MENU

Capture and decode FM radio

Fraida Fund
10 FEBRUARY 2016 on education, software defined radio, wireless

This experiment is meant to teach the basics of FM signal processing.

It should take about 60-120 minutes to run depending on your familiarity with the reference material, but you will need to have reserved that time in advance. This experiment uses wireless resources, and you can only use wireless resources on GENI during a reservation.

To reproduce this experiment on GENI, you will need an account on the GENI Portal, and you will need to have joined a project. You should have already uploaded your SSH keys to the portal. The project lead of the project you belong to must have enabled wireless for the project. Finally, you must have reserved time on a wireless testbed that has RTL software defined radio devices. These instructions specifically use RTL-SDR devices on the "grid" testbed at ORBIT.

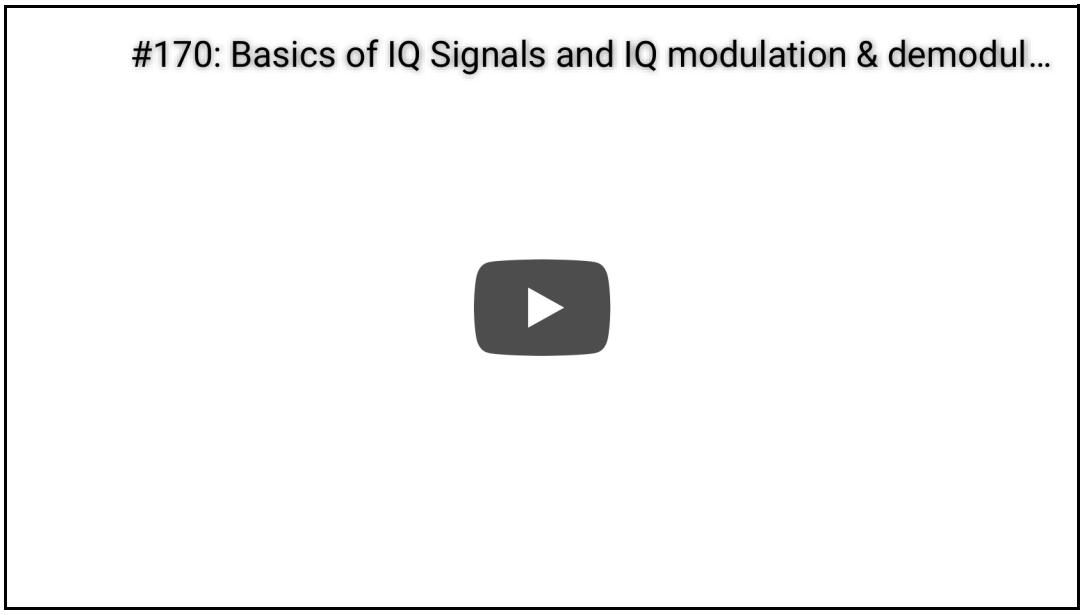
- Skip to Results
- Skip to Run my experiment

Background

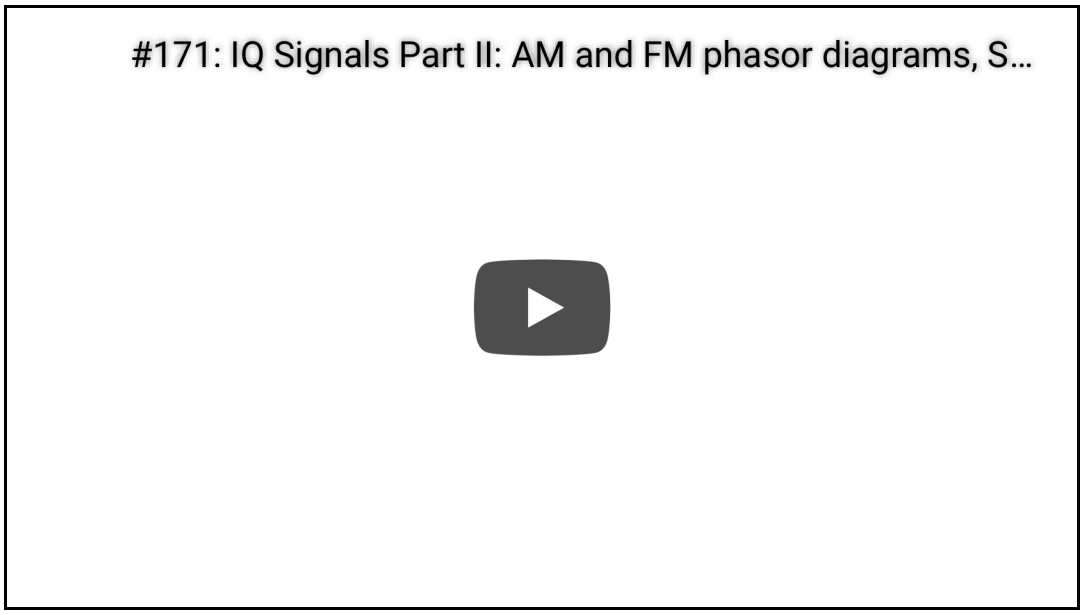
A software defined radio captures IQ samples and passes them to a host computer for further processing in software. In this

experiment, we will capture a slice of spectrum that includes an FM radio transmission, then demodulate that signal and turn it into an audio file.

For more information about IQ signals, and IQ signal modulation and demodulation, watch the following video:



And, here's a video that shows what FM signals in particular look like as IQ samples:



Generally, we know that an FM signal modulates the message signal

$$m(t)$$

on the carrier signal so that the derivative of the phase deviation,

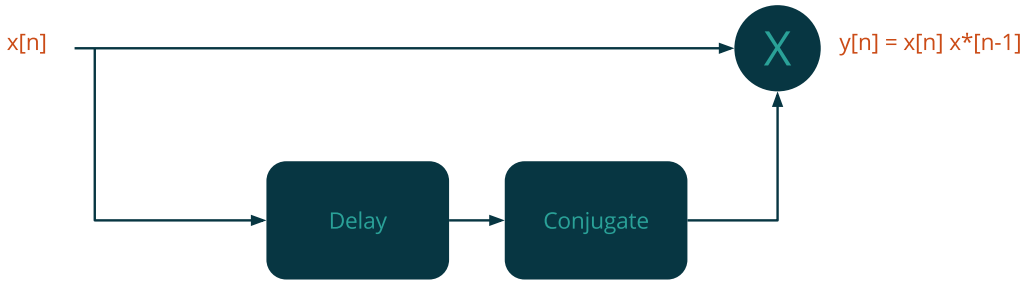
$$\frac{d\phi(t)}{dt}$$

is proportional to the message:

$$x_{FM}(t) = A_c \cos\Big(2\pi f_c t + \phi(t)\Big) = A_c \cos\Big(2\pi f_c t + 2\pi f_\Delta \int_0^t m($$

So recovering the message from the FM signal should be a simple matter of calculating the rate of change of the phase of the

