits per symbol:     %d" % self.bits_per_symbol()
        print "Gray code:           %s" % self._gray_code
        print "RRC roll-off factor: %.2f" % self._excess_bw

    def _setup_logging(self):
        print "Modulation logging turned on."
#        self.connect(self.bytes2chunks,
#                     gr.file_sink(gr.sizeof_char, "tx_bytes2chunks.dat"))
#        self.connect(self.symbol_mapper,
#                     gr.file_sink(gr.sizeof_char, "tx_graycoder.dat"))
#        self.connect(self.diffenc,
#                     gr.file_sink(gr.sizeof_char, "tx_diffenc.dat"))
#        self.connect(self.chunks2symbols,
#                     gr.file_sink(gr.sizeof_gr_complex, "tx_chunks2symbols.dat"))
#        self.connect(self.rrc_filter,
#                     gr.file_sink(gr.sizeof_gr_complex, "tx_rrc_filter.dat"))


# /////////////////////////////////////////////////////////////////////////////
#                             BPSK demodulator
# /////////////////////////////////////////////////////////////////////////////

class bpsk_demod(gr.hier_block2):

    def __init__(self,
                 samples_per_symbol=_def_samples_per_symbol,
                 excess_bw=_def_excess_bw,
                 costas_alpha=_def_costas_alpha,
                 gain_mu=_def_gain_mu,
```

```
        mu=_def_mu,
        omega_relative_limit=_def_omega_relative_limit,
        gray_code=_def_gray_code,
        verbose=_def_verbose,
        log=_def_log):
"""
    Hierarchical block for RRC-filtered differential BPSK demodulation

    The input is the complex modulated signal at baseband.
    The output is a stream of bits packed 1 bit per byte (LSB)

    @param samples_per_symbol: samples per symbol >= 2
    @type samples_per_symbol: float
    @param excess_bw: Root-raised cosine filter excess bandwidth
    @type excess_bw: float
@param costas_alpha: loop filter gain
@type costas_alphas: float
@param gain_mu: for M&M block
@type gain_mu: float
@param mu: for M&M block
@type mu: float
@param omega_relative_limit: for M&M block
@type omega_relative_limit: float
@param gray_code: Tell modulator to Gray code the bits
@type gray_code: bool
@param verbose: Print information about modulator?
@type verbose: bool
@param debug: Print modualtion data to files?
@type debug: bool
```

```
        """

        gr.hier_block2.__init__(self, "bpsk_demod",
                                gr.io_signature(1, 1, gr.sizeof_gr_complex), # Input
signature
                                gr.io_signature(1, 1, gr.sizeof_char))       # Output signature

        self._samples_per_symbol = samples_per_symbol
        self._excess_bw = excess_bw
        self._costas_alpha = costas_alpha
        self._mm_gain_mu = gain_mu
        self._mm_mu = mu
        self._mm_omega_relative_limit = omega_relative_limit
        self._gray_code = gray_code

        if samples_per_symbol < 2:
            raise TypeError, "samples_per_symbol must be >= 2, is %r" %
(samples_per_symbol,)

        arity = pow(2,self.bits_per_symbol()) #arity = 2^1 = 2

        # Automatic gain control
        scale = (1.0/16384.0)
        self.pre_scaler = gr.multiply_const_cc(scale)   # scale the signal from full-range to
+-1
        #self.agc = gr.agc2_cc(0.6e-1, 1e-3, 1, 1, 100)
        self.agc = gr.feedforward_agc_cc(16, 2.0)

        # RRC data filter
```

```
ntaps = 11 * samples_per_symbol
self.rrc_taps = gr.firdes.root_raised_cosine(
    1.0,                    # gain
    self._samples_per_symbol, # sampling rate
    1.0,                    # symbol rate
    self._excess_bw,          # excess bandwidth (roll-off factor)
    ntaps)
self.rrc_filter = gr.interp_fir_filter_ccf(1, self.rrc_taps)

# symbol clock recovery
if not self._mm_gain_mu:
    self._mm_gain_mu = 0.1

self._mm_omega = self._samples_per_symbol
self._mm_gain_omega = .25 * self._mm_gain_mu * self._mm_gain_mu
self._costas_beta  = 0.25 * self._costas_alpha * self._costas_alpha
fmin = -0.1
fmax = 0.1

self.receiver=gr.mpsk_receiver_cc(arity, 0,
                        self._costas_alpha, self._costas_beta,
                        fmin, fmax,
                        self._mm_mu, self._mm_gain_mu,
                        self._mm_omega, self._mm_gain_omega,
                        self._mm_omega_relative_limit)

# Do differential decoding based on phase change of symbols
self.diffdec = gr.diff_phasor_cc()
```

```
        # find closest constellation point
        rot = 1
        rotated_const = map(lambda pt: pt * rot, psk.constellation[arity])
        self.slicer = gr.constellation_decoder_cb(rotated_const, range(arity))

        if self._gray_code:
            self.symbol_mapper = gr.map_bb(psk.gray_to_binary[arity])          # [0, 1]
        else:
            self.symbol_mapper = gr.map_bb(psk.ungray_to_binary[arity])      # [0, 1]

        # unpack the k bit vector into a stream of bits
        self.unpack = gr.unpack_k_bits_bb(self.bits_per_symbol())

        if verbose:
            self._print_verbage()

        if log:
            self._setup_logging()

        # Connect and Initialize base class
        self.connect(self, self.pre_scaler, self.agc, self.rrc_filter, self.receiver,
                     self.diffdec, self.slicer, self.symbol_mapper, self.unpack, self)
    def samples_per_symbol(self):
        return self._samples_per_symbol

    def bits_per_symbol(self=None):   # staticmethod that's also callable on an instance
        return 1
    bits_per_symbol = staticmethod(bits_per_symbol)      # make it a static method.
RTFM
```

```
    def _print_verbage(self):
        print "\nDemodulator:"
        print "bits per symbol:     %d"   % self.bits_per_symbol()
        print "Gray code:           %s"   % self._gray_code print
        "RRC roll-off factor: %.2f" % self._excess_bw print
        "Costas Loop alpha:   %.2e" % self._costas_alpha print
        "Costas Loop beta:          %.2e" % self._costas_beta
        print "M&M mu:              %.2f" % self._mm_mu
        print "M&M mu gain:         %.2e" % self._mm_gain_mu
        print "M&M omega:           %.2f" % self._mm_omega
        print "M&M omega gain:      %.2e" % self._mm_gain_omega
        print "M&M omega limit:     %.2f" % self._mm_omega_relative_limit

    def _setup_logging(self):
        print "Modulation logging turned on."
#       self.connect(self.pre_scaler,
#                    gr.file_sink(gr.sizeof_gr_complex, "rx_prescaler.dat"))
#       self.connect(self.agc,
#                    gr.file_sink(gr.sizeof_gr_complex, "rx_agc.dat"))
#       self.connect(self.rrc_filter,
#                    gr.file_sink(gr.sizeof_gr_complex, "rx_rrc_filter.dat"))
#       self.connect(self.receiver,
#                    gr.file_sink(gr.sizeof_gr_complex, "rx_receiver.dat"))
#       self.connect(self.diffdec,
#                    gr.file_sink(gr.sizeof_gr_complex, "rx_diffdec.dat"))
#       self.connect(self.slicer,
#                    gr.file_sink(gr.sizeof_char, "rx_slicer.dat"))
#       self.connect(self.symbol_mapper,
```

```
#                   gr.file_sink(gr.sizeof_char, "rx_symbol_mapper.dat"))
#         self.connect(self.unpack,
#                   gr.file_sink(gr.sizeof_char, "rx_unpack.dat"))

    def add_options(parser):
        """
        Adds BPSK demodulation-specific options to the standard parser
        """
        parser.add_option("", "--excess-bw", type="float", default=_def_excess_bw,
                      help="set RRC excess bandwith factor [default=%default] (PSK)")
        parser.add_option("", "--no-gray-code", dest="gray_code",
                      action="store_false", default=_def_gray_code,
                      help="disable gray coding on modulated bits (PSK)")
        parser.add_option("", "--costas-alpha", type="float", default=None,
                      help="set Costas loop alpha value [default=%default] (PSK)")
        parser.add_option("", "--gain-mu", type="float", default=_def_gain_mu,
                      help="set M&M symbol sync loop gain mu value [default=%default]
(GMSK/PSK)")
        parser.add_option("", "--mu", type="float", default=_def_mu,
                      help="set M&M symbol sync loop mu value [default=%default]
(GMSK/PSK)")
        parser.add_option("", "--omega-relative-limit", type="float",
default=_def_omega_relative_limit,
                      help="M&M clock recovery omega relative limit [default=%default]
(GMSK/PSK)")
    add_options=staticmethod(add_options)

    def extract_kwargs_from_options(options):
        """
```

```
        Given command line options, create dictionary suitable for passing to __init__
        """
        return modulation_utils.extract_kwargs_from_options(
                 bpsk_demod.__init__, ('self',), options)
    extract_kwargs_from_options=staticmethod(extract_kwargs_from_options)
#
# Add these to the mod/demod registry
#
modulation_utils.add_type_1_mod('bpsk', bpsk_mod)
modulation_utils.add_type_1_demod('bpsk', bpsk_demod)
```

C.4 Source Code for generic_usrp.py

