

Spectrum Sensing using Software Radios

Design Document May10-06

Team members:

Matt Dolter

Eun Kim

Seun Guy Min

Advisor:

Dr. Zhengdao Wang

December 4th 2009



DISCLAIMER: This document was developed as part of the requirements of an electrical and computer engineering course at Iowa State University, Ames, Iowa. The document does not constitute a professional engineering design or a professional land surveying document. Although the information is intended to be accurate, the associated students, faculty, and Iowa State University make no claims, promises, or guarantees about the accuracy, completeness, quality, or adequacy of the information. Document users shall ensure that any such use does not violate any laws with regard to professional licensing and certification requirements. Such use includes any work resulting from this student-prepared document that is required to be under the responsible charge of a licensed engineer or surveyor. This document is copyrighted by the students who produced the document and the associated faculty advisors. No part may be reproduced without the written permission of the senior design course coordinator.



Table of Contents

List of Figures	4
Concept Sketch	5
System Design	5
System Requirements	5
Functional Decomposition	5
System Analysis.....	6
Functional Requirements.....	7
Non-Functional Requirements.....	7
Detailed Design	7
System Inputs / Outputs	7
User Interface	8
Hardware Specifications	8
Software Specifications.....	11
Testing and Evaluation.....	13
Test Plan.....	13
Modeling.....	14
Project Specific Components	16
Software Design	16
Appendix:	19
References	22



List of Figures

Figure 1 - General concept of cognitive radios and spectral sensing [7]	5
Figure 2 - A picture of the systems input flow	7
Figure 3 - A Diagram of the process for transmission [8]	8
Figure 4 - A picture of the main board inside the USRP with daughter boards attached [5]	9
Figure 5 - USRP block diagram [3]	10
Figure 6 - FPGA internal processes [4]	11
Figure 7 - Picture of the code structure	12
Figure 8 - Basic structure of python code	12
Figure 9 - Spectrum sample	14
Figure 10 - MatLab plot of captured samples	15



Concept Sketch

The main concept of the project is based on cognitive radios. Through spectrum sensing the radio is able to find opportunity bands for transmission (See Figure 1). The sensing is done through the energy detector method and using data from statistical analysis. Detection of the primary user is the first goal of this project. Next, the secondary user parameters for transmission will need to be determined. This will be based on the specifications of the secondary user. The last part of this project is implementing secondary user transmission of data.

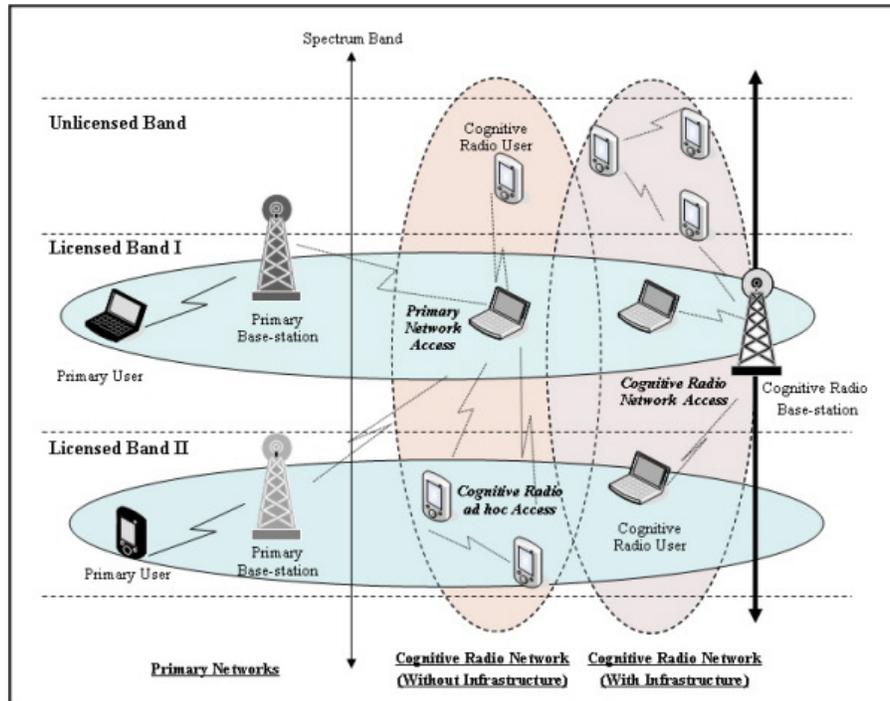


Figure 1 - General concept of cognitive radios and spectral sensing [7]

