

Air Band Scanner with Retransmission over Local FM Radio Frequencies Using a Software Defined Radio: Final Report

*Matthew Dolloff
Cleveland State University
EEC 687, Spring 2009*

1 Introduction

The popularity of software defined radios (SDR) is growing steadily in modern wireless communication research. SDR's allow the user to reconfigure the radio very quickly with no hardware modifications. The functions typically found in the hardware of the radio (signal processing, etc.) are all done in software. For example, this allows the SDR to receive AM radio transmission at one moment in time and then with a quick software reconfiguration to be transmitting data at 2.4GHz.

Even NASA is getting involved in the research of SDRs. The CoNNeCT program is examining the use of software defined radios on future manned space flight missions [1].

2 Background

2.1 Hardware

The primary hardware used in this project is the Universal Software Radio Peripheral (USRP). The USRP is manufactured by Ettus Research LLC and serves as the programmable hardware portion of the SDR. It has a Universal Serial Bus (USB) connection to interface with a computer. In addition, it has several different daughter board cards that plug into the USRP main board. The daughter cards allow for different frequencies since certain hardware has frequency limitations. In this project the original USRP 1 was used, not the newer USRP 2.



Figure 1: USRP [2]

2.2 Software

The USRP only provides the hardware portion of the system. The software portion of the project is implemented using GNU Radio. GNU Radio is an open source package that implements the modulation and demodulation functions that are typically implemented in the hardware portion of a radio.

GNU Radio libraries are implemented in C++. While the main GNU Radio functions are implemented in C++, the front end of GNU Radio applications are typically written in the Python scripting language. Python allows the user to write simple scripts that invoke the C++ libraries. It also allows for the user to create simple Python Graphical User Interfaces (GUI's) using wxPython (a standard python package).

GNU Radio version 3.1.3 was used in this project.

2.3 Air Band

All commercial and private aircraft have to communicate with the ground in order to travel safely. In order to accomplish this, part of the frequency spectrum is set aside by the Federal Communications Commission (FCC). The frequencies from 108 MHz to 137 MHz are set aside by the FCC [3]. Air traffic controllers communicate with planes over this band of frequencies. Different frequencies are used depending what the aircraft is doing (landing, taking off, taxiing, etc.). By using different frequencies air traffic controllers are able to minimize the number of aircraft trying to communicate at the same time, and it allows aircraft just passing through the airspace high above an airport to not have to be distracted by the much busier "chatter" on the frequencies used for landing, etc. In fact Cleveland Hopkins International Airport has approximately 14 different frequencies it uses [4].

Despite the frequencies only being slightly higher than the FM radio frequency range, air band transmissions are done using amplitude modulation (AM).

3 Project Goal

The goal of this project is to receive aircraft transmissions with the TVRX daughter board. The received transmission will then be retransmitted, using a BasicTX daughter, over an FM frequency that can be received using a standard FM radio. Figure 2 is a graphic of how the system will work. In addition to serving as an airband repeater, the radio will also be able to receive any AM signal that is within the frequency range of the chosen daughterboard. The TVRX daughterboard used in this implementation can receive 50 MHz to 870 MHz. Essentially, this project will create an AM to FM repeater.

