Tutorial: Practical Use of SDR for Machine Learning in RF Environments

ACM-SE 2022

2022-04-18
12:30 – 15:30 **(Central Time)**

Neel Pandeya
National Instruments
Austin, Texas, USA
neel.pandeya@ni.com
neel.pandeya@ettus.com

Agenda

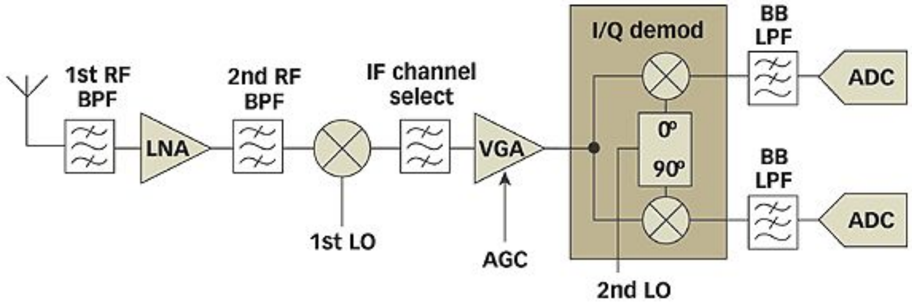
- Introduction to SDR concepts, architecture, applications
- Overview of USRP B200, B210, B200mini
- Overview of SDR toolchains
- Overview of Radio Transport Protocols and Wireshark
- Overview of I/Q Data Rates and Sampling Rates
- Introduction to UHD
    - Building, installing, and configuring UHD on Linux
    - Various UHD Utility Programs
    - Using the UHD API from C++ and Python
    - Packet Flow Errors
- Introduction to GNU Radio
    - Building, installing, and configuring GNU Radio on Linux
    - Various GNU Radio Utility Programs
    - Using GRC, and creating and running flowgraphs
    - Examples with DTMF, filters, etc.
- Record & Playback of Signals
    - I/Q data formats, Digital RF, SigMF
- Introduction to GQRX
    - Building, installing, and configuring GQRX on Linux
    - Spectrum monitoring
    - Demo of gr-paint
- Implementing an FM receiver and transmitter in GNU Radio and gr-rds
- Technical Resources, Getting Help & Technical Support, Upcoming Events

What is Software-Defined Radio (SDR)

- A radio in which some or all of the physical-layer functions are implemented in software running on a microprocessor (CPU) and/or on an FPGA
- Physical-layer algorithms from DSP and communications theory run as real-time software on a CPU and/or FPGA
- Software can run on an embedded DSP chip (e.g., Analog Devices TigerSHARC, Texas Instruments C6400) or a general-purpose CPU (e.g., Intel x86, ARM Cortex-M)
- Joe Mitola first coined the term “SDR” in an IEEE paper 1991

SDR Architecture

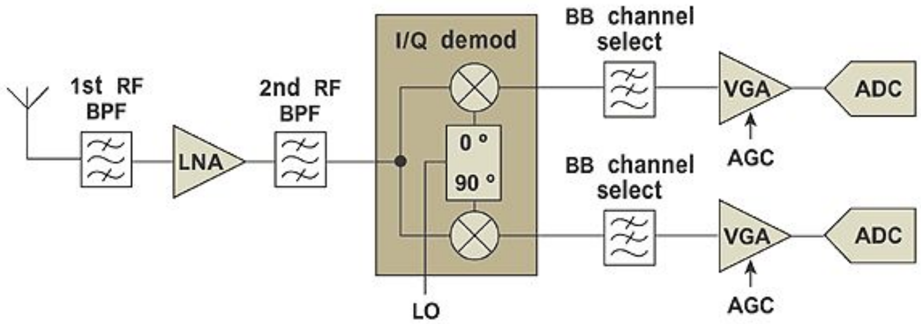
- Most radios use the classic superheterodyne receiver architecture
  - The RF signal from the antenna is mixed with a local oscillator to produce an intermediate frequency (IF) signal
  - The IF signal is a fixed lower-frequency signal, which is then filtered and further mixed (downconverted) to baseband



1. This block diagram shows a simplified superheterodyne receiver.

SDR Architecture

- Most SDR uses a direct-conversion receiver (DCR) architecture
  - Also called Zero-IF receiver, and homodyne receiver
  - Eliminates the intermediate frequency (IF) by translating the band of interest directly to baseband
  - The frequency of the LO is set to the same frequency as the transmitted/desired RF signal



2. This block diagram represents a simplified version of a direct-conversion receiver.

SDR Architecture

- Quadrature Sampling and I/Q Data
  - “I” is the in-phase (not shifted) data component
  - “Q” is the quadrature-phase (shifted by 90 degrees) data component
- Why do we use I/Q Data?
  - To fully determine the frequency and phase of a signal, and to be able to distinguish between positive and negative frequencies (needed for digitally processing the signal)
- Why 90 degrees?
  - So that the two components are orthogonal, meaning that their correlation is zero, further meaning that if the cosine signal is multiplied with the sine signal, and then the summation of the result is taken, this sum will be zero (the integral of the product of the sine and the cosine is zero).
  - A change in one component does not affect the other component

SDR Architecture

- Nyquist-Shannon Sampling Theorem
  - $f_s > 2 * f_{max}$
  - You must sample at least at twice the bandwidth of the signal,
    at least at twice the highest frequency component
  - If the sampling rate is lower than twice the bandwidth of the signal, then there will be *aliasing*, and information will be lost (the signal will likely be "*damaged*")
  - Due to the quadrature sampling used in the USRP devices, the sampling rate can be equal to the signal bandwidth
  - Often, wireless standards will prescribe or require specific sampling rates, above the minimum sampling rate required

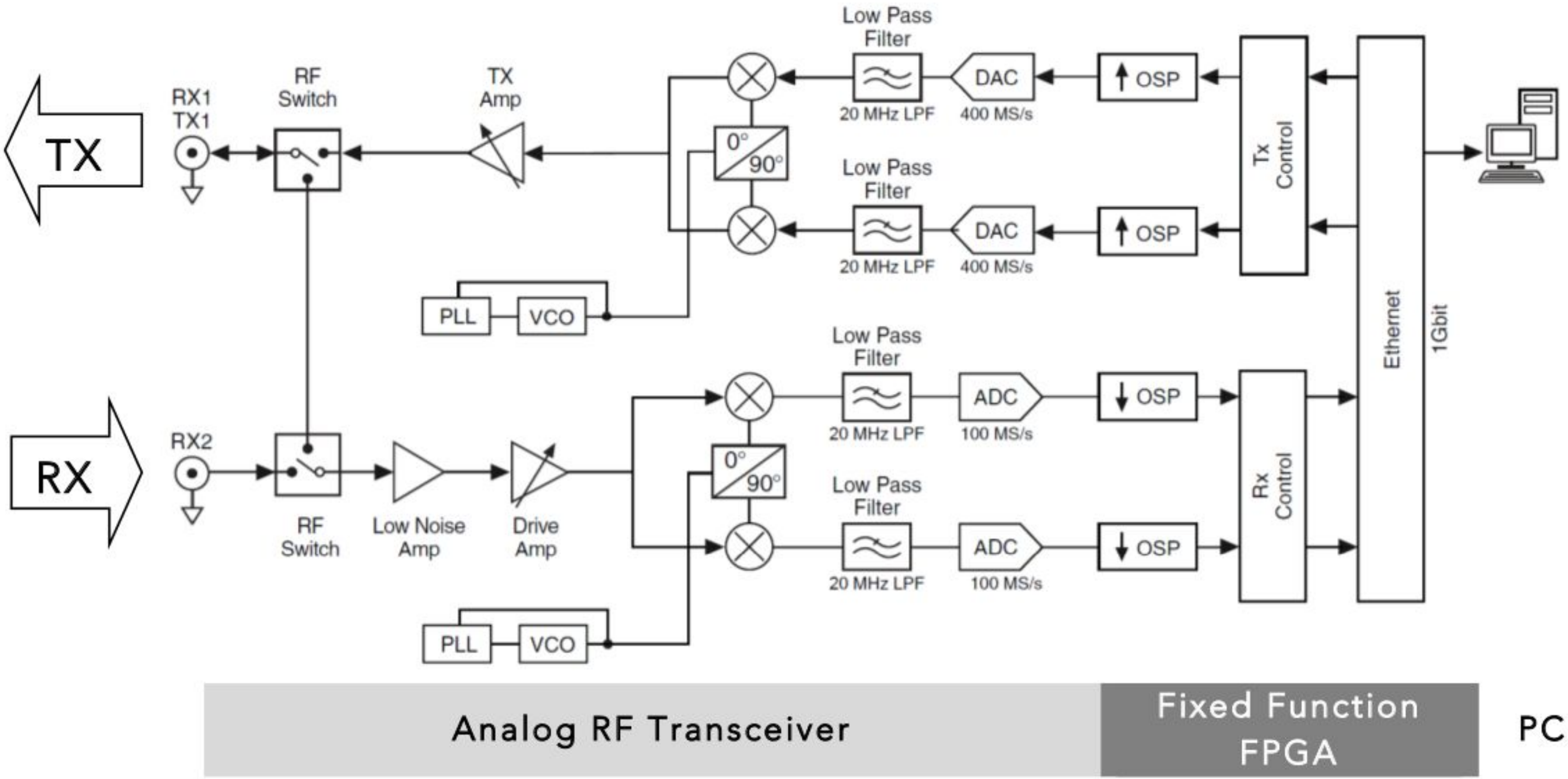
SDR Architecture

- Fast Fourier Transform (FFT)
  - An FFT is an algorithm that computes the Discrete Fourier Transform (DFT) of a signal, or its inverse (IDFT)
  - Fourier analysis converts a signal from the “time domain” to a representation in the frequency domain, and vice-versa
  - When we plot the FFT of a signal on a graph, it shows the spectrum of a signal, which is a plot of all the sine waves and cosine waves that constitute a signal, and this is called the spectrum of the signal
  - The sine waves and cosine waves are the fundamental building blocks of the signal (i.e., all signals can be constructed using sine waves and cosine waves), and the FFT plot is a visualization of those constituent and specific sine waves and cosine waves, at specific frequencies and at specific magnitudes

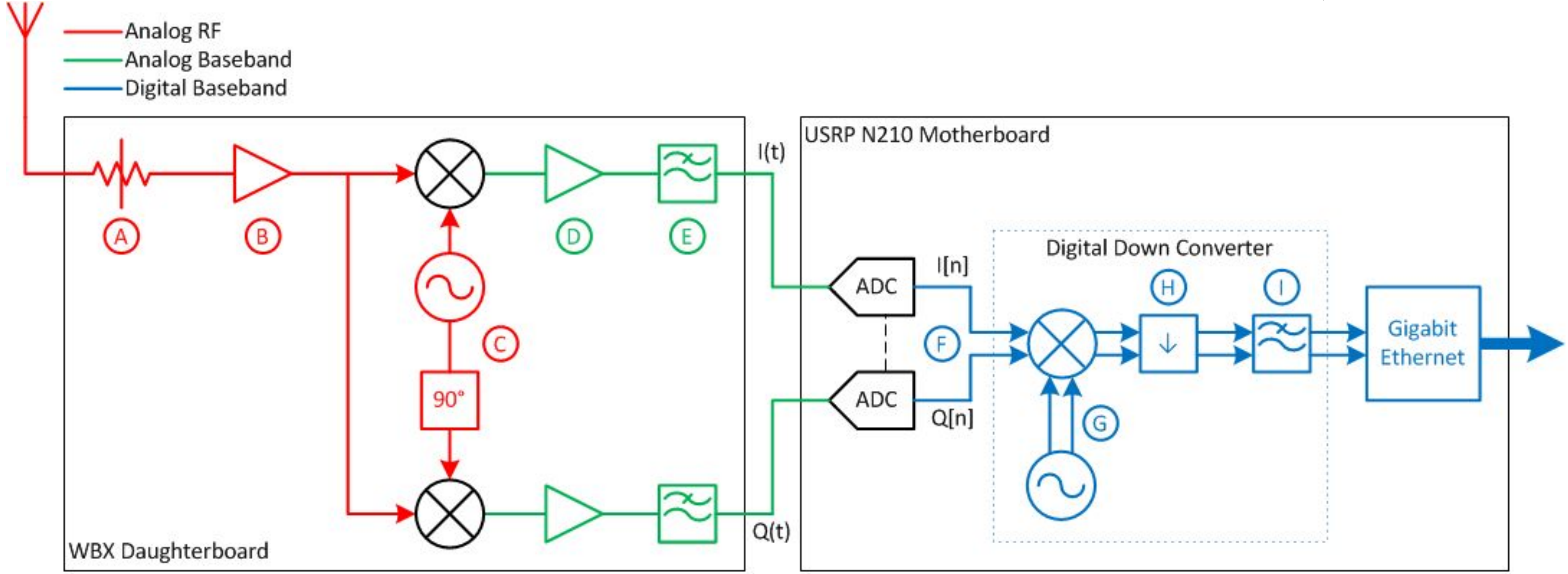
SDR Architecture

- Decibel (dB)
  - dB is a relative unit of measurement that expresses the ratio of two values *on a logarithmic scale*
  - dB is a *relative* value of power, not an *absolute* value of power, so it is dimensionless
  - $Ratio_{dB} = 10 * log_{10}(P_{measured} / P_{reference})$
  - When something doubles, it changes by +3 dB
  - When something halves, it changes by -3 dB
  - When something increases by a factor of ten, it changes by +10 dB
  - When something decreases by a factor of ten, it changes by -10 dB
  - When something increases by a factor of one hundred, it changes by +20 dB
  - 0 dB is equal to the reference value (i.e., 0 dBm is equal to one mW of power)
  - dBm is power with respect to 1 mW, so it is an *absolute* value of power, with units of mW
  - Why do we use dB?
    - Logarithmic scales are useful for measuring values that have a very large ranges of values
      - Used to measure sound level, earthquake intensity (Richter Scale), etc.

USRP Architecture



USRP Architecture



(A) Variable Attenuator: 0.5 dB steps up to 31.5 dB

(B) RF Amplifier: 13.2 dB at 2.0 GHz

(C) Wideband Frequency Synthesizer: 50 MHz to 2.2 GHz

(D) Baseband Amplifier: 18dB

(E) Lowpass Filter: 50 MHz

(F) Dual ADC: 100MSPS

(G) Numerically Controlled Oscillator (NCO) and CORDIC

(H) Programmable decimation by N=2(1..256)

(I) Halfband Filter

Why Use SDR?

- Traditional radios are hard-wired to specific frequency bands and communication protocols
    - Fixed-function, Black Box
    - Can’t be easily modified, can’t easily access internal values and states
- SDR provides:
    - Flexibility
    - Upgradability
    - Reconfigurability
    - Lower Cost

Applications of SDR

- Voice-band Soft-modems / WinModems in 1990s and 2000s
- Cellular handsets (baseband processors such as Qualcomm Snapdragon, MediaTek, etc.)
- Cellular 4G/LTE and 5G/NR basestations (Eurecom OpenAirInterface (OAI), SRS srsRAN, Amarisoft)
- Cellular protocol stack emulation (2G/GSM, 3G/WCDMA, 4G/LTE, 5G/NR)
- GPS Receivers and Simulators
- Adaptive Radio and Cognitive Radio
- Satellite Communications (Ground Stations)
- Wireless Security Research
- Spectrum Monitoring
- Waveform Prototyping
- Wireless Systems Testing / Wireless Testbeds
- Radio Astronomy
- Drone Communications, Drone Detection, Drone Defense
- Direction Finding / Angle-of-Arrival
- Phased Arrays, Beam-forming and Beam-steering, MIMO Systems

AI and ML have a role in all of these applications spaces!

Software Toolchains for SDR & USRP

- Processing can be either real-time or off-line / post-processing
- C++ and Python with the the USRP Hardware Driver (UHD) API *(open-source)*
- GNU Radio (Python, NumPy, SciPy, Matplotlib, etc.) *(open-source)*
- LabVIEW™ (National Instruments)
- MATLAB™ and Simulink™ (The MathWorks)
- Application-Specific:
  - Cellular: Eurecom OpenAirInterface (OAI), SRS srsRAN, Amarisoft
  - GPS: GNSS-SDR, GPS-SDR-Sim, Skydel Solutions SDX
  - Spectrum Monitoring: Fosphor, SDR++, GQRX
  - Amateur Radio: HDSDR, SDR#, SDR-Console

USRP Background

- About Ettus Research:
    - Founded in 2004 by Matt Ettus
    - Acquired by National Instruments in 2010
    - Offices in Santa Clara, California, USA; Austin, Texas, USA; Dresden, Germany
    - “USRP” is an acronym for *Universal Software Radio Peripheral*
- USRP Device Families:
    - B-series (B200, B210, B200mini): USB 3.0 host interface
    - N-series (N200, N210, N300/N310, N320/N321): 1 Gbps Ethernet host interface
    - X-series (X300, X310, X410): 1, 10, 100 Gbps Ethernet host interface
    - E-series (E310, E312, E313, E320): Embedded stand-alone SDR with ARM CPU

USRP B200

Ettus Research™
A National Instruments Company

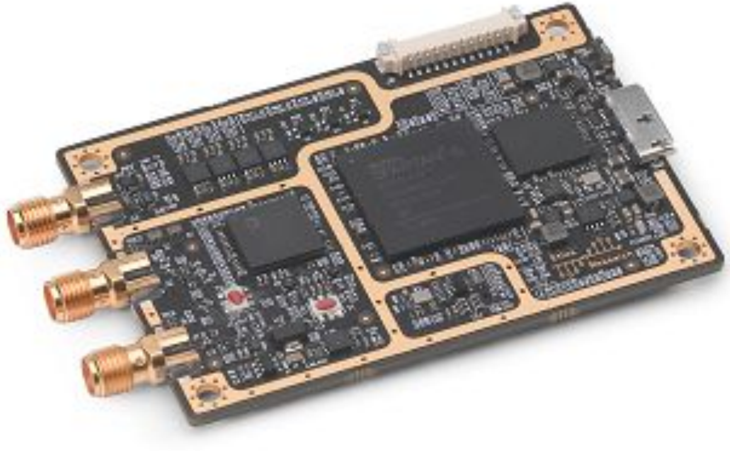
- Xilinx Spartan 6 XC6SLX75 FPGA
- Analog Devices AD9364 RFIC direct-conversion transceiver
- Frequency range: 70 MHz to 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- Maximum sampling rate of 61.44 Msps
- 1 Tx channel & 1 Rx channel
- USB 3.0 connectivity
- Optional GPSDO module

USRP B210

- Xilinx Spartan 6 XC6SLX150 FPGA
- Analog Devices AD9361 RFIC direct-conversion transceiver
- Frequency range: 70 MHz to 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- Maximum sampling rate of 61.44 Msps
- 2 Tx channels & 2 Rx channels
- USB 3.0 connectivity
- Optional GPSDO module

USRP B200mini

- Xilinx Spartan-6 XC6SLX75 FPGA
- Analog Devices AD9364 RFIC direct-conversion transceiver
- Frequency range: 70 MHz to 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- Maximum sampling rate of 61.44 Msps
- 1 Tx channel & 1 Rx channel
- USB 3.0 connectivity
- Powered from the USB 3.0 bus
- Size of a business card or credit card



Sampling Rates

- Integer decimation of the Master Clock Rate (MCR)
  - Even decimation rate preferred
  - Odd decimation rate allowed but with warning of CIC filter roll-off attenuation
  - For B200, B210, B200mini:
    - All based on AD9361
    - MCR can be anything between 1 MHz and 61.44 MHz (30.76 MHz in 2x2)
    - Decimation rates between 1 and 1024

I/Q Data Rates

- On the USRP B200, B210, B200mini, the I/Q data samples can be:
  - 16-bit I, 16-bit Q, for a total of 4 bytes per complex sample
  - 12-bit I, 12-bit Q, for a total of 3 bytes per complex sample
    - The ADC & DAC on the AD9361 are 12 bits anyway, so no loss of data or dynamic range
- USB 2.0 is 480 Mbits/s (60 MB/s) theoretical, so ~35 MB/s practical throughput, or ~8 Msps
- USB 3.0 is 5 Gbits/s (625 MB/s) theoretical, so ~350 MB/s practical throughput, or ~80 Msps
- 1 GbE is 1000 Mbits/sec (125 MB/sec) theoretical, so ~25 Msps, practical throughput
- 10 GbE is 10000 Mbits/sec (1250 MB/sec) theoretical, so ~250 Msps practical throughput
- Consider the load of the I/Q data rate on the transport, the CPU, and the disk
- Example: LTE signal, 20 MHz channel bandwidth
  - 30.72 Msps sampling rate, per 3GPP specifications
  - At 4 bytes per complex sample, the data rate is 122.88 Mbytes/s

USRP Hardware Driver (UHD)

- Provides a single, common interface (API) for all USRP devices
- Host-side software driver running in user-space
- Open-source and hosted on GitHub
- Cross-platform (Windows, macOS, Linux)
- Four components: host-side software; FPGA; MPM; firmware
- **`https://github.com/EttusResearch`**

USRP Hardware Driver (UHD)

Ettus Research™
A National Instruments Company

Application			
LabVIEW	**C++**	**GNU Radio Python / GRC / C++**	**Matlab**
UHD Driver			
Windows	**macOS**	**Linux**	**Embedded Linux**
Hardware Motherboard (FPGA) Daughterboard Antenna			

UHD Version Numbering

- UHD and GNU Radio use a modified semantic version numbering (`major.API.ABI.patch`)
  - MAJOR version as necessitated by product generation & architecture
  - API version, incremented when incompatible API changes are made
  - ABI version, incremented when incompatible ABI changes are made
  - PATCH version, incremented when backwards-compatible bug fixes are made
- The API number changes whenever there is any change to the API
- The ABI pertains to how external applications communicate with (link to) the UHD library
- The patch number is incremented when patches are made, typically for bug fixes

Radio Transport Protocols

Radio transport protocols are used to exchange I/Q samples (or other items) between host computer and USRP devices over Ethernet and USB

The USRP B200, B210, B200mini use the **CHDR** (compressed header) protocol, which is based on VITA-49.2

It is pronounced like the cheese "cheddar"

I/Q data traffic can be used in Wireshark, and there is a dissector for CHDR packets

Radio Transport Protocols

The CHDR packet:

Address (Bytes)	Length (Bytes)	Payload
0	8	Compressed Header (CHDR)
8	8	Fractional Time (Optional!)
8/16	-	Data

Radio Transport Protocols

The 64 bits in the compressed header have the following meaning:

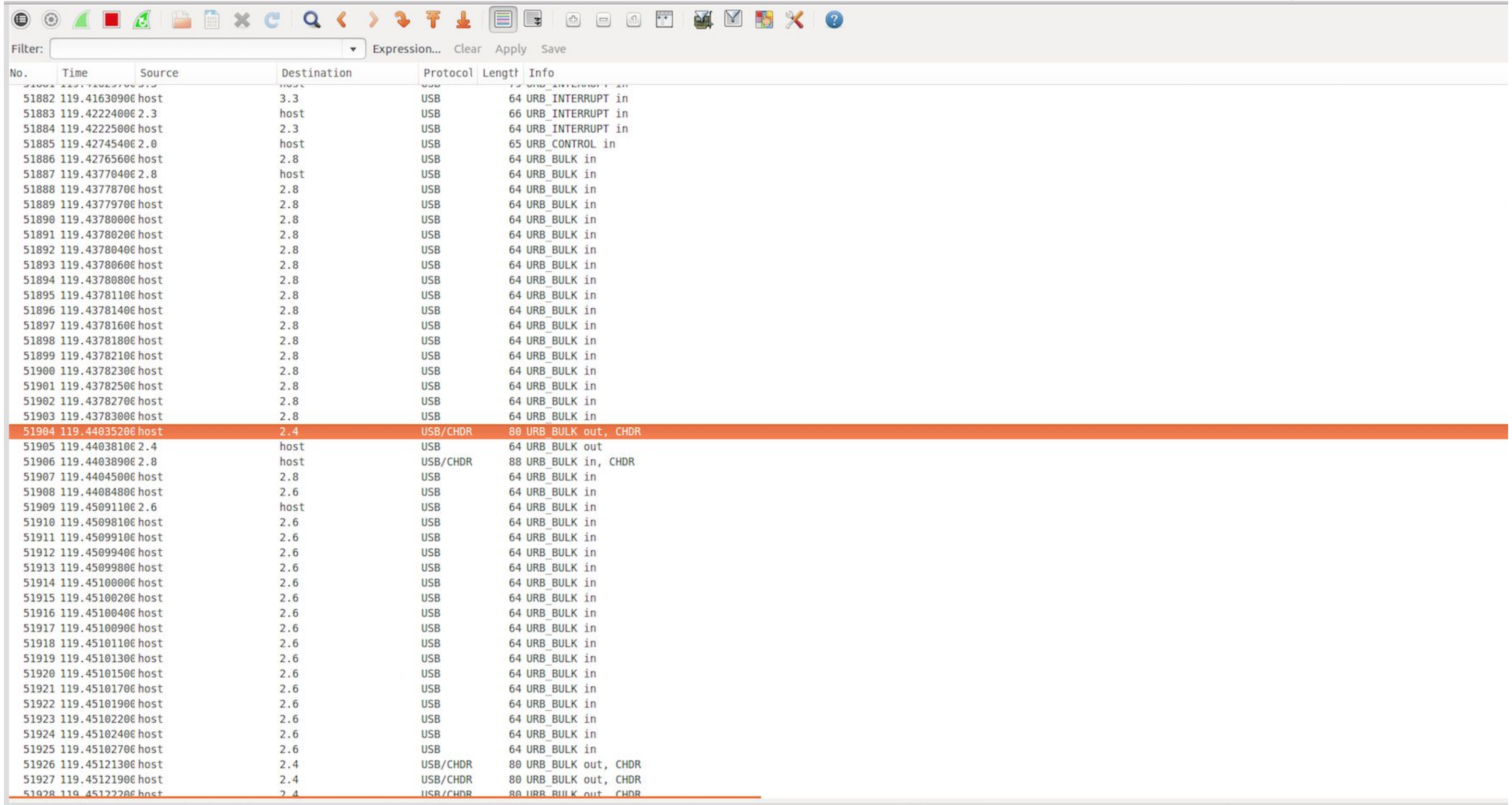
Bits	Meaning
63:62	Packet Type
61	Has fractional time stamp (1: Yes)
60	End-of-burst or error flag
59:48	12-bit sequence number
47:32	Total packet length in Bytes
31:0	Stream ID (SID)

Radio Transport Protocols

The packet type is determined mainly by the first two bits, although the EOB or error flag are also taken into consideration:

Bit 63	Bit 62	Bit 60	Packet Type
0	0	0	Data
0	0	1	Data (End-of-burst)
0	1	0	Flow Control
1	0	0	Command Packet
1	1	0	Command Response
1	1	1	Command Response (Error)

Viewing I/Q Traffic (CHDR) in Wireshark



No.	Time	Source	Destination	Protocol	Length	Info
51882	119.4163090	host	3.3	USB	64	URB_INTERRUPT in
51883	119.4222400	2.3	host	USB	66	URB_INTERRUPT in
51884	119.4222500	host	2.3	USB	64	URB_INTERRUPT in
51885	119.4274540	2.0	host	USB	65	URB_CONTROL in
51886	119.4276560	host	2.8	USB	64	URB_BULK in
51887	119.4377040	2.8	host	USB	64	URB_BULK in
51888	119.4377870	host	2.8	USB	64	URB_BULK in
51889	119.4377970	host	2.8	USB	64	URB_BULK in
51890	119.4378000	host	2.8	USB	64	URB_BULK in
51891	119.4378020	host	2.8	USB	64	URB_BULK in
51892	119.4378040	host	2.8	USB	64	URB_BULK in
51893	119.4378060	host	2.8	USB	64	URB_BULK in
51894	119.4378080	host	2.8	USB	64	URB_BULK in
51895	119.4378110	host	2.8	USB	64	URB_BULK in
51896	119.4378140	host	2.8	USB	64	URB_BULK in
51897	119.4378160	host	2.8	USB	64	URB_BULK in
51898	119.4378180	host	2.8	USB	64	URB_BULK in
51899	119.4378210	host	2.8	USB	64	URB_BULK in
51900	119.4378230	host	2.8	USB	64	URB_BULK in
51901	119.4378250	host	2.8	USB	64	URB_BULK in
51902	119.4378270	host	2.8	USB	64	URB_BULK in
51903	119.4378300	host	2.8	USB	64	URB_BULK in
51904	119.4403520	host	2.4	USB/CHDR	80	URB_BULK out, CHDR
51905	119.4403810	2.4	host	USB	64	URB_BULK out
51906	119.4403890	2.8	host	USB/CHDR	88	URB_BULK in, CHDR
51907	119.4404500	host	2.8	USB	64	URB_BULK in
51908	119.4408480	host	2.6	USB	64	URB_BULK in
51909	119.4509110	2.6	host	USB	64	URB_BULK in
51910	119.4509810	host	2.6	USB	64	URB_BULK in
51911	119.4509910	host	2.6	USB	64	URB_BULK in
51912	119.4509940	host	2.6	USB	64	URB_BULK in
51913	119.4509980	host	2.6	USB	64	URB_BULK in
51914	119.4510000	host	2.6	USB	64	URB_BULK in
51915	119.4510020	host	2.6	USB	64	URB_BULK in
51916	119.4510040	host	2.6	USB	64	URB_BULK in
51917	119.4510090	host	2.6	USB	64	URB_BULK in
51918	119.4510110	host	2.6	USB	64	URB_BULK in
51919	119.4510130	host	2.6	USB	64	URB_BULK in
51920	119.4510150	host	2.6	USB	64	URB_BULK in
51921	119.4510170	host	2.6	USB	64	URB_BULK in
51922	119.4510190	host	2.6	USB	64	URB_BULK in
51923	119.4510220	host	2.6	USB	64	URB_BULK in
51924	119.4510240	host	2.6	USB	64	URB_BULK in
51925	119.4510270	host	2.6	USB	64	URB_BULK in
51926	119.4512130	host	2.4	USB/CHDR	80	URB_BULK out, CHDR
51927	119.4512190	host	2.4	USB/CHDR	80	URB_BULK out, CHDR
51928	119.4512220	host	2.4	USB/CHDR	80	URB_BULK out, CHDR

