User's Manual for USRP BASED COGNITIVE RADIO Test Bed

A Practical Guide

August 2014

Wireless Communications Coding and Cognitive Radio Lab

IIT KANPUR

Table of Contents

HOW TO READ THE MANUAL 5
1. Introduction 6
2 .Prerequisites 7
2.1 .a Hardware Prerequisites 7
2.1. b Hardware Configuration 7
2.2 a Software Prerequisites 10
2.2. b Software Configuration 11
2.2.b. 1 UHD+GNU Radio installation 11
2.2 b.2 Python +SQL installation+ Other dependencies 11
2.2.b. 3 USRP Networking Setup 12
3. Unpacking of test bed frame work 13
3.1 Through various files and Scripts 13
3.2 Through classes and Functions 15
3.2. a Spectrum Sensing 15
3.2. b TX & RX 16
3.2.b. 1 Transmission of Data 16
3 .2.b. 2 Reception of Data 17
3.3 Classes and functions: A description 18
Spec_sense.py: Class: my_top_block - 18
Spec_sense.py: Function :main_loop – 18
Spec_sense.py: Class: tune – 18
Cog_tx.py: Class: my_top_block - 19
Cog_tx.py :Function:main - 19
Transmit_path.py: Class: transmit_path - 19
Receive _path.py: Class: receive_path - 19
Uhd_interface.py: Class: uhd_transmitter - 20
Uhd_interface.py: Class: uhd_receiver - 20

Sbs_class.py: Class: sbs_class - 20
Cog_tx.py: Function: best_freq - 20
Transmit_path.py: Function: send_pkt - 21
Receive_path.py: Function: packet_receiver.py - 21
Cog_tx.py: Function: phy_rx_callback - 21
Cog_tx.py: Function: os.write - 21
Cog_tx.py: Function: set_freq_sink - 22
4. Building a Cognitive Radio Test Bed 23
5. Codes 27
Cog_tx.py 36
uhd_interface.py 52
transmit_path.py 55
receive_path.py 59
sbs_class.py 64
usage_plot.py 74
6. Section II: SDR Implementation of SISO/MIMO based Cognitive Radio Test bed 78
7. Section II: Prerequisites 79
7. 1. a Section II: Additional Hardware prerequisites 79
7.1. b Section II: Hardware configuration 79
7.2 a Section II: Software Prerequisites 80
7.2 b Section II: Software configuration 81
7.2.b.1 INSTALLING UHD+GNURADIO 81
8. Section II: Unpacking of test bed framework 83
8.1 Files and Scripts 83
8.2 CLASSES AND FUNCTIONS 85
8.2.a Transmission of Data: Flowgraph 85
8.2 .b Transmission of Data: An exposition of classes & functions 86
8.2.b.1 main_loop:CR_TX 86
8.2.b.2 gui_thread:GUI_MAIN_FRAME 86
8.2.b.3 main_frame:GUI_MAIN_FRAME 86
8.2.b.4 tb_options_tx_gui: TB_OPTIONS 86
8.2.b.5 start_flowgraph: GLOBAL_CONTROL 86

8.2.b.6 top_block_tx_gui: TOP_BLOCK 87
8.2.b.7 mimo_transmit_path: FOI_MIMO_TRANSMIT_PATH 87
8.2.b.8 siso_transmit_path: FOI_SISO_TRANSMIT_PATH 87
8.2.b.8 new_freq_module: COORDINATION_MODULE 87
8.2.b.8 probe_for_spectrum: COORDINATION_MODULE 87
8.2.b.8 send_thread(run): SEND_THREAD 88
8.2.b.9 image_ mode(run): SEND_THREAD 88
8.2.b.10 file_ mode(run): SEND_THREAD 88
8.2.b.10 stream_ mode(run): SEND_THREAD 88
8.2.b.11 Spectral Database & Send_pkt 88
9. Section II: Building a Cognitive Radio Test Bed 89
10. Codes 93
10.1 CR_TX.py 93
10.2 COORDINATION_MODULE 97
10.3 SPEC_SENSE 100
10.4 TOP_BLOCK 109
10.5 SEND_THREAD 113
10.6 GUI_MAIN_FRAME_TX 119
10.7 GLOBAL_CONTROL_UNIT 121
10.8 TB_OPTIONS 123
10.9 CTRL_PANEL 125
10.10 FOI_MIMO_TX_PATH 134
10.11 FOI_SISO_TX_PATH 136

HOW TO READ THE MANUAL

The manual very verbosely describes files, functions and codes.

For just implementation of test bed, user has to go through **chapter 2 and chapter 4**.

For complete details on codes of the system user has to go through **chapter 5**

SECTION –II

For just implementation of test bed, user has to go through **chapter 7 and chapter 9**.

For complete details of codes of the system user has to go through **chapter 10**

1. Introduction

The wireless networks completely changed the landscape of communication, utilized the frequency spectrum to achieve its needs. Today spectrum chunks are the biggest piece of real estate handled by corporate sectors to provide suitable services to customer's .In this competition spectrum allocation has been very static, unable to adapt to the current pattern of consumer demands. This lead to creation of flexible protocols whose implementation requires entire set of new smart nodes to be deployed in the network.

One of the prototype devices to hit the market is Universal Software Radio Peripheral (USRP) which is our basic node for Cognitive Radio Test Bed. To come up with a suitable set up, where all protocols of Physical layer to Networking layer can be seamlessly switched and instantly deployed with a relative ease requires a more coherent approach from a software perspective. Influx of different types of software's to cater various needs of the Test Bed such as SQL for Database aspects of the project and Python to create a flow graph around all functional blocks of the project requires curating all the steps so as to achieve a harmonious utilization of the various aspects of these software packages.

This manual tries to be a basic handbook to start a journey into the world of Gnu Radio coupled with USRP to suitably deploy a Test Bed demonstrating various basic principles of Cognitive Radio. The technical jargon has been kept a bare minimum so as to focus on the "How's" of the implementation rather than "Why's" of the project. A functional flow graph has been chosen to devour all the contents of the project and show all the interdependencies between them. Also various basic commands of software packages are listed which are frequently utilized to tune parameters of the functions and probe into various "on the run "aspects of the device.

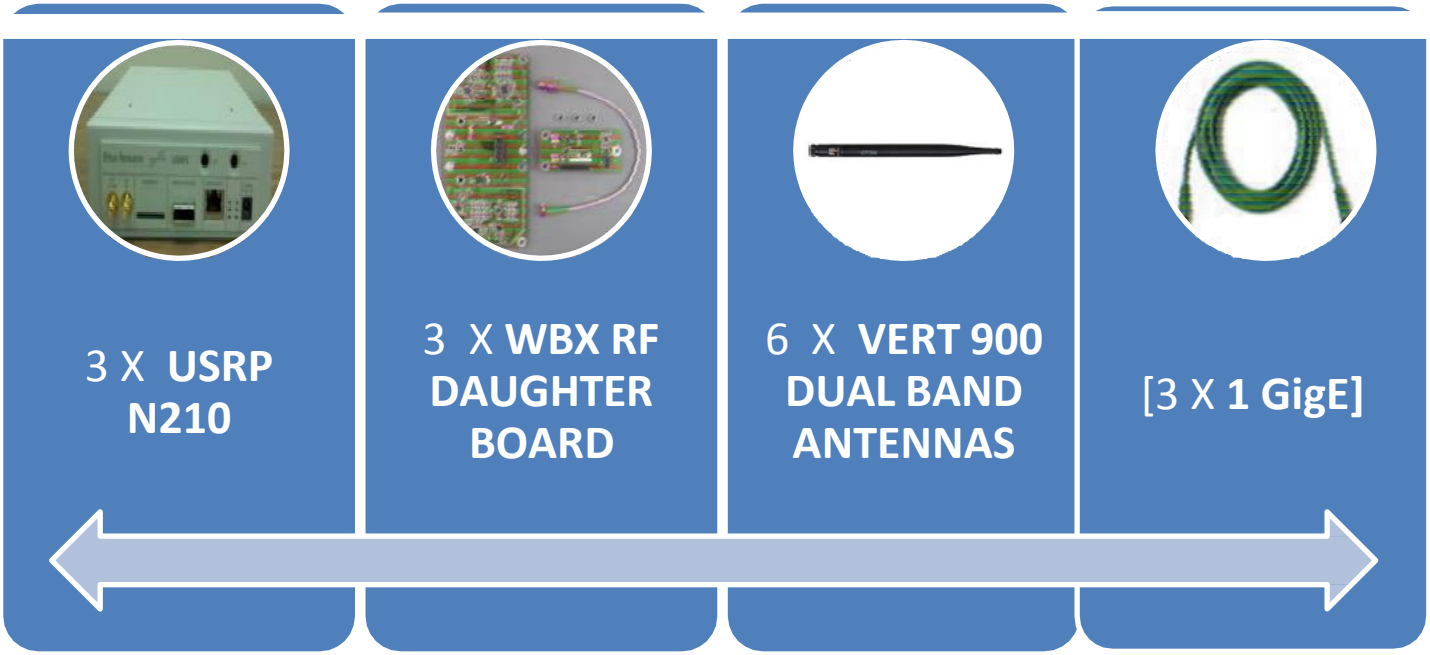
Lastly this manual is developed with assumptions about the users who has a programming background and has an avid knowledge of the technical aspects of the wireless communication related to cognitive radio. The manual is not comprehensive enough to encompass every aspect but few things to be added in near future are fine tune settings for non LOS based setup and more abstract way of handling various pop up errors during "On the run" of the test bed.

2 .Prerequisites

The process starts from procuring the essential hardware and software packages for the project. The number of hardware terminals are not strict but apt according to our model ,choice of software depends on the level of abstraction from which we want to implement the cognitive radio test bed.

2.1 .a Hardware Prerequisites

Below are elements which are can be collectively referred as "USRP KIT" and part of the idea of keeping the hardware aspect highly modularized is to keep up with diverse requirements in terms of computation power, frequency to operate on etc.



2.1. b Hardware Configuration

Although there are videos around the web how to basically combine these elements to form a unit ,it is very easy to configure the hardware .

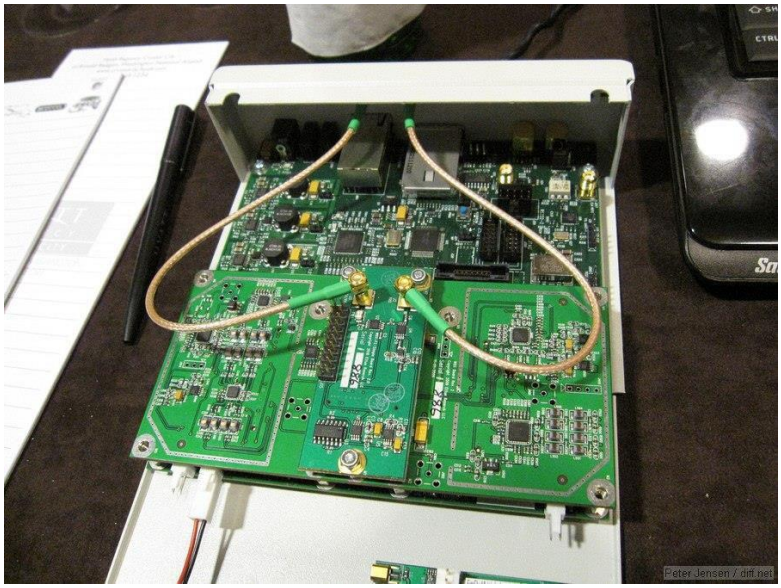
The two important things in this aspect are -:

(a) Install WBX inside USRP N210
(b) Plug the VERT 900 antennas

In a span of few steps through images showing the end state of that step we will configure the hardware devices.

Step 1- Unpack the USRP N210 Device

Step 2 – Install the WBX daughter board inside the USRP N210 kit



Step 3 - Plug the VERT 900 antennas on the front ports of U"RP N210 reading "RF1" and "RF2".

Step 4 – Pack the U"RP N210 and plug the 1 GigE cable into port reading "Ethernet".

Well the above few steps shouldn't take much of time , 15 ~ 20 minutes is enough to configure the entire set up of USRP N210. Although this should be visual enough but cue can be taken from the video from ettus **[LINK]** which shows how to unpack and install WBX board. At the end the type of antennas, RF Daughter board and devices were chosen to work for our design and requirements and which can be swiftly switched for different set of requirements.

2.2 a Software Prerequisites

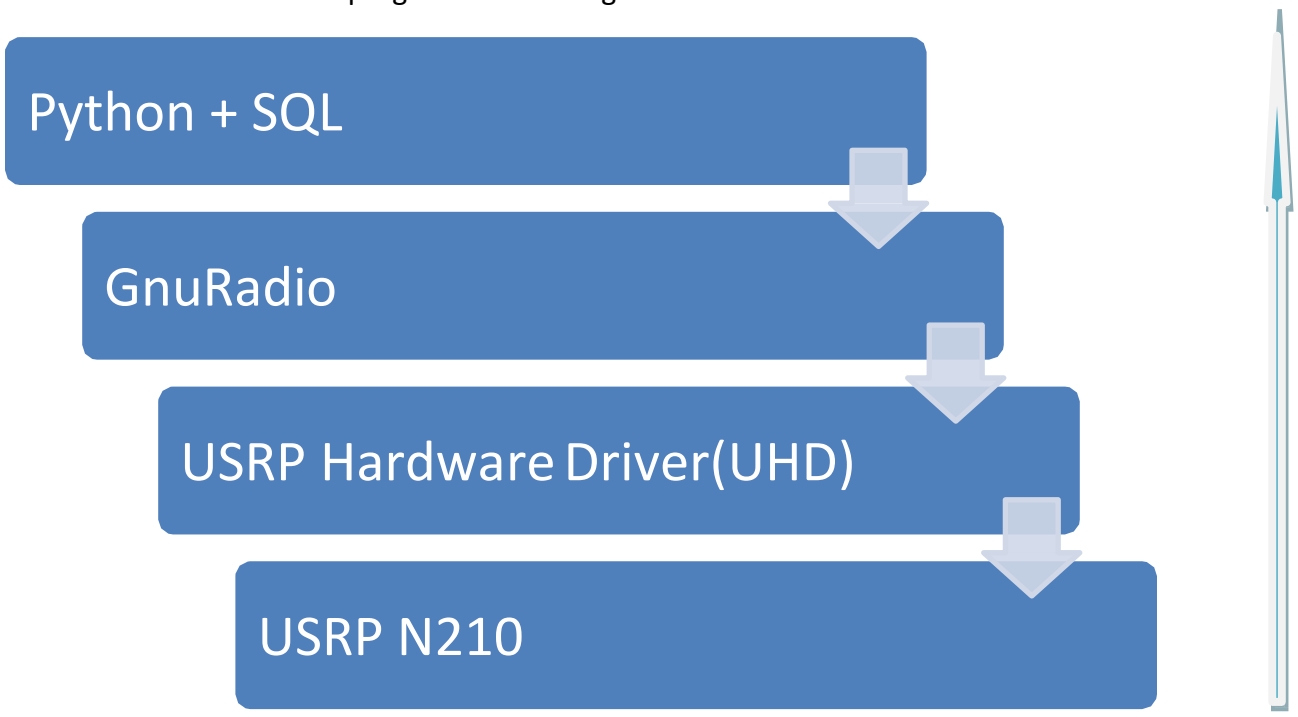
The abstraction of hardware through software is the choice depending upon the level of interface required by the user. Basically there are two approaches

(c) **Modularized Visual Blocks** – Simulink, LabView have suitable high level of abstraction based interface coupled with UHD to configure all aspects of the project. They have a lot of predefined blocks which can be plugged and mixed to do the job.

(d) **Terminal Based** - These are also interfaced with UHD driver but with a lower level of abstraction, simultaneously increasing the flexibility of configuring all parameters and ability to be out of reach from general non programmers .Although these also have option for implementing in a GUI format but Script based implementation has truly the characteristics of being flexible. E.g. – GNU Radio+ UHD

Here we are opting for a more terminal based implementation thus the requirements are different from the other approach.

The prerequisites are shown with their level of abstraction from hardware and software packages by upward arrow and with downward progression showing the flow of instructions from terminal to hardware.



2.2. b Software Configuration

Few things which are recommended but not absolutely essential are

a) Go for a fresh installation of latest Linux version.

b) If your connection to the web is through a proxy server it needs to properly configured so that your required data can be fetched from central repositories of UBUNTU.

1) This method uses the apt.conf file which is found in your **/etc/apt/** directory .

```
$ sudo gedit /etc/apt/apt.conf
```

2) Add these lines to your */etc/apt/apt.conf* file (substitute your details for your proxy address and proxy port). Save the apt.conf file.

```
$ Acquire::http::Proxy "http://user:pass@proxy_host:port";
$Acquire::https::Proxy "https://user:pass@proxy_host:port";
$Acquire::ftp::Proxy "ftp://user:pass@proxy_host:port";
```

c) This command should be periodically run to update ubuntu and all subsidiary packages to ensure smooth functioning of the system.

```
$ sudo apt-get update
```

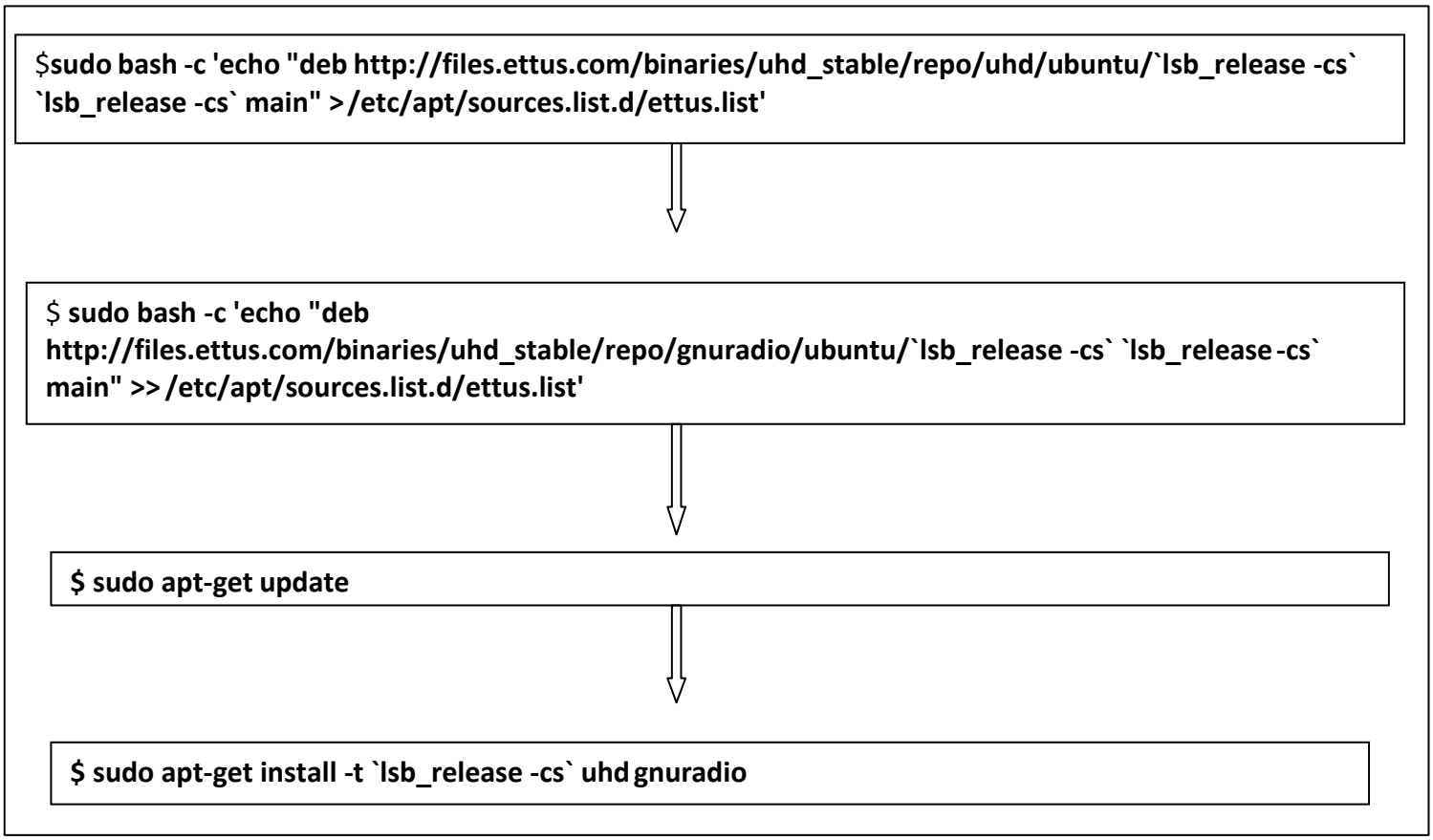
2.2.b. 1 UHD+GNU Radio installation

These instructions are currently supported for fewer platforms of ubuntu so please take into consideration while installing a copy in the system. Currently supported platforms are

(a)UBUNTU 12.04,12.10,13.10

(B) FEDORA 19-20

The steps are to be copied and pasted on terminal to install a software development tool kit which will generate a interface with the hardware and also receive corresponding updates in futures are as follows -:



2.2 b.2 Python +SQL installation+ Other dependencies

- Python is required to build a flow graph around the functional blocks from libraries of GNU Radio. Generally few versions are supported by Gnu Radio such as Python 2.6(all versions) and Python 2.7(all versions).
- There are various other dependencies to GNU Radio such as WX- graphical user interface ,collection of wavelet blocks, QT – graphical user interface, documentation of predefined libraries etc.

For UBUNTU

```
$ sudo apt-get install python python-wxgtk2.8 pyqt4-dev-tools python-qwt5-qt4 python-numpy
  libboost-all-dev libusb-1.0.0-dev
```

- SQL is a very essential software package as it provides solutions regarding storage, processing and faster extraction of useful data which is of paramount importance during the "Run –Time " of various script files based on UHD+GNU Radio on USRP.

```
$ sudo apt-get sqlite3
```

- There are GUI editors to create ,edit and visualize database files compatible with sqlite .It is based on QT-based graphical interface and actively used for relevant data without using complicated SQL commands. It can be set up also easily through ubuntu software centre by downloading software from this [Link] link.

2.2.b. 3 USRP Networking Setup

The USRP generally supports Gigabit Ethernet and its more pragmatic to support higher data rates as all signal processing is to be locally done at host computer. There are few settings which are required to establish communication at IP/UDP Layer over Gigabit.

Before configuring few things to be kept in mind are –

- The default IP address of USRP is **192.168.10.2**
- It is required to configure a static IP address over host Ethernet's interface
- A static IP address for host is typically **192.168.10.1** and subnet mask is **255.255.255.0**

Set up the host interface

```
$ sudo ifconfig <interface> 192.168.10.1
```

Change the USRP address

The need for change can be for multitude reasons such as –

- To satisfy any specific network configuration.
- To connect and establish various data paths for multiple U"RP's on a single host computer.

Suppose the interface is **etho** and IP address is **192.168.10.2** which needs to changed over **to 192.168.10.3**
The following method uses Ethernet packets to bypass IP/UDP layer to communicate information to the USRP.

```
cd <install-path>/lib/uhd/utils
sudo ./usrp2_recovery.py --ifc=eth0 --new-ip=192.168.10.3
```

3. Unpacking of test bed frame work

The huge number of classes and functions are amalgamated to behave as single entity from where filling up a few option parsers such as operating range of frequency, type of modulation , bit rate ,power of transmission for each packet etc. creates a dynamic TCP/IP link demonstrating basic principles of cognitive radio test bed.

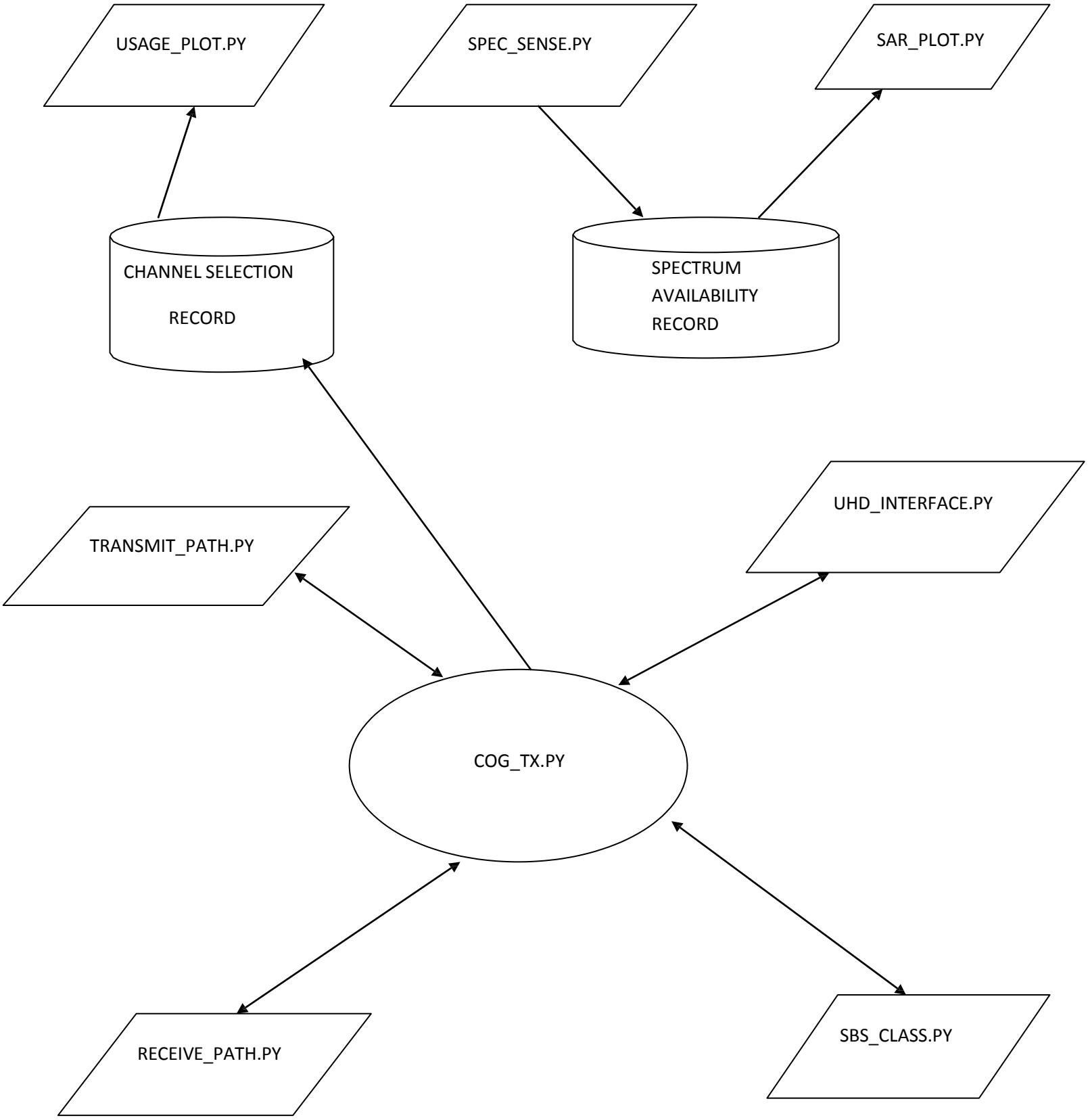
To explain the complex interdependencies the flow is depicted

1) **Through various Files and Scripts**
   The general flow is shown through dependencies across files and scripts.
2) **Through Functions and Classes**
   The sequential flow through various classes and function showing how each contribute towards the creation, transmission and reception of data.