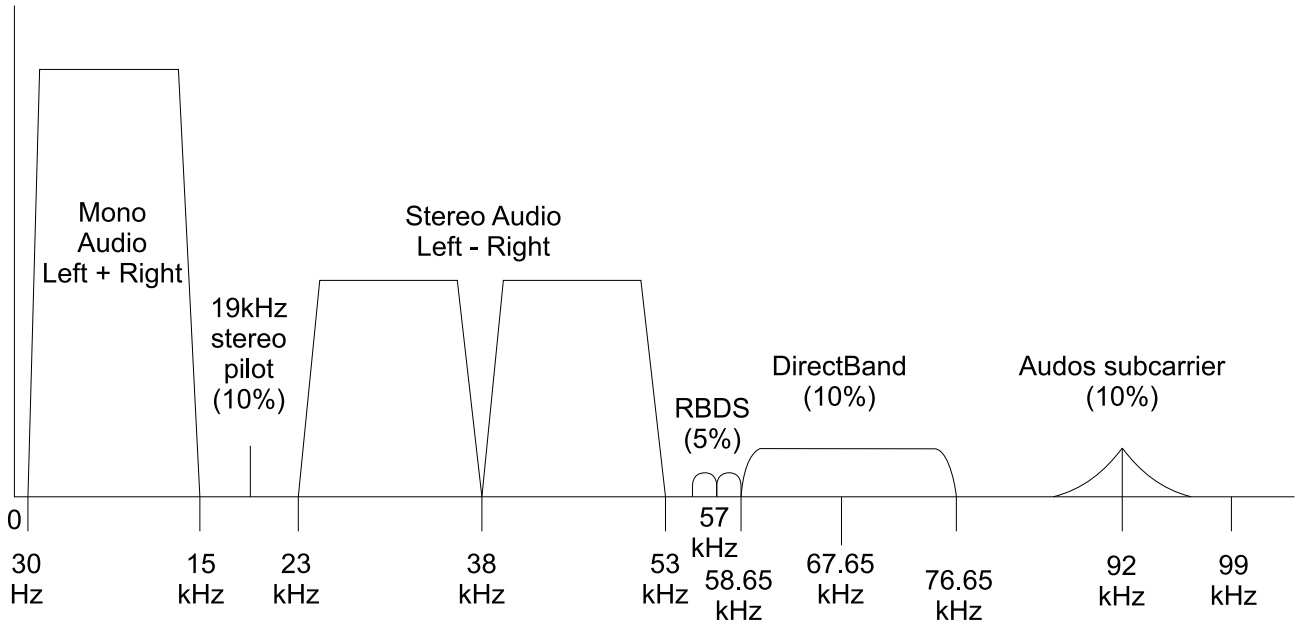
received signal. For that stage, we'll use a kind of frequency discriminator called a polar discriminator. A polar discriminator measures the phase difference between consecutive samples of a complex-sampled FM signal. More specifically, it takes successive complex-valued samples and multiplies the new sample by the conjugate of the old sample. Then it takes the angle of this complex value. This turns out to be the instantaneous frequency of the sampled FM signal.



In practice, things are a little more complicated. First of all, the captured IQ data we will be working with was sampled at a rate of 1140000 Hz, at a center frequency offset from the signal of interest by 250000 Hz (this helps avoid DC offset problems.) So we will need to move that radio channel down to baseband (center it at 0 Hz) and then filter and decimate it to focus on just the FM broadcast signal (which has a 200 kHz bandwidth.) That 200 kHz baseband signal is what we will pass to the frequency discriminator.

FM radio broadcast signals contain several sub-signals: mono audio, stereo audio, digital data, and more. After the frequency

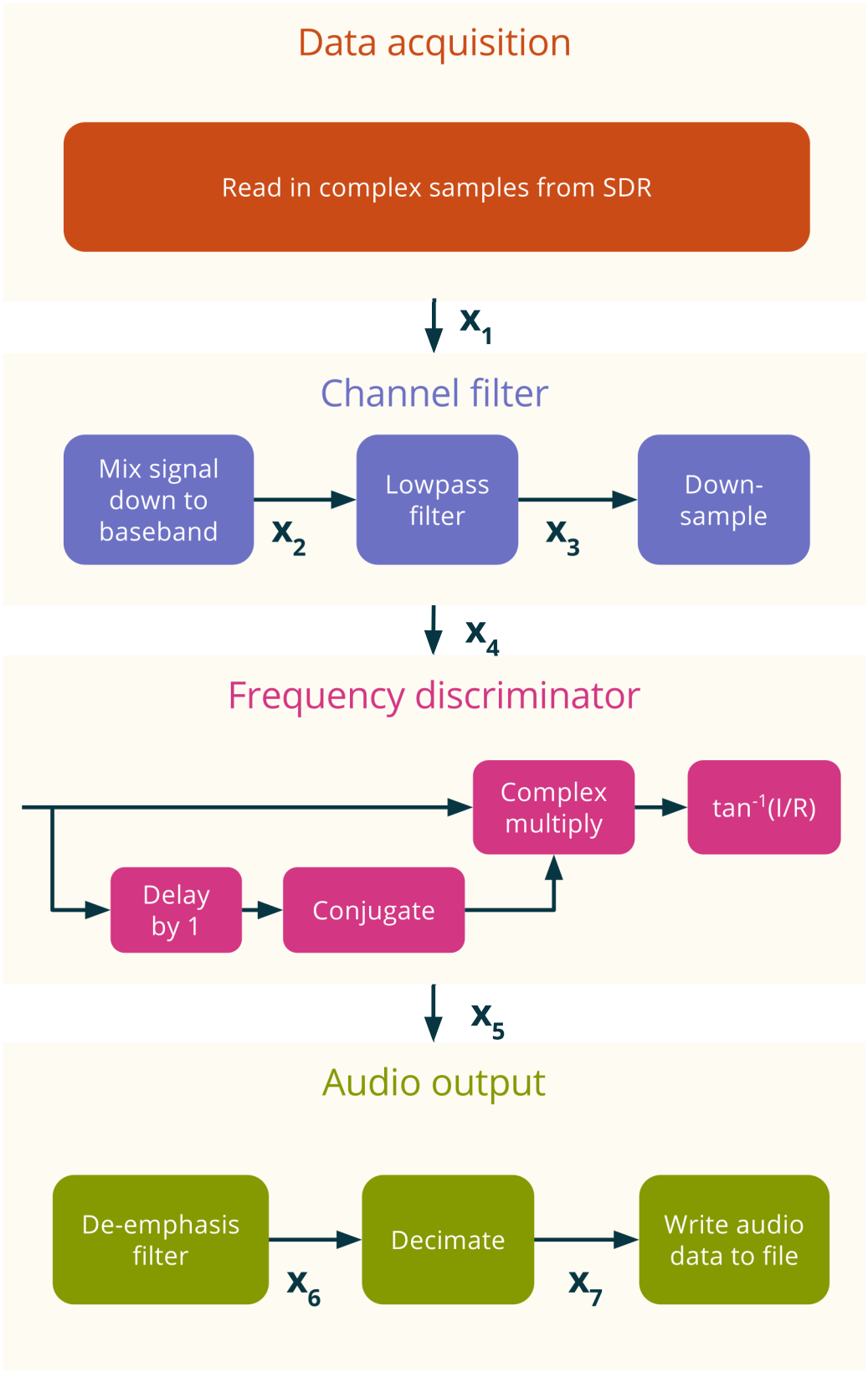
discrimination stage, we should be able to create an image of our 200 kHz broadcast signal just like the one below, and identify which carriers are present in your signal.



