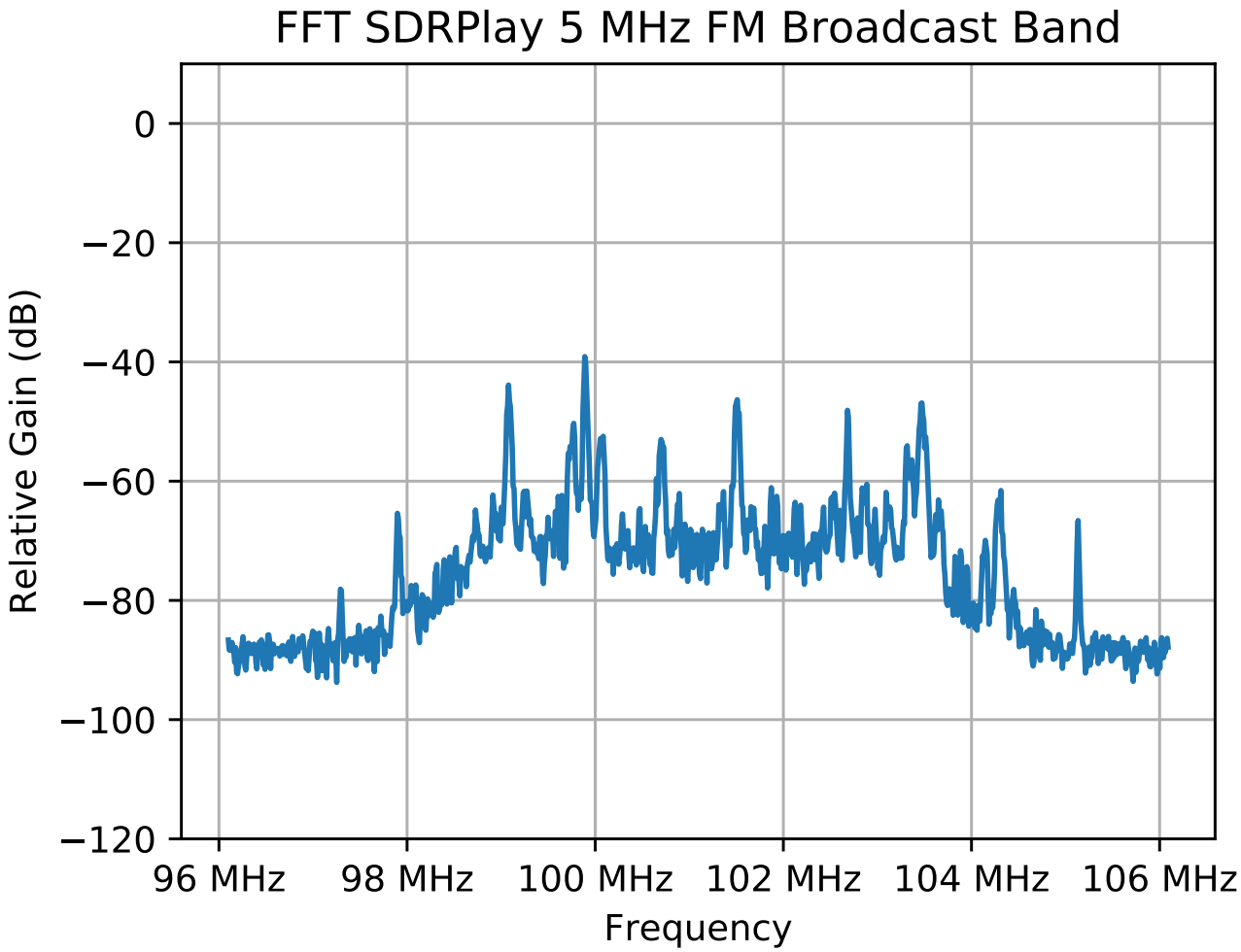

Fun with GNU Radio and Digital RF

Gregory Raven KF5N

December 2, 2018

Fun with GNU Radio and Digital RF

Contents

List of Figures

Chapter 1

Introduction

This project is a brief look at the Python package "Digital RF". The Digital RF repository is here:

https://github.com/MITHaystack/digital_rf

From the github repository:

> The Digital RF project encompasses a standardized HDF5 format for reading and writing of radio frequency data and the software for doing so.

"Digital RF" is a key component of the Ham Science Space Weather project:

http://hamsci.org/articles/personal-space-weather-station

I first learned about this project at the 2018 TAPR Conference in Albuquerque, New Mexico. Here is a link to the slides from the presentation by Nathaniel Frissell:

http://hamsci.org/sites/default/files/pages/swstation/20181116_TAPR_Sunday_Seminar_Frissell_W2NAF.pdf

Digital RF is introduced on slide 86.

1.1 What is described here?

This is not a detailed technical analysis. It is only an introduction to the technology with a bunch of qualitative comments mixed in.

I was curious about the technology, and this project was a cheap and (somewhat) quick way to see what it does. Ham Science is an interesting branch of the amateur radio hobby!

1.2 What Equipment is Involved?

I decided on this "mid-tier" SDR receiver for use with this project:

https://www.sdrplay.com/rsp2

Note that the RSP2 can be locked to an external frequency reference. I didn't use this feature, however, it seems like a good feature to have for more demanding projects in the future.

FM broadcast band signals were used for testing purposes. This antenna worked well with the RSP2:

https://www.amazon.com/Portable-Antenna-Bundle-Adapter-Telescopic/dp/B07DPNPK4D/ref=lp_10230687011_1_8?srs=10230687011&ie=UTF8&qid=1542991022&sr=8-8

The primary computer is a desktop Lenovo TS140 ThinkServer with a Samsung SSD drive. Two other machines were evaluated, including one with an old-style rotary magnetic disk drive.