```
#
# Copyright 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.
#

USRP1_TYPE = 'usrp1'
USRP2_TYPE = 'usrp2'
DUMMY_TYPE = 'dummy'
#usrp2 rates common for decim and interp
_USRP2_RATES = range(4, 128+1, 1) + range(130, 256+1, 2) + range(260, 512+1, 4)
#dummy common rates
_DUMMY_XRATES = range(4, 512, 2)
_DUMMY_CONVERTER_RATE = 100e6
#dummy freq result
class _dummy_freq_result(object):
    def __init__(self, target_freq):
        self.baseband_freq = target_freq
        self.dxc_freq = 0
        self.residual_freq = 0
from gnuradio import gr, usrp, usrp2

########################################################################
# generic usrp common stuff
########################################################################
class _generic_usrp_base(object):

    def __init__(self, which=0, subdev_spec=None, interface="", mac_addr="",
        fusb_block_size=0, fusb_nblocks=0, usrpx=None, lo_offset=None, gain=None):
        self._lo_offset = lo_offset
```

```
    #usrp options self._which =
    which self._subdev_spec =
    subdev_spec
    #usrp2 options
    self._interface = interface
    self._mac_addr = mac_addr
    #fusb options
    self._fusb_block_size = fusb_block_size
    self._fusb_nblocks = fusb_nblocks
    #pick which usrp model
    if usrpx == '0': self._setup_usrpx(DUMMY_TYPE)
    elif usrpx == '1' or self._subdev_spec: self._setup_usrpx(USRP1_TYPE)
    elif usrpx == '2' or self._mac_addr: self._setup_usrpx(USRP2_TYPE)
    else: #automatic
        try: self._setup_usrpx(USRP2_TYPE)
        except:
            try: self._setup_usrpx(USRP1_TYPE)
            except: raise Exception, 'Failed to automatically setup a usrp device.'
    #post usrp setup
    if self._lo_offset is not None:
        self.set_lo_offset(self._lo_offset)
    self.set_gain(gain)
    self.set_auto_tr(True)
def _setup_usrpx(self, type):
    """
    Call the appropriate setup method.
    @param type the usrp type constant
    """
    self._type = type
```

```
    if self._type == USRP1_TYPE: self._setup_usrp1()
    elif self._type == USRP2_TYPE: self._setup_usrp2()
    elif self._type == DUMMY_TYPE: self._setup_dummy()

def __str__(self):
    if self._type == USRP1_TYPE: return self._subdev.side_and_name()
    elif self._type == USRP2_TYPE:
        return 'Interface: %s    MAC Address: %s    D-Board ID: 0x%.2x'%(
            self._u.interface_name(), self._u.mac_addr(), self._u.daughterboard_id())
    elif self._type == DUMMY_TYPE: return 'Dummy USRP Device'

def gain(self): return self._gain

def set_gain(self, gain=None):
    #automatic gain calculation
    r = self.gain_range()
    if gain is None: gain = (r[0] + r[1])/2 # set gain to midpoint
    #set gain for usrp
    self._gain = gain
    if self._type == USRP1_TYPE: return self._subdev.set_gain(gain)
    elif self._type == USRP2_TYPE: return self._u.set_gain(gain)
    elif self._type == DUMMY_TYPE: return True

def gain_range(self):
    if self._type == USRP1_TYPE: return self._subdev.gain_range()
    elif self._type == USRP2_TYPE: return self._u.gain_range()
    elif self._type == DUMMY_TYPE: return (0, 0, 0)

def set_center_freq(self, target_freq):
```

```
        if self._type == USRP1_TYPE:
            return self._u.tune(self._dxc, self._subdev, target_freq)
        elif self._type == USRP2_TYPE:
            return self._u.set_center_freq(target_freq)
        elif self._type == DUMMY_TYPE: return _dummy_freq_result(target_freq)

    def freq_range(self):
        if self._type == USRP1_TYPE: return self._subdev.freq_range()
        elif self._type == USRP2_TYPE: return self._u.freq_range()
        elif self._type == DUMMY_TYPE: return (-10e9, 10e9, 100e3)

    def set_lo_offset(self, lo_offset):
        if self._type == USRP1_TYPE: return self._subdev.set_lo_offset(lo_offset)
        elif self._type == USRP2_TYPE: return self._u.set_lo_offset(lo_offset)
        elif self._type == DUMMY_TYPE: return True

    def set_auto_tr(self, enable):
        if self._type == USRP1_TYPE: return self._subdev.set_auto_tr(enable)

    def __del__(self):
        try: # Avoid weak reference error
            del self._u
            del self._subdev
        except: pass

########################################################################
# generic usrp source
########################################################################
class generic_usrp_source_c(_generic_usrp_base, gr.hier_block2):
```

```
    """
    Create a generic usrp source that represents usrp and usrp2.
    Take usrp and usrp2 constructor arguments and try to figure out usrp or usrp2.
    Provide generic access methods so the API looks the same for both.
    """

    def __init__(self, **kwargs):
        gr.hier_block2.__init__(self, "generic_usrp_source",
            gr.io_signature(0, 0, 0), # Input signature
            gr.io_signature(1, 1, gr.sizeof_gr_complex)) # Output signature
        _generic_usrp_base.__init__(self, **kwargs)
        self.connect(self._u, self)

    ####################################################################
    # generic access methods
    ####################################################################
    def set_decim(self, decim):
        if decim not in self.get_decim_rates(): return False
        if self._type == USRP1_TYPE: return self._u.set_decim_rate(decim)
        elif self._type == USRP2_TYPE: return self._u.set_decim(decim)
        elif self._type == DUMMY_TYPE: return True

    def get_decim_rates(self):
        if self._type == USRP1_TYPE: return range(8, 256+1, 2) #default firmware w/ hb
filters
        if self._type == USRP2_TYPE: return _USRP2_RATES
        elif self._type == DUMMY_TYPE: return _DUMMY_XRATES

    def adc_rate(self):
```

```
        if self._type == USRP1_TYPE: return self._u.adc_rate()
        if self._type == USRP2_TYPE: return self._u.adc_rate()
        elif self._type == DUMMY_TYPE: return _DUMMY_CONVERTER_RATE

    ####################################################################
    # setup usrp methods
    ####################################################################
    def _setup_usrp1(self):
        self._u = usrp.source_c (self._which,
                         fusb_block_size=self._fusb_block_size,
                         fusb_nblocks=self._fusb_nblocks)
        # determine the daughterboard subdevice we're using
        if self._subdev_spec is None:
            self._subdev_spec = usrp.pick_rx_subdevice(self._u) self._subdev =
        usrp.selected_subdev(self._u, self._subdev_spec)
        self._u.set_mux(usrp.determine_rx_mux_value(self._u, self._subdev_spec))
        self._dxc = 0

    def _setup_usrp2(self):
        self._u = usrp2.source_32fc(self._interface, self._mac_addr)

    def _setup_dummy(self): self._u = gr.null_source(gr.sizeof_gr_complex)

########################################################################
# generic usrp sink
########################################################################
class generic_usrp_sink_c(_generic_usrp_base, gr.hier_block2):
    """
    Create a generic usrp sink that represents usrp and usrp2.
```

```
Take usrp and usrp2 constructor arguments and try to figure out usrp or usrp2.
Provide generic access methods so the API looks the same for both.
"""

def __init__(self, **kwargs):
    gr.hier_block2.__init__(self, "generic_usrp_sink",
        gr.io_signature(1, 1, gr.sizeof_gr_complex), # Input signature
        gr.io_signature(0, 0, 0)) # Output signature
    _generic_usrp_base.__init__(self, **kwargs)
    if self._type == USRP1_TYPE: #scale 0.0 to 1.0 input for usrp1
        self.connect(self, gr.multiply_const_cc((2**15)-1), self._u)
    else: self.connect(self, self._u)

