Final Project Report

Modulate of Internet Radio Into FM Using GNU Radio

Department of Electrical and Computer Engineering
Cleveland State University

Mobile Computing Class
By Elie Salameh

I-Introduction:

Internet radio (also known as Webcasting) is becoming more and more popular. Especially that it covers a wide range of choices for users (specific kind of music, news, commerce…), it is also very convenient for people traveling all the time or living away from there home country, internet radio helps them keep in touch with there favorite radio stations. Technology is growing fast and so is the availability of Internet connections. Nowadays we can access the Internet from any location around the world (if we had the right technology) and thus in our cars. So why don’t we use this Internet connection to listen to our favorite Internet radio station from around the world while we are driving our cars or even when walking around our houses.

In this project, we use GNU radio along with a USRP device to try modulating Internet radio into FM.

II-Background:

What is GNU radio?

Briefly GNU radio is a free software radio tool. In other words it is a tool that help us implement a radio (transmitter or receiver) in software rather than hardware. It helps us get rid of as much hardware as possible eventually bringing the software as close as possible to the antenna. It uses Python as programming language for its simplicity and flexibility. GNU radio is mostly associated with the USRP device (universal software radio peripheral), which is the hardware end of the software radio. USRP has a wide range of frequency for wireless transmission and reception or even wired communication.

What is FM modulation?

FM modulation is technique used to transmit information over electromagnetic waves. It uses frequency variations. It is used over a wide range of today’s technology (FM radio stations, some TV signals, VCR…)

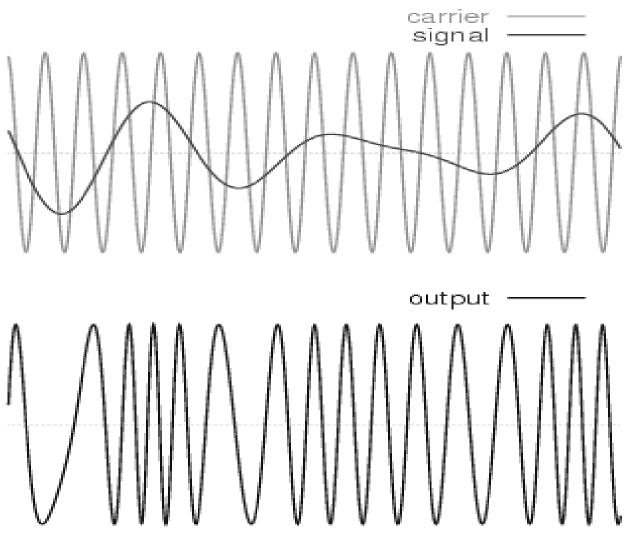


Figure 1-FM modulation.

III-Topic

Project Goal

Given the availability of the GNU radio and the USRP at our mobile computing lab. I decided to modulate the Internet radio signal into an FM signal and transmit it wirelessly and receive it on a radio device.

Experiment Setup

To run this experiment the following was done:

- Install and configure a Linux based operating system on a machine. (cygwin might work too but I do not recommend it, a lot of patches are needed and sometimes it acts up). In this case Ubuntu was installed and configured.
- Install and configure GNU radio on Ubuntu. The instructions on how to do that are available in reference [5].
- Setup the USRP device: connect the daughter boards needed for the experiment in this case Basic Tx was mounted on the USRP. The choice of daughter boards is mainly based on the type of project one is doing and on the frequency that one is using. In this case, an FM transmission is made on a frequency range from 87.0 to 107.9 MHZ. Basic Tx will work fine in this project. Information on Basic Tx can be found in reference [6]

These are mainly the three steps needed to setup this experiment. Now we can start working on the project.

In a few words we can describe the project as follows:

1- Using GNU radio to modulate audio signal into FM
2- Importing internet radio stream into GNU radio
3- Use the USRP to transmit the FM signal generated by GNU radio

I-using GNU radio to modulate audio signal into FM:

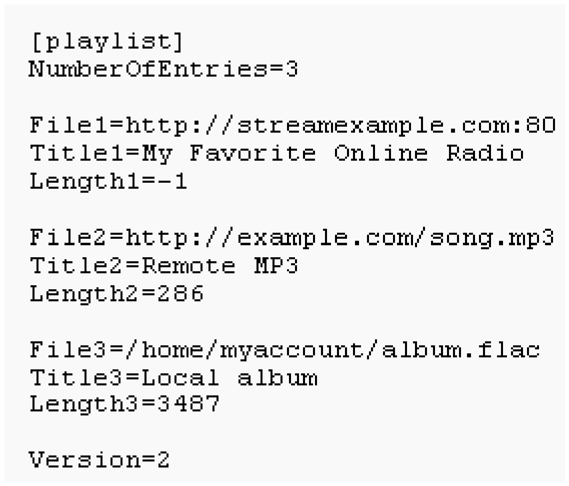
After a lot of research I found out that GNU radio accept audio files in the ".raw" format. Using fm_tx4.py as a skeleton code I started working on how to modulate audio signal into FM. Luckily GNU radio provides a lot of blocks for signal processing and one of the useful blocks is wfm_tx() [7]; which stands for wide band FM transmitter. This block was used to modulate the signal to FM.