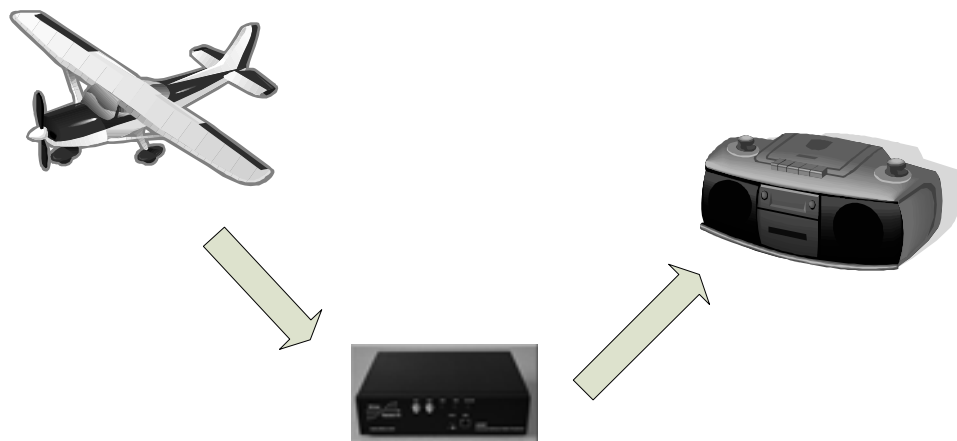


Figure 2: Project Goal

4 Project Development

This project was broken into three separate pieces to facilitate modular development. The three separate phases are described in the following sections.

4.1 Part 1: Air Band Reception

This part of the project was based off of the example program `usrp_am_mw_rcv.py` in the GNU Radio software package. Initially the program was used “as-is” to verify that the USRP was working correctly. Once it was determined that the USRP was working correctly by receiving a local AM radio station. The development progressed into receiving an airband transmission.

When attempting to receive an airband transmission, several problems were encountered. First of all it was difficult to determine which frequency Cleveland Hopkins was actually using at a given point in time. Using Airnav.com as a guide to the frequencies Hopkins using the different frequencies were scanned manually. The difficult part was actually hearing a transmission. Airband transmissions are much sparser in nature (not like a radio station that is continuous).

In addition the standard “rabbit ears” antenna that was being used was not adequate to receive airband transmission reliably. In order to overcome this problem, a quarter-wave ground plane antenna was built. The antenna was constructed out of an SMA-239 bulkhead connector and 10 AWG copper house wiring. The vertical piece of the antenna is 23” long and the legs of the antenna are approximately 24” long and bent at a 45 degree angle from the base [5]. Figure 3 is a picture that was used as the model for the antenna that was built.



Figure 3: Quarter Wave Ground Plane Antenna [5]

By virtue of the lengths of the vertical and the legs, this antenna is tuned to 120 MHz. This makes it ideal, from a frequency standpoint, for receiving airband transmissions. To make it even better, from a signal strength standpoint, it would be mounted on the roof of a building to maximize its reception capabilities. However, since portability was important in this project that was not possible, and indeed the reception could still be better, but is significantly better than the initially used “rabbit ears.”

4.2 Part 2: FM Transmission

In order to accomplish the retransmission portion of the project an example program was found on the internet that streamed MP3 files to an FM frequency that could be picked up on an FM radio. That application was actually linked in the GNU Radio mailing lists [6]. In order to make this application work however the newest version of Sox had to be compiled and installed.

A simple 900 MHz antenna was connected to the output of the BasicTX daughter board that was used to transmit the MP3's. However this again appeared to be an inadequate antenna. Instead a simple single wire antenna normally connected to a stereo as the transmitting antenna. This worked much better and the signal strength received on the portable FM radio was much stronger.

Once the transmission of the MP3's was working it was possible to take the sample code from the GNU Radio software and the example code from the mailing lists and create an output stream that was possible to use as a sink for the repeater.

4.3 Integration and Overall Design

The final integration of the project started by using the aforementioned `usrp_am_mw_rcv.py` example program. The necessary code to retransmit the received signal developed in Part 2 was added, and unneeded code was removed. The flow of the signal processing blocks can be seen in Figure 4.

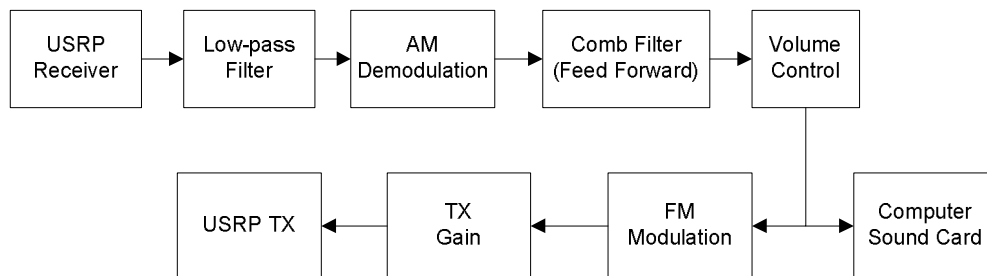


Figure 4: Function Block Diagram

It is worth noting that the software was setup to sink the demodulated AM signal to both the computer sound card and the FM modulation string. This allows the user to use either one, or both, to listen to the received signal.

