GNU Radio

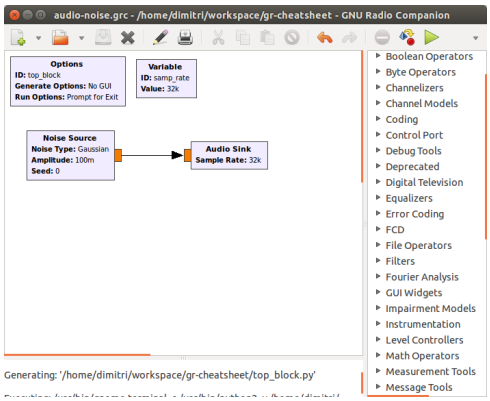
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

1 Installation

gnuradio–install.sh

```
# Install pybombs via python-pip
sudo pip install PyBOMBS

# Add recipe lists from git repositories
pybombs recipes add gr-recipes git+https://github.com/
    ↪ gnuradio/gr-recipes.git
pybombs recipes add gr-etcetera git+https://github.com/
    ↪ gnuradio/gr-etcetera.git

# Set installation folder to '~/Desktop/pybombs'
pybombs prefix init ~/Desktop/pybombs -a myprefix

# Enable documentation
pybombs config builddocs=ON

# Run gnuradio installation with verbose output
pybombs -vv install gnuradio

# Publish install variables as environment variables
source ~/Desktop/pybombs/setup_env.sh

# Apply also after re-booting
echo 'source ~/Desktop/pybombs/setup_env.sh' >> ~/.
    ↪ profile
echo 'source ~/Desktop/pybombs/setup_env.sh' >> ~/.bashrc

# Run GNU Radio Companion
gnuradio-companion
```

2 Getting Started



top_block.py

```
from gnuradio import analog
from gnuradio import audio
from gnuradio import eng_notation
from gnuradio import gr
from optparse import OptionParser


class top_block(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "Top Block")

        samp_rate = 32000
        self.audio = audio.sink(samp_rate, '', True)
        self.noise = analog.noise_source_f(analog.
    ↪ GR_GAUSSIAN, 0.1, 0)

        self.connect(self.noise, self.audio)

def main(top_block_cls=top_block, options=None):

    tb = top_block_cls()
    tb.start()
    try:
        raw_input('Press Enter to quit: ')
    except EOFError:
        pass
    tb.stop()
    tb.wait()


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

3 Gnu Radio Basics

3.1 Create Hierarchical Block

inputLayer.py

```
from gnuradio import blocks
from gnuradio import fft
from gnuradio import gr
from gnuradio.fft import window

class inputLayer(gr.hier_block2):
    def __init__(self, vlen=64):
        gr.hier_block2.__init__(
            self, "Input Layer",
            gr.io_signature(
                1, 1,
                gr.sizeof_gr_complex*vlen
            ),
            gr.io_signature(
                1, 1,
                gr.sizeof_float*vlen
            ),
        )

        # Blocks
        fft = fft.fft_vcc(
            vlen, True,
            (window.rectangular(vlen)),
            True, 8
        )
        log = blocks.nlog10_ff(20, vlen, 0)
        mag = blocks.complex_to_mag(vlen)
        mult = blocks.multiply_const_vcc(
            ([1./float(vlen), ]*vlen)
        )

        # Connections
        self.connect(self, mult)
        self.connect(mult, fft)
        self.connect(fft, mag)
        self.connect(mag, log)
        self.connect(log, self)
```

3.2 Create Python Block

vector_sum_vff.py

```
import numpy
from gnuradio import gr

class vector_sum_vff(gr.sync_block):
    def __init__(self, vlen):
        self.vlen = vlen
        gr.sync_block.__init__(self,
            name="vector_sum_vff",
            in_sig=[(numpy.float32, vlen)],
            out_sig=[(numpy.float32, 1)])

    def work(self,input_items,output_items):
        in0 = input_items[0]
        out = output_items[0]
        out[:] = numpy.sum(in0[0:1], axis=1)
        return 1
```

3.3 Post-Processing

read_binary_file.m

```
% Open recorded cfile
f = fopen ('filename.cfile', 'rb');

% Activate recorded data type
%type = 'int'; % For int values
%type = 'char'; % For char values
%type = 'short'; % For cshort values
type = 'float'; % For float/complex values

% Read
v = fread (f, Inf, type);

% Activate for complex data type:
%v = v(1:2:end)+v(2:2:end)*j;

% Close cfile
fclose (f);

% Plot values
plot(v)
```