3.1 Through various files and Scripts

To start with CR_DEMO_TX folder there are four files and four scripts which work harmoniously to make our life easier.

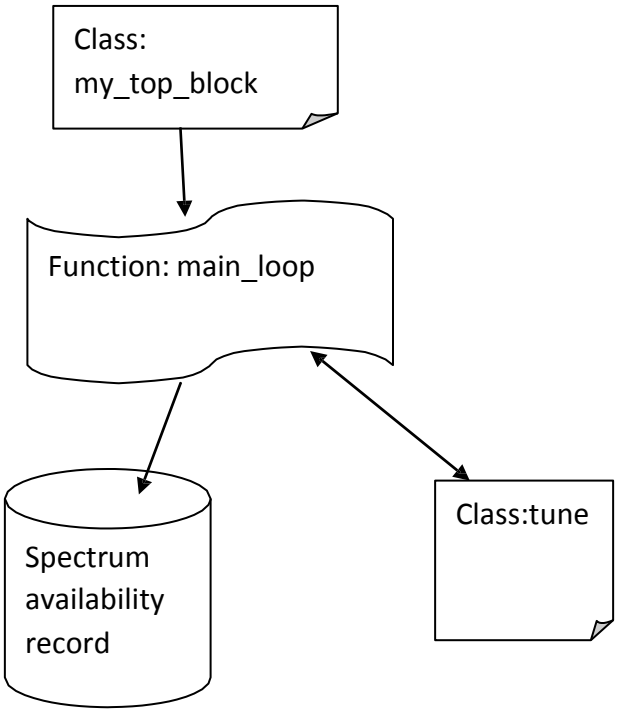
3) **A brief overview of files and scripts through text and flow graphs**
   a) **COG_TX.PY** - It is the main script of the project which has tentacles across all the files and extracts links which help it to interface useful information from/to the hardware component.
   b) **UHD_INTERFACE.PY** – This file contains a huge number of functions directly influencing the UHD driver to tune various parameters of the project such as tuning frequency ,gain ,sample rates as per the requirements of the project.
   c) **TRANSMIT_PATH.PY** – This file contains all the buttons to specifically tune the transmitter side of the project such as send packet function, set transmission amplitude etc.
   d) **RECEIVE _PATH.PY** - This file contains all the specific functions to tune the receiver aspect of hardware component such as demodulators, channel bandwidth factor, bit-rate etc.
   e) **SPEC_SENSE.PY** – This is a very important script which is utilized to jump start the sensing protocol for cognitive radio. It creates a spectrum availability record.
   f) **SBS_CLASS.PY** – This is sort of a pruned ""PEC_"EN"E.PY" file which is utilized to sense a specific band of spectrum (SBS) for a few times and declare its availability .
   g) **SAR_PLOT.PY**- The abbreviation of SAR – Spectrum availability record speaks more than its name as it helps to plot real time record of whether a channel is ready or busy on a certain power threshold.
   h) **USAGE_PLOT.PY** – This script shows the dynamic transmission of the packets through various frequencies with their corresponding power levels and at the end, the completion of the goal.


USAGE_PLOT.PY
SPEC_SENSE.PY
SAR_PLOT.PY
CHANNEL SELECTION RECORD
SPECTRUM AVAILABILITY RECORD
TRANSMIT_PATH.PY
UHD_INTERFACE.PY
COG_TX.PY
RECEIVE_PATH.PY
SBS_CLASS.PY

3.2 Through classes and Functions

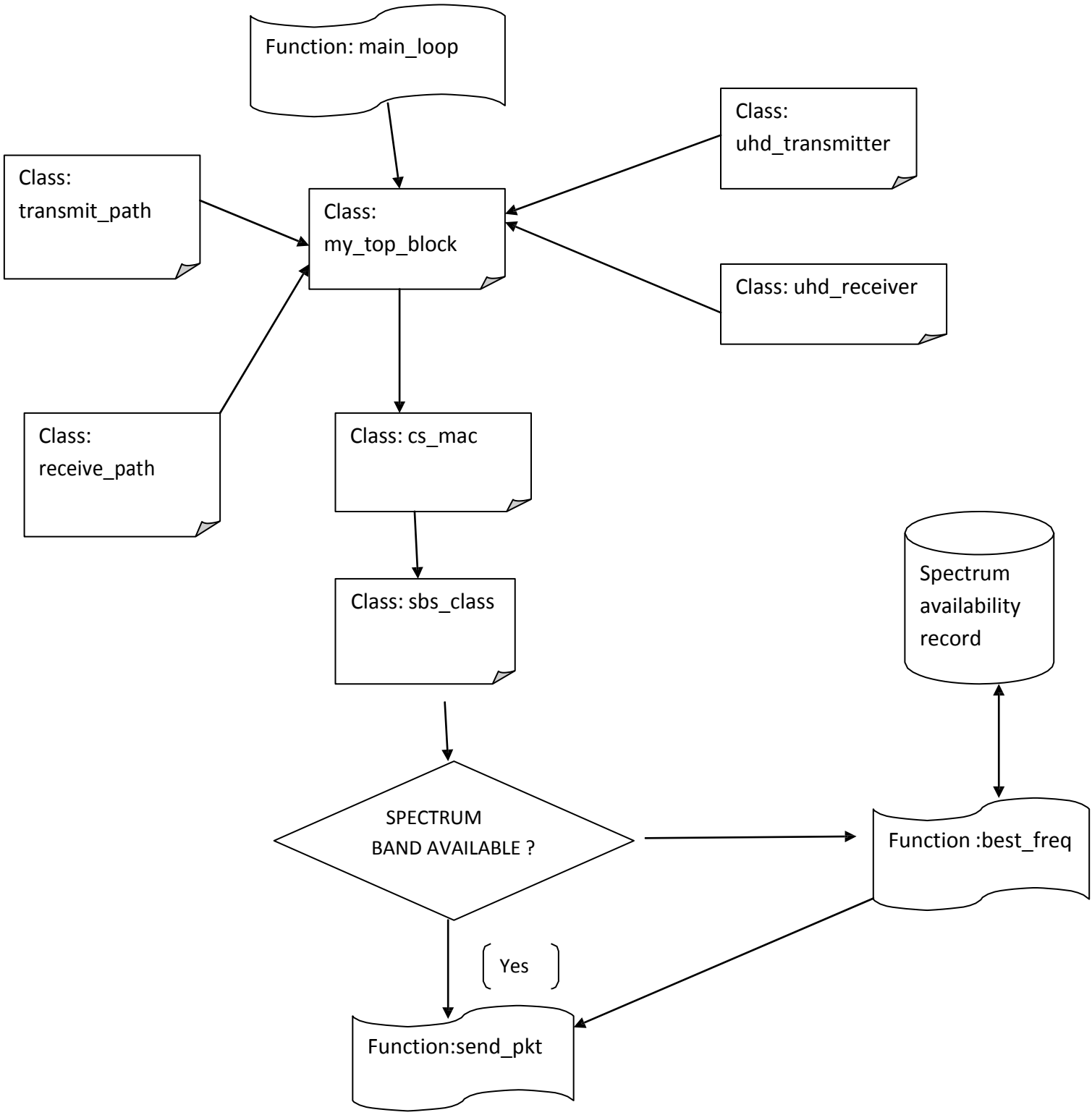
These depict the flow of data and instructions through numerous classes and functions as per the command line arguments given while initiating the script. As it becomes cumbersome to relate the hierarchy and functionality of various classes and functions at one stroke, **these flow graphs contain both the links to the actual code and more verbose idea about their parameters and functionalities.[not added]**

3.2. a Spectrum Sensing

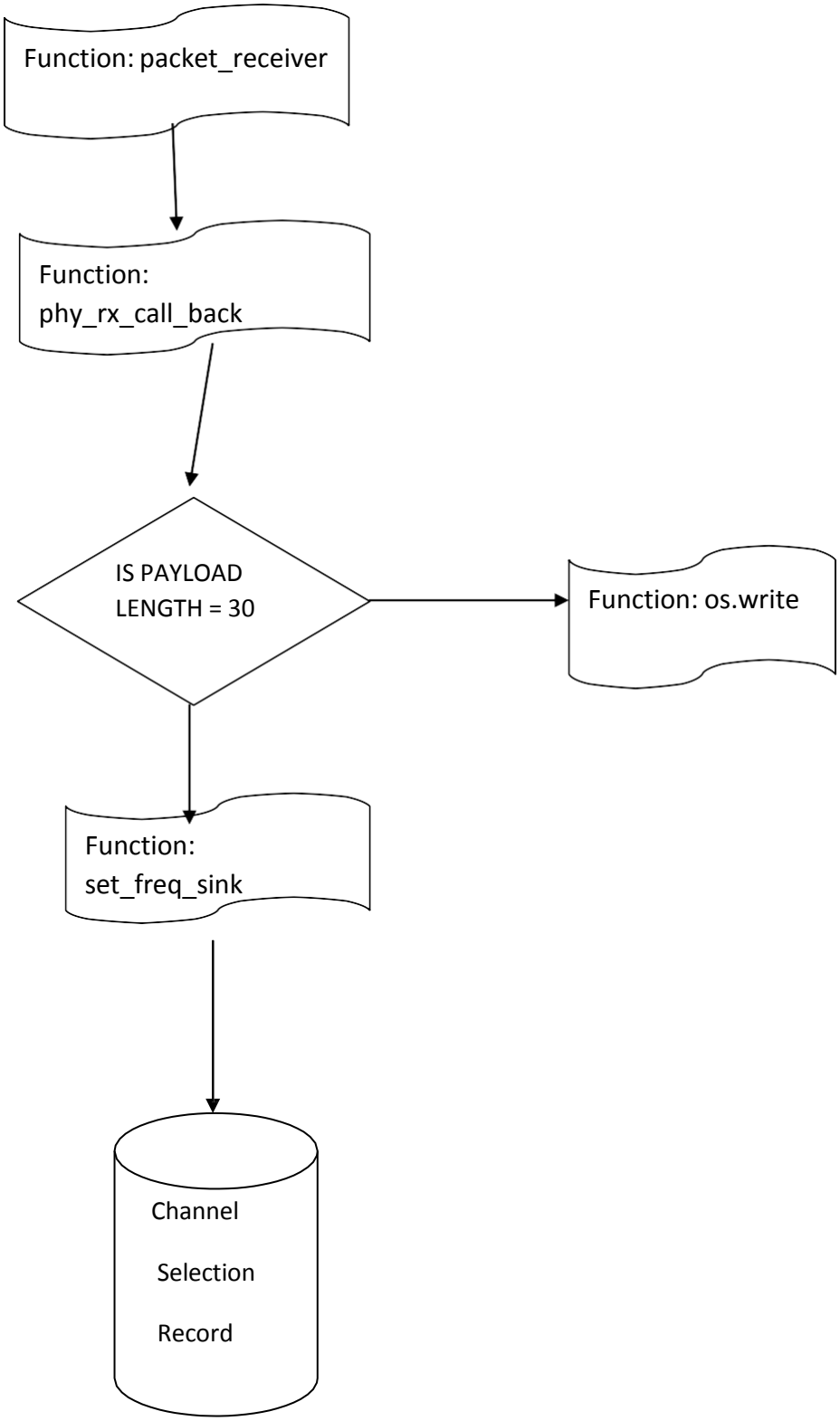


3.2. b TX & RX

3.2.b. 1 Transmission of Data



3 .2.b. 2 Reception of Data



3.3 Classes and functions: A description

The classes under their umbrella have loads of functions which are quite orthogonal to each other and also perform operations on different member variables inherited from a same class. Here the block elements are typically described by their **file names, class/function and reference name to call.**

Spec_sense.py: Class: my_top_block -

Function -This class is on the top hierarchical block in a flow graph containing a hold on all the primary and secondary threads which start, pause and stop the flow during different events of the program. It creates more easier alternatives to give flexible command line arguments through option parsers.

Parameters – gr.top_block ,a class predefined and containing a lot of libraries about running the flowgraph.

Spec_sense.py: Function :main_loop –

Function – This function contains all the iterative statements to continuously scan the spectrum and provide useful data to the tune class. It also creates and populates database as per the schema provided before the execution of the script.

Parameters – my_top_block class object, it fetches all user fed input from this inherited class.

Spec_sense.py: Class: tune –

Function- The class contains specific functions to tune to different centre frequencies across the spectrum and handle the message queue of the USRP .

Parameters - gr.feval_dd, a block inherited to be sub classed for SWIG implementation.

Cog_tx.py: Class: my_top_block -

Function -This class is the top hierarchical block in a flow graph containing a hold on all the primary and secondary threads which start, pause and stop the flow during different events of the program. It creates more easier alternatives to give flexible command line arguments through option parsers.

Parameters – gr.top_block ,a class predefined and containing a lot of libraries about running the flowgraph.

Cog_tx.py :Function:main -

Function – This is a function which performs the task of receiving user inputs and jump starting the flowgraph.

Parameters - None

Transmit_path.py: Class: transmit_path -

Function -This is one of the hierarchical blocks along he graph with the task of creating options for transmission related inputs and perform transmission related functions such as "sending packet" directly linked with the uhd driver.

Parameters – gr.hier_block 2 ,a class predefined and containing a lot of libraries about creating hierarchical blocks from top_block. Also extracts user defined options for transmission.

Receive _path.py: Class: receive_path -

Function -This class is one of the the hierarchical blocks linked to the top block for managing the flow of instructions and data in the receiver side of the flow graph. It has specific callbacks to MAC layer for further processing of received data and probes to implement collision avoiding protocols for packets.

Parameters – gr.hier_block 2, a class predefined and containing a lot of libraries about creating hierarchical blocks from top_block.

Uhd_interface.py: Class: uhd_transmitter -

Function -This class provides direct link to the uhd driver which in turn connects to USRP.The flow graph passing through this class responds to "sink" aspects of U"RP.

Parameters – uhd_interface

Uhd_interface.py: Class: uhd_receiver -

Function - This class provides direct link to the uhd driver which in turn connects to USRP.The flow graph passing through this class responds to the "source" aspects of USRP.

Parameters – uhd_interface

Cog_tx.py: Class: cs_mac -

Function -This class contains functions helping them to read packets from TUN/TAP interface and maintain a to/from relationship with the physical layer.

Parameters – object,It inherits all the functions within the scope of python libraries.

Sbs_class.py: Class: sbs_class -

Function -This class contains suitable functions to perform "Fine sensing" requirements of the model. It repeatedly targets a small band of frequency spectrum rather than wide band sensing.

Parameters – gr.heir_block2 and the incumbent spectrum band from where transmission occurs.

Cog_tx.py: Function: best_freq -

Function -This function under the umbrella of the MAC class connects to the spectral availability record (SAR) and extracts the best channel from those records.

Parameters – Incumbent transmission frequency, It fetches these information from Mac layer so as

to exclude the concurrent frequency record from database.

Transmit_path.py: Function: send_pkt -

Function -This function under the domain of the transmit_path class adds overheads to the payload to make it a packet and send it on the concurrent transmission band.

Parameters – Payload

Receive_path.py: Function: packet_receiver.py -

Function -This function under the umbrella of the receive _path class contains all the information fetched from user for demodulating the signals. It demodulates signals and removes various headers , preamble to provide payloads without any overheads. It performs call back to the MAC layer for suitable action to the received packets.

Parameters – user defined options for receive_path class

Cog_tx.py: Function: phy_rx_callback -

Function -This function under the umbrella of the cs_mac class contains suitable algorithms to respond according to the type of packet received at physical layer. It performs read and write operations from/to the PIPES of the operating system which then can be accessible to any os applications.

Parameters – Received packet from physical layer containing user defined data.

Cog_tx.py: Function: os.write -

Function -This function writes the payload to the PIPES of the operating system with a tag of it being received from USRP. The applications of os then fetch this payload for further processing.

Parameters - TUN/TAP handler, payload

Cog_tx.py: Function: set_freq_sink -

Function -This function under the domain of the top_ block class is a high level interface from the uhd_interface class for real time switching of centre frequency at the device level.

Parameters – New best frequency from database.

4. Building a Cognitive Radio Test Bed

The typical sequence of commands are to jolt the scripts into action by filling them with suitable option parsers such as operating frequency ,bit rate etc. as per the requirements of the model. To demonstrate ,it requires executing **four scripts distributed over time** fulfilling various requirements of sensing , transmission and reception of data. Similarly same scripts run over the reception side of test bed which echoes suitable acknowledgments to fulfill reception side of the protocols.

Before running the commands the only thing to be kept in mind is time taken for execution of spectrum sensing script. It takes time in order of few minutes to create & populate the records in the database for the first time .Thus creating a temporary pause before the swift running of the processes.

Transmission

Step 1 – The **IP addresses of different subnet** must be assigned to both U"RP's of the transmission side. The scripts are designed with following IP settings –

Sensing unit – 192.168.10.2
Transmission unit – 192.168.20.2

Reception

Step2 - The script at reception side is designed with following IP setting-

Reception unit -192.168.10.2

Transmission

Step3- In first round only the spectrum sensing script "spec_sense.py" is made to run till itpopulates the database for **the first time with all the info. about channels across the entire chosen spectrum**. As per the reference given above, during "ab-intio" running of this script no other commands to be executed.

Few parameters for user choice **which are to be given in order** -:

(a) Starting frequency

(b) Ending Frequency

```
$ sudo python spec_sense.py 850M 950M
```

Here the starting frequency is 850 MHz and ending frequency is 950 MHz. But any arbitrary frequency between 830 MHz to 960 MHz can be chosen.

Step4- The next step is to execute a script for real time display of available and busy channels of the chosen spectrum.

```
$ sudo python sar_plot.py
```

Reception

Transmission

Reception

Step5 The next move would be to decide upon the set of **arbitrary transmission and reception frequencies of Reception side.**

The second thing would be **assigning complementary frequencies to transmission side of the set up**.

For E.g

If at reception side ,

Tx –freq is 870M, Rx-freq is 935M

then at transmission side ,

Tx –freq is 935M, Rx-freq is 870M

```
$ sudo python cog_rx.py --tx-freq 870M
-- rx-freq 935M --bitrate 1250k
```

Step6 - The next step would be configuring a virtual Ethernet interface for communicating with other devices.

```
$ sudo ifconfig gr0 192.168.200.1
```

Transmission

Step7 The script for setting up transmission and reception frequencies at transmission side is

```
$ sudo python cog_tx.py  --tx-freq
935M -- rx-freq 870M  --bitrate
1250k
```

Step8 – The virtual Ethernet interface in same subnet Is established at transmission side to communicate with other devices.

```
$sudo    ifconfig  gr0 192.168.200.2
```

Step9 That's it. The set up is ready for communicating to other devices .

To test whether a TCP/IP link has been established between transmission side and reception ,ping commands can be executed against their addresses.

```
$ ping 192.168.200.1
```

Reception

Step10 The similar test could be executed from reception side to transmission side.

```
$ ping 192.168.200.2
```

If ping results are fine,

That's it, we are done building the system.

5. Codes

The codes are given here with a aim of providing reference to all previously mentioned classes and functions to facilitate quicker context based access for easier understanding.. These are written in python and helps us to create flow graphs across all functional blocks to achieve procedural requirements of the protocols.

Spec_sense.py

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

from gnuradio import gr, digital
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

# from current dir
from receive_path  import receive_path
from transmit_path import transmit_path
from uhd_interface import uhd_transmitter
from uhd_interface import uhd_receiver
from uhd_interface import uhd_interface
from sbs_class import *
from sbs_ import *
from main_bar import *
from usrp_uhd import *



import os, sys
import random, time, struct
import  socket
import threading
import subprocess
import sqlite3
import  random
import inspect
import traceback

IFF_TUN      = 0x0001
IFF_TAP      = 0x0002
IFF_NO_PI    = 0x1000
IFF_ONE_QUEUE = 0x2000
IFF_VNET_HDR = 0x4000
IFF_MULTI_QUEUE =0x0100

def open_tun_interface(tun_device_filename):
    from fcntl import ioctl

    mode = IFF_TAP | IFF_NO_PI
    TUNSETIFF = 0x400454ca
```

```
    tun = os.open(tun_device_filename, os.O_RDWR)
    ifs = ioctl(tun, TUNSETIFF, struct.pack("16sH", "gr%d", mode))


    ifname = ifs[:16].strip("\x00")
    return (tun, ifname)

#                       the flow graph


class my_top_block(gr.top_block):

    def   init  (self, mod_class, demod_class,
                 rx_callback, options):

        gr.top_block.  init  (self)


        args = mod_class.extract_kwargs_from_options(options)
        symbol_rate = options.bitrate / mod_class(**args).bits_per_symbol()

        self.source = uhd_receiver(options.args, symbol_rate,
                                   options.samples_per_symbol,
                                   options.rx_freq, options.rx_gain,
                                   options.spec, options.antenna,
                                   options.verbose)

        self.sink = uhd_transmitter(options.args, symbol_rate,
                                    options.samples_per_symbol,
                                    options.tx_freq, options.tx_gain,
                                    options.spec, options.antenna,
                                    options.verbose)

        options.samples_per_symbol = self.source._sps



        self.txpath = transmit_path(mod_class, options)
        self.rxpath = receive_path(demod_class, rx_callback, options)
        self.sense_rxpath = sbs_class(self.source)
        self.connect(self.txpath,  self.sink)
        self.connect(self.source,  self.rxpath)
        self.connect(self.source, self.sense_rxpath)

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

    def carrier_sensed(self):
        """
        Return True if the receive path thinks there's carrier
        """
        return self.rxpath.carrier_sensed()

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

```
        self.sink.set_freq(target_freq)
        self.source.set_freq(target_freq)


    def set_freq_source(self, target_freq):

        return self.source.set_freq(target_freq)


    def set_freq_sink(self, target_freq):


        return self.sink.set_freq(target_freq)


    def sense_busy(self,target_freq):

        return self.sense_rxpath.sense_busy(target_freq)



#                           Carrier Sense MAC


class cs_mac(object):


    def init (self,tun_fd, verbose=False):
        self.tun_fd = tun_fd
        self.verbose = verbose
        self.tb = None              # top block (access to PHY)


    def set_top_block(self, tb):
        self.tb = tb

    def phy_rx_callback(self, ok, payload):

        global final_freq,e, change_count,ACK_string,pd,k,oldfreq

        if self.verbose:
            print "Rx: ok = %r  len(payload) = %4d" % (ok, len(payload))

        if ok:
            os.write(self.tun_fd, payload)
            print " writing to os"
            if (len(payload)==30 and pd==payload):
                #e1.set()
                final_freq = int(payload,2)
                '''ACK = 12345678
                ACK_string = bin(ACK)[2:]
                payload = '%s' %  ACK_string
                print "Sent ACK = ", len(payload)
                self.tb.send_pkt(payload)
                time.sleep(0.0012)'''
                ofreq = self.tb.sink._freq
                self.tb.sink._freq = self.tb.set_freq_sink(final_freq)
                if ofreq == self.tb.sink._freq:
```

```
                    self.rt_ch(self.tb.sink._freq,oldfreq)
                else:
                    self.rt_ch(self.tb.sink._freq,ofreq)
                change_count += 1
                print "No. of Changes = ", change_count
                e.set()
                print "e is set"
            else:
                pass

    def best_freq(self,freq):
        db_filename ='spec.db'
        with sqlite3.connect(db_filename) as conn:
            cursor= conn.cursor()
            extractbestchannel="""select ctfreq from spec where status like 'Av%' and
ctfreq not in (?) order by pwdbm asc"""
            cursor.execute(extractbestchannel,(freq,))

            for row in cursor.fetchmany(1):
                pass

        return row[0]



    def rt_ch(self,freq,oldfreq=0):
        db_filename ='chsel.db'
        isolationlevels= 'IMMEDIATE'
        with sqlite3.connect(db_filename,isolation_level = isolationlevels) as conn1:
            insertchosen ="""update chsel set sel='Yes' where centfreq=?"""
            conn1.execute(insertchosen,(freq,))
            if oldfreq==0:
                pass

            else:
                print "break down solved"
                insertchosen ="""update chsel set sel='No' where centfreq=?"""
                conn1.execute(insertchosen,(oldfreq,))
        return







    def main_loop(self):

        """
        Main loop for MAC.
        Only returns if we get an error reading from TUN.

        FIXME: may want to check for EINTR and EAGAIN and reissue read
        """
        min_delay = 0.001

        while 1 :
```

```
global e,t,bsyn,bsynack,oldfreq,pd,counter

if counter ==0:
    self.rt_ch(self.tb.sink._freq)
    counter +=1

#if sense_busy():
if (self.tb.sense_busy(self.tb.sink._freq)):
    oldfreq=self.tb.sink._freq
    nxtfreq = self.best_freq(self.tb.sink._freq)
    string = bin(nxtfreq)[2:]
    payload = '%s' %  string
    pd=payload
    final_freq = int(payload,2)
    self.tb.send_pkt(payload)
    print "Sent SYN = ", len(payload)
    e.wait(0.8)
    while not e.isSet():
        self.tb.sink._freq = self.tb.set_freq_sink(final_freq)
        self.tb.send_pkt(payload)
        print "Tx:TX len(payload) = %4d newfreq" % (len(payload),)
        bsynack +=1
        time.sleep(0.8)
        if not e.isSet():
            if (self.tb.sink._freq !=oldfreq):
                self.tb.sink._freq = self.tb.set_freq_sink(oldfreq)
            self.tb.send_pkt(payload)
            print "Tx:TX len(payload) = %4d oldfreq" % (len(payload),)
            bsyn +=1
            time.sleep(1.5)



    print "bsyn= %d and bsynack=%d"%(bsyn,bsynack)
    print "after set"
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
    if self.verbose:
        print "Tx:TX len(payload) = %4d" % (len(payload),)
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
    if self.verbose:
        print "Tx:TX len(payload) = %4d" % (len(payload),)
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
    if self.verbose:
        print "Tx:TX len(payload) = %4d" % (len(payload),)
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
    if self.verbose:
        print "Tx:TX len(payload) = %4d" % (len(payload),)

    e.clear()
else:

    print "In os section"
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
    if self.verbose:
```

```
                    print "Tx:TX len(payload) = %4d" % (len(payload),)
                payload = os.read(self.tun_fd, 10*1024)
                self.tb.send_pkt(payload)
                print "send the payload"
                if self.verbose:
                    print "Tx: len(payload) = %4d" % (len(payload),)
                payload = os.read(self.tun_fd, 10*1024)
                self.tb.send_pkt(payload)
                if self.verbose:
                    print "Tx:TX len(payload) = %4d" % (len(payload),)
                payload = os.read(self.tun_fd, 10*1024)
                self.tb.send_pkt(payload)
                if self.verbose:
                    print "Tx:TX len(payload) = %4d" % (len(payload),)

                if not payload:
                    self.tb.send_pkt(eof=True)
                    break



def main():
    global e,y, change_count,bsyn,e1,bsynack,oldfreq,pd,counter,k
    change_count = 0
    counter =0
    bsyn=0
    bsynack=0
    payload=0
    k=''
    e = threading.Event()
    e1= threading.Event()
    mods = digital.modulation_utils.type_1_mods()
    print "Select modulation from: %s [default=%%default]" % (', '.join(mods.keys()),)

    demods = digital.modulation_utils.type_1_demods()

    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("-v","--verbose", action="store_true", default=False)
    expert_grp.add_option("-c", "--carrier-threshold", type="eng_float", default=30,
                          help="set carrier detect threshold (dB) [default=%default]")
    expert_grp.add_option("","--tun-device-filename", default="/dev/net/tun",
                          help="path to tun device file [default=%default]")

    transmit_path.add_options(parser, expert_grp)
    receive_path.add_options(parser, expert_grp)
    uhd_receiver.add_options(parser)
    uhd_transmitter.add_options(parser)

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