Typical baseband spectrum of an FM broadcast signal. Image from Wikimedia Commons, is in the public domain.

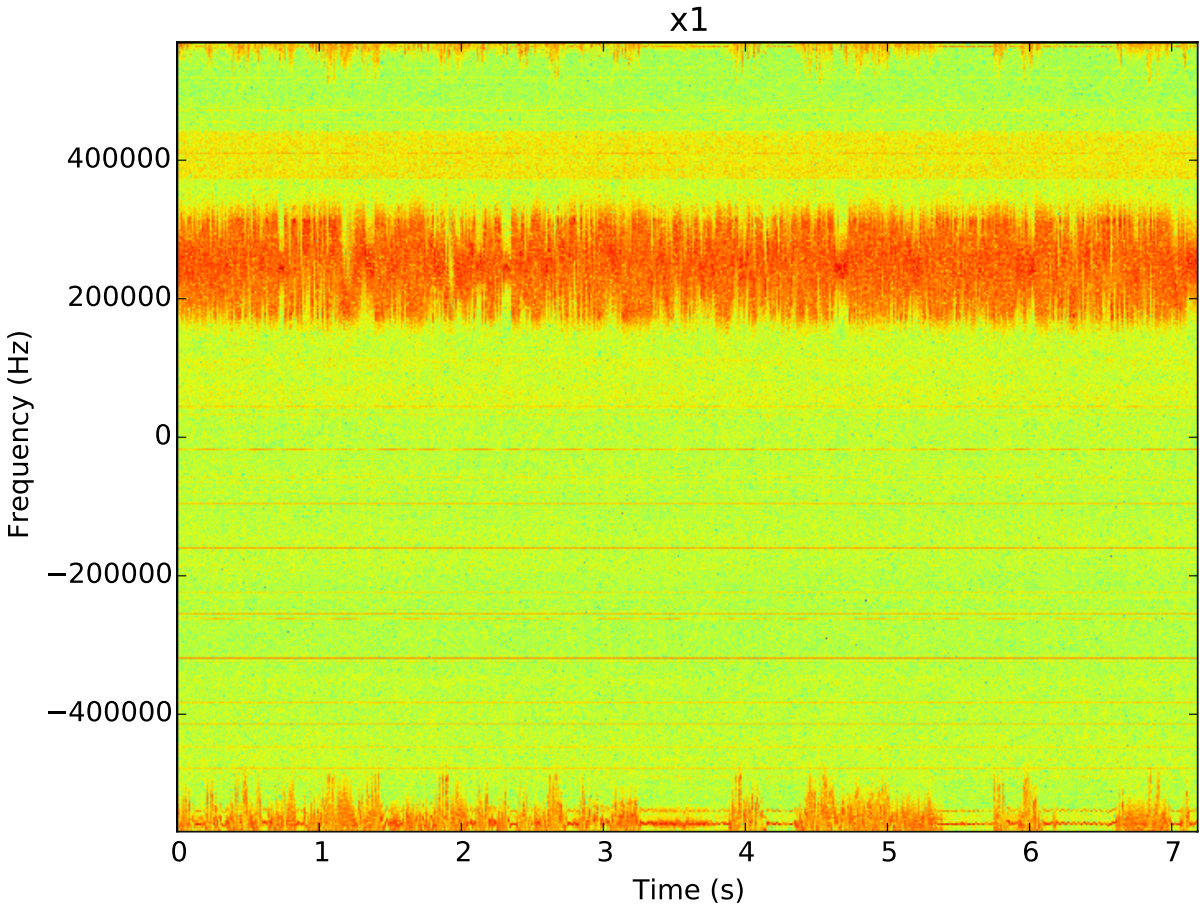
We're only going to work with the mono audio channel. we need to pass it through a de-emphasis filter (this is to compensate for an analogous emphasis filter That was applied to the data at the transmitter.) we'll also need to decimate the signal down to the standard audio sampling rate (roughly 44.1-48 kHz). Finally, we will be able to write our audio data to a file and play it back.

Here is a block diagram of the decoder implementation we are going to create:

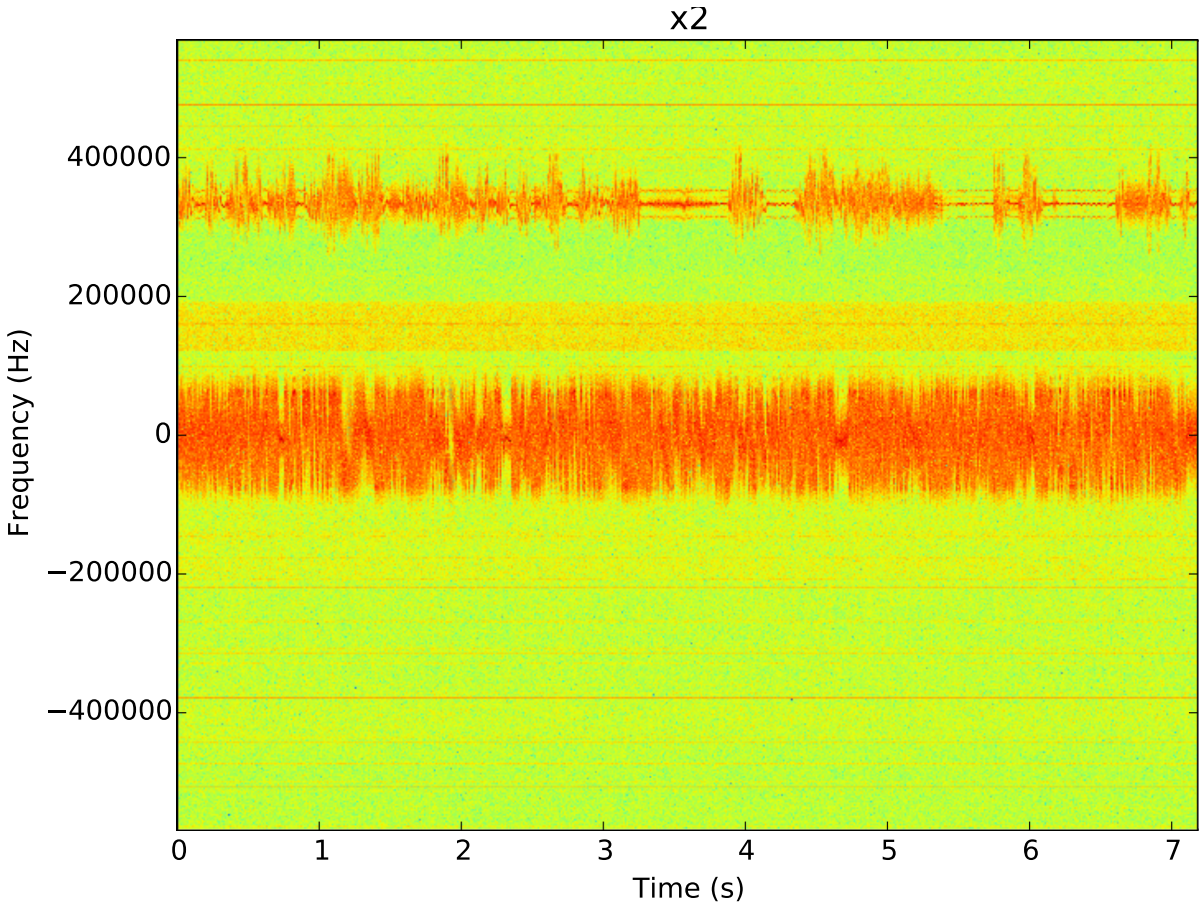

Data acquisition
Read in complex samples from SDR
x1
Channel filter
Mix signal down to baseband
x2
Lowpass filter
x3
Down-sample
x4
Frequency discriminator
Complex multiply
tan-1(I/R)
Delay by 1
Conjugate
x5
Audio output
De-emphasis filter
x6
Decimate
x7
Write audio data to file

Results

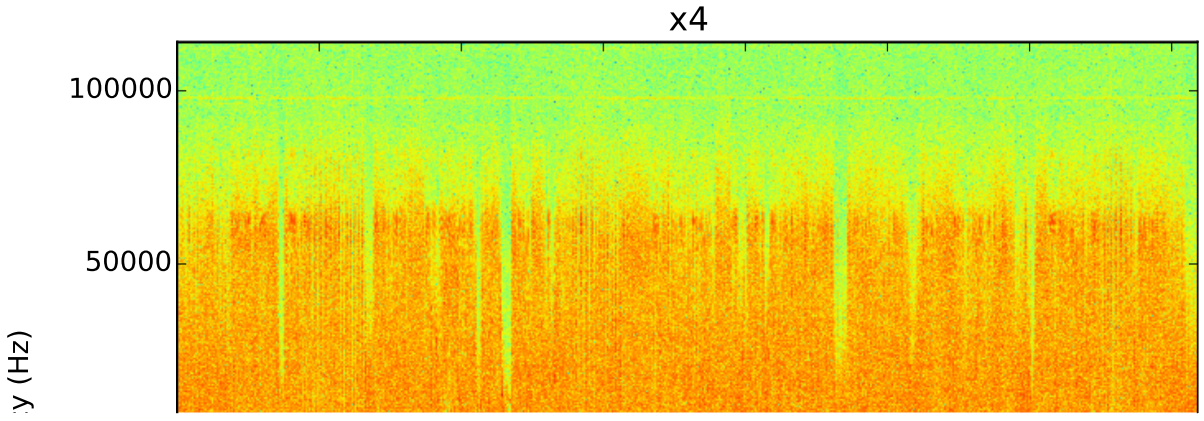
When we first read in our samples (x_1 in the block diagram), the signal looks like this:

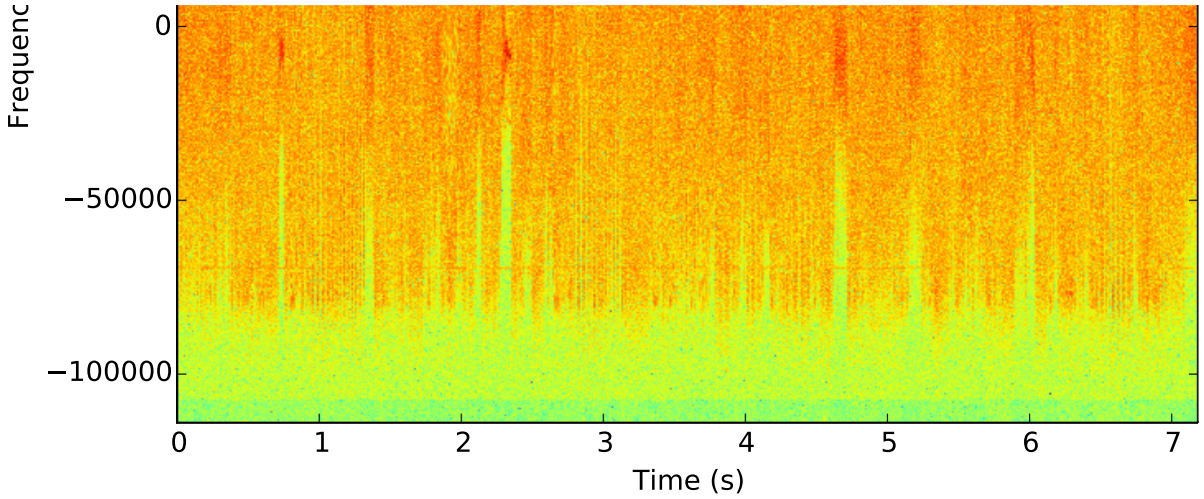


Then, we downconvert the signal and the result (x_2) looks like this:

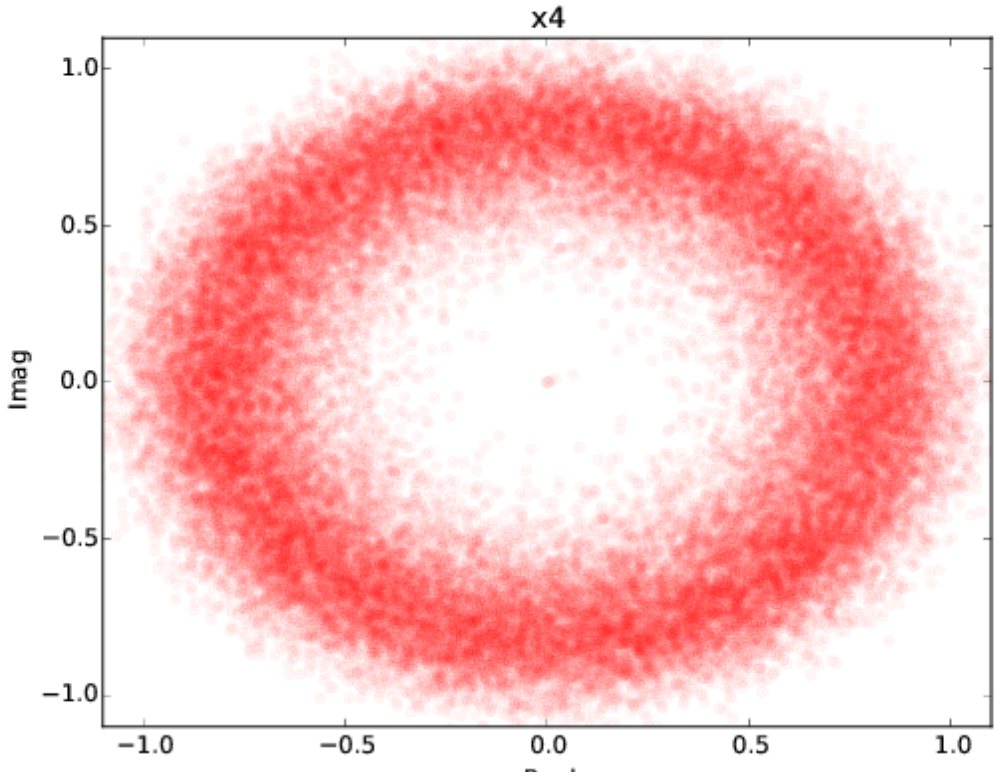


After filtering and downsampling (x_4), we have reduced the bandwidth:



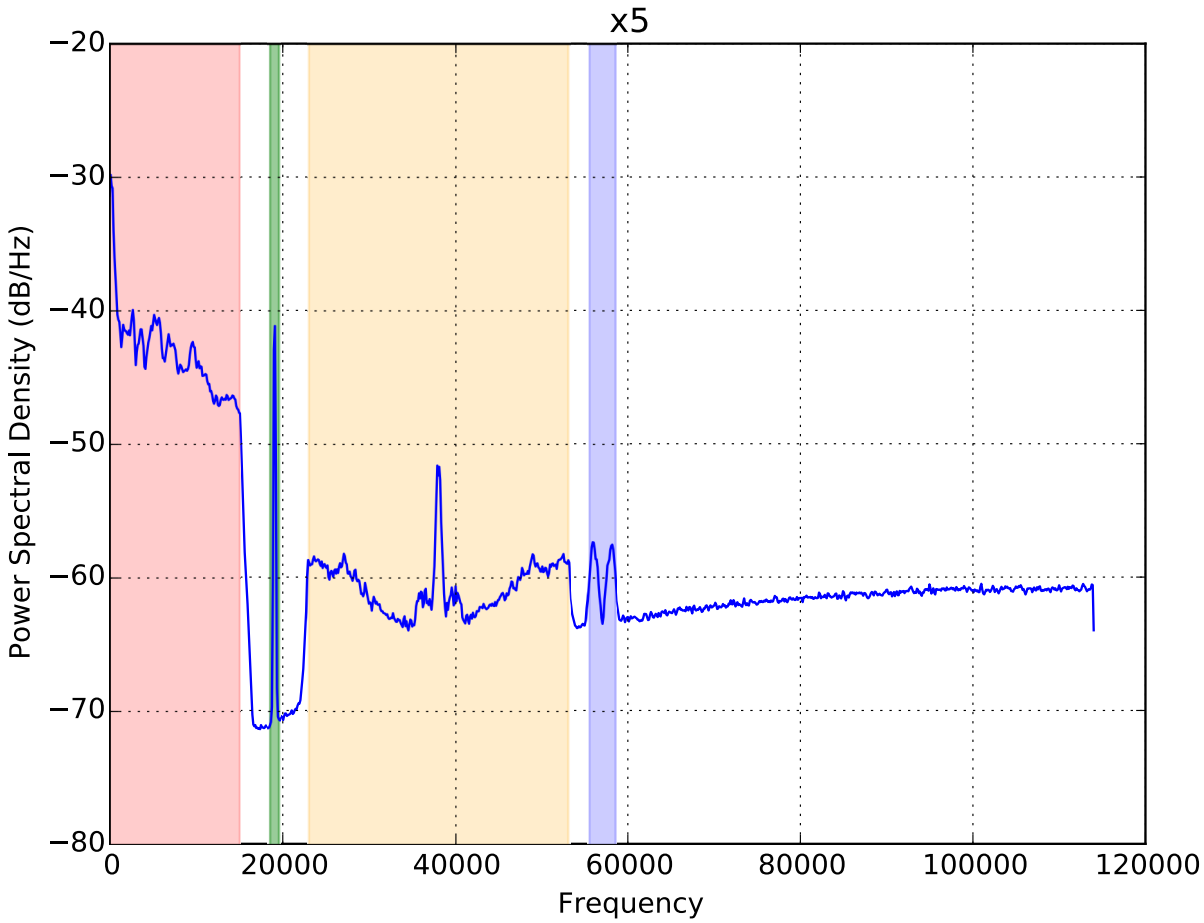


Also, now that we have thrown out a lot of noise and kept only the FM signal, our constellation plot looks like an actual FM constellation:



Real

After demodulating, the signal (x_5) is real, and we can clearly distinguish different parts of the FM signal:



Different parts of the radio broadcast signal are clearly visible in this image. The mono audio signal on the far left is of most interest to us. We can also see the pilot tone at 19 kHz, which is used to help decode the stereo audio centered at 38 kHz. To the left of

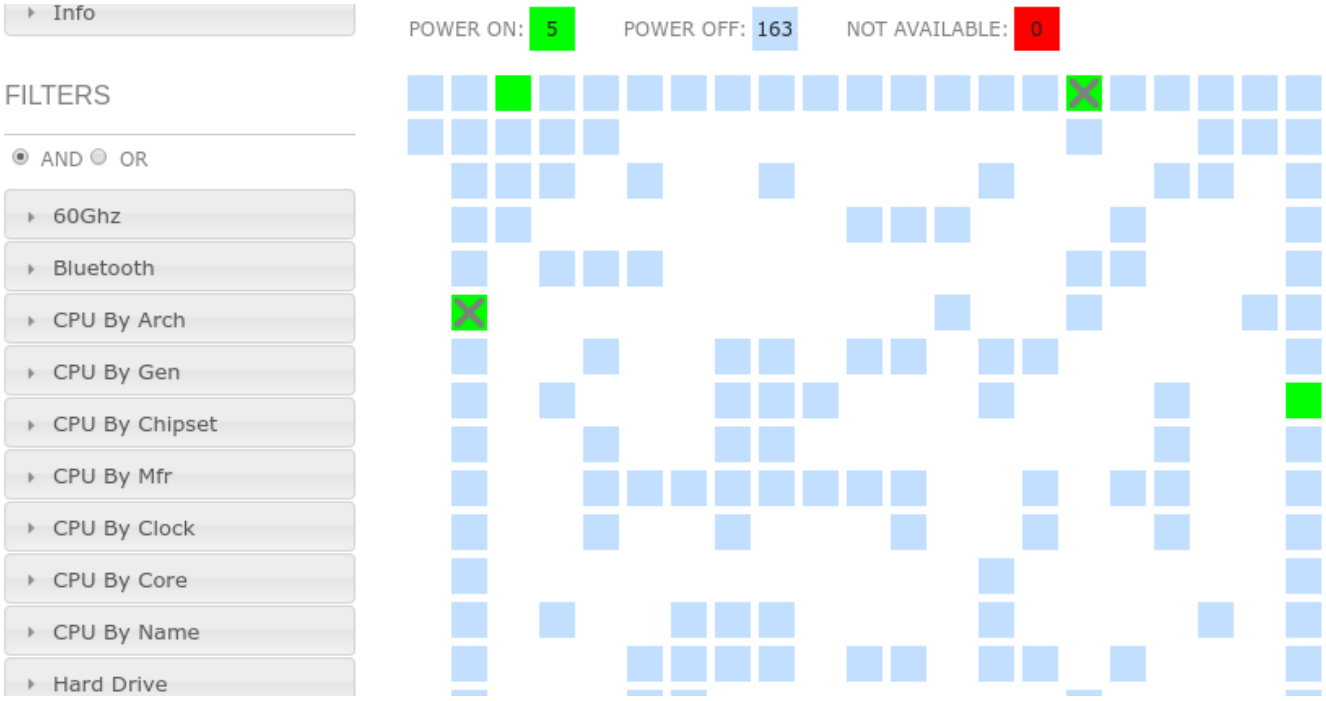
that, we can see digital data carried at 57 kHz - this often includes time, station identification and program information.