II- Importing Internet radio stream into GNU radio

The biggest challenge in this project was to import the Internet audio stream into GNU radio. After a lot of research I found out that Internet radio are not uniform in a sense that every station has its own audio format. Some of the types of formats used are: mp3, OGG Vorbis, windows media audio, real audio…

The plan was to record the audio stream from the internet to a mp3 file than load that file into GNU radio. Another challenge occurred which is how to transform the mp3 file into a .raw audio file. I found out that this could be done using the Linux command "sox": Sox- Sound eXchange: universal sound sample translator [8]. Sox can translate almost all kind of audio files to any format needed. It takes a lot of arguments; some of them are: input file, output file, extension of output file, sampling rate, size of the sample….
This command can be used as follows: *sox test.mp3 -r 32000 -t raw -l -c 1 test.raw*

When trying to figure out how to record the internet radio into an mp3 file, I found out a play list format which is ".pls". This format can take as arguments Internet URL's in addition to local addresses of mp3 files. I modified the code to be able to read mp3 files from the play list. But when I tried to include url's in that file I got an error that says that the sox command cannot handle such kind of files. So now only local mo3 files could be imported to GNU radio.



```
[playlist]
NumberOfEntries=3

File1=http://streamexample.com:80
Title1=My Favorite Online Radio
Length1=-1

File2=http://example.com/song.mp3
Title2=Remote MP3
Length2=286

File3=/home/myaccount/album.flac
Title3=Local album
Length3=3487

Version=2
```

Figure 2- .pls format [1]

One more solution was there and it was recording the Internet radio into mp3 files. A lot of software's are available to purchase online. Most of them record anything that is played on the sound card. Also I found a freeware called opD2d [9], which does the same thing, but unfortunately it didn't work; somehow the recorded mp3 file is corrupted and could not be played.

III- Use the USRP to transmit the FM signal generated by GNU radio

After the code is finalized the FM signal was transmitted using the USRP on which we have mounted the basic TX daughter board to which an antenna was attached.

Results

The results of this experiment were pretty convincing since I was able to transmit an mp3 file and receive it on radio device.

Discussion

Future work might be done in a sense that to find a way to import directly the Internet radio audio stream into GNU radio.

GNU radio is a bit complicated to use since it has a lot of features, but once you get comfortable with python it will be fun to try all the different features that can be used in it.

The commented python code can be found in appendix A.

This code could be run using the following command:
Sudo python "filename.py" –l "playlistfilename.pls"
Note that the .pls file should contain the location the mp3 file in the format shown in figure 2.

IV-References:

[1]-www.wikipedia.org
[2]-http://www.ettus.com/
[3]-http://academic.csuohio.edu/yuc/mobile08/08_week04_USRP.pdf
[4]-http://www.gnu.org/software/gnuradio/index.html

[5]- http://www.gnuradio.org/trac/wiki/UbuntuInstall
[6]- http://www.ettus.com/Download.html
[7]- http://www.gnuradio.org/trac/browser/gnuradio/trunk/gnuradio-core/src/python/gnuradio/blksimpl/wfm_tx.py?rev=3534
[8]- http://sox.sourceforge.net/
[9]- http://www.opcode.co.uk/opd2d/default.asp