Viewing I/Q Traffic (CHDR) in Wireshark

```
▾Frame 51904: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 3
  Interface id: 3
  Encapsulation type: USB packets with Linux header and padding (115)
  Arrival Time: Apr 12, 2017 16:13:52.247684000 PDT
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1492038832.247684000 seconds
  [Time delta from previous captured frame: 0.002522000 seconds]
  [Time delta from previous displayed frame: 0.002522000 seconds]
  [Time since reference or first frame: 119.440352000 seconds]
  Frame Number: 51904
  Frame Length: 80 bytes (640 bits)
  Capture Length: 80 bytes (640 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: usb:chdr]
▾USB URB
  URB id: 0xffff8807c8b330c0
  URB type: URB_SUBMIT ('S')
  URB transfer type: URB_BULK (0x03)
 ▶Endpoint: 0x04, Direction: OUT
  Device: 2
  URB bus id: 4
  Device setup request: not relevant ('-')
  Data: present (0)
  URB sec: 1492038832
  URB usec: 247684
  URB status: Operation now in progress (-EINPROGRESS) (-115)
  URB length [bytes]: 16
  Data length [bytes]: 16
  [Response in: 51905]
  [bInterfaceClass: Vendor Specific (0xff)]
▾UHD CHDR
 ▶1000 .... = Header bits: 0x08
  .... 0000 0000 0000 = Sequence ID: 0
  Packet size: 16
 ▾Stream ID: 0.0.0.64 (0.0.0.64)
   Source device: 0
   Source endpoint: 0
   Destination device: 0
   Destination endpoint: 64
 ▾Response: 2000000000000000
   Status code: 32
   .... 0000 0000 0000 = Response to sequence ID: 0
  Payload: 2000000000000000

0000  c0 30 b3 c8 07 88 ff ff  53 03 04 02 04 00 2d 00   .0...... S.....-.
0010  b0 b4 ee 58 00 00 00 00  84 c7 03 00 8d ff ff ff   ...X.... ........
0020  10 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0040  10 00 00 80 40 00 00 00  20 00 00 00 00 00 00 00   ....@...  .......
```

Viewing I/Q Traffic (CHDR) in Wireshark

Ettus Research™
A National Instruments Company

No.	Time	Source	Destination	Protocol	Length	Info
32345	22.729325000	2.3	host	USB	66	URB_INTERRUPT in
32346	22.729326000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32347	22.729327000	host	2.3	USB	64	URB_INTERRUPT in
32348	22.729376000	host	2.6	USB	64	URB_BULK in
32349	22.731365000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32350	22.731427000	host	2.6	USB	64	URB_BULK in
32351	22.733409000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32352	22.733469000	host	2.6	USB	64	URB_BULK in
32353	22.735452000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32354	22.735480000	host	2.6	USB	64	URB_BULK in
32355	22.737497000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32356	22.737525000	host	2.6	USB	64	URB_BULK in
32357	22.739540000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32358	22.739578000	host	2.6	USB	64	URB_BULK in
32359	22.741585000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32360	22.741624000	host	2.6	USB	64	URB_BULK in
32361	22.743628000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32362	22.743656000	host	2.6	USB	64	URB_BULK in
32363	22.745673000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32364	22.745701000	host	2.6	USB	64	URB_BULK in
32365	22.747717000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32366	22.747756000	host	2.6	USB	64	URB_BULK in
32367	22.749761000	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32368	22.749800000	host	2.6	USB	64	URB_BULK in

Viewing I/Q Traffic (CHDR) in Wireshark

Ettus Research™
A National Instruments Company

```
▼ UHD CHDR
  ▶ 0010 .... = Header bits: 0x02
    .... 0000 0001 0101 = Sequence ID: 21
    Packet size: 8192
  ▼ Stream ID: 0.0.0.160 (0.0.0.160)
      Source device: 0
      Source endpoint: 0
      Destination device: 0
      Destination endpoint: 160
    Time: 124421747
    Payload: 0000ffff02000100ffffdff00000100fefffeff0100ffff...

0040  00 20 15 20 a0 00 00 00  00 00 00 00 73 86 6a 07   . ......  ....s.j.
0050  00 00 ff ff 02 00 01 00  ff ff fd ff 00 00 01 00   ........  ........
0060  fe ff fe ff 01 00 ff ff  ff ff fe ff fe ff 01 00   ........  ........
0070  fc ff fc ff fe ff 04 00  01 00 01 00 fb ff 01 00   ........  ........
0080  00 00 04 00 ff ff fe ff  fa ff 03 00 03 00 04 00   ........  ........
0090  01 00 00 00 fa ff 04 00  fc ff 05 00 fe ff 00 00   ........  ........
00a0  fe ff fe ff 02 00 ff ff  f9 ff f8 ff fc ff 05 00   ........  ........
00b0  fc ff 08 00 fc ff 02 00  00 00 00 00 fe ff 03 00   ........  ........
00c0  01 00 fe ff 02 00 03 00  fb ff 05 00 f7 ff 01 00   ........  ........

Sequence ID (chdr.seq), 2 bytes        Packets: 54045 · Displayed: 54045 (100.0%)
```

Viewing I/Q Traffic (CHDR) in Wireshark

Ettus Research™
A National Instruments Company

```
▼ UHD CHDR
  ▶ 0010 .... = Header bits: 0x02
    .... 0000 0001 0101 = Sequence ID: 21
    Packet size: 8192
  ▼ Stream ID: 0.0.0.160 (0.0.0.160)
      Source device: 0
      Source endpoint: 0
      Destination device: 0
      Destination endpoint: 160
    Time: 124421747
    Payload: 0000ffff02000100ffffdff00000100fefffeff0100ffff...

0040  00 20 15 20 a0 00 00 00  00 00 00 00 73 86 6a 07   . . ....  ....s.j.
0050  00 00 ff ff 02 00 01 00  ff ff fd ff 00 00 01 00   ........  ........
0060  fe ff fe ff 01 00 ff ff  ff ff fe ff fe ff 01 00   ........  ........
0070  fc ff fc ff fe ff 04 00  01 00 01 00 fb ff 01 00   ........  ........
0080  00 00 04 00 ff ff fe ff  fa ff 03 00 03 00 04 00   ........  ........
0090  01 00 00 00 fa ff 04 00  fc ff 05 00 fe ff 00 00   ........  ........
00a0  fe ff fe ff 02 00 ff ff  f9 ff f8 ff fc ff 05 00   ........  ........
00b0  fc ff 08 00 fc ff 02 00  00 00 00 00 fe ff 03 00   ........  ........
00c0  01 00 fe ff 02 00 03 00  fb ff 05 00 f7 ff 01 00   ........  ........

Stream ID (chdr.sid), 4 bytes          Packets: 56369 · Displayed: 56369 (100.0%)
```

The UHD Repository on GitHub

`host/`
This folder contains the source code for the host-side driver

`firmware/`
This folder contains the source code for all microcontrollers in USRP hardware

`fpga/`
This folder contains the source code and build scripts for the USRP FPGAs

`mpm/`
This folder contains the source code for the Module Peripheral Manager (MPM) for embedded USRP devices

`images/`
This folder contains tools for downloading the USRP FPGA images, which are located in the `/usr/local/share/uhd/images` folder by default

`tools/`
This folder contains additional tools and utility programs

Installing UHD from Source Code

1. `sudo apt-get install libboost-all-dev libusb-1.0-0-dev python-mako doxygen python-docutils cmake build-essential libncurses5 libncurses5-dev`
2. `mkdir ~/workarea; cd ~/workarea`
3. `git clone git://github.com/EttusResearch/uhd.git`
4. `cd uhd/`
5. `git checkout v4.1.0.5`
6. `cd host/`
7. `mkdir build && cd build`
8. `cmake ../`
9. `make -j4`
10. `make test`
11. `sudo make install`
12. `sudo ldconfig`

Installing UHD from Binary Package

- Binary packages available on Ubuntu Launchpad PPA
- Recommend building from source code
  - Much more flexible when doing development
- The binary packages are less flexible and are often older or out-of-date
  - Use when doing deployment

Post-Installation Steps

- Add this line to your **`$HOME/.bashrc`** file, and *source* it, or logout and log back in:

```
export LD_LIBRARY_PATH=/usr/local/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

- On Linux, **`udev`** handles USB plug and unplug events. The following commands install a udev rule so that non-root users may access the device. Without this, you will not see the radio as a normal user. This step is only necessary for devices that use USB to connect to the host computer, such as the B200, B210, and B200mini.

```
cd <path-to-uhd-repository>/uhd/host/utils
sudo cp uhd-usrp.rules /etc/udev/rules.d/
sudo udevadm control --reload-rules
sudo udevadm trigger
```

- For USRP devices that use Ethernet to connect to the host computer, such as the N200, N210, X300, X310, set the IP address of your system to 192.168.10.1, with a netmask of 255.255.255.0. The default IP address of the USRP is 192.168.10.2 (for 1 GbE), and 192.168.40.2 (for 10 GbE), with a netmask of 255.255.255.0.
- Use Network Manager GUI (in Unity, KDE, GNOME, Xfce, etc.) to set the IP address. If you set the IP address from the command line with **`ifconfig`**, then Network Manager may probably overwrite this.

UHD Utility - `uhd_images_downloader`

`sudo /usr/local/lib/uhd/utils/uhd_images_downloader.py`

```
user@host:~$ sudo /usr/local/lib/uhd/utils/uhd_images_downloader.py
Images destination:      /usr/local/share/uhd/images
Downloading images from: http://files.ettus.com/binaries/images/uhd-images_003.009.002-release.zip
Downloading images to:   /tmp/tmpGYYPwE/uhd-images_003.009.002-release.zip
26296 kB / 26296 kB (100%)

Images successfully installed to: /usr/local/share/uhd/images
user@host:~$
```

UHD Utility - `uhd_images_downloader`

Ettus Research™
A National Instruments Company

```
user@host:~$ tree /usr/local/share/uhd/images/
/usr/local/share/uhd/images/
├── 003.009.002.tag
├── bit
│   ├── usrp_n200_r3_fpga.bit
│   ├── usrp_n200_r4_fpga.bit
│   ├── usrp_n210_r3_fpga.bit
│   └── usrp_n210_r4_fpga.bit
├── LICENSE
├── octoclock_bootloader.hex
├── octoclock_r4_fw.hex
├── usrp1_fpga_4rx.rbf
├── usrp1_fpga.rbf
├── usrp1_fw.ihx
├── usrp2_fpga.bin
├── usrp2_fw.bin
├── usrp_b100_fpga_2rx.bin
├── usrp_b100_fpga.bin
├── usrp_b100_fw.ihx
├── usrp_b200_fpga.bin
├── usrp_b200_fw.hex
├── usrp_b200mini_fpga.bin
├── usrp_b205mini_fpga.bin
├── usrp_b210_fpga.bin
├── usrp_e100_fpga_v2.bin
├── usrp_e110_fpga.bin
```

```
├── usrp_e310_fpga.bit
├── usrp_e310_fpga_idle.bit
├── usrp_e310_fpga_sg3.bit
├── usrp_e3xx_fpga_idle.bit
├── usrp_e3xx_fpga_idle_sg3.bit
├── usrp_n200_fw.bin
├── usrp_n200_r2_fpga.bin
├── usrp_n200_r3_fpga.bin
├── usrp_n200_r4_fpga.bin
├── usrp_n210_fw.bin
├── usrp_n210_r2_fpga.bin
├── usrp_n210_r3_fpga.bin
├── usrp_n210_r4_fpga.bin
├── usrp_x300_fpga_HGS.bit
├── usrp_x300_fpga_HGS.lvbitx
├── usrp_x310_fpga_HGS.bit
├── usrp_x310_fpga_HGS.lvbitx
└── winusb_driver
    ├── amd64
    │   ├── WdfCoInstaller01009.dll
    │   └── winusbcoinstaller2.dll
    ├── erllc_uhd_b100.inf
    ├── erllc_uhd_b200.inf
    ├── erllc_uhd_b200_reinit.inf
    ├── erllc_uhd_usrp1.inf
    └── x86
        ├── WdfCoInstaller01009.dll
        └── winusbcoinstaller2.dll

4 directories, 48 files
user@host:~$
```

UHD Utility - `uhd_find_devices`

Uses broadcast packets for device discovery.
Often blocked by routers, switches, firewalls.

View firewall settings with:
`sudo iptables -L`

```
user@host:~$ uhd_find_devices
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.002-0-gf18abe54

--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    type: usrp2
    addr: 192.168.10.2
    name:
    serial: F38688


user@host:~$
```

UHD Utility - uhd_usrp_probe


```
user@host:~$ uhd_usrp_probe
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.002-0-gf18abe54

-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
  _____________________________________________________
 /
|       Device: USRP2 / N-Series Device
|     _____________________________________________________
|    /
|   |       Mboard: N210r4
|   |   hardware: 2577
|   |   mac-addr: 00:80:2f:0a:d5:bd
|   |   ip-addr: 192.168.10.2
|   |   subnet: 255.255.255.255
|   |   gateway: 255.255.255.255
|   |   gpsdo: none
|   |   serial: F38688
|   |   FW Version: 12.4
|   |   FPGA Version: 11.1
|   |
|   |   Time sources: none, external, _external_, mimo
|   |   Clock sources: internal, external, mimo
|   |   Sensors: mimo_locked, ref_locked
|   |     _____________________________________________________
|   |    /
|   |   |       RX DSP: 0
|   |   |   Freq range: -50.000 to 50.000 MHz
|   |     _____________________________________________________
|   |    /
|   |   |       RX DSP: 1
|   |   |   Freq range: -50.000 to 50.000 MHz
|   |     _____________________________________________________
|   |    /
|   |   |       RX Dboard: A
|   |   |   ID: WBX, WBX + Simple GDB (0x0053)
|   |   |   Serial: 7f708b81
|   |   |     _____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 0
```

```
|   |   |    /
|   |   |   |       RX Frontend: 0
|   |   |   |   Name: WBXv2 RX+GDB
|   |   |   |   Antennas: TX/RX, RX2, CAL
|   |   |   |   Sensors: lo_locked
|   |   |   |   Freq range: 68.750 to 2200.000 MHz
|   |   |   |   Gain range PGA0: 0.0 to 31.5 step 0.5 dB
|   |   |   |   Bandwidth range: 40000000.0 to 40000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |     _____________________________________________________
|   |   |    /
|   |   |   |       RX Codec: A
|   |   |   |   Name: ads62p44
|   |   |   |   Gain range digital: 0.0 to 6.0 step 0.5 dB
|   |   |   |   Gain range fine: 0.0 to 0.5 step 0.1 dB
|   |     _____________________________________________________
|   |    /
|   |   |       TX DSP: 0
|   |   |   Freq range: -50.000 to 50.000 MHz
|   |     _____________________________________________________
|   |    /
|   |   |       TX Dboard: A
|   |   |   ID: WBX (0x0052)
|   |   |   Serial: b9e625d4
|   |   |     _____________________________________________________
|   |   |    /
|   |   |   |       TX Frontend: 0
|   |   |   |   Name: WBXv2 TX+GDB
|   |   |   |   Antennas: TX/RX, CAL
|   |   |   |   Sensors: lo_locked
|   |   |   |   Freq range: 68.750 to 2200.000 MHz
|   |   |   |   Gain range PGA0: 0.0 to 25.0 step 0.1 dB
|   |   |   |   Bandwidth range: 40000000.0 to 40000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |     _____________________________________________________
|   |   |    /
|   |   |   |       TX Codec: A
|   |   |   |   Name: ad9777
|   |   |   |   Gain Elements: None
```

UHD Arguments

Ettus Research™
A National Instruments Company

Most UHD applications and examples make use of the **`--args`** parameter to select specific devices

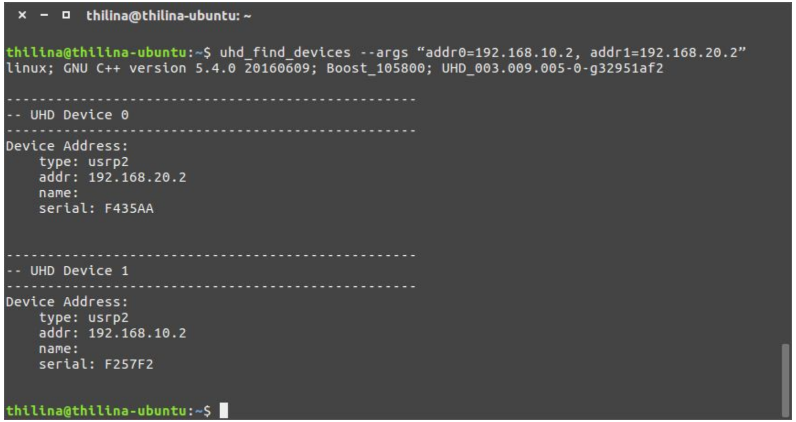
Common argument keys: **`serial`**, **`addr`**, **`resource`**, **`name`**, **`type`**, **`vid/pid`**.

`$ uhd_find_devices --args “addr=192.168.10.2”` *(for USRP N2xx / X3xx)*

`$ uhd_find_devices --args “type=b200,serial=xxxxxxx”` *(for B2xx)*

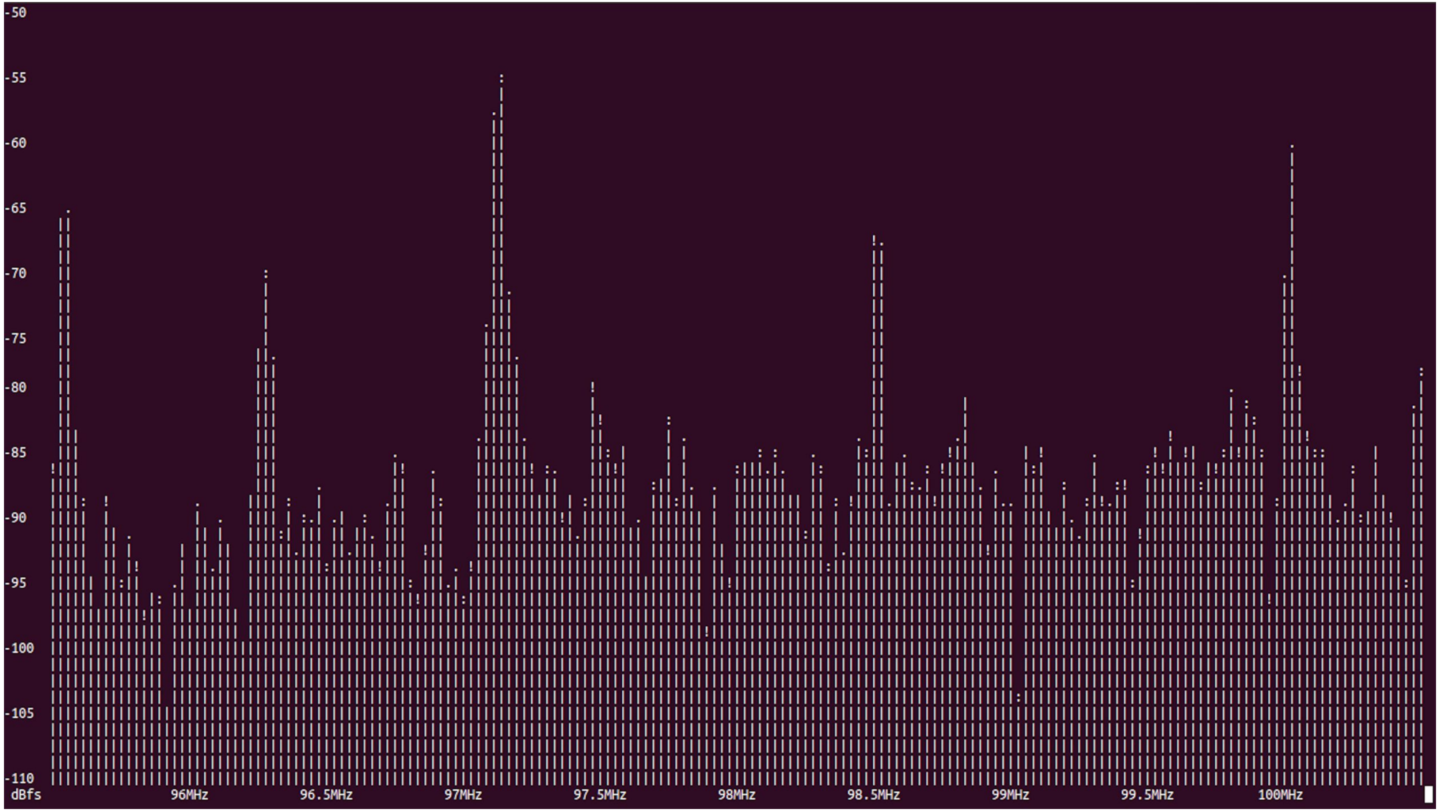
Note that multiple arguments are comma-delimited

This will return the devices at the specific IP address, and can be used to overcome previously mentioned network obstacles.



UHD Example Programs

`rx_ascii_art_dft --freq 98e6 --rate 1e6 --gain 20 --ref-lvl -50`



Verifying USRP using UHD

`benchmark_rate --rx_rate 10e6 --tx_rate 10e6`

```
user@host:/usr/local/lib/uhd/examples$ ./benchmark_rate --rx_rate 10e6 --tx_rate 10e6
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.002-0-gf18abe54


Creating the usrp device with: ...
-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
Using Device: Single USRP:
  Device: USRP2 / N-Series Device
  Mboard 0: N210r4
  RX Channel: 0
    RX DSP: 0
    RX Dboard: A
    RX Subdev: WBXv2 RX+GDB
  TX Channel: 0
    TX DSP: 0
    TX Dboard: A
    TX Subdev: WBXv2 TX+GDB

Testing receive rate 10.000000 Msps on 1 channels
Testing transmit rate 10.000000 Msps on 1 channels

Benchmark rate summary:
  Num received samples:    100116852
  Num dropped samples:     0
  Num overflows detected:  0
  Num transmitted samples: 100229019
  Num sequence errors:     0
  Num underflows detected: 0


Done!
```

Verifying USRP using UHD

Ettus Research™ A National Instruments Company

rx_samples_to_file --freq 98e6 --gain 20 --rate 1e6 usrp_samples.dat

```
user@host:~$ /usr/local/lib/uhd/examples/rx_samples_to_file --args="type=usrp2,addr=192.168.10.2" --freq 98e6 --gain 20 --rate 5e6 usrp_samples.dat
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.002-0-gf18abe54


Creating the usrp device with: type=usrp2,addr=192.168.10.2...
-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
Using Device: Single USRP:
  Device: USRP2 / N-Series Device
  Mboard 0: N210r4
  RX Channel: 0
    RX DSP: 0
    RX Dboard: A
    RX Subdev: WBXv2 RX+GDB
  TX Channel: 0
    TX DSP: 0
    TX Dboard: A
    TX Subdev: WBXv2 TX+GDB

Setting RX Rate: 5.000000 Msps...
Actual RX Rate: 5.000000 Msps...

Setting RX Freq: 98.000000 MHz...
Actual RX Freq: 98.000000 MHz...

Setting RX Gain: 20.000000 dB...
Actual RX Gain: 20.000000 dB...

Waiting for "lo_locked": ++++++++++ locked.

Press Ctrl + C to stop streaming...
^C
Done!

user@host:~$ ls -al usrp_samples.dat
-rw-rw-r-- 1 user user 307640000 Jan 20 10:16 usrp_samples.dat
user@host:~$ 
```

Verifying USRP using UHD

`tx_samples_from_file --freq 915e6 --rate 1e6 --gain 0 usrp_samples.dat`

```
user@host:~$ /usr/local/lib/uhd/examples/tx_samples_from_file --args="type=usrp2,addr=192.168.10.2" --freq 915e6 --rate 5e6 --gain 0 usrp_samples.dat
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.002-0-gf18abe54


Creating the usrp device with: type=usrp2,addr=192.168.10.2...
-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
-- Detecting internal GPSDO.... No GPSDO found
Using Device: Single USRP:
  Device: USRP2 / N-Series Device
  Mboard 0: N210r4
  RX Channel: 0
    RX DSP: 0
    RX Dboard: A
    RX Subdev: WBXv2 RX+GDB
  TX Channel: 0
    TX DSP: 0
    TX Dboard: A
    TX Subdev: WBXv2 TX+GDB

Setting TX Rate: 5.000000 Msps...
Actual TX Rate: 5.000000 Msps...

Setting TX Freq: 915.000000 MHz...
Actual TX Freq: 915.000000 MHz...

Setting TX Gain: 0.000000 dB...
Actual TX Gain: 0.000000 dB...

Checking TX: LO: locked ...

Done!

user@host:~$
```

UHD Utility Programs

- Default installation location is **`/usr/local/lib/uhd/utils`**
- **`uhd_config_info`**
  - Prints detailed UHD configuration information
- **`uhd_images_downloader`**
  - Downloads FPGA images for the current UHD version
- **`uhd_image_loader`**
  - Writes an FPGA image into the flash memory for the X300/X310 FPGA
- **`usrp_burn_mb_eeprom`**
  - Reading and writing motherboard EEPROM
- **`usrp_burn_db_eeprom`**
  - Reading and writing daughterboard EEPROM

UHD Example Programs

- Default installation location is **`/usr/local/lib/uhd/examples`**
- **`rx_ascii_art_dft`**
  - Creates ASCII/Ncurses FFT
  - **`./rx_ascii_art_dft --freq 98e6 --rate 5e6 --gain 20 --bw 5e6 --ref-lvl -50`**
- **`rx_samples_to_file`**
  - Saves samples to file
  - **`./rx_samples_to_file --freq 98e6 --rate 5e6 --gain 20 usrp_samples.dat`**
- **`tx_samples_from_file`**
  - Transmits samples from file
  - **`./tx_samples_from_file --freq 915e6 --rate 5e6 --gain 10 usrp_samples.dat`**
- **`benchmark_rate`**
  - Benchmarks interface with device
  - **`./benchmark_rate --rx_rate 10e6 --tx_rate 10e6`**
- **`tx_waveforms`**
  - Transmits specific waveform
  - **`./tx_waveforms --freq 915e6 --rate 5e6 --gain 0`**

Packet Flow Errors

- Packet flow errors printed in console/terminal as upper-case letters:
- Underrun on Tx (“U”):
  - Samples not being produced by the host application fast enough. CPU governor or other power management not configured correctly.
- Overrun on Rx (“O”):
  - Samples not being consumed by the host application fast enough. CPU governor or other power management not configured correctly.
- Sequence Error on Tx (“S”):
  - Network hardware failure. Check host NIC, cable, switch, etc. Frame size might not work with the current NIC's MTU.
- Dropped Packet on Rx (“D”):
  - Network hardware failure. Check host NIC, cable, switch, etc. PCIe bus on host cannot sustain throughput. CPU governor or other power management not configured correctly. Frame size might not work with the current NIC's MTU. Check "`ethtool -S <interface>`".
- Late Packet on Tx (“L”):
  - Samples are not being produced by user's application fast enough. CPU governor or other power management not configured correctly. Incorrect/invalid time_spec provided. Usually on MIMO.

Using UHD API

- The UHD API can be used from:
  - C++ (native)
  - Python 3
- For C++, can compile with:
  - GCC
  - LLVM/Clang
  - Microsoft Visual Studio
  - macOS Xcode
- Uses the CMake build system
  - An example `CMakeLists.txt` file provided for getting started with building custom stand-alone applications

Using UHD API from C++

```
#include <uhd/utils/thread_priority.hpp>
#include <uhd/utils/safe_main.hpp>
#include <uhd/usrp/multi_usrp.hpp>
#include <uhd/exception.hpp>
#include <uhd/types/tune_request.hpp>
#include <boost/program_options.hpp>
#include <boost/format.hpp>
#include <boost/thread.hpp>
#include <iostream>

int UHD_SAFE_MAIN(int argc, char *argv[]) {

...


 return EXIT_SUCCESS;
}
```

Using UHD API from C++

```
int UHD_SAFE_MAIN(int argc, char *argv[]) {
    uhd::set_thread_priority_safe();

    std::string device_args("type=b200");
    std::string subdev("A:0");
    std::string ant("TX/RX");
    std::string ref("internal");

    double rate(1e6);
    double freq(915e6);
    double gain(10);

    //create a usrp device
    std::cout << std::endl;
    std::cout << boost::format("Creating the usrp device with: %s...") % device_args << std::endl;
    uhd::usrp::multi_usrp::sptr usrp = uhd::usrp::multi_usrp::make(device_args);

    // Lock mboard clocks
    std::cout << boost::format("Lock mboard clocks: %f") % ref << std::endl;
    usrp->set_clock_source(ref);
```

Using UHD API from C++

Ettus Research™
A National Instruments Company

```
//always select the subdevice first, the channel mapping affects the other settings
std::cout << boost::format("subdev set to: %f") % subdev << std::endl;
usrp->set_rx_subdev_spec(subdev);
std::cout << boost::format("Using Device: %s") % usrp->get_pp_string() << std::endl;

//set the sample rate
if (rate <= 0.0) {
    std::cerr << "Please specify a valid sample rate" << std::endl;
    return ~0;
}

// set sample rate
std::cout << boost::format("Setting RX Rate: %f Msps...") % (rate / 1e6) << std::endl;
usrp->set_rx_rate(rate);
std::cout << boost::format("Actual RX Rate: %f Msps...") % (usrp->get_rx_rate() / 1e6) << std::endl << std::endl;

// set freq
std::cout << boost::format("Setting RX Freq: %f MHz...") % (freq / 1e6) << std::endl;
uhd::tune_request_t tune_request(freq);
usrp->set_rx_freq(tune_request);
std::cout << boost::format("Actual RX Freq: %f MHz...") % (usrp->get_rx_freq() / 1e6) << std::endl << std::endl;
```

Using UHD API from C++

```
    // set the rf gain
    std::cout << boost::format("Setting RX Gain: %f dB...") % gain << std::endl;
    usrp->set_rx_gain(gain);
    std::cout << boost::format("Actual RX Gain: %f dB...") % usrp->get_rx_gain() << std::endl << std::endl;

    // set the antenna
    std::cout << boost::format("Setting RX Antenna: %s") % ant << std::endl;
    usrp->set_rx_antenna(ant);
    std::cout << boost::format("Actual RX Antenna: %s") % usrp->get_rx_antenna() << std::endl << std::endl;

    return EXIT_SUCCESS;
}
```

Building UHD C++ Program

- Use the **`uhd/host/examples/init_usrp/CMakeLists.txt`** file as template
- Add the names of your C++ source files to the **`add_executable(…)`** section
- Put both modified **`CMakeLists.txt`** file and C++ file into an empty folder
- Create a "build" folder and invoke CMake the usual way:

```
mkdir build
cd build
cmake ../
make -j4
```

Building UHD C++ Program

- **init_usrp** example included as **~/ettus_workshop/examples/usrp_basic**

```
$ cd ~/ettus_workshop/examples/usrp_basic

$ mkdir build

$ cd build

$ cmake ..