System Design

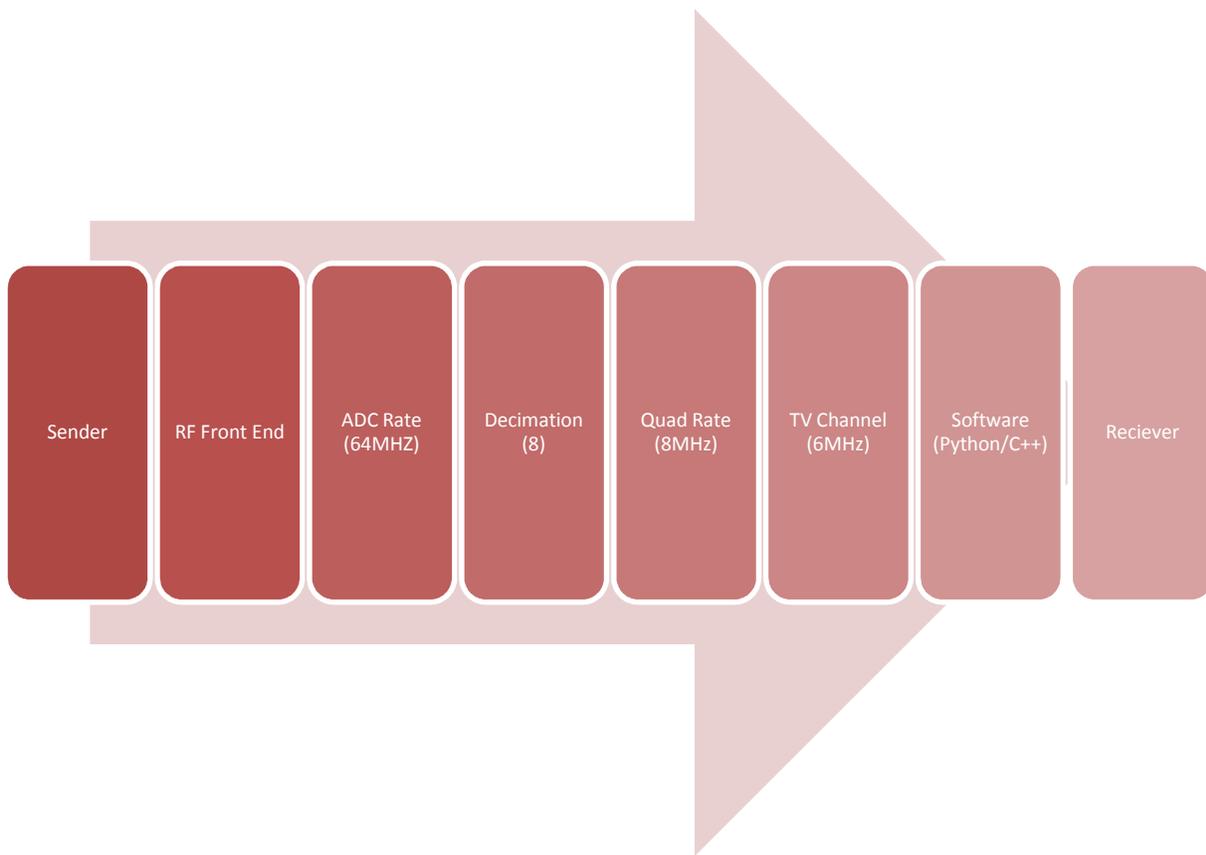
System Requirements

For the spectrum sensing, there are two ways to detect primary user spectrum. One is overlay method to utilize the white spectrum which is not being used by the primary user at a particular time. The other is underlay method to use the spectrum at the same time with priority users by making the primary user hardly feel the interference with very low transmitting power. For this project we use overlay method, so the user needs to transmit data over empty frequencies determined by sensing. Also, 100-800 MHz is the spectral range of interest, and user can change parameters of the sensing.

Functional Decomposition

For the functional decomposition, we determine how the functions such as software and hardware can be decomposed.





System Analysis

1. USRP(universal software radio peripheral) will sample the information of primary user spectrum at a rate of 64M complex samples bites which is referred to ADC sampling rate. So our ADC sampling rate is 64MHz.
2. PC with USB connection can handle 32MB /sec, and each sample is 4 byte, so USB can support only 8MB/sec of data.
3. ADC sampling rate (64MHz) is too high to handle because PC can only receive 8MHz. For that reason, we need to do down-sampling. Decimation by a factor of at least 8 is required for no data loss in this case.
4. By taking decimation of data by a factor of at 8, our quad rate will be 8MHz(=64M/8), and this is big enough because we only need 6MHz wide bands which is a bandwidth of TV channel.(8Mhz>6Mhz)
5. Then, the data need to move to the lower frequency which is called IF frequency by multiplying cos/sin wave generation and passing the low-pass filter.
6. Finally, we can display this data of primary user spectrum by using software such as python, C++. For that process, we will use TV spectrum/channel, so the software program is going to loop over the 100-800MHz band in increments of 6MHz.



Functional Requirements

FR01: As per IEEE 802.22, the system will have a 90% probability of detection and a 10% probability of false alarm. [8]

FR02: The user will be able to control the parameters of the sensing to optimize either length or accuracy of the sensing.

FR03: The transmitter will be able to transmit over the 100 to 800 MHz range.

FR04: The system should stop transmitting if it detects primary user activity at any time.

Non-Functional Requirements

NFR01: The system should have a user friendly interface.

NFR02: Code should be written in an understandable manner to allow for extensibility in future.

Detailed Design

System Inputs / Outputs

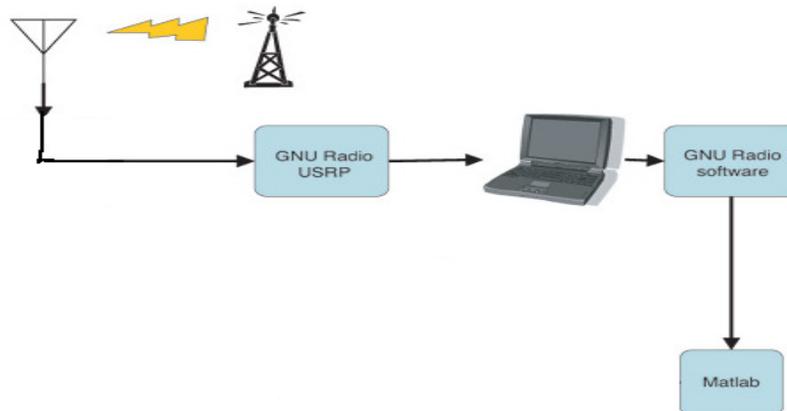


Figure 2 - A picture of the systems input flow

The picture above shows how the signals flow through the system. The antenna picks of the signals from the air and feeds them into the USRP. From the USRP, the signals are sampled and transmitted to the PC. The code in GNU radio processes the samples and outputs to a file readable by MatLab.