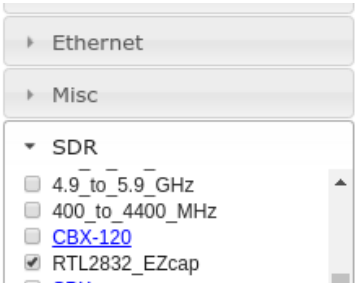
Run my experiment

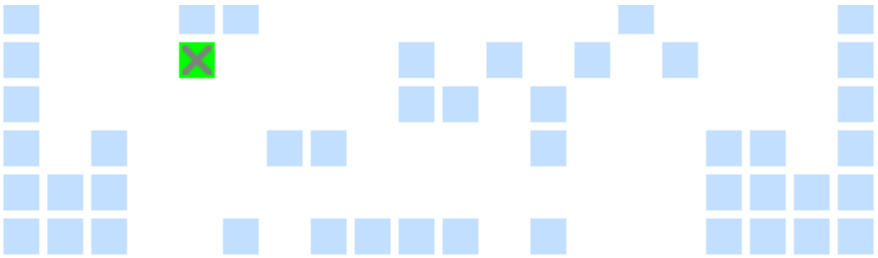
Just want to get results quickly? Skip to the tl;dr version.

These instructions show how to run this experiment on the ORBIT "grid" testbed. To use it, you need to have reserved time on the testbed.

To find the RTL-SDR-equipped nodes on the ORBIT grid, log on to http://geni.orbit-lab.org, click on "Control Panel", then click on "Status Page." Choose "grid" from the tabs at the top, then check the "RTL2832_EZcap" box in the SDR panel. The nodes marked with an "X" have RTL-SDR dongles:







At the beginning of your reservation, SSH in to grid.orbit-lab.org using your GENI keys and your GENI wireless username (which is usually your regular GENI username prefixed with "geni-", e.g. "geni-ffund01"). Load the standard disk image onto an RTL-equipped node:

```
# Assume you are running on node6-1
omf load -i baseline.ndz -t node6-1.grid.orbit-lab.org
# Or on other nodes:
# omf load -i baseline.ndz -t node16-16.grid.orbit-lab.org
# omf load -i baseline.ndz -t node20-6.grid.orbit-lab.org
```

Wait for the disk load process to finish, then turn the node on with

```
omf tell -a on -t node6-1.grid.orbit-lab.org
# Or on other nodes:
# omf tell -a on -t node16-16.grid.orbit-lab.org
# omf tell -a on -t node20-6.grid.orbit-lab.org
```

Wait for the nodes to come on. Then log in, e.g. on the grid

console run

```
ssh root@node6-1
# Or on other nodes:
# ssh root@node16-16
# ssh root@node20-6
```

Start by installing some software:

```
apt-get update  # update list of available software
apt-get -y install git cmake libusb-1.0-0-dev python python-pip
python-dev
apt-get -y install python-scipy python-numpy python-matplotlib
```

Get the RTL software libraries:

```
# Remove other RTL-SDR driver, if it is loaded
modprobe -r dvb_usb_rtl28xxu

git clone https://github.com/steve-m/librtlsdr
cd librtlsdr
mkdir build
cd build
cmake ../
make
make install
```

```
ldconfig
cd

pip install pyrtlsdr
```

Now we're ready to start listening to some FM radio. Specifically, since we're using WINLAB, we'll listen in on 88.7 FM, Rutgers Radio.

Run

```
python
```

to open a Python shell.

Our first step will be to capture some samples off the air. Run

```
from rtlsdr import RtlSdr
import numpy as np
import scipy.signal as signal

import matplotlib
matplotlib.use('Agg') # necessary for headless mode
# see http://stackoverflow.com/a/3054314/3524528
import matplotlib.pyplot as plt


sdr = RtlSdr()
```

In the next command, we will specify the frequency at which to capture samples. We will use 88.7MHz, Rutgers Radio:

```
F_station = int(88.7e6)   # Rutgers Radio
F_offset = 250000         # Offset to capture at
# We capture at an offset to avoid DC spike
Fc = F_station - F_offset # Capture center frequency
Fs = int(1140000)         # Sample rate
N = int(8192000)            # Samples to capture

# configure device
sdr.sample_rate = Fs      # Hz
sdr.center_freq = Fc      # Hz
sdr.gain = 'auto'

# Read samples
samples = sdr.read_samples(N)

# Clean up the SDR device
sdr.close()
del(sdr)

# Convert samples to a numpy array
x1 = np.array(samples).astype("complex64")
```

Now that we've acquired our samples, we can plot the spectogram:

```
plt.specgram(x1, NFFT=2048, Fs=Fs)
plt.title("x1")
plt.ylim(-Fs/2, Fs/2)
plt.savefig("x1_spec.pdf", bbox_inches='tight', pad_inches=0.5)
```

```
plt.close()
```

Copy this file to your own computer by running the following command in a local terminal (*not* on the node or on the grid console):

```
scp -o "StrictHostKeyChecking no"  -o "ProxyCommand ssh
USERNAME@grid.orbit-lab.org nc %h %p" root@node6-1:/root
/x1_spec.pdf .
```

where USERNAME is your GENI wireless username, and you specify the correct node hostname (here node16). This command copies the x1_spec.pdf file to your laptop over a two-hop SSH tunnel.