$ make

$ ./usrp_basic

$ ldd ./usrp_basic
```

Building UHD C++ Program


```
### Make the executable ########################################################
add_executable(init_usrp init_usrp.cpp)

SET(CMAKE_BUILD_TYPE "Release")
MESSAGE(STATUS "******************************************************************************")
MESSAGE(STATUS "* NOTE: When building your own app, you probably need all kinds of different  ")
MESSAGE(STATUS "* compiler flags. This is just an example, so it's unlikely these settings    ")
MESSAGE(STATUS "* exactly match what you require. Make sure to double-check compiler and     ")
MESSAGE(STATUS "* linker flags to make sure your specific requirements are included.          ")
MESSAGE(STATUS "******************************************************************************")

# Shared library case: All we need to do is link against the library, and
# anything else we need (in this case, some Boost libraries):
if(NOT UHD_USE_STATIC_LIBS)
    message(STATUS "Linking against shared UHD library.")
    target_link_libraries(init_usrp ${UHD_LIBRARIES} ${Boost_LIBRARIES})
# Shared library case: All we need to do is link against the library, and
# anything else we need (in this case, some Boost libraries):
else(NOT UHD_USE_STATIC_LIBS)
    message(STATUS "Linking against static UHD library.")
    target_link_libraries(init_usrp
        # We could use ${UHD_LIBRARIES}, but linking requires some extra flags,
        # so we use this convenience variable provided to us
        ${UHD_STATIC_LIB_LINK_FLAG}
        # Also, when linking statically, we need to pull in all the deps for
        # UHD as well, because the dependencies don't get resolved automatically
        ${UHD_STATIC_LIB_DEPS}
    )
endif(NOT UHD_USE_STATIC_LIBS)

### Once it's built... #########################################################
# Here, you would have commands to install your program.
# We will skip these in this example.
```

GNU Radio

- Open-source framework for SDR and signal processing
- Block-based dataflow architecture
- Each block runs in its own thread
- Data flows through a graph called a Flowgraph
- Blocks are nodes in a Flowgraph, and perform operations and signal processing
- Signals normalized between -1.0 and +1.0
- Similar in concept to LabVIEW™ and Simulink™
- Running C++ and Python under-the-hood
- Can write code directly, or use the GNU Radio Companion (GRC) graphical tool
- Hosted on GitHub at **`https://github.com/gnuradio/gnuradio`**
- Homepage is **`http://gnuradio.org/`**

Installing GNU Radio from Source Code

Install Ubuntu 20.04 Dependencies:

```
sudo apt-get -y install git swig cmake doxygen build-essential libboost-all-dev libtool libusb-1.0-0
libusb-1.0-0-dev libudev-dev libncurses5-dev libfftw3-bin libfftw3-dev libfftw3-doc libcppunit-1.13-0v5
libcppunit-dev libcppunit-doc ncurses-bin cpufrequtils python-numpy python-numpy-doc python-numpy-dbg
python-scipy python-docutils qt4-bin-dbg qt4-default qt4-doc libqt4-dev libqt4-dev-bin python-qt4
python-qt4-dbg python-qt4-dev python-qt4-doc python-qt4-doc libqwt6abi1 libfftw3-bin libfftw3-dev
libfftw3-doc ncurses-bin libncurses5 libncurses5-dev libncurses5-dbg libfontconfig1-dev libxrender-dev
libpulse-dev swig g++ automake autoconf libtool python-dev libfftw3-dev libcppunit-dev libboost-all-dev
libusb-dev libusb-1.0-0-dev fort77 libsdl1.2-dev python-wxgtk3.0 git-core libqt4-dev python-numpy ccache
python-opengl libgsl-dev python-cheetah python-mako python-lxml doxygen qt4-default qt4-dev-tools
libusb-1.0-0-dev libqwt5-qt4-dev libqwtplot3d-qt4-dev pyqt4-dev-tools python-qwt5-qt4 cmake git-core wget
libxi-dev gtk2-engines-pixbuf r-base-dev python-tk liborc-0.4-0 liborc-0.4-dev libasound2-dev python-gtk2
libzmq-dev libzmq1 python-requests python-sphinx libcomedi-dev python-zmq tree
```

Installing GNU Radio from Source Code

Ettus Research™
A National Instruments Company

```
1.   cd ~/workarea
2.   git clone --recursive https://github.com/gnuradio/gnuradio.git
3.   cd gnuradio/
4.   git checkout v3.8.5.0
5.   mkdir build && cd build
6.   cmake ../
7.   make -j4
8.   sudo make install
9.   sudo ldconfig
```

Installing GNU Radio from Binary Package

- Binary packages available on Ubuntu Launchpad PPA
  - Recommend building from source code
    - Much more flexible when doing development
  - The binary packages are less flexible and are often older or out-of-date
    - Use when doing deployment

GNU Radio Utility Program

- Utility program to print detailed GNU Radio configuration information
  - **`gnuradio-config-info --version`** (or **`-v`**)
  - **`gnuradio-config-info --prefix`**
  - **`gnuradio-config-info --enabled-components`**
  - **`gnuradio-config-info --print-all`**

GNU Radio Examples

- Many examples included with GNU Radio installation
- Located at:

```
<install_path>/share/gnuradio/examples/

/usr/local/share/gnuradio/examples/
```

Dual-tone multi-frequency signaling (DTMF)

In-band telecommunication signaling system using the voice-frequency band over telephone lines between telephone equipment and other communications devices

The DTMF telephone keypad is laid out in a 4×4 matrix of push buttons in which each row represents the low frequency component and each column represents the high frequency component of the DTMF signal.

DTMF keypad frequencies

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

GNU Radio Dial Tone Example

- Dial Tone Example
  - Generates a PSTN dial tone
  - Does not use any hardware
  - Verifies that all libraries can be found, and the GR run-time is working
  - Run the following example:

    ```
    $ python ~/ettus_workshop/flowgraphs/dial_tone_basic.py
    ```

  - Flowgraph located at:

    ```
    ~/ettus_workshop/flowgraphs/dial_tone_basic.grc
    ```

Dial Tone Example: Python Code

Location: ~/ettus_workshop/flowgraphs/dial_tone_basic.py

```
from gnuradio import analog
from gnuradio import audio
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from optparse import OptionParser


class dial_tone_basic(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "Dial Tone Basic")

        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 32000
```

Dial Tone Example: Python Code

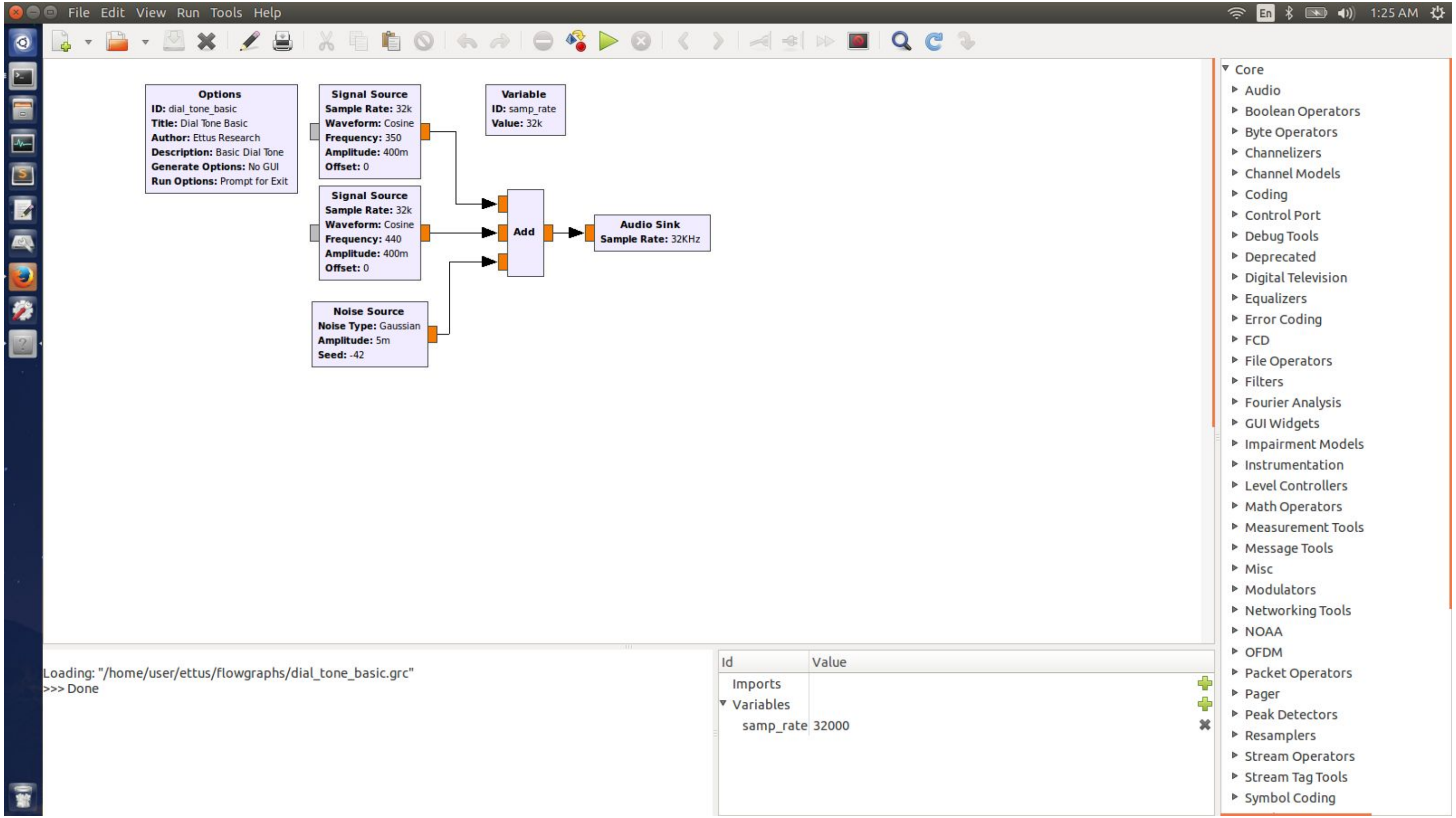
```
##################################################
# Blocks
##################################################
self.blocks_add_xx = blocks.add_vff(1)
self.audio_sink = audio.sink(32000, '', True)
self.analog_sig_source_x_1 = analog.sig_source_f(samp_rate, analog.GR_COS_WAVE, 440, .4, 0)
self.analog_sig_source_x_0 = analog.sig_source_f(samp_rate, analog.GR_COS_WAVE, 350, .4, 0)
self.analog_noise_source_x_0 = analog.noise_source_f(analog.GR_GAUSSIAN, .005, -42)

##################################################
# Connections
##################################################
self.connect((self.analog_noise_source_x_0, 0), (self.blocks_add_xx, 2))
self.connect((self.analog_sig_source_x_0, 0), (self.blocks_add_xx, 0))
self.connect((self.analog_sig_source_x_1, 0), (self.blocks_add_xx, 1))
self.connect((self.blocks_add_xx, 0), (self.audio_sink, 0))
```

Dial Tone Example: Python Code

```
    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.analog_sig_source_x_1.set_sampling_freq(self.samp_rate)
        self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)


def main(top_block_cls=dial_tone_basic, 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()
```

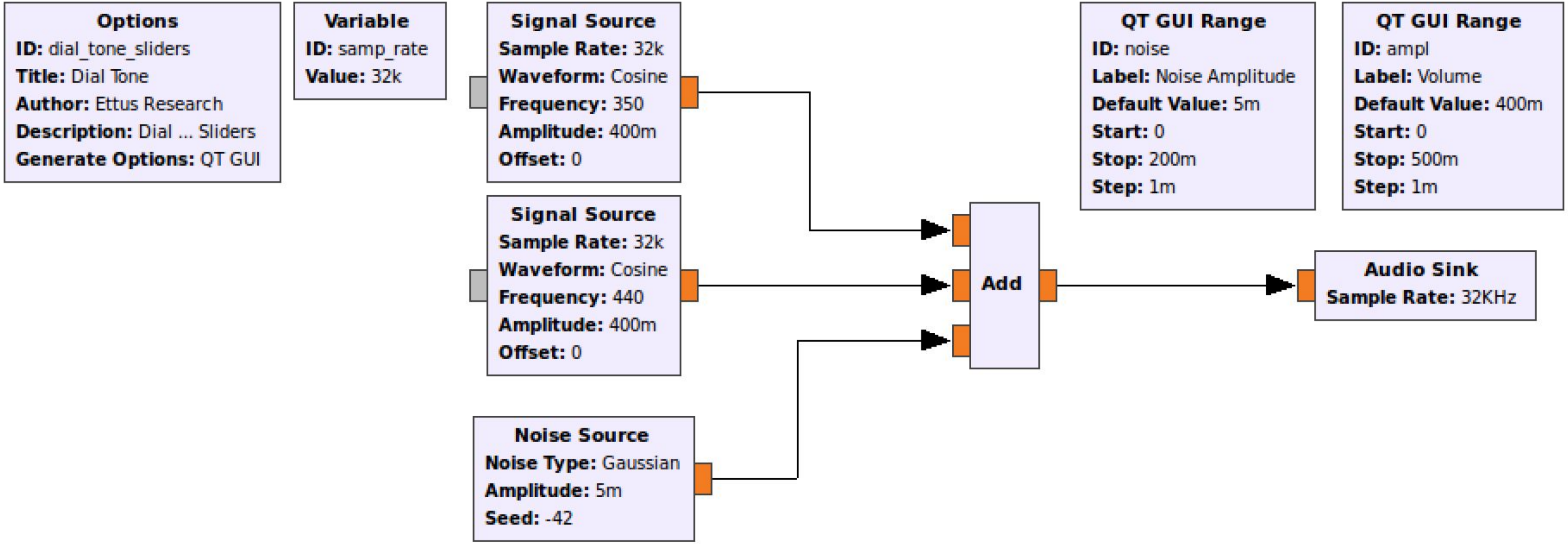
Dial Tone Example: Flowgraph

Location: ~/ettus_workshop/flowgraphs/dial_tone_basic.grc



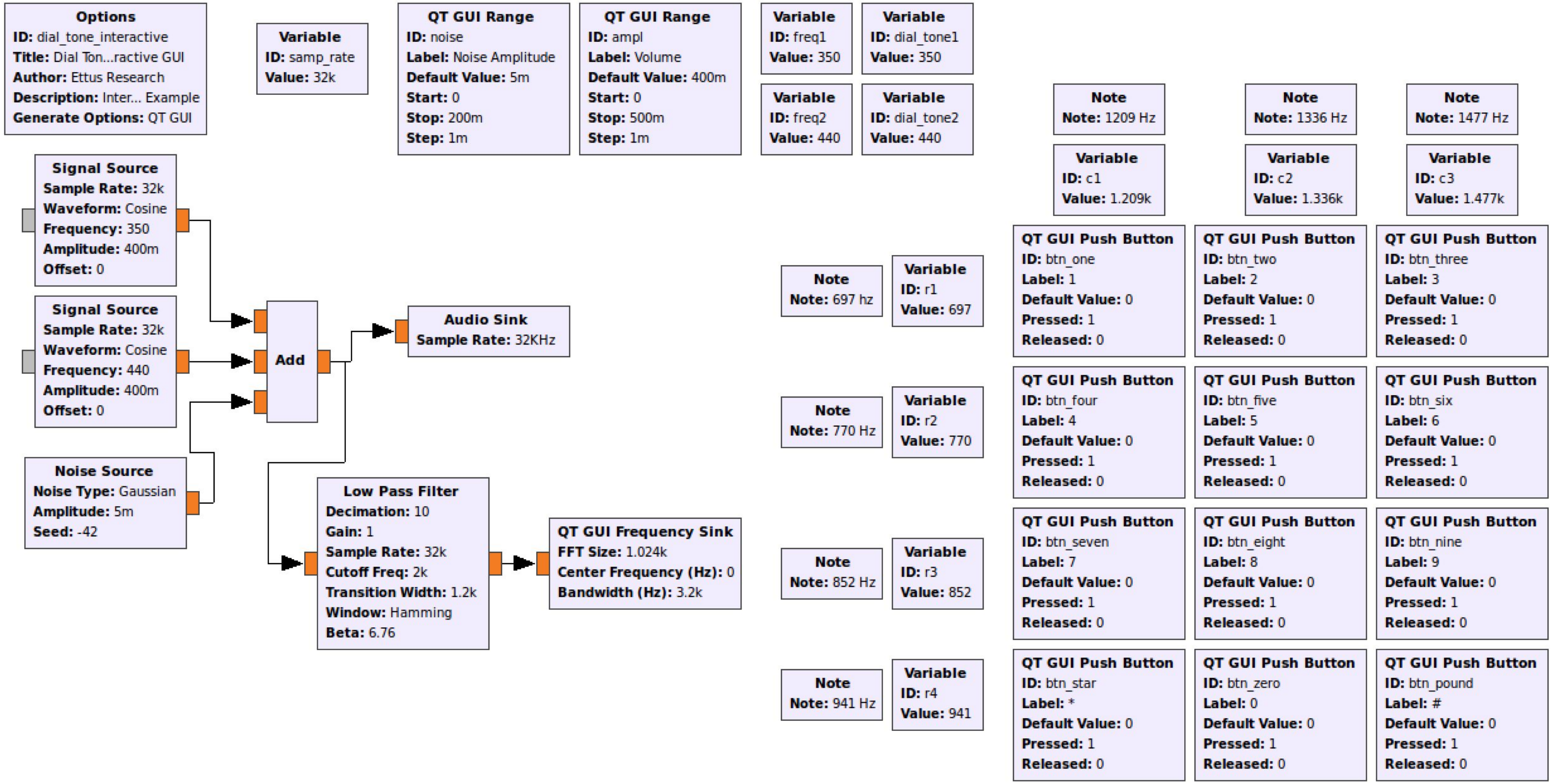
Example: Dial Tone with Slider Widgets

Location: ~/ettus_workshop/flowgraphs/dial_tone_sliders.grc



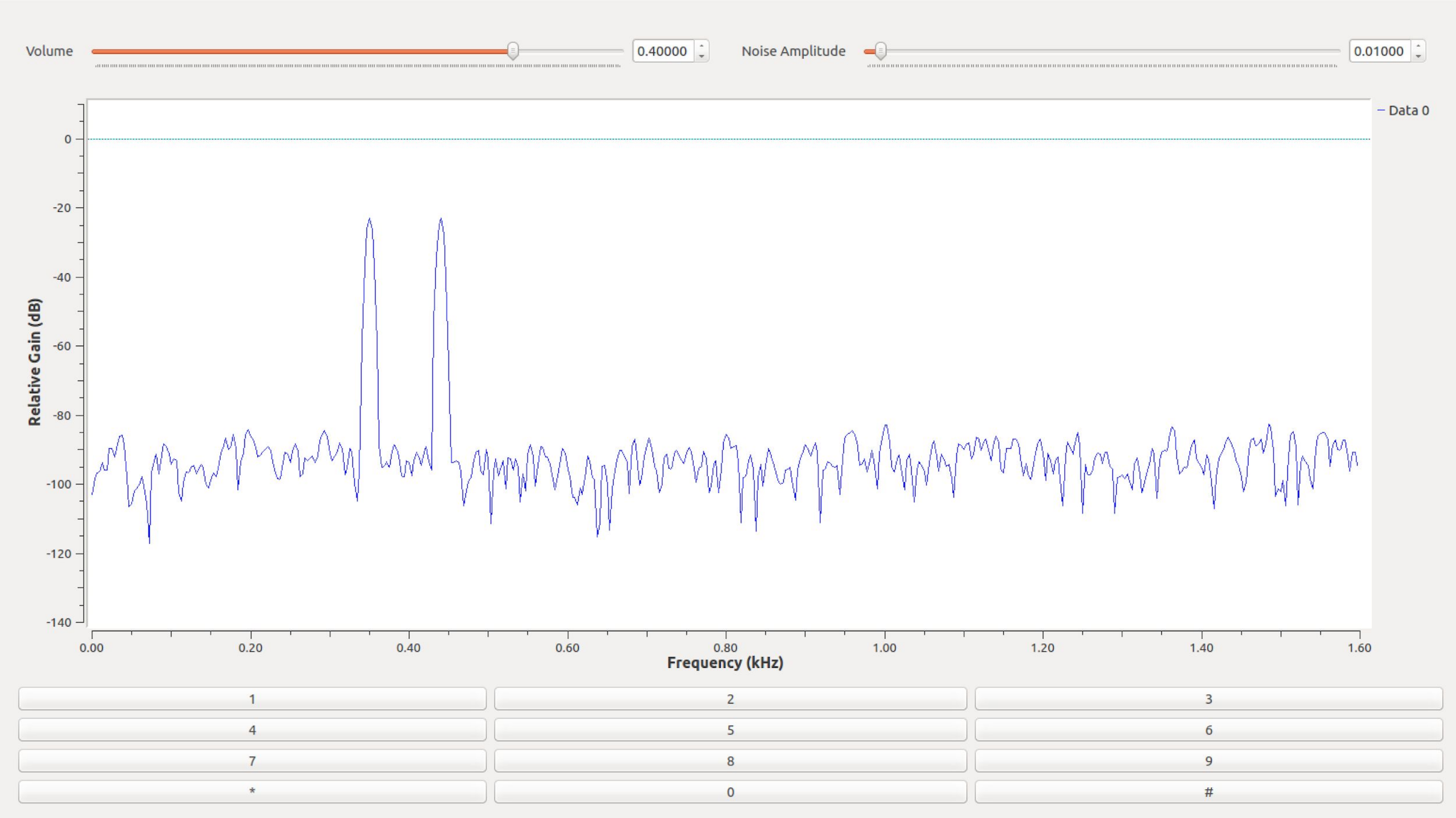
Example: Dial Tone / Touch Tone

Location: ~/ettus_workshop/flowgraphs/dial_tone_interactive.grc



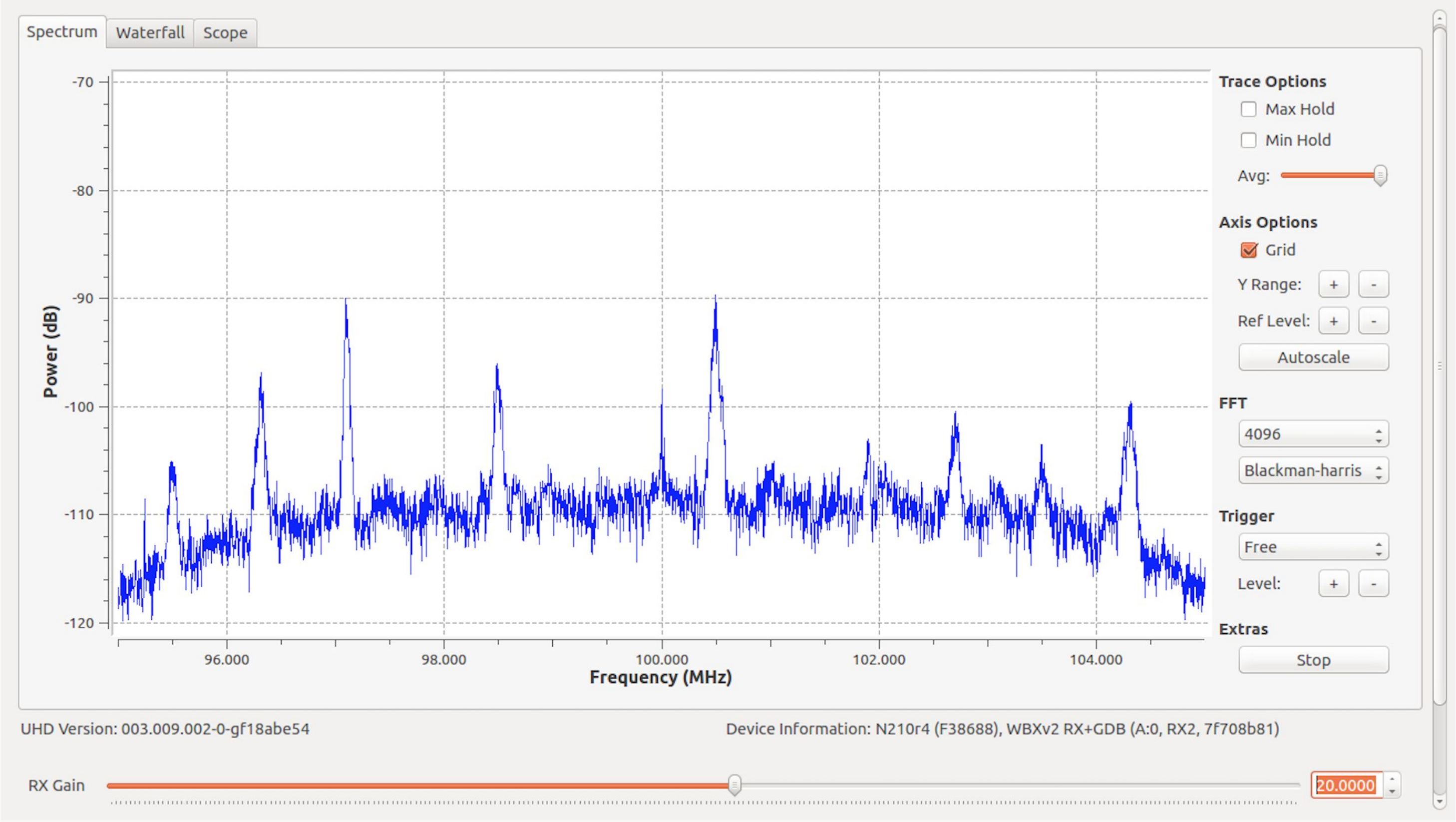
Example: Dial Tone / Touch Tone

Location: ~/ettus_workshop/flowgraphs/dial_tone_interactive.grc



Spectrum Display Tool `uhd_fft`

```
uhd_fft --args "addr=192.168.10.2" --freq 100e6 -s 10e6 -g 20
```



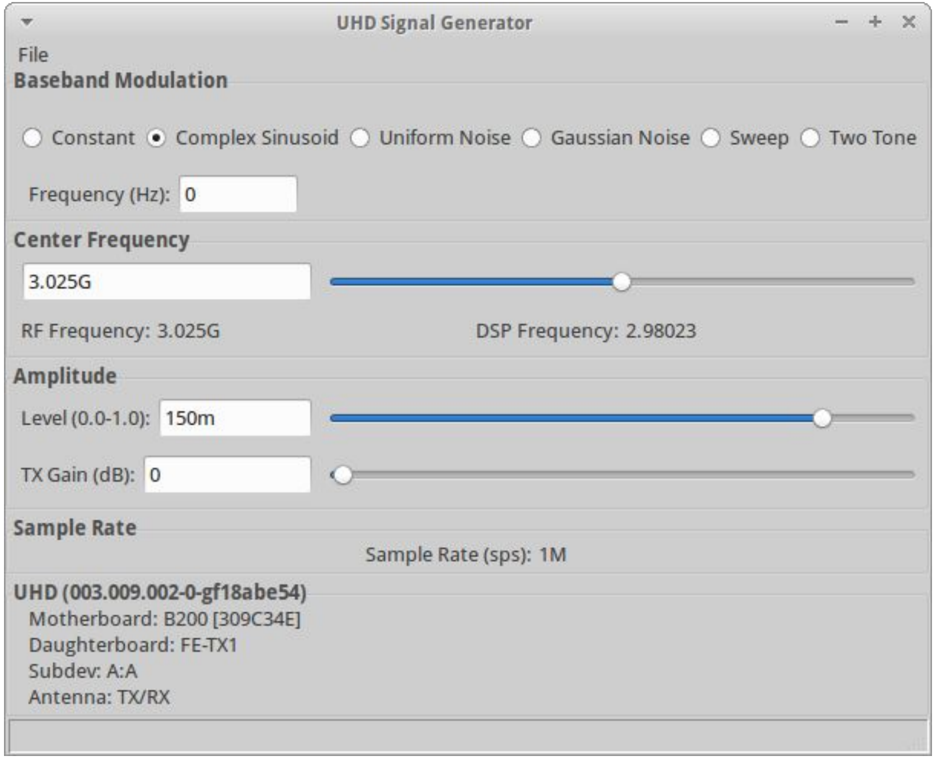
Signal Transmit Tool `uhd_siggen`

`uhd_siggen --args “addr=192.168.10.2” --freq 915e6 -g 0`

```
user@host:~$ uhd_siggen -f 915e6 -g 0
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.002-0-gf18abe54

-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
[UHD-SIGGEN] UHD Signal Generator
[UHD-SIGGEN] UHD Version: 003.009.002-0-gf18abe54
[UHD-SIGGEN] Using USRP configuration:
[UHD-SIGGEN]   Motherboard: N210r4 (F38688)
[UHD-SIGGEN]   Daughterboard: WBXv2 TX+GDB, b9e625d4
[UHD-SIGGEN]   Subdev: A:0
[UHD-SIGGEN]   Antenna: TX/RX