Inputs:

User sensing parameters

User transmission data



Outputs:

Sensing data and decision

Transmission status

User Interface

The user interface for our project will be the command line in Linux. All input and output from and to the user will be conducted here. There will be parameters of the sensing that the user can choose. After sensing the user will be able to specify a data source and destination for transmission. The transmission will take place based on the data obtained from the sensing. The interface will be kept clean and understandable in order facilitate ease of system use.

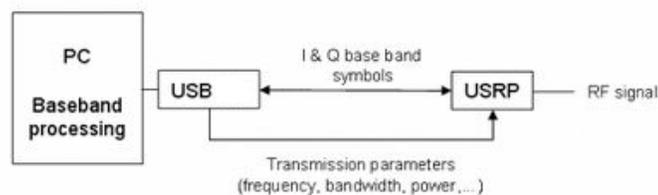


Figure 3 - A Diagram of the process for transmission [8]

Hardware Specifications

1. Hardware Overview

The hardware for most of this project is being supplied by the Iowa State University ECpE department. The department owns several software defined radios (SDRs) that are available for use. These will be programmed to achieve the goals of this project. The SDRs are made by Ettus Research and feature a main programmable processing board that has four slots; two for transmitter cards and two for receiver cards. They may also be referred to Universal Software Radio Peripherals (USRPs).



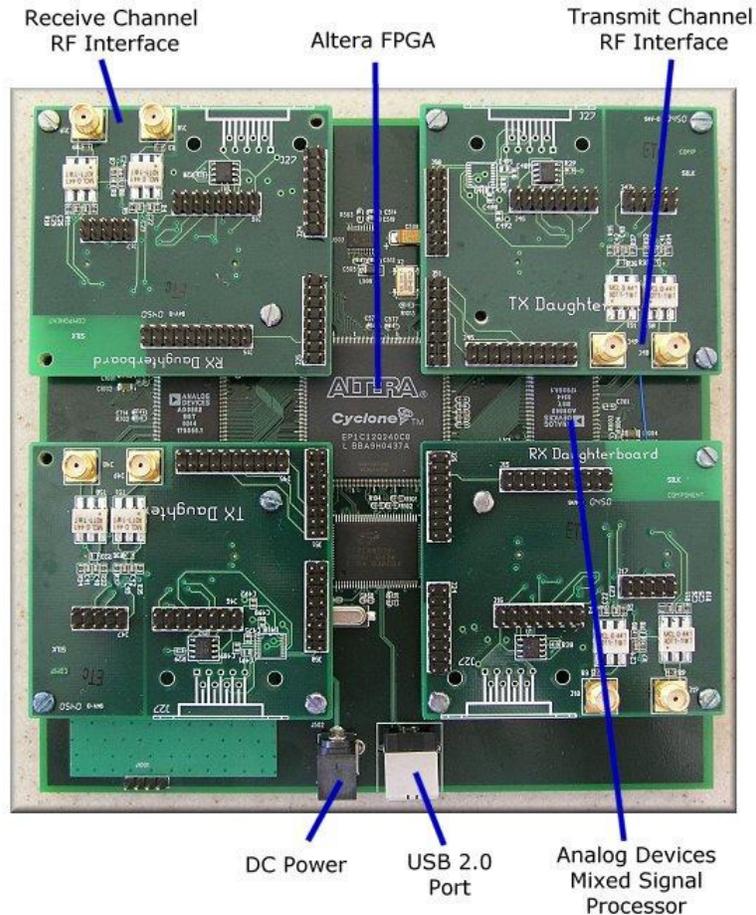


Figure 4 - A picture of the main board inside the USRP with daughter boards attached [5]

The receiver card that will be used for this project is a TV spectrum receiver. It can receive signals in the frequency band from 50 – 860 MHz. By default, it receives 6 MHz wide bands of data at a time (this is the same width as each broadcast TV station). The center frequency is controlled by the user.

For the transmission part of this project, a separate TV spectrum transmitter may need to be bought. This may be challenging because transmission in the TV spectrum is restricted to licensed users.

2. Internal Hardware Structure

The picture below shows how the data flows through the USRP. The red blocks represent the daughter boards (two for receiving and two for transmitting). The receive daughter boards are connected to 12 bit analog to digital converters (ADCs). Here, the signals are sampled and sent to the FPGA. The FPGA does digital down converting on the signals and sends them of the USB link to the PC.