In addition to the code to demodulate and remodulate a GUI was created to enable the user to quickly and easily change to local airport frequencies as well as to program their own frequencies in. It also was based off of the `usrp_am_mw_rcv.py` example program, but was heavily modified to suit the needs of this application. Figure 5 is a snap shot of the GUI.



Figure 5: Control GUI

A full code listing with comments can be found in Appendix A. A full listing of all of the software used in the development of this program can be found in Appendix B.

V. Future Work

There are three areas that would be worth investing a little more time on.

First, it would be worth spending some time further investigating the filters used in the project to maximize their capability for airband reception. It would seem that there is probably some room for improvement in this area.

Secondly, adding a squelch feature to the receive side would also be a major improvement. It would allow for the static to be filtered out and only when there was an actual transmission would something be played over the sound card or transmitted over the FM transmitter.

Lastly, the addition of an automatic variable gain would be ideal. Some transmissions from aircraft are very strong (they are close to the antenna) and others are very weak. Therefore the gain setting cannot be tuned appropriately for all incoming traffic. An automatic gain adjustment would help with this.

VI. References

- [1] NASA CoNNeCT http://www.nasa.gov/mission_pages/station/science/experiments/CoNNeCT.html
- [2] GNU Radio Overview http://www.kd7lmo.net/ground_gnuradio.html
- [3] Airband <http://en.wikipedia.org/wiki/Airband>

[4] Cleveland Hopkins FAA Information <http://www.airnav.com/airport/KCLE>

[5] Building An Air Band Antenna <http://www.radioreference.com/forums/antennas-coax-forum/137059-building-airband-antenna-2.html>

[6] Listen to Mp3 via FM broadcast <http://lists.gnu.org/archive/html/discuss-gnuradio/2009-03/msg00297.html>