The primary operating system used is Ubuntu 18.04. The standard Debian distribution was tested on one of the auxiliary machines.

A few brief tests with a Synology "Network Attached Storage" device were conducted.

1.3 Github Repository for the Project

The docs and Python scripts are located in this repository:

https://github.com/Greg-R/explore_digital_rf

1.4 Main Goal of the Project

The primary goal of the project was to use Digital RF sources and sinks in a GNU Radio Companion "flowgraph". RF data of 5 MHz bandwidth was written to a local file. This data was then read back into the GRC and demodulated.

In addition to read/write of RF data, a short sample of the data was read into a Python script, and a plot of the RF spectrum was created using Python numeric and scientific libraries.

Chapter 2

Install Digital RF

The following sections assume you have access to a machine with a Linux OS. I used Ubuntu 18.04 on the primary machines, and Debian 9 on an older machine. The Python work was done within an "environment" with the latest version Python 3.7.1.

2.1 Installation of Digital_RF

Installation of the Digital_RF Package is trivial. Use pip at the command line:

```
pip install digital_rf
```

This may install the GNU Radio Companion blocks files in an unusual location:

```
/home/(username)/.local/share/gnuradio/grc/blocks
```

This path needs to be added to the GNU Radio config.conf file as follows. If this file was not created during installation, create the file here:

```
/home/(userid)/.gnuradio/config.conf
```

Add this section (if it does not already exist) with the following line:

```
[grc]
local_blocks_path = /home/(username)/.local/share/gnuradio/grc/blocks
```

Chapter 3

Install SDR Play Driver and GNU Radio Components

Two components are required to use the SDRPlay RSP2 receiver with GNU Radio.

- Hardware driver
- GNU Radio components

3.1 Installation of the SDRPlay Driver

Download the driver installation code from this page:

https://www.sdrplay.com/downloads/

Click the Linux x86 tab, and download the "API/HW DRIVER – V2.13 (20TH JUNE 2018)" file.

Run the installer and follow the instructions. You will be guided through the usual license acceptance agreement. This should install quickly.

3.2 Installation of gr-sdrplay

The gitlab repository is here:

https://gitlab.com/HB9FXQ/gr-sdrplay.git

Clone the repository to your Linux machine:

```
git clone https://gitlab.com/HB9FXQ/gr-sdrplay.git
```

Note this PDF which is located in the repository makes the installation process easy, as you can cut and paste the commands:

https://www.sdrplay.com/docs/gr-sdrplay-workflow.pdf

Here are the installation commands:

```
cd gr-sdrplay
mkdir build && cmake .. && make
sudo make install && sudo ldconfig
```

If all went well, the next time you open GNU Radio Companion you should see the SDRPlay sources listed under the “(no module specified)” category in the library at the extreme lower right corner of the GUI.

Chapter 4

GNU Radio Flowgraph with Digital RF Sink and Source

Getting GNU Radio installed and working properly is the hardest part of this project. Next is the fun part! It is time to capture data using Digital RF.

Both the SDRPlay and Digital RF components will appear in the “(no module specified)” menu in the lower right corner of GNU Radio companion:

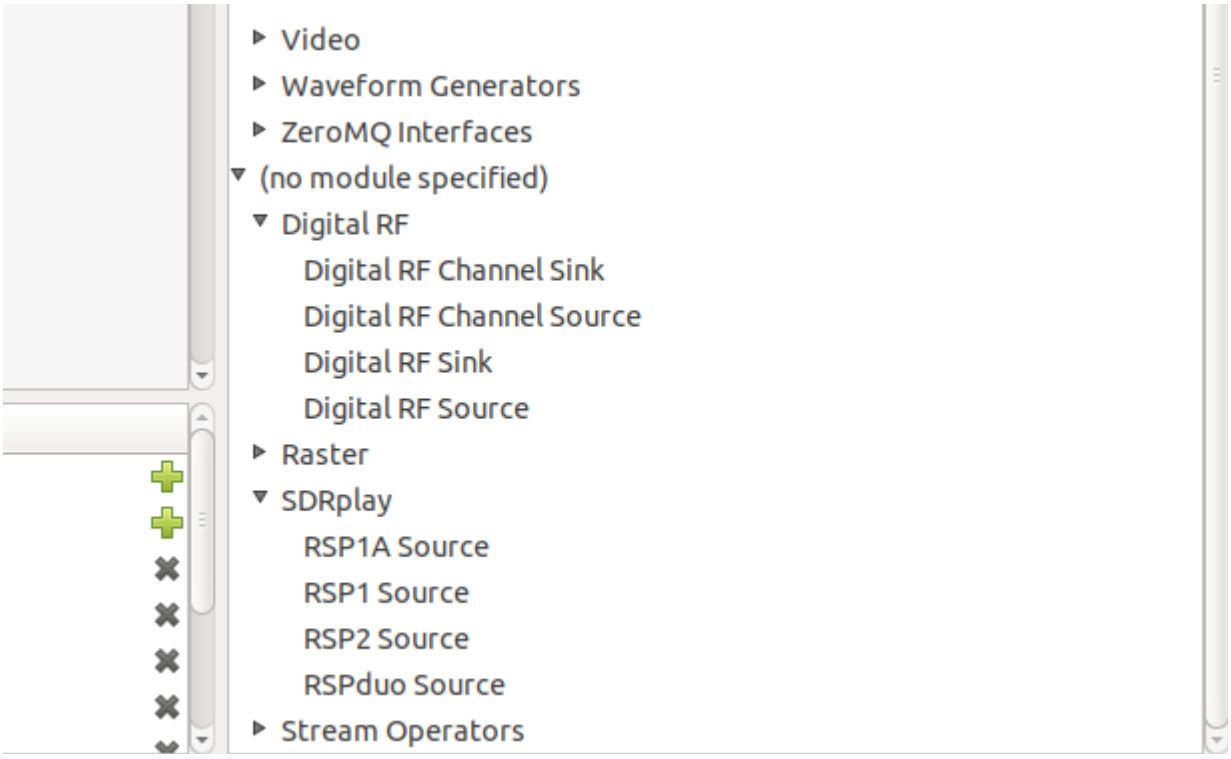

Figure 4.1: GNU Radio Companion ”(no module specified)”

The SDRPlay will require the “RSP2 Source.” We will use the “Digital RF Channel Sink” and the

"Digital RF Channel Source" for experimenting with Digital RF.

4.1 FM Broadcast Receiver "Flowgraph"

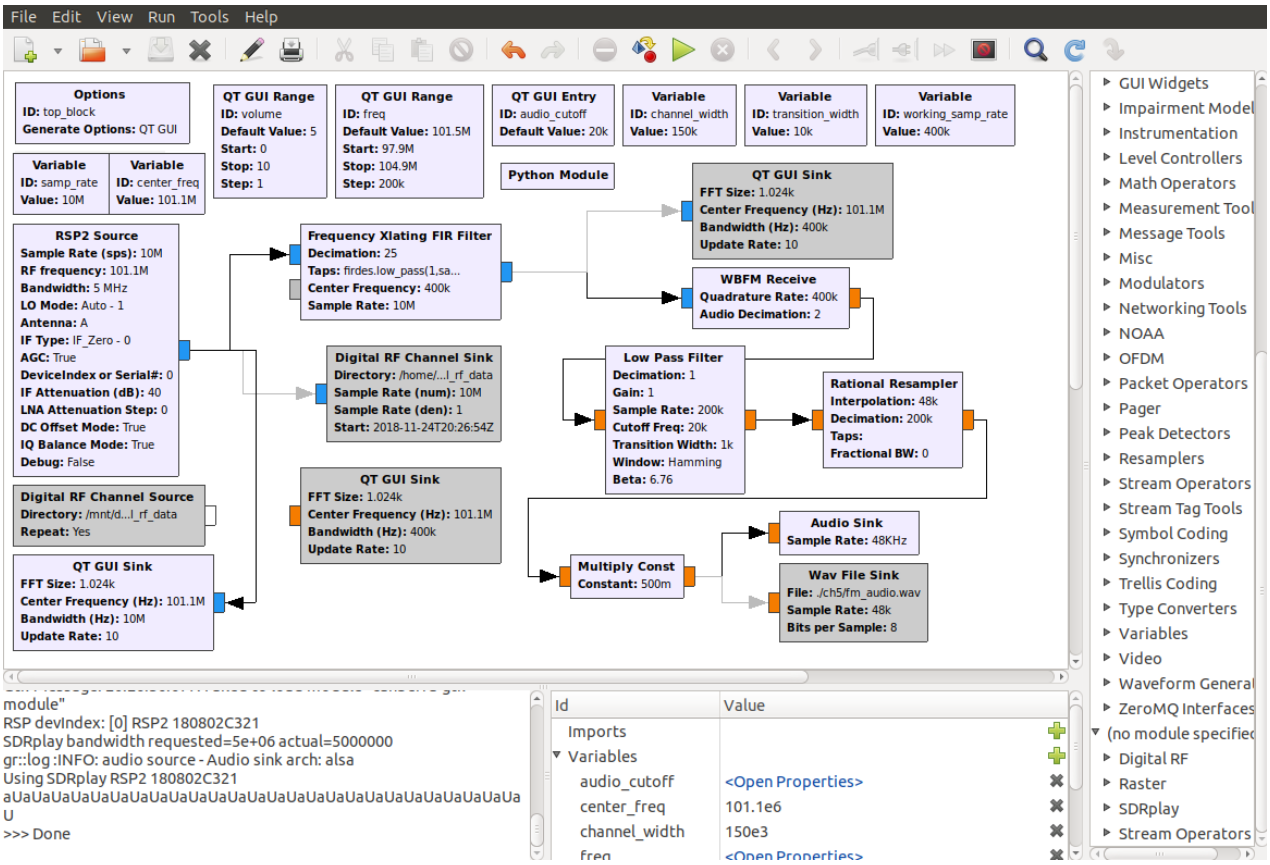

Figure 4.2: GRC Flowgraph with RSP2 Source and Digital RF Sources and Sinks

The flowgraph is a simple DSP implementation of an FM broadcast receiver.

Why FM broadcast? The provides a consistent set of evenly spaced signals (every 200 kHz) which is easy to demodulate for testing purposes. A small whip antenna connected directly to the RSP2 provided sufficiently strong signals across the FM broadcast band.

Components can be enabled and disabled in the GRC flowgraph. The image above shows several of the components disabled. This functionality was used in various experiments with reading from and writing to the Digital RF sinks and sources.

4.2 FM Broadcast Spectrum as Received by the RSP2

The RSP2 bandwidth and sample rate was empirically adjusted to find the maximum bandwidth possible using the Lenovo TS140 machine. The bandwidth is set to 5 MHz, with a barely Nyquist sample rate of 10 MHz. Attempting to push the bandwidth above this resulted in lots of undersampling in the Linux Pulseaudio system. This is easily heard as distorted audio from the speaker.