####################################################################
# generic access methods
####################################################################
def set_interp(self, interp):
    if interp not in self.get_interp_rates(): return False
    if self._type == USRP1_TYPE: return self._u.set_interp_rate(interp)
    elif self._type == USRP2_TYPE: return self._u.set_interp(interp)
    elif self._type == DUMMY_TYPE: return True

def get_interp_rates(self):
    if self._type == USRP1_TYPE: return range(16, 512+1, 4)
    if self._type == USRP2_TYPE: return _USRP2_RATES
    elif self._type == DUMMY_TYPE: return _DUMMY_XRATES

def dac_rate(self):
    if self._type == USRP1_TYPE: return self._u.dac_rate()
```

```
    if self._type == USRP2_TYPE: return self._u.dac_rate()
    elif self._type == DUMMY_TYPE: return _DUMMY_CONVERTER_RATE

  ######################################################################
  # setup usrp methods
  ######################################################################
  def _setup_usrp1(self):
    self._u = usrp.sink_c (self._which,
                fusb_block_size=self._fusb_block_size,
                fusb_nblocks=self._fusb_nblocks)
    # determine the daughterboard subdevice we're using
    if self._subdev_spec is None:
      self._subdev_spec = usrp.pick_tx_subdevice(self._u) self._subdev =
    usrp.selected_subdev(self._u, self._subdev_spec)
    self._u.set_mux(usrp.determine_tx_mux_value(self._u, self._subdev_spec))
    self._dxc = self._subdev.which()

  def _setup_usrp2(self): self._u = usrp2.sink_32fc(self._interface, self._mac_addr)

  def _setup_dummy(self): self._u = gr.null_sink(gr.sizeof_gr_complex)
```

C.5 Source Code for pick_britrate.py

```
#
# Copyright 2005,2006 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 eng_notation
_default_bitrate = 500e3
_valid_samples_per_symbol = (2,3,4,5,6,7)
def _gen_tx_info(converter_rate, xrates):
    results = []
    for samples_per_symbol in _valid_samples_per_symbol:
        for interp in xrates:
            bitrate = converter_rate / interp / samples_per_symbol
            results.append((bitrate, samples_per_symbol, interp))
    results.sort()
    return results

def _gen_rx_info(converter_rate, xrates):
```

```
    results = []
    for samples_per_symbol in _valid_samples_per_symbol:
        for decim in xrates:
            bitrate = converter_rate / decim / samples_per_symbol
            results.append((bitrate, samples_per_symbol, decim))
    results.sort()
    return results

def _filter_info(info, samples_per_symbol, xrate):
    if samples_per_symbol is not None:
        info = [x for x in info if x[1] == samples_per_symbol]
    if xrate is not None:
        info = [x for x in info if x[2] == xrate]
    return info

def _pick_best(target_bitrate, bits_per_symbol, info):
    """
    @returns tuple (bitrate, samples_per_symbol, interp_rate_or_decim_rate)
    """
    if len(info) == 0:
        raise RuntimeError, "info is zero length!"
    if target_bitrate is None:     # return the fastest one
        return info[-1]

    # convert bit rate to symbol rate
    target_symbolrate = target_bitrate / bits_per_symbol

    # Find the closest matching symbol rate.
    # In the event of a tie, the one with the lowest samples_per_symbol wins.
```

```
    # (We already sorted them, so the first one is the one we take)

    best = info[0]
    best_delta = abs(target_symbolrate - best[0])
    for x in info[1:]:
        delta = abs(target_symbolrate - x[0])
        if delta < best_delta:
            best_delta  =  delta
            best = x

    # convert symbol rate back to bit rate
    return ((best[0] * bits_per_symbol),) + best[1:]

def _pick_bitrate(bitrate, bits_per_symbol, samples_per_symbol,
                  xrate, converter_rate, xrates, gen_info):
    """
    @returns tuple (bitrate, samples_per_symbol, interp_rate_or_decim_rate)
    """
    if not isinstance(bits_per_symbol, int) or bits_per_symbol < 1:
        raise ValueError, "bits_per_symbol must be an int >= 1"

    if samples_per_symbol is not None and xrate is not None:  # completely determined
        return (float(converter_rate) / xrate / samples_per_symbol,
                samples_per_symbol, xrate)
    if bitrate is None and samples_per_symbol is None and xrate is None:
        bitrate = _default_bitrate
    # now we have a target bitrate and possibly an xrate or
    # samples_per_symbol constraint, but not both of them.
```

```
    ret = _pick_best(bitrate, bits_per_symbol,
                  _filter_info(gen_info(converter_rate, xrates), samples_per_symbol, xrate))
    print "Actual Bitrate:", eng_notation.num_to_str(ret[0])
    return ret

# ---------------------------------------------------------------------------------------

def pick_tx_bitrate(bitrate, bits_per_symbol, samples_per_symbol,
                 interp_rate, converter_rate, possible_interps):
    """
    Given the 4 input parameters, return at configuration that matches

    @param bitrate: desired bitrate or None
    @type bitrate: number or None
    @param bits_per_symbol: E.g., BPSK -> 1, QPSK -> 2, 8-PSK -> 3
    @type bits_per_symbol: integer >= 1
    @param samples_per_symbol: samples/baud (aka samples/symbol)
    @type samples_per_symbol: number or None
    @param interp_rate: USRP interpolation factor
    @type interp_rate: integer or None
    @param converter_rate: converter sample rate in Hz
    @type converter_rate: number
    @param possible_interps: a list of possible rates
    @type possible_interps: a list of integers
    @returns tuple (bitrate, samples_per_symbol, interp_rate)
    """
    print "Requested TX Bitrate:", bitrate and eng_notation.num_to_str(bitrate) or 'Auto',
    return _pick_bitrate(bitrate, bits_per_symbol, samples_per_symbol,
                   interp_rate, converter_rate, possible_interps, _gen_tx_info)
```

```
def pick_rx_bitrate(bitrate, bits_per_symbol, samples_per_symbol,
                    decim_rate, converter_rate, possible_decims):
    """
    Given the 4 input parameters, return at configuration that matches

    @param bitrate: desired bitrate or None
    @type bitrate: number or None
    @param bits_per_symbol: E.g., BPSK -> 1, QPSK -> 2, 8-PSK -> 3
    @type bits_per_symbol: integer >= 1
    @param samples_per_symbol: samples/baud (aka samples/symbol)
    @type samples_per_symbol: number or None
    @param decim_rate: USRP decimation factor
    @type decim_rate: integer or None
    @param converter_rate: converter sample rate in Hz
    @type converter_rate: number
    @param possible_decims: a list of possible rates
    @type possible_decims: a list of integers

    @returns tuple (bitrate, samples_per_symbol, decim_rate)
    """
    print "Requested RX Bitrate:", bitrate and eng_notation.num_to_str(bitrate) or 'Auto'
    return _pick_bitrate(bitrate, bits_per_symbol, samples_per_symbol,
                         decim_rate, converter_rate, possible_decims, _gen_rx_info)
```

C.6 Source Code for psk.py

```
#
# Copyright 2005,2006 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 math import pi, sqrt, log10
import math, cmath

# The following algorithm generates Gray coded constellations for M-PSK for M=[2,4,8]
def make_gray_constellation(m):
    # number of bits/symbol (log2(M))
    k = int(log10(m) / log10(2.0))

    coeff = 1
    const_map = []
    bits = [0]*3
```

```
    for i in range(m):
        # get a vector of the k bits to use in this mapping
        bits[3-k:3] = [((i&(0x01 << k-j-1)) >> k-j-1) for j in range(k)]

        theta = -(2*bits[0]-1)*(2*pi/m)*(bits[0]+abs(bits[1]-bits[2])+2*bits[1])
        re = math.cos(theta)
        im = math.sin(theta)
        const_map.append(complex(re, im))   # plug it into the constellation

    # return the constellation; by default, it is normalized
    return const_map

# This makes a constellation that increments around the unit circle
def make_constellation(m):
    return [cmath.exp(i * 2 * pi / m * 1j) for i in range(m)]

# Common definition of constellations for Tx and Rx
constellation = {
    2 : make_constellation(2),          # BPSK
    4 : make_constellation(4),          # QPSK
    8 : make_constellation(8)           # 8PSK
    }

gray_constellation = {
    2 : make_gray_constellation(2),          # BPSK
    4 : make_gray_constellation(4),          # QPSK
    8 : make_gray_constellation(8)           # 8PSK
    }
```