[UHD-SIGGEN] Press Enter to quit:

user@host:~$
```

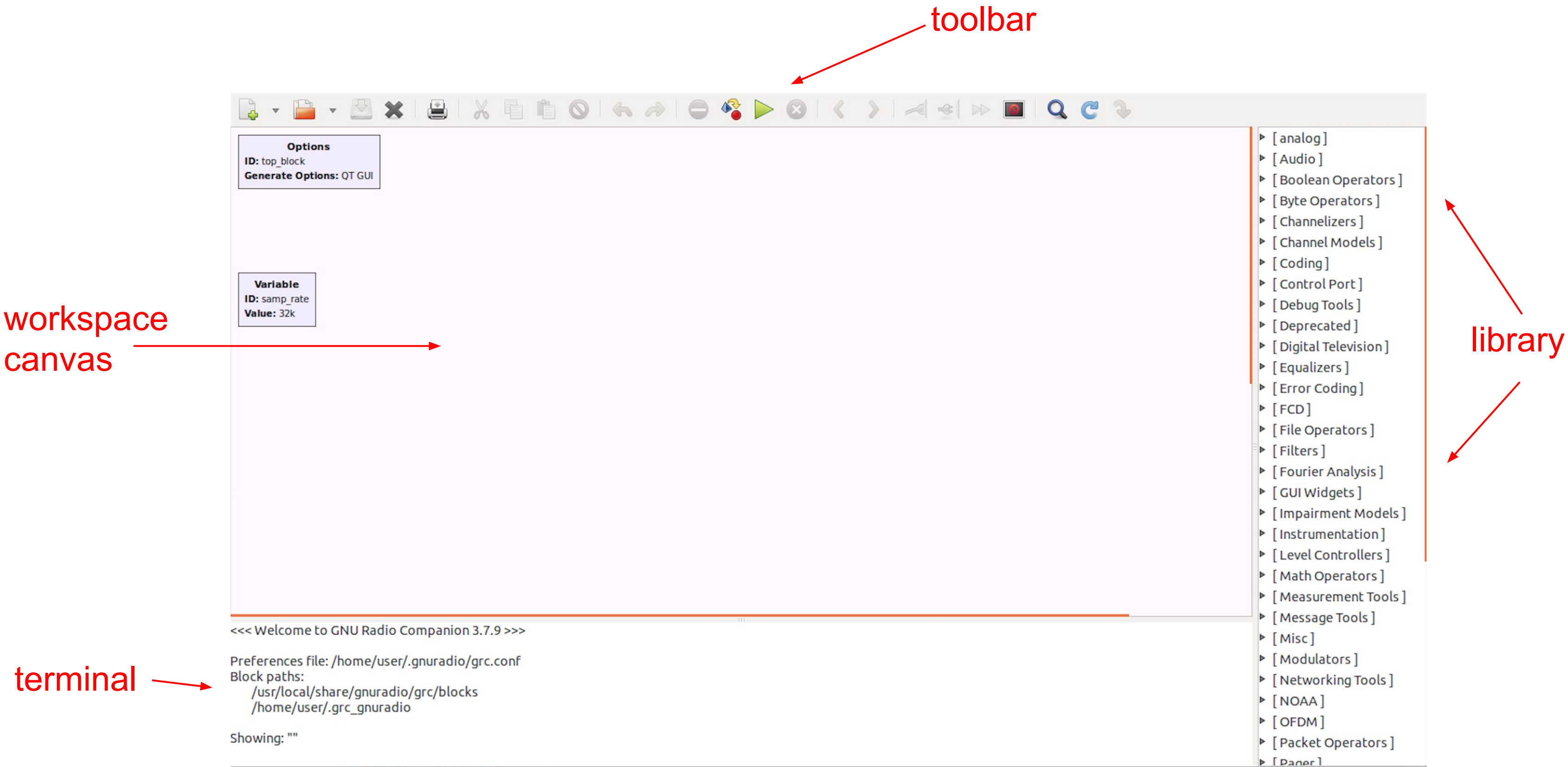
Signal Transmit Tool `uhd_siggen_gui`

```
uhd_siggen_gui --args "addr=192.168.10.2" --freq 3025e6 -g 0
```

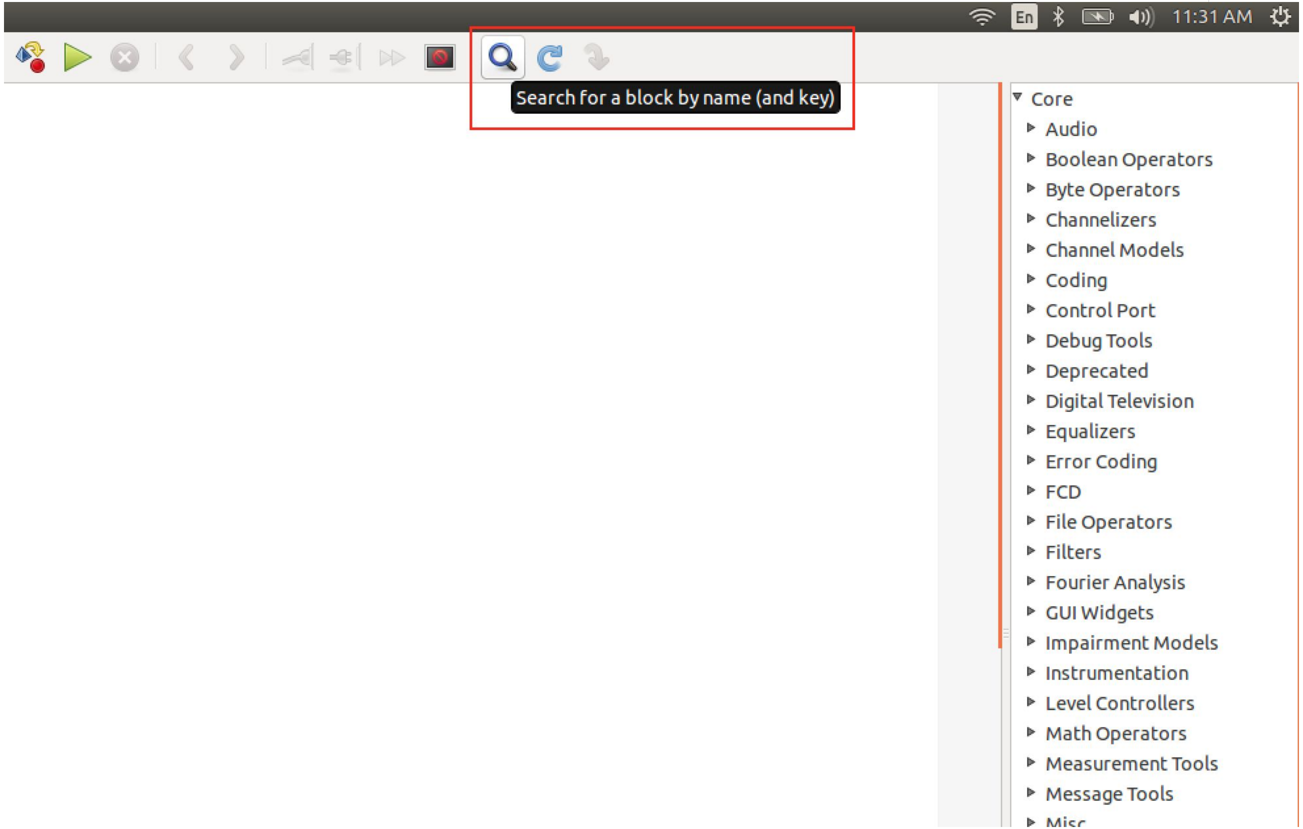


Using gnuradio-companion

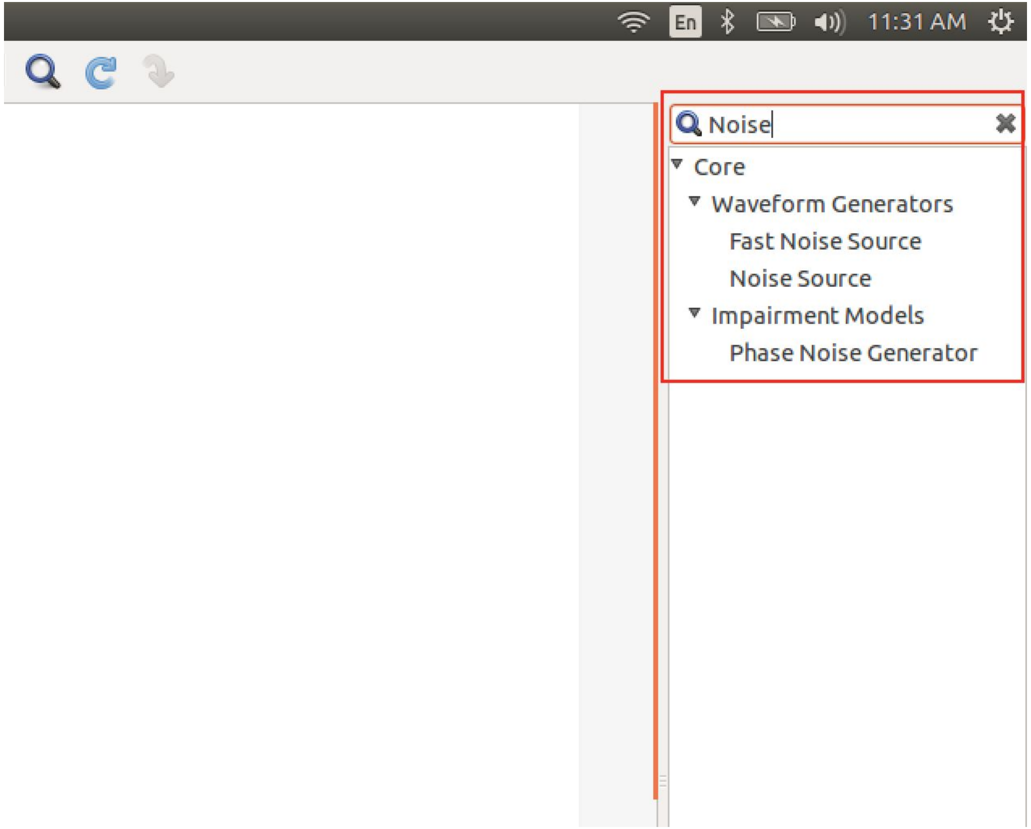
At a command prompt, type: **`gnuradio-companion`**



Using gnuradio-companion - Search



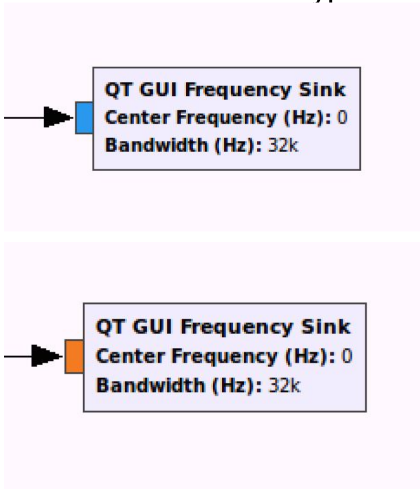
Using gnuradio-companion - Search



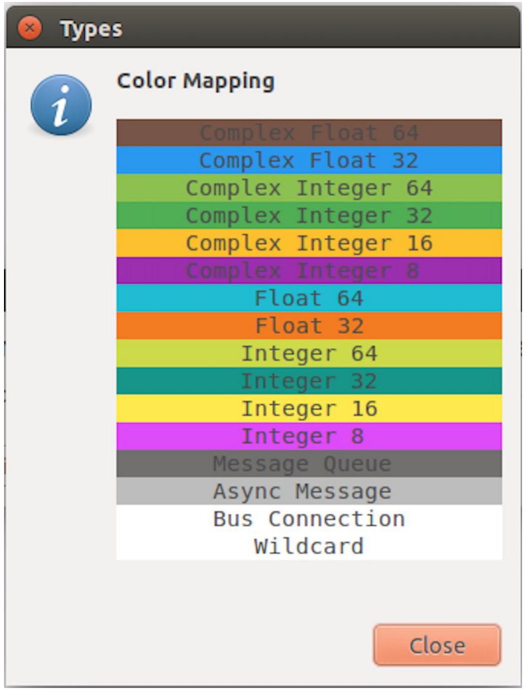
Using gnuradio-companion

Blocks have ports which input and output specific data types.

The color of the port indicates its data type.



Help -> Types

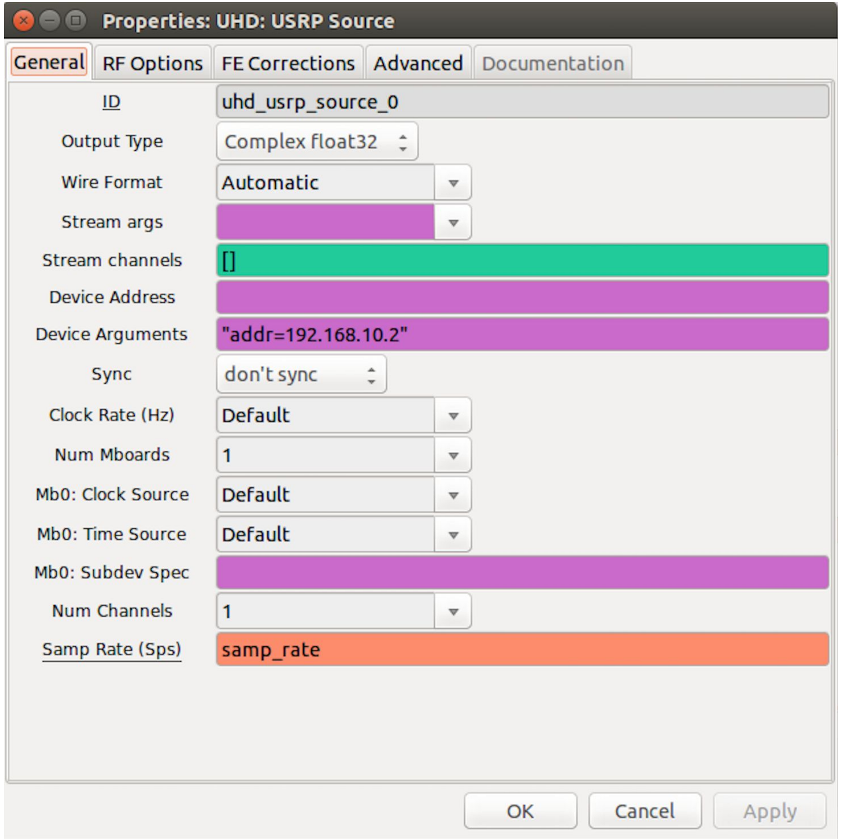


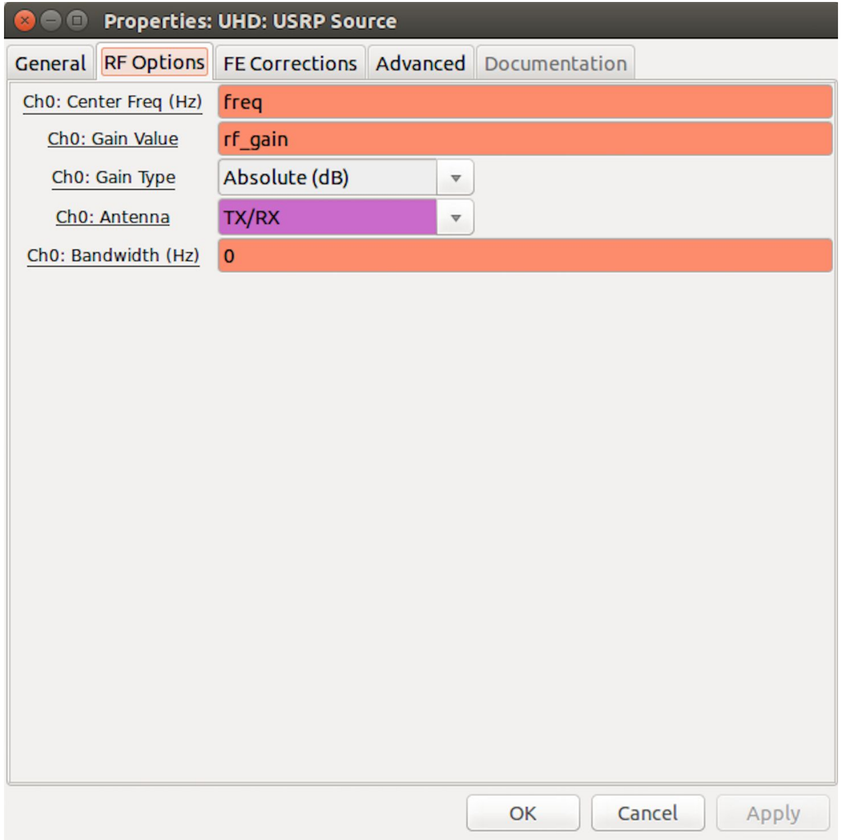
Hot keys:

- Up/Down arrows change data type
- E/D keys enable/disable blocks

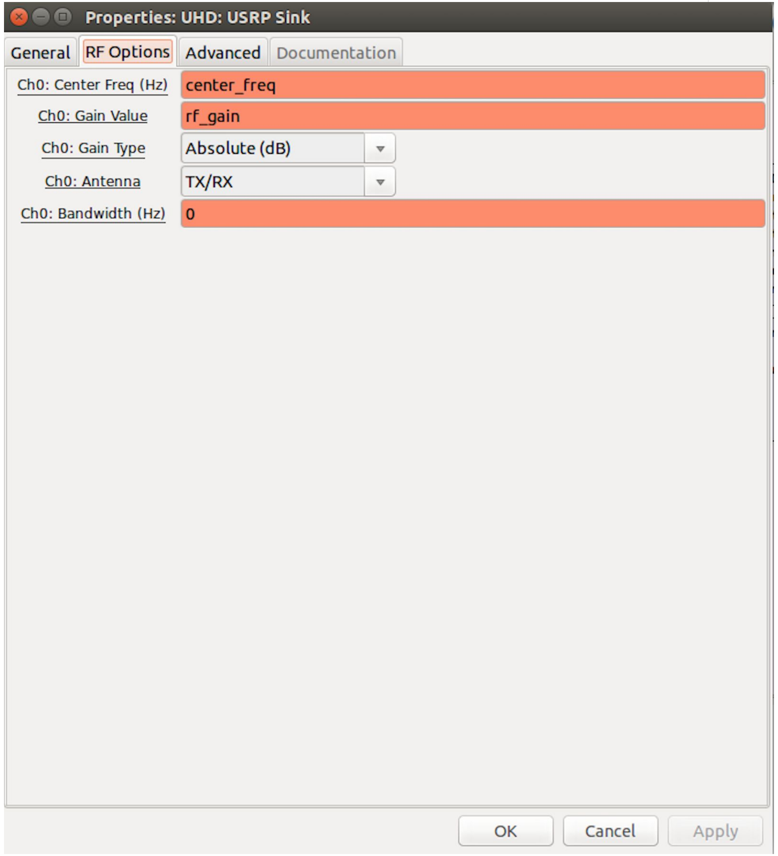
Using gnuradio-companion

Every block has properties that can be viewed and set



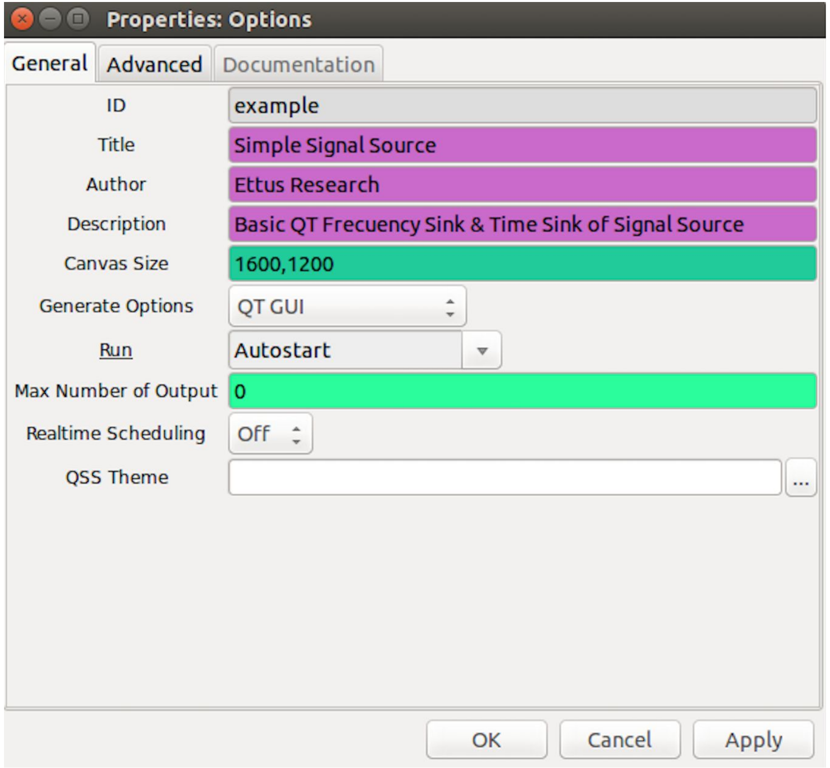


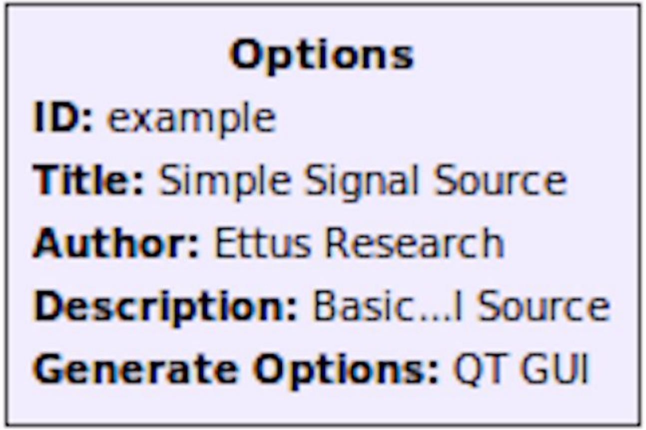
Using gnuradio-companion



Ettus Research™
A National Instruments Company

Options Block





Options Block

- **ID:** File name of generated Python code
- **TITLE:** Title of flowgraph
- **AUTHOR:** Author of flowgraph
- **DESCRIPTION:** Description of flowgraph
- **CANVAS SIZE:** Size of working area for flowgraph
- **GENERATE OPTIONS:** QT GUI, WX GUI, No GUI, HIER BLOCK, HIER BLOCK (QT GUI)
- **RUN:** Autostart / OFF
- **MAX NUMBER OF OUTPUTS:** Limits max number of outputs of any block
- **REALTIME SCHEDULING:** Use real-time CPU scheduling to run flowgraph
- **QSS THEME:** Theme of flowgraph <install_path>/share/gnuradio/themes/

Throttle Block

- Distinct from a mathematical (DSP) calculation context, sample rate also refers to the rate at which samples pass through the flowgraph

- If there is no rate control, hardware clock, or throttling mechanism, then the samples will be generated, pass through the flowgraph, and be consumed as fast as possible (i.e., the flowgraph will be only CPU-bound)

- This is desirable if you want to perform some specific DSP on data as quickly as possible (e.g., read from a file, re-sample, and write it back to disk)

- Only a block that represents some underlying hardware with its own clock (e.g. USRP, sound card), or the Throttle Block itself, will use 'Sample Rate' to set that hardware clock, and therefore have the effect of applying rate control to the samples in the flowgraph

- Not having a Throttle Block in a flowgraph where it's needed may result in the flowgraph consuming 100% of your CPU, and your system becoming unresponsive

Throttle Block (cont'd)

- A Throttle Block will simply apply host-based timing (against the 'wall clock') to control the rate of the samples it produces (i.e. samples that it makes available on its outputs to downstream blocks)

- A hardware Sink block will consume samples at a fixed rate (relative to the wall clock)

- The Throttle Block, or a hardware Sink block, will apply 'back pressure' to the upstream blocks (the rate of work of the upstream blocks will be limited by the throttling effect of this rate-controlling block)

- A hardware Source block will produce samples at a fixed rate (relative to the wall clock)

- In general, there should only ever be one block in a flowgraph that has the ability to throttle sample flow

Components of GNU Radio

- GNU Radio is comprised of components
- Components consist of blocks as well as other functionality
- The top-level components included in the GNU Radio distribution are:

Fundamentals

- **`gr-analog`**
  - Blocks for analog communications
- **`gr-block`**
  - Basic block library
- **`gr-digital`**
  - Blocks for digital communications
- **`gr-fec`**
  - Forward Error Correction signal processing blocks

Components of GNU Radio

- **`gr-fft`**
  - FFT signal processing blocks
- **`gr-filter`**
  - Filter signal processing blocks
- **`gr-runtime`**
  - GNU Radio core runtime infrastructure
- **`gr-trellis`**
  - Trellis-based algorithms for GNU Radio
- **`gr-vocoder`**
  - Blocks implementing voice codecs
- **`gr-wavelet`**
  - Wavelet signal processing blocks for GNU Radio

Components of GNU Radio

Graphical Interfaces

- **`gr-qtgui`**
    - QT 5 GUI Interface
    - QT is the default/primary GUI toolkit
    - wxWidgets fully deprecated and no longer supported

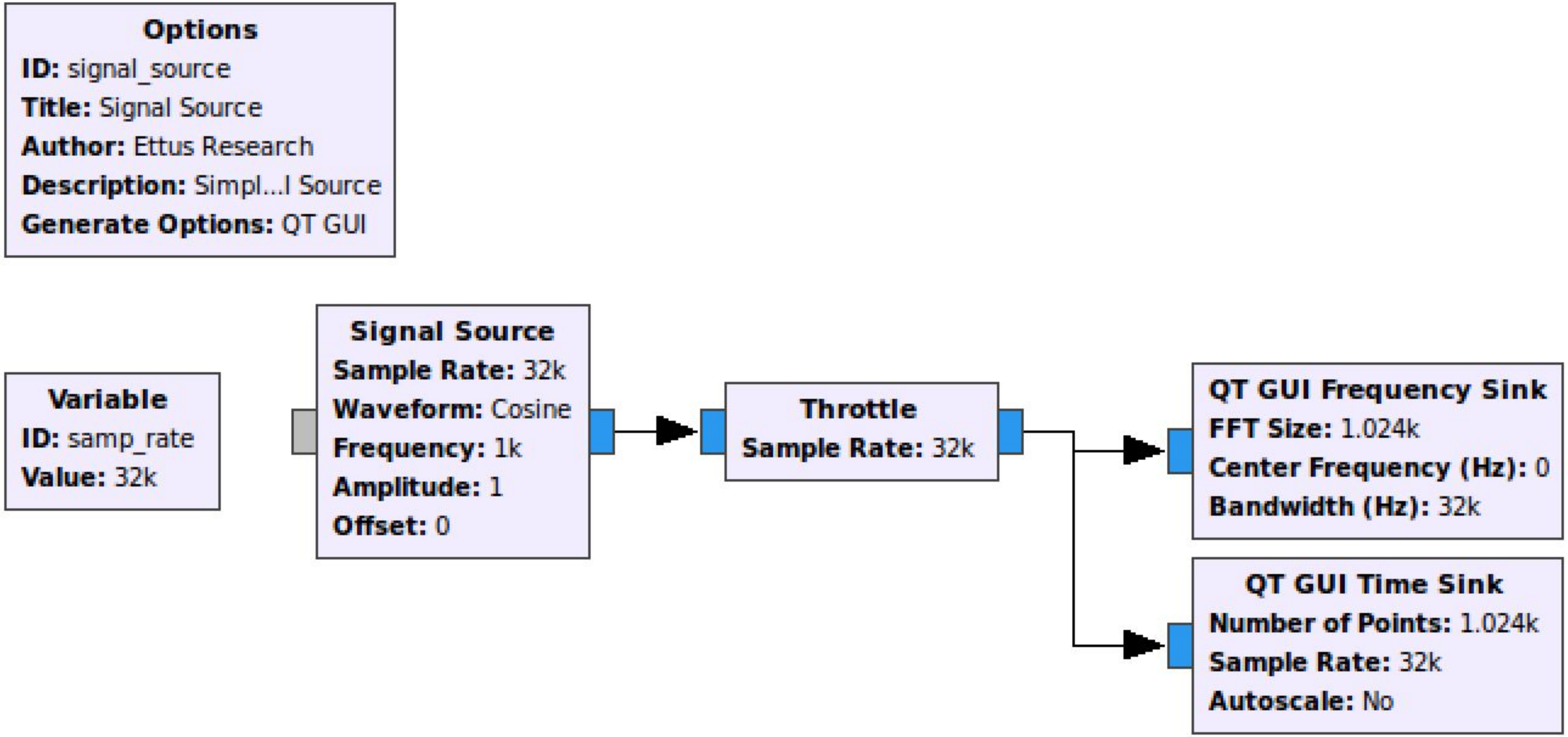
Components of GNU Radio

Hardware Interfaces

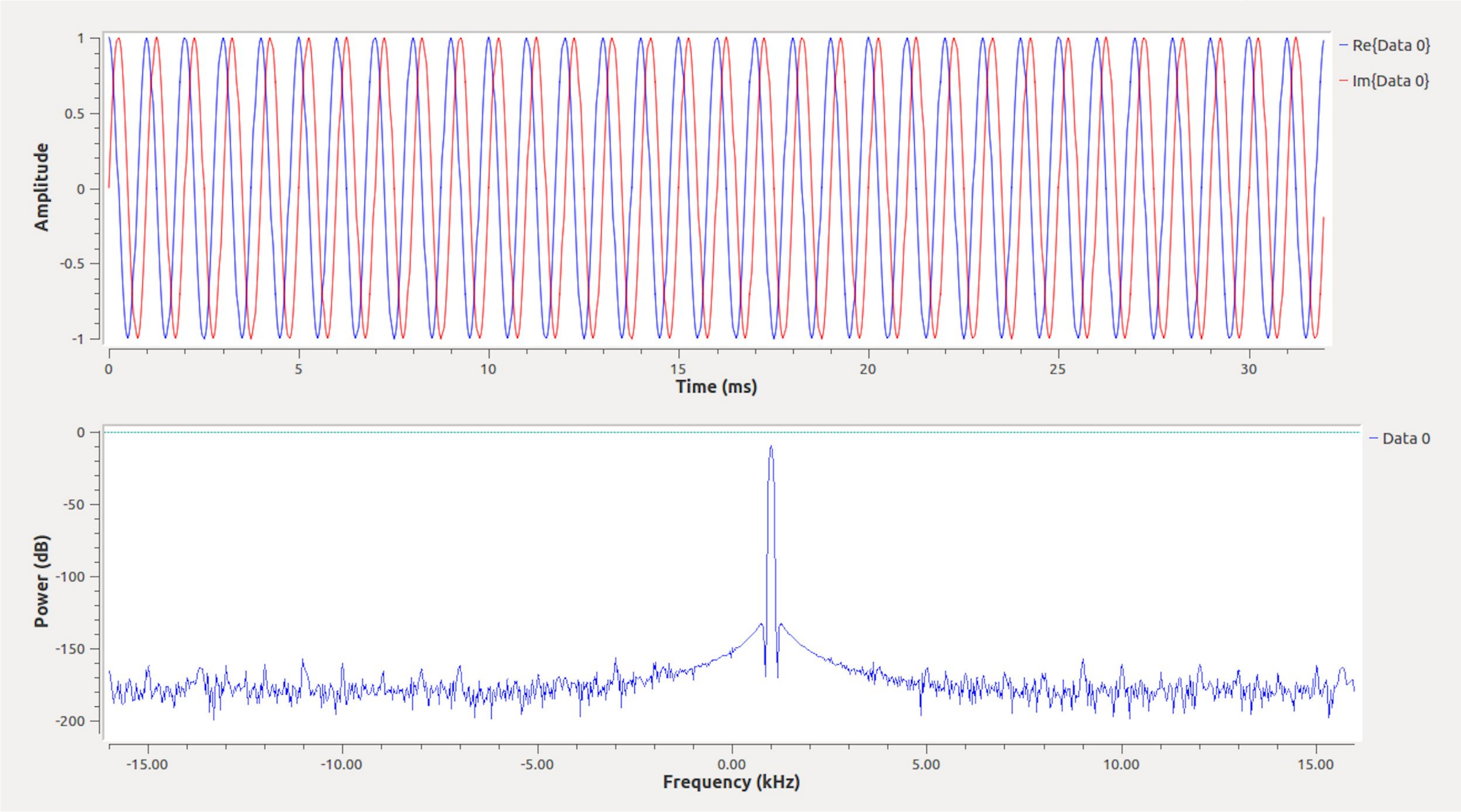
- **`gr-audio`**
  - Block for all supported audio sound systems
- **`gr-comedi`**
  - Blocks for the comedi library
- **`gr-fcd`**
  - Funcube Dongle source block for GNU Radio
- **`gr-shd`**
  - Blocks for the Simplex Hardware Driver (SHD)
- **`gr-uhd`**
  - Blocks to interface with USRP / UHD
- **`gr-osmocom`**
  - Universal Block to interface with various SDR Hardware

Example: Signal Source

Location: ~/ettus_workshop/flowgraphs/signal_source.grc

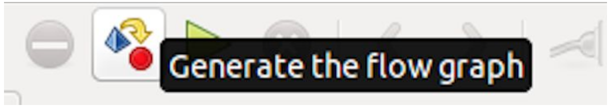


Example: Signal Source Running



Using GNU Radio from Python

Generate Python from GRC Flow graph



Invoke directly from the Linux command line:

`$ python example_3.py`