5 MHz was deemed acceptable bandwidth for the purposes of this project. The RSP2 claims 8 MHz maximum bandwidth. Perhaps some future optimizations will allow reaching its upper limit.

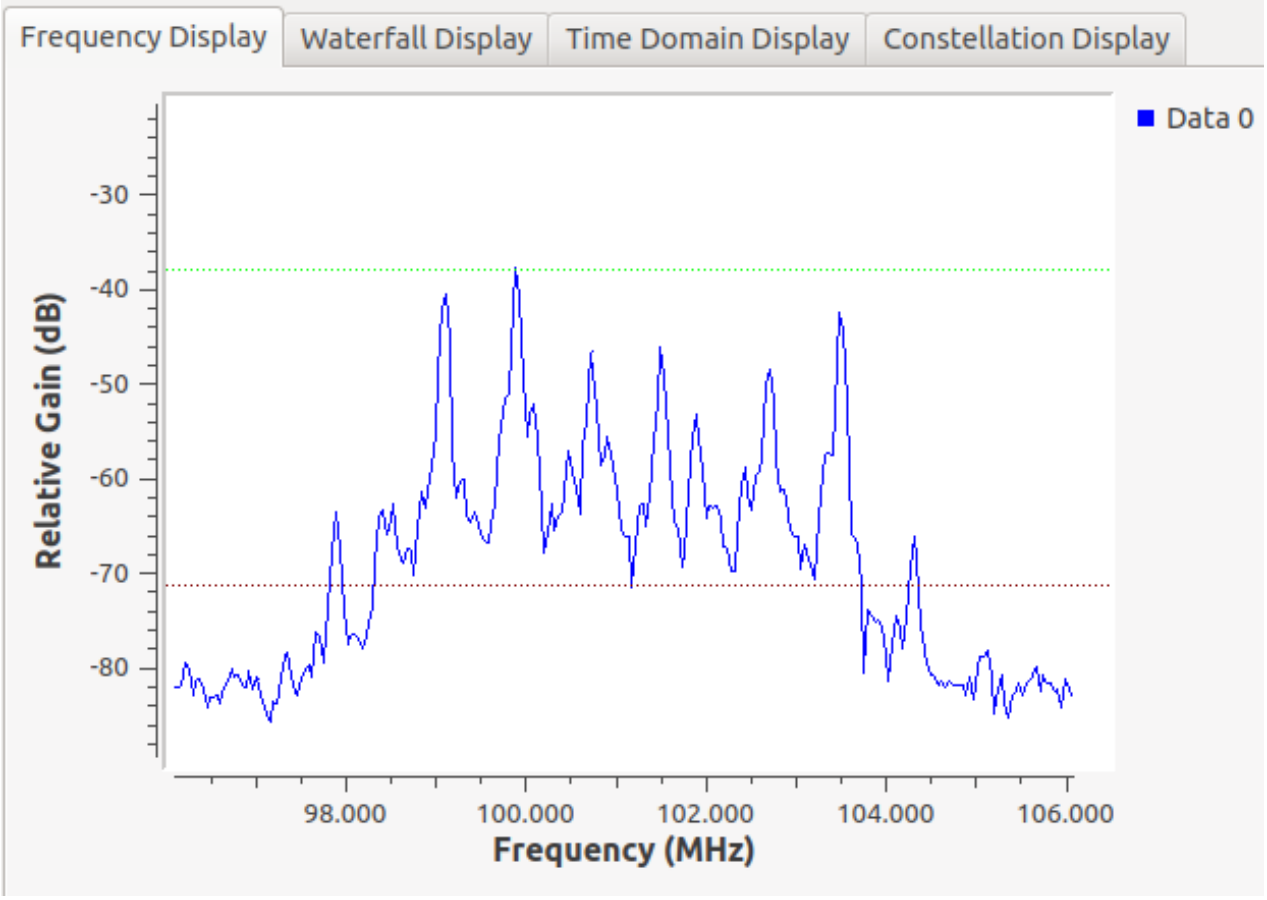

Figure 4.3: 5 MHz Chunk of the FM Broadcast Spectrum as Seen by the RSP2

In the above frequency display the local FM broadcast stations are easily spotted. This consistent and reliable spectrum made the remaining tests easy to accomplish.

4.3 Adding the Digital RF Sink

The “RF Channel Sink” is dragged from the “No Module Specified -> Digital RF” from the GRC Library menu.

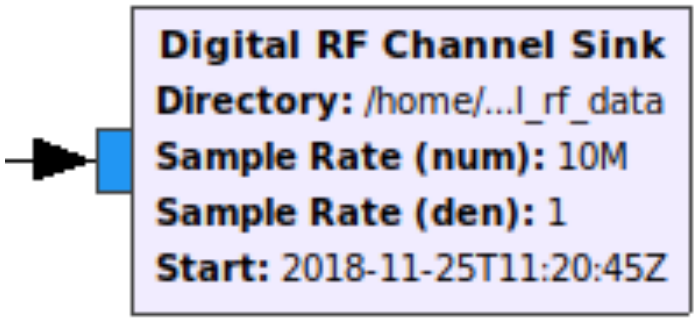

Figure 4.4: The Digital RF Channel Sink

The options required for this sink are simple:

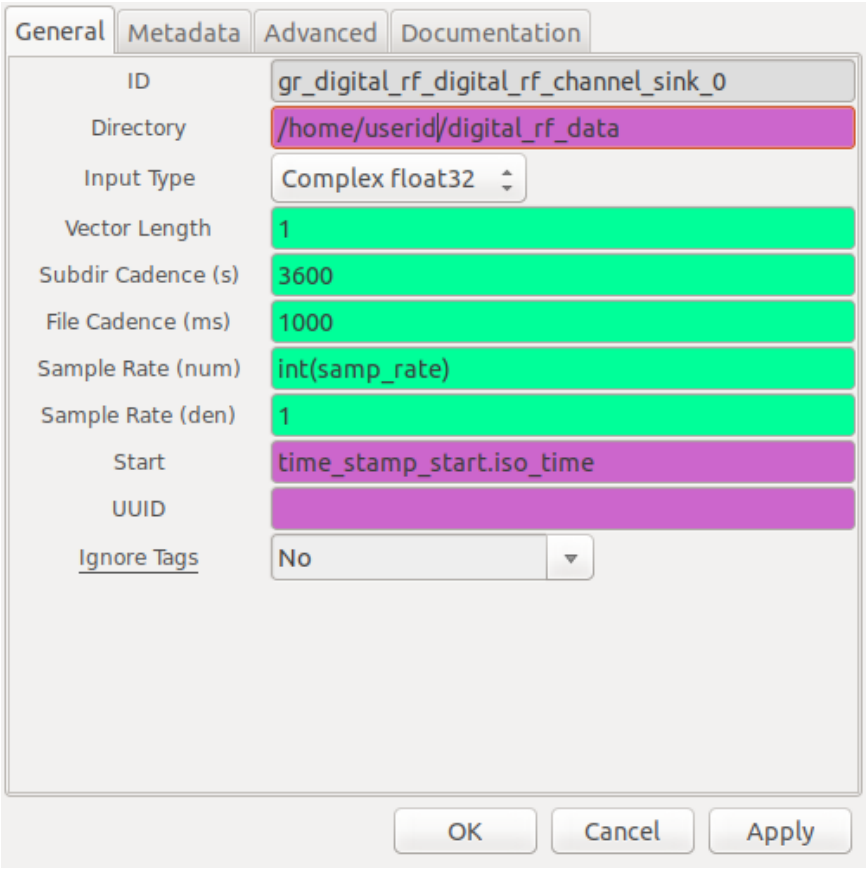

Figure 4.5: Digital RF Channel Sink Options

"Directory" is the file path to which the data will be written. Make sure this is a high bandwidth path! I used a path in my home directory which is written to a SATA SSD drive. It's very fast!

Chose the type of data. I am currently using Complexfloat32, which is probably more resolution than required. The sample rate uses the same variable "samp_rate" used throughout the flowgraph.

"Start" is the time-stamp for the beginning of the data. I used an ISO8601 formatted string. The string is obtained via a Python module which is included via the "Python Module" component located in the "Misc" menu of the library.

4.3.1 Python Time Stamping Module

The Python module which generates the time stamp for the RF Channel Sink:

```
import datetime

iso_time = datetime.datetime.now().replace(microsecond=0).isoformat() +'Z'
```

The above code is typed into the "Code" editor in the Python Module component GUI.

The module's ID is the name of the module. Returning a time string is this one-liner:

```
time_stamp_start.iso_time
```

The above is typed into the "Start" entry of the RF Channel Sink.

It is a bit unclear how this time stamping function works. Is there significant latency between the retrieval of the time and the capture of the first sample? For now, it is close enough.

4.4 Writing an HDF5 Format Data File with the Digital RF Sink

Make sure the Digital RF Channel Sink is enabled and connected to the RSP2 Source component.

Simply run the Flowgraph. The HDF5 formatted file system will be written to the Directory path entered in the sink component. That's it!

A quick one-minute run with 5 MHz bandwidth created 5 gigabytes of data!

4.5 Reading the HDF5 Data Into the Flowgraph

Instantiate a Digital RF Channel Source into your flowgraph:

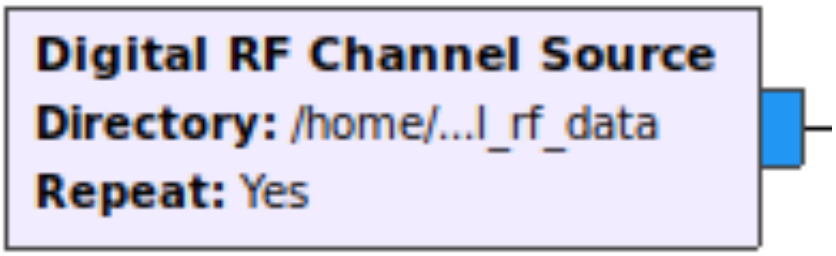

Figure 4.6: Digital RF Channel Source

The source has the following options:

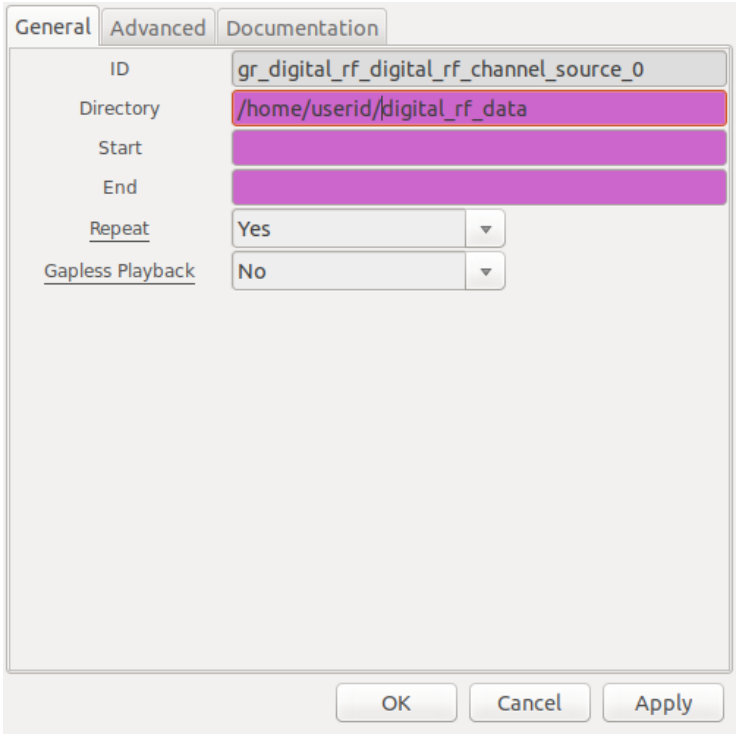

Figure 4.7: Digital RF Channel Source Options

The option for “Directory” should be the same file path entered into the sink component. Switch “Repeat” to “Yes”.

Deactivate the RSP2 source and the RF Channel Sink. Connect the output of the RF Channel Source to the input of the Frequency Translating FIR filter and the QT GUI Sink.

Run the flowgraph. If your system is fast enough, it will appear as though the RSP2 hardware is still connected! I was able to tune and demodulate several of the captured FM broadcast stations. There is a noticeable gap when the file repeats. It is really amazing to smoothly stream 5 MHz of spectrum from a 5 gigabyte file which plays in one minute on commodity hardware!

Reading and writing to the SSD drive worked very well. It will be used as the “standard” for comparison to other hardware.

Chapter 5

Experimenting with Slower Hardware

5.1 Fastest Hardware Specifications

The successful test with Digital RF Sinks and Sources in GNU radio was carried out on a reasonably powerful machine. Here is an abbreviated output from the inxi -Fx command:

```
System:    Host: greg-lenovo Kernel: 4.15.0-39-generic x86_64 bits: 64 gcc: 7.3.0
Desktop: Gnome 3.28.3 (Gtk 3.22.30) Distro: Ubuntu 18.04.1 LTS
Machine:   Device: desktop System: LENOVO product: ThinkServer TS140 v: 70A4001MUX serial:
Mobo: LENOVO model: ThinkServer TS140 serial: N/A BIOS: LENOVO v: FBKTCZAUS date: 05/03/201
CPU:       Quad core Intel Xeon E3-1225 v3 (-MT-MCP-) arch: Haswell rev.3 cache: 8192 KB
clock speeds: max: 3600 MHz 1: 2811 MHz 2: 2983 MHz 3: 2893 MHz 4: 2912 MHz
Card-2 Intel Xeon E3-1200 v3/4th Gen Core Processor HD Audio Controller
Sound: Advanced Linux Sound Architecture v: k4.15.0-39-generic
Network:   Card: Intel Ethernet Connection I217-LM driver: e1000e v: 3.2.6-k port: f080 bus
IF: eno1 state: up speed: 1000 Mbps duplex: full mac: 6c:0b:84:09:f9:de
Drives:    HDD Total Size: 500.1GB (41.6% used)
ID-1: /dev/sda model: Samsung_SSD_860 size: 500.1GB
Partition: ID-1: / size: 458G used: 194G (45%) fs: ext4 dev: /dev/sda1
```

Note that the drive that the RF Channel Sink writes to is a Samsung solid-state drive. The specifications claim “Up to 520 MB/s Max. Sequential Write Speed”.

5.2 Read/Write over LAN to Network Attached Storage

The next test was writing to a “Network Attached Storage” device on the local area network. The connection is 1000MB over a cable which is approximately 10 meters long. I have never verified the actual write speed, as it has always performed adequately.

The Synology NAS is using 6 TB Western Digital mechanical hard drives:

WD 6TB Red 5400 rpm SATA III 3.5" Internal NAS HDD a sustained rate of up to 175 MB/s.

The NAS is a Synology DS216+II. This device claims "... delivers up to an average 113.08 MB/s reading and 111.78 MB/s writing speed".

The resultant audio in both read and write modes was choppy. GNU Radio reports numerous "underruns". The underrun density is not high. I am guessing one step down in bandwidth will cure the problem for read/write to the NAS. This will wait for future trials.

5.3 Dell Laptop SanDisk SSD

I have some other machines, and I tried the same read/write HDF5 experiment on them.

First, a Dell laptop which I purchased at a hamfest for $25. It had a dead hard drive, and it was replaced with a 240 GB SSD. This is its hardware:

```
System:    Host: greg-inspiron Kernel: 4.15.0-39-generic x86_64 bits: 64 gcc: 7.3.0
Desktop: Gnome 3.28.3 (Gtk 3.22.30) Distro: Ubuntu 18.04.1 LTS
Machine:   Device: portable System: Dell product: Inspiron 3542 serial: N/A
Mobo: Dell model: 0V25HW v: A04 serial: N/A BIOS: Dell v: A04 date: 08/05/2014
CPU:       Dual core Intel Core i3-4030U (-MT-MCP-) arch: Haswell rev.1 cache: 3072 KB
clock speeds: max: 1900 MHz 1: 938 MHz 2: 1021 MHz 3: 905 MHz 4: 1018 MHz
Card-2 Intel Haswell-ULT HD Audio Controller driver: snd_hda_intel bus-ID: 00:03.0
Sound: Advanced Linux Sound Architecture v: k4.15.0-39-generic
Network:   Card-1: Qualcomm Atheros QCA9565 / AR9565 Wireless Network Adapter
IF: wlp6s0 state: up mac: ec:0e:c4:12:7a:5b
Card-3: Atheros usb-ID: 001-007
IF: null-if-id state: N/A speed: N/A duplex: N/A mac: N/A
Drives:    HDD Total Size: 240.1GB (25.1% used)
ID-1: /dev/sda model: SanDisk_SDSSDA24 size: 240.1GB temp: 33C
Partition: ID-1: / size: 220G used: 57G (27%) fs: ext4 dev: /dev/sda1
```

The SanDisk SSD in this machine claims "SATA 6.0 Gb/s".