```
# -----------------------
# Do Gray code
# -----------------------
# binary to gray coding -- constellation does Gray coding
binary_to_gray = {
    2 : range(2),
    4 : [0,1,3,2],
    8 : [0, 1, 3, 2, 7, 6, 4, 5]
    }

# gray to binary
gray_to_binary = {
    2 : range(2),
    4 : [0,1,3,2],
    8 : [0, 1, 3, 2, 6, 7, 5, 4]
    }

# -----------------------
# Don't Gray code
# -----------------------
# identity mapping
binary_to_ungray = {
    2 : range(2),
    4 : range(4),
    8 : range(8)
    }

# identity mapping
ungray_to_binary = {
```

```
2 : range(2),
4 : range(4),
8 : range(8)
}
```

C.7 Source Code for receive_path.py

```
#!/usr/bin/env python
#
# Copyright 2005,2006,2007 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, blks2
from gnuradio import eng_notation
import copy
import sys

# ///////////////////////////////////////////////////////////////////////////
#                              receive path
# ///////////////////////////////////////////////////////////////////////////

class receive_path(gr.hier_block2):
    def __init__(self, demod_class, rx_callback, options):
        gr.hier_block2.__init__(self, "receive_path",
                                gr.io_signature(1, 1, gr.sizeof_gr_complex), # Input
signature
                                gr.io_signature(0, 0, 0))           # Output signature

        options = copy.copy(options)    # make a copy so we can destructively modify

        self._verbose            = options.verbose
        self._bitrate            = options.bitrate         # desired bit rate
        self._samples_per_symbol = options.samples_per_symbol  # desired
samples/symbol

        self._rx_callback   = rx_callback      # this callback is fired when there's a packet
available
        self._demod_class   = demod_class      # the demodulator_class we're using

        # Get demod_kwargs
```

```
demod_kwargs = self._demod_class.extract_kwargs_from_options(options)

# Design filter to get actual channel we want
sw_decim = 1
chan_coeffs = gr.firdes.low_pass (1.0,                  # gain
                                  sw_decim * self._samples_per_symbol, # sampling rate
                                  1.0,                  # midpoint of trans. band
                                  0.5,                  # width of trans. band
                                  gr.firdes.WIN_HANN)   # filter type
self.channel_filter = gr.fft_filter_ccc(sw_decim, chan_coeffs)

# receiver
self.packet_receiver = \
   blks2.demod_pkts(self._demod_class(**demod_kwargs),
                    access_code=None,
                    callback=self._rx_callback,
                    threshold=-1)

# Carrier Sensing Blocks
alpha = 0.001
thresh = 30   # in dB, will have to adjust
self.probe = gr.probe_avg_mag_sqrd_c(thresh,alpha)

# Display some information about the setup
if self._verbose:
   self._print_verbage()

   # connect block input to channel filter
   self.connect(self, self.channel_filter)
```

```
        # connect the channel input filter to the carrier power detector
        self.connect(self.channel_filter, self.probe)

        # connect channel filter to the packet receiver
        self.connect(self.channel_filter, self.packet_receiver)

    def bitrate(self):
        return self._bitrate

    def samples_per_symbol(self):
        return self._samples_per_symbol

    def carrier_sensed(self):
        """
        Return True if we think carrier is present.
        """
        #return self.probe.level() > X
        return self.probe.unmuted()

    def carrier_threshold(self):
        """
        Return current setting in dB.
        """
        return self.probe.threshold()

    def set_carrier_threshold(self, threshold_in_db):
        """
        Set carrier threshold.
```

```
        @param threshold_in_db: set detection threshold
        @type threshold_in_db:  float (dB) """
        self.probe.set_threshold(threshold_in_db)


    def add_options(normal, expert):
        """
        Adds receiver-specific options to the Options Parser
        """
        if not normal.has_option("--bitrate"):
            normal.add_option("-r", "--bitrate", type="eng_float", default=100e3,
                              help="specify bitrate [default=%default].")
        normal.add_option("-v", "--verbose", action="store_true", default=False)
        expert.add_option("-S", "--samples-per-symbol", type="int", default=2,
                          help="set samples/symbol [default=%default]")
        expert.add_option("", "--log", action="store_true", default=False,
                          help="Log all parts of flow graph to files (CAUTION: lots of data)")

    # Make a static method to call before instantiation
    add_options = staticmethod(add_options)

    def _print_verbage(self):
        """
        Prints information about the receive path
        """
        print "\nReceive Path:"
        print "modulation:      %s"    % (self._demod_class.__name__)
```

```
    print "bitrate:        %sb/s" % (eng_notation.num_to_str(self._bitrate))
    print "samples/symbol:  %3d"   % (self._samples_per_symbol)
```

C.8 Source Code for transmit_path.py

```
#
# Copyright 2005, 2006, 2007 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, blks2
from gnuradio import eng_notation

import copy
```

```
import sys

# /////////////////////////////////////////////////////////////////////////////
#                        transmit path
# /////////////////////////////////////////////////////////////////////////////

class transmit_path(gr.hier_block2):
    def __init__(self, modulator_class, options):
      '''
      See below for what options should hold
      '''
     gr.hier_block2.__init__(self, "transmit_path",
                          gr.io_signature(0, 0, 0),                  # Input signature
                          gr.io_signature(1, 1, gr.sizeof_gr_complex)) # Output signature

      options = copy.copy(options)    # make a copy so we can destructively modify

      self._verbose           = options.verbose
      self._tx_amplitude      = options.tx_amplitude    # digital amplitude sent to USRP
      self._bitrate           = options.bitrate         # desired bit rate
      self._samples_per_symbol = options.samples_per_symbol  # desired samples/baud

      self._modulator_class = modulator_class         # the modulator_class we are using
      # Get mod_kwargs
      mod_kwargs = self._modulator_class.extract_kwargs_from_options(options)

      # transmitter
     modulator = self._modulator_class(**mod_kwargs)
      self.packet_transmitter = \
```

```
        blks2.mod_pkts(modulator,
                       access_code=None,
                       msgq_limit=4,
                       pad_for_usrp=True)

    self.amp = gr.multiply_const_cc(1)
    self.set_tx_amplitude(self._tx_amplitude)

    # Display some information about the setup
    if self._verbose:
        self._print_verbage()

    # Connect components in the flowgraph
    self.connect(self.packet_transmitter, self.amp, self)

def set_tx_amplitude(self, ampl):
    """
    Sets the transmit amplitude sent to the USRP in volts
    @param: ampl 0 <= ampl < 1.
    """
    self._tx_amplitude = max(0.0, min(ampl, 1))
    self.amp.set_k(self._tx_amplitude)

def send_pkt(self, payload='', eof=False):
    """
    Calls the transmitter method to send a packet
    """
    return self.packet_transmitter.send_pkt(payload, eof)
```

```
    def bitrate(self):
        return self._bitrate

    def samples_per_symbol(self):
        return self._samples_per_symbol

    def add_options(normal, expert):
        """
        Adds transmitter-specific options to the Options Parser
        """
        if not normal.has_option('--bitrate'):
            normal.add_option("-r", "--bitrate", type="eng_float", default=100e3,
                              help="specify bitrate [default=%default].")
        normal.add_option("", "--tx-amplitude", type="eng_float", default=0.250,
metavar="AMPL",
                          help="set transmitter digital amplitude: 0 <= AMPL < 1
[default=%default]")
        normal.add_option("-v", "--verbose", action="store_true", default=False)

        expert.add_option("-S", "--samples-per-symbol", type="int", default=2,
                          help="set samples/symbol [default=%default]")
        expert.add_option("", "--log", action="store_true", default=False,
                          help="Log all parts of flow graph to file (CAUTION: lots of data)")

    # Make a static method to call before instantiation
    add_options = staticmethod(add_options)

    def _print_verbage(self):
        """
```

```
        Prints information about the transmit path
        """
        print "Tx amplitude     %s"    % (self._tx_amplitude)
        print "modulation:      %s"    % (self._modulator_class.__name__)
        print "bitrate:         %sb/s" % (eng_notation.num_to_str(self._bitrate))
        print "samples/symbol:  %3d"   % (self._samples_per_symbol)
```

C.9 Source Code for usrp_options.py