VII. Other Related Readings

[1] GNU Radio <http://www.gnu.org/software/gnuradio/>

[2] Ettus Research and USRP <http://www.ettus.com/>

Appendix A: Code Listings

repeater.py

```
#!/usr/bin/env python
#
#This program was developed by Matt Dolloff.
#It was derived from the usrp_am_mw_rcv.py script included with
#the GNU Radio Package and with help from a MP3 to FM transmitter
#found within the GNU Radio Mailing lists (thanks to Juha Vierinen)
#

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

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

    @return a subdev_spec
    """
    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("-r", "--rx-freq", type="eng_float", default=124.0e6,
            help="set rx frequency to RXFREQ", metavar="RXFREQ")
        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)")
        parser.add_option("-V", "--volume", type="eng_float", default=None,
            help="set volume (default is midpoint)")
        parser.add_option("-O", "--audio-output", type="string", default="",
            help="pcm device name. E.g., hw:0,0 or surround51 or /dev/dsp")
        parser.add_option("-T", "--tx-subdev-spec", type="subdev", default=None,
            help="select USRP Tx side A or B (default=A)")
        parser.add_option("-t", "--tx-freq", type="eng_float", default=90.0e6,
            help="set tx frequency to TXFREQ", metavar="TXFREQ")

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

# build graph

#TODO: add an AGC after the channel filter and before the AM_demod

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

adc_rate = self.usrc.adc_rate()      # 64 MS/s
usrp_decim = 250
self.usrc.set_decim_rate(usrp_decim)
usrp_rate = adc_rate / usrp_decim    # 256 kS/s
chanfilt_decim = 4
demod_rate = usrp_rate / chanfilt_decim # 64 kHz
audio_decimation = 2
audio_rate = demod_rate / audio_decimation # 32 kHz

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

self.usrc.set_mux(usrp.determine_rx_mux_value(self.usrc, options.rx_subdev_spec))
self.rx_subdev = usrp.selected_subdev(self.usrc, options.rx_subdev_spec)
print "Using RX d'board %s" % (self.rx_subdev.side_and_name(),)

self.squelch = gr.simple_squelch_cc(5e-1, 0)

chan_filt_coeffs = optfir.low_pass (1,          # gain
                                     usrp_rate, # sampling rate
                                     8e3,        # passband cutoff
                                     12e3,       # stopband cutoff
                                     1.0,        # passband ripple
                                     60)         # stopband attenuation

#print len(chan_filt_coeffs)
self.chan_filt = gr.fir_filter_ccf (chanfilt_decim, chan_filt_coeffs)
if self.use_IF:
    # Turn IF to baseband and filter.
    print "used the if statement"
    print self.IF_freq
    self.chan_filt = gr.freq_xlating_fir_filter_ccf (chanfilt_decim, chan_filt_coeffs, self.IF_freq,
usrp_rate)
else:
    print "used the else statement"
    self.chan_filt = gr.fir_filter_ccf (chanfilt_decim, chan_filt_coeffs)
self.am_demod = gr.complex_to_mag()

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

audio_filt_coeffs = optfir.low_pass (1,          # gain
                                     demod_rate, # sampling rate
                                     8e3,        # passband cutoff
                                     10e3,       # stopband cutoff
                                     0.1,        # passband ripple
                                     60)         # stopband attenuation

self.audio_filt=gr.fir_filter_fff(audio_decimation,audio_filt_coeffs)
# sound card as final sink
audio_sink = audio.sink (int (audio_rate),
                        options.audio_output,
                        False) # ok_to_block

```

```

# now wire it all together
self.connect (self.usrc,
              #self.squelch,
              self.chan_filt,
              self.am_demod,
              self.audio_filt,
              self.volume_control,
              audio_sink)

#set up the USRP as a sink too at the same rate
self.usink = usrp.sink_c()
dac_rate = self.usink.dac_rate()
usrp_interp = 400
self.usink.set_interp_rate(usrp_interp)
usrp_tx_rate = dac_rate / usrp_interp # 320 kS/s
sw_interp = 10
sink_audio_rate = usrp_tx_rate / sw_interp # 32 kS/s

#determin the daughter board
if options.tx_subdev_spec is None:
    options.tx_subdev_spec = usrp.pick_tx_subdevice(self.usink)

#setup the mux
m = usrp.determine_tx_mux_value(self.usink, options.tx_subdev_spec)

self.tx_subdev = usrp.selected_subdev(self.usink, options.tx_subdev_spec)
print "Using TX d'board %s" % (self.tx_subdev.side_and_name(),)

#set gain
self.tx_subdev.set_gain(self.tx_subdev.gain_range()[1])

#set tx freq
self.usink.tune(self.tx_subdev._which, self.tx_subdev, options.tx_freq)
self.tx_subdev.set_enable(True)

#Tx Gain
txgain = gr.multiply_const_cc(4000.0)

#fm modulation
fmtx = blks2.wfm_tx(sink_audio_rate, usrp_tx_rate,max_dev=75e3, tau=75e-6)

#wire this part up
self.connect(self.volume_control, fmtx, txgain, self.usink)

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

if options.gain is None:
    g = self.rx_subdev.gain_range()
    if True:
        # if no gain was specified, use the maximum gain available
        # (usefull for Basic_RX which is relatively deaf and the most probable board to be used for AM)
        # TODO: check db type to decide on default gain.
        options.gain = float(g[1])
    else:
        # if no gain was specified, use the mid-point in dB
        options.gain = float(g[0]+g[1])/2

if options.volume is None:
    g = self.volume_range()
    options.volume = float(g[0]*3+g[1])/4

if abs(options.rx_freq) < 1e3:
    options.rx_freq *= 1e3

# set initial values

self.set_gain(options.gain)
self.set_vol(options.volume)