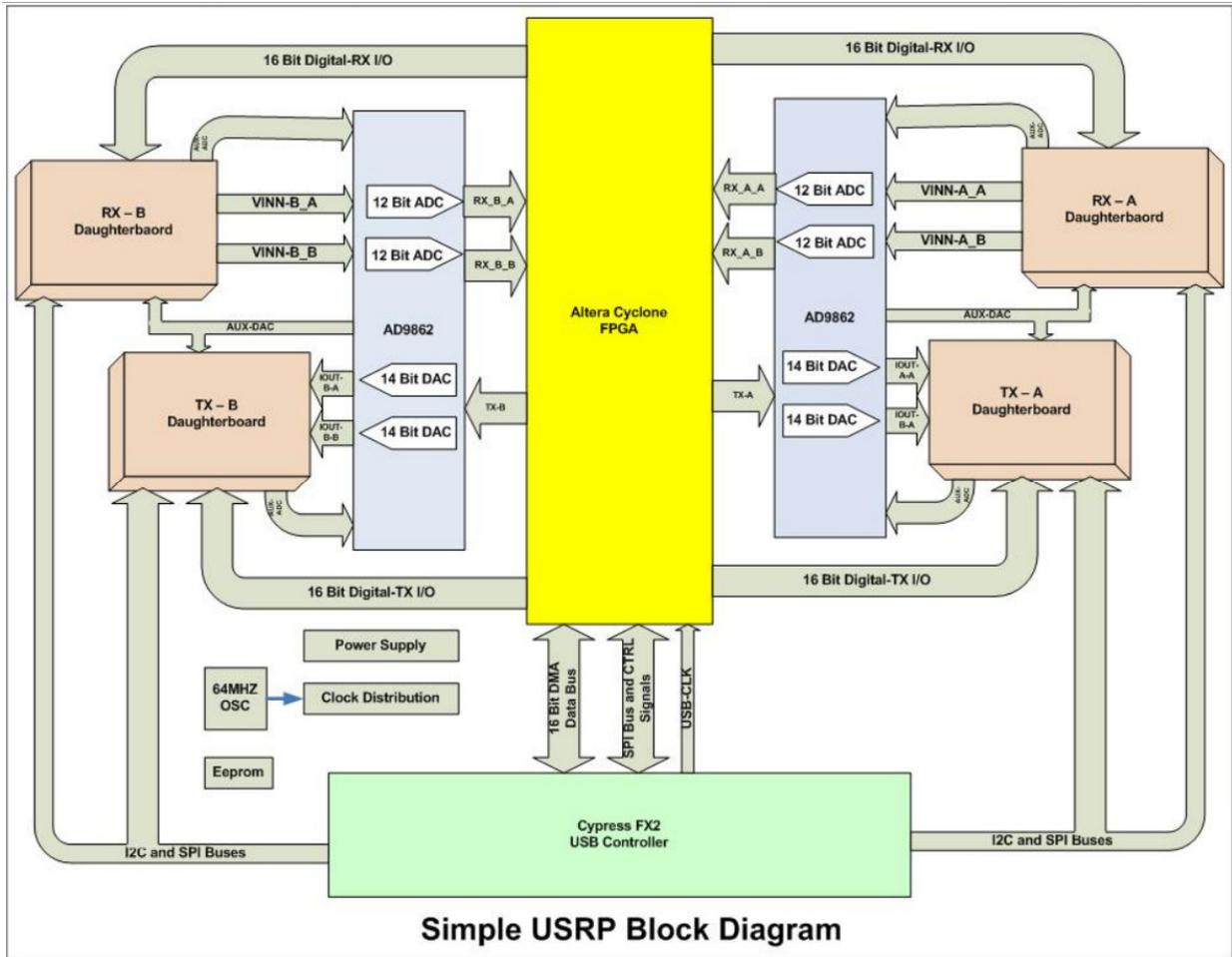


Figure 5 - USRP block diagram [3]

The SDRs interface with the computers through USB. From USB specifications, the data rate and therefore, the sampling rate, can be calculated as follows. USB can support up to 32 MB/sec of data. Each complex sample from the USRP is 4 bytes, so the data link can transfer 8 M complex samples per second. By default, the USRP samples at a rate of 64 M complex samples per second. The system will need to decimate by at least a factor of 8 for no loss of data.



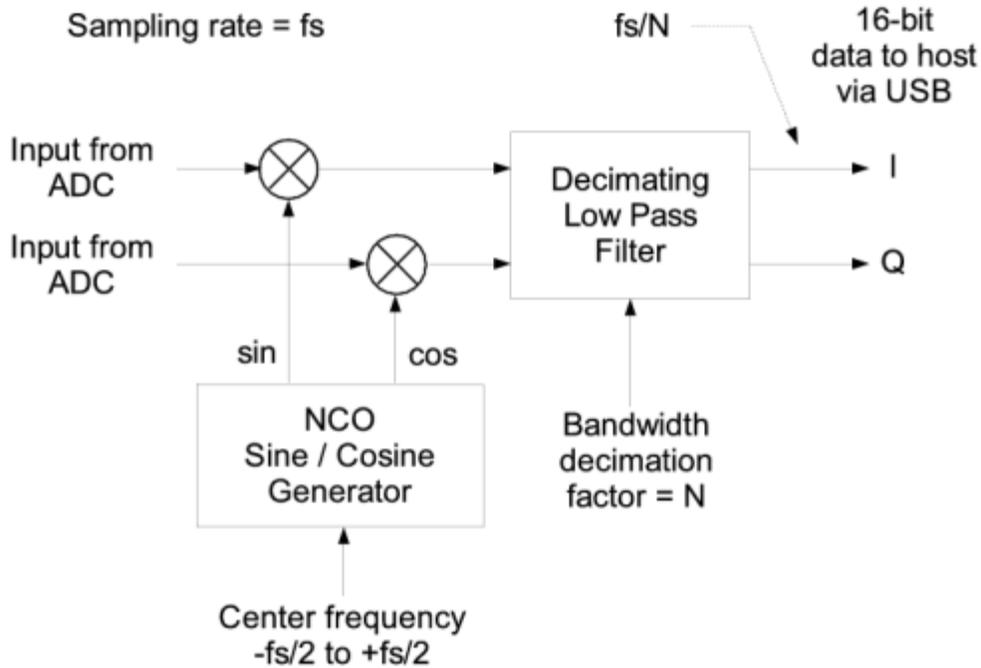


Figure 6 - FPGA internal processes [4]

The above picture shows, in detail, what happens inside the FPGA. The multiplication by sin and cos down convert the signals to base band. I and Q represent the real and imaginary parts of the signal. The low pass filter decimates the samples by a factor of N.

Software Specifications

The USRPs made by Ettus Research are programmed and controlled by open source products. Therefore, the radios will be used on Linux machines with GNU radio software installed. This has all been set up in the SDR lab at Iowa State University.