```
    for demod in demods.values():
        demod.add_options(expert_grp)

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


    (tun_fd, tun_ifname) = open_tun_interface(options.tun_device_filename)

    r = gr.enable_realtime_scheduling()
    if r == gr.RT_OK:
        realtime = True
        print "real time enabled"
    else:
        realtime = False
        print "Note: failed to enable realtime scheduling"

    # instantiate the MAC
    mac = cs_mac(tun_fd, verbose=True)

    # build the graph (PHY)
    global tb
    tb = my_top_block(mods[options.modulation],
                      demods[options.modulation],
                      mac.phy_rx_callback,
                      options)

    mac.set_top_block(tb)     # give the MAC a handle for the PHY

    if tb.txpath.bitrate() != tb.rxpath.bitrate():
        print "WARNING: Transmit bitrate = %sb/sec, Receive bitrate = %sb/sec" % (
            eng_notation.num_to_str(tb.txpath.bitrate()),
            eng_notation.num_to_str(tb.rxpath.bitrate()))

    print "modulation:     %s"   % (options.modulation,)
    print "freq:           %s"      % (eng_notation.num_to_str(options.tx_freq))
    print "bitrate:        %sb/sec" % (eng_notation.num_to_str(tb.txpath.bitrate()),)
    print "samples/symbol: %3d" % (tb.txpath.samples_per_symbol(),)

    tb.rxpath.set_carrier_threshold(options.carrier_threshold)
    print "Carrier sense threshold:", options.carrier_threshold, "dB"

    print
    print "Allocated virtual ethernet interface: %s" % (tun_ifname,)
    print "You must now use ifconfig to set its IP address. E.g.,"
    print
    print "  $ sudo ifconfig %s 192.168.200.1" % (tun_ifname,)


    tb.start()

    mac.main_loop()

    tb.stop()
    tb.wait()

if __name__ == '__main__':
```

```
    try:

        main()
    except KeyboardInterrupt:
        pass
```

SAR_PLOT.PY

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import sqlite3
import os
import time


class Main_Data(object):

    def   init  (self):

        self.plt = self.setup_backend()
        self.fig = self.plt.figure(figsize=(25,15))
        win = self.fig.canvas.manager.window
        win.after(1, self.animate())
        self.plt.ion()
        self.plt.show()
        while True:
            win.after(1, self.animate())


    def setup_backend(self,backend='TkAgg'):
        import sys
        del sys.modules['matplotlib.backends']
        del sys.modules['matplotlib.pyplot']
        import matplotlib as mpl
        mpl.use(backend)            import matplotlib.pyplot as plt
        return plt


    def animate(self):

        global counter
        if counter == 0:
            isolationlevels= 'IMMEDIATE'
            db_filename ='spec.db'

            with sqlite3.connect(db_filename) as
                conn: conn.row_factory = sqlite3.Row
                cursor= conn.cursor()
                cursor.execute("select * from spec")
                datas= cursor.fetchall()
                self.ctfreqb=[]
                self.pwdbmb=[]
```

```
                self.ctfreqa=[]
                self.pwdbma=[]
                for row in datas:

                    if row['status']=='Busy':
                        self.ctfreqb.append(row['ctfreq']/1e6)
                        self.pwdbmb.append(115+row['pwdbm'])

                    else:
                        self.ctfreqa.append(row['ctfreq']/1e6)
                        self.pwdbma.append(115+row['pwdbm'])
            self.bar_width =0.2
            self.opacity=0.4
            self.error_config = {'ecolor': '1'}
            self.plt.xlim(870,920)
            self.plt.ylim(-115,-60)
            self.rects1 = self.plt.bar(self.ctfreqb, self.pwdbmb, self.bar_width,bottom=-
115,
                     alpha=self.opacity,
                     color='r',

                     label='Busy channel')
            self.rects2 = self.plt.bar(self.ctfreqa, self.pwdbma, self.bar_width,bottom
=-115,
                     alpha=self.opacity,
                     color='g',

                     label='Available channel')

            #self.plt.gca().invert_yaxis()
            self.plt.legend()
            counter +=1
            self.fig.canvas.draw()

        else:

            db_filename ='spec.db'

            with sqlite3.connect(db_filename) as conn:
                conn.row_factory  = sqlite3.Row
                cursor= conn.cursor()
                cursor.execute("select * from spec")
                datas= cursor.fetchall()
                del self.pwdbmb[:]
                del self.pwdbma[:]
                del self.ctfreqb[:]
                del self.ctfreqa[:]
                self.pwdbmb=[]
                self.pwdbma=[]
                self.ctfreqb=[]
                self.ctfreqa=[]

                for row in datas:

                    if row['status']=='Busy':
                        self.ctfreqb.append(row['ctfreq']/1e6)
                        self.pwdbmb.append(115+row['pwdbm'])

                    else:
```

```
                        self.ctfreqa.append(row['ctfreq']/1e6)
                        self.pwdbma.append(115+row['pwdbm'])

            '''print "coming here"
            [bar.set_height(self.pwdbmb[i]) for i,bar in enumerate(self.rects1)]

            [bar.set_height(self.pwdbmb[i]) for i,bar in enumerate(self.rects2)]'''
            self.plt.clf()
            self.plt.xlabel('Frequency in MHz')
            self.plt.ylabel('Power in dBm')
            self.plt.xlim(870,920)
            self.plt.ylim(-115,-60)
            self.rects1 = self.plt.bar(self.ctfreqb, self.pwdbmb, self.bar_width,bottom=-
115,
                     alpha=self.opacity,
                     color='r',

                     label='Busy channel')
            self.rects2 = self.plt.bar(self.ctfreqa, self.pwdbma,self.bar_width,bottom =-
115,
                     alpha=self.opacity,
                     color='g',

                     label='Available channel')

            #self.plt.gca().invert_yaxis()
            self.plt.legend()

            self.fig.canvas.draw()



if   name___== '  main  ':
    try:
        global counter
        counter=0
        k=  Main_Data()


    except KeyboardInterrupt:

        pass
```

Cog_tx.py

#!/usr/bin/env python

from gnuradio import gr, digital

from gnuradio import eng_notation

```
from gnuradio.eng_option import eng_option

from optparse import OptionParser


# from current dir

from receive_path import receive_path

from transmit_path import transmit_path

from uhd_interface import uhd_transmitter

from uhd_interface import uhd_receiver

from uhd_interface import uhd_interface

from sbs_class import *

from sbs_ import  *

from main_bar import *

from usrp_uhd import *



import os, sys

import random, time, struct

import socket

import threading

import subprocess

import sqlite3

import random

import inspect

import traceback
```

```
IFF_TUN                 = 0x0001
IFF_TAP         = 0x0002
IFF_NO_PI       = 0x1000
IFF_ONE_QUEUE = 0x2000
IFF_VNET_HDR = 0x4000
IFF_MULTI_QUEUE =0x0100


def open_tun_interface(tun_device_filename):
   from fcntl import ioctl


   mode = IFF_TAP | IFF_NO_PI
   TUNSETIFF = 0x400454ca



    # Open control device and request interface
   tun = os.open(tun_device_filename, os.O_RDWR)
   ifs = ioctl(tun, TUNSETIFF, struct.pack("16sH", "gr%d", mode))


   # Retreive real interface name from control device
   ifname = ifs[:16].strip("\x00")
   return (tun, ifname)


#               the flow graph



class my_top_block(gr.top_block):
```

```
def __init (self, mod_class, demod_class,
        rx_callback, options):

  gr.top_block.__init    (self)


  args = mod_class.extract_kwargs_from_options(options)
  symbol_rate = options.bitrate / mod_class(**args).bits_per_symbol()

  self.source = uhd_receiver(options.args, symbol_rate,
                options.samples_per_symbol,
                options.rx_freq, options.rx_gain,
                options.spec, options.antenna,
                options.verbose)

  self.sink = uhd_transmitter(options.args, symbol_rate,
                 options.samples_per_symbol,
                 options.tx_freq, options.tx_gain,
                 options.spec, options.antenna,
                 options.verbose)

  options.samples_per_symbol = self.source._sps
```

```
    self.txpath = transmit_path(mod_class, options)
    self.rxpath = receive_path(demod_class, rx_callback, options)
    self.sense_rxpath = sbs_class(self.source)
    self.connect(self.txpath, self.sink)
    self.connect(self.source, self.rxpath)
    self.connect(self.source,  self.sense_rxpath)

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

def carrier_sensed(self):
    """
    Return True if the receive path thinks there's carrier
    """
    return self.rxpath.carrier_sensed()

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

    self.sink.set_freq(target_freq)
    self.source.set_freq(target_freq)

def set_freq_source(self, target_freq):
```

```
        return self.source.set_freq(target_freq)


    def set_freq_sink(self, target_freq):


        return self.sink.set_freq(target_freq)


    def sense_busy(self,target_freq):


        return  self.sense_rxpath.sense_busy(target_freq)




#                       Carrier Sense MAC


class cs_mac(object):


    def __init (self,tun_fd, verbose=False):
        self.tun_fd = tun_fd
        self.verbose = verbose
        self.tb = None           # top block (access to PHY)
```

```
def set_top_block(self, tb):
    self.tb = tb

def phy_rx_callback(self, ok, payload):

    global final_freq,e, change_count,ACK_string,pd,k,oldfreq

    if self.verbose:
        print "Rx: ok = %r  len(payload) = %4d" % (ok, len(payload))

    if ok:
        os.write(self.tun_fd, payload)
        print " writing to os"
        if (len(payload)==30 and pd==payload):
            #e1.set()
            final_freq = int(payload,2)
            '''ACK = 12345678
            ACK_string = bin(ACK)[2:]
            payload = '%s' %  ACK_string
            print "Sent ACK = ", len(payload)
            self.tb.send_pkt(payload)
            time.sleep(0.0012)'''
            ofreq = self.tb.sink._freq
            self.tb.sink._freq = self.tb.set_freq_sink(final_freq)
```

```
            if ofreq == self.tb.sink._freq:
                self.rt_ch(self.tb.sink._freq,oldfreq)
            else:
                self.rt_ch(self.tb.sink._freq,ofreq)
            change_count += 1
            print "No. of Changes = ", change_count
            e.set()
            print "e is set"
        else:
            pass


def best_freq(self,freq):
    db_filename ='spec.db'
    with sqlite3.connect(db_filename) as conn:
        cursor= conn.cursor()
        extractbestchannel="""select ctfreq from spec where status like 'Av%' and ctfreq not in (?) order by pwdbm asc"""
        cursor.execute(extractbestchannel,(freq,))

        for row in cursor.fetchmany(1):
            pass

    return row[0]



def rt_ch(self,freq,oldfreq=0):
```

```
    db_filename ='chsel.db'
    isolationlevels= 'IMMEDIATE'
    with sqlite3.connect(db_filename,isolation_level = isolationlevels) as conn1:
        insertchosen ="""update chsel set sel='Yes' where centfreq=?"""
        conn1.execute(insertchosen,(freq,))
        if oldfreq==0:
            pass

        else:
            print "break down solved"
            insertchosen ="""update chsel set sel='No' where centfreq=?"""
            conn1.execute(insertchosen,(oldfreq,))
    return

def main_loop(self):

    min_delay = 0.001

    while 1 :

        global e,t,bsyn,bsynack,oldfreq,pd,counter

        if counter ==0:
            self.rt_ch(self.tb.sink._freq)
            counter +=1
```

```
#if sense_busy():
if (self.tb.sense_busy(self.tb.sink._freq)):
    oldfreq=self.tb.sink._freq
    nxtfreq = self.best_freq(self.tb.sink._freq)
    string = bin(nxtfreq)[2:]
    payload = '%s' %  string
    pd=payload
    final_freq = int(payload,2)
    self.tb.send_pkt(payload)
    print "Sent SYN = ", len(payload)
    e.wait(0.8)
    while not e.isSet():
        self.tb.sink._freq = self.tb.set_freq_sink(final_freq)
        self.tb.send_pkt(payload)
        print "Tx:TX len(payload) = %4d newfreq" % (len(payload),)
        bsynack +=1
        time.sleep(0.8)
        if not e.isSet():
            if (self.tb.sink._freq !=oldfreq):
                self.tb.sink._freq = self.tb.set_freq_sink(oldfreq)
            self.tb.send_pkt(payload)
            print "Tx:TX len(payload) = %4d oldfreq" % (len(payload),)
            bsyn +=1
            time.sleep(1.5)
```

```
    print "bsyn= %d and bsynack=%d"%(bsyn,bsynack)
    print "after set"
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
    if self.verbose:
        print "Tx:TX len(payload) = %4d" % (len(payload),)
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
    if self.verbose:
        print "Tx:TX len(payload) = %4d" % (len(payload),)
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
    if self.verbose:
        print "Tx:TX len(payload) = %4d" % (len(payload),)
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
    if self.verbose:
        print "Tx:TX len(payload) = %4d" % (len(payload),)

    e.clear()
else:

    print "In os section"
    payload = os.read(self.tun_fd, 10*1024)
    self.tb.send_pkt(payload)
```

```
            if self.verbose:
                print "Tx:TX len(payload) = %4d" % (len(payload),)
            payload = os.read(self.tun_fd, 10*1024)
            self.tb.send_pkt(payload)
            print "send the payload"
            if self.verbose:
                print "Tx: len(payload) = %4d" % (len(payload),)
            payload = os.read(self.tun_fd, 10*1024)
            self.tb.send_pkt(payload)
            if self.verbose:
                print "Tx:TX len(payload) = %4d" % (len(payload),)
            payload = os.read(self.tun_fd, 10*1024)
            self.tb.send_pkt(payload)
            if self.verbose:
                print "Tx:TX len(payload) = %4d" % (len(payload),)

            if not payload:
                self.tb.send_pkt(eof=True)
                break


def main():
    global e,y, change_count,bsyn,e1,bsynack,oldfreq,pd,counter,k
    change_count = 0
    counter =0
```

```
bsyn=0

bsynack=0

payload=0

k=''

e = threading.Event()

e1= threading.Event()

mods = digital.modulation_utils.type_1_mods()

print "Select modulation from: %s [default=%%default]" % (', '.join(mods.keys()),)


demods = digital.modulation_utils.type_1_demods()


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("-v","--verbose", action="store_true", default=False)

expert_grp.add_option("-c", "--carrier-threshold", type="eng_float", default=30,

                    help="set carrier detect threshold (dB) [default=%default]")

expert_grp.add_option("","--tun-device-filename",  default="/dev/net/tun",

                    help="path to tun device file [default=%default]")
```

```
transmit_path.add_options(parser, expert_grp)

receive_path.add_options(parser, expert_grp)

uhd_receiver.add_options(parser)

uhd_transmitter.add_options(parser)


for mod in mods.values():

    mod.add_options(expert_grp)


for demod in demods.values():

    demod.add_options(expert_grp)


(options, args) = parser.parse_args ()

if len(args) != 0:

    parser.print_help(sys.stderr)

    sys.exit(1)




(tun_fd, tun_ifname) = open_tun_interface(options.tun_device_filename)


# Attempt to enable realtime scheduling

r = gr.enable_realtime_scheduling()

if r == gr.RT_OK:

    realtime = True

    print "real time enabled"

else:

    realtime = False
```

```
    print "Note: failed to enable realtime scheduling"


# instantiate the MAC
mac = cs_mac(tun_fd, verbose=True)


# build the graph (PHY)
global tb
tb = my_top_block(mods[options.modulation],
                  demods[options.modulation],
                  mac.phy_rx_callback,
                  options)


mac.set_top_block(tb)    # give the MAC a handle for the PHY


if tb.txpath.bitrate() != tb.rxpath.bitrate():
    print "WARNING: Transmit bitrate = %sb/sec, Receive bitrate = %sb/sec" % (
        eng_notation.num_to_str(tb.txpath.bitrate()),
        eng_notation.num_to_str(tb.rxpath.bitrate()))


print "modulation:     %s"   % (options.modulation,)
print "freq:           %s"     % (eng_notation.num_to_str(options.tx_freq))
print "bitrate:        %sb/sec" % (eng_notation.num_to_str(tb.txpath.bitrate()),)
print "samples/symbol: %3d" % (tb.txpath.samples_per_symbol(),)


tb.rxpath.set_carrier_threshold(options.carrier_threshold)
print "Carrier sense threshold:", options.carrier_threshold, "dB"
```

```
    print
    print "Allocated virtual ethernet interface: %s" % (tun_ifname,)
    print "You must now use ifconfig to set its IP address. E.g.,"
    print
    print " $ sudo ifconfig %s 192.168.200.1" % (tun_ifname,)
    print
    print "Be sure to use a different address in the same subnet for each machine."
    print



    tb.start


    mac.main_loop()
    tb.stop()
  tb.wait()



if  name__ == '  main__':
    try:


        main()
    except KeyboardInterrupt:
        pass
```

uhd_interface.py

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

from gnuradio import gr, uhd
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

import sys

def add_freq_option(parser):

    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")

class uhd_interface:
    def init (self, istx, args, sym_rate, sps, freq=None,
                  gain=None, spec=None, antenna=None):

        if(istx):
            self.u = uhd.usrp_sink(device_addr=args, stream_args=uhd.stream_args('fc32'))
        else:
            self.u = uhd.usrp_source(device_addr=args,
stream_args=uhd.stream_args('fc32'))

        # Set the subdevice spec
        if(spec):
            self.u.set_subdev_spec(spec, 0)

        # Set the antenna
        if(antenna):
            self.u.set_antenna(antenna, 0)

        self._args = args
        self._ant  = antenna
        self._spec = spec
        self._gain = self.set_gain(gain)
        self._freq = self.set_freq(freq)

        self._rate, self._sps = self.set_sample_rate(sym_rate, sps)

    def set_sample_rate(self, sym_rate, req_sps):
        start_sps = req_sps
        while(True):
            asked_samp_rate = sym_rate * req_sps
            print "asked samp rate is %d",asked_samp_rate
            self.u.set_samp_rate(asked_samp_rate)
            actual_samp_rate = self.u.get_samp_rate()
```

```
            sps = actual_samp_rate/sym_rate
            if(sps < 2):
                req_sps +=1
            else:
                actual_sps = sps
                break

        if(sps != req_sps):
            print "\nSymbol Rate:          %f" % (sym_rate)
            print "Requested sps:       %f" % (start_sps)
            print "Given sample rate:   %f" % (actual_samp_rate)
            print "Actual sps for rate: %f" % (actual_sps)

        if(actual_samp_rate != asked_samp_rate):
            print "\nRequested sample rate: %f" % (asked_samp_rate)
            print "Actual sample rate: %f" % (actual_samp_rate)

        return (actual_samp_rate, actual_sps)

    def get_sample_rate(self):
        return self.u.get_samp_rate()

    def set_gain(self, gain=None):
        if gain is None:
            # if no gain was specified, use the mid-point in dB
            g = self.u.get_gain_range()
            gain = float(g.start()+g.stop())/2
            print "\nNo gain specified."
            print "Setting gain to %f (from [%f, %f])" % \
                (gain, g.start(), g.stop())

        self.u.set_gain(gain, 0)
        return gain

    def set_freq(self, freq=None):
        if(freq is None):
            sys.stderr.write("You must specify -f FREQ or --freq FREQ\n")
            sys.exit(1)

        r = self.u.set_center_freq(freq, 0)
        if r:
            print r
            return freq
        else:
            frange = self.u.get_freq_range()
            sys.stderr.write(("\nRequested frequency (%f) out or range [%f, %f]\n") % \
                                (freq, frange.start(), frange.stop()))
            sys.exit(1)

#-------------------------------------------------------------------#
#   TRANSMITTER
#-------------------------------------------------------------------#

class uhd_transmitter(uhd_interface, gr.hier_block2):
    def init (self, args, sym_rate, sps, freq=None, gain=None,
                 spec=None, antenna=None, verbose=False):
        gr.hier_block2.  init  (self, "uhd_transmitter",
                                gr.io_signature(1,1,gr.sizeof_gr_complex),
                                gr.io_signature(0,0,0))
```

```
        # Set up the UHD interface as a transmitter
        uhd_interface.  init  (self, True, args, sym_rate, sps,
                               freq, gain, spec, antenna)

        self.connect(self, self.u)

        if(verbose):
            self._print_verbage()

    def add_options(parser):
        add_freq_option(parser)
        parser.add_option("-a", "--args", type="string", default="addr=192.168.20.2",
                          help="UHD device address args [default=%default]")
        parser.add_option("", "--spec", type="string", default=None,
                          help="Subdevice of UHD device where appropriate")
        parser.add_option("-A", "--antenna", type="string", default=None,
                          help="select Rx Antenna where appropriate")
        parser.add_option("", "--tx-freq", type="eng_float", default=None,
                          help="set transmit frequency to FREQ [default=%default]",
                          metavar="FREQ")
        parser.add_option("", "--tx-gain", type="eng_float", default=None,
                          help="set transmit gain in dB (default is midpoint)")
        parser.add_option("-v", "--verbose", action="store_true", default=False)

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

    def _print_verbage(self):
        """
        Prints information about the UHD transmitter
        """
        print "\nUHD Transmitter:"
        print "Args:     %s"    % (self._args)
        print "Freq:        %sHz"  % (eng_notation.num_to_str(self._freq))
        print "Gain:        %f dB" % (self._gain)
        print "Sample Rate: %ssps" % (eng_notation.num_to_str(self._rate))
        print "Antenna:     %s"    % (self._ant)
        print "Subdev Sec:  %s"    % (self._spec)


#----------------------------------------------------------------------#
#   RECEIVER
#----------------------------------------------------------------------#


class uhd_receiver(uhd_interface, gr.hier_block2):
    def init (self, args, sym_rate, sps, freq=None, gain=None,
                  spec=None, antenna=None, verbose=False):
        gr.hier_block2.  init  (self, "uhd_receiver",
                                gr.io_signature(0,0,0),
                                gr.io_signature(1,1,gr.sizeof_gr_complex))

        # Set up the UHD interface as a receiver
        uhd_interface.  init  (self, False, args, sym_rate, sps,
                               freq, gain, spec, antenna)

        self.connect(self.u, self)
```

```
        if(verbose):
            self._print_verbage()

    def add_options(parser):
        add_freq_option(parser)
        parser.add_option("-a", "--args", type="string", default="addr=192.168.20.2",
                          help="UHD device address args [default=%default]")
        parser.add_option("", "--spec", type="string", default=None,
                          help="Subdevice of UHD device where appropriate")
        parser.add_option("-A", "--antenna", type="string", default=None,
                          help="select Rx Antenna where appropriate")
        parser.add_option("", "--rx-freq", type="eng_float", default=None,
                          help="set receive frequency to FREQ [default=%default]",
                          metavar="FREQ")
        parser.add_option("", "--rx-gain", type="eng_float", default=None,
                          help="set receive gain in dB (default is midpoint)")
        if not parser.has_option("--verbose"):
            parser.add_option("-v", "--verbose", action="store_true", default=False)

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

    def _print_verbage(self):
        """
        Prints information about the UHD transmitter
        """
        print "\nUHD Receiver:"
        print "UHD Args:    %s"    % (self._args)
        print "Freq:        %sHz"  % (eng_notation.num_to_str(self._freq))
        print "Gain:        %f dB" % (self._gain)
        print "Sample Rate: %ssps" % (eng_notation.num_to_str(self._rate))
        print "Antenna:     %s"    % (self._ant)
        print "Spec:        %s"    % (self._spec)
```

transmit_path.py

```
from gnuradio import gr

from gnuradio import blocks

from gnuradio import eng_notation

from gnuradio import digital


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),
                                        gr.io_signature(1,1,gr.sizeof_gr_complex))

        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._modulator_class = modulator_class     # the modulator_class we are using

        # Get mod_kwargs
        mod_kwargs = self._modulator_class.extract_kwargs_from_options(options)

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

```
    self.packet_transmitter = \
        digital.mod_pkts(self.modulator,
                         access_code=None,
                         msgq_limit=4,
                         pad_for_usrp=True)

    self.amp = blocks.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):
```

```
    return self.packet_transmitter.send_pkt(payload, eof)

def bitrate(self):
    return self._bitrate

def samples_per_symbol(self):
    return self.modulator._samples_per_symbol

def differential(self):
    return self.modulator._differential

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=500e3,
                          help="specify bitrate [default=%default].")
    normal.add_option("", "--tx-amplitude", type="eng_float",
                      default=0.7500, metavar="AMPL",
                      help="set transmitter digital amplitude: 0 <= AMPL < 1 [default=%default]")
    normal.add_option("-v", "--verbose", action="store_true",
                      default=True)

    expert.add_option("-S", "--samples-per-symbol", type="float",
```

```
                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:  %.4f"  % (self.samples_per_symbol())
    print "Differential:    %s"    % (self.differential())
```

receive_path.py

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

from gnuradio import gr, gru
from gnuradio import filter
from gnuradio import analog
```

```
from gnuradio import eng_notation
from gnuradio import digital

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),
                                 gr.io_signature(0, 0, 0))

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

        self._verbose     = options.verbose
        self._bitrate     = options.bitrate  # desired bit rate