```
>>> import gnuradio
>>> ...
```

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Example3
# Generated: Wed Jan 20 12:05:57 2016
##################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

from PyQt4 import Qt
from gnuradio import analog
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.qtgui import Range, RangeWidget
from optparse import OptionParser
import sys
import time


class example3(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Example3")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Example3")
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "example3")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())
```

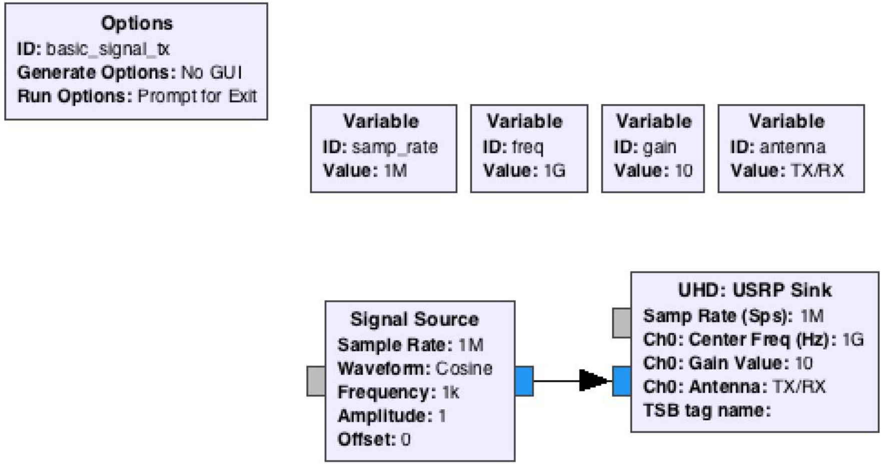
Using GNU Radio from Python

```
        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 5e6
        self.rf_gain = rf_gain = 0
        self.freq = freq = 1e6
        self.center_freq = center_freq = 915000000
        self.amp = amp = 0.5

        ##################################################
        # Blocks
        ##################################################
        self._rf_gain_range = Range(0, 25, 1, 0, 200)
        self._rf_gain_win = RangeWidget(self._rf_gain_range, self.set_rf_gain, "RF Gain", "counter_slider", float)
        self.top_layout.addWidget(self._rf_gain_win)
        self._freq_range = Range(0, 5e6, 1000, 1e6, 200)
        self._freq_win = RangeWidget(self._freq_range, self.set_freq, "Freq", "counter_slider", float)
        self.top_layout.addWidget(self._freq_win)
        self._amp_range = Range(0, 1, .1, 0.5, 200)
        self._amp_win = RangeWidget(self._amp_range, self.set_amp, "Amp", "counter_slider", float)
        self.top_layout.addWidget(self._amp_win)
        self.uhd_usrp_sink_0 = uhd.usrp_sink(
                ",".join(("", "type=usrp2,addr=192.168.10.2")),
                uhd.stream_args(
                        cpu_format="fc32",
                        channels=range(1),
                ),
        )
        self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_0.set_center_freq(center_freq, 0)
        self.uhd_usrp_sink_0.set_gain(rf_gain, 0)
        self.uhd_usrp_sink_0.set_antenna("TX/RX", 0)
        self.analog_sig_source_x_0 = analog.sig_source_c(samp_rate, analog.GR_COS_WAVE, freq, amp, 0)

        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_sig_source_x_0, 0), (self.uhd_usrp_sink_0, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "example3")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()


    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
        self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)

    def get_rf_gain(self):
        return self.rf_gain
```

```
    def set_rf_gain(self, rf_gain):
        self.rf_gain = rf_gain
        self.uhd_usrp_sink_0.set_gain(self.rf_gain, 0)


    def get_freq(self):
        return self.freq

    def set_freq(self, freq):
        self.freq = freq
        self.analog_sig_source_x_0.set_frequency(self.freq)

    def get_center_freq(self):
        return self.center_freq

    def set_center_freq(self, center_freq):
        self.center_freq = center_freq
        self.uhd_usrp_sink_0.set_center_freq(self.center_freq, 0)

    def get_amp(self):
        return self.amp

    def set_amp(self, amp):
        self.amp = amp
        self.analog_sig_source_x_0.set_amplitude(self.amp)


def main(top_block_cls=example3, options=None):

    from distutils.version import StrictVersion
    if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls()
    tb.start()
    tb.show()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()


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

Example: Basic Signal Transmission

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.grc



Example: Basic Signal Transmission

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Basic Signal Tx
# Generated: Mon Apr 10 21:33:56 2017
##################################################

from gnuradio import analog
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from optparse import OptionParser
import time
```

Setting Python Environment
Basic Informational Header

Required GNU Radio / Python Imports

Example: Basic Signal Transmission

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.py

```
class basic_signal_tx(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "Basic Signal Tx")
```

Top Level Class

- Class name is set by “ID” in “Options” Block

Example: Basic Signal Transmission

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.py

```
####################################################
# Variables
####################################################
self.samp_rate = samp_rate = 1e6
self.gain = gain = 10
self.freq = freq = 1e9
self.antenna = antenna = "TX/RX"
```

All Variables are contained within Parent Class

Example: Basic Signal Transmission

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.py

```
##################################################
# Blocks
##################################################
self.uhd_usrp_sink_0 = uhd.usrp_sink(
  ",".join(("", "")),
  uhd.stream_args(
       cpu_format="fc32",
       channels=range(1),
  ),
)
self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
self.uhd_usrp_sink_0.set_center_freq(freq, 0)
self.uhd_usrp_sink_0.set_gain(gain, 0)
self.uhd_usrp_sink_0.set_antenna(antenna, 0)
self.analog_sig_source_x_0 = analog.sig_source_c(samp_rate, analog.GR_COS_WAVE, 1000, 1, 0)
```

Creation of UHD Sink Block

Calls to apply Sample Rate, Center Frequency, Gain, Antenna Selection

Creation of Signal Source Block

Example: Basic Signal Transmission

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.py

```
##################################################
# Connections
##################################################
self.connect((self.analog_sig_source_x_0, 0), (self.uhd_usrp_sink_0, 0))
```

Creating the connection between Signal Source and UHD Sink Block

Example: Basic Signal Transmission

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.py

All Variables have getters/setters

Setters will recall UHD method to apply any updated value

```
def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
    self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)

def get_gain(self):
    return self.gain

def set_gain(self, gain):
    self.gain = gain
    self.uhd_usrp_sink_0.set_gain(self.gain, 0)
```

Example: Basic Signal Transmission

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.py

```
def get_freq(self):
    return self.freq

def set_freq(self, freq):
    self.freq = freq
    self.uhd_usrp_sink_0.set_center_freq(self.freq, 0)

def get_antenna(self):
    return self.antenna

def set_antenna(self, antenna):
    self.antenna = antenna
    self.uhd_usrp_sink_0.set_antenna(self.antenna, 0)
```

Example: Basic Signal Transmission

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.py

```
def main(top_block_cls=basic_signal_tx, 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()
```

Passing of created class to main()

Initialization of “Top Block”

Starting of “Top Block / Sample Streaming”

Try/Run until raw input is entered

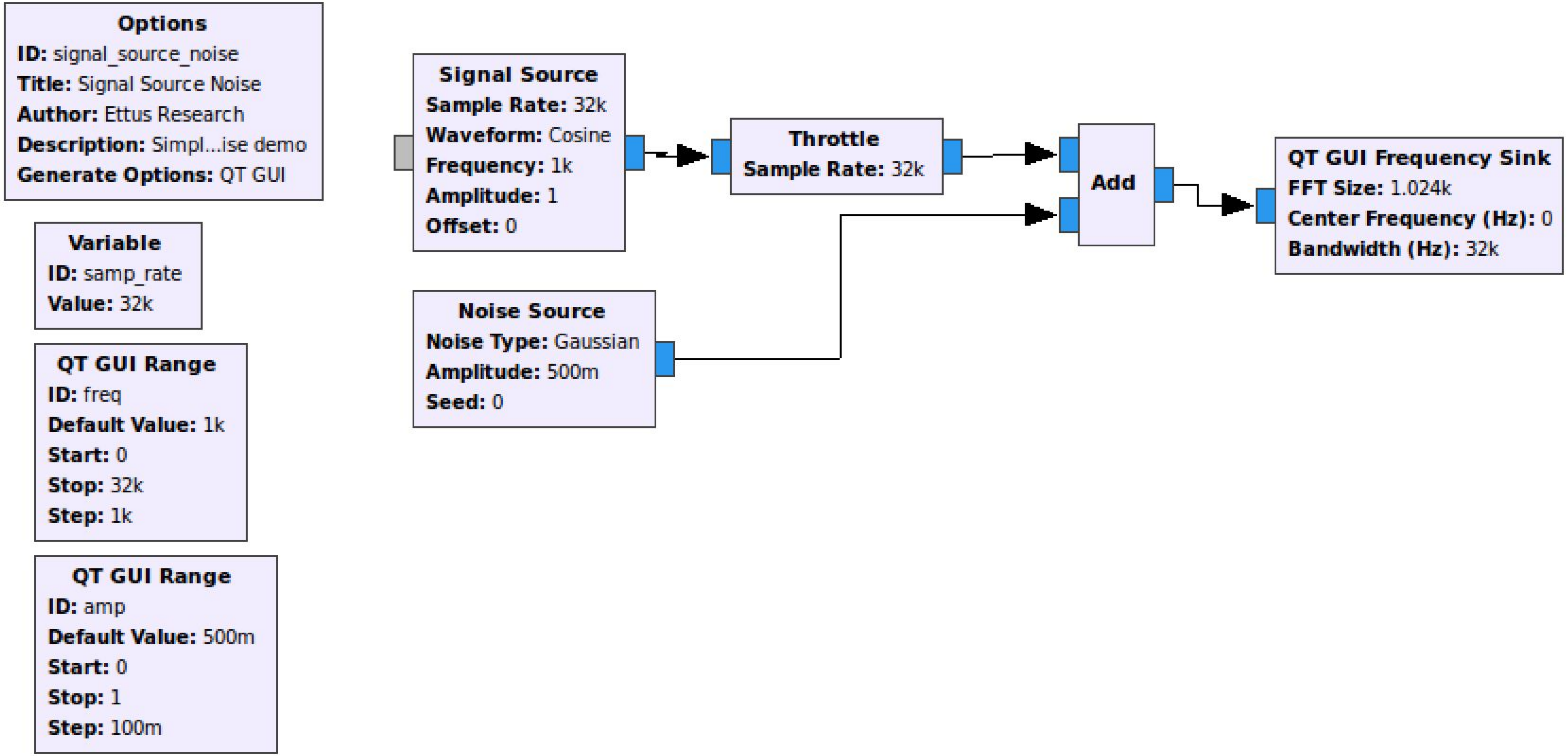
Stopping of Flowgraph / Top Block

Waits until the .stop() call has propagated through all blocks before returning

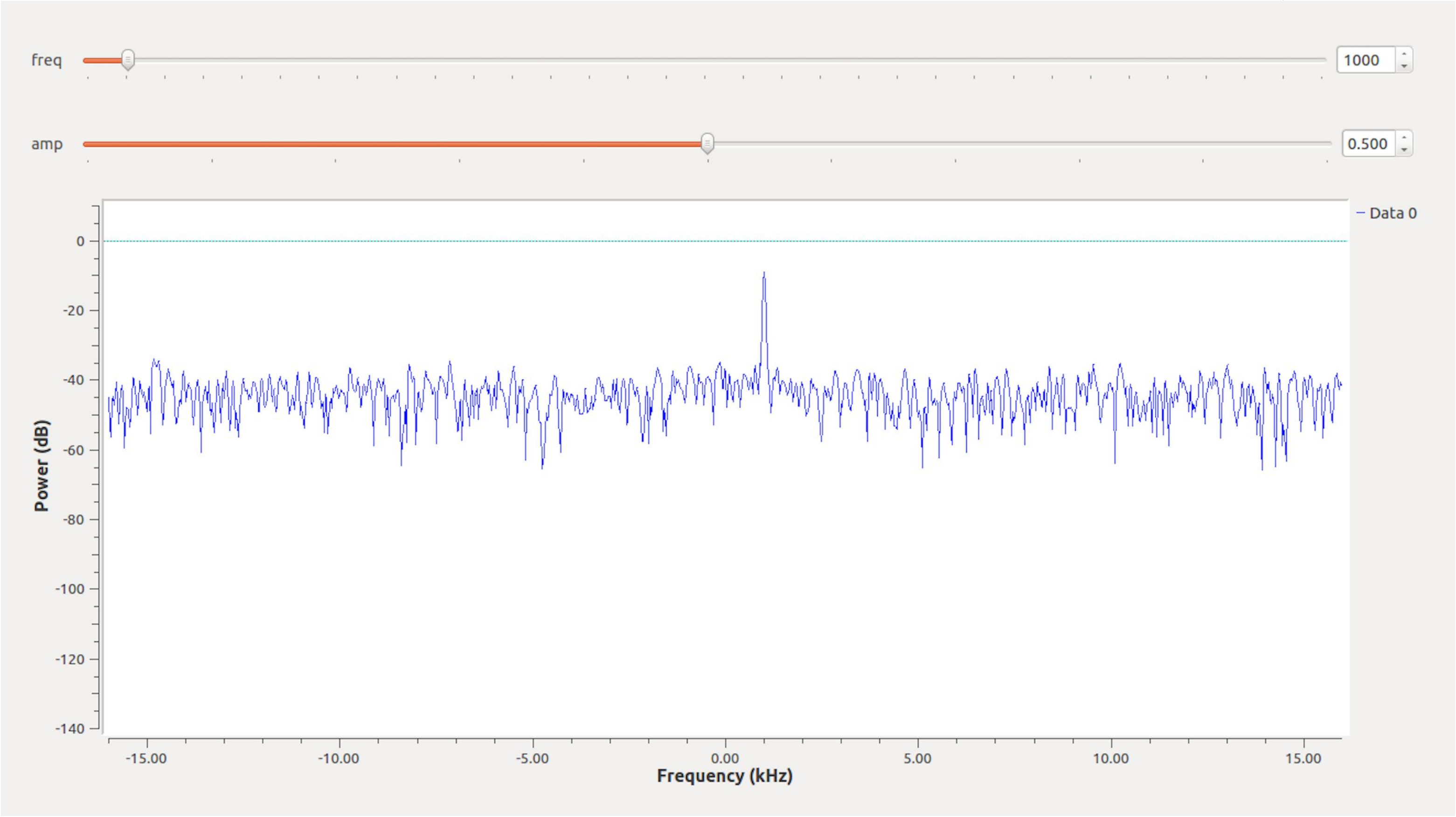
Execution of main() function to Python Interpreter

Example: Signal Source with Noise

Location: ~/ettus_workshop/flowgraphs/signal_source_noise.grc

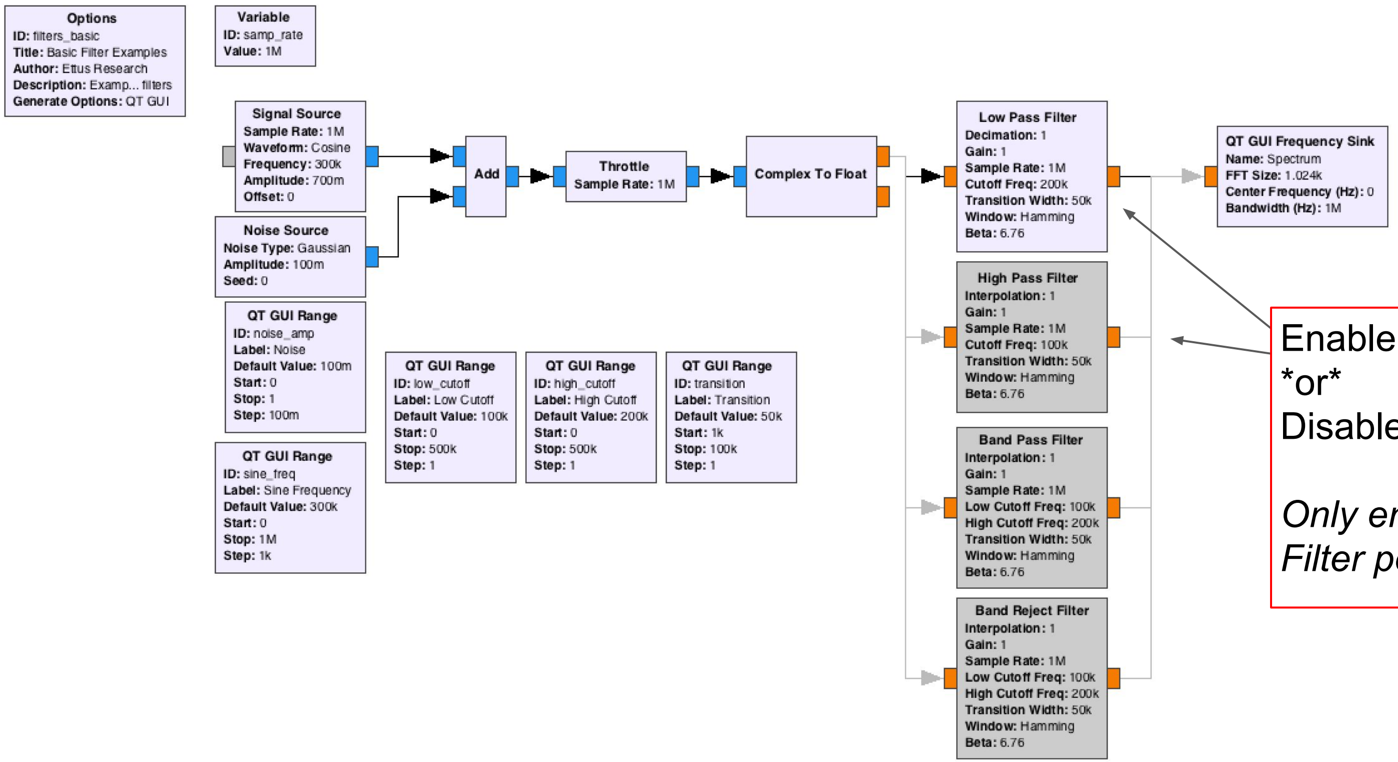


Example: Signal Source with Noise Running



Example: Filters - Flowgraph

Location: ~/ettus_workshop/flowgraphs/filters_basic.grc

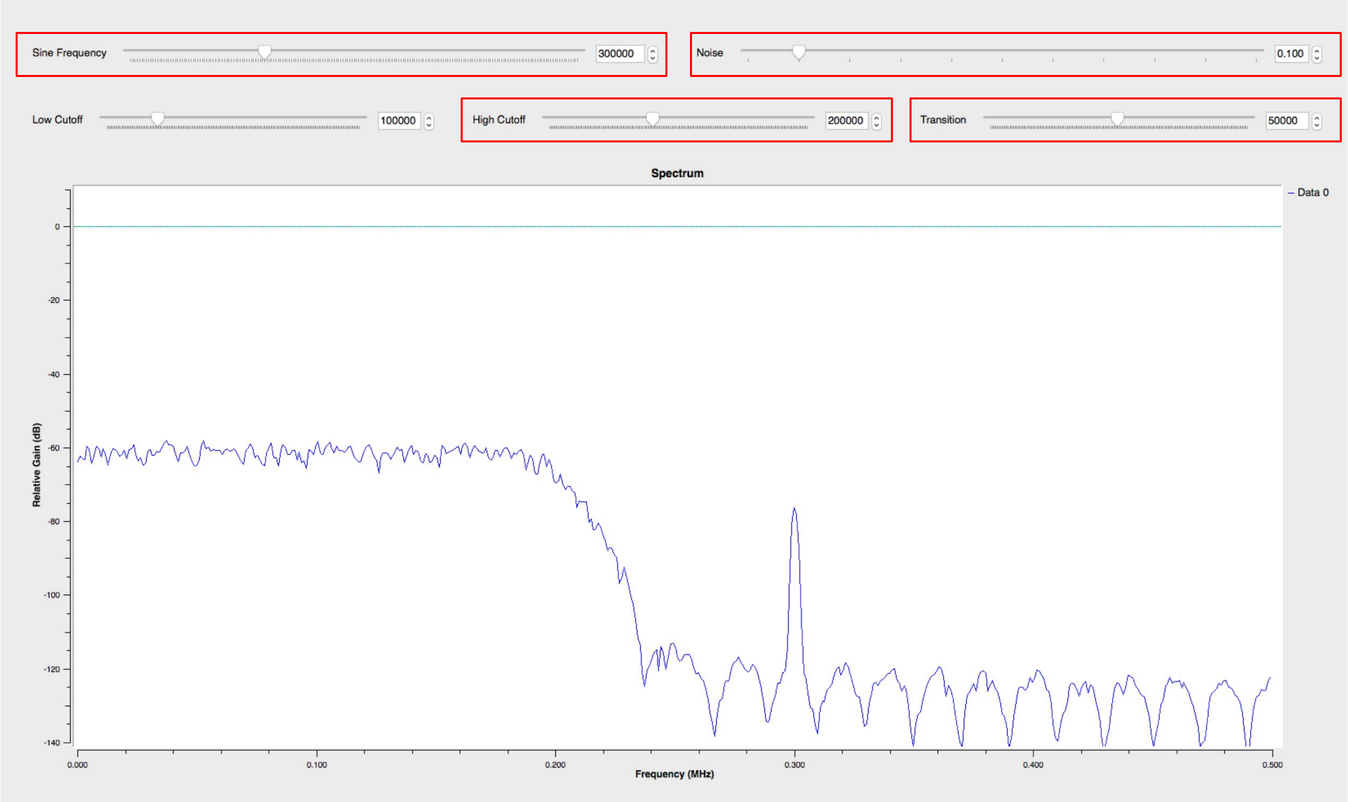


Enable [E]
or
Disable [D]

Only enable one Filter per run

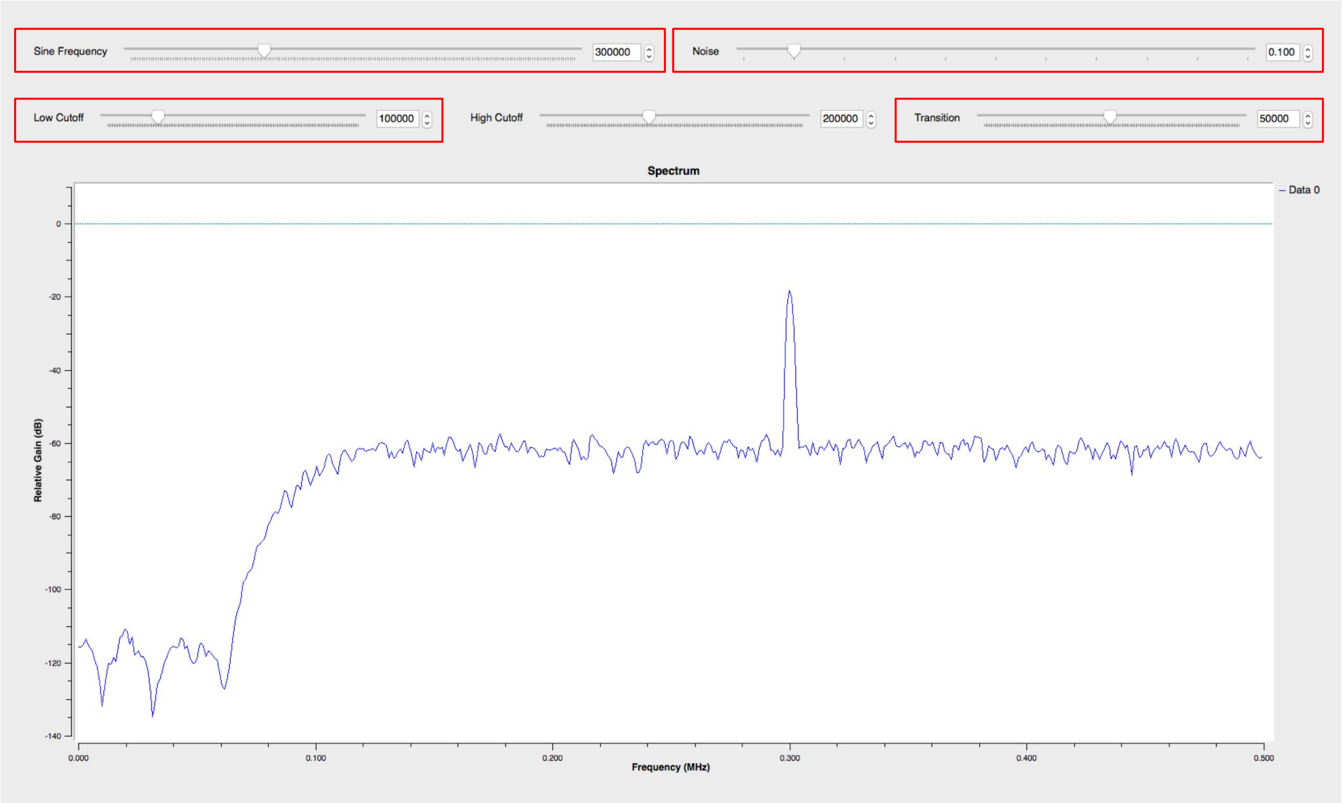
Example: Filters - Low Pass

Location: ~/ettus_workshop/flowgraphs/filters.grc



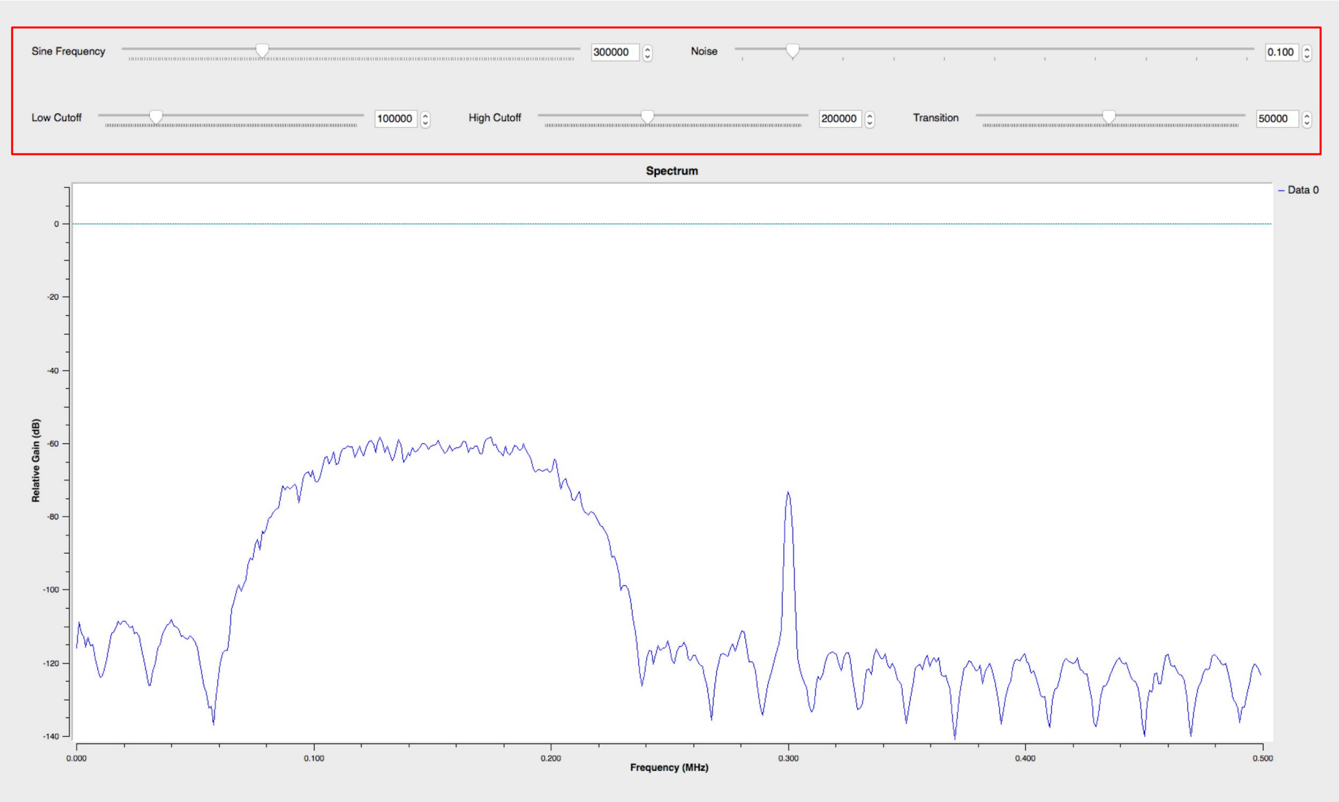
Example: Filters - High Pass

Location: ~/ettus_workshop/flowgraphs/filters.grc



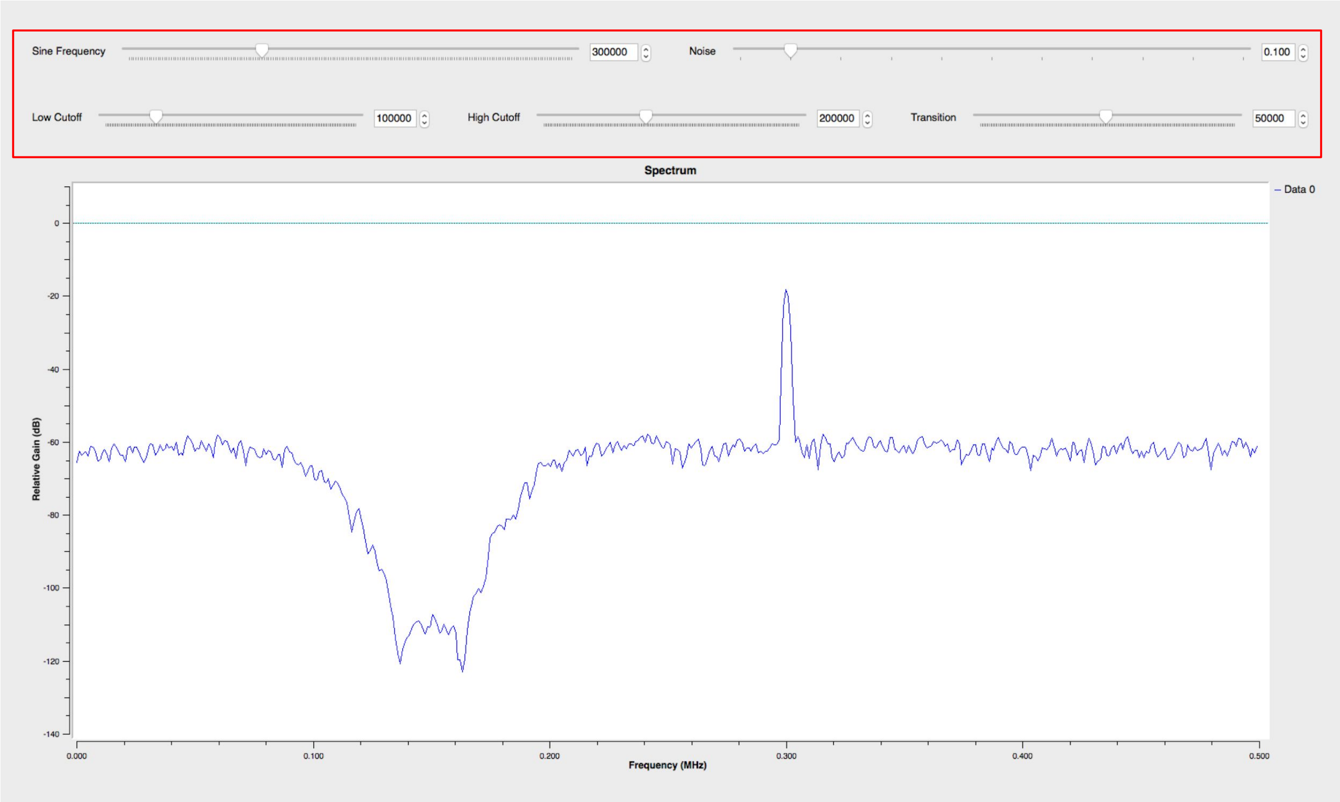
Example: Filters - Band Pass

Location: `~/ettus_workshop/flowgraphs/filters.grc`



Example: Filters - Band Reject

Location: `~/ettus_workshop/flowgraphs/filters.grc`



Out-of-Tree (OOT) Modules

- An OOT module is a GNU Radio component that does not live within the GNU Radio source tree, and is not included with the GNU Radio distribution
- OOT modules allow third-parties to extend GNU Radio with their functions and blocks