The radios are programmed in two ways. First, signal processing blocks are written in C++ programming language. This is considered the lower level programming. The GNU radio community has already built an extensive library of signal processing blocks for communal use. The second part of the programming is done in Python. This involves importing the libraries of blocks, creating instances of the blocks, connecting them all together in a flow graph, and running the program.

This project will focus on the Python programming aspect of the USRPs since there are many C++ signal processing blocks already created. We may need to eventually program a few of these blocks if we find that the library is missing something we need.



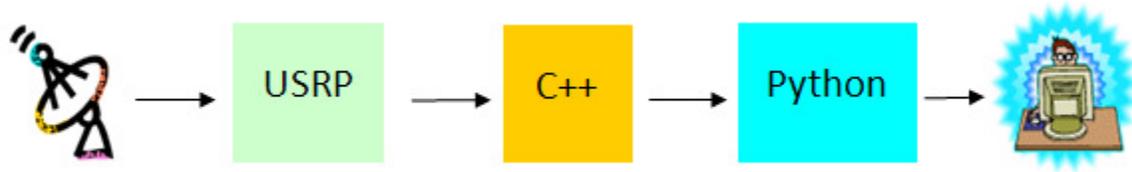


Figure 7 - Picture of the code structure

1. Python Structure

Python has extensive library for users. We use the standard library directly or do some modification to obtain the desired purpose. To use Python portions of the library, we need to import a something called a module or package to our program. In the package case, we need to import `'__init__.py'` first and then call other modules which are in the same directory. Since Python has interpreter, we don't need to do compile.

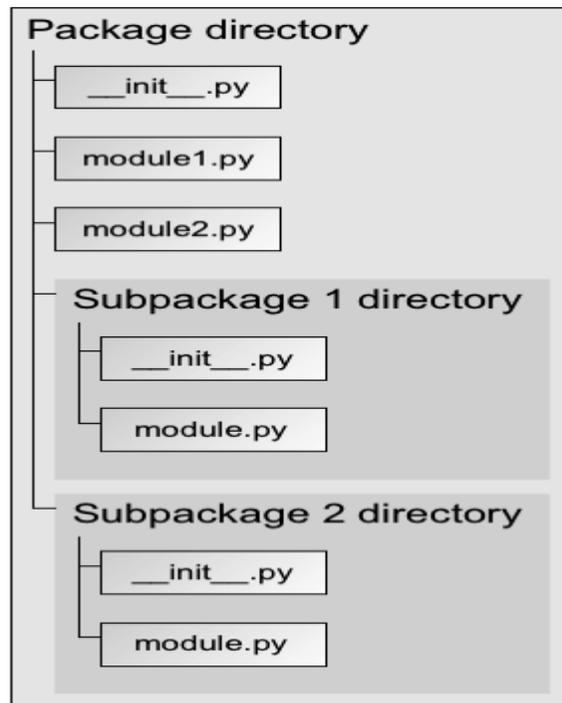


Figure 8 - Basic structure of python code

2. Packages and Modules used for this project

gnuradio: Main package. All gnuradio things are in the package.

gr: GNU radio sub package. Swig output and other modules into the `gnuradio.gr` namespace.

gru: GNU radio sub package.



eng_notation: Python file. Allows use of engineering notation.

usrp: Python file. Set up hardware configuration interface for the USRP.

blks2: GNU radio package. The blocks in the package are implemented using hier_block2.

gnuradio.wxgui: GNU radio package. Import WX Python based GNU radio extension.

stdgui2: Python file. A simple WX GUI for GNU radio applications. Use hier_block2.

fftsink2: Python file which is in WX GUI sub package. GNU radio spectrum analyzer using stdgui2 and heir_block2.

form: Python file. Import GNU radio WX GUI form.

usrp_dbid: Python file. The file contains all USRP daughter boards ID's.

3. Functions of the Python library used

top_block(): Top-level hierarchical block representing a flow graph. Use string as parameters.

source_x(): Interface to USRP RX path.

tune(): Set the center frequency for USRP. Tuning is a two step process. First ask the front-end to tune as close to the desired frequency as it can. Then use the result of that operation and our target_frequency to determine the value for the digital down converter.

set_mux(): Determines which ADC is connected to each DDC input.

fft_sink_c: FFT sink for complex data samples.

set_freq(): Set the frequency. Get center frequency from user.

file_sink: Write a stream to a binary file.

Testing and Evaluation

Test Plan

Testing will need to be done at various stages throughout the project. The first testing that will be done is whether or not the data collection being done for modeling purposes is accurate. The current plan for this testing to compare energy detection data to current TV broadcast schedule to see if the data looks right. Another part of this testing might be done with a spectrum analyzer. The output of the spectrum analyzer could be compared to the collected data to see if they match.

The next part of testing in this project will be during the decision making part. Once the program decides which frequencies are currently being used and which ones are not, the decisions can be compared to the TV broadcast schedule as in the previous situation.



Lastly, the transmission part of this project will present one more testing situation. Was the transmitted data correctly received and decoded. This can be tested by using a second SDR to sense the spectrum and find out if there is data being transmitted at the correct frequency. This will use the sensing part of the project and depend on its accuracy.

Modeling

Three types of spectrum output captured are shown below. Center frequency is 150 MHz and bandwidth is 3 MHz for testing.