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

        self._chbw_factor = options.chbw_factor # channel filter bandwidth factor
```

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

    # Build the demodulator
    self.demodulator = self._demod_class(**demod_kwargs)

    # Make sure the channel BW factor is between 1 and sps/2
    # or the filter won't work.
    if(self._chbw_factor < 1.0 or self._chbw_factor > self.samples_per_symbol()/2):
        sys.stderr.write("Channel bandwidth factor ({0}) must be within the range [1.0, {1}].\n".format(self._chbw_factor,
self.samples_per_symbol()/2))
        sys.exit(1)

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

    # receiver
    self.packet_receiver = \
        digital.demod_pkts(self.demodulator,
                           access_code=None,
```

```
                callback=self._rx_callback,
                threshold=-1)

    # Carrier Sensing Blocks
    alpha = 0.001
    thresh =10# in dB, will have to adjust
    self.probe = analog.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.demodulator._samples_per_symbol
```

```
def differential(self):
    return self.demodulator._differential

def carrier_sensed(self):
    """
    Return True if we think carrier is present.
    """
    print "carrier level=",self.probe.level()

    print "called carrier sensed"
    return self.probe.unmuted()

def carrier_level(self):

    return self.probe.level()

def carrier_threshold(self):
    return self.probe.threshold()

def set_carrier_threshold(self, threshold_in_db):
    self.probe.set_threshold(threshold_in_db)

def add_options(normal, expert):
    if not normal.has_option("--bitrate"):
        normal.add_option("-r", "--bitrate", type="eng_float", default=500e3,
```

```
                help="specify bitrate [default=%default].")
        normal.add_option("-v", "--verbose", action="store_true", default=False)
        expert.add_option("-S", "--samples-per-symbol", type="float", 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)")
        expert.add_option("", "--chbw-factor", type="float", default=1.0,
                          help="Channel bandwidth = chbw_factor x signal bandwidth [defaut=%default]")


    add_options = staticmethod(add_options)



    def _print_verbage(self):
        print "\nReceive Path:"
        print "modulation:      %s"    % (self._demod_class.    name__)
        print "bitrate:         %sb/s" % (eng_notation.num_to_str(self._bitrate))
        print "samples/symbol:  %.4f"    % (self.samples_per_symbol())
        print "Differential:    %s"    % (self.differential())
```

sbs_class.py

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


from gnuradio import gr, eng_notation
from gnuradio import blocks
```

```
from gnuradio import audio

from gnuradio import filter

from gnuradio import fft

from gnuradio import uhd

from gnuradio.eng_option import eng_option

from optparse import OptionParser

import sys

import math

import struct

import threading

from datetime import datetime

import logging

import time

import subprocess

import sqlite3

import os

import inspect


class tune(gr.feval_dd):

    def __init    (self, sbs):


        gr.feval_dd.__init (self)

        self.sbs = sbs


    def eval(self, ignore):
```

```
        try:
            #print "no. of msg in q are %d"%self.sbs.msgq.count()
            new_freq = self.sbs.set_next_freq()

            # wait until msgq is empty before continuing
            while(self.sbs.msgq.full_p()):
                print "msgq full, holding.."
                time.sleep(.1)'''

            return new_freq

        except Exception, e:
            print "tune: Exception: ", e

class parse_msg(object):
    def __init    (self, msg):

        #print "collect ct and data"
        self.center_freq = msg.arg1()
        self.vlen = int(msg.arg2())
        assert(msg.length() == self.vlen * gr.sizeof_float)

        # FIXME consider using NumPy array
        t = msg.to_string()
```

```
        self.raw_data = t

        self.data = struct.unpack('%df' % (self.vlen,), t)




class sbs_class(gr.hier_block2):


    def __init    (self,link):

        gr.hier_block2.__init__(self,  "sense_receive_path",

                                gr.io_signature(1, 1, gr.sizeof_gr_complex),

                                gr.io_signature(0, 0, 0))



        self.it=0

        self.channel_bandwidth = 200000

        self.min_center_freq = 0

        self.max_center_freq = 0

        self.link = link

        self.next_freq= 0

        self.freq_step = 0

        self.usrp_rate =900e3

        self.fft_size = 64

        self.lo_offset =0

        self.squelch_threshold =7
```

```
self.s2v = blocks.stream_to_vector(gr.sizeof_gr_complex, self.fft_size)

mywindow = filter.window.blackmanharris(self.fft_size)
self.ffter = fft.fft_vcc(self.fft_size, True, mywindow, True)
power = 0
for tap in mywindow:
    power += tap*tap

self.c2mag = blocks.complex_to_mag_squared(self.fft_size)
td = 0.2
dd = 0.1

tune_delay = max(0, int(round(td * self.usrp_rate / self.fft_size))) # in fft_frames
dwell_delay = max(1, int(round(dd * self.usrp_rate / self.fft_size))) # in fft_frames

self.msgq = gr.msg_queue(1)
self._tune_callback = tune(self)        # hang on to this to keep it from being GC'd
self.stats = blocks.bin_statistics_f(self.fft_size, self.msgq,
                                     self._tune_callback, tune_delay,
                                     dwell_delay)

self.connect(self, self.s2v, self.ffter, self.c2mag, self.stats)


def set_next_freq(self):
```

```
    if (self.min_center_freq!=0):
        target_freq = self.next_freq
        self.next_freq = self.next_freq + self.freq_step
        if self.next_freq >= self.max_center_freq:
            self.next_freq = self.min_center_freq

        if not self.set_freq(target_freq):
            print "Failed to set frequency to", target_freq
            sys.exit(1)


        return target_freq

    else:
        if self.it==0:
            self.set_freq(self.link._freq)
            self.it =1
            return self.link._freq
        else:
            return self.link._freq


def set_freq(self, target_freq):

    r = self.link.u.set_center_freq(uhd.tune_request(target_freq, rf_freq=(target_freq+
```

```
self.lo_offset),rf_freq_policy=uhd.tune_request.POLICY_MANUAL))
        if r:

          return True

        return False

    def nearest_freq(self, freq, channel_bandwidth):
        nrfreq = round(freq / channel_bandwidth, 0) * channel_bandwidth
        return nrfreq


    def sense_busy(self,ctfreq):


        try:

            print("")
            print"entering to sense_busy"
            td=0.2
            dd=0.1

            self.next_freq = ctfreq
            self.min_freq = ctfreq - 100000
            self.max_freq = ctfreq + 100000
            self.usrp_rate= 900e3
```

```
self.freq_step = self.nearest_freq((0.75 * self.usrp_rate), self.channel_bandwidth)

self.min_center_freq = self.min_freq + (self.freq_step/2)

steps = math.ceil((self.max_freq - self.min_freq) / self.freq_step)

self.nsteps = steps

self.max_center_freq = self.min_center_freq + (self.nsteps * self.freq_step)


self.lo_offset = 0



def bin_freq(i_bin, center_freq):


   freq = center_freq


   return freq


bin_start = int(self.fft_size * ((1 - 0.75) / 2))

bin_stop = int(self.fft_size - bin_start)

m = parse_msg(self.msgq.delete_head())


while m.center_freq!=ctfreq:

    m = parse_msg(self.msgq.delete_head())

else:

   pass


print "center frequency",m.center_freq
```

```
F = 0

T = 0


for i_bin in range(bin_start, bin_stop):



    center_freq = m.center_freq

    freq = bin_freq(i_bin, center_freq)

    #print "center frequecny", center_freq

    #noise_floor_db = -174 + 10*math.log10(tb.channel_bandwidth)print

    noise_floor_db = 10*math.log10(min(m.data)/self.usrp_rate)

    #noise_floor_db=  -174+10*math.log10(self.channel_bandwidth)+5

    #print "Noise_Floor = ", noise_floor_db

    power_db = 10*math.log10(m.data[i_bin]/self.usrp_rate) - noise_floor_db

    Absolute_Power = 10*math.log10(m.data[i_bin]/self.usrp_rate)

    #print "Absolute_Power",Absolute_Power

    #print"noise floor",noise_floor_dba

    #print "power - noise = ",power_db

    if (power_db > self.squelch_threshold) and (freq >= self.min_freq) and (freq <= self.max_freq):



        T = T+1

    else:

        F = F+1
```

```
td=0
dd=0
print "False = ", F
print "True = ", T
self.it=0
self.msgq.delete_head()
self.min_center_freq = 0
self.link.u.set_center_freq(self.link._freq)

if F > 9*T:
   print "dont change the frequency"
   return False
print "change the frequency"
return True



'''if 10*T >F:
   print "change the frequency"


   return True


print "dont change the frequency"


return False'''


except Exception,e:
```

```
        print e
```

usage_plot.py

```
import numpy as np

import matplotlib.pyplot as plt

import matplotlib.animation as animation

import sqlite3

import os

import time


class Main_Data(object):


   def __init    (self):



      self.plt = self.setup_backend()

      self.fig = self.plt.figure(num =1,figsize=(25,15))

      self.win = self.fig.canvas.manager.window

      self.win.after(1, self.animate())

      self.plt.ion()

      self.plt.show()

      while True:

         self.win.after(1, self.animate())
```

```
def setup_backend(self,backend='TkAgg'):
    import sys
    del sys.modules['matplotlib.backends']
    del sys.modules['matplotlib.pyplot']
    import matplotlib as mpl
    mpl.use(backend) # do this before importing pyplot
    import matplotlib.pyplot as plt
    return plt



def animate(self):
    global counter,freq
    db_filename ='chsel.db'
    db_flname ='spec.db'
    with sqlite3.connect(db_filename) as conn:
        conn.row_factory = sqlite3.Row
        cursor= conn.cursor()
        cursor.execute("select * from chsel")
        datas= cursor.fetchall()
        if counter !=0:
            self.plt.clf()
            del self.pwdbm[:]
            del self.centfreq[:]


        self.centfreq=[]
```

```
self.pwdbm =[]


for row in datas:


   self.centfreq.append(row['centfreq']/1e6)
   if row['sel'] =='Yes':
      freq=row['centfreq']
      print "freq is %d" %freq
      with sqlite3.connect(db_flname) as conn1:
         #conn1.row_factory = sqlite3.Row
         cursr= conn1.cursor()
         extractpwrchannel="""select pwdbm from spec where ctfreq =?"""
         cursr.execute(extractpwrchannel,(freq,))


         for pwr in  cursr.fetchmany(1):
            print "power =%f"%pwr
            self.pwdbm.append(115+pwr[0])


   else:
      self.pwdbm.append(-115)
self.plt.xlabel('Frequency in MHz')
self.plt.ylabel('Power in dBm')
self.bar_width =0.7
self.opacity=0.4
self.error_config = {'ecolor': '1'}
self.plt.ylim(-115,-90)
```

```
        self.plt.xlim(840,960)
        #if counter==0:
        self.rects= self.plt.bar(self.centfreq, self.pwdbm, self.bar_width,bottom=-115,
              alpha=self.opacity,
              color='g',
              label='Channel utilised is %f MHz  '%((freq)/1e6))

        '''else:

           [bar.set_height(self.pwdbm[i-1]) for i,bar in enumerate(self.rects)]'''

        self.plt.legend()
        counter +=1
        self.fig.canvas.draw()



if  name__ == '   main__':
   try:
      global counter,freq
      counter=0
      freq=0
      k=  Main_Data()

   except KeyboardInterrupt:

      pass
```

6. Section II: SDR Implementation of SISO/MIMO based Cognitive Radio Test bed

Objectives of the second phase of our work are augmenting the framework to provide a slew of options for proof-of-concept regarding to the cognitive radio before the user. It includes introducing SISO/MIMO modes of operation dovetailing it with OFDM which is now the industry standard of 4^{Th} Generation of Mobile Communications Technology. Here the above PHY layer techniques are coupled with Cognitive radio and targeted on GSM spectral range to test the feasibility and efficiency of such systems.

The GSM spectral range is one of the busiest pieces of spectrum chunk, utilizing it for secondary users with higher no. of sub channels was a commendable challenge. The problem faced to design such framework is of many dimensions including problems from spectral database, Synchronization of the devices utilized in the system etc. The solutions consisting of procuring devices like GPS Disciplined Oscillator, MIMO Cable and various modifications to the software packages and codes, to cater to the needs of the system. The additional USRP devices are also utilized to run the system in MIMO mode and reduce the error rate of received bits. A control layer is inserted in the framework to deftly manage the various types of application data sent and easily track the status quo of the same. A higher layer of abstraction in form of Graphical User Interface (GUI) containing all the transmission parameters to modulate according to the design of the system is available for the user. Also the GUI is attached with a suitable window, related to the choosing of specific bit array from the bitmap of a image for transmission to the receiver.

As the first part of the document very verbosely described all the necessary prerequisites of implementing a test bed, this part of the document will supplement it with more on the newer hardware parts of the system and not repeat any such redundant information. It will follow the same line of thought as of the first part in regarding to the unpacking of the newer test bed framework. The test bed also has many lacunae which will be pointed as the description takes us to the various files and classes of the codes, related to the system. As usual the technical jargons are kept to the minimum and requested to see the other complementary document delving on such matters.

Hope fully, a coherent set of instructions related to building such test bed is given and also the creators of the manual request the readers to supplant it with better and correct material as they jump into the race of implementing such test Bed.

7. Section II: Prerequisites

As clearly mentioned in the section II introduction, only the additional requirements for creating the test bed will be mentioned. Every such augmented framework requires input to both software and hardware aspects of the systems.

7. 1. a Section II: Additional Hardware prerequisites

The additional equipments procured to be integrated into the USRP or utilized to connect various USRP are

These devices under the newer framework help us to accomplish many objectives such as synchronization of USRP Using GPSDO, transmitting Sync and data information to the other slave devices using MIMO Cable attached to MIMO Expansion Slot and more USRP for MIMO mode of operation.

7.1. b Section II: Hardware configuration

The few more steps in regard to hardware configuration are as follows-:

Step 1

After configuring the WBX Daughter board in the USRP N210, another module needs to be installed inside the USRP. It produces **two references signals namely 10MHz and 1PPS. These are joined with the two ports of USRP named as "REF IN " & "PPS IN" RESPECTIVELY**. The instruction sheet for attaching such module is quite technical and some kind of supervision is necessary .The link for such instruction is attached here.[Installing GPSDO]

Such installation needs to be done on both the sides of the system, transmitter and receiver. The slave devices (additional USRP) need not be installed with GPSDO as they receive such instructions from Master devices (GPSDO installed USRP). These are quite different from OCTO-Clock as they help us to set up a node at any point and easily emulate dispersive models where each node consisting of URSP KIT is quite apart from each other.

Step 2

The MIMO cable is used to pair a URSP N210 series for swift synchronization of both time and frequency of such systems. As Shown in Figure below, Where the MIMO expansion slot is quite easily seen, the cable is plugged.

The other end of the cable is plugged into the MIMO expansion slot of the slave device.

It should look something similar to the image attached above, where the thick expansion cable is connected to Both the USRP, keeping its length and USRP position in mind. More details regarding scaling such systems using USRP and MIMO cable can be found here. [Sync & MIMO]

Step 3

After such installations, easier stuff like plugging of VERT 900 and Gigabit cable should be done in their respective port and slot. The URSP kit is now ready to be used for all kinds of operations related to SISO/MIMO mode of operation.

7.2 a Section II: Software Prerequisites

The abstraction level depends upon the flexibility of blocks available and ability of the user to program them according to the needs of the project. On taking many considerations, terminal based UHD+GNURADIO open source software was chosen to tune USRP to our needs. Although very succinctly described in the first part of the document, the additional things regarding various versions of software packages to run the current version of the codes is mentioned here.

Here it warned that, as the software utilized is open source there is no authority to keep track of the compatibility among software packages. The problem with each newer version is, the name of the command and the libraries they belong to may change acc. to the whims of the ETTUS guys. So a lot of effort will be put forth in testing various combinations of UHD and GNU Radio which could be 100% compatible with each other and the operating system of the Host. Also the older versions are timely deleted from the ETTUS servers, so the most ancient version compatible with all should be chosen for installation in the host.

7.2 b Section II: Software configuration

The various dimensions of software configuration such as installation of SQL, Python and Networking setup are dealt in the first part of the document. Here only the installation of UHD+GNURADIO is given as the codes may not be compatible with the latest versions of such packages.

7.2.b.1 INSTALLING UHD+GNURADIO

Various implementations done with same objective were clearly on older versions of Ubuntu. Among such versions **Ubuntu 10.04 LTS** was found to be the most compatible and common host OS .Then the process started to find suitable UHD and GNURADIO for successful flow of instructions from terminal to the hardware. The instructions are given below to install specific versions of software using Tar Ball files.

Install UHD_003_005_001

Step 1: Install prerequisites

```
$ sudo apt-get install libboost-all-dev libusb-1.0-0-dev python-cheetah
doxygen python-docutils
```

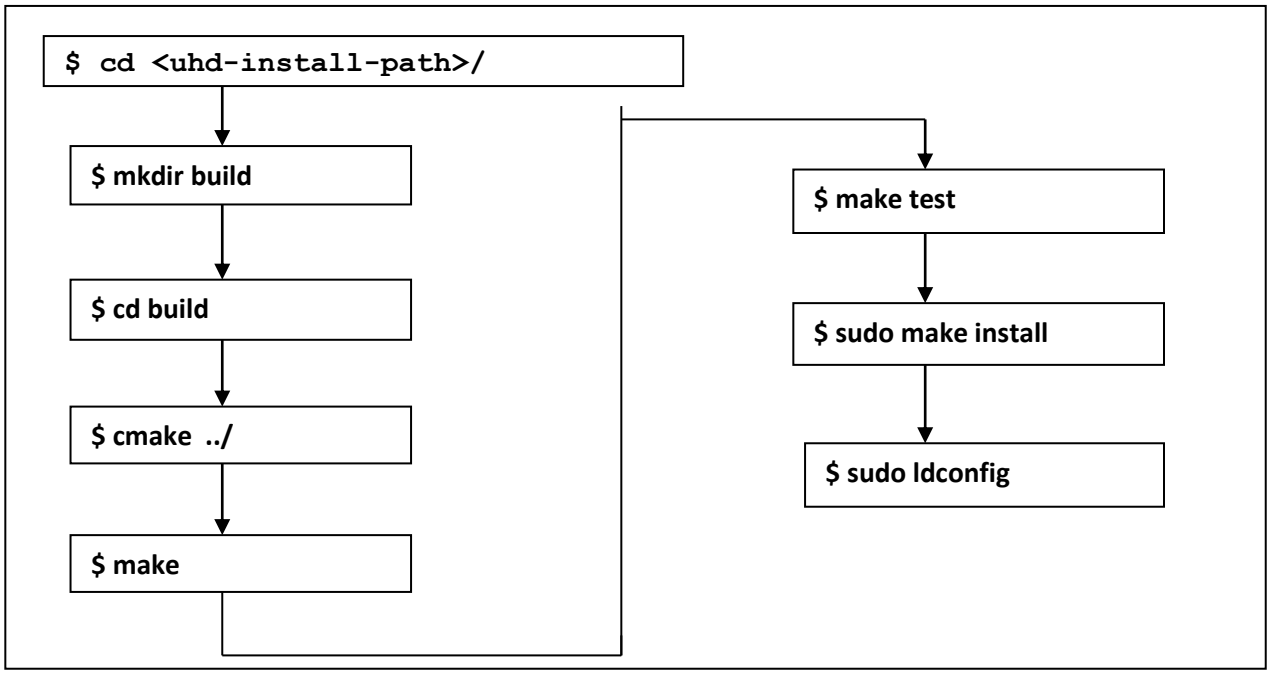
Step 2: Download UHD 003_005_001

The mentioned version can be downloaded from the following link

https://github.com/EttusResearch/UHD-Mirror/tags

Step 3: Make & Install

After downloading such binaries, the "make & install" method of Linux should be utilized for the software Package. The sequence of instructions is given below.



Step4: Install prerequisites for GNU Radio

```
$ 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
```

Step5: Download GNU Radio 3.6.0

The GNU Radio binaries also should be downloaded from ETTUS servers. The mentioned version can be downloaded from the following link.

http://gnuradio.org/releases/gnuradio/

Step 6: Install GNU Radio

To install GNU Radio from Tar Ball files, the above "make & install" method should be followed as described in the STEP 3. The first block containing instructions to switch to the UHD downloaded directory should be replaced by GNU radio downloaded directory. The rest steps should be followed as they are mentioned.