```

```

if not(self.set_freq(options.rx_freq)):
    self._set_status_msg("Failed to set initial frequency")

def _set_status_msg(self, msg, which=0):
    self.frame.GetStatusBar().SetStatusText(msg, which)

def _build_gui(self, vbox, usrp_rate, demod_rate, audio_rate):

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

    # preset values
    self.panell = wx.Panel(self.panel, -1, (-1,-1))
    self.hbox1 = wx.BoxSizer(wx.HORIZONTAL)

    # CLE #####
    self.cleSB = wx.StaticBox(self.panell, -1, "Cleveland Hopkins")
    self.cleBS = wx.StaticBoxSizer(self.cleSB, wx.VERTICAL)

    self.cleGS = wx.GridSizer(4,2,4,4)
    self.cleST6L = wx.StaticText(self.panell, -1, "Runway 6L/24R")
    self.cleST6R = wx.StaticText(self.panell, -1, "Runway 6R/24L")

    self.cleBtnAppr6R = wx.Button(self.panell, constants.ID_CLE_APPR_06R,
                                   "Approach")
    self.cleBtnGrnd6R = wx.Button(self.panell, constants.ID_CLE_GROUND_06R,
                                   "Ground")
    self.cleBtnTower6R = wx.Button(self.panell, constants.ID_CLE_TOWER_06R,
                                   "Tower")
    self.cleBtnAppr6L = wx.Button(self.panell, constants.ID_CLE_APPR_06L,
                                   "Approach")
    self.cleBtnGrnd6L = wx.Button(self.panell, constants.ID_CLE_GROUND_06L,
                                   "Ground")
    self.cleBtnTower6L = wx.Button(self.panell, constants.ID_CLE_TOWER_06L,
                                   "Tower")

    self.cleGS.AddMany([(self.cleST6R,-1,wx.ALIGN_CENTER_HORIZONTAL |
                          wx.ALIGN_BOTTOM),
                        (self.cleST6L,-1,wx.ALIGN_CENTER_HORIZONTAL |
                          wx.ALIGN_BOTTOM),
                        (self.cleBtnTower6R,-1,wx.ALIGN_CENTER_HORIZONTAL |
                          wx.ALIGN_BOTTOM),
                        (self.cleBtnTower6L,-1,wx.ALIGN_CENTER_HORIZONTAL |
                          wx.ALIGN_BOTTOM),
                        (self.cleBtnAppr6R,-1,wx.ALIGN_CENTER_HORIZONTAL |
                          wx.ALIGN_BOTTOM),
                        (self.cleBtnAppr6L,-1,wx.ALIGN_CENTER_HORIZONTAL |
                          wx.ALIGN_BOTTOM),
                        (self.cleBtnGrnd6R,-1,wx.ALIGN_CENTER_HORIZONTAL |
                          wx.ALIGN_BOTTOM),
                        (self.cleBtnGrnd6L,-1,wx.ALIGN_CENTER_HORIZONTAL |
                          wx.ALIGN_BOTTOM)])

    self.cleBS.Add(self.cleGS)
    self.hbox1.Add(self.cleBS, -1, wx.ALL, 10 )
    # end of CLE #####

    #vbox for right side of gui
    self.vboxRight = wx.BoxSizer(wx.VERTICAL)

    # Burke #####
    self.bklSB = wx.StaticBox(self.panell, -1, "Burke Lakefront",size=wx.Size(125,50))
    self.bklBS = wx.StaticBoxSizer(self.bklSB, wx.VERTICAL)

    self.bklGS = wx.GridSizer(2,1,4,4)
    self.bklButtonGround = wx.Button(self.panell,
                                       constants.ID_BURKE_GROUND,
                                       "Ground")
    self.bklButtonTower = wx.Button(self.panell,
                                       constants.ID_BURKE_TOWER,