Appendix A:

```
#!/usr/bin/env python2.4


from gnuradio import gr, eng_notation
from gnuradio import usrp
from gnuradio import audio
from gnuradio import blks
from gnuradio.eng_option import eng_option
from optparse import OptionParser
from usrpm import usrp_dbid

import math, re, sys, thread, time, tempfile, os, random

# this function sends a command to Linux that will
# use sox to transform mp3 to raw
def mp3toraw(filename,outputfile):
    print("nice -n 19 sox \"%s\" -r 32000 -t raw -f -l -c 1 %s\n" %
(filename,outputfile))
    os.system("nice -n 19 sox \"%s\" -r 32000 -t raw -f -l -c 1 %s" %
(filename,outputfile))

# this function will read the location of the mp3 files out of the
#.pls file
def read_playlist(fname):
    input = open(fname, 'r')
    playlist=[]
    l = input.readline()
    # NumberOfEntries
    l = input.readline()
    nentries = int(re.findall("NumberOfEntries=([0-9]+)",l)[0])

    print "Number of items in list %d\n" % nentries
    i = 1
    while l:
        l=input.readline()

        filepath = re.findall("File[0-9]+=(.*)$",l)
        if filepath:
            print filepath[0]
            playlist.append(filepath[0])
            i = i + 1

    input.close()
    return(playlist)
#this function will create a temprary .raw file that will used by the
sox command
def mktempfn():
    tf = tempfile.mkstemp(".raw")
    outputfile = tf[1]
    os.close(tf[0])
    os.remove(tf[1])
    return(outputfile)


# this code is used for mudulation
# it takes options from the command line
# most of this code is take from fm_tx4.py
class wfm_tx:
    def __init__(self):

        parser = OptionParser (option_class=eng_option)
        parser.add_option("-T", "--tx-subdev-spec", type="subdev", 
default=None,
                          help="select USRP Tx side A or B")
        parser.add_option("-f", "--freq", type="eng_float", 
default=90.1e6,
                           help="set Tx frequency to FREQ (default 
90.1e6)", metavar="FREQ")

        parser.add_option("-l","--playlist", action="store", 
default=None,
                          help="MP3 playlist containing files to air.")

        parser.add_option("","--debug", action="store_true", 
default=False,
                          help="Launch Tx debugger")
        (options, args) = parser.parse_args ()

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

        if options.playlist == None:
            print "No playlist specified\n"
            sys.exit()

        # parse playlist
        playlist = read_playlist(options.playlist)


        # setup IQ rate to 320kS/s and audio rate to 32kS/s
        self.u = usrp.sink_c()
        self.dac_rate = self.u.dac_rate()                    # 128 MS/s
        self.usrp_interp = 400
        self.u.set_interp_rate(self.usrp_interp)
        self.usrp_rate = self.dac_rate / self.usrp_interp    # 320 kS/s
        self.sw_interp = 10
        self.audio_rate = self.usrp_rate / self.sw_interp    # 32 kS/s

        # determine the daughterboard subdevice we're using
        if options.tx_subdev_spec is None:
            options.tx_subdev_spec = usrp.pick_tx_subdevice(self.u)

        m = usrp.determine_tx_mux_value(self.u, options.tx_subdev_spec)
        self.u.set_mux(m)
        self.subdev = usrp.selected_subdev(self.u,
options.tx_subdev_spec)
        print "Using TX d'board %s" % (self.subdev.side_and_name(),)

        self.subdev.set_gain(self.subdev.gain_range()[1])    # set max
Tx gain

        if not self.set_freq(options.freq):
            freq_range = self.subdev.freq_range()
            print "Failed to set frequency to %s.  Daughterboard
supports %s to %s" % (
                eng_notation.num_to_str(options.freq),
                eng_notation.num_to_str(freq_range[0]),
                eng_notation.num_to_str(freq_range[1]))
            raise SystemExit
        self.subdev.set_enable(True)                         # enable
transmitter
        print "TX freq %1.2f MHz\n" % (options.freq/1e6)

        gain = gr.multiply_const_cc(4000.0)


        i = 0

        while 1:
            self.fg = gr.flow_graph()
            outputfile = mktempfn()
            # write raw sound to named pipe in background
            thread.start_new_thread(mp3toraw,(playlist[i],outputfile))
            # wait untill the conversion of mp3 to raw is completed
            time.sleep(3)

            print "File size %d\n" % int(os.stat(outputfile)[6])
            #specify the source file
            src = gr.file_source(gr.sizeof_float, outputfile, False)
            #configure the modulation block
            fmtx = blks.wfm_tx(self.fg, self.audio_rate,
self.usrp_rate,max_dev=75e3, tau=75e-6)

            # connect blocks
            self.fg.connect(src, fmtx, gain, self.u)

            print "starting to transmit\n"
            self.fg.run()
            print "done..."
            # remove the temprary file
            os.remove(outputfile)
            # stop the sox command if it is still working
            os.system("killall sox")



    def set_freq(self, target_freq):
        """
        Set the center frequency we're interested in.
        """
        r = self.u.tune(self.subdev._which, self.subdev, target_freq)
        if r:
            print "r.baseband_freq =",
eng_notation.num_to_str(r.baseband_freq)
            print "r.dxc_freq      =",
eng_notation.num_to_str(r.dxc_freq)
            print "r.residual_freq =",
eng_notation.num_to_str(r.residual_freq)
            print "r.inverted      =", r.inverted
            return True
        return False

if __name__ == '__main__':
    wfm_tx()
```