- Comprehensive GNU Radio Archive Network (CGRAN)
  - Directory of open-source OOT modules
  - Not a hosting site
  - Most OOT modules are hosted on GitHub
  - **`http://www.cgran.org/`**

- **`gr_modtool`**
  - The swiss army knife of module editing / creating
  - **`https://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules`**

CGRAN

Ettus Research™
A National Instruments Company

CGRAN Projects · Documentation · GNU Radio · VOLK

The Comprehensive GNU Radio Archive Network

The Comprehensive GNU Radio Archive Network (CGRAN) is a free open source repository for 3rd party GNU Radio applications a.k.a Out Of Tree Modules that are not officially supported by the GNU Radio project.

Browse~Checkout~Hack

Search

Name	Tags	Description	Repository
gr-eventstream	scheduler, streams, bursty	The event stream scheduler	Github
Receiver for Vaisala Weather Sonde		Receiver for Vaisala Weather Sonde	Github
gr-pyqt	gui, plotting, pyqt, pyqwt	Python QT Plotters and Message Tools Repo	Github
gr-pcap	pcap, packet	PCAP recording and playback	Github
gr-microtelecom	hardware, source	Microtelecom's Perseus SDR source module	Github
gr-lte	LTE, synchronization, estimation, PBCH	LTE downlink receiver blocks	Github
gr-nmea	sdr, gps, nmea	interface to NMEA and GPSD sources	Github
gr-ieee802-11	IEEE 802.11, WiFi, OFDM	IEEE 802.11 a/g/p Transceiver	Github
An IEEE 802.15.4 (ZigBee) Transceiver	sdr, IEEE 802.15.4, ZigBee	gr-ieee802-15-4	Github

Out-of-Tree Module Installation

```
1.  git clone <repository>

2.  cd <repository-path>

3.  mkdir build && cd build

4.  cmake ../

5.  make -j4

6.  sudo make install

7.  sudo ldconfig
```

Record & Playback of Signals

- There are many ways to record files and playback files, and this can be highly customized
- Use UHD utility programs
  - Use existing utility programs in GitHub
    - **`https://github.com/EttusResearch/uhd/tree/master/host/examples`**
  - Customize them to modify and add functionality
  - Either in C++ or in Python
    - **`https://github.com/EttusResearch/uhd/blob/master/host/examples/rx_samples_to_file.cpp`**
    - **`https://github.com/EttusResearch/uhd/blob/master/host/examples/python/rx_to_file.py`**
- Use GNU Radio
  - Easy to do with a flowgraph
  - Can easily add in-line, real-time signal processing
- Various data types supported: Complex, Int, Short, Float, Double, etc.
- The higher the sampling rate:
  - The higher the disk usage
  - The higher the disk IO
    - Use NVMe disks, not SATA disks, not external USB flash disks

Record & Playback of Signals

Ettus Research™ A National Instruments Company

- Record Signal using UHD utility program:

```
$ /usr/local/lib/uhd/examples/rx_samples_to_file \
    --args "type=b200" \
    --type float \
    --freq 433.72e6 \
    --rate 1e6 \
    --gain 10 \
    --ant TX/RX \
    --bw 1e6 \
    --file my_iq_datafile.f32
```

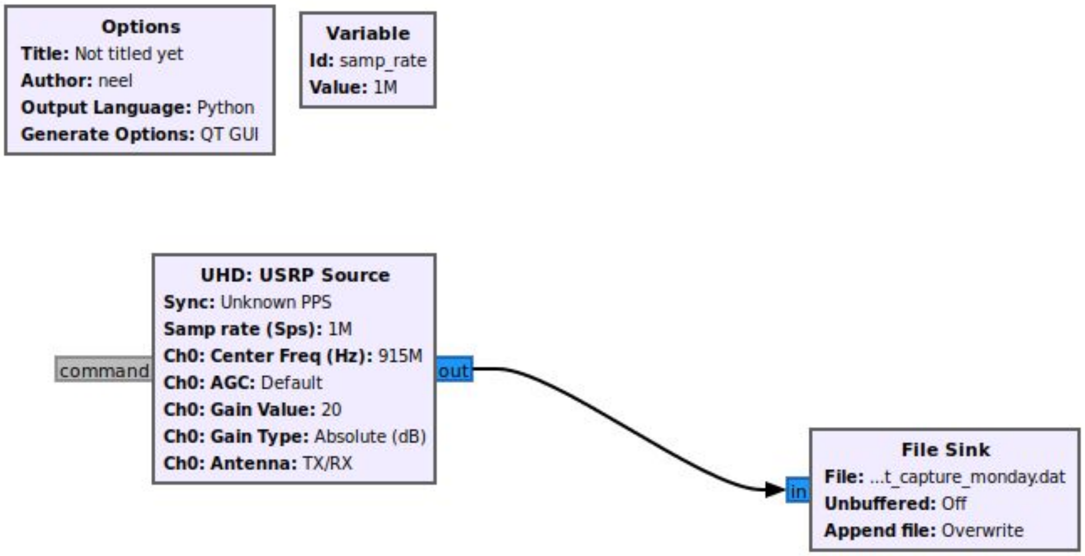
Record & Playback of Signals

- Playback Signal using UHD utility program:

```
$ /usr/local/lib/uhd/examples/tx_samples_from_file \
   --args "type=b200" \
   --type float \
   --freq 433.72e6 \
   --rate 1e6 \
   --gain 50 \
   --ant TX/RX \
   --bw 1e6 \
   --file my_iq_datafile.f32
```

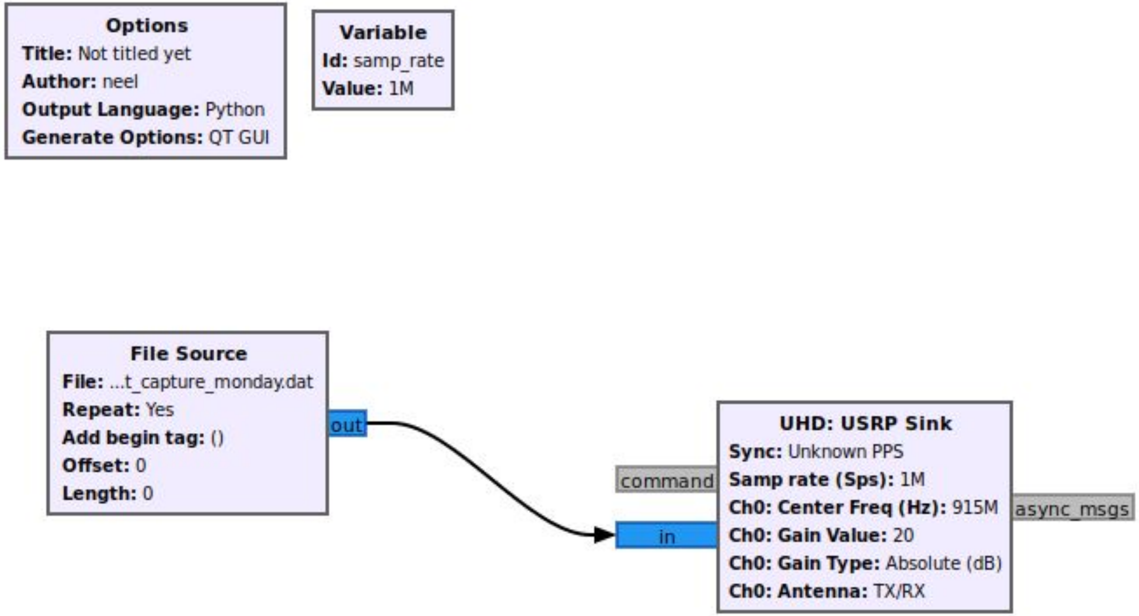
Record & Playback of Signals

- Record Signal using GNU Radio flowgraph:



Record & Playback of Signals

- Playback Signal using GNU Radio flowgraph:



Signal Data Formats

- The previous examples read and write raw I/Q data files
  - Fast to read, write, and process, but there is no header and no metadata
- Often, over time, it becomes difficult to manage large sets of raw signal capture files
  - You cannot remember, and/or did not document, the system configuration used for the capture
  - Trying to track this in the filename is tedious, error-prone, and does not scale
    - Cannot easily annotate captures
  - Difficult to archive, organize, and then later re-use captures
  - Difficult to share captures with other colleagues for collaboration
  - Difficult to create, organize, and publish data sets consisting of multiple captures
  - Inhibits the ability to reproduce research results
  - Basically, highly susceptible to "*bit rot*"

SigMF

- The Signal Metadata Format (SigMF) specifies a way to describe sets of recorded digital signal samples with metadata written in JSON. SigMF can be used to describe general information about a collection of samples, the characteristics of the system that generated the samples, and features of the signal itself.
- Designed to enable easy sharing, archiving, and publishing of datasets
- Open specification and open-source implementation on GitHub
- Can be used from C++, Python, GNU Radio *(not specific to GNU Radio)*
- Metadata is written with JSON
- **`https://github.com/gnuradio/SigMF`**
- **`https://pypi.org/project/SigMF/`**
- A SigMF recording is one flat data file and one flat metadata file
  - The data file is just raw samples
  - The metadata file is a JSON file with several sections
- Recordings can be stored and distributed in an archive format
- Archives have a defined directory structure for including multiple recordings

SigMF

- The JSON file contains several sections:
  - Global section: The General information about the file. The minimal information needed to parse the dataset file. Example fields:
    - Datatype: How are the samples stored?
    - Sample Rate: What is the sample rate at which this data was recorded?
    - Author: Who created these files?
    - Version: Which version of the SigMF specification was used to create this capture?
    - License: What is the license of this data?
    - Hash: A hash of the data to provide proof of integrity.
    - Description: A top-level description of the capture dataset.

SigMF

- The JSON file contains several sections:
  - Captures section: An array of segments that describe the parameters of the capture, starting at a certain sample index. Example fields:
    - Center Frequency: At what frequency was the radio tuned to during the capture?
    - Timestamp: What is the timestamp of a particular sample index?
  - Annotations section: An array of segments that describe features or provides comments about the signal data. Specified by sample number. Can be *"code comments"* like "detected interference here", "classified modulation as QAM64", "cat jumped on antenna", etc.

SigMF

- How to handle continuously-varying fields/metadata?
  - Dealing with fields that are continuously changing can be a significant challenge for metadata
  - Examples:
    - If your receiver is in a vehicle, how do you record the changing geolocation in a useful way?
    - If your antenna is a spinning dish, how do you record the changing azimuth of your aperture?
  - These continuously-varying fields/metadata are just another SigMF recording
- Several open-source tools now have SigMF integration:
  - Inspectrum
  - Universal Radio Hacker (URH)
  - GNU Radio (dedicated blocks)

SigMF

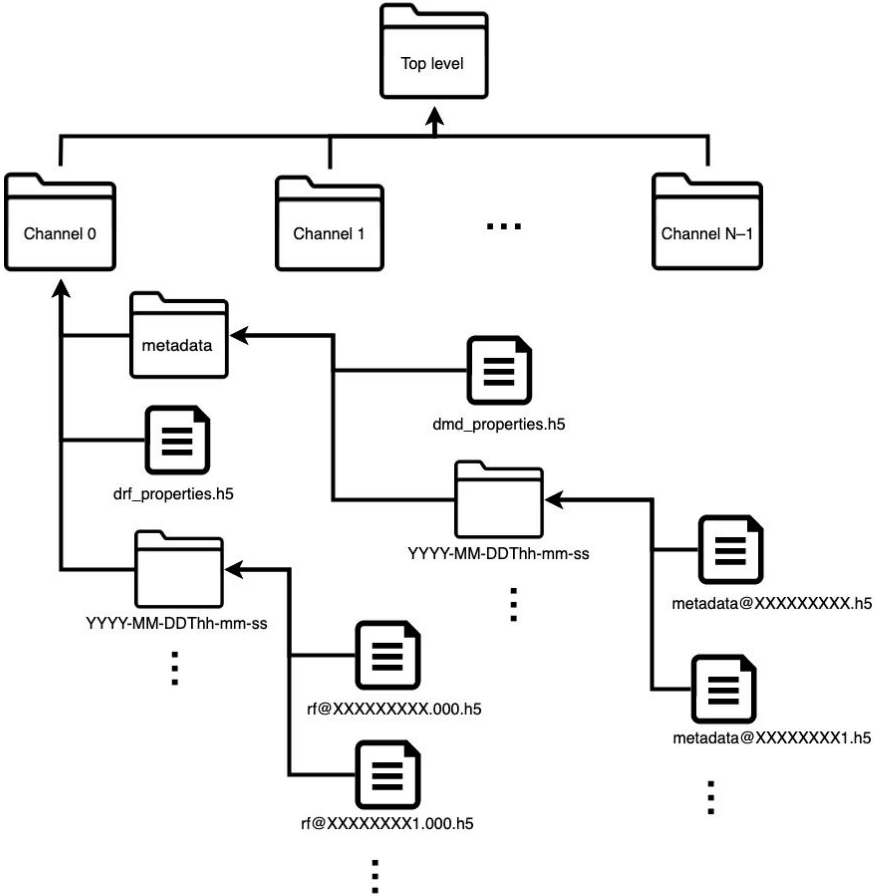
Ettus Research™
A National Instruments Company

```
{
    "global": {
        "core:datatype": "cf32",            # The datatype of the recording (here, complex 32-bit float)
        "core:sample_rate": 10000000,       # The sample rate of the recording (10 MHz, here).
        "core:version": "0.0.1",            # Version of the SigMF spec used.
        "core:description": "An example metadafile for a SigMF recording.",
    },

    "capture": [
        # The `capture` object contains a list of segments, sorted by the `sample_start` value
        {
            "core:sample_start": 0,         # The sample index that these parameters take effect.
            "core:frequency": 900000000,    # The center frequency of the recording (900 MHz, here).
            "core:time": "2017-02-01T11:33:17,053240428+01:00",
        },
        {
            "core:sample_start": 100000,    # Mandatory
            "core:frequency": 950000000,    # Now at 950 MHz
        },
    ],

    "annotations": [
        # The `annotations` object contains a list of segments, sorted by the `sample_start` value
        {
            "core:sample_start": 1000000,   # The sample index at which this annotation first applies.
            "core:sample_count": 120000,    # The number of samples that this annotation applies to.
            "core:comment": "Some text comment about stuff happening",
        },
    ],
}
```

Digital RF

- Driven by MIT Haystack Observatory
- Based on the more-generalized Hierarchical Data Format (HDF), version 5
- Specifically designed to store and organize large amounts of data
- HDF5 used by NASA, NOAA, and many other government agencies and scientific research organizations
- HDF5 has a hierarchical structure, and is more complicated than SigMF, which is flat and easy-to-parse
- **`https://github.com/MITHaystack/digital_rf`**
- **`https://en.wikipedia.org/wiki/Hierarchical_Data_Format`**
- The Digital RF software suite includes:
    - Libraries for reading and writing data in C, Python, GNU Radio (dedicated blocks), and Matlab
    - The **`thor.py`** UHD radio recorder script
    - Python tools for managing and processing Digital RF data
    - Example scripts that demonstrate basic usage
    - Example applications that encompass a complete data recording and processing chain

Digital RF


Top level
Channel 0
Channel 1
...
Channel N–1
metadata
dmd_properties.h5
drf_properties.h5
YYYY-MM-DDThh-mm-ss
YYYY-MM-DDThh-mm-ss
metadata@XXXXXXXXX.h5
rf@XXXXXXXXX.000.h5
metadata@XXXXXXXX1.h5
rf@XXXXXXXX1.000.h5

Open Datasets in SigMF and Digital RF

- Many open datasets are now being published using SigMF and Digital RF formats
- Some example public datasets:
  - "An IEEE 802.11 a/g (WiFi) massive-scale and labeled datasets for Radio Fingerprinting" from Northeastern University (NEU) in Boston
    - **`https://www.northeastern.edu/wiot/wp-content/uploads/2020/07/dataset_release.pdf`**
  - "RF Datasets For Machine Learning" from DeepSig
    - **`https://www.deepsig.ai/datasets`**
  - "Comprehensive LoRa RF Datasets for Device Fingerprinting Using Deep Learning" from Oregon State University
    - **`http://research.engr.oregonstate.edu/hamdaoui/sites/research.engr.oregonstate.edu.hamdaoui/files/release_note_2021.pdf`**
  - Data from the NASA Voyager 1 space probe from Daniel Estévez
    - **`https://destevez.net/2021/09/decoding-voyager-1/`**

gr-osmosdr

- Generic SDR hardware interface for GNU Radio
- Uses UHD under-the-hood
- Needed for GQRX
- **`https://github.com/osmocom/gr-osmosdr`**

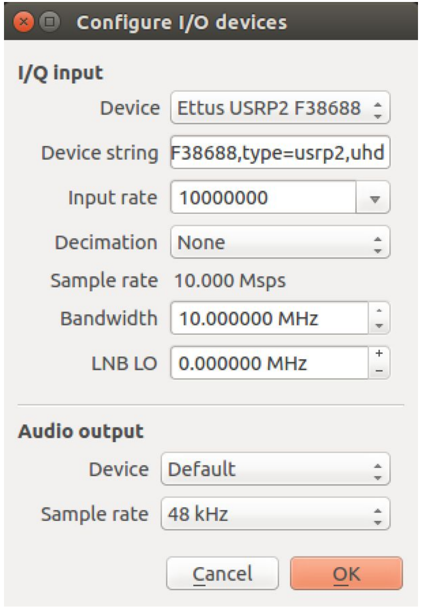
1. **`git clone git://git.osmocom.org/gr-osmosdr`**
2. **`cd gr-osmosdr/`**
3. **`mkdir build && cd build`**
4. **`cmake ../`**
5. **`make -j4`**
6. **`sudo make install`**
7. **`sudo ldconfig`**

GQRX

- A free open-source SDR receiver built on GNU Radio and QT
- Features:
  - Real-time FFT plot and waterfall
  - Demodulators for AM, SSB, NBFM (mono), WBFM (stereo)
  - Record and playback to/from IQ file
  - Basic remote control through TCP socket connection
- Created by Alexandru Csete in Denmark
- **`http://gqrx.dk/`**
- **`https://github.com/csete/gqrx`**

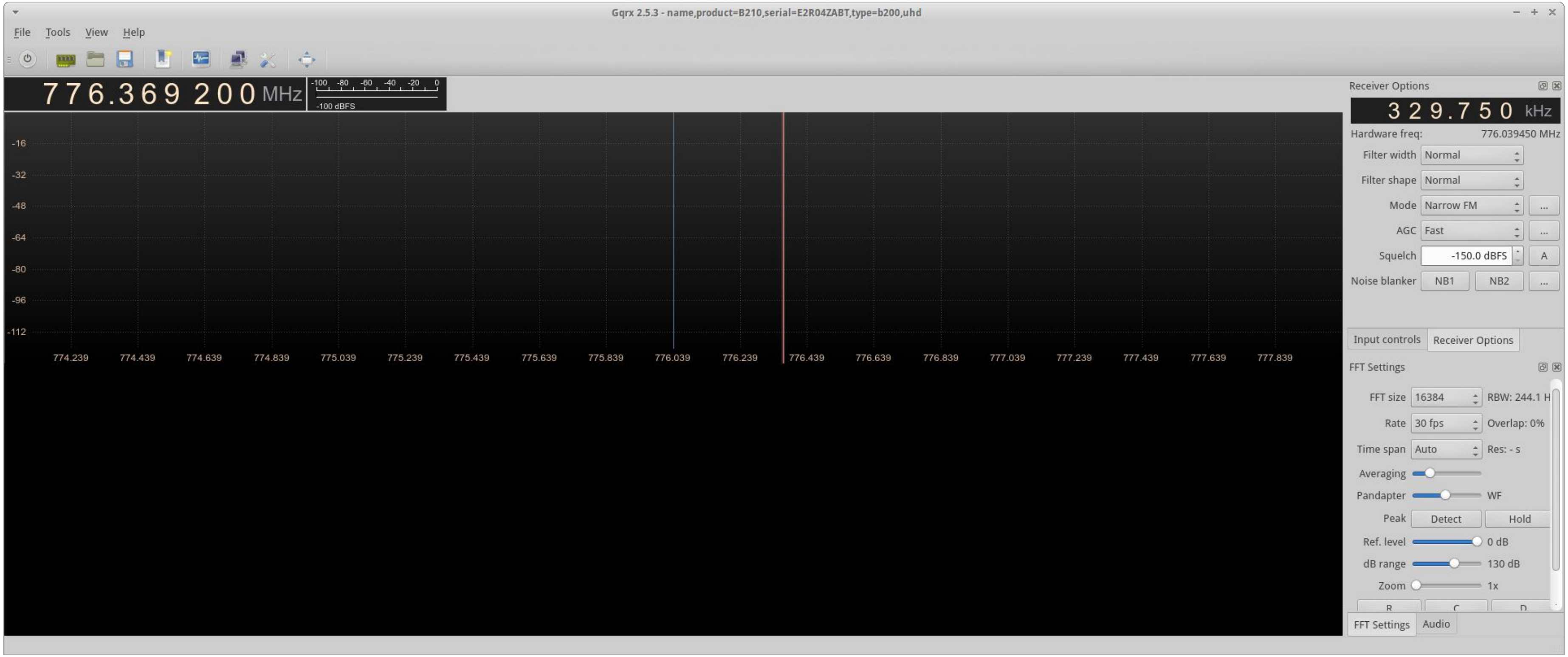
Installing GQRX

1. `sudo apt-get install qt5-default qttools5-dev-tools libqt5svg5 libqt5svg5-dev`
2. `git clone https://github.com/csete/gqrx.git`
3. `cd gqrx`
4. `mkdir build && cd build`
5. `cmake ../`
6. `make -j4`
7. `sudo make install`
8. `sudo ldconfig`

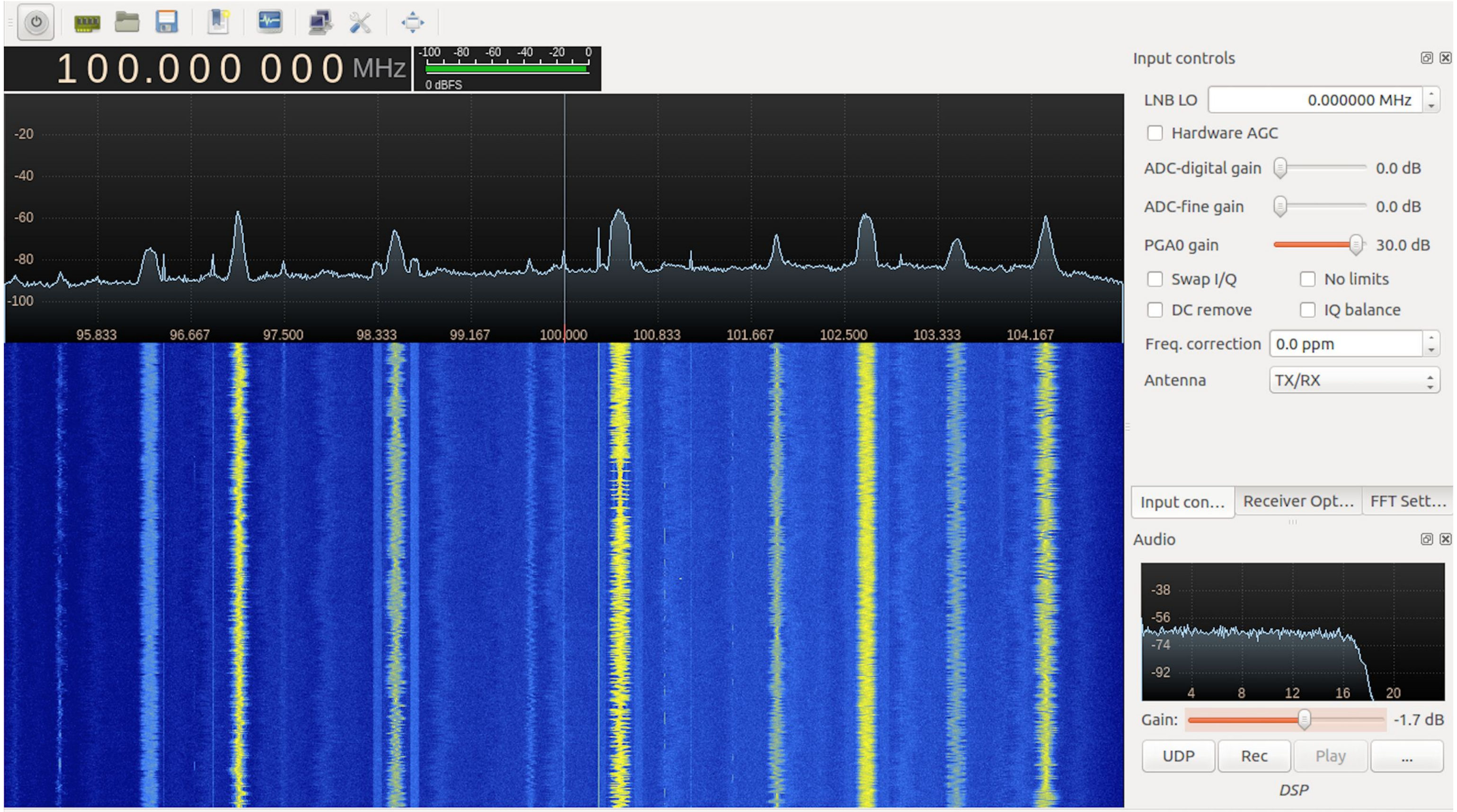


- To start, run at command prompt: **`gqrx`**
- Select Device, Set Input Rate, Decimation and Bandwidth