Looking at the spectogram, we can see that as expected, our FM radio signal is located at an offset from the center. To shift it, in the Python window, run

```
# To mix the data down, generate a digital complex exponential
# (with the same length as x1) with phase -F_offset/Fs
fc1 = np.exp(-1.0j*2.0*np.pi* F_offset/Fs*np.arange(len(x1)))
# Now, just multiply x1 and the digital complex expontential
x2 = x1 * fc1
```

and generate the plot of your shifted signal with

```
plt.specgram(x2, NFFT=2048, Fs=Fs)
```

```
plt.title("x2")
plt.xlabel("Time (s)")
plt.ylabel("Frequency (Hz)")
plt.ylim(-Fs/2, Fs/2)
plt.xlim(0,len(x2)/Fs)
plt.ticklabel_format(style='plain', axis='y' )
plt.savefig("x2_spec.pdf", bbox_inches='tight', pad_inches=0.5)
plt.close()
```

Copy this file by running, in a local terminal,

```
scp -o "StrictHostKeyChecking no"  -o "ProxyCommand ssh
USERNAME@grid.orbit-lab.org nc %h %p" root@node6-1:/root
/x2_spec.pdf .
```

again using your GENI wireless username and the correct node number. If you view this spectogram, you should see the FM signal now centered at baseband.

Our next step will be to filter and then downsample the signal to focus only the FM radio signal.

```
# An FM broadcast signal has  a bandwidth of 200 kHz
f_bw = 200000
n_taps = 64
# Use Remez algorithm to design filter coefficients
lpf = signal.remez(n_taps, [0, f_bw, f_bw+(Fs/2-f_bw)/4, Fs/2],
x3 = signal.lfilter(lpf, 1.0, x2)

dec_rate = int(Fs / f_bw)
```

```
x4 = x3[0::dec_rate]
# Calculate the new sampling rate
Fs_y = Fs/dec_rate
```

Alternatively, we could do these in one step with the `decimate` function:

```
# An FM broadcast signal has  a bandwidth of 200 kHz
f_bw = 200000
dec_rate = int(Fs / f_bw)
x4 = signal.decimate(x2, dec_rate)
# Calculate the new sampling rate
Fs_y = Fs/dec_rate
```

Now we are working with a narrower view of the spectrum, as seen in the spectogram of x_4:

```
plt.specgram(x4, NFFT=2048, Fs=Fs_y)
plt.title("x4")
plt.ylim(-Fs_y/2, Fs_y/2)
plt.xlim(0,len(x4)/Fs_y)
plt.ticklabel_format(style='plain', axis='y' )
plt.savefig("x4_spec.pdf", bbox_inches='tight', pad_inches=0.5)
plt.close()
```

which you can copy by running the SCP command in your local terminal:

```
scp -o "StrictHostKeyChecking no"  -o "ProxyCommand ssh
USERNAME@grid.orbit-lab.org nc %h %p" root@node6-1:/root
```

```
/x4_spec.pdf .
```

We can also plot the constellation, which should have the circular pattern typical of an FM signal:

```
# Plot the constellation of x4.  What does it look like?
plt.scatter(np.real(x4[0:50000]), np.imag(x4[0:50000]), color="r
plt.title("x4")
plt.xlabel("Real")
plt.xlim(-1.1,1.1)
plt.ylabel("Imag")
plt.ylim(-1.1,1.1)
plt.savefig("x4_const.pdf", bbox_inches='tight', pad_inches=0.5)
plt.close()
```

and in the local terminal, run

```
scp -o "StrictHostKeyChecking no"  -o "ProxyCommand ssh
USERNAME@grid.orbit-lab.org nc %h %p" root@node6-1:/root
/x4_const.pdf .
```

If your constellation looks like a filled circle rather than an outline, this suggests a noisy signal. You'll know for certain when you listen to the audio output!

Since we are left with just the 200kHz FM broadcast signal, we can now demodulate it with our polar discriminator:

```
### Polar discriminator
y5 = x4[1:] * np.conj(x4[:-1])
x5 = np.angle(y5)
```

and we can visualize the signal with:

```
# Note: x5 is now an array of real, not complex, values
# As a result, the PSDs will now be plotted single-sided by defa
# a real signal has a symmetric spectrum)
# Plot the PSD of x5
plt.psd(x5, NFFT=2048, Fs=Fs_y, color="blue")
plt.title("x5")
plt.axvspan(0,              15000,         color="red", alpha=0.2
plt.axvspan(19000-500,      19000+500,     color="green", alpha=0
plt.axvspan(19000*2-15000, 19000*2+15000, color="orange", alpha=
plt.axvspan(19000*3-1500,  19000*3+1500,  color="blue", alpha=0.
plt.ticklabel_format(style='plain', axis='y' )
plt.savefig("x5_psd.pdf", bbox_inches='tight', pad_inches=0.5)
plt.close()
```

Transfer the file with the SCP command

```
scp -o "StrictHostKeyChecking no"  -o "ProxyCommand ssh
USERNAME@grid.orbit-lab.org nc %h %p" root@node6-1:/root
/x5_psd.pdf .
```

Compare this to the figure above showing the parts of the FM broadcast signal. Broadcasts vary with respect to which parts of the signal they include.

Now we're ready for the de-emphasis filter:

```
# The de-emphasis filter
# Given a signal 'x5' (in a numpy array) with sampling rate Fs_y
d = Fs_y * 75e-6   # Calculate the # of samples to hit the -3dB
x = np.exp(-1/d)   # Calculate the decay between each sample
b = [1-x]          # Create the filter coefficients
a = [1,-x]
x6 = signal.lfilter(b,a,x5)
```

And then we can decimate once again to focus on the mono audio part of the broadcast:

```
# Find a decimation rate to achieve audio sampling rate between
audio_freq = 44100.0
dec_audio = int(Fs_y/audio_freq)
Fs_audio = Fs_y / dec_audio

x7 = signal.decimate(x6, dec_audio)
```

and finally, we can write to an audio file:

```
# Scale audio to adjust volume
x7 *= 10000 / np.max(np.abs(x7))
# Save to file as 16-bit signed single-channel audio samples
x7.astype("int16").tofile("wbfm-mono.raw")
```

Find out what your audio sampling rate by checking the value of

```
print(Fs_audio)
```

Copy the audio file to your laptop by running the SCP command in your local terminal:

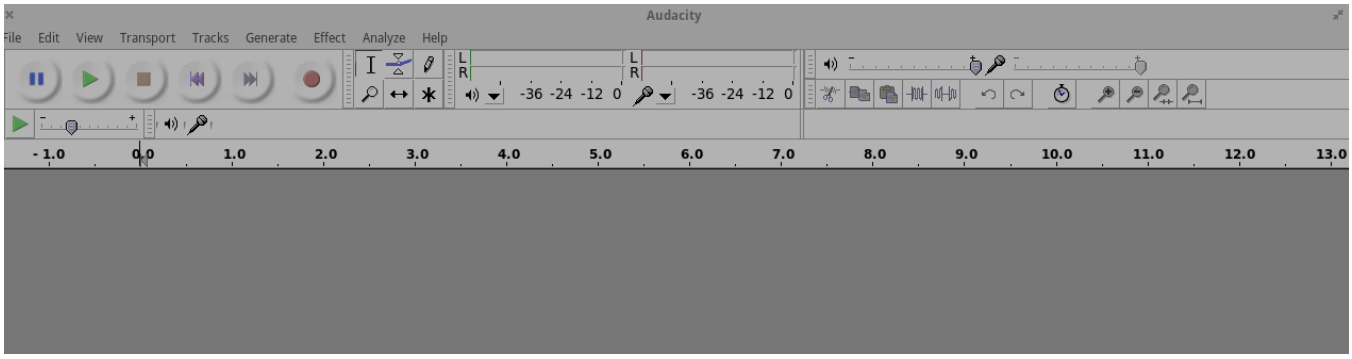
```
scp -o "StrictHostKeyChecking no"  -o "ProxyCommand ssh
USERNAME@grid.orbit-lab.org nc %h %p" root@node6-1:/root/wbfm-
mono.raw .
```