1. Spectrum in frequency domain using FFT

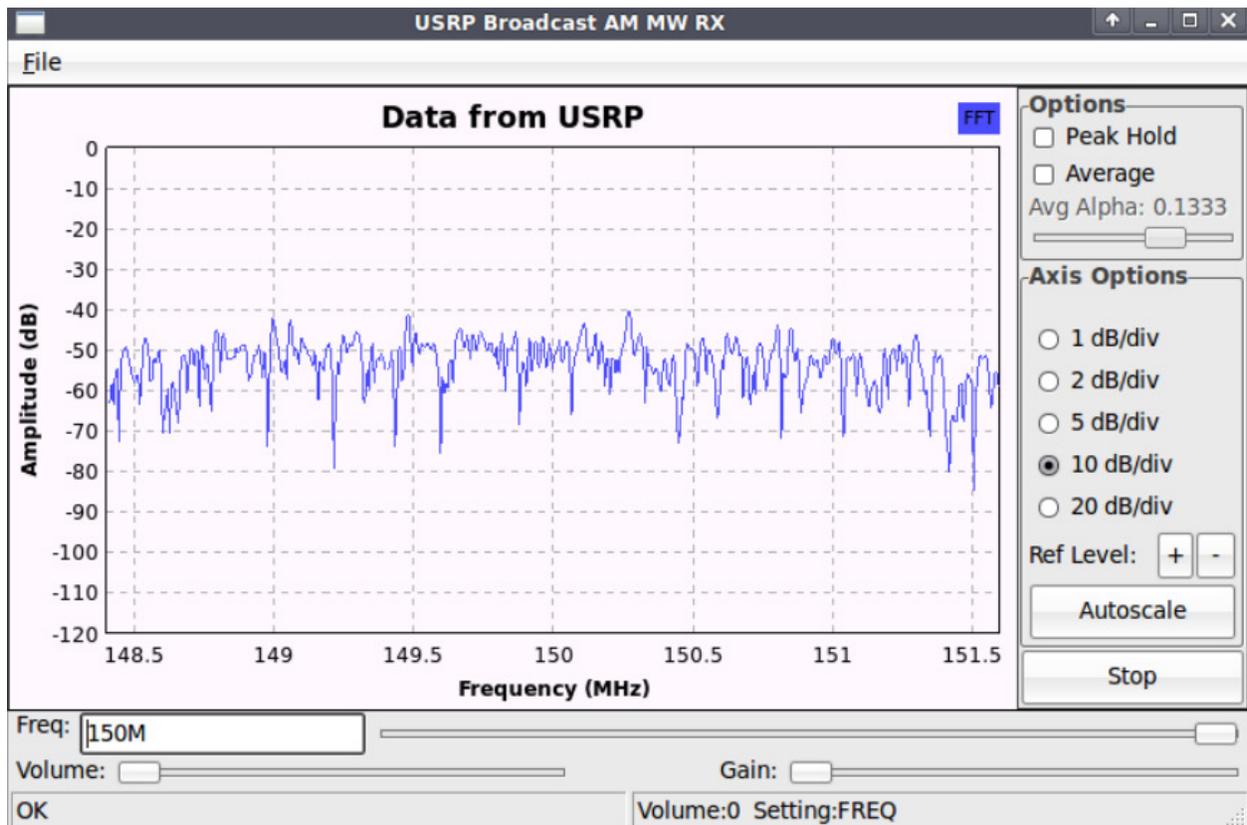


Figure 9 - Spectrum sample

The picture above is captured spectrum in frequency domain with going through Fast Fourier Transform (FFT). The bandwidth USRP covers is 6 MHz but here we set that decimation is 20, which works as a low pass filter to get 3 MHz with center frequency 150 MHz for testing.

2. 1000 samples of spectrum in time domain (sampling rate : 3.2 MHz)



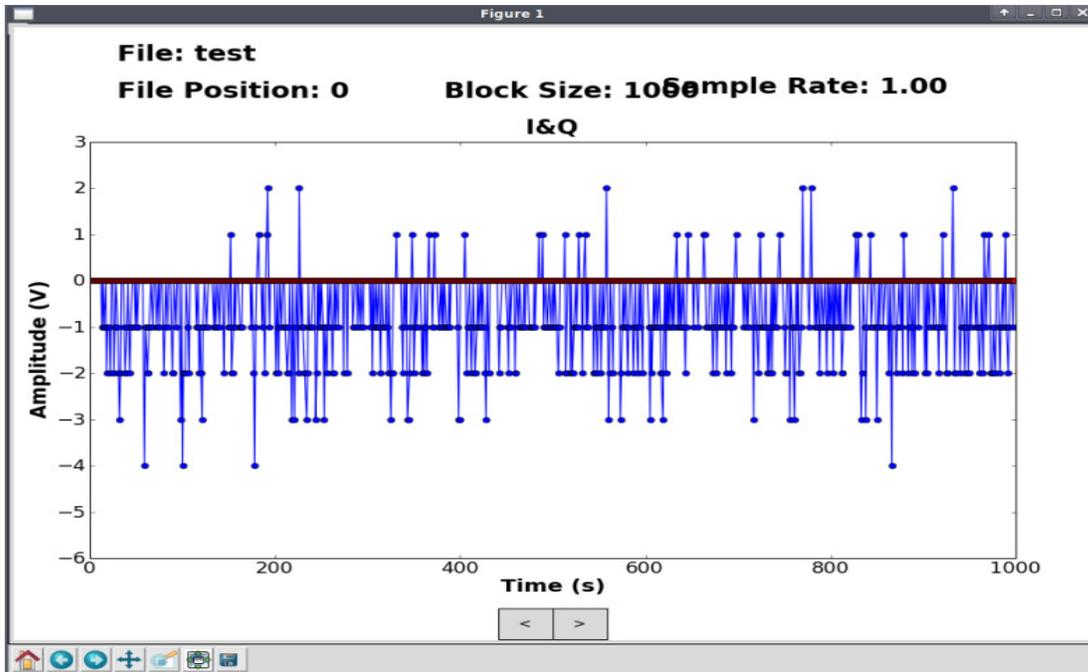


Figure 10 - MatLab plot of captured samples

We plot binary data after capturing frequency. Sampling rate we set is 3.2 Mbps. The picture above shows the first part of digitized data with amplitude in time domain.