```
$ cd <gnuradio-install-path>/
```

After embarking on such a long procedure of installing so many things, I would also present some easier alternatives to this strenuous method. The installation of debian files (extension with .deb) is -a piece of cake. The files can be downloaded From ETTUS servers. They can be installed through UBUNTU software center, before any installation it finds all prerequisites for the current package and automatically installs it. As there is no modularized installation, any compatibility issue will not be handled properly. Anyway the links from where such files can be downloaded is provided below.

http://files.ettus.com/binaries/

8. Section II: Unpacking of test bed framework

The number of files and scripts work in tandem to create a framework which can sense spectrum, transmit data with choice FFT length etc. on chosen spectrum , receive adequate acknowledgements to track all packets & to serve all such statistics on silver platter, thus fulfilling functions of the Test Bed. Another way is to lock on specific functions or Class definitions which can show a baser module of such framework and give insight on the actual workings of the code.

8.1 Files and Scripts

This section contains list of files, scripts and their roles played within the functions of the framework. It also contains a flow graph depicting the relationship among files and scripts.

(A) **CR_TX** This is the most important script of the framework, it co-ordinates with all other files and is the center point of both the transmission and receiver sections of the TX side.

(B) Spec_Sense Another script which senses the amount of spectrum chunk as ordained by the user, writes to the database and provides easier options to change all parameters of the script easily.

(C) Co-ordination_module This file is a part of the control layer where it interacts with database, extracts all requisite blocks satisfying spectrum requirements and adds guard band to the chosen block. It then send the necessary information for tuning the device to the selected frequency, If such spectrum blocks exist.

(D) Global_control _unit This file is the pit for accumulation of all information regarding creation of transmission and reception blocks of the Tx side .It provides immediate control of all the parameters , which are fed to blocks for creation of pathways for flow of instructions from terminal to device and vice versa.

(E) Send_Thread This file is the repository of all the modularized sections pertaining to various applications choices provided at the interface. The control layer's commands are also placed in those sections related to tagging of the packets and a tracking system for the same.

(F) Tb_options This is the axis of the framework around which all of its operations revolves. It is the baser interface for all transmission and receptions parameters which can be changed at the GUI interface by the user.

(G) Top_block This file creates the transmission and reception pathways after receiving adequate instructions from all other files .All the choices received from the user are utilized to tune the USRP. It creates separate pathways for transmission and reception section and jump start the system.

(H) Ctrl_panel This panel creates a window GUI for receiving instructions on various parameters, importantly for starting and stopping of the system.

(I) Foi_mimo_tx_path The file receives instructions from Top_block for dovetailing its pathway with the Transmission section of the Tx side. It consists of two pathways catering to the needs of the user for choosing Coded or non-coded MIMO mode of operation

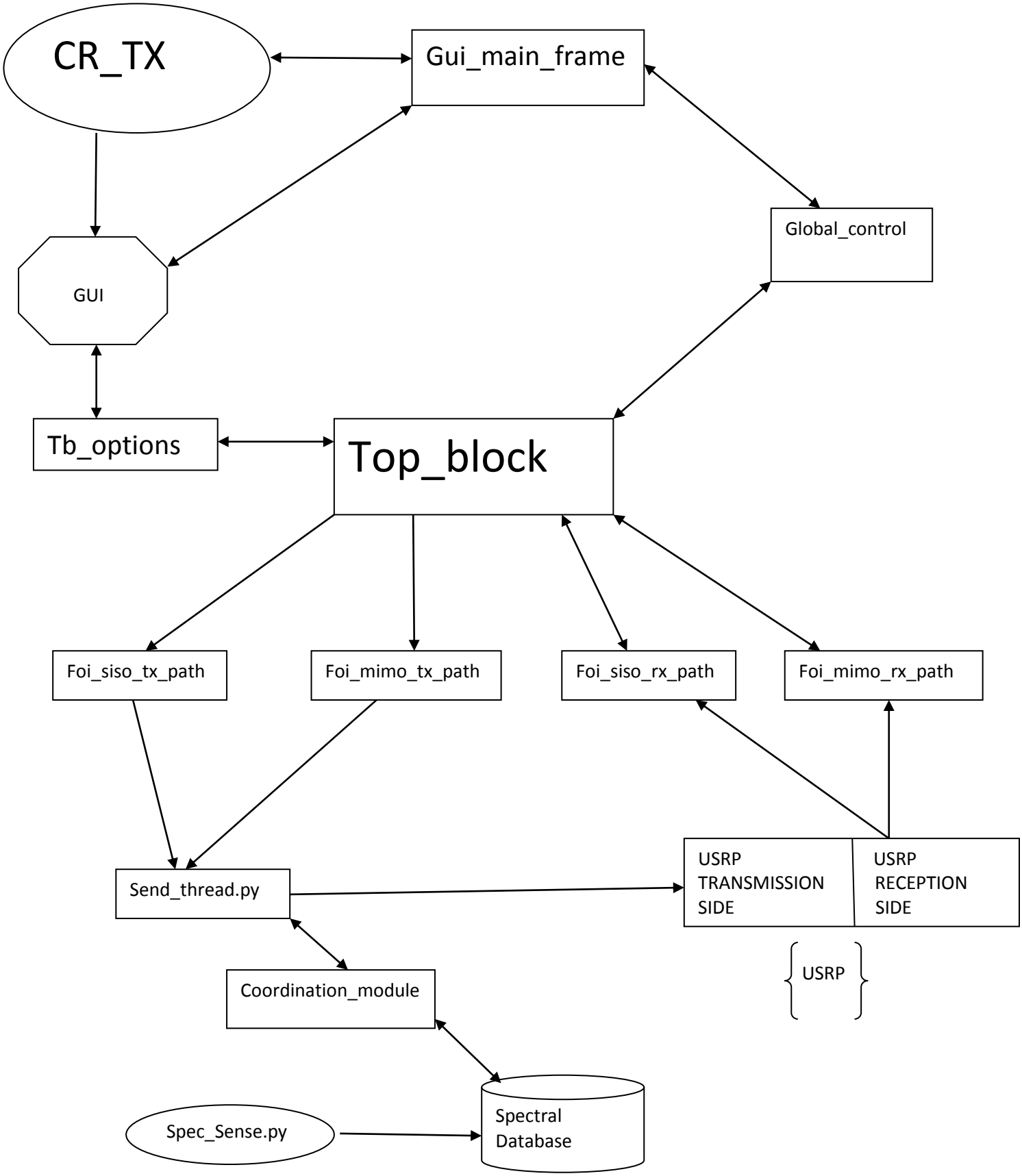
(J) Foi_siso_tx_path The file receives instructions from Top_block for dovetailing its pathway with the Transmission section of the Tx side. It consists of two pathways catering to the needs of the user for choosing Coded or non-coded SISO mode of operation.

(K) Foi_mimo_rx_path The file receives instructions from Top_block for dovetailing its pathway with the reception section of the Tx side. It consists of two pathways catering to the needs of the user for choosing Coded or non-coded MIMO mode of operation

(L) Foi_siso_rx_path The file receives instructions from Top_block for dovetailing its pathway with the reception section of the Tx side. It consists of two pathways catering to the needs of the user for choosing Coded or non-coded SISO mode of operation

(M) GUI_main_frame_tx The file co-ordinates all the windows of the GUI and provides the highest abstraction regarding capture of instructions emanating from GUI.

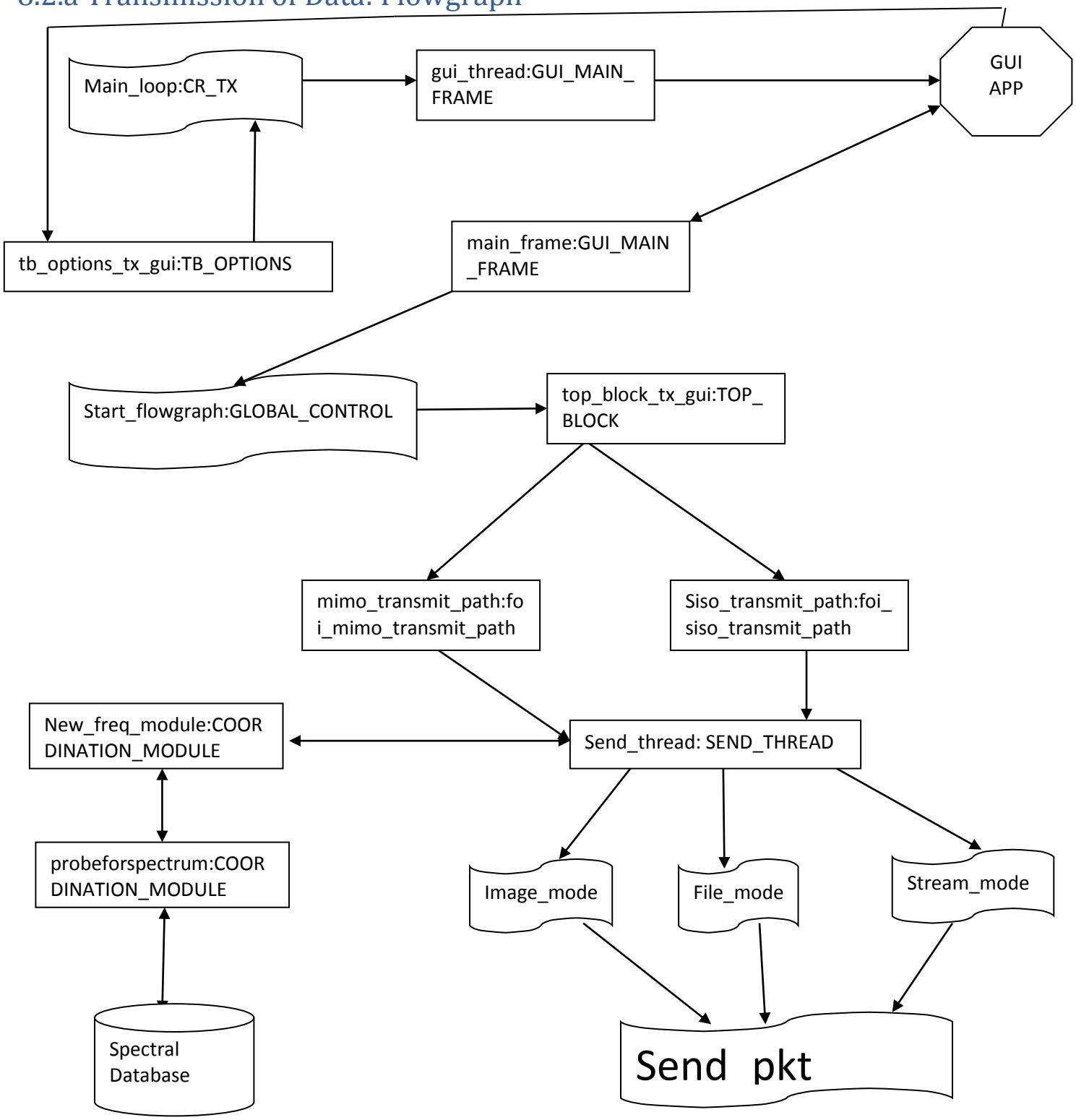
The files are executed as per the command structure embedded in CR_TX script and eventually everything can be traced to and from this script. Although such framework involves flow of instructions among various files in different sequences, the system doesn't trundle at all. As it is highly modularized with clear set of instructions for a specific task, changing and correcting them isn't cumbersome.


CR_TX
Gui_main_frame
GUI
Global_control
Tb_options
Top_block
Foi_siso_tx_path
Foi_mimo_tx_path
Foi_siso_rx_path
Foi_mimo_rx_path
Send_thread.py
USRP TRANSMISSION SIDE
USRP RECEPTION SIDE
USRP
Coordination_module
Spectral Database
Spec_Sense.py

8.2 CLASSES AND FUNCTIONS

The previous section had shown the flow of instructions from terminal to USRP using files and scripts as the basic blocks. In this section, the objective is to demonstrate the path of a packet in either transmission mode through various classes and functions. As we present the flow graphs fulfilling the above objective, we will also expound on all the mentioned classes and functions. It will probe deeper into their input parameters and their working on such parameters.

8.2.a Transmission of Data: Flowgraph



8.2 .b Transmission of Data: An exposition of classes & functions

The flow graph traces out the path of instructions coupled with data among such a complex mixture of files and scripts. In the notation of each block, functions are mentioned along with the files, as not to lose the bigger picture. Although it doesn't cover all functions and classes, still it provides a lower abstraction of the framework.

8.2.b.1 main_loop:CR_TX

FUNCTION It is fundamentally the main thread along which WX APP(Basic for any GUI), Gui_main_frame and global_ctrl_unit class are created. It also sets up the virtual port and provides an IP to it. It also caters for the creation of modularized acknowledgment section for each type of application data such as image and file. The statistics related to the packet received etc. are also generated by this function.

PARAMETERS It receives the acknowledgment packets and all transmission parameters.

8.2.b.2 gui_thread:GUI_MAIN_FRAME

FUNCTION It creates a daemon thread and launches the WXAPP .So it ensures that irrespective of main thread ending Its operations, this thread won't die. This allows the app to be active for any number of operations regarding burst of Data from TX side.

PARAMETRS It takes cue from the WX app declared in the main loop of CR_TX script.

8.2.b.3 main_frame:GUI_MAIN_FRAME

FUNCTION It performs multitude operations including creation of all the windows in the WXapp, calculating the statistics And provides information to the main_loop of CR_TX. It also manages the start and stop operations of the TOP_BLOCK. It also provides a pathway for the call_back operation to reach the main_loop.

PARAMETERS It receives all parameters from the main_loop during its initiation from the same.

8.2.b.4 tb_options_tx_gui: TB_OPTIONS

FUNCTION It is the repository of all the relevant parameters related to transmission and reception. This class provides easy access to such parameters from any point in the framework.

PARAMETERS NONE

8.2.b.5 start_flowgraph: GLOBAL_CONTROL

FUNCTION It receives all the parameters from GUI_MAIN_FRAME class and initiates the creation of top_block_tx_rx and Send thread class. It also attaches the data to be sent, to the SEND_THREAD class.

PARAMETERS It receives all the transmission parameters and data to be sent.

8.2.b.6 top_block_tx_gui: TOP_BLOCK

FUNCTION According to the choices ordained by the user at the GUI, it carves out the SISO or MIMO pathways along with various other parameters. It also connects all the pathways with the send thread class to send the packet to its modularized application section and finally transmission.

PARAMETERS It receives all user choices from the GUI.

8.2.b.7 mimo_transmit_path: FOI_MIMO_TRANSMIT_PATH

FUNCTION It is another hierarchical block in the pathway created by TOP_BLOCK , under which all block pertaining to MIMO operation is placed. The class provides two distinct pathways for coded and non coded mode of operation and connects all the relevant blocks of it. Also functions like controlling the amplitude of signals and connecting all the pathways to send_thread class is available.

PARAMETERS It receives all user choices from the TOP_BLOCK.

8.2.b.8 siso_transmit_path: FOI_SISO_TRANSMIT_PATH

FUNCTION It is another hierarchical block in the pathway created by TOP_BLOCK , under which all block pertaining to SISO operation is placed. The class provides two distinct pathways for coded and non coded mode of operation and connects all the relevant blocks of it. Also functions like controlling the amplitude of signals and connecting all the pathways to send_thread class is available.

PARAMETERS It receives all user choices from the TOP_BLOCK.

8.2.b.8 new_freq_module: COORDINATION_MODULE

FUNCTION On the call of the control layer for new set of spectrum chunk, it performs multitude of operations like scanning database etc. and finally passing on such information to the Send_thread class.

PARAMETERS The required no. of bins and the current frequency

8.2.b.8 probe_for_spectrum: COORDINATION_MODULE

FUNCTION It actually interacts with the database in the concerned frequency range, extracts all such blocks satisfying the bandwidth requirements of the USER. It also sorts them and allocates guard band around them. Finally It calculates the center frequency of the best block and send such information to the new_freq_module of the same class.

PARAMETERS Required no. of bins for transmission.

8.2.b.8 send_thread(run): SEND_THREAD

FUNCTION This class provides separate pathways according to the type of data to be sent to the receiver. It also connects to the coordination_module for information regarding available spectrum chunk. The control layer is also in the send_thread class performing all of its desired functions.

PARAMETERS Literally every other parameter of the main_loop of CR_TX script.

8.2.b.9 image_ mode(run): SEND_THREAD

FUNCTION This function receives the data to be sent from GUI_MAIN_FRAME class. It extracts a certain length of data from the received data and tags it as the image mode packet. It also ask for the status quo on the current channel being used for Transmission from control layer.

PARAMETERS Transmission parameters and bit array from the bit map of the image.

8.2.b.10 file_ mode(run): SEND_THREAD

FUNCTION This function receives the data to be sent from file location given in the tb_options _tx_gui from TB_OPTIONS class. It extracts a certain length of data from the received data and tags it as the file mode packet. It also asks for the status quo on the current channel being used for Transmission from control layer.

PARAMETERS Transmission parameters and data from the file .

8.2.b.10 stream_ mode(run): SEND_THREAD

FUNCTION This function receives the data to be sent from concerned OS applications running concurrently along the WXAPP. It forwards a certain length of data and tags it as the stream mode packet. It also asks for the status quo on the current channel being used for Transmission from control layer.

PARAMETERS Transmission parameters and data from the OS Application.

8.2.b.11 Spectral Database & Send_pkt

FUNCTION The spectral database contains records of bins of certain bandwidth ordained by user and provides power profile and availability status of each bin .This is generated by spec_sense.py SCRIPT and runs continuously in the background providing the current report on the bins. Send_pkt functions finally transmits the packet in one or multiple antennas as chosen by the user.

PARAMETERS Range of frequency to be scanned, size of bins to be created and mode of operation whether SISO or MIMO.

9. Section II: Building a Cognitive Radio Test Bed

After coming across all the scripts and files within the purview of the framework, It becomes imperative to see all of them working together as a coherent unit and running on USRP. The instructions are just to initiate scripts along Rx and Tx side sequentially, plug the choices into the concerned GUI of the framework and kick start the system on both the sides.

To demonstrate the steps taken, the transmission of image from Tx side to Rx side is chosen. Also the various windows of the GUI is depicted along the steps to connect them easily. Before initiating any such scripts, all USRP must be assigned a unique IP on Tx or Rx side of the network. For e.g If the IP is 192.168.20.2, we must run the following command at the terminal and wait for response.

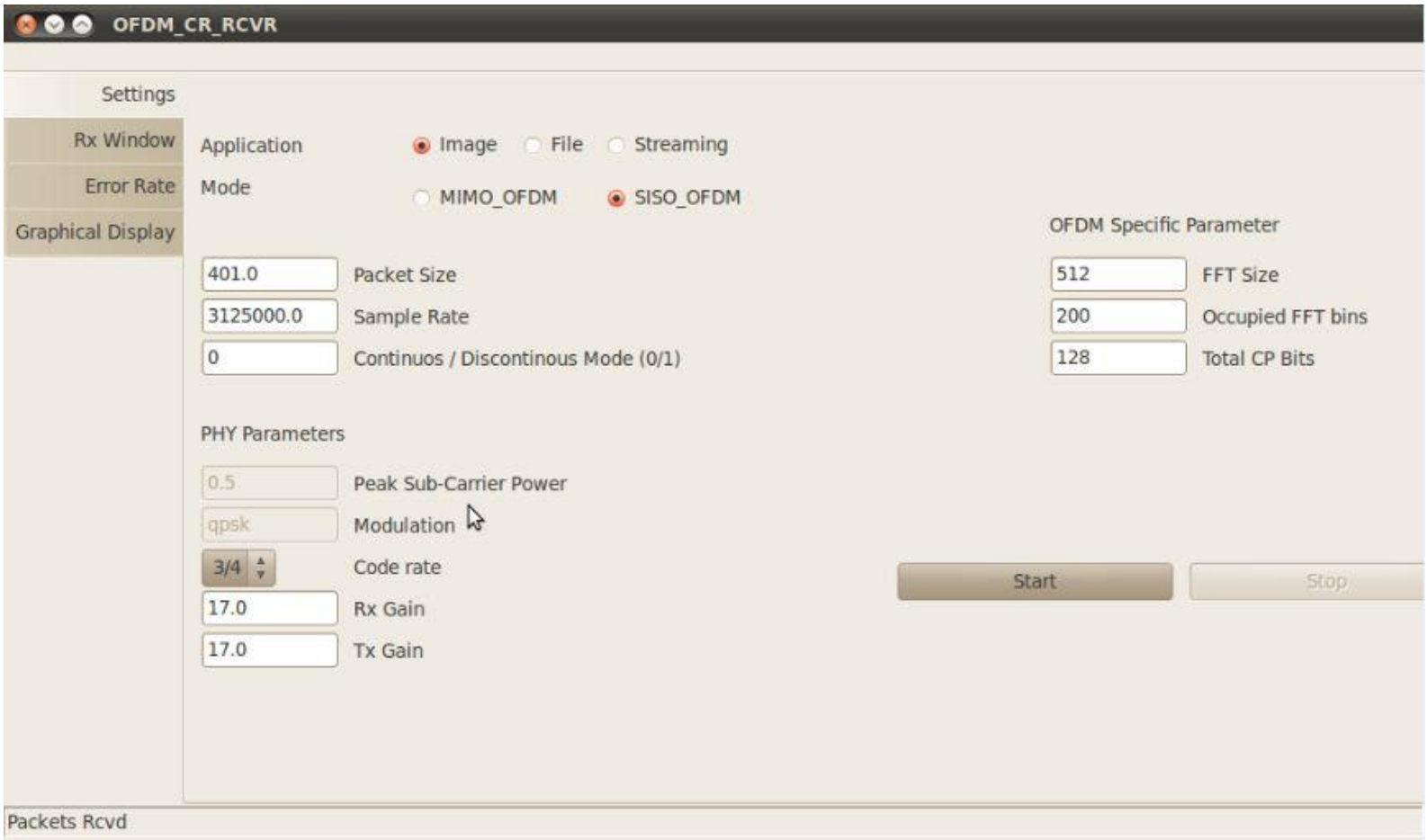
```
$ ping 192.168.20.2
```

STEP 1 SET UP THE Rx

To initiate the GUI on the RX side, run the CR_RX script at the terminal.

```
$ sudo ./CR_RX .PY
```

After initiating the script, it would pop up a GUI as shown below.



STEP 2 PLUG THE PARAMETERS

In the GUI, the user plugs the parameters according to the system model chosen (here image mode) and CLICKS ON THE START BUTTON.

STEP3 SWITCH TO Rx WINDOW

After starting the Rx side, it is wished that we switch to "RX window of the GUI" as we are running the image mode.

As it is clearly depicted, the image transmitted from the Tx side will finally land up on the sub windows of the Rx window. The image received would give a fair idea about the differences between SISO & MIMO mode. To make the process easier, the CLEAR AND START button are also given on this window. It would facilitate in the process of clearing the image for next image and starting the process without going back to SETTINGS window.

STEP 4 SET UP THE TX:populate the database

After initiating the Rx side, the spectral database needs to be populated before running CR_TX script on the Tx side.

To populate the database the SPEC_SENSE.py script needs to fired at the terminal.

```
$ sudo ./spec_sense.py 900M 950M
```

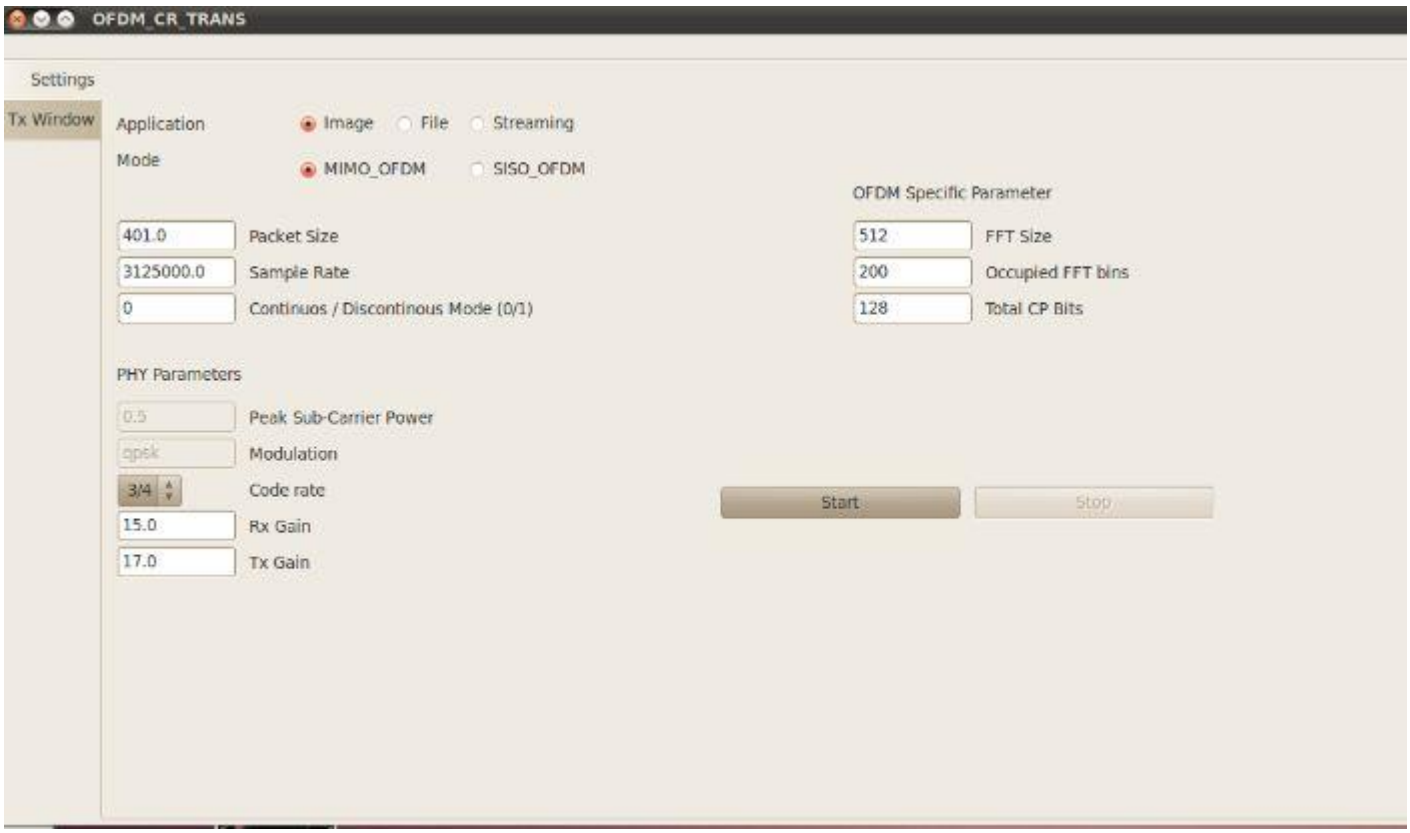
The following command would scan frequencies from 900 MHz to 950MHz and would update the records of such database every ~ 3 seconds.

STEP 5 SET UP THE TX: Initiate the Script

After waiting for a few seconds so that the database would be populated with all the data required for each bin, We initiate the Script.

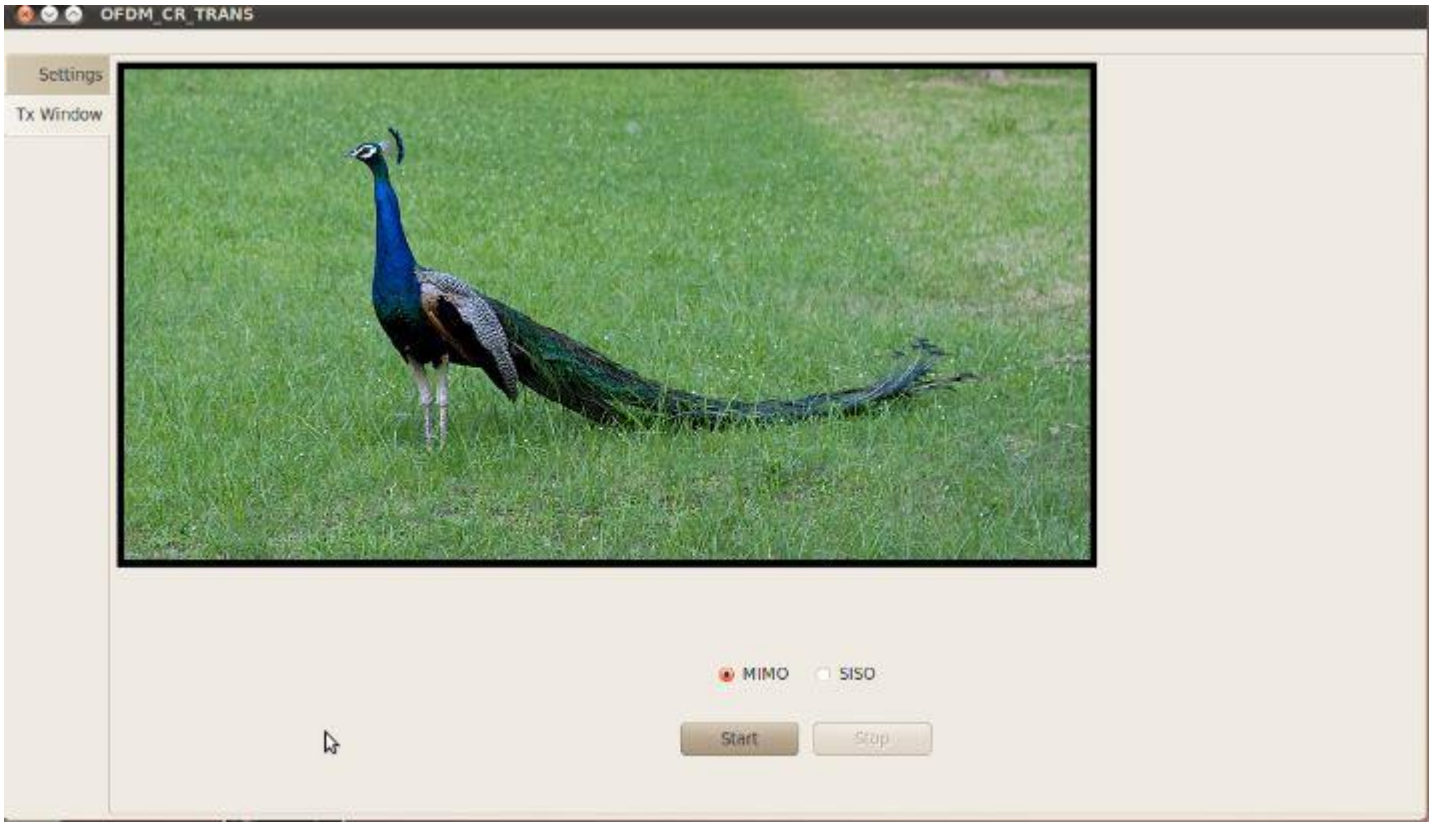
```
$ sudo ./CR_TX.py
```

Once we run this script at the terminal, the GUI of the Tx side would pop up.



STEP 6 SET UP THE TX: PLUG THE PARAMETERS & SWITCH TO TX WINDOW

As we have the chosen the image mode, please plug the same parameters on Tx GUI as done at the Rx GUI and Switch to the Tx window of the GUI. It would appear something as given below-:



Move the cursor to choose a section of the image and transfer the bit array of the chosen section to the GUI_MAIN_FRAME. Choose the mode of operation and CLICK THE START BUTTON available on the window. After clicking on the start button, It would perform all operations as mentioned in the document. Please refer to Rx window of RX GUI so as to see the received images in both the modes. Switch on to other windows to capture the packet error rate and constellation plots of the received data.

10. Codes

The framework is devoured by an exposition of all files and scripts along with detailed explanations of classes and files. Here all those files and scripts are given to provide a quick reference to a person reading this document.

10.1 CR_TX.py

```
#!/usr/bin/env python
from foimimo import *

from foimimo.foi_mimo_tx_path import mimo_transmit_path as mimo_tx_path
from foimimo.foi_siso_tx_path import siso_transmit_path as siso_tx_path

from gnuradio import gr
from gnuradio import uhd
import time, struct, sys, math
import gnuradio.gr.gr_threading as _threading
import wx,os,subprocess

# import local gui classes
from gui.foimimo_gui_main_frame_tx import GUI_thread, main_frame
from gui.foimimo_events import *
from gui.foimimo_global_control_unit import global_ctrl_unit
from gui.foimimo_sendThread import Send_thread

VERBOSE = 0
global k,pkt_num_rcvd,freq_ack
pkt_num_rcvd=0
k=0
freq_ack=0
#tunelling
IFF_TUN                 = 0x0001
IFF_TAP                 = 0x0002
IFF_NO_PI     = 0x1000
IFF_ONE_QUEUE = 0x2000
IFF_VNET_HDR = 0x4000
IFF_MULTI_QUEUE =0x0100



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

def main():
   global n_rcvd, n_right,per_history #packet history
   per_history =[]

   def open_tun_interface(tun_device_filename):
      from fcntl import ioctl
```

```
        mode = IFF_TAP | IFF_NO_PI
        TUNSETIFF = 0x400454ca


         # Open control device and request interface
        tun = os.open(tun_device_filename, os.O_RDWR)
        ifs = ioctl(tun, TUNSETIFF, struct.pack("16sH", "gr%d", mode))