```
#
# Copyright 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.
#
```

```
_parser_to_groups_dict = dict()
class _parser_groups(object):
    def __init__(self, parser):
        self.usrpx_grp = parser.add_option_group("General USRP Options")
        self.usrp1_grp = parser.add_option_group("USRP1 Specific Options")
        self.usrp1exp_grp = parser.add_option_group("USRP1 Expert Options")
        self.usrp2_grp = parser.add_option_group("USRP2 Specific Options")

import generic_usrp

def _add_options(parser):
    """
    Add options to manually choose between usrp or usrp2.
    Add options for usb. Add options common to source and sink.
    @param parser: instance of OptionParser
    @return the parser group
    """
    #cache groups so they dont get added twice on tranceiver apps
    if not _parser_to_groups_dict.has_key(parser): _parser_to_groups_dict[parser] =
_parser_groups(parser)
    pg = _parser_to_groups_dict[parser]
    #pick usrp or usrp2
    pg.usrpx_grp.add_option("-u", "--usrpx", type="string", default=None,
                  help="specify which usrp model: 1 for USRP, 2 for USRP2
[default=auto]")
    #fast usb options
    pg.usrp1exp_grp.add_option("-B", "--fusb-block-size", type="int", default=0,
                  help="specify fast usb block size [default=%default]")
```

```
    pg.usrp1exp_grp.add_option("-N", "--fusb-nblocks", type="int", default=0,
                  help="specify number of fast usb blocks [default=%default]")
    #lo offset
    pg.usrpx_grp.add_option("--lo-offset", type="eng_float", default=None,
                  help="set LO Offset in Hz [default=automatic].")
    #usrp options
    pg.usrp1_grp.add_option("-w", "--which", type="int", default=0,
                  help="select USRP board [default=%default]")
    #usrp2 options
    pg.usrp2_grp.add_option("-e", "--interface", type="string", default="eth0",
                  help="Use USRP2 at specified Ethernet interface [default=%default]")
    pg.usrp2_grp.add_option("-a", "--mac-addr", type="string", default="",
                  help="Use USRP2 at specified MAC address [default=None]")
    return pg

def add_rx_options(parser):
    """
    Add receive specific usrp options.
    @param parser: instance of OptionParser
    """
    pg = _add_options(parser)
    pg.usrp1_grp.add_option("-R", "--rx-subdev-spec", type="subdev", default=None,
                  help="select USRP Rx side A or B")
    pg.usrpx_grp.add_option("--rx-gain", type="eng_float", default=None,
metavar="GAIN",
                  help="set receiver gain in dB [default=midpoint].  See also --show-rx-gain-
range")
    pg.usrpx_grp.add_option("--show-rx-gain-range", action="store_true", default=False,
                  help="print min and max Rx gain available on selected daughterboard")
```

```
    pg.usrpx_grp.add_option("-d", "--decim", type="intx", default=None,
                  help="set fpga decimation rate to DECIM [default=%default]")

def create_usrp_source(options):
    u = generic_usrp.generic_usrp_source_c(
        usrpx=options.usrpx,
        which=options.which,
        subdev_spec=options.rx_subdev_spec,
        interface=options.interface,
        mac_addr=options.mac_addr,
        fusb_block_size=options.fusb_block_size,
        fusb_nblocks=options.fusb_nblocks,
        lo_offset=options.lo_offset,
        gain=options.rx_gain,
    )
    if options.show_rx_gain_range:
        print "Rx Gain Range: minimum = %g, maximum = %g, step size =
%g"%tuple(u.gain_range())
    return u

def add_tx_options(parser):
    """
    Add transmit specific usrp options.
    @param parser: instance of OptionParser
    """
    pg = _add_options(parser)
    pg.usrp1_grp.add_option("-T", "--tx-subdev-spec", type="subdev", default=None,
                  help="select USRP Rx side A or B")
```

```
    pg.usrpx_grp.add_option("--tx-gain", type="eng_float", default=None,
metavar="GAIN",
                      help="set transmitter gain in dB [default=midpoint].  See also --show-tx-
gain-range")
    pg.usrpx_grp.add_option("--show-tx-gain-range", action="store_true", default=False,
                      help="print min and max Tx gain available on selected daughterboard")
    pg.usrpx_grp.add_option("-i", "--interp", type="intx", default=None,
                      help="set fpga interpolation rate to INTERP [default=%default]")

def create_usrp_sink(options):
    u = generic_usrp.generic_usrp_sink_c(
        usrpx=options.usrpx,
        which=options.which,
        subdev_spec=options.tx_subdev_spec,
        interface=options.interface,
        mac_addr=options.mac_addr,
        fusb_block_size=options.fusb_block_size,
        fusb_nblocks=options.fusb_nblocks,
        lo_offset=options.lo_offset,
        gain=options.tx_gain,
    )
    if options.show_tx_gain_range:
        print "Tx Gain Range: minimum = %g, maximum = %g, step size =
%g"%tuple(u.gain_range())
    return u
```

C.10 Source Code for usrp_receive_path.py

```
#
```

```
# Copyright 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
import usrp_options
import receive_path
from pick_bitrate import pick_rx_bitrate
from gnuradio import eng_notation

def add_freq_option(parser):
    """
    Hackery that has the -f / --freq option set both tx_freq and rx_freq
```

```
    """
    def freq_callback(option, opt_str, value, parser):
        parser.values.rx_freq = value
        parser.values.tx_freq = value

    if not parser.has_option('--freq'):
        parser.add_option('-f', '--freq', type="eng_float",
                          action="callback", callback=freq_callback,
                          help="set Tx and/or Rx frequency to FREQ [default=%default]",
                          metavar="FREQ")

def add_options(parser, expert): add_freq_option(parser)
    usrp_options.add_rx_options(parser)
    receive_path.receive_path.add_options(parser, expert)
    expert.add_option("", "--rx-freq", type="eng_float", default=None,
                          help="set Rx frequency to FREQ [default=%default]",
metavar="FREQ")
    parser.add_option("-v", "--verbose", action="store_true", default=False)

class usrp_receive_path(gr.hier_block2):

    def __init__(self, demod_class, rx_callback, options):
        '''
        See below for what options should hold
        '''
        gr.hier_block2.__init__(self, "usrp_receive_path",
                gr.io_signature(0, 0, 0),                    # Input signature
                gr.io_signature(0, 0, 0)) # Output signature
```

```
        if options.rx_freq is None:
            sys.stderr.write("-f FREQ or --freq FREQ or --rx-freq FREQ must be
specified\n")
            raise SystemExit
        rx_path = receive_path.receive_path(demod_class, rx_callback, options)
        for attr in dir(rx_path): #forward the methods
            if not attr.startswith('_') and not hasattr(self, attr):
                setattr(self, attr, getattr(rx_path, attr))
        #setup usrp
        self._demod_class = demod_class
        self._setup_usrp_source(options)
        #connect
        self.connect(self.u, rx_path)

    def _setup_usrp_source(self, options):
        self.u = usrp_options.create_usrp_source(options)
        adc_rate = self.u.adc_rate()
        if options.verbose:
            print 'USRP Source:', self.u
        (self._bitrate, self._samples_per_symbol, self._decim) = \
                pick_rx_bitrate(options.bitrate, self._demod_class.bits_per_symbol(), \
                                options.samples_per_symbol, options.decim, adc_rate,  \
                                self.u.get_decim_rates())

        self.u.set_decim(self._decim)

        if not self.u.set_center_freq(options.rx_freq):
            print "Failed to set Rx frequency to %s" %
(eng_notation.num_to_str(options.rx_freq))
```

```
        raise ValueError, eng_notation.num_to_str(options.rx_freq)
```

C.11 Source Code for usrp_tramsmit_path.py

```
#
# Copyright 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
import usrp_options
import transmit_path
from pick_bitrate import pick_tx_bitrate
from gnuradio import eng_notation
```

```
def add_freq_option(parser):
    """
    Hackery that has the -f / --freq option set both tx_freq and rx_freq
    """
    def freq_callback(option, opt_str, value, parser):
        parser.values.rx_freq = value
        parser.values.tx_freq = value

    if not parser.has_option('--freq'):
        parser.add_option('-f', '--freq', type="eng_float",
                          action="callback", callback=freq_callback,
                          help="set Tx and/or Rx frequency to FREQ [default=%default]",
                          metavar="FREQ")

def add_options(parser, expert): add_freq_option(parser)
    usrp_options.add_tx_options(parser)
    transmit_path.transmit_path.add_options(parser, expert)
    expert.add_option("", "--tx-freq", type="eng_float", default=None,
                          help="set transmit frequency to FREQ [default=%default]",
metavar="FREQ")
    parser.add_option("-v", "--verbose", action="store_true", default=False)

class usrp_transmit_path(gr.hier_block2):
    def __init__(self, modulator_class, options):
        '''
        See below for what options should hold
        '''
```