```

```

        "Tower")
self.bklGS.AddMany([(self.bklButtonGround,-1,wx.ALIGN_CENTER),
                    (self.bklButtonTower,-1,wx.ALIGN_CENTER)])
self.bklBS.Add(self.bklGS, -1, wx.ALIGN_CENTER | wx.EXPAND)
self.vboxRight.Add(self.bklBS, -1, wx.TOP, 10)
# end of Burke #####

# radio stations #####
self.radioSB = wx.StaticBox(self.panell, -1, "AM Radio Sations")
self.radioBS = wx.StaticBoxSizer(self.radioSB, wx.VERTICAL)

self.radioGS = wx.GridSizer(2,2,4,4)
self.radioST850 = wx.StaticText(self.panell, -1, "Sports Radio")
self.radioButton850 = wx.Button(self.panell,
                                constants.ID_AM_850,
                                "WKNR")
self.radioST1420 = wx.StaticText(self.panell, -1, "Talk Radio")
self.radioButton1420 = wx.Button(self.panell,
                                constants.ID_AM_1420,
                                "WTAM")
self.radioGS.AddMany([(self.radioST850,-1,wx.ALIGN_CENTER),
                      (self.radioButton850,-1,wx.ALIGN_CENTER),
                      (self.radioST1420,-1,wx.ALIGN_CENTER),
                      (self.radioButton1420,-1,wx.ALIGN_CENTER)])
self.radioBS.Add(self.radioGS)
self.vboxRight.Add(self.radioBS, -1, wx.TOP, 10)
# end of radio #####

#add right pane vbox
self.hbox1.Add(self.vboxRight)

#add CSU logo
self.hbox1.Add((75,0),0)
csuBitmap = wx.Image('csu_150.png', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
logo = wx.StaticBitmap(self.panell, -1, csuBitmap)
self.hbox1.Add(logo, 0, wx.CENTER , border=5)

#add presets panel
self.panell1.SetSizer(self.hbox1)

#add presets box
vbox.Add(self.panell1)

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

hbox = wx.BoxSizer(wx.HORIZONTAL)
hbox.Add((5,0), 0)
myform['freq'] = form.float_field(
    parent=self.panel, sizer=hbox, label="Freq", weight=1,
    callback=myform.check_input_and_call(_form_set_freq, self._set_status_msg))

hbox.Add((5,0), 0)
myform['freq_slider'] = \
    form.quantized_slider_field(parent=self.panel, sizer=hbox, weight=3,
                               range=(520.0e3, 1611.0e3, 1.0e3),
                               callback=self.set_freq)

hbox.Add((5,0), 0)
vbox.Add(hbox, 0, wx.EXPAND)

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

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

hbox.Add((5,0), 1)

myform['gain'] = \
    form.quantized_slider_field(parent=self.panel, sizer=hbox, label="Gain",

```

```

        weight=3, range=self.rx_subdev.gain_range(),
        callback=self.set_gain)

hbox.Add((5,0), 0)
vbox.Add(hbox, 0, wx.EXPAND)

# Setup the button bindings #####
self.cleBtnAppr6L.Bind(wx.EVT_BUTTON, self.onPreButton)
self.cleBtnAppr6R.Bind(wx.EVT_BUTTON, self.onPreButton)
self.cleBtnGrnd6L.Bind(wx.EVT_BUTTON, self.onPreButton)
self.cleBtnGrnd6R.Bind(wx.EVT_BUTTON, self.onPreButton)
self.cleBtnTower6L.Bind(wx.EVT_BUTTON, self.onPreButton)
self.cleBtnTower6R.Bind(wx.EVT_BUTTON, self.onPreButton)
self.bklButtonGround.Bind(wx.EVT_BUTTON, self.onPreButton)
self.bklButtonTower.Bind(wx.EVT_BUTTON, self.onPreButton)
self.radioButton1420.Bind(wx.EVT_BUTTON, self.onPreButton)
self.radioButton850.Bind(wx.EVT_BUTTON, self.onPreButton)
# End of bindings #####

def on_button (self, event):
    if event.value == 0:          # button up
        return
    self.rot = 0
    if self.state == "FREQ":
        self.state = "VOL"
    else:
        self.state = "FREQ"
    self.update_status_bar ()

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

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

    @param target_freq: frequency in Hz
    @rtype: bool

    Tuning is a two step process.  First we ask the front-end to
    tune as close to the desired frequency as it can.  Then we use
    the result of that operation and our target_frequency to
    determine the value for the digital down converter.
    """
    r = usrp.tune(self.usrc, 0, self.rx_subdev, target_freq + self.IF_freq)
    #TODO: check if db is inverting the spectrum or not to decide if we should do + self.IF_freq or -
    self.IF_freq

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

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

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

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