3. Part of text form of spectrum transformed from binary using MatLab

After getting binary data we made transformation from binary to text file to be readable using MatLab. The MatLab codes are in box and part of the output is below.

```

fid = fopen('test','r')
A = fread(fid, 'int16');
dlmwrite('data.txt',A,',')
fclose(fid)

```

```

0
-15968
0
16704
0
0
0
0
16792
0

```



```
-16256
0
16672
0
0
0
0
-16032
0
-16384
0
-16384
0
0
```

Project Specific Components

Software Design

The Python program below is modified from Python AM demodulation program for the purpose of the project, capturing spectrum.

The code below imports packages and modules from python library to capture frequency. Detail descriptions of them in Software Specifications section. All are in gnuradio package and before import them we need to write 'def __init__' to indicate that gnuradio is package.

```
from gnuradio import gr, gru, eng_notation
from gnuradio import usrp
from gnuradio import blks2
from gnuradio.eng_option import eng_option
from gnuradio.wxgui import slider, powermate
from gnuradio.wxgui import stdgui2, fftsink2, form
from optparse import OptionParser
from usrpm import usrp_dbid
```

To interface with USRP, the devices used for interface are needed to be defined in program. Below codes shows the names of devices we used to communicate between USRP and computer.



```
def pick_subdevice(u):
    return usrp.pick_subdev(u, (usrp_dbid.BASIC_RX, usrp_dbid.LF_RX,
                               usrp_dbid.TV_RX, usrp_dbid.TV_RX_REV_2,
                               usrp_dbid.TV_RX_REV_3))
```

The `wfm_rx` block is the main class of the program. Signal processes such as the method of demodulation, making graph, making output, etc are defined in the class. It also defines graphical layout of output.

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

'`usrp.source_c()`' has captured frequency information as complex numbers. '`usrp_decim`' works two ways: first, it decreases sampling rate. Once signal is captured, it is sampled with 64 Mbps. USB used for interface only supports 8 Mbps so the sampled information is needed to down. Second it works as low pass filter. Since the bandwidth USRP covers is 3 MHz we need only the range. Therefore, we use decimation = 10 which covers enough bandwidth we desire.

```
        self.u = usrp.source_c()    # usrp is data source
        adc_rate = self.u.adc_rate()
        usrp_decim = 10
        self.u.set_decim_rate(usrp_decim)
        usrp_rate = adc_rate / usrp_decim
```

We need frequency information to analyze, so we made output as a file. GNU radio library already has the function so we used it. Since the sampling rate of the received data from USRP to computer is 6.4 Mbps (64 Mbps divided by 10), we used the same sampling rate to make a file.

```
        self.dst = gr.file_sink(gr.sizeof_gr_complex, 'test')
        self.head = gr.head(gr.sizeof_gr_complex, int(6.4e6))
        self.connect(self.u, self.head, self.dst)
```



Even though we don't need to use visual graph see the frequency captured is better while capturing frequency. Therefore we import the graphical module of python and connected source data with it.

```
self.src_fft = fftsink2.fft_sink_c(self.panel, title="Data from USRP",
                                  fft_size=512, sample_rate=usrp_rate, ref_scale=32768.0, ref_level=0.0,
                                  y_divs=12)

self.connect (self.u, self.src_fft)

vbox.Add (self.src_fft.win, 4, wx.EXPAND)
```

Since we use 'fft_sink' as visual we get our desire frequency from the window and 'usrp.tune()' get the frequency. 'usrp.tune()' uses the target frequency as center frequency.

```
def set_freq(self, target_freq):

    r = usrp.tune(self.u, 0, self.subdev, target_freq)

    if r:

        self.freq = target_freq

        self.myform['freq'].set_value(target_freq) # update displayed value

        self.myform['freq_slider'].set_value(target_freq) # updated value

        self.update_status_bar()

        self._set_status_msg("OK", 0)

    return True
```