```
        gr.hier_block2.__init__(self, "usrp_transmit_path",
                gr.io_signature(0, 0, 0),                    # Input signature
                gr.io_signature(0, 0, 0)) # Output signature
        if options.tx_freq is None:
            sys.stderr.write("-f FREQ or --freq FREQ or --tx-freq FREQ must be
specified\n")
            raise SystemExit
        tx_path = transmit_path.transmit_path(modulator_class, options)
        for attr in dir(tx_path): #forward the methods
            if not attr.startswith('_') and not hasattr(self, attr):
                setattr(self, attr, getattr(tx_path, attr))
        #setup usrp
        self._modulator_class = modulator_class
        self._setup_usrp_sink(options)
        #connect
        self.connect(tx_path, self.u)

    def _setup_usrp_sink(self, options):
        """
        Creates a USRP sink, determines the settings for best bitrate,
        and attaches to the transmitter's subdevice.
        """
        self.u = usrp_options.create_usrp_sink(options)
        dac_rate = self.u.dac_rate()
        if options.verbose:
            print 'USRP Sink:', self.u
        (self._bitrate, self._samples_per_symbol, self._interp) = \
                    pick_tx_bitrate(options.bitrate, self._modulator_class.bits_per_symbol(), \
                                options.samples_per_symbol, options.interp, dac_rate, \
```

```
                        self.u.get_interp_rates())

        self.u.set_interp(self._interp)
        self.u.set_auto_tr(True)

        if not self.u.set_center_freq(options.tx_freq):
            print "Failed to set Rx frequency to %s" %
(eng_notation.num_to_str(options.tx_freq))
            raise ValueError, eng_notation.num_to_str(options.tx_freq)
```

C.12 Source Code for usrp_wfm_rcv.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, eng_notation, optfir
from gnuradio import audio
from gnuradio import usrp
from gnuradio import blks2
from gnuradio.eng_option import eng_option
from gnuradio.wxgui import slider, powermate
from gnuradio.wxgui import stdgui2, fftsink2, form
from optparse import OptionParser
from usrpm import usrp_dbid
import sys
import math
import wx

def pick_subdevice(u):
    """
    The user didn't specify a subdevice on the command line.
    Try for one of these, in order: TV_RX, BASIC_RX, whatever is on side A.

    @return a subdev_spec
    """
    return usrp.pick_subdev(u, (usrp_dbid.TV_RX,
                          usrp_dbid.TV_RX_REV_2,
                                  usrp_dbid.TV_RX_REV_3,
```

```
                    usrp_dbid.TV_RX_MIMO,
                usrp_dbid.TV_RX_REV_2_MIMO,
                    usrp_dbid.TV_RX_REV_3_MIMO,
                usrp_dbid.BASIC_RX))


class wfm_rx_block (stdgui2.std_top_block):
    def __init__(self,frame,panel,vbox,argv):
        stdgui2.std_top_block.__init__ (self,frame,panel,vbox,argv)

        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=A)")
        parser.add_option("-f", "--freq", type="eng_float", default=100.1e6,
                          help="set frequency to FREQ", metavar="FREQ")
        parser.add_option("-g", "--gain", type="eng_float", default=40,
                          help="set gain in dB (default is midpoint)")
        parser.add_option("-V", "--volume", type="eng_float", default=None,
                          help="set volume (default is midpoint)")
        parser.add_option("-O", "--audio-output", type="string", default="",
                          help="pcm device name.  E.g., hw:0,0 or surround51 or /dev/dsp")

        (options, args) = parser.parse_args()
        if len(args) != 0:
            parser.print_help()
            sys.exit(1)

        self.frame = frame
        self.panel = panel
```

```
self.vol = 0
self.state = "FREQ"
self.freq = 0

# build graph

self.u = usrp.source_c()                    # usrp is data source

adc_rate = self.u.adc_rate()                # 64 MS/s
usrp_decim = 150
self.u.set_decim_rate(usrp_decim)
usrp_rate = adc_rate / usrp_decim           # 320 kS/s
chanfilt_decim = 1
demod_rate = usrp_rate / chanfilt_decim
audio_decimation = 10
audio_rate = demod_rate / audio_decimation  # 32 kHz

if options.rx_subdev_spec is None:
    options.rx_subdev_spec = pick_subdevice(self.u)

self.u.set_mux(usrp.determine_rx_mux_value(self.u, options.rx_subdev_spec))
self.subdev = usrp.selected_subdev(self.u, options.rx_subdev_spec)
print "Using RX d'board %s" % (self.subdev.side_and_name(),)
dbid = self.subdev.dbid()
if not (dbid == usrp_dbid.BASIC_RX or
        dbid == usrp_dbid.TV_RX or
        dbid == usrp_dbid.TV_RX_REV_2 or
        dbid == usrp_dbid.TV_RX_REV_3 or
```

```
        dbid == usrp_dbid.TV_RX_MIMO or
        dbid == usrp_dbid.TV_RX_REV_2_MIMO or
        dbid == usrp_dbid.TV_RX_REV_3_MIMO
):
    print "This daughterboard does not cover the required frequency range"
    print "for this application.  Please use a BasicRX or TVRX daughterboard."
    raw_input("Press ENTER to continue anyway, or Ctrl-C to exit.")

chan_filt_coeffs = optfir.low_pass (1,          # gain
                                    usrp_rate,   # sampling rate
                                    80e3,        # passband cutoff
                                    115e3,       # stopband cutoff
                                    0.1,         # passband ripple
                                    60)          # stopband attenuation
#print len(chan_filt_coeffs)
chan_filt = gr.fir_filter_ccf (chanfilt_decim, chan_filt_coeffs)

self.guts = blks2.wfm_rcv (demod_rate, audio_decimation)

self.volume_control = gr.multiply_const_ff(self.vol)

# sound card as final sink
audio_sink = audio.sink (int (audio_rate),
                         options.audio_output,
                         False)  # ok_to_block

# now wire it all together
self.connect (self.u, chan_filt, self.guts, self.volume_control, audio_sink)
```

```
    self._build_gui(vbox, usrp_rate, demod_rate, audio_rate)

    if options.gain is None:
        # if no gain was specified, use the mid-point in dB
        g = self.subdev.gain_range()
        options.gain = float(g[0]+g[1])/2

    if options.volume is None: g =
        self.volume_range() options.volume
        = float(g[0]+g[1])/2

    if abs(options.freq) < 1e6:
        options.freq *= 1e6

    # set initial values

    self.set_gain(options.gain)
    self.set_vol(options.volume)
    if not(self.set_freq(options.freq)):
        self._set_status_msg("Failed to set initial frequency")


def _set_status_msg(self, msg, which=0):
    self.frame.GetStatusBar().SetStatusText(msg, which)
def _build_gui(self, vbox, usrp_rate, demod_rate, audio_rate):

    def _form_set_freq(kv):
        return self.set_freq(kv['freq'])
```

```
if 1:
    self.src_fft = fftsink2.fft_sink_c(self.panel, title="Data from USRP",
                                       fft_size=512, sample_rate=usrp_rate,
                                                    ref_scale=32768.0, ref_level=0, y_divs=12)
    self.connect (self.u, self.src_fft)
    vbox.Add (self.src_fft.win, 4, wx.EXPAND)

if 1:
    post_filt_fft = fftsink2.fft_sink_f(self.panel, title="Post Demod",
                                        fft_size=1024, sample_rate=usrp_rate,
                                        y_per_div=10, ref_level=0)
    self.connect (self.guts.fm_demod, post_filt_fft)
    vbox.Add (post_filt_fft.win, 4, wx.EXPAND)

if 0:
    post_deemph_fft = fftsink2.fft_sink_f(self.panel, title="Post Deemph",
                                          fft_size=512, sample_rate=audio_rate,
                                          y_per_div=10, ref_level=-20)
    self.connect (self.guts.deemph, post_deemph_fft)
    vbox.Add (post_deemph_fft.win, 4, wx.EXPAND)


# control area form at bottom
self.myform = myform = form.form()

hbox = wx.BoxSizer(wx.HORIZONTAL)
hbox.Add((5,0), 0)
myform['freq'] = form.float_field(
```

```
    parent=self.panel, sizer=hbox, label="Freq", weight=1,
    callback=myform.check_input_and_call(_form_set_freq, self._set_status_msg))

hbox.Add((5,0), 0)
myform['freq_slider'] = \
    form.quantized_slider_field(parent=self.panel, sizer=hbox, weight=3,
                                range=(87.9e6, 108.1e6, 0.1e6),
                                callback=self.set_freq)
hbox.Add((5,0), 0)
vbox.Add(hbox, 0, wx.EXPAND)

hbox = wx.BoxSizer(wx.HORIZONTAL)
hbox.Add((5,0), 0)

myform['volume'] = \
    form.quantized_slider_field(parent=self.panel, sizer=hbox, label="Volume",
                                weight=3, range=self.volume_range(),
                                callback=self.set_vol)
hbox.Add((5,0), 1)

myform['gain'] = \
    form.quantized_slider_field(parent=self.panel, sizer=hbox, label="Gain",
                                weight=3, range=self.subdev.gain_range(),
                                callback=self.set_gain)
hbox.Add((5,0), 0)
vbox.Add(hbox, 0, wx.EXPAND)

try:
    self.knob = powermate.powermate(self.frame)
```

```
        self.rot = 0
        powermate.EVT_POWERMATE_ROTATE (self.frame, self.on_rotate)
        powermate.EVT_POWERMATE_BUTTON (self.frame, self.on_button)
    except:
        print "FYI: No Powermate or Contour Knob found"


def on_rotate (self, event):
    self.rot += event.delta
    if (self.state == "FREQ"):
        if self.rot >= 3:
            self.set_freq(self.freq + .1e6)
            self.rot -= 3
        elif self.rot <=-3:
            self.set_freq(self.freq - .1e6)
            self.rot += 3
    else:
        step = self.volume_range()[2]
        if self.rot >= 3:
            self.set_vol(self.vol + step)
            self.rot -= 3
        elif self.rot <=-3:
            self.set_vol(self.vol - step)
            self.rot += 3
def on_button (self, event):
    if event.value == 0:        # button up
        return
    self.rot = 0
    if self.state == "FREQ":
```

```
            self.state = "VOL"
        else:
            self.state = "FREQ"
        self.update_status_bar ()


    def set_vol (self, vol):
        g = self.volume_range()
        self.vol = max(g[0], min(g[1], vol))
        self.volume_control.set_k(10**(self.vol/10))
        self.myform['volume'].set_value(self.vol)
        self.update_status_bar ()

    def set_freq(self, target_freq):
        """
        Set the center frequency we're interested in.

        @param target_freq: frequency in Hz
        @rypte: bool

        Tuning is a two step process.  First we ask the front-end to
        tune as close to the desired frequency as it can.  Then we use
        the result of that operation and our target_frequency to
        determine the value for the digital down converter.
        """
        r = usrp.tune(self.u, 0, self.subdev, target_freq)

        if r:
            self.freq = target_freq
```

```
            self.myform['freq'].set_value(target_freq)        # update displayed value
            self.myform['freq_slider'].set_value(target_freq)  # update displayed value
            self.update_status_bar()
            self._set_status_msg("OK", 0)
            return True

        self._set_status_msg("Failed", 0)
        return False

    def set_gain(self, gain):
        self.myform['gain'].set_value(gain)     # update displayed value
        self.subdev.set_gain(gain)

    def update_status_bar (self):
        msg = "Volume:%r  Setting:%s" % (self.vol, self.state)
        self._set_status_msg(msg, 1)
        self.src_fft.set_baseband_freq(self.freq)

    def volume_range(self):
        return (-20.0, 0.0, 0.5)

if __name__ == '__main__':
    app = stdgui2.stdapp (wfm_rx_block, "USRP WFM RX")
    app.MainLoop ()
```