        # Retreive real interface name from control device
        ifname = ifs[:16].strip("\x00")
        return (tun, ifname)

      def rx_callback(ok, payload):
        print "**********yipeee in the rcvd section ****************************"
        global per_history,k,freq_ack              # packet error history
        options = global_ctrl.get_options()
        k=k+1
        print "payload number",k
        os.write(tun_fd,payload)
        if ord(payload[0])==99 and ord(payload[1]) ==104 and ord(payload[2])==103:
          freq =
ord(payload[3])*10e7+ord(payload[4])*10e6+ord(payload[5])*10e5+ord(payload[6])*10e4+ord(payload[7])*10e3
          if freq_ack!= freq:
            freq_ack= freq
            print "*****************************************************Freq change  packet
Acknowledged*************************************************************************"
            options.cog_pkt_rcvd =1
          else:
            print "*****************************************************FReq already
Acknowledged*************************************************************************"

            options.cog_pkt_rcvd =1
            #e.set()


        elif ord(payload[0])==115 and ord(payload[1])==116 and ord(payload[2])==114 and options.streaming_mode==1:
          print "streaming mode"
          os.write(tun_fd,payload[5:])
          print "packets written to os"
          (pktnum,) = struct.unpack('!H', payload[3:5])
          print "Rcvd pkt number %d"%(pktnum)
          if options.pkt_num_rcvd!= pktnum and pktnum==options.pkt_num_rcvd+1:
            options.pkt_num_rcvd= pktnum
            options.e1.set()
            options.e1.clear()
          print "\n"
          sys.stdout.flush()

        else:
```

```
        (pkt_num,)= struct.unpack('!H',payload[0:2])

        # Received one packet
        if pkt_num==(options.pkt_num_rcvd+1):
        print "################################################pkt_num
rcvddddddddddddddddddddddddddddddddddddddddd#################################################################
#######",pkt_num
            #pkt_num_rcvd = pkt_num
            options.pkt_num_rcvd = pkt_num
            bit_rcvd = 0
            bit_right = 0
            n_rcvd = 1
            n_right = 0
            if ok:
               n_right = 1

            # Calculate a floating packet error rate
            if ok: per_history.append(1)
            else: per_history.append(0)
            if ok:
               options.rcv = 1
            per_avg_nr = options.per_avg_nr # nr of packets to make average over
            if n_rcvd > per_avg_nr: per_history.pop(0)
            per_avg = (len(per_history)-sum(per_history))     # none=0, max=per_avg_nr(eller n_rcvd)
            per_avg = 100*(float(per_avg)/len(per_history))    # in %

            '''# Post event to update the per graph in the gui:
            event = per_event(myEVT_PER,-1)
            event.set_per_val(per_avg)
            wx.PostEvent(frame.tech_panel.on_per_event(event),event)

            # Post event to update received data image in the gui:
            if (ok or options.write_all):
               event = rcvd_data_event(myEVT_PKT,-1)
               event.set_data(payload)
               wx.PostEvent(frame.demo_panel.on_rcvd_data_event(event),event)'''
            # Receive data and print to screen
            if options.image_mode:
               print "TB: Got data with len", len(payload)," status:",ok
               sys.stdout.flush()

               if VERBOSE:
                  printlst = list()
                  for x in payload:
                        t = hex(ord(x)).replace('0x', '')
                        if(len(t) == 1):
                              t = '0' + t
                        printlst.append(t)
                  printable = ''.join(printlst)
```

```
            print printable
            print "\n"
            sys.stdout.flush()

        # Receive data and write to file
        elif options.file_mode:
            if True:
                print "TB: Got data with len", len(payload)," status:",ok
                sys.stdout.flush()

            else:
                pass
        else:
            print "packet discarded"
            print "last pkt num",options.pkt_num_rcvd
        # Receive data and print to screen
        '''elif options.benchmark_mode:
            if VERBOSE:
                printlst = list()
                for x in payload:
                    t = hex(ord(x)).replace('0x', '')
                    if(len(t) == 1):
                        t = '0' + t
                    printlst.append(t)
                printable = ''.join(printlst)

                print printable
                print "\n"
                sys.stdout.flush()
        elif options.ber_mode:
            bit_rcvd += 8*len(payload[2:])
            if ok:
                (pktno,) = struct.unpack('!H', payload[0:2])
                bit_right += 8*len(payload[2:])
            else:
                for x in payload[2:]:
                    xored = ord(x)^0 # we send 0 (zeros as payload)
                    if xored == 0:
                        bit_right += 8
                    else:
                        #convert to a string of '0' and '1'
                        xored = bin(xored) #obs, count ones not zeros
                        xored = xored[2:]
                        bit_right = 8
                        for b in xored:
                            bit_right -=int(b)
        global_ctrl.update_statistics(n_rcvd, n_right, bit_rcvd, bit_right)'''

    ###--- Bad header counter - Callback ---###
    def bad_header_callback():
```

```
        global_ctrl.update_bad_header(1)

    ###--- Reset all counters etc. ---###
    def reset_main_variables():
        global n_rcvd, n_right
        n_rcvd = 0
        n_right = 0
        global per_history  # packet error history
        per_history = []    # init

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

    ###--- GUI ---###
    (tun_fd, tun_ifname) = open_tun_interface("/dev/net/tun")
    app = wx.App(False)
    global_ctrl = global_ctrl_unit("tx_gui")
    frame = main_frame(rx_callback,bad_header_callback,reset_main_variables,global_ctrl,app,tun_fd)
    gui = GUI_thread(app)
    print "Allocated virtual ethernet interface: %s" % (tun_ifname,)
    print "You must now use ifconfig to set its IP address. E.g.,"
    print "  $ sudo ifconfig %s 192.168.200.1" % (tun_ifname,)
    print "tun_fd",tun_fd
    subprocess.call('sudo ifconfig %s 192.168.200.1' %(tun_ifname),shell=True)
    print "Virtual ip configured at the terminal"

    while global_ctrl.GUIAlive:
        time.sleep(1)

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

10.2 COORDINATION_MODULE

```
#!/usr/bin/env python
import sqlite3

class newfreq_module():
    def __init__(self,options,freq):

        currentfreq = freq #current freq
        pb = probeforspectrum(options)
        pb.generate_hexbin_list(options)
        pb.add_guard_band(pb.binlist,2)
```

```
        pb.sorted_avail_list =pb.get_avail_block_sorted(pb.new_avail_subc_bin)
        demand= int(options.sample_rate/20e3)+10
        self.newfreq = pb.get_ctfreq(demand,pb.sorted_avail_list,options,currentfreq)



class probeforspectrum():

    def __init__(self,options):

        self.min=900e6
        self.max=950e6
        self.db_flname ='spec.db'

    def get_ctfreq(self,demand,slist,options,freq):
        previous =freq
        for i in range(len(slist)):
            sb_crr_length = slist[i][0]
            if sb_crr_length > demand:
                ct_freq = self.min+(slist[i][1])*20e3 +(slist[i][0]/2)*20e3+10e3
            if ct_freq ==previous:
                pass
            else:
                #previous=ct_freq
                print "ct_freq",ct_freq
                return ct_freq



    def get_avail_block_sorted(self,subc_bin):
        block_i = []
        block = []
        length = 0
        if (subc_bin[0] == 1):
            index = 0
            block_i.append(index)
        for i in range(1, len(subc_bin) -1):
            if subc_bin[i-1] == 0 and subc_bin[i] == 1:
                index = i
                block_i.append(index)
            if subc_bin[i-1] == 1 and subc_bin[i] == 0:
                length = i - index
                block_i.insert(0,length)
                block.append(block_i)
                block_i = []
        if (subc_bin[len(subc_bin) - 1] == 1) :
            length = len(subc_bin) - index
            block_i.insert(0,length)
            block.append(block_i)
        block.sort()
```

```
    block.reverse()
    #print subc_bin
    print "block_list",block
    return block


def add_guard_band(self,avail_subc_bin, gbsize):
    self.new_avail_subc_bin = avail_subc_bin
    last = int(avail_subc_bin[0])
    for i in range(0, len(avail_subc_bin)):
        current = int(avail_subc_bin[i])
        if current != last:
            if current == 1:
                for j in range(0, gbsize):
                    if i-j-1 >= 0:
                        self.new_avail_subc_bin[i-j-1] = 0
            else:
                for j in range(0, gbsize):
                    if i+j < len(avail_subc_bin):
                        self.new_avail_subc_bin[i+j] = 0
        last = current

#
    for i in range(0, len(self.new_avail_subc_bin)/2):
        if int(self.new_avail_subc_bin[2*i]) == 0 or int(self.new_avail_subc_bin[2*i+1]) == 0:
            self.new_avail_subc_bin[2*i] = 0
            self.new_avail_subc_bin[2*i + 1] = 0
    #print self.new_avail_subc_bin
    return self.new_avail_subc_bin

def get_subc_block_sorted(subc_bin):
    block_i = []
    block = []
    length = 0
    if (subc_bin[0] == 1):
        index = 0
        block_i.append(index)
    for i in range(1, len(subc_bin) -1):
        if subc_bin[i-1] == 0 and subc_bin[i] == 1:
            index = i
            block_i.append(index)
        if subc_bin[i-1] == 1 and subc_bin[i] == 0:
            length = i - index
            block_i.insert(0,length)
            block.append(block_i)
            block_i = []
    if (subc_bin[len(subc_bin) - 1] == 1) :
        length = len(subc_bin) - index
        block_i.insert(0,length)
        block.append(block_i)
```

```
        block.sort()
        block.reverse()
        return block

    def avail_bin2str(self,subc_bin):
    #convert available binary list to string
        subc_str = ""
        for i in range(0, len(subc_bin)):
            if (i+1) % 4 == 0:
                tmp = int(subc_bin[i-3]*8 + subc_bin[i-2]*4 + subc_bin[i-1]*2 + subc_bin[i])
                subc_str = subc_str + hex(tmp).replace('0x',"")
        return subc_str



    def generate_hexbin_list(self,options):
        #create string of hexadecimal and binary based notation of available blocks
        self.centfreq=[]
        self.binlist=[]
        with sqlite3.connect(self.db_flname) as conn:
            conn.row_factory = sqlite3.Row
            cursor= conn.cursor()
            cursor.execute("PRAGMA synchronous=OFF")
            cursor.execute("PRAGMA journal_mode=WAL")
            query = "select * from spec where ctfreq>=? and ctfreq<=?"
            cursor.execute(query,(self.min,self.max,))
            datas= cursor.fetchall()
            for row in datas:
                self.centfreq.append(row['ctfreq']/1e6)
                if row['status'] == 'Available':
                    self.binlist.append(1)
                else:
                    self.binlist.append(0)


            self.string = self.avail_bin2str(self.binlist)
```

10.3 SPEC_SENSE

```
#!/usr/bin/env python
from gnuradio import gr, eng_notation,window
#from gnuradio import blocks
from gnuradio import audio
#from gnuradio import filter
#from gnuradio import fft
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import sys
import math
import struct
```

```
import threading
from datetime import datetime
import logging
import time
import subprocess
import sqlite3
import os

# logging format

logging.basicConfig(level= logging.DEBUG,format = '%(asctime)s (%(threadName) -10s) %(message)s ')


class tune(gr.feval_dd):
    """
    This class allows C++ code to callback into python.
    """
    def __init__(self, tb):
        gr.feval_dd.__init__(self)
        self.tb = tb

    def eval(self, ignore):
        """
        This method is called from blocks.bin_statistics_f when it wants
        to change the center frequency.  This method tunes the front
        end to the new center frequency, and returns the new frequency
        as its result.
        """

        try:
            # We use this try block so that if something goes wrong
            # from here down, at least we'll have a prayer of knowing
            # what went wrong.  Without this, you get a very
            # mysterious:
            #
            #   terminate called after throwing an instance of
            #   'Swig::DirectorMethodException' Aborted
            #
            # message on stderr.  Not exactly helpful ;)


            new_freq = self.tb.set_next_freq()

            # wait until msgq is empty before continuing
            while(self.tb.msgq.full_p()):
                #print "msgq full, holding.."
                time.sleep(0.1)

            return new_freq
```

```
        except Exception, e:
            print "tune: Exception: ", e


class parse_msg(object):
    def __init__(self, msg):
        self.center_freq = msg.arg1()
        self.vlen = int(msg.arg2())
        assert(msg.length() == self.vlen * gr.sizeof_float)

        # FIXME consider using NumPy array
        t = msg.to_string()
        self.raw_data = t
        self.data = struct.unpack('%df' % (self.vlen,), t)


class my_top_block(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self)

        usage = "usage: %prog [option] BS min_freq max_freq"

        parser = OptionParser(option_class=eng_option, usage=usage)
        parser.add_option("-a", "--args", type="string", default="addr=192.168.10.2",
                          help="UHD device device address args [default=%default]")
        parser.add_option("", "--spec", type="string", default="A:0",
                           help="Subdevice of UHD device where appropriate")
        parser.add_option("-A", "--antenna", type="string", default="RX2",
                          help="select Rx Antenna where appropriate")
        parser.add_option("-s", "--samp-rate", type="eng_float", default=25e6,
                          help="set sample rate [default=%default]")
        parser.add_option("-g", "--gain", type="eng_float", default=None,
                          help="set gain in dB (default is midpoint)")
        parser.add_option("", "--tune-delay", type="eng_float",
                          default=0.25, metavar="SECS",
                          help="time to delay (in seconds) after changing frequency [default=%default]")
        parser.add_option("", "--dwell-delay", type="eng_float",
                          default=0.25, metavar="SECS",
                          help="time to dwell (in seconds) at a given frequency [default=%default]")
        parser.add_option("-b", "--channel-bandwidth", type="eng_float",
                          default=20e3, metavar="Hz",
                          help="channel bandwidth of fft bins in Hz [default=%default]")
        parser.add_option("-l", "--lo-offset", type="eng_float",
                          default=0, metavar="Hz",
                          help="lo_offset in Hz [default=%default]")
        parser.add_option("-q", "--squelch-threshold", type="eng_float",
                          default=10, metavar="dB",
                          help="squelch threshold in dB [default=%default]")
        parser.add_option("-F", "--fft-size", type="int", default=None,
```

```
                help="specify number of FFT bins [default=samp_rate/channel_bw]")
        parser.add_option("", "--real-time", action="store_true", default=False,
                help="Attempt to enable real-time scheduling")

        (options, args) = parser.parse_args()
        if len(args) != 2:
           parser.print_help()
           logging.debug("exiting")
           sys.exit(1)

        self.channel_bandwidth = options.channel_bandwidth

        self.min_freq = eng_notation.str_to_num(args[0])
        self.max_freq = eng_notation.str_to_num(args[1])

        if self.min_freq > self.max_freq:
           # swap them
           self.min_freq, self.max_freq = self.max_freq, self.min_freq

        if not options.real_time:
           realtime = False
        else:
           # Attempt to enable realtime scheduling
           r = gr.enable_realtime_scheduling()
           if r == gr.RT_OK:
              realtime = True
           else:
              realtime = False
              print "Note: failed to enable realtime scheduling"

        # build graph


        self.u = uhd.usrp_source(device_addr="addr=192.168.10.2",io_type =
uhd.io_type.COMPLEX_FLOAT32,num_channels=1)

        # Set the subdevice spec
        if(options.spec):
           self.u.set_subdev_spec(options.spec, 0)

        # Set the antenna
        if(options.antenna):
           self.u.set_antenna(options.antenna, 0)

        self.u.set_samp_rate(options.samp_rate)
        self.usrp_rate = usrp_rate = self.u.get_samp_rate()

        self.lo_offset = options.lo_offset

        if options.fft_size is None:
```

```
    self.fft_size = int(self.usrp_rate/self.channel_bandwidth)
else:
    self.fft_size = options.fft_size

self.squelch_threshold = options.squelch_threshold

s2v = gr.stream_to_vector(gr.sizeof_gr_complex, self.fft_size)


mywindow = window.blackmanharris(self.fft_size)
#print mywindow
ffter = gr.fft_vcc(self.fft_size, True, mywindow, True)
#print ffter
power = 0
for tap in mywindow:
    power += tap*tap

c2mag = gr.complex_to_mag_squared(self.fft_size)

# FIXME the log10 primitive is dog slow
#log = blocks.nlog10_ff(10, self.fft_size,
#                       -20*math.log10(self.fft_size)-10*math.log10(power/self.fft_size))

# Set the freq_step to 75% of the actual data throughput.
# This allows us to discard the bins on both ends of the spectrum.

self.freq_step = self.nearest_freq((0.75 * self.usrp_rate), self.channel_bandwidth)
print "freq step" ,self.freq_step
self.min_center_freq = self.min_freq + (self.freq_step/2)
print " min center freq" ,self.min_center_freq
nsteps = math.ceil((self.max_freq - self.min_freq) / self.freq_step)
self.nsteps = nsteps
print "n steps", nsteps
self.max_center_freq = self.min_center_freq + (nsteps * self.freq_step)

self.next_freq = self.min_center_freq

tune_delay  = max(0, int(round(options.tune_delay * usrp_rate / self.fft_size)))  # in fft_frames
print "tune delay in frames " , tune_delay
dwell_delay = max(1, int(round(options.dwell_delay * usrp_rate / self.fft_size))) # in fft_frames

self.msgq = gr.msg_queue(1)
print"message queue", self.msgq
self._tune_callback = tune(self)       # hang on to this to keep it from being GC'd
stats = gr.bin_statistics_f(self.fft_size, self.msgq,
                            self._tune_callback, tune_delay,
                            dwell_delay)

# FIXME leave out the log10 until we speed it up
  #self.connect(self.u, s2v, ffter, c2mag, log, stats)
```

```
        self.connect(self.u, s2v, ffter, c2mag, stats)

        if options.gain is None:
            # if no gain was specified, use the mid-point in dB
            g = self.u.get_gain_range()
            options.gain = float(g.start()+g.stop())/2.0

        self.set_gain(options.gain)
        print "gain =", options.gain

    def set_next_freq(self):
        target_freq = self.next_freq
        self.next_freq = self.next_freq + self.freq_step
        if self.next_freq >= self.max_center_freq:
            self.next_freq = self.min_center_freq

        if not self.set_freq(target_freq):
            print "Failed to set frequency to", target_freq
            sys.exit(1)

        return target_freq


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

        Args:
            target_freq: frequency in Hz
        @rypte: bool
        """

        r = self.u.set_center_freq(uhd.tune_request(target_freq, rf_freq=(target_freq +
self.lo_offset),rf_freq_policy=uhd.tune_request.POLICY_MANUAL))
        if r:
            return True

        return False

    def set_gain(self, gain):
        self.u.set_gain(gain)

    def nearest_freq(self, freq, channel_bandwidth):
        freq = round(freq / channel_bandwidth, 0) * channel_bandwidth
        return freq

def main_loop(tb):

    isolationlevels= 'IMMEDIATE'
    logging.debug("entered Main looop")
```

```
db_filename ='spec.db'
schema_filename = 'spectable.sql'
schema_flname ='chseltable.sql'
db_chsel_filename ='chsel.db'
def bin_freq(i_bin, center_freq):
   #hz_per_bin = tb.usrp_rate / tb.fft_size
   freq = center_freq - (tb.usrp_rate / 2) + (tb.channel_bandwidth * i_bin)
   #print "freq original:",freq
   #freq = nearest_freq(freq, tb.channel_bandwidth)
   #print "freq rounded:",freq
   return freq

bin_start = int(tb.fft_size * ((1 - 0.75) / 2))
bin_stop = int(tb.fft_size - bin_start)
#noise_floor_dbm = -174+10*math.log10(tb.usrp_rate)+5
#noise_floor_dbm = -174+10*math.log10(tb.channel_bandwidth)+5
#print "noise floor is %d" %noise_floor_dbm
db_is_new = not os.path.exists(db_filename)

with sqlite3.connect(db_filename,isolation_level = isolationlevels) as conn:

   if db_is_new:
      print 'creating schema for sensing'

      with open(schema_filename,'rt') as f:
         schema = f.read()

      conn.executescript(schema)

      '''with sqlite3.connect(db_chsel_filename,isolation_level = isolationlevels) as conn1:


         print "creating schema for chsel"

         with open(schema_flname,'rt') as g:
            schema_csl = g.read()

         conn1.executescript(schema_csl)'''




   conn.row_factory = sqlite3.Row
   conn.execute("PRAGMA synchronous=OFF")
   conn.execute("PRAGMA journal_mode=WAL")
   cursor= conn.cursor()
   cursor.execute("select count(stfreq) from spec")
```

```
for row in cursor.fetchmany(1):
   if row[0]>0:
      print"updating"

   else:
      print"inserting"



lst =list()
lst_avail =list()
lst_busy=list()

nsteps=tb.nsteps

while (nsteps>0):
   # Get the next message sent from the C++ code (blocking call).
   # It contains the center frequency and the mag squared of the fft
   m = parse_msg(tb.msgq.delete_head())

   # m.center_freq is the center frequency at the time of capture
   # m.data are the mag_squared of the fft output
   # m.raw_data is a string that contains the binary floats.
   # You could write this as binary to a file.

   querychsel ="""insert into chsel(centfreq,sel) values (?,?)"""
   querybusy="""insert into spec(stfreq,enfreq,ctfreq,pwdbm,status) values(?,?,?,?,'Busy')"""
   queryavail="""insert into spec(stfreq,enfreq,ctfreq,pwdbm,status) values(?,?,?,?,'Available')"""
   querybusyupdate ="""update spec set status='Busy',pwdbm=? where stfreq=?"""
   queryavailupdate ="""update spec set status='Available',pwdbm=? where stfreq=?"""
   print "center frequency" ,m.center_freq
   for i_bin in range(bin_start, bin_stop):

      center_freq = m.center_freq
      freq = bin_freq(i_bin, center_freq)
      noise_floor_dbm = 10*math.log10(min(m.data)/tb.usrp_rate)

      power_dbm = 10*math.log10(m.data[i_bin]/tb.usrp_rate)
      stfreq=freq
      enfreq=freq+tb.channel_bandwidth
      seltxt='No'
      ctfreq= (stfreq+enfreq)/2
      #power_dbm = 10*math.log10(m.data[i_bin]/(tb.channel_bandwidth*tb.fft_size))
      #power_dbm = 10*math.log10(m.data[i_bin]/(0.001*tb.channel_bandwidth*tb.fft_size))
      #print "Noise_floor = ", noise_floor_dbm
      '''print "m.data[i_bin] = ", m.data[i_bin]
      print "usrp_rate - ",tb.usrp_rate'''
      #power_dbm = -10*math.log10(m.data[i_bin]/0.001*tb.usrp_rate)
      #power_dbm_W_USRP = 10*math.log10((m.data[i_bin]*tb.channel_bandwidth)/(0.001*tb.usrp_rate))
```

```
            #print "power_dbm = ", power_dbm
            pwdbm = power_dbm
            a = power_dbm - noise_floor_dbm
           # print "Power - Noise = ", a

            if ((power_dbm - noise_floor_dbm) > tb.squelch_threshold):

               if (row[0]>0):

                  conn.execute(querybusyupdate,(pwdbm,stfreq,))

               else:

                  conn.execute(querybusy,(stfreq,enfreq,ctfreq,pwdbm))

            else:

               if (row[0]>0):

                  conn.execute(queryavailupdate,(pwdbm,stfreq,))

               else:

                  conn.execute(queryavail,(stfreq,enfreq,ctfreq,pwdbm))



         nsteps -=1




class Main_Data(object):

   def Inf_run(self):
      tb = my_top_block()
      tb.start()
        t=time.time()
      while True:

         main_loop(tb)

if __name__ == '__main__':
   print " The spectrum Detection  will be running for 2ms as per IEEE 802.22 standard and then data (MAC Frame)  will be sent for 10ms  in the band found as per the detection"

   try:
      Main_Data().Inf_run()
```

```
except KeyboardInterrupt:
    command ="rm spec.db"
    #command1 ="rm chsel.db"
    logging.debug("after interrupt")
    subprocess.call(command,shell=True)
    #subprocess.call(command1,shell=True)
```