Appendix

Python program code

```
from gnuradio import gr, gru, eng_notation
from gnuradio import usrp
from gnuradio import blks2
from gnuradio.eng_option import eng_option
from gnuradio.wxgui import slider, powermate
from gnuradio.wxgui import stdgui2, fftsink2, form
from optparse import OptionParser
from usrpm import usrp_dbid
import sys
import math
import wx

def pick_subdevice(u):
    return usrp.pick_subdev(u, (usrp_dbid.BASIC_RX, usrp_dbid.LF_RX, usrp_dbid.TV_RX,
                               usrp_dbid.TV_RX_REV_2, usrp_dbid.TV_RX_REV_3))

class wfm_rx_block (stdgui2.std_top_block):
    def __init__(self,frame,panel,vbox,argv):
        stdgui2.std_top_block.__init__(self,frame,panel,vbox,argv)
        parser=OptionParser(option_class=eng_option)
        parser.add_option("-R", "--rx-subdev-spec", type="subdev", default=None,
                          help="select USRP Rx side A or B (default=A)")
        parser.add_option("-f", "--freq", type="eng_float", default=1008.0e3,
                          help="set frequency to FREQ", metavar="FREQ")
        parser.add_option("-I", "--use-if-freq", action="store_true", default=False,
                          help="use intermediate freq (compensates DC problems in quadrature boards)")
        parser.add_option("-g", "--gain", type="eng_float", default=None,
                          help="set gain in dB (default is maximum)")
        (options, args) = parser.parse_args()
        if len(args) != 0:
            parser.print_help()
            sys.exit(1)
        self.frame = frame
        self.panel = panel
        self.use_IF=options.use_if_freq
        if self.use_IF:
            self.IF_freq=64000.0
        else:
            self.IF_freq=0.0
        self.state = "FREQ"
        self.u = usrp.source_c()          # usrp is data source
        adc_rate = self.u.adc_rate()
        usrp_decim = 10
        self.u.set_decim_rate(usrp_decim)
        usrp_rate = adc_rate / usrp_decim
        self.u.set_mux(usrp.determine_rx_mux_value(self.u, options.rx_subdev_spec))
        self.subdev = usrp.selected_subdev(self.u, options.rx_subdev_spec)
        print "Using RX d'board %s" % (self.subdev.side_and_name(),)
        self.dst = gr.file_sink(gr.sizeof_gr_complex, 'test')
```



```

self.head = gr.head(gr.sizeof_gr_complex, int(64e6))
self.connect(self.u, self.head, self.dst)
if options.gain is None:
    g = self.subdev.gain_range()
    if True:
        options.gain = float(g[1])
    else:
        options.gain = float(g[0]+g[1])/2
def _set_status_msg(self, msg, which=0):
    self.frame.GetStatusBar().SetStatusText(msg, which)
def _build_gui(self, vbox, usrp_rate, demod_rate, audio_rate):
    def _form_set_freq(kv):
        return self.set_freq(kv['freq'])
    if 1:
        self.src_fft = fftsink2.fft_sink_c(self.panel, title="Data from USRP",
            fft_size=512, sample_rate=usrp_rate, ref_scale=32768.0, ref_level=0.0, y_divs=12)
        self.connect (self.u, self.src_fft)
        vbox.Add (self.src_fft.win, 4, wx.EXPAND)
    # control area form at bottom
    self.myform = myform = form.form()
    hbox = wx.BoxSizer(wx.HORIZONTAL)
    hbox.Add((5,0), 0)
    myform['freq'] = form.float_field(
        parent=self.panel, sizer=hbox, label="Freq", weight=1,
        callback=myform.check_input_and_call(_form_set_freq, self._set_status_msg))
    myform['gain'] = \
        form.quantized_slider_field(parent=self.panel, sizer=hbox, label="Gain",
            weight=3, range=self.subdev.gain_range(), callback=self.set_gain)
        hbox.Add((5,0), 0)
    vbox.Add(hbox, 0, wx.EXPAND)
    try:
        self.knob = powermate.powermate(self.frame)
        self.rot = 0
        powermate.EVT_POWERMATE_ROTATE (self.frame, self.on_rotate)
        powermate.EVT_POWERMATE_BUTTON (self.frame, self.on_button)
    except:
        print "FYI: No Powermate or Contour Knob found"
def on_rotate (self, event):
    self.rot += event.delta
    if (self.state == "FREQ"):
        if self.rot >= 3:
            self.set_freq(self.freq + .1e6)
            self.rot -= 3
        elif self.rot <=-3:
            self.set_freq(self.freq - .1e6)
            self.rot += 3
def set_freq(self, target_freq):
    r = usrp.tune(self.u, 0, self.subdev, target_freq)

    if r:
        self.freq = target_freq

```



```
        self.myform['freq'].set_value(target_freq) # update displayed value
        self.myform['freq_slider'].set_value(target_freq) # updated value
        self.update_status_bar()
        self._set_status_msg("OK", 0)
        return True
    self._set_status_msg("Failed", 0)
    return False
def set_gain(self, gain):
    self.myform['gain'].set_value(gain) # update displayed value
    self.subdev.set_gain(gain)
if __name__ == '__main__':
    app = stdgui2.stdapp (wfm_rx_block, "USRP Broadcast")
    app.MainLoop ()
```



References

- [1] Department of Electrical and Computer Engineering “Simple User Manual for Gnuradio 3.1.1” Dec 5, 2009. <http://www.ece.jhu.edu/~cooper/SWRadio/Simple-Gnuradio-User-Manual-v1.0.pdf>
- [2] Ettus Research LLC. Retrieved October 3, 2009. From the World Wide Web : <http://www.ettus.com/>
- [3] Firas Abbas Hamza. “The USRP Under 1.5X Magnifying Lens”
- [4] GNU Radio “usrp_am_mw_rcv.py” Dec 5, 2009.
http://gnuradio.org/trac/browser/gnuradio/trunk/gnuradio-examples/python/usrp/usrp_am_mw_rcv.py?rev=7606
<http://www.gnu.org/software/gnuradio/doc/ddc.png>
- [5] John Jackey. FPGA for MRFM Cantilever Control. Retrieved December 4, 2009. From the World Wide Web : <http://www.research.cornell.edu/KIC/events/MRFM2006/pdfs/Jacky%20talk/jacky-talk.html>
- [6] Microsoft Office Online. Retrieved October 3, 2009. from the World Wide Web : <http://office.microsoft.com/en-us/clipart/results.aspx?qu=technology&sc=20#492>
- [7] School of Electrical and Computer Engineering at the Georgia Institute of Technology. (2008). Retrieved October 3, 2009. from the World Wide Web : http://www.ece.gatech.edu/research/labs/bwn/CR/cog_arch.jpg
- [8] Tevfik Yücek and Hüseyin Arslan. (FIRST QUARTER 2009). “A Survey of Spectrum Sensing Algorithms for Cognitive Radio Applications” *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, VOL. 11, NO. 1
- [9] Tools 4 SDR. Dec. 5 2009.
http://www.tools4sdr.com/v2/images/thumb/5/5f/USRP_Scheme_Block.jpg/400px-USRP_Scheme_Block.jpg