APPENDIX D

WAVEFORM DEVELOPMENT GUIDE COMPONENT DESCRIPTIONS [6]

D.1 am_demod:

(a) Description: Demodulator for amplitude modulated signals

(b) Properties: None

(c) Interfaces:

-- Provides: Rx_In_from_USRP_or_Decimator (complexShort)

-- Uses: Out_to_sound_card (complexShort)

(d) Additional Notes: Component names don't start with AM_ because this is interpreted as the start of an automake macro

D.2 amplifier

(a) Description: Fixed gain amplifier for I and Q channels

(b) Properties:

-- I_gain; float; Gain for the I channel in linear units

-- Q_gain; float; Gain for the Q channel in linear units

(c) Interfaces:

-- Provides: dataIn (complexShort)

-- Uses: dataOut (complexShort)

D.3 AutomaticGainControl

(a) Description: Automatic Gain Control

(b) Properties:

-- energy_lo; float; Low energy threshold

-- energy_hi; float; High energy threshold

-- k_attack; float; Attack time constant

-- k_release; float; Release time constant

-- g_min; float; Minimum gain value

-- g_max; float; Maximum gain value

-- rssi_pass; float; Received strength level above which data will be passed

dInterfaces

(c) Interfaces:

-- Provides: data_in (complexShort)

-- Uses: data_out (complexShort)

D.4 Channel

(a) Description: Simulates channels with varying complexity and effects

(b) Properties:

-- AWGN Noise Power; long; The power of the AWGN noise

-- Fading Type; string; Specifies the fading type. Valid values are 'Ricean' and 'None'

-- Envelope Fading Only; string; True if fading doesn't affect the signal phase, valid values are True and False

-- K Fading factor; double; The Ricean K factor, 0 implies Rayleigh fading

-- Max doppler rate; double; The maximum doppler rate as divided by the sampling rate

-- port_list; string; Returns a sequence of strings with the names of the available Provides ports

(c) Interfaces:

-- Provides: data_in (complexShort)

-- Uses: data_out (complexShort)

D.5 ChannelDemo

(a) Description: Simulates a very basic AWGN channel.

(b) Properties:

-- noise_std_dev; short; Standard deviation of noise

-- phase_offset; float; Phase offset in degrees

(c) Interfaces:

-- Provides: samples_in (complexShort)

-- Uses: samples_out (complexShort)

D.6 Conv_Dec

(a) Description: A convolutional decoder

(b) Properties:

-- rate_index; short; The index of the decoding rate from the supported rates table. For a custom rate use rate_index 0

-- mode; short; (Unknown description)

-- k; short; Input bits at a time (for the custom rate_index=0) with encoder rate k/n

-- K; short; Constraint length (for the custom rate_index=0)

-- n; short; Output bits at a time for the custom rate_index=0, has to match the number of supplied polynomials (rate_index=0). The encoder rate = k/n

-- generatorPolynomials; short; A list of generator polynomials in OCTAL for the custom rate_index=0

-- port_list; string; Returns a sequence of strings with the names of the available Provides ports

(c) Interfaces:

-- Provides: bits_to_dec_in (realChar)

-- Uses: decoded_bits (realChar)

D.7 Conv_Enc

(a) Description: A convolutional encoder

(b) Properties:

-- rate_index; short; The index of the decoding rate from the supported rates table. For a custom rate use rate_index 0

-- mode; short; (Unknown description)

-- k; short; Input bits at a time (for the custom rate_index=0) with encoder rate k/n

-- K; short; Constraint length (for the custom rate_index=0)

-- n; short; Output bits at a time for the custom rate_index=0, has to match the number of supplied polynomials (rate_index=0). The encoder rate = k/n

-- generatorPolynomials; short; A list of generator polynomials in OCTAL for the custom rate_index=0

-- port_list; string; Returns a sequence of strings with the names of the available * Interfaces:

-- Provides: bits_to_enc_in (realChar)

-- Uses: encoded_bits (realChar)

D.8 Decimator

(a) Description: Decimates the input signal

(b) Properties:

-- DecimateBy?; ushort; The decimation factor

-- filter; float; Filter coefficients for the decimator

-- Filter Type; string; Specifies the filter type, IIR or FIR. The IIR option uses auto-generated coefficients only.