10.4 TOP_BLOCK

```
from gnuradio import gr,uhd
from foimimo import *

import math
import time

from foimimo.foi_mimo_rx_path import mimo_receive_path as mimo_rx_path
from foimimo.foi_mimo_tx_path import mimo_transmit_path as mimo_tx_path
from foimimo.foi_siso_rx_path import siso_receive_path as siso_rx_path
from foimimo.foi_siso_tx_path import siso_transmit_path as siso_tx_path

class top_block_rx_gui():

    def __init__(self, rx_callback, bad_header_callback, options, all_gui_sinks):


        ######################################################
        # Variables
        ######################################################

        symbols_per_packet = math.ceil(((4+options.size+4) * 8) / options.occupied_tones)
        samples_per_packet = (symbols_per_packet+2) * (options.fft_length+options.cp_length)
        print "Symbols per Packet: ", symbols_per_packet
        print "Samples per Packet: ", samples_per_packet
        if options.discontinuous:
            stream_size = [100000, int(options.discontinuous*samples_per_packet)]
        else:
            stream_size = [0, 100000]


        ######################################################
        # Blocks
        ######################################################
        if options.siso:
            device_addr = "addr="+options.usrp_addr0
            print "Using USRP unit with device adress: ",device_addr
            self.uhd_usrp_source = uhd.single_usrp_source(
            device_addr=device_addr,
            io_type=uhd.io_type.COMPLEX_FLOAT32,
            num_channels=1,
            )
```

```
    self.uhd_usrp_source.set_clock_config(uhd.clock_config.external());
    self.uhd_usrp_source.set_time_next_pps(uhd.time_spec())
    self.uhd_usrp_source.set_samp_rate(options.sample_rate)
    self.uhd_usrp_source.set_center_freq(options.center_freq, 0)
    self.uhd_usrp_source.set_gain(options.gain_rx, 0)

    self.foi_rxpath = siso_rx_path(rx_callback, bad_header_callback, options)

else:
    device_addr = "addr0="+options.usrp_addr0+", addr1="+options.usrp_addr1
    print "Using USRP units with device adress: ",device_addr
    self.uhd_usrp_source = uhd.multi_usrp_source(
       device_addr=device_addr,
       io_type=uhd.io_type.COMPLEX_FLOAT32,
       num_channels=2,
       )
    self.antenna0 = self.uhd_usrp_source.get_antenna(0)
    self.antenna1 = self.uhd_usrp_source.get_antenna(1)
    print "Using antenna " , self.antenna0, " and ", self.antenna1
    # Set usrp source parameters
    #self.uhd_usrp_source.set_clock_config(uhd.clock_config.external(), uhd.ALL_MBOARDS);
    #self.uhd_usrp_source.set_time_now(uhd.time_spec(time.time()), 0)
      #self.uhd_usrp_source.set_time_now(uhd.time_spec(time.time()), 1)


      self.uhd_usrp_source.set_clock_config(uhd.clock_config.internal(), 0)
      _config = uhd.clock_config()
      _config.ref_source = uhd.clock_config.REF_MIMO
      _config.pps_source = uhd.clock_config.PPS_MIMO
      _config.pps_polarity = uhd.clock_config.PPS_POS
      self.uhd_usrp_source.set_clock_config(_config, 1)
      #self.uhd_usrp_source.set_time_source("mimo", 1)
      #self.uhd_usrp_source.set_time_now(uhd.time_spec_t(time.time()), uhd.ALL_MBOARDS)
      self.uhd_usrp_source.set_time_unknown_pps(uhd.time_spec(0.0))
    self.uhd_usrp_source.set_samp_rate(options.sample_rate)
    self.uhd_usrp_source.set_center_freq(options.center_freq, 0)
    self.uhd_usrp_source.set_center_freq(options.center_freq, 1)
    self.uhd_usrp_source.set_gain(options.gain_rx, 0)
    self.uhd_usrp_source.set_gain(options.gain_rx, 1)

    self.foi_rxpath = mimo_rx_path(rx_callback, bad_header_callback, options)

# GUI sinks:
# Get sinks in list of tuples all_gui_sinks
# Tuples consist of (sink,name)
'''i = [x[1] for x in all_gui_sinks].index("fft_channel_filter_0")
self.wxgui_fftsink_chanfilt0 = all_gui_sinks[i][0]
i = [x[1] for x in all_gui_sinks].index("fft_channel_filter_1")
self.wxgui_fftsink_chanfilt1 = all_gui_sinks[i][0]
i = [x[1] for x in all_gui_sinks].index("constellation_fft_0")
```

```
        self.wxgui_constellationsink_fft0 = all_gui_sinks[i][0]
        i = [x[1] for x in all_gui_sinks].index("constellation_fft_1")
        self.wxgui_constellationsink_fft1 = all_gui_sinks[i][0]
         = [x[1] for x in all_gui_sinks].index("constellation_frame_acq")
        self.wxgui_constellationsink_frameacq = all_gui_sinks[i][0]
        i = [x[1] for x in all_gui_sinks].index("constellation_frame_sink")
        self.wxgui_constellationsink_framesink = all_gui_sinks[i][0]'''

        ##################################################
        # Connections
        ##################################################

class top_block_tx_gui():

    def __init__(self, options):


        ##################################################
        # Variables
        ##################################################

        code = {"":[1,1],
              "3/4":[3.0, 4.0],
              "2/3":[2.0, 3.0],
              "1/3":[1.0, 3.0]}
        (k,n) = code[options.code_rate]

        symbols_per_packet = int(math.ceil(((((options.size-4)*(n/k))+4) * 8.0) / (2.0*options.occupied_tones)))
        samples_per_packet = (symbols_per_packet+2) * (options.fft_length+options.cp_length)
        print "\nSymbols per Packet: ", symbols_per_packet
        print "Samples per Packet: " + str(samples_per_packet) + "\n"

        if options.discontinuous:
            stream_size = [100000, int(options.discontinuous*samples_per_packet)]
        else:
            stream_size = [0, 100000]


        ##################################################
        # Blocks
        ##################################################
        if options.siso:
            print "SISO"
            device_addr = "addr0="+options.usrp_addr0
            print "Using USRP unit with device adress: ",device_addr
            self.uhd_usrp_sink = uhd.single_usrp_sink(
                device_addr=device_addr,
                io_type=uhd.io_type.COMPLEX_FLOAT32,
                num_channels=1,
                )
```

```
    # Set usrp sink parameters
    self.uhd_usrp_sink.set_clock_config(uhd.clock_config.external(), uhd.ALL_MBOARDS);
    self.uhd_usrp_sink.set_time_next_pps(uhd.time_spec(0.0))
    self.uhd_usrp_sink.set_samp_rate(options.sample_rate)
    self.uhd_usrp_sink.set_center_freq(options.center_freq, 0)
    self.uhd_usrp_sink.set_gain(options.gain_tx, 0)
    self.uhd_usrp_sink.set_center_freq(options.center_freq, 0)
    self.uhd_usrp_sink.set_gain(options.gain_tx, 0)
    antenna_0 = self.uhd_usrp_sink.get_antenna(0)
    print "Transmitting on antenna0:",antenna_0

    self.foi_txpath = siso_tx_path(options)

else:
    print "MIMO"
    device_addr = "addr0="+options.usrp_addr0+", addr1="+options.usrp_addr1
    print "Using USRP units with device adress: ",device_addr
    self.uhd_usrp_sink = uhd.multi_usrp_sink(
       device_addr=device_addr,
       io_type=uhd.io_type.COMPLEX_FLOAT32,
       num_channels=2,
       )
    # Set usrp sink parameters
    #self.uhd_usrp_sink.set_clock_config(uhd.clock_config.external(), uhd.ALL_MBOARDS);


      #self.uhd_usrp_sink.set_clock_config(uhd.clock_config.external(), 1)
      self.uhd_usrp_sink.set_clock_config(uhd.clock_config.internal(), 0)
      _config = uhd.clock_config()
      _config.ref_source = uhd.clock_config.REF_MIMO
      _config.pps_source = uhd.clock_config.PPS_MIMO
      self.uhd_usrp_sink.set_clock_config(_config, 1)
      #time = uhd.time_spec.get_system_time()
      #print "system time",time
      #_time.full= self.uhd.time_spec.get_full_secs()
      #_time.frac= self.uhd.time_spec.get_frac_secs()
    self.uhd_usrp_sink.set_time_unknown_pps(uhd.time_spec(0.0))
    self.uhd_usrp_sink.set_samp_rate(options.sample_rate)
    self.uhd_usrp_sink.set_center_freq(options.center_freq, 0)
    self.uhd_usrp_sink.set_center_freq(options.center_freq, 1)
    self.uhd_usrp_sink.set_gain(options.gain_tx, 0)
    self.uhd_usrp_sink.set_gain(options.gain_tx, 1)
    antenna_0 = self.uhd_usrp_sink.get_antenna(0)
    antenna_1 = self.uhd_usrp_sink.get_antenna(1)
    print "Transmitting on antenna0:",antenna_0,"and antenna1:",antenna_1

    self.foi_txpath = mimo_tx_path(options)
```

```
    ###--- Transmitter - Send packet ---###
    def send_pkt(self,payload='', eof=False):
        return self.foi_txpath.send_pkt(payload, eof)

class top_block_tx_rx(gr.top_block):

    def __init__(self,rx_callback,bad_header_callback,options,all_sinks):
        gr.top_block.__init__(self)
        self.tx_block = top_block_tx_gui(options)
        self.rx_block = top_block_rx_gui(rx_callback,bad_header_callback,options,all_sinks)

        ##########################
        # connections
        ##########################
        if options.siso:
            self.connect((self.tx_block.foi_txpath, 0), (self.tx_block.uhd_usrp_sink, 0))
            self.connect((self.rx_block.uhd_usrp_source, 0), (self.rx_block.foi_rxpath, 0))
        else:
            self.connect((self.tx_block.foi_txpath, 0), (self.tx_block.uhd_usrp_sink, 0))
            self.connect((self.tx_block.foi_txpath, 1), (self.tx_block.uhd_usrp_sink, 1))
            self.connect((self.rx_block.uhd_usrp_source, 0), (self.rx_block.foi_rxpath, 0))
            self.connect((self.rx_block.uhd_usrp_source, 1), (self.rx_block.foi_rxpath, 1))
```

10.5 SEND_THREAD

```
 import struct,sys
import gnuradio.gr.gr_threading as _threading
import sqlite3
import random
import time,random
import threading
from sbs_ import *
from coordination_module import *
import os

global previous,td,e,count,chgfrq,frq_chk
td=0.2;count=0;e=0;chgfrq=0;frq_chk=0
previous=time.time()
e =threading.Event()

class Send_thread(_threading.Thread):
    def __init__(self, global_ctrl, topblock, send_data,rx_callback='',bad_header_callback='',all_sinks=0,tun_fd=None):
        _threading.Thread.__init__(self)
        self.global_ctrl = global_ctrl
        self.topblock = topblock
        self.send_pkt = topblock.tx_block.send_pkt
        self.setDaemon(1)
        self.send_data = send_data
        self.db_flname ='spec.db'
        self.rx_callback= rx_callback
```

```
        self.bad_header_callback=bad_header_callback
        self.all_sinks = all_sinks
        self.tun_fd = 3

    def run(self):
      ###--- Transmitter - send data ---###
      options = self.global_ctrl.get_options()
      if options.benchmark_mode:
          # generate and send packets
          npixels = options.npixels
          n = 0
          s_pktno = 0
          color = 0
          pkt_size = int(options.size)-8
          while n < npixels:
              if npixels-n < pkt_size:
                  pkt_size = int(npixels-n)
              #pkt_contents = struct.pack('!H', s_pktno) + (pkt_size - 2) * (chr(0 & 0xff) + chr(color & 0xff) + chr(0 & 0xff))
#green colors...
              pkt_contents = pkt_size/3 * (chr(0 & 0xff) + chr(color & 0xff) + chr(0 & 0xff)) #green colors...
              self.send_pkt(pkt_contents)
              sys.stdout.write(".")
              n += pkt_size
              s_pktno += 1
              color += 1
              if color==(256): color=0

          self.send_pkt(eof=True)
          print "\nThe transmitter program sent: ", s_pktno-1, "OFDM packets"
      elif options.file_mode:
          #send packets
          global count
          pkt_size = int(options.size)-8-2 #1 for pkt_index
          n_sent = 1
          # Read input data from file:
          in_file = open(options.input_filename,mode="rb")
          indata = in_file.read(pkt_size)
          read_pkt_size = len(indata)
          if True:
              print "TB: Reading first pkt of size",read_pkt_size
          while (indata) and n_sent<99998:
              while time.time()-previous<(td+2.8):
                  print "***********packet number=%d***************"%(n_sent)
                  cin = struct.pack('!H',n_sent)
                  payload =cin+indata
                  self.send_pkt(payload)
                  print "TB: Sent pkt! %d"%(n_sent)
                  time.sleep(0.2)
                  if (n_sent)!= options.pkt_num_rcvd:
                      self.send_pkt(payload)
```

```
                print "pkt_not rcvd, so sending them anyway"
            else:
                n_sent+= 1
                indata = in_file.read(pkt_size)
                read_pkt_size = len(indata)
                if True:
                    print "TB: Reading another pkt of size",read_pkt_size
                if read_pkt_size ==0:
                    break

        else:
            global td,previous
            k=time.time()
            print "Activating co-oridination module"
            self.remote_coordination(options)
            td= time.time()-k
            #print "time for co-ordination",td
            td1 = time.time() -previous
            print "diff. between current and last co-oridnation",td1

    else:

        if True:
            print "Sent the last empty packet and eof"
    in_file.close()
    print "Sent",n_sent,"packets."

elif options.image_mode:
    pkt_size = int(options.size)-8-2       #Remember to send a multiple of 3 to get one pixel
    n_sent = 1
    indata = self.send_data
    tot_size = len(indata)
    nr_pkts = tot_size/pkt_size # rounds by automatic int/int
    if(tot_size%pkt_size): nr_pkts+=1
    print "Total number of pckts",nr_pkts
    for i in range(0,nr_pkts):

        while time.time()-previous<(td+2.8):
            if(n_sent == nr_pkts):
                print "################################packet number=%d
sent#######################################################"%(n_sent)
                cin = struct.pack('!H',n_sent)
                pkt_contents =cin+indata[i*pkt_size:]
                self.send_pkt(pkt_contents)
                print "Yeah Last TB: Sent pkt!"
                time.sleep(0.2)
                print
"*************************************************completed*********************************
************************"
                while (n_sent)!= options.pkt_num_rcvd:
```

```
                self.send_pkt(pkt_contents)
                print "pkt %d not rcvd, so sending them anyway"%(n_sent)
                time.sleep(0.2)
              break

            else:
              print "##################################packet number
senttttttttttttttttttttt####################################################################################
################# %d"%(n_sent)
              cin = struct.pack('!H',n_sent)
              pkt_contents = cin+indata[i*pkt_size:(i+1)*pkt_size]
              self.send_pkt(pkt_contents)
              print "TB: Sent pkt!"
              time.sleep(0.2)
              while (n_sent)!= options.pkt_num_rcvd:
                self.send_pkt(pkt_contents)
                print "pkt %d not rcvd, so sending them anyway"%(n_sent)
                time.sleep(0.2)
              n_sent+= 1


          else:
            global td,previous
            k=time.time()
            print "Activating co-oridination module"
            self.remote_coordination(options)
            td= time.time()-k
            #print "time for co-ordination",td
            td1 = time.time() -previous
            print "diff. between current and last co-oridnation",td1

        #self.send_pkt("")       # Sent one empty pkt to get last pkt... bug?
        #self.send_pkt(eof=True)
        if True:
          print "Sent the last empty packet and eof"
        print "Sent",n_sent,"packets."


      elif options.streaming_mode:
        i=0
        while True:
          while time.time()-previous<(td+2.8):
            i+=1
            pkt_size = int(options.size)-8-5
            contents = os.read(self.tun_fd,388)
            cin =struct.pack('!H',i)
            pkt_contents= chr(115)+chr(116)+chr(114)+cin+contents+(393-len(contents)-5)*chr(0 & 0xff)
            self.send_pkt(pkt_contents)
            print "#######################################################################pakt num %d
sent##########################################################################"%(i)
```

```
            options.e1.wait(0.3)
            print "options pakt num rcvd at send thread",options.pkt_num_rcvd
            while i!=options.pkt_num_rcvd:
               self.send_pkt(pkt_contents)
               print "Pkt %d not rcvd, trying to muzzle through"%(i)
               options.e1.wait(0.3)

         else:
            pass
            k=time.time()
            print "Activating co-oridination module"
            self.remote_coordination(options)
            td= time.time()-k
            #print "time for co-ordination",td
            td1 = time.time() -previous
            print "diff. between current and last co-oridnation",td1



   elif options.ber_mode:
     # generate and send packets
     nbytes = options.npixels
     n = 0
     s_pktno = 0
     pkt_size = int(options.size)-8
     while n < nbytes:
        pkt_contents = struct.pack('!H', s_pktno) + (pkt_size - 2) * chr(0 & 0xff)

        self.send_pkt(pkt_contents)
        #sys.stdout.write(".")
        n += pkt_size
        s_pktno += 1

     self.send_pkt(eof=True)
     print "\nThe transmitter program sent: ", s_pktno-1, "OFDM packets"

  def remote_coordination(self,options):
     global previous,count,e,chgfrq
     print "mimo options",options.mimo
     print "chgfrq",chgfrq
     #print "co-ordinating after=",time.time()-previous
     previous = time.time()
     #print "**********************starting co-ordination*****************************"
     current_freq = self.topblock.tx_block.uhd_usrp_sink.get_center_freq()
     print "current freq=",current_freq
     nwfrq = newfreq_module(options,self.topblock.tx_block.uhd_usrp_sink.get_center_freq())
     nwfrq.newfreq-=(20000*chgfrq)
     chgfrq+=1
     frqchr = self.parsing_freq(nwfrq.newfreq)
     pkt_size =int(options.size)-8
```

```
        pkt_contents= chr(99)+chr(104)+chr(103)+frqchr+chr(101)+chr(110)+chr(100)+(pkt_size-11)* chr(0&0xff)
        #print "length of pkt_contents freq chng",len(pkt_contents)
        options.pkt_content = pkt_contents
        self.send_pkt(pkt_contents)
        while options.e.is_set():
          e.clear()
        print " pckt ack status event ",options.e.is_set()
        print "cog pkt rcvd status",options.cog_pkt_rcvd
        print "Freq. info pcket sent ###############################################"
        options.e.wait(0.3)
        while not options.cog_pkt_rcvd ==1:
          print "Frq. info pckt not rcvd, trying again"
          print "switching to %d Hz frequency for sending ACK
packets****************************"%(nwfrq.newfreq)
          if options.siso:
            self.topblock.tx_block.uhd_usrp_sink.set_center_freq(nwfrq.newfreq,0)
          else:
            self.topblock.tx_block.uhd_usrp_sink.set_center_freq(nwfrq.newfreq,0)

            self.topblock.tx_block.uhd_usrp_sink.set_center_freq(nwfrq.newfreq,1)

          self.send_pkt(pkt_contents)
          print "freq info. pkt sent"
          options.e.wait(0.3)

        else:
          e.set()
          e.clear()
          print "xchngggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
doneeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee"
          print "switching to %d Hz frequency"%(nwfrq.newfreq)
          if options.siso:
            self.topblock.tx_block.uhd_usrp_sink.set_center_freq(nwfrq.newfreq,0)
          else:
            self.topblock.tx_block.uhd_usrp_sink.set_center_freq(nwfrq.newfreq,0)

            self.topblock.tx_block.uhd_usrp_sink.set_center_freq(nwfrq.newfreq,1)

          options.cog_pkt_rcvd=0


    def parsing_freq(self,freq):
        sfreq=str(freq)
        f1 =chr(int(sfreq[0]))
        f2 = chr(int(sfreq[1]))
        f3 = chr(int(sfreq[2]))
        f4 = chr(int(sfreq[3]))
        f5 = chr(int(sfreq[4]))
        frqchr = f1+f2+f3+f4+f5
        return frqchr
```

10.6 GUI_MAIN_FRAME_TX

```
import gnuradio.gr.gr_threading as _threading

from grc_gnuradio import wxgui as grc_wxgui
import wx

# import local gui classes
from foimimo_tx_panel import txPanel
from foimimo_ctrl_panel import ctrlPanel

class GUI_thread(_threading.Thread):
   def __init__(self, app):
      _threading.Thread.__init__(self)
      self.setDaemon(1)
      self.app = app
      self.start()

   def run(self):
      self.app.MainLoop()

class main_frame(wx.Frame):
   def __init__(self, rx_callback, bad_header_callback, reset_main_variables, global_ctrl, app,tun_fd):
      wx.Frame.__init__(self, None, -1, "OFDM_CR_TRANS")

      # Global variables init
      self.global_ctrl = global_ctrl

      # Variables needed when starting top block
      self.options = self.global_ctrl.get_options()
      self.rx_callback = rx_callback
      self.bad_header_callback = bad_header_callback
      self.reset_main_variables = reset_main_variables
      self.tun_fd = tun_fd

      # Other variables to init
      self.data_to_send = []

      # Menus and bars
      nstatus = 2
      #self.CreateStatusBar (nstatus)
      self.create_menus()
      self.Bind (wx.EVT_CLOSE, self.on_close)

      # Window layout and size
      self.vbox = wx.BoxSizer(wx.VERTICAL)

      # Panels and notebooks
      self.main_panel = grc_wxgui.Panel(self,wx.VERTICAL)
      self.gui_notebook_main = wx.Notebook(self.main_panel.GetWin(), style=wx.NB_LEFT,size=(1180,630))
      self.main_panel.Add(self.gui_notebook_main)
```

```
        self.vbox.Add(self.main_panel,0)

        self.tx_panel = txPanel(self.gui_notebook_main,self.global_ctrl,self,wx.HORIZONTAL)
        self.ctrl_panel = ctrlPanel(self.gui_notebook_main,self.global_ctrl,self,wx.HORIZONTAL)
        self.gui_notebook_main.AddPage(self.ctrl_panel, "Settings")
          self.gui_notebook_main.AddPage(self.tx_panel, "Tx Window")

        #self.StatusBar.SetStatusText('Status: off')

        self.SetSizerAndFit(self.vbox)
        self.SetAutoLayout(True)
        self.Show(True)

    def start_tb(self):

self.global_ctrl.start_flowgraph(self.rx_callback,self.bad_header_callback,0,send_data=self.data_to_send,tun_fd=self.t
un_fd)
        '''if self.global_ctrl.get_options_mimo():
            self.StatusBar.SetStatusText('Status: running in MIMO mode')
        else:
            self.StatusBar.SetStatusText('Status: running in SISO mode')'''

    def stop_tb(self):
        self.global_ctrl.stop_flowgraph()
        #self.StatusBar.SetStatusText('Status: off')

    def set_data_to_send(self, data):
        self.data_to_send = data

    def create_menus(self):
        filemenu = wx.Menu ()
        main_menubar = wx.MenuBar()
        main_menubar.Append(filemenu, "&File")
        #self.SetMenuBar(main_menubar)

        menu_exit = filemenu.Append(wx.ID_EXIT, 'E&xit', 'Exit')
        self.Bind(wx.EVT_MENU, self.on_exit, menu_exit)

    def on_exit(self, event):
        self.global_ctrl.stop_flowgraph()
        self.global_ctrl.exit()
        ok1 = self.DestroyChildren()
        ok2 = self.Destroy()

    def on_close(self,event):
        self.on_exit(wx.EVT_MENU)
```

10.7 GLOBAL_CONTROL_UNIT

```
 from foimimo_tb_options import *
from foimimo_top_blocks import *
from gui.foimimo_sendThread import Send_thread

class global_ctrl_unit:
   def __init__(self,mode):
      # Init values of global variables:
      self.mode = mode #rx_gui, tx_gui,
      self.running = False
      self.default_options()
      self.focus_view = ""
      self.GUIAlive = True
      self.reset_statistics()

   def reset_statistics(self):
      self.npkt_rcvd = 0
      self.npkt_right = 0
      self.nbit_rcvd = 0
      self.nbit_right = 0
      self.nr_bad_headers = 0

   def get_pkt_statistics(self):
      return (self.npkt_rcvd,self.npkt_right,self.nr_bad_headers)

   def get_bit_statistics(self):
      return (self.nbit_rcvd,self.nbit_right)

   def update_statistics(self,npkt_rcvd,npkt_right,nbit_rcvd,nbit_right):
      self.npkt_rcvd += npkt_rcvd
      self.npkt_right += npkt_right
      self.nbit_rcvd += nbit_rcvd
      self.nbit_right += nbit_right

   def update_bad_header(self, new_bad_headers):
      self.nr_bad_headers += new_bad_headers

   def start_flowgraph(self,rx_callback='',bad_header_callback='',all_sinks=0,send_data=[],tun_fd=None):
      self.reset_statistics()
      self.rx_callback = rx_callback
      self.bad_header_callback = bad_header_callback
      self.all_sinks = all_sinks
      self.tun_fd =tun_fd
      print "gloibal ctrl tun_fd",self.tun_fd
      # Create a top block and a sender thread with given options:
      if self.mode == "rx_gui":
         self.top_block = 0
         self.top_block = top_block_rx_gui(self.rx_callback,self.bad_header_callback,self.options,self.all_sinks)
      elif self.mode == "tx_gui":
```

```
            self.top_block = 0
            self.top_block = top_block_tx_rx(self.rx_callback,self.bad_header_callback,self.options,self.all_sinks)
            self.sender = 0
            print "Length of send data", len(send_data)
            self.sender = Send_thread(self, self.top_block,
send_data,self.rx_callback,self.bad_header_callback,self.all_sinks,self.tun_fd)    # Start sender thread
            self.sender.start()
        else:
            print "unknown mode"

        # start the top block and the sender thread:
        self.top_block.start()
        self.running = True

    def stop_flowgraph(self):
        if self.running:
            self.top_block.stop()
            self.top_block.wait()
            self.top_block = 0
            self.running = False
        else:
            pass

    def get_running(self):
        # Return values for self.running
        return self.running
    def set_running(self,value):
        # Set values for self.running
        self.running = value

    def get_options_mimo(self):
        # Return values for self.options.siso
        return not self.options.siso
    def set_options_mimo(self,value):
        # Get values for self.options.siso
        if value: # bool or int value
            self.options.siso = 0
        else:
            self.options.siso = 1

    def get_options(self):
        # Return values for self.options
        return self.options
    def set_options(self,options_given):
        # Set values for self.options
        self.options = options_given

    def default_options(self):
        # Set and return default values for self.options
        if self.mode == "rx_gui":
```

```
            self.options = tb_options_rx_gui()
        elif self.mode == "tx_gui":
            self.options = tb_options_tx_gui()
        else:
            print "unknown mode"
        return self.options

    def set_focus_view(self,view):
        self.focus_view = view
    def get_focus_view(self):
        return self.focus_view
    def exit(self):
        self.GUIAlive = False
```

10.8 TB_OPTIONS

```
import gnuradio.gr.gr_threading as _threading
import threading
class tb_options_rx_gui:
    def __init__(self):
        #Normal:
        self.size = float(401)
        self.npixels = float(1000000.0)
        self.input_filename = str("/home/dsplab/Desktop/demo_image.jpg")
        self.output_filename_mimo = str("/home/dsplab/Desktop/recvd_image.jpg")
        self.output_filename_siso = str("/home/dsplab/Desktop/recvd_image.jpg")
        self.snr = float(30) #Not in GUI, used by ofdm receiver, estimated snr for cp synch
        self.discontinuous = int(0)
        self.tx_amplitude = float(0.5)
        self.modulation = str("qpsk")
        self.code_rate = str("3/4")    # will not be set in gui, only first choice of dropdown works, need to change order in
foimimo_ctrl_panel.py
        self.siso = 1
        self.mimo = 0
        self.verbose = 0
        self.write_all = 1
        self.per_avg_nr = int(20)
        self.image_mode = 1     # only one of the three modes can be set to 1
        self.file_mode = 0
        self.benchmark_mode = 0
        #Expert:
        self.fft_length = int(512)
        self.occupied_tones = int(200)
        self.cp_length = int(128)
        self.log = 0
        #USRP:
        self.usrp_rx = True
        self.usrp_tx = True
        self.usrp_addr0 = str("192.168.30.2")
        self.usrp_addr1 = str("192.168.20.2")
```

```
        self.center_freq = float(860000000.0)
        self.sample_rate = float(3125000.0)
        self.gain_rx = float(15.0)
        self.gain_tx = float(17.0)

    class tb_options_tx_gui:
       def __init__(self):
        #Normal:
        self.size = float(401)
        self.npixels = float(1000000.0)
        self.input_filename = str("/home/dsplab/Desktop/defense.jpg")
        self.output_filename_mimo = str("/home/dsplab/Desktop/defense.jpg")
        self.output_filename_siso = str("/home/dsplab/Desktop/defense.jpg")
        self.snr = float(30) #Not in GUI, used by ofdm receiver, estimated snr for cp synch
        self.discontinuous = int(0)
        self.tx_amplitude = float(0.5)
        self.modulation = str("qpsk")
        self.code_rate = str("3/4")    # will not be set in gui, only first choice of dropdown works, need to change order in
    foimimo_ctrl_panel.py
        self.siso = 0
        self.mimo = 0
        self.verbose = 0
        self.write_all = 0
        self.per_avg_nr = int(20)
        self.image_mode = 1     # only one of the three modes can be set to 1
        self.file_mode = 0
        self.benchmark_mode = 0
        self.streaming_mode= 0
        #Expert:
        self.fft_length = int(512)
        self.occupied_tones = int(200)
        self.cp_length = int(128)
        self.log = 0
        #USRP:
        self.usrp_rx = True
        self.usrp_tx = True
        self.usrp_addr0 = str("192.168.30.2")
        self.usrp_addr1 = str("192.168.20.2")
        self.center_freq = float(875000000.0)
        self.sample_rate = float(3125000.0)
        self.gain_rx = float(15.0)
        self.gain_tx = float(17.0)
        #Feedbak dept.
        self.rcv=0
        self.pkt_num_rcvd=0
        self.cog_pkt_rcvd=0
        self.e = threading.Event()
        self.e1 = threading.Event()
```