GQRX Screenshot

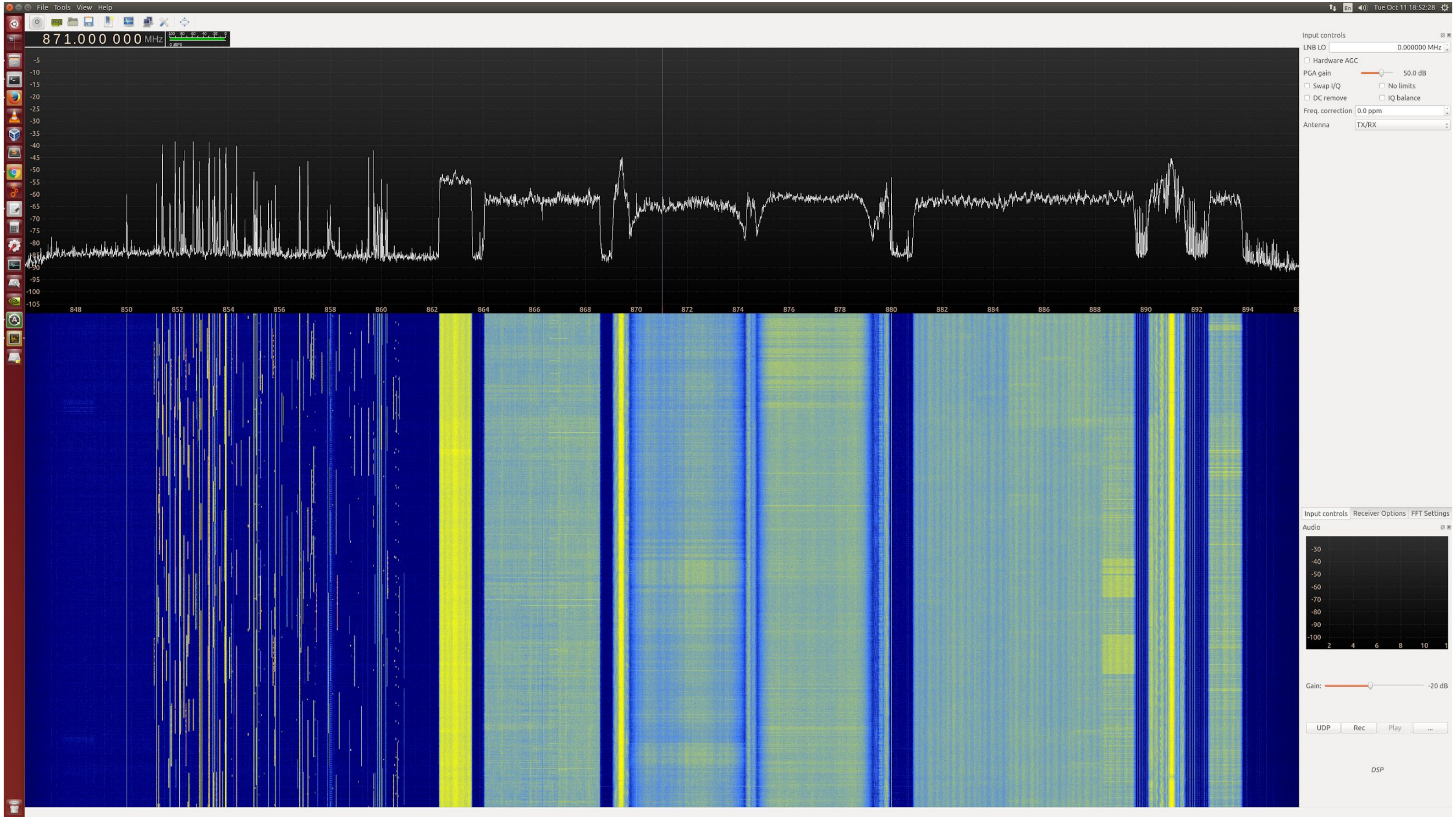


GQRX Screenshot



Demo - GQRX (1M Point FFT / 50 MS/s)

Frequency 871 MHz - NFM, P25, LTE, GSM, WCDMA



gr-paint

- Based on “Spectrum Painter” by polygon
    - Github: https://github.com/polygon/spectrum_painter
- gr-OOT created by Ron “drmpeg” Economos
- SDR based OFDM transmitter that "paints" monochrome images into the waterfall
- Converts a byte stream of image data into a 4K IFFT OFDM IQ sequence for transmission
- Github: https://github.com/drmpeg/gr-paint

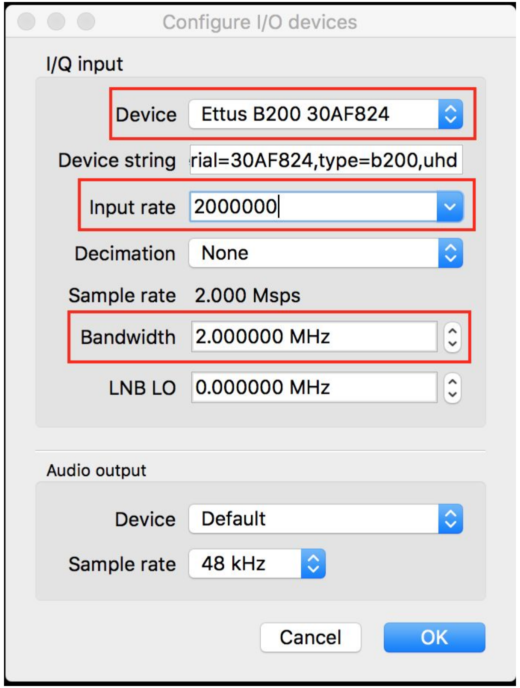
gr-paint - Installation

1. `git clone https://github.com/drmpeg/gr-paint.git`
2. `cd gr-paint`
3. `mkdir build`
4. `cd build`
5. `cmake ..`
6. `make`
7. `sudo make install`
8. `sudo ldconfig`

gr-paint - RX demo

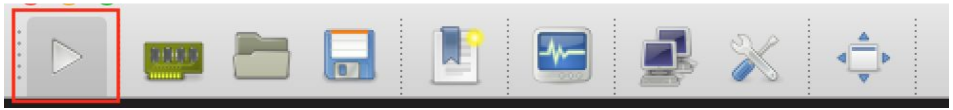
Ettus Research™
A National Instruments Company

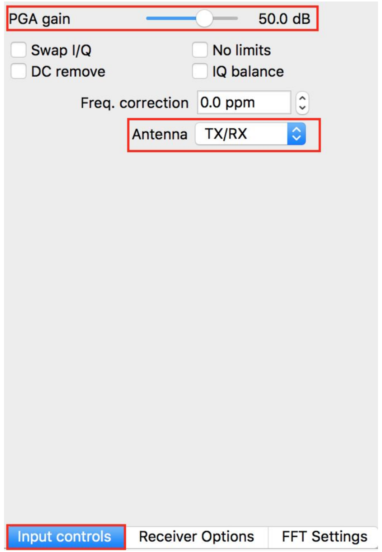
1. Open GQRX (gqrx -r)
2. Open Devices Menu (or auto popup)
3. Select USRP device
4. Set 2 MS/s sample rate
5. Set 2 MHz Bandwidth
6. Click OK



gr-paint - RX demo

1. In Main GQRX window, click "Play" button
2. Tune to 915 MHz
3. Under "Input controls" tab set Gain to 50-70dB
4. Select proper Antenna





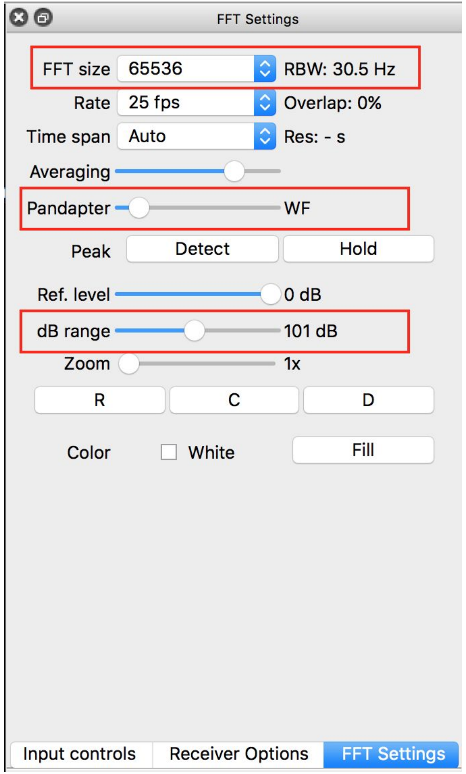
gr-paint - RX demo

1. Under “FFT Settings” tab set:

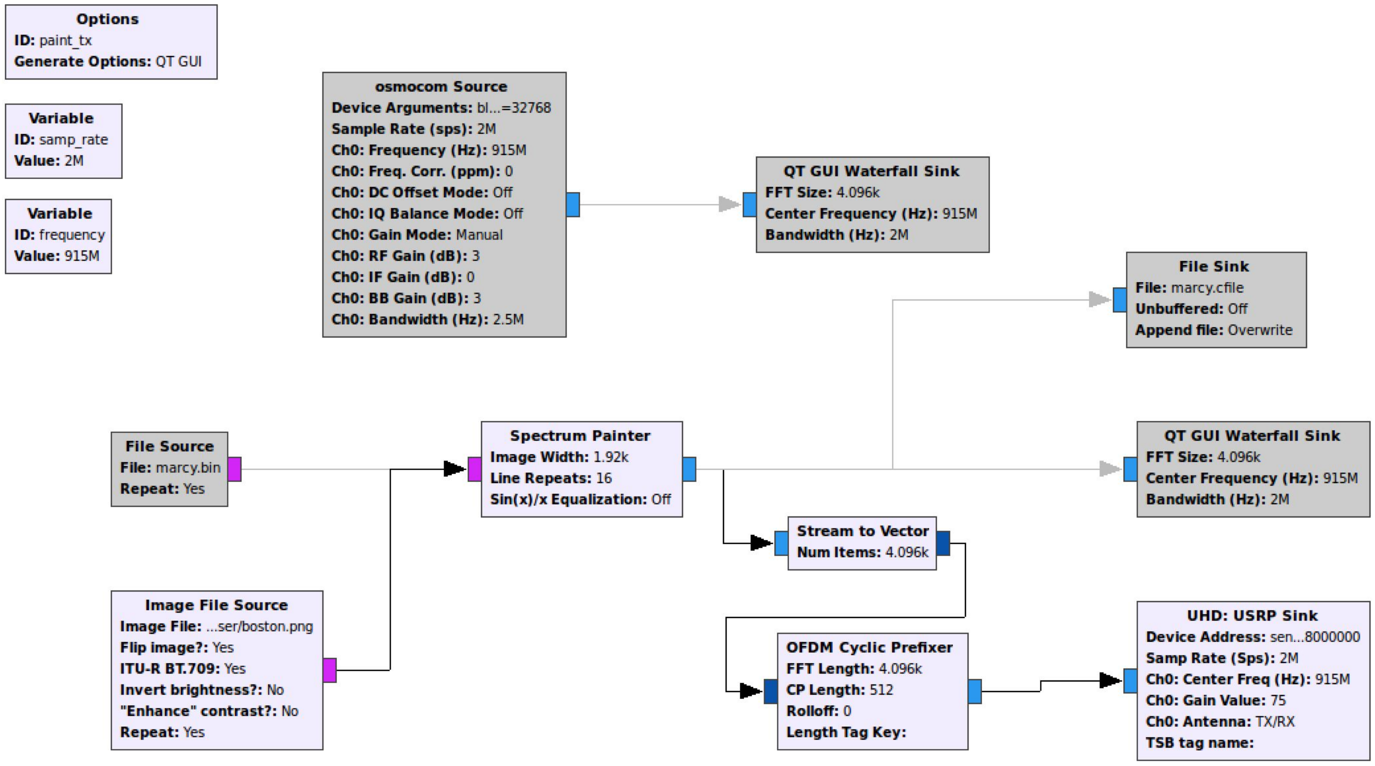
   FFT Size: 65536

   Adjust Pandapter to the left to make Waterfall larger, FFT smaller

   dB Range to ~ 100 dB



Demo - gr-paint



Demo - gr-paint

Signal Identification Resources

- SpectrumWiki - **`http://www.spectrumwiki.com/Index.aspx`**

SpectrumWiki™
The Radio Spectrum. Online.

Login/Register

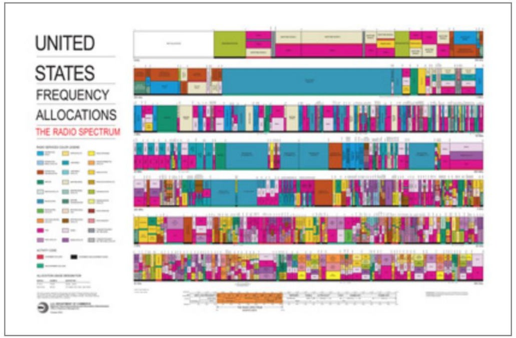
Frequency: MHz Go

HOME | UTILITIES | WIKI | REFERENCE | ABOUT

Welcome to SpectrumWiki

SpectrumWiki.com is an aggregate of information about the radio spectrum and its many uses. On a band-by-band basis, SpectrumWiki.com contains:

- Allocations
- Pertinent regulatory footnotes
- FCC rule parts
- U.S. spectrum auction revenue
- Engineering data



Additional information about the radio spectrum is contributed on a crowd-sourced basis through a wiki environment, including:

- Spectrum usage (systems and applications)
- Regulatory and legislative actions
- Band plans
- Spectrum measurements
- Historical data

and more. Please see below for more information about contributing to SpectrumWiki.

Latest Wiki Entries

Here are a few of the latest new and modified entries in SpectrumWiki:

⊞ **Terminal Doppler Weather Radar**

⊞ **PMR446 and dPMR446**

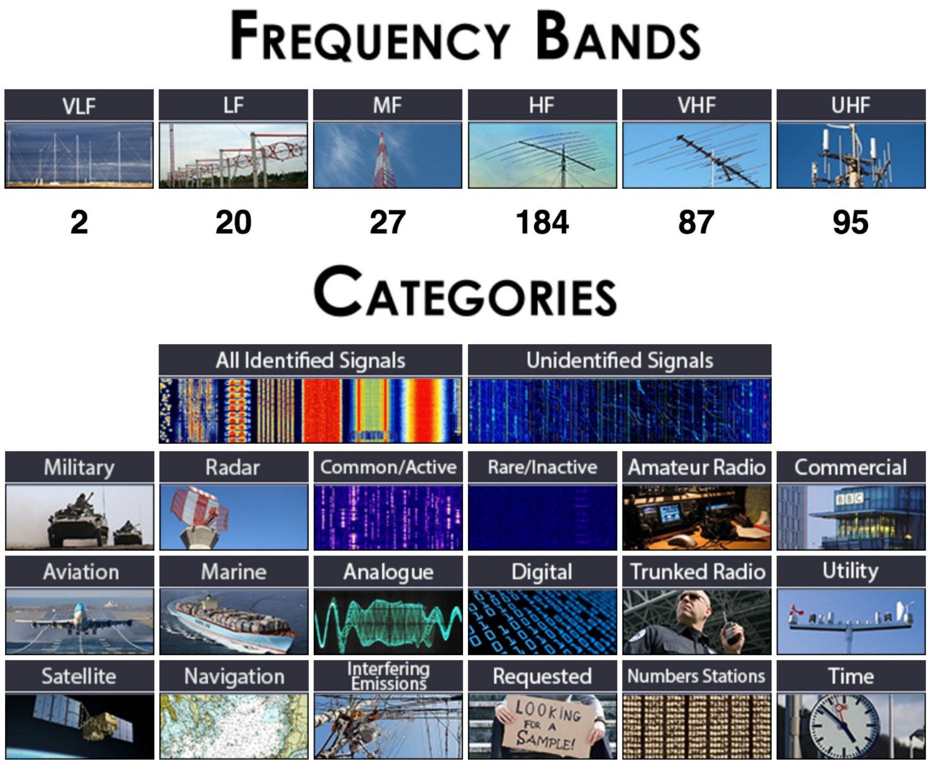
⊞ **Globalstar (MSS, ATC, & TLPS)**

⊞ **ESA Sentinel-1 Satellite C-band Synthetic Aperture Radar (C-SAR)**

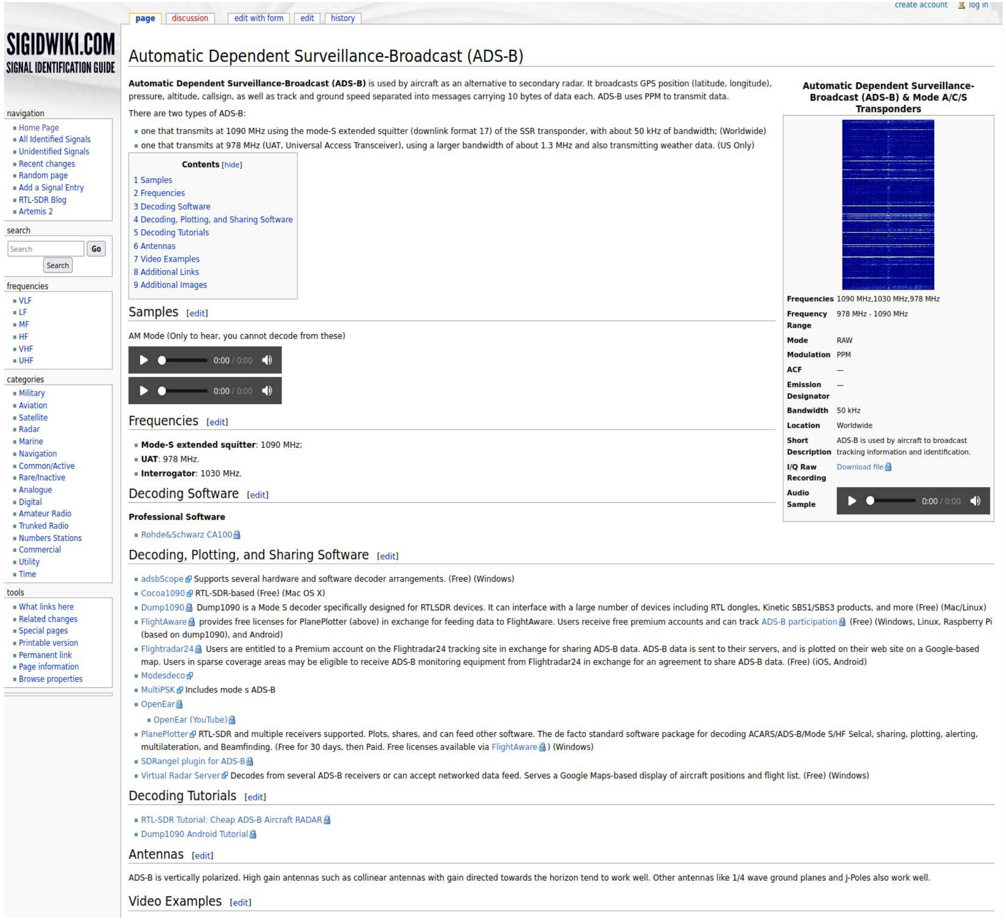
⊞ **Positive Train Control**

Signal Identification Resources

- Signal Identification Guide
  - `http://www.sigidwiki.com/wiki/Signal_Identification_Guide`



Signal Identification Resources



create account log in

page | discussion | edit with form | edit | history

SIGIDWIKI.COM
SIGNAL IDENTIFICATION GUIDE

navigation
- Home Page
- All Identified Signals
- Unidentified Signals
- Recent changes
- Random page
- Add a Signal Entry
- RTL-SDR Blog
- Artemis 2

search
Search Go Search

frequencies
- VLF
- LF
- MF
- HF
- VHF
- UHF

categories
- Military
- Aviation
- Satellite
- Radar
- Marine
- Navigation
- Common/Active
- Rare/Inactive
- Analogue
- Digital
- Amateur Radio
- Trunked Radio
- Numbers Stations
- Commercial
- Utility
- Time

tools
- What links here
- Related changes
- Special pages
- Printable version
- Permanent link
- Page information
- Browse properties

Automatic Dependent Surveillance-Broadcast (ADS-B)

Automatic Dependent Surveillance-Broadcast (ADS-B) is used by aircraft as an alternative to secondary radar. It broadcasts GPS position (latitude, longitude), pressure, altitude, callsign, as well as track and ground speed separated into messages carrying 10 bytes of data each. ADS-B uses PPM to transmit data.

There are two types of ADS-B:

- one that transmits at 1090 MHz using the mode-S extended squitter (downlink format 17) of the SSR transponder, with about 50 kHz of bandwidth; (Worldwide)
- one that transmits at 978 MHz (UAT, Universal Access Transceiver), using a larger bandwidth of about 1.3 MHz and also transmitting weather data. (US Only)

Contents [hide]

Automatic Dependent Surveillance-Broadcast (ADS-B) & Mode A/C/S Transponders

Frequencies	1090 MHz,1030 MHz,978 MHz
Frequency Range	978 MHz - 1090 MHz
Mode	RAW
Modulation	PPM
ACF	—
Emission Designator	—
Bandwidth	50 kHz
Location	Worldwide
Short Description	ADS-B is used by aircraft to broadcast tracking information and identification.
I/Q Raw Recording	Download file
Audio Sample	0:00 / 0:00

Samples [edit]

AM Mode (Only to hear, you cannot decode from these)

0:00 / 0:00

0:00 / 0:00

Frequencies [edit]

- **Mode-S extended squitter**: 1090 MHz;
- **UAT**: 978 MHz.
- **Interrogator**: 1030 MHz.

Decoding Software [edit]

Professional Software

- Rohde&Schwarz CA100

Decoding, Plotting, and Sharing Software [edit]

- adsbScope Supports several hardware and software decoder arrangements. (Free) (Windows)
- Cocoa1090 RTL-SDR-based (Free) (Mac OS X)
- Dump1090 Dump1090 is a Mode S decoder specifically designed for RTLSDR devices. It can interface with a large number of devices including RTL dongles, Kinetic SBS1/SBS3 products, and more (Free) (Mac/Linux)
- FlightAware provides free licenses for PlanePlotter (above) in exchange for feeding data to FlightAware. Users receive free premium accounts and can track ADS-B participation (Free) (Windows, Linux, Raspberry Pi (based on dump1090), and Android)
- Flightradar24 Users are entitled to a Premium account on the Flightradar24 tracking site in exchange for sharing ADS-B data. ADS-B data is sent to their servers, and is plotted on their web site on a Google-based map. Users in sparse coverage areas may be eligible to receive ADS-B monitoring equipment from Flightradar24 in exchange for an agreement to share ADS-B data. (Free) (iOS, Android)
- Modesdeco
- MultiPSK Includes mode s ADS-B
- OpenEar
  - OpenEar (YouTube)
- PlanePlotter RTL-SDR and multiple receivers supported. Plots, shares, and can feed other software. The de facto standard software package for decoding ACARS/ADS-B/Mode S/HF Selcal, sharing, plotting, alerting, multilateration, and Beamfinding. (Free for 30 days, then Paid. Free licenses available via FlightAware) (Windows)
- SDRangel plugin for ADS-B
- Virtual Radar Server Decodes from several ADS-B receivers or can accept networked data feed. Serves a Google Maps-based display of aircraft positions and flight list. (Free) (Windows)

Decoding Tutorials [edit]

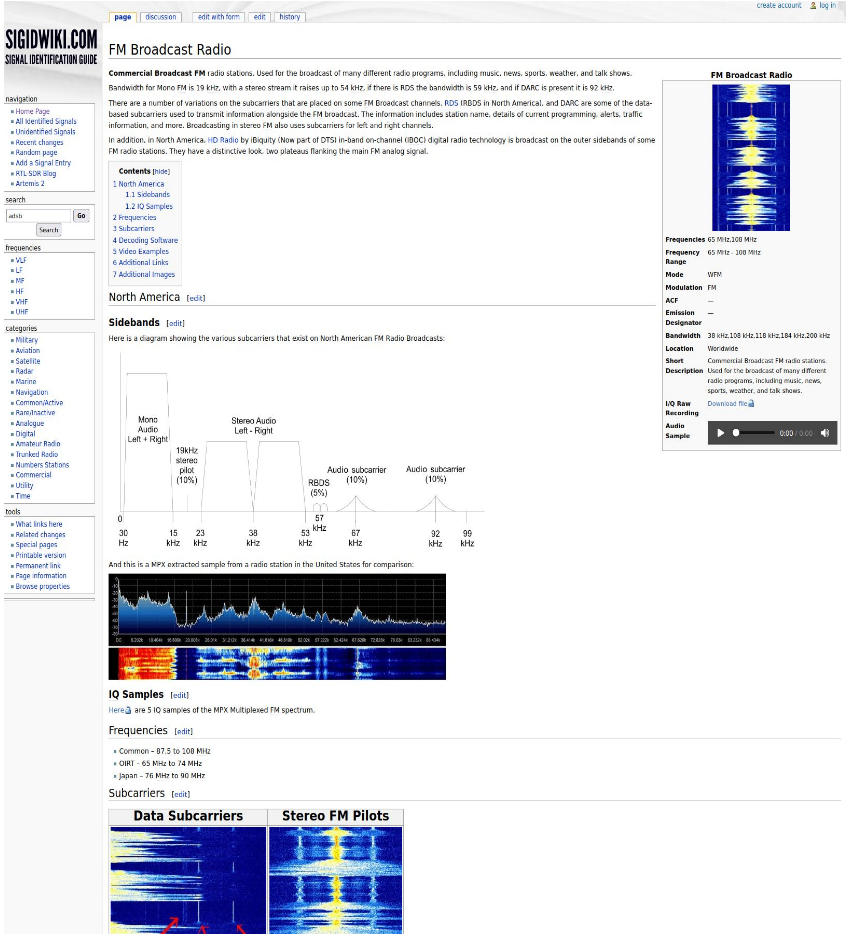
- RTL-SDR Tutorial: Cheap ADS-B Aircraft RADAR
- Dump1090 Android Tutorial

Antennas [edit]

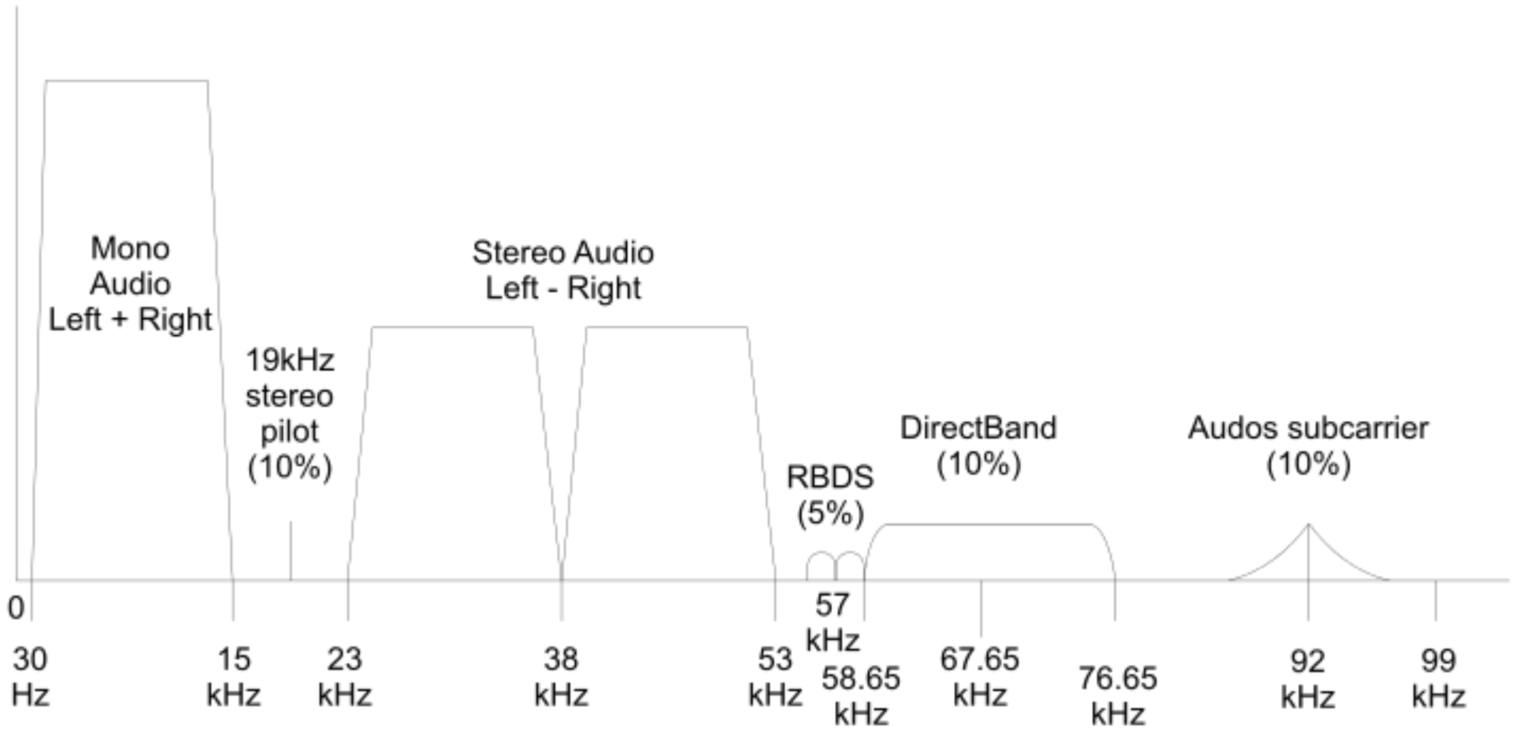
ADS-B is vertically polarized. High gain antennas such as collinear antennas with gain directed towards the horizon tend to work well. Other antennas like 1/4 wave ground planes and J-Poles also work well.