(c) Interfaces:

-- Provides: inData (complexShort)

-- Uses: outData (complexShort)

D.9 DigitalDemodulator

(a) Description: A digital modulator

(b) Properties: ModScheme?; string; Type of demodulation scheme to use (BPSK, QPSK, 8PSK, 16QAM, 4PAM)

(c) Interfaces:

-- Provides: bitsIn (realChar)

-- Uses: symbolsOut (complexShort)

D.10 DigitalModulator

(a) Description: A digital modulator

(b) Properties: ModScheme?; string; Type of demodulation scheme to use (BPSK, QPSK, 8PSK, 16QAM, 4PAM)

(c) Interfaces:

-- Provides: bitsIn (realChar)

-- Uses: symbolsOut (complexShort)

D.11 FrameAssembler

(a) Description: Assembles frames for the modem

(b) Properties:

-- mod_type; string; Modulation type: BPSK, QPSK, 8PSK, 16QAM

-- FrameSizeOptionNumber?; ushort; Frame Size Option Number

-- FrameSizeOption?1; ushort; Frame Size for Option 1

-- FrameSizeOption?2; ushort Frame Size for Option 2

-- FrameSizeOption?3; ushort; Frame Size for Option 3

-- FrameSizeOption?4; ushort; Frame Size for Option 4

(c) Interfaces:

-- Provides: SymbolsIn? (complexShort)

-- Uses: FrameSymbolsOut? (complexShort)

D.12 Interpolator

(a) Description: Interpolates the input signal

(b) Properties:

-- InterpFactor? (k); ushort; Interpolation factor

-- filter; float; Interpolating filter coefficients

-- pulse_shape; string; Type of pulse shape to use for the filter prototype

-- m; ushort; Symbol Delay

-- beta; float; Excess bandwidth factor

(c) Interfaces:

-- Provides: inData (complexShort)

-- Uses: outData (complexShort)

D.13 JPEG_VideoViewer

(a) Description: Displays a video feed from a webcam

(b) Properties: None

(c) Interfaces:

-- Provides: JPEG_DataIn (realChar)

-- Uses: None

(d) Additional Notes: Related to the WebCamCapture? component

D.14 OSSIETalk

(a) Description: Audio capture with CVSD

(b) Properties: None

(c) Interfaces:

-- Provides: from_radio (realChar)

-- Uses: to_radio (realChar)

D.15 PacketResizer

(a) Description: Resizes packets

(b) Properties:

-- Packet Size; ulong; This is the output packet size. Input data will be buffered of broken apart to match the size.

-- port_list; string; Returns a sequence of strings with the names of the available Provides ports

(c) Interfaces:

-- Provides: packet_in (realChar)

-- Uses: packet_out (realChar)

D.16 pass_data

(a) Description: A pass-through component

(b) Properties: port_list; string; Returns a sequence of strings with the names of available Provides ports

(c) Interfaces:

-- Provides: cshort_in (complexShort)

-- Uses: cshort_out (complexShort)

-- Uses: send_timing_report (timingStatus)

D.17 readBytesfromFile

(a) Description: Reads bytes from an input file

(b) Properties: bufferLength; ulong; Define the output buffer length in bits

(c) Interfaces:

-- Provides: None

-- Uses: outputBits (realChar)

D.18 RxDemo

(a) Description: Compares

(b) Properties: None

(c) Interfaces:

-- Provides: symbols_in (complexShort)

-- Uses: None

(d) Additional Notes: This component is directly related to the TxDemo? component. Use is basically limited to the ossie_demo waveform.

D.19 SymbolSyncPoly

(a) Description: Symbol Synchronizer

(b) Properties:

-- pulse_shape; string; Type of pulse shape to use for matched filter

-- k; ushort; Samples per symbol

-- m; ushort; Symbol delay

-- beta; float; Excess bandwidth factor

-- Npfb; ushort; Number of filters in bank

(c) Interfaces:

-- Provides: baseband_in (complexShort)

-- Uses: symbols_out (complexShort)

D.20 TxDemo

(a) Description: Generates test symbols

(b) Properties: packet_delay_ms; short; Delay between generated packets (milliseconds)

(c) Interfaces:

-- Provides: None

-- Uses: symbols_out (complexShort)

(d) Additional Notes: Directly related to the RxDemo? component.

D.21 USRP_Commander

(a) Description: A controller for the USRP1 hardware

(b) Properties:

-- rx_freq; float; Receiver frequency

-- tx_freq; float; Transmitter frequency

-- tx_interp; short; Transmitter interpolation factor

-- rx_decim; float; Receiver decimation factor

-- rx_size; ulong; Receiver data packet size

-- rx_gain; float; Receiver gain

-- rx_gain_max; short; If 1, sets rx_gain to max and rx_gain property is ignored.

-- tx_start; short; Start the transmitter when the component starts

-- rx_start; short; Start the receiver when the component starts

(c) Interfaces:

-- Provides: None

-- Uses: TX_Control (TX_Control)

-- Uses: RX_Control (RX_Control)

-- Uses: Data_Control (Resource)

D.22 WebCamCapture

(a) Description: Captures a video stream over a webcam

(b) Properties: quality; short; Quality of JPEG (between 1 and 5)

(c) Interfaces:

-- Provides: None

-- Uses: JPEG_DataOut (realChar)

(d) Additional Notes: Related to JPEG_VideoViewer

D.23 WFMDemod

(a) Description: FM Demodulator

(b) Properties: port_list; string; Returns a sequence of strings with the names of the available Provides ports

(c) Interfaces:

-- Provides: dataIn (complexShort)

-- Uses: dataOut (complexShort)

D.24 writeBytestoFile

(a) Description: Writes bytes to an output file

(b) Properties: port_list; string; Returns a sequence of strings with the names of the available Provides ports

(c) Interfaces:

-- Provides: inputBits (realChar)

-- Uses: None

Table D.1. Device Descriptions

Device Name	Name in platform	Description	Interface Data Types	Properties Listing
GPP	GPP	General Purpose Processor, Executable Device	N/A	None
soundCardCapture	Sound_in	ALSA Sound Card Microphone Capture	AudioPort?, complexShort	sample_rate
soundCardPlayback	Sound_out	ALSA Sound Card Speaker Playback	AudioPort?, complexShort	sample_rate
USRP	USRP	Ettus Research USRP 1	complexShort	None
USRP2	USRP2	Ettus Research USRP 2	complexShort	None
XilinxFPGA	XilinxFPGA	Xilinx FPGA Abstraction, Loadable Device	N/A	None

REFERENCE LIST

[1] Debian Installer team. *Ubuntu 10.04 LTS Ubuntu Installation Guide*: 2004.

[2] Ubuntu. URL http://www.ubuntu.com/download/ubuntu/download. October 2011.

[3] *Building GNU Radio on Ubuntu Linux*. URL http://gnuradio.org/redmine/projects/gnuradio/wiki/UbuntuInstall. October 2011.

[4] Matt Carrick, Drew Cormier, Christopher Covington, Carl B. Dietrich, Joseph Gaeddert, Benjamin Hilburn, C. Ian Phelps, Shereef Sayed, Deepan Seeralan, Jason Snyder, Haris Volos. *OSSIE 0.8.2 Installation and User Guide*. April 2011

[5] OSSIE Labs. URL http://ossie.wireless.vt.edu/download/labs/. October 2011.

[6] *Waveform Development Guide (under development).* URL http://ossie.wireless.vt.edu/trac/wiki/WaveformDevelopmentGuide. October 2011.

[7] GNU Radio Mailing Lists. October 2011. URL http://www.gnu.org/software/gnuradio/mailinglists.html.

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

[9] Eric Blossom. *Exploring GNU Radio*. GNU Radio, November 2004.

[10] OSSIE. http://ossie.wireless.vt.edu/. October 2011.

[11] OSSIE Mailing Lists. http://listserv.vt.edu/cgi-bin/wa?A0=OSSIE-DISCUSS. October 2011

[12] Edoardo Paone. *Open-Source SCA Implementation-Embedded and Software Communication Architecture*. 2010.

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

[14] Wikipedia. http://en.wikipedia.org/. October 2011.

[15] Matt Ettus. *USRP family brochure*. Ettus Research. http://www.ettus.com.

[16] Michael Ihde, Philip Balister, and Shereef Sayed. *How should assembly controller start other components.* OSSIE-discuss forum, October 2011. URL http://listserv.vt.edu/archives/open-source.html.

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

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

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

[20] Constantine A. Balanis. *Antenna Theory Analysis and Design*. Wiley-Interscience, 3nd ed., April 4, 2005

[21] U.S. Department of Commerce, National Telecommunications and Information Administration Office of Spectrum Management. *United States Frequency Allocations. the Radio Spectrum*. October 2003.

[22] Krishna Sankar. *Coherent Demodulation of DBPSK*. URL http://www.dsplog.com/2007/09/30/coherent-demodulation-of-dbpsk/. Septermber 2007.

[23] Johnny Lee. *A Bidirectional Two-Hop Relay Network Using GNU Radio And USRP. Master's thesis*. Uiversity of North Texas, Denton, Texas. August 2011

[24] Ettus Research LLC. URL http://www.ettus.com/order. October 2011.

[25] Matt Ettus, Ettus Research LLC. *USRP User's and Developer's Guide*. URL http://matt@ettus.com. October 2011.

[26] WARP Home Page. URL http://warp.rice.edu/. October 2011