10.9 CTRL_PANEL

```
from grc_gnuradio import wxgui as grc_wxgui
import wx

class ctrlPanel(grc_wxgui.Panel):
   def __init__(self, parent, global_ctrl, main_frame, orient=wx.HORIZONTAL):
      grc_wxgui.Panel.__init__ (self,parent,orient)

      # Init variables
      self.main_frame = main_frame
      self.global_ctrl = global_ctrl
      self.options_enabled = True
      self.options = self.global_ctrl.default_options()

      # Sizes and positions:
      box_width = 100
      box_size = (box_width,-1)

      x_boxline1 = 10
      x_textline1 = x_boxline1 + box_width + 10
      x_checkline1 = x_boxline1

      x_boxline2 = x_textline1 + 500
      x_textline2 = x_boxline2 + box_width + 10
      x_checkline2 = x_boxline2

      y_row = 30
      y_offs = 10
      y_text_offs = 6

      x_button = x_boxline1
      y_button = 500

      button_width = 200
      button_size = (button_width,-1)
      y_buttonrow = 45

      y_checkbox_offs = 8

      # All options:
      # Normal:
      row_nr=0
      #self.text_options = wx.StaticText(parent=self, label="Normal settings:", pos=(x_boxline1,y_offs+y_row*row_nr))
      row_nr+=1

      self.radio_options_image_mode = wx.RadioButton(parent=self, label="Image",
pos=(x_textline1+40,y_offs+y_row*row_nr), style=wx.RB_GROUP)
      self.radio_options_file_mode = wx.RadioButton(parent=self, label="File",
pos=(x_textline1+120,y_offs+y_row*row_nr))
      self.radio_options_streaming_mode = wx.RadioButton(parent=self, label="Streaming",
```

```
pos=(x_textline1+180,y_offs+y_row*row_nr))
        self.text_transmission_mode = wx.StaticText(parent=self, label="Application",
pos=(x_boxline1,y_offs+y_row*row_nr+3))
        row_nr+=1
        self.radio_options_mimo = wx.RadioButton(parent=self, label="MIMO_OFDM",
pos=(x_textline1+40,y_offs+y_row*row_nr+y_checkbox_offs), style=wx.RB_GROUP)
        self.radio_options_siso = wx.RadioButton(parent=self, label="SISO_OFDM",
pos=(x_textline1+180,y_offs+y_row*row_nr+y_checkbox_offs))
        self.text_transmission_mode = wx.StaticText(parent=self, label="Mode",
pos=(x_boxline1,y_offs+y_row*row_nr+3))

        #self.radio_options_benchmark_mode = wx.RadioButton(parent=self, label="Benchmark",
pos=(x_textline1+190,y_offs+y_row*row_nr))
        #row_nr+=1
        #self.text_npixels = wx.StaticText(parent=self, label="- Number of pixels to transmit",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
        #self.textbox_options_npixels =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
        #row_nr+=1
        #self.text_input_filename = wx.StaticText(parent=self, label="- File containing input data",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
        #self.textbox_options_input_filename =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
        #row_nr+=1
        #self.text_output_filename_mimo = wx.StaticText(parent=self, label="- File to be written with output data in
mimo mode", pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
        #self.textbox_options_output_filename_mimo =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
        #row_nr+=1
        #self.text_output_filename_siso = wx.StaticText(parent=self, label="- File to be written with output data in siso
mode", pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
        #self.textbox_options_output_filename_siso =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
        row_nr+=2
        self.text_size = wx.StaticText(parent=self, label="Packet Size",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
        self.textbox_options_size =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
        row_nr+=1
        self.text_sample_rate = wx.StaticText(parent=self, label="Sample Rate",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
        self.textbox_options_sample_rate =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
        row_nr+=1
        self.text_discontinuous = wx.StaticText(parent=self, label="Continuos / Discontinous Mode (0/1)",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
        self.textbox_options_discontinuous =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
        row_nr+=2
```

```
        self.text_options = wx.StaticText(parent=self, label="PHY Parameters", pos=(x_boxline1,y_offs+y_row*row_nr))
        row_nr+=1
        self.text_tx_amplitude = wx.StaticText(parent=self, label="Peak Sub-Carrier Power",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
    self.textbox_options_tx_amplitude =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
    row_nr+=1
    self.text_modulation = wx.StaticText(parent=self, label="Modulation",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
    self.textbox_options_modulation =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
    row_nr+=1

    #row_nr+=0.8
    #self.checkbox_options_write_all = wx.CheckBox(parent=self, label="Write all received packets (not only those
that are ok)", pos=(x_checkline1,y_offs+y_row*row_nr+y_checkbox_offs))
    #row_nr+=0.8
    #self.checkbox_options_verbose = wx.CheckBox(parent=self, label="Verbose",
pos=(x_checkline1,y_offs+y_row*row_nr+y_checkbox_offs))
    #row_nr+=1.2
    #self.text_per_avg_nr = wx.StaticText(parent=self, label="- Nr of received packets to make packet error rate
average over", pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
    #self.textbox_options_per_avg_nr =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
    #row_nr += 1
    self.text_code_rate = wx.StaticText(parent=self, label="Code rate",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
    self.coding_options = wx.Choice(self,-1,pos=(x_boxline1,y_offs+y_row*row_nr),choices=["3/4","2/3","1/3",""])
      row_nr+=1
    self.text_gain_rx = wx.StaticText(parent=self, label="Rx Gain",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
    self.textbox_options_gain_rx =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))
    row_nr+=1
    self.text_gain_tx = wx.StaticText(parent=self, label="Tx Gain",
pos=(x_textline1,y_offs+y_row*row_nr+y_text_offs))
    self.textbox_options_gain_tx =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline1,y_offs+y_row*row_nr))


    #Expert:
    row_nr=3
    self.text_expert = wx.StaticText(parent=self, label="OFDM Specific Parameter",
pos=(x_boxline2,y_offs+y_row*row_nr))
    row_nr+=1
    self.text_fft_length = wx.StaticText(parent=self, label="FFT Size",
pos=(x_textline2,y_offs+y_row*row_nr+y_text_offs))
    self.textbox_options_fft_length =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline2,y_offs+y_row*row_nr))
    row_nr+=1
```

```
        self.text_occupied_tones = wx.StaticText(parent=self, label="Occupied FFT bins",
pos=(x_textline2,y_offs+y_row*row_nr+y_text_offs))
        self.textbox_options_occupied_tones =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline2,y_offs+y_row*row_nr))
        row_nr+=1
        self.text_cp_length = wx.StaticText(parent=self, label="Total CP Bits",
pos=(x_textline2,y_offs+y_row*row_nr+y_text_offs))
        self.textbox_options_cp_length =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline2,y_offs+y_row*row_nr))
        #row_nr+=1
        #self.checkbox_options_log = wx.CheckBox(parent=self, label="Log all parts of flow graph to files (CAUTION: lots of
data)", pos=(x_checkline2,y_offs+y_row*row_nr+y_checkbox_offs))

        #USRP:
        '''row_nr+=2
        self.text_usrp = wx.StaticText(parent=self, label="USRP settings:", pos=(x_boxline2,y_offs+y_row*row_nr))
        row_nr+=1
        self.text_usrp_addr0 = wx.StaticText(parent=self, label="- USRP address, first unit",
pos=(x_textline2,y_offs+y_row*row_nr+y_text_offs))
        self.textbox_options_usrp_addr0 =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline2,y_offs+y_row*row_nr))
        row_nr+=1
        self.text_usrp_addr1 = wx.StaticText(parent=self, label="- USRP address, second unit",
pos=(x_textline2,y_offs+y_row*row_nr+y_text_offs))
        self.textbox_options_usrp_addr1 =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline2,y_offs+y_row*row_nr))
        row_nr+=1
        self.text_center_freq = wx.StaticText(parent=self, label="- USRP center frequency",
pos=(x_textline2,y_offs+y_row*row_nr+y_text_offs))
        self.textbox_options_center_freq =
wx.TextCtrl(self,size=box_size,style=wx.TE_PROCESS_ENTER,pos=(x_boxline2,y_offs+y_row*row_nr))'''


        #Buttons:
        self.start_button = wx.Button(parent=self,label="Start",pos=(x_button+500,y_button-150),size=button_size)
        self.stop_button = wx.Button(parent=self,label="Stop",pos=(x_button+500+button_width+10,y_button-
150),size=button_size)
        #self.settings_button = wx.Button(parent=self,label="Activate
Settings",pos=(x_button,y_button+35),size=button_size)
        #self.default_button = wx.Button(parent=self,label="Go back to default
settings!",pos=(x_button+button_width+10,y_button+35),size=button_size)

        # Bindings of buttons
        #self.default_button.Bind(wx.EVT_BUTTON, self.on_default_settings)
        #self.settings_button.Bind(wx.EVT_BUTTON, self.on_activate_settings)
        self.start_button.Bind(wx.EVT_BUTTON, self.on_start)
        self.stop_button.Bind(wx.EVT_BUTTON, self.on_stop)

        self.radio_options_mimo.Bind(wx.EVT_RADIOBUTTON, self.on_mimo_options)
        self.radio_options_siso.Bind(wx.EVT_RADIOBUTTON, self.on_mimo_options)
```

```
        self.radio_options_file_mode.Bind(wx.EVT_RADIOBUTTON, self.on_transmission_mode)
        self.radio_options_image_mode.Bind(wx.EVT_RADIOBUTTON, self.on_transmission_mode)
        self.radio_options_streaming_mode.Bind(wx.EVT_RADIOBUTTON, self.on_transmission_mode)
        #self.radio_options_benchmark_mode.Bind(wx.EVT_RADIOBUTTON, self.on_transmission_mode)

        self.Bind(wx.EVT_PAINT,self.on_paint)

        # Update default settings
        self.set_default_values()
        self.update_start_buttons()
        if self.global_ctrl.get_running():
            self.enable_settings()
        else:
            self.disable_settings()

    def set_default_values(self):
        self.options = self.global_ctrl.default_options()
        #Normal:
        self.textbox_options_size.SetValue(str(self.options.size))
        #self.textbox_options_npixels.SetValue(str(self.options.npixels))
        #self.textbox_options_input_filename.SetValue(self.options.input_filename)
        #self.textbox_options_output_filename_mimo.SetValue(self.options.output_filename_mimo)
        #self.textbox_options_output_filename_siso.SetValue(self.options.output_filename_siso)
        self.textbox_options_sample_rate.SetValue(str(self.options.sample_rate))
        self.textbox_options_discontinuous.SetValue(str(self.options.discontinuous))
        self.textbox_options_tx_amplitude.SetValue(str(self.options.tx_amplitude))
        self.textbox_options_modulation.SetValue(self.options.modulation)
        self.radio_options_mimo.SetValue(not self.options.siso)
        self.radio_options_siso.SetValue(self.options.siso)
        #self.radio_options_benchmark_mode.SetValue(self.options.benchmark_mode)
        self.radio_options_file_mode.SetValue(self.options.file_mode)
        self.radio_options_image_mode.SetValue(self.options.image_mode)
        self.radio_options_streaming_mode.SetValue(self.options.streaming_mode)
        #self.checkbox_options_verbose.SetValue(self.options.verbose)
        #self.checkbox_options_write_all.SetValue(self.options.write_all)
        #self.textbox_options_per_avg_nr.SetValue(str(self.options.per_avg_nr))
        self.coding_options.SetSelection(0)
        #Expert:
        self.textbox_options_fft_length.SetValue(str(self.options.fft_length))
        self.textbox_options_occupied_tones.SetValue(str(self.options.occupied_tones))
        self.textbox_options_cp_length.SetValue(str(self.options.cp_length))
        #self.checkbox_options_log.SetValue(self.options.log)
        #USRP:
        #self.textbox_options_usrp_addr0.SetValue(self.options.usrp_addr0)
        #self.textbox_options_usrp_addr1.SetValue(self.options.usrp_addr1)
        #self.textbox_options_center_freq.SetValue(str(self.options.center_freq))
        self.textbox_options_gain_rx.SetValue(str(self.options.gain_rx))
        self.textbox_options_gain_tx.SetValue(str(self.options.gain_tx))

        self.update_mode_options()
```

```
def set_option_parameters(self):
    #Normal:
    self.options.size = float(self.textbox_options_size.GetValue())
    #self.options.npixels = float(self.textbox_options_npixels.GetValue())
    #self.options.input_filename = str(self.textbox_options_input_filename.GetValue())
    #self.options.output_filename_mimo = str(self.textbox_options_output_filename_mimo.GetValue())
    #self.options.output_filename_siso = str(self.textbox_options_output_filename_siso.GetValue())
    self.options.sample_rate = float(self.textbox_options_sample_rate.GetValue())
    self.options.discontinuous = int(self.textbox_options_discontinuous.GetValue())
    self.options.tx_amplitude = float(self.textbox_options_tx_amplitude.GetValue())
    self.options.modulation = self.textbox_options_modulation.GetValue()
    self.options.siso = not self.radio_options_mimo.GetValue()
    self.options.mimo = self.radio_options_mimo.GetValue()
    #self.options.benchmark_mode = self.radio_options_benchmark_mode.GetValue()
    self.options.file_mode = self.radio_options_file_mode.GetValue()
    self.options.image_mode = self.radio_options_image_mode.GetValue()
    self.options.streaming_mode = self.radio_options_streaming_mode.GetValue()
    #self.options.verbose = self.checkbox_options_verbose.GetValue()
    #self.options.write_all = self.checkbox_options_write_all.GetValue()
    #self.options.per_avg_nr = int(self.textbox_options_per_avg_nr.GetValue())
    self.options.code_rate = self.coding_options.GetStringSelection()
    #Expert:
    self.options.fft_length = int(self.textbox_options_fft_length.GetValue())
    self.options.occupied_tones = int(self.textbox_options_occupied_tones.GetValue())
    self.options.cp_length = int(self.textbox_options_cp_length.GetValue())
    #self.options.log = self.checkbox_options_log.GetValue()
    #USRP:
    #self.options.usrp_addr0 = str(self.textbox_options_usrp_addr0.GetValue())
    #self.options.usrp_addr1 = str(self.textbox_options_usrp_addr1.GetValue())
    #self.options.center_freq = float(self.textbox_options_center_freq.GetValue())
    self.options.gain_rx = float(self.textbox_options_gain_rx.GetValue())
    self.options.gain_tx = float(self.textbox_options_gain_tx.GetValue())

    self.global_ctrl.set_options(self.options)

def disable_settings(self):
    #Normal:
    self.textbox_options_size.Disable()
    #self.textbox_options_npixels.Disable()
    #self.textbox_options_input_filename.Disable()
    #self.textbox_options_output_filename_mimo.Disable()
    #self.textbox_options_output_filename_siso.Disable()
    self.textbox_options_sample_rate.Disable()
    self.textbox_options_discontinuous.Disable()
    self.textbox_options_tx_amplitude.Disable()
    self.textbox_options_modulation.Disable()
    self.radio_options_mimo.Disable()
    self.radio_options_siso.Disable()
    #self.radio_options_benchmark_mode.Disable()
```

```
        self.radio_options_file_mode.Disable()
        self.radio_options_image_mode.Disable()
        self.radio_options_streaming_mode.Disable()
        #self.checkbox_options_verbose.Disable()
        #self.checkbox_options_write_all.Disable()
        #self.textbox_options_per_avg_nr.Disable()
        self.coding_options.Disable()
        #Expert:
        self.textbox_options_fft_length.Disable()
        self.textbox_options_occupied_tones.Disable()
        self.textbox_options_cp_length.Disable()
        #self.checkbox_options_log.Disable()
        #USRP:
        #self.textbox_options_usrp_addr0.Disable()
        #self.textbox_options_usrp_addr1.Disable()
        #self.textbox_options_center_freq.Disable()
        self.textbox_options_gain_rx.Disable()
        self.textbox_options_gain_tx.Disable()
        #Buttons:
        #self.default_button.Disable()
        #self.settings_button.Disable()

        self.options_enabled = False

    def enable_settings(self):
        #Normal:
        self.textbox_options_size.Enable()
        self.textbox_options_sample_rate.Enable()
        self.textbox_options_discontinuous.Enable()
        #self.textbox_options_modulation.Enable() #only qpsk for now...
        self.radio_options_mimo.Enable()
        self.radio_options_siso.Enable()
        #self.radio_options_benchmark_mode.Enable()
        self.radio_options_file_mode.Enable()
        self.radio_options_image_mode.Enable()
        self.radio_options_streaming_mode.Enable()
        #self.checkbox_options_verbose.Enable()
        #self.checkbox_options_write_all.Enable()
        #self.textbox_options_per_avg_nr.Enable()
        self.coding_options.Enable()
        # tx
        if not self.options.usrp_rx:
            #self.textbox_options_npixels.Enable()
            #self.textbox_options_input_filename.Enable()
            self.textbox_options_tx_amplitude.Enable()
        # rx
        '''if not self.options.usrp_tx:
            self.textbox_options_output_filename_mimo.Enable()
            self.textbox_options_output_filename_siso.Enable()'''
        #Expert:
```

```
        self.textbox_options_fft_length.Enable()
        self.textbox_options_occupied_tones.Enable()
        self.textbox_options_cp_length.Enable()
        #self.checkbox_options_log.Enable()
        #USRP:
        '''if self.options.usrp_rx or self.options.usrp_tx:
            self.textbox_options_usrp_addr0.Enable()
            self.textbox_options_usrp_addr1.Enable()
            self.textbox_options_center_freq.Enable()'''
        if self.options.usrp_rx:
            self.textbox_options_gain_rx.Enable()
        if self.options.usrp_tx:
            self.textbox_options_gain_tx.Enable()

        #Buttons:
        '''self.default_button.Enable()
        self.settings_button.Enable()'''
        #Adjust and disable either of the io mode options:
        self.update_mode_options()

        self.options_enabled = True

    def update_start_buttons(self):
        if self.global_ctrl.get_running():
            self.start_button.Disable()
            self.stop_button.Enable()
        else:
            self.start_button.Enable()
            self.stop_button.Disable()

    def update_mode_options(self):
        options = self.global_ctrl.get_options()
        #self.textbox_options_npixels.Disable()
        #self.textbox_options_input_filename.Disable()
        #self.textbox_options_output_filename_mimo.Disable()
        #self.textbox_options_output_filename_siso.Disable()
        if not self.global_ctrl.get_running():
            if self.radio_options_image_mode.GetValue():
                pass
            '''elif self.radio_options_file_mode.GetValue():
                if not options.usrp_rx: # benchmark or tx
                    #self.textbox_options_input_filename.Enable()
                if not self.options.usrp_tx: # benchmark or rx
                    #self.textbox_options_output_filename_mimo.Enable()
                    #self.textbox_options_output_filename_siso.Enable()
            elif not options.usrp_rx:
                self.textbox_options_npixels.Enable()'''

    def on_transmission_mode(self,e):
        self.update_mode_options()
```

```
def on_default_settings(self,e):
    self.set_default_values()

def on_activate_settings(self,e):
    self.set_option_parameters()

def on_start(self,e):
    self.set_option_parameters()
    self.main_frame.start_tb()
    self.disable_settings()
    self.update_start_buttons()

def on_stop(self,e):
    self.main_frame.stop_tb()
    self.enable_settings()
    self.update_start_buttons()

def on_mimo_options(self,event):
    if self.radio_options_mimo.GetValue():
        self.global_ctrl.set_options_mimo(1)
    elif self.radio_options_siso.GetValue():
        self.global_ctrl.set_options_mimo(0)

def on_paint(self,event):
    running_status = self.global_ctrl.get_running()
    if running_status and self.options_enabled:
        self.disable_settings()
    elif not self.options_enabled and not running_status:
        self.enable_settings()

    self.update_start_buttons()

    focus = self.global_ctrl.get_focus_view()
    if focus == "ctrl":
        pass
    else:
        self.global_ctrl.set_focus_view("ctrl")
        mimo_status = self.global_ctrl.get_options_mimo()
        if mimo_status:
            self.radio_options_mimo.SetValue(True)
            self.radio_options_siso.SetValue(False)
        else:
            self.radio_options_mimo.SetValue(False)
            self.radio_options_siso.SetValue(True)
```

10.10 FOI_MIMO_TX_PATH

```
from gnuradio import gr, gru, blks2,trellis
from gnuradio import eng_notation

import copy
import sys
import math

import foimimo
from foimimo.coding import encoding
from foimimo.ofdm_mimo import ofdm_mimo_mod
from foimimo.ofdm_mimo_with_coding import ofdm_mimo_mod_with_coding

class mimo_transmit_path(gr.hier_block2):
   def __init__(self, options):
      gr.hier_block2.__init__(self, "MIMO_transmit_path",
            gr.io_signature(0, 0, 0), # Input signature
            gr.io_signature2(2, 2, gr.sizeof_gr_complex,gr.sizeof_gr_complex)) # Output signature

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

      self._verbose     = options.verbose        # turn verbose mode on/off
      self._tx_amplitude = options.tx_amplitude    # digital amplitude sent to USRP
      self._code_rate    = options.code_rate

      self.amp_0 = gr.multiply_const_cc(1)
      self.amp_1 = gr.multiply_const_cc(1)
      self.set_tx_amplitude(self._tx_amplitude)

      if self._code_rate != "":
         # [ref] Proakis pp. 493-496
         # Rate  Generator in octal
         #  3/4   13  25  61 47
         #  2/3  236 155 337
         #  1/3   13  15  17
         code = {"1/1":[2, 2,[1, 0, 0, 1]],
               "3/4":[3,4,[0, 1, 2, 3, 1, 2, 2, 1, 3, 1, 1, 1]],
               "2/3":[2, 3, [11, 6, 11, 6, 11, 15]],
               "1/3":[1, 3, [11, 13, 15]]}
         selected_code = code[self._code_rate]

         self._msgq_limit = 2
         self._encode_fsm = trellis.fsm(selected_code[0],selected_code[1],selected_code[2])

         self.encoder = encoding(options,self._encode_fsm, self._msgq_limit)

         self.ofdm_mimo_tx = ofdm_mimo_mod_with_coding(options, self._msgq_limit, pad_for_usrp=False)

         # Create and setup transmit path flow graph
         self.connect((self.encoder,0), (self.ofdm_mimo_tx,0))
```

```
            self.connect((self.encoder,1), (self.ofdm_mimo_tx,1))
            self.connect((self.ofdm_mimo_tx,0), (self.amp_0,0), (self,0))
            self.connect((self.ofdm_mimo_tx,1), (self.amp_1,0), (self,1))
        else:
            self.ofdm_mimo_tx = ofdm_mimo_mod(options)
            self.connect((self.ofdm_mimo_tx,0), (self.amp_0,0), (self,0))
            self.connect((self.ofdm_mimo_tx,1), (self.amp_1,0), (self,1))


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

    def set_tx_amplitude(self, ampl):
        """
        Sets the transmit amplitude sent to the USRP
        @param: ampl 0 <= ampl < 32768.  Try 8000
        """
        self._tx_amplitude = max(0.0, min(ampl, 32767.0))
        self.amp_0.set_k(self._tx_amplitude)
        self.amp_1.set_k(self._tx_amplitude)

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

        if self._code_rate != "":
            return self.encoder.send_pkt(payload, eof)
        else:
            return self.ofdm_mimo_tx.send_pkt(payload, eof)

    def add_options(normal, expert):
        """
        Adds transmitter-specific options to the Options Parser
        """
        normal.add_option("-c", "--code-rate", type="string", default="",
                          help="set code rate (3/4, 2/3 or 1/3), empty for uncoded")
        normal.add_option("-a", "--tx-amplitude", type="eng_float", default=1, metavar="AMPL",
                          help="scale digital tx amplitude [default=%default]")
        normal.add_option("-v", "--verbose", action="store_true", default=False)
        expert.add_option("", "--log", action="store_true", default=False,
                          help="Log all parts of flow graph to file (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 "TX code rate:    " + self._code_rate
```

10.11 FOI_SISO_TX_PATH

```
from gnuradio import gr, gru, blks2,trellis
from gnuradio import eng_notation
import copy
import sys
import math

from foimimo.ofdm import ofdm_demod
from foimimo.ofdm_with_coding import ofdm_demod_with_coding
from foimimo.coding import decoding

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

class siso_receive_path(gr.hier_block2):
   def __init__(self, rx_callback, bad_header_callback, options):

        gr.hier_block2.__init__(self, "siso_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._log         = options.log
      self._rx_callback = rx_callback     # this callback is fired when there's a packet available
      self._bad_header_callback = bad_header_callback # this callback is fired when a packet with bad header is thrown
away

      self._code_rate    = options.code_rate

      if self._code_rate != "":
         # [ref] Proakis pp. 493-496
         # Rate  Generator in octal
         #  3/4   13  25  61 47
         #  2/3  236 155 337
         #  1/3   13  15  17
         code = {"1/1":[2, 2,[1, 0, 0, 1]],
               "3/4":[3,4,[0, 1, 2, 3, 1, 2, 2, 1, 3, 1, 1, 1]],
               "2/3":[2, 3, [11, 6, 11, 6, 11, 15]],
               "1/3":[1, 3, [11, 13, 15]]}
         selected_code = code[self._code_rate]
         # receiver
```

```
            self._decode_fsm = trellis.fsm(selected_code[0],selected_code[1],selected_code[2])
            self.ofdm_siso_demod = ofdm_demod_with_coding(options,
                                    int(round(math.log(self._decode_fsm.I())/math.log(2))),
                                    int(round(math.log(self._decode_fsm.O())/math.log(2))),
                                    bad_header_callback=self._bad_header_callback)
            self.decoder = decoding(options, self._decode_fsm, callback=self._rx_callback)

            self.connect((self,0), (self.ofdm_siso_demod,0))
            self.connect((self.ofdm_siso_demod,0), (self.decoder,0))
            self.connect((self.ofdm_siso_demod,1), (self.decoder,1))
        else:
            self.ofdm_siso_demod = ofdm_demod(options, callback=self._rx_callback)

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

            self.connect(self, self.ofdm_siso_demod)
            self.connect(self.ofdm_siso_demod, self.probe)

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

    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
    """
    normal.add_option("-c", "--code-rate", type="string", default="",
                      help="set code rate (3/4, 2/3 or 1/3), empty for uncoded [default=%default]")
    normal.add_option("-v", "--verbose", action="store_true", default=False)
    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
    """
    pass
```