```

```

        self.src_fft.set_baseband_freq(self.rx_freq)
    except:
        None

def volume_range(self):
    return (-40.0, 0.0, 0.5)

def onPreButton(self, event):
    id = event.GetId()
    if id == constants.ID_CLE_APPR_06R:
        self.set_freq(constants.F_CLE_APPR_06R)
    elif id == constants.ID_CLE_APPR_06L:
        self.set_freq(constants.F_CLE_APPR_06L)
    elif id == constants.ID_CLE_TOWER_06R:
        self.set_freq(constants.F_CLE_TOWER_06R)
    elif id == constants.ID_CLE_TOWER_06L:
        self.set_freq(constants.F_CLE_TOWER_06L)
    elif id == constants.ID_CLE_GROUND_06R:
        self.set_freq(constants.F_CLE_GROUND_06R)
    elif id == constants.ID_CLE_GROUND_06L:
        self.set_freq(constants.F_CLE_GROUND_06L)
    elif id == constants.ID_BURKE_GROUND:
        self.set_freq(constants.F_BURKE_GROUND)
    elif id == constants.ID_BURKE_TOWER:
        self.set_freq(constants.F_BURKE_TOWER)
    elif id == constants.ID_AM_1420:
        self.set_freq(constants.F_AM_1420)
    elif id == constants.ID_AM_850:
        self.set_freq(constants.F_AM_850)
    return

if __name__ == '__main__':
    app = stdgui2.stdapp (wfm_rx_block, "USRP AM to FM Repeater")
    app.MainLoop ()

```

constants.py

```

#This file contains all the constants
#used in creating the GUI for repeater.py
#AM Radio Stations
F_AM_850=850e3
F_AM_1420=1420e3
ID_AM_850=1100
ID_AM_1420=1101

#Burke Lakefront
F_BURKE_GROUND=121.9e6
F_BURKE_TOWER=124.3e6
ID_BURKE_GROUND=1003
ID_BURKE_TOWER=1000

#Cleveland Hopkins
F_CLE_GROUND_06R=121.7e6
F_CLE_GROUND_06L=133.6e6
F_CLE_TOWER_ALL=257.8e6
F_CLE_TOWER_06R=120.9e6
F_CLE_TOWER_06L=124.5e6
F_CLE_APPR_06R=124.0e6
F_CLE_APPR_06L=119.625e6
F_CLE_B_CLEARANCE_340_200=125.35e6
F_CLE_B_CLEARANCE_201_339=126.35e6
ID_CLE_GROUND_06R=1004
ID_CLE_GROUND_06L=1005
ID_CLE_TOWER_ALL=1006
ID_CLE_TOWER_06R=1007
ID_CLE_TOWER_06L=1008
ID_CLE_APPR_06R=1009
ID_CLE_APPR_06L=1010
ID_CLE_B_CLEARANCE_340_200=1011
ID_CLE_B_CLEARANCE_201_339=1012

```

Appendix B: Software Used, Version, Purpose

Software Package	Version	Use
Eclipse	3.4.2	Python code development
PyDev	1.4.5	Plug-in for Eclipse to enable Python development
Python	2.5.4	Scripting language used to access GNU Radio Libraries
GNU Radio	3.1.3	Libraries used to interface with the USRP and do signal processing
Sox	14.2.0	Used in getting MP3 to FM player working
Ubuntu Linux	8.04	Main OS used for development