The hamfest laptop performed surprisingly well! I noted only an occasional gap in the audio played back from the Digital RF Source component.

Just for fun, I attempted to write to the NAS over WIFI, which is how I typically connect the laptop to my home network. The performance was terrible! It's probably 95% gap, and 5% valid data. The data pipe just can't keep up with the incoming data from the receiver.

5.4 Lenovo ThinkCentre "Kiosk Computer"

This is another hamfest special. A "Kiosk Computer" which in a small desktop format:

```
System:    Host: thinkcentre Kernel: 4.9.0-8-amd64 x86_64 (64 bit gcc: 6.3.0)
Desktop: Gnome 3.22.3 (Gtk 3.22.11-1) Distro: Debian GNU/Linux 9 (stretch)
Machine:   Device: desktop System: LENOVO product: 6071A37 v: ThinkCentre M57
```

```
Mobo: LENOVO model: LENOVO BIOS: LENOVO v: 2RKT64BUS date: 01/08/2014
CPU:       Dual core Intel Core2 Duo E6550 (-MCP-) cache: 4096 KB
clock speeds: max: 2333 MHz 1: 2000 MHz 2: 2000 MHz
Sound: Advanced Linux Sound Architecture v: k4.9.0-8-amd64
Network:   Card: Intel 82566DM-2 Gigabit Network Connection driver: e1000e v: 3.2.6-k port
IF: enp0s25 state: up speed: 1000 Mbps duplex: full mac: 00:21:86:11:c0:7c
Drives:    HDD Total Size: 120.0GB (27.5% used)
ID-1: /dev/sda model: KINGSTON_SA400S3 size: 120.0GB temp: 33C
```

This Lenovo ThinkCentre machine is from about the 2006 time period.

A $24 Kingston SSD was installed in this machine. Ubuntu 18.04 did not boot properly for some unknown reason, so I tried the latest Debian distro and it worked perfectly. Build and install of GNU Radio also proceeded without incident.

This machine is located in the shack, and I have used it successfully for narrow-band HF reception.

However, a 5 MHz IQ data stream is beyond its capability. Audio was terribly distorted. The test revealed essentially 95% buffer underflows. This machine is unusable at this data rate.

Figure 5.1: The Vintage "Kiosk Computer"

Chapter 6

Inspecting HDF5 Data with Python

So now that you have gigabytes of RF Digital data on your local drive, what do you do next?

How about analyzing it with Python? We'll take a sample of the data and examine it, and then plot the spectrum using Python libraries.

Example code is needed. There are two sources for this:

https://github.com/MITHaystack/digital_rf

The "Example Usage" section at the above link is almost all you need.

More detailed examples are found in the Python examples folder:

https://github.com/MITHaystack/digital_rf/tree/master/python/examples

Using the example code, a short Python script was written which reads a chunk of data and performs an FFT. The goal was to duplicate the display of GRC's QT GUI Sink.

The code for the Python script is in the python folder in the this project's github repository:

`digital_rf_read_fft_plot.py`

Both of these plots use 1024 points for the FFT with an average setting of 10. The plots were done with different random snapshots of data, and they are not identical. They do show the relatively constant FM broadcast spectrum captured by the SDRPlay device.

Jump to the next page, which shows GRC versus Python processed data.

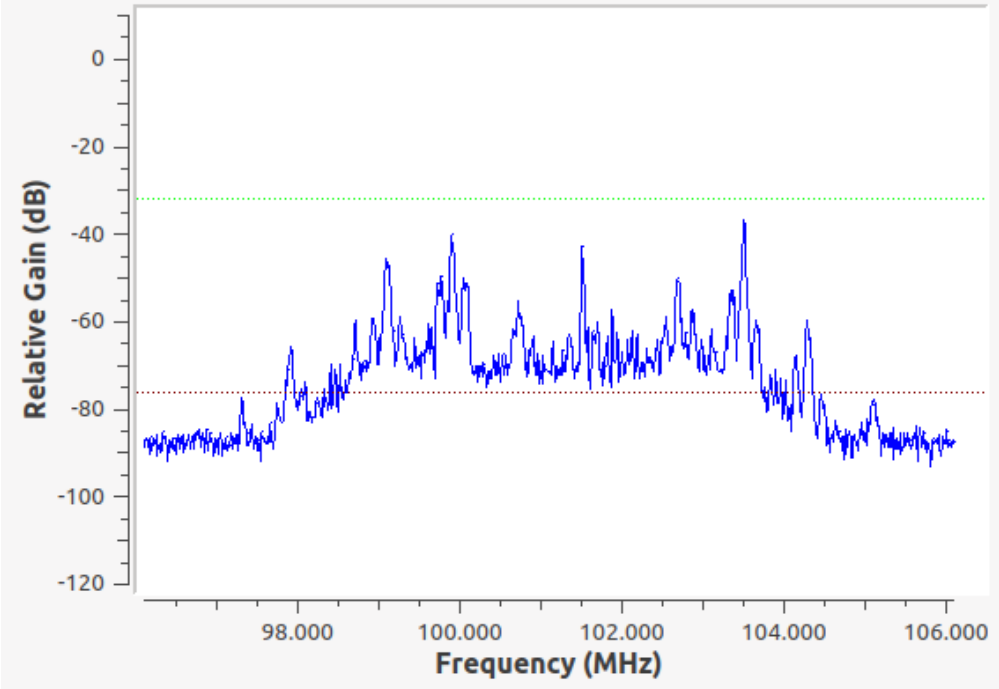

Figure 6.1: GRC's QT GUI Sink FM Broadcast Spectrum (Live data from SDRPlay)