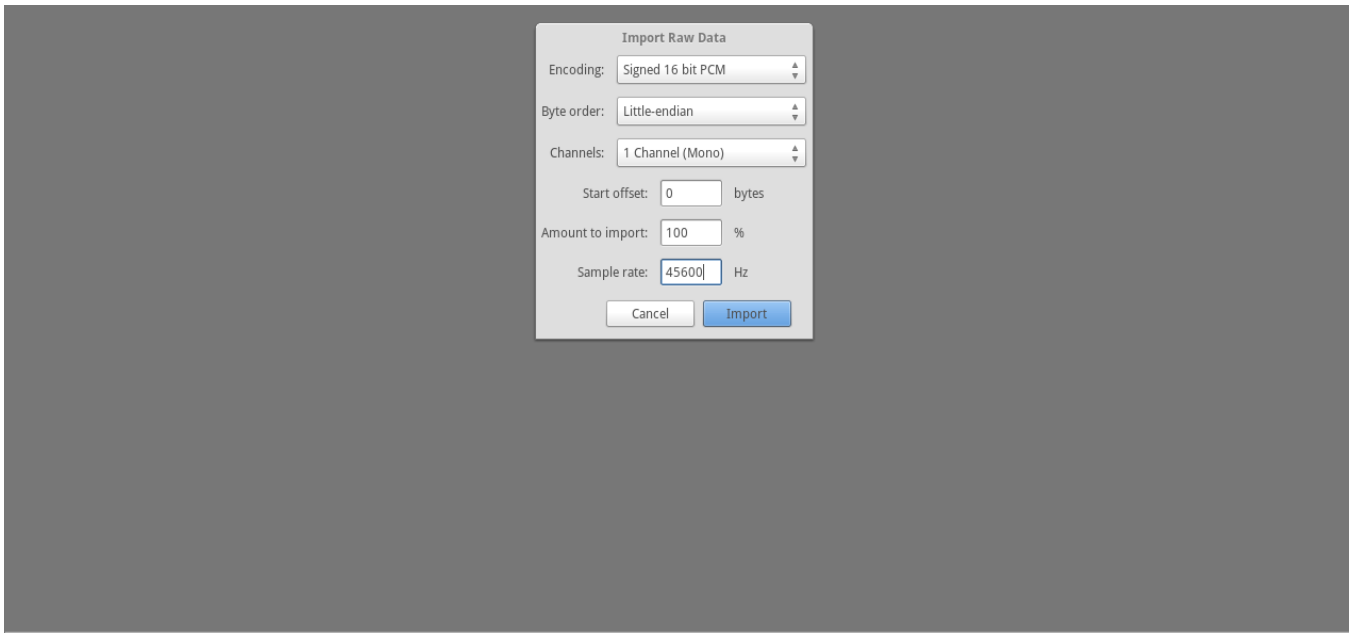
If you are running Linux, you can play back this file from the terminal with

```
aplay wbfm-mono.raw -r 45600 -f S16_LE -t raw -c 1
```

where the value you pass to the "-r" argument is the audio frequency (here, 45600 Hz).

Alternatively, you can use Audacity (available for Windows, Linux, and Mac) to play back your decoded audio file after you transfer it to your computer. From the "File" menu, choose "Import > Raw Data" and then make sure to use the appropriate settings (set the sample rate to whatever rate your audio data is sampled at):





tl;dr version

These instructions show how to run this experiment on the "grid" testbed on the ORBIT aggregate. To use it, you need to have reserved time on the testbed.

At the beginning of your reservation, SSH in to grid.orbit-lab.org using your GENI keys and your GENI wireless username (which is usually your regular GENI username prefixed with "geni-", e.g. "geni-ffund01"). Load the standard software radio disk image onto an RTL-equipped node:

```
# Assume you are running on node6-1
omf load -i baseline.ndz -t node6-1.grid.orbit-lab.org
# Or on other nodes:
# omf load -i baseline.ndz -t node16-16.grid.orbit-lab.org
```

```
# omf load -i baseline.ndz -t node20-6.grid.orbit-lab.org
```

Wait for the disk load process to finish, then turn the node on with

```
omf tell -a on -t node6-1.grid.orbit-lab.org
# Or on other nodes:
# omf tell -a on -t node16-16.grid.orbit-lab.org
# omf tell -a on -t node20-6.grid.orbit-lab.org
```

Wait for the nodes to come on. Then log in, e.g. on the grid console run

```
ssh root@node6-1
# Or on other nodes:
# ssh root@node16-16
# ssh root@node20-6
```

Set up the necessary software environment with

```
wget https://git.io/vPy2w -O fm-radio-setup.sh
bash fm-radio-setup.sh
wget https://git.io/vPy2M -O fm-radio.py
```

Then run

```
python fm-radio.py 88.7e6
```

to capture Rutgers Radio at 88.7 FM (88.7 MHz).

Transfer files to your computer with

```
scp -o "StrictHostKeyChecking no"  -o "ProxyCommand ssh
USERNAME@grid.orbit-lab.org nc %h %p" root@node6-1:/root/*.pdf
.
scp -o "StrictHostKeyChecking no"  -o "ProxyCommand ssh
USERNAME@grid.orbit-lab.org nc %h %p" root@node6-1:/root/wbfm-
mono.raw .
```

substituting your GENI wireless username in place of "USERNAME" in the commands above and the name of the node you are using in place of "node6-1".

Notes

This experiment was developed on the following software versions:

- Ubuntu 14.04.3
- Python 2.7.5-5ubuntu3
- numpy 1.8.2-0ubuntu0.1
- scipy 0.13.3-1build1

- pyrtlsdr 0.2.0
- librtlsdr from github with most recent commit hash 8b4d755ba1b889510fba30f627ee08736203070d

The setup script and Python script can be found in this gist.

Fraida Fund

Read more posts by this author.

Brooklyn, NY

http://witestlab.poly.edu/~ffund/

Share this post

Did you reproduce this experiment? Have useful information to share with other intrepid researchers? Post it here! Comments are posted following moderation.

Ashanthi Maxworth • 4 months ago

Thanks a lot for this article! This covers most of the signal processing required for received signal processing.

READ THIS NEXT

Worst-case packet delay in an unstable, finite M/M/1 queue

This experiment is about the effect of queue size on packet delay in a queue where the service time...

YOU MIGHT ENJOY

Average length of an M/M/1 queue

This experiment reproduces a classic result in queueing theory: the length of the M/M/1 queue as its...