Taking a chunk of data from the HDF5 file system, processing, and display with Python:

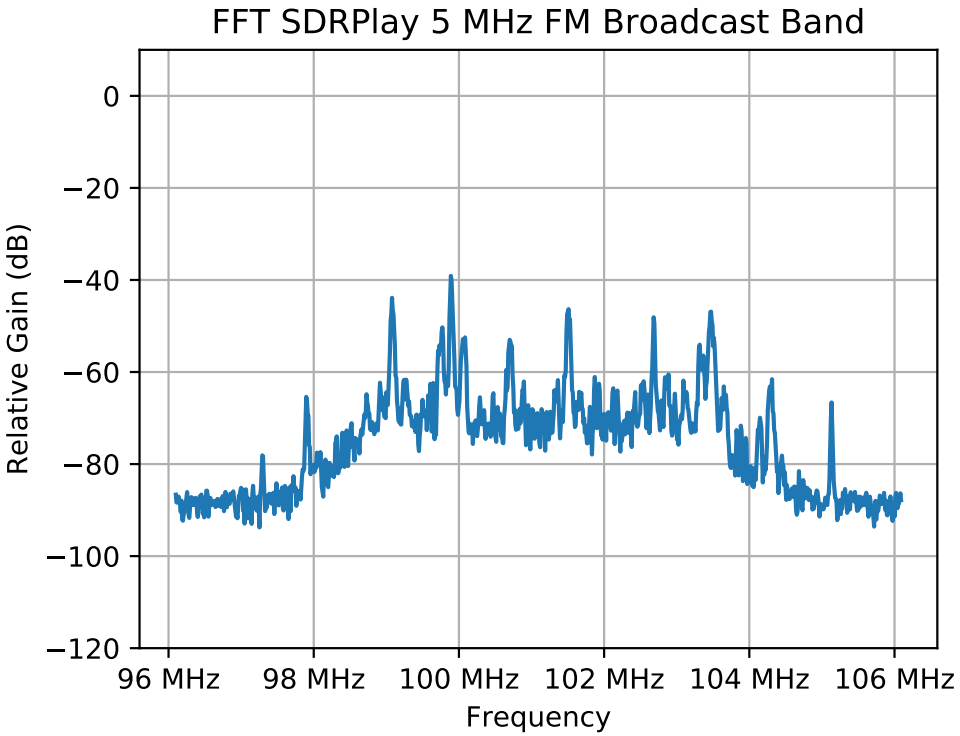

Figure 6.2: FFT of Digital RF HDF5 data, Points = 1024, Average = 10

Chapter 7

Conclusion and Comments

In spite of the short length of this report, the time consumed by coming up to speed on GNU Radio and Digital RF was a lot! Lots of hours were burned experimenting with GNU Radio especially.

But it was worth it! GNU Radio is a complex software system, and once you get past the installation process, it goes pretty smoothly. It didn't crash or otherwise break my computer. Very impressive!

Digital RF worked flawlessly. With the included GRC source and sink components, it was really quite simple to use. The goal of reading and writing large quantities of complex RF data was achieved.

The SDRPlay receiver worked well, at least on the FM broadcast band. I don't have a good HF antenna, so I can't comment on it's performance on lower frequencies. The integration into GNU Radio is very new. I did see some problems with having to re-insert the USB connection after stopping and re-starting GRC. This didn't happen on all the machines tried.

It was quite awe inspiring to see a gigabyte of data being written to the SSD drive in a few seconds! It's another technological re-calibration we will all have to make, as the gigabyte is the new megabyte. Soon a terabyte will be a trivial bit of data.

Python + Numpy + Scipy + Pandas is awesome! I wonder how much exposure the amateur radio community has had to these tools?

Next project! Install a decent loop antenna in the noisy environment I reside in, and give HF radio analysis a try.

Appendix A

Links to Resources

A collection of links to resources related to this project.

The github repository for this project.

https://github.com/Greg-R/explore_digital_rf

TAPR is a community that provides leadership and resources to radio amateurs for the purpose of advancing the radio art.

https://tapr.org/

The TAPR 2018 Conference Sunday Seminar "Personal Space Weather Station" slides by Nathaniel Frissell.

http://hamsci.org/sites/default/files/pages/swstation/20181116_TAPR_Sunday_Seminar_Frissell_W2NAF.pdf

Ham Radio Science Citizen Investigation

http://hamsci.org/

Personal Space Weather Station

http://hamsci.org/basic-project/personal-space-weather-station

The Digital RF project encompasses a standardized HDF5 format for reading and writing of radio frequency data and the software for doing so.

https://github.com/MITHaystack/digital_rf

HDF5 is a data model, library, and file format for storing and managing data.

https://portal.hdfgroup.org/display/support

SDRPlay, the makers of the RSP2 software defined radio used in this project.

https://www.sdrplay.com/rsp2