Video Examples [edit]

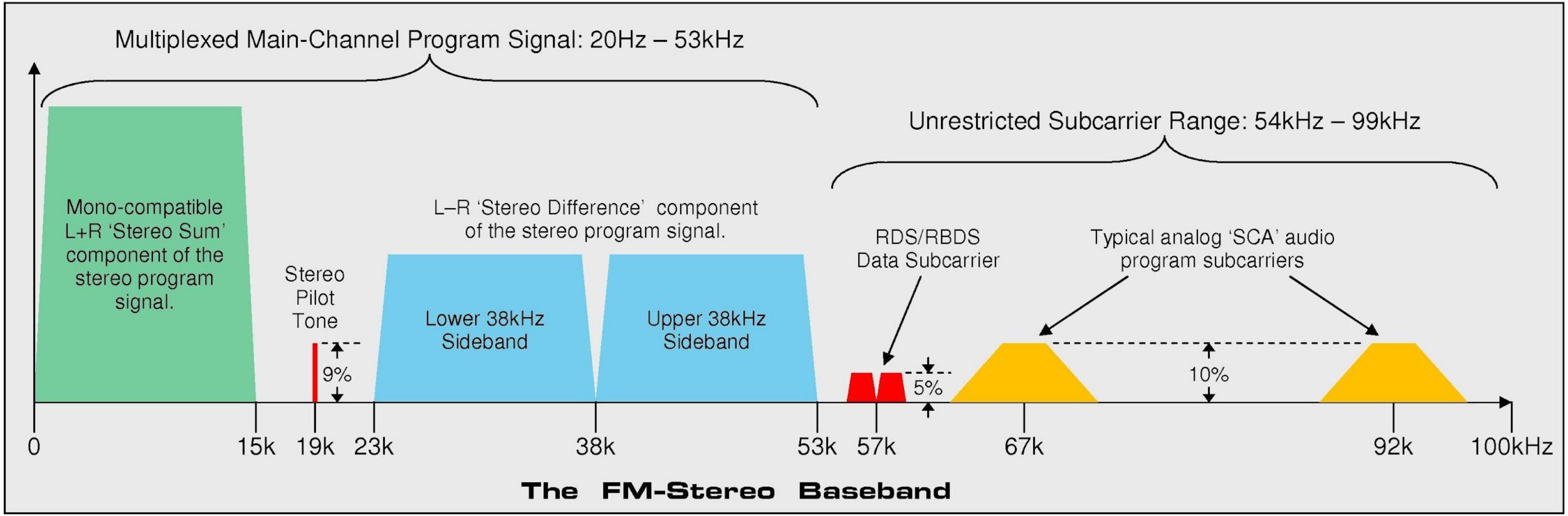
Signal Identification Resources



Broadcast FM Spectrum



Broadcast FM Spectrum



FM Radio Broadcasting

- Commercial FM radio is usually between frequencies 87.8 and 108.0 MHz (USA/Canada)
  - 101 channels total
  - Channels are 200 KHz wide, aligned to a multiple of 100 KHz
  - The FCC physically spaces local FM channels 400 KHz apart
  - In USA and Canada, only odd multiples are used
  - In parts of Europe, India, and Africa, even and odd multiples are used
  - The maximum permitted frequency error of the unmodulated carrier is specified to be within 2000 Hz of the assigned frequency
  - System was originally mono, and stereo was added later in 1960s

RDS / RBDS

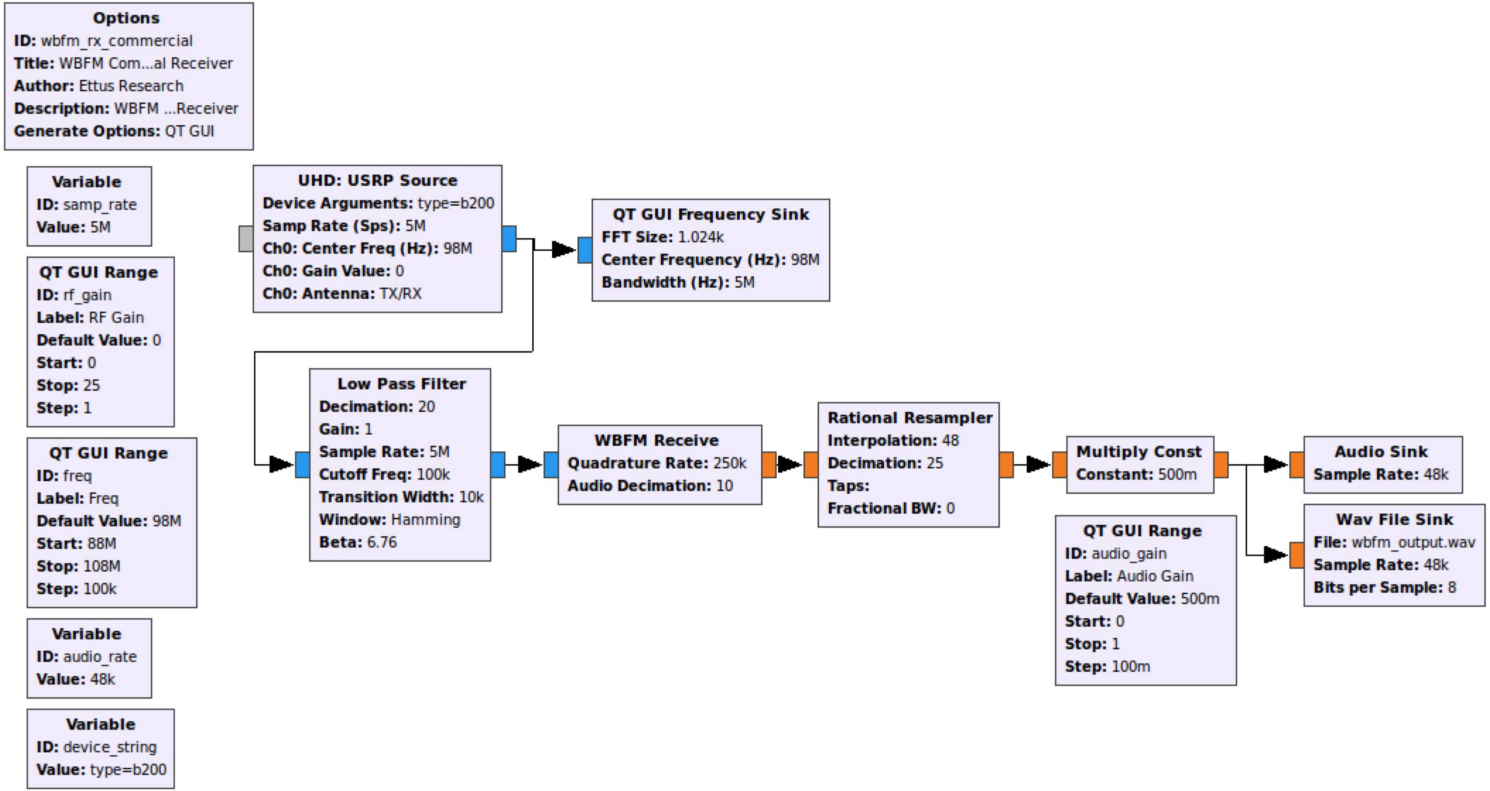
- RDS is the Radio Data System, created in 1984
    - In the USA, known as Radio Broadcast Data System (RBDS)
- Standard for embedding small amounts of digital data into commercial FM broadcasts
- RDS transmits time, station identification, programme information, and radio text (currently-playing song title and artist)
- 4 KHz-wide BPSK signal, data rate of 1187.5 bits per second, on a 57 KHz sub-carrier
- The sub-carrier is set to the third harmonic of the 19 KHz stereo pilot tone
- There are exactly 48 cycles of the sub-carrier during every data bit
- Uses CRC for error correction

RDS Information Fields

- **AF** (alternative frequencies) -- This allows a receiver to re-tune to a different frequency providing the same station when the first signal becomes too weak (e.g., when moving out of range). This is often used in car stereo systems.
- **CT** (clock time) -- Can synchronize a clock in the receiver or the main clock in a car. Due to transmission vagaries, CT can only be accurate to within 100 ms of UTC.
- **EON** (enhanced other networks) -- Allows the receiver to monitor other networks or stations for traffic programmes, and automatically temporarily tune into that station.
- **PI** (programme identification) -- This is the unique code that identifies the station. Every station receives a specific code with a country prefix. In the US, PI is determined by applying a formula to the station's call sign.
- **PS** (programme service) -- This is simply an eight-character static display that represents the call letters or station identity name. Most RDS capable receivers display this information and, if the station is stored in the receiver's presets, will cache this information with the frequency and other details associated with that preset.
- **PTY** (programme type) -- This coding of up to 31 pre-defined programme types (e.g., in Europe: PTY1 News, PTY6 Drama, PTY11 Rock music) allows users to find similar programming by genre. PTY31 seems to be reserved for emergency announcements in the event of natural disasters or other major calamities.
- **REG** (regional) -- This is mainly used in countries where national broadcasters run "region-specific" programming such as regional opt-outs on some of their transmitters. This functionality allows the user to "lock-down" the set to their current region or let the radio tune into other region-specific programming as they move into the other region.
- **RT** (radio text) -- This function allows a radio station to transmit a 64-character free-form text that can be either static (such as station slogans) or in sync with the programming (such as the title and artist of the currently playing song).
- **TA**, **TP** (traffic announcement, traffic programme) -- The receiver can often be set to pay special attention to this flag and, for example, stop the tape/pause the CD or retune to receive a traffic bulletin. The TP flag is used to allow the user to find only those stations that regularly broadcast traffic bulletins whereas the TA flag is used to signal an actual traffic bulletin in progress, with radio units perhaps performing other actions such as stopping a cassette tape (so the radio can be heard) or raising the volume during the traffic bulletin.
- **TMC** (traffic message channel) -- Digitally encoded traffic information. Not all RDS equipment supports this, but it is often available for automotive navigation systems. In many countries only encrypted traffic data is broadcast, and so an appropriate decoder, possibly tied to a subscription service, is required to use the traffic data.

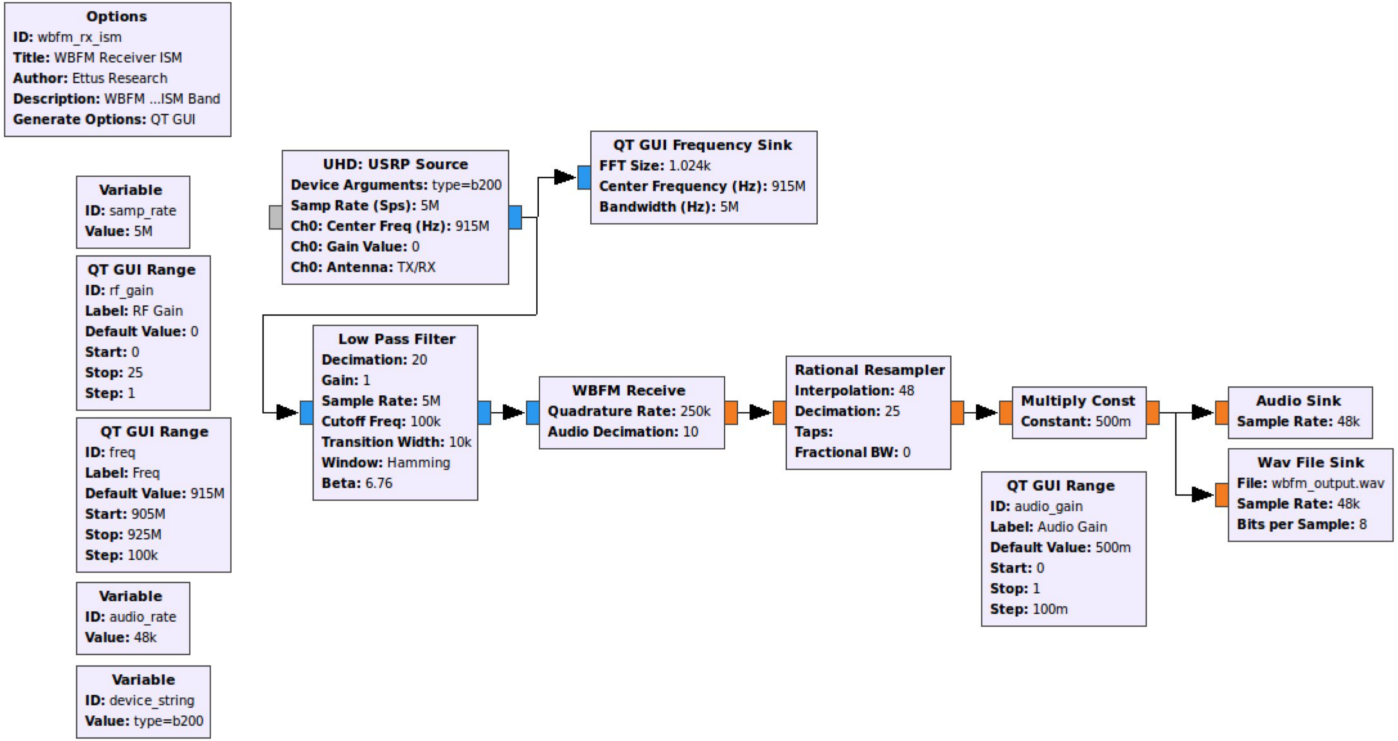
FM Receiver in GRC

Location: ~/ettus_workshop/flowgraphs/wbfm_rx_commercial.grc



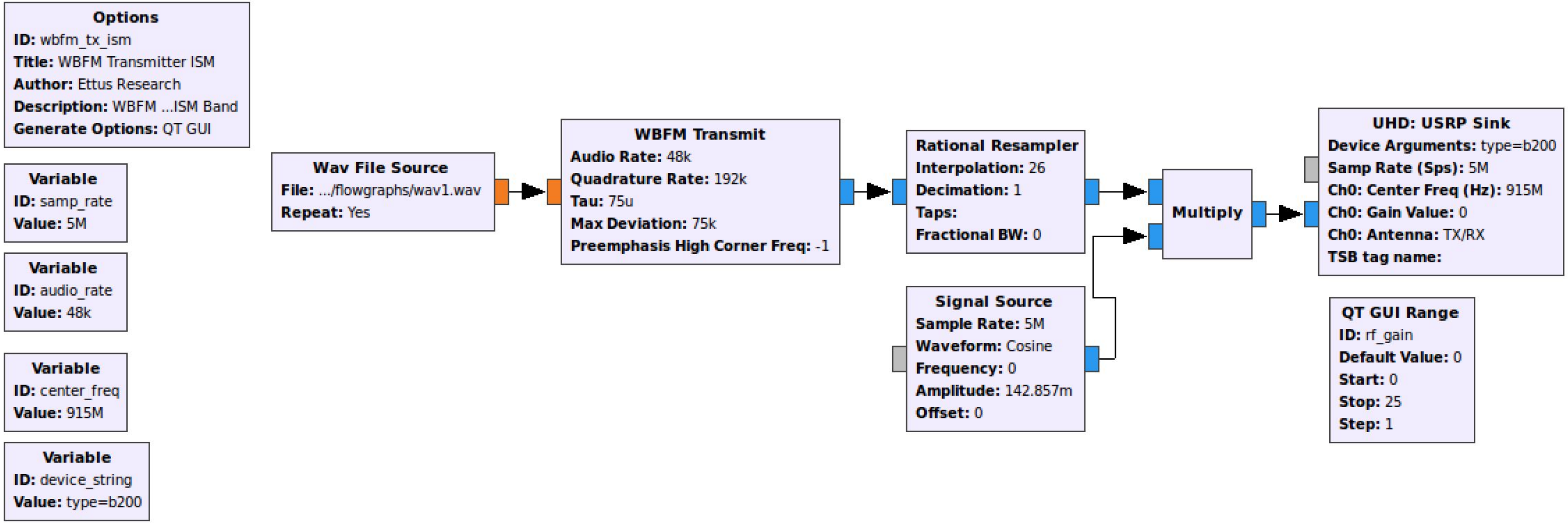
FM Receiver in GRC (ISM)

Location: ~/ettus_workshop/flowgraphs/wbfm_rx_ism.grc



FM Transmitter in GRC (ISM)

Location: ~/ettus_workshop/flowgraphs/wbfm_tx_ism.grc



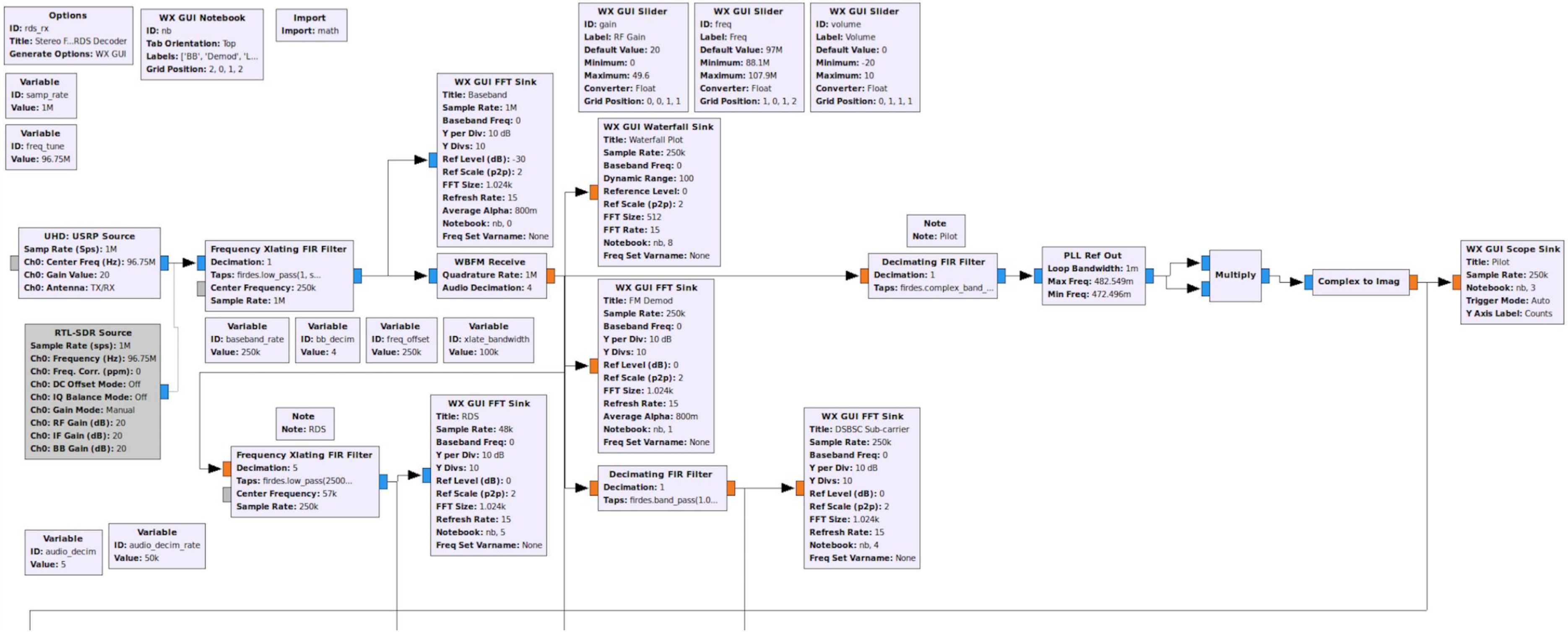
Out-of-Tree Module Installation: gr-rds

1. `sudo apt-get install liblog4cpp5-dev`
2. `git clone https://github.com/bastibl/gr-rds.git`
3. `cd gr-rds`
4. `mkdir build && cd build`
5. `cmake ../`
6. `make -j4`
7. `sudo make install`
8. `sudo ldconfig`

- In GRC, open: **`~/ettus_workshop/flowgraphs/rds_rx.grc`**
- Verify correct antenna under **`Options`** Block
- Start flowgraph
- Tune to strong station with RDS
- Adjust Gain slider if needed

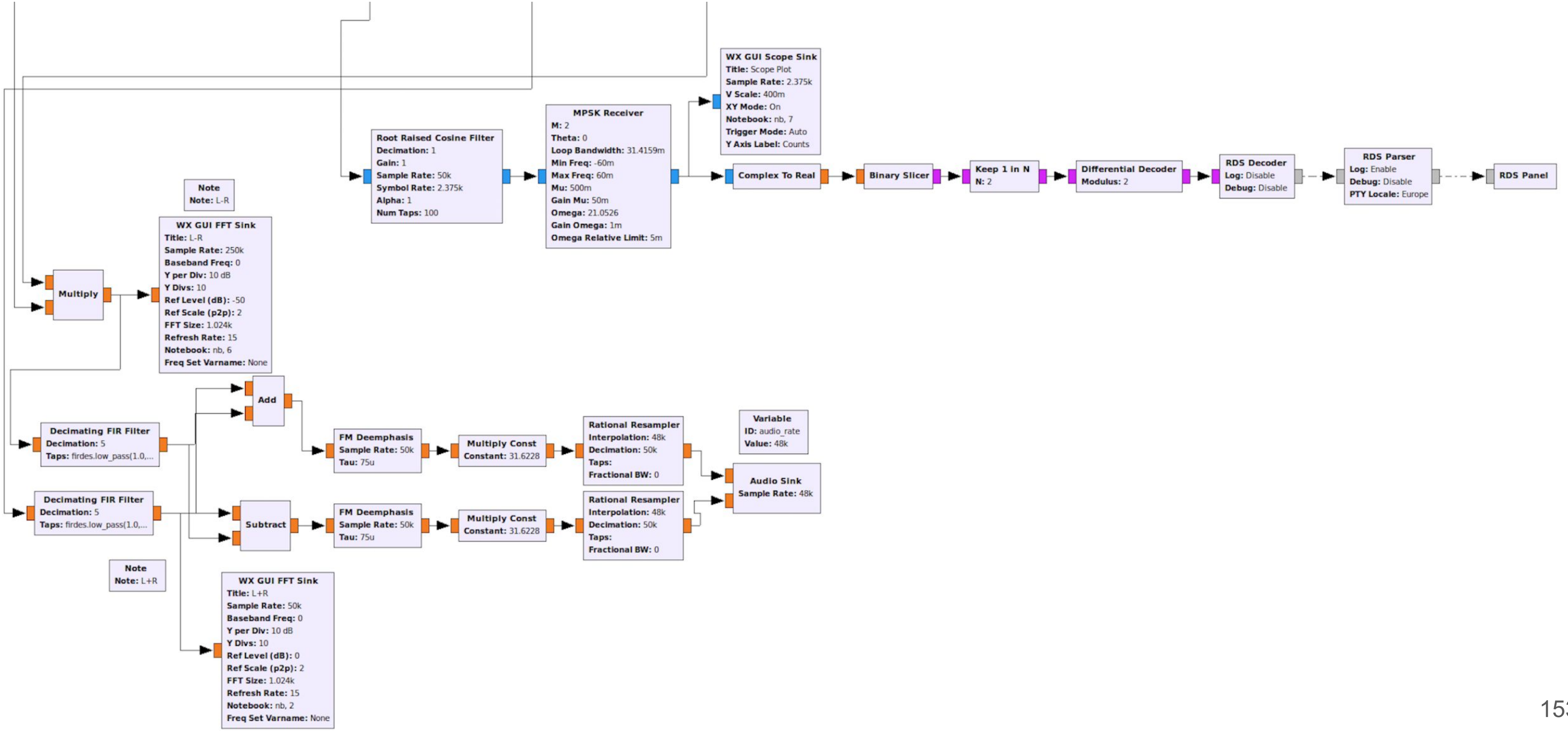
FM RDS Receiver in GRC - Part 1

Location: ~/ettus_workshop/flowgraphs/rds_rx.grc

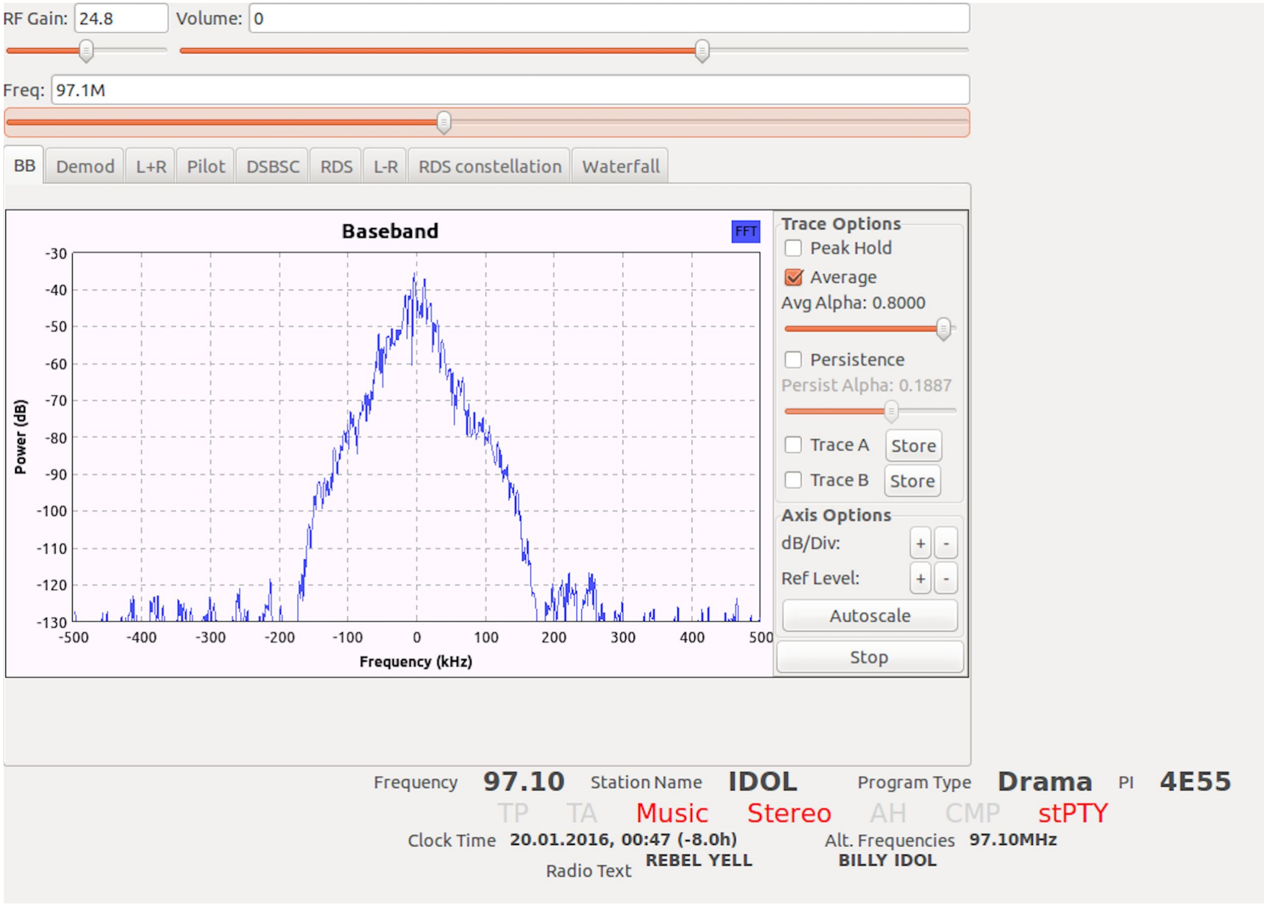


FM RDS Receiver in GRC - Part 2

Location: ~/ettus_workshop/flowgraphs/rds_rx.grc

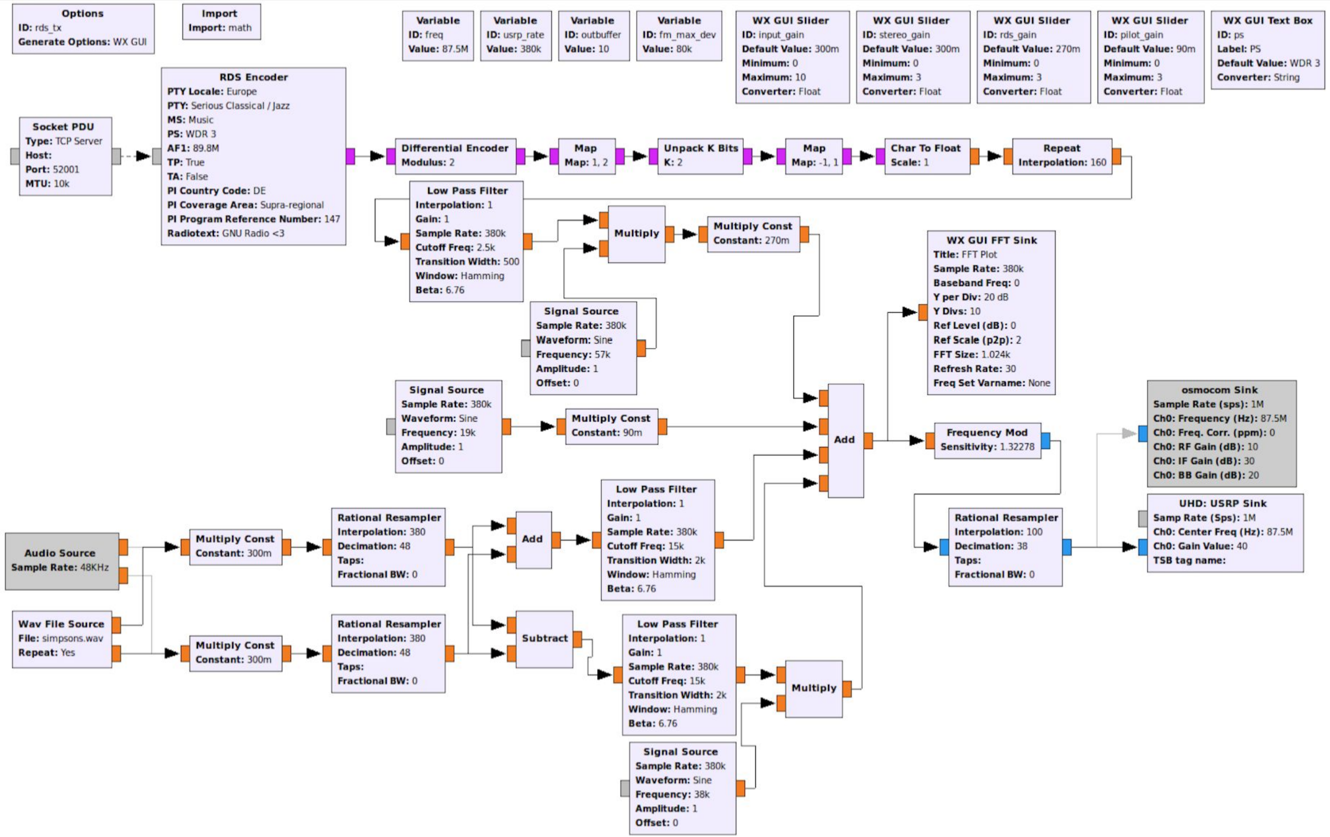


Out-of-Tree Module Installation: gr-rds



FM RDS Transmitter in GRC

Location: ~/ettus_workshop/flowgraphs/rds_tx.grc



Supported Hardware & Software

- Supported Hardware
  - Depends on processing load and I/Q data rates
  - Intel i7 or i9 CPU is ideal
    - Minimum 3.0 GHz clock speed, 6 cores
  - Intel i3 and i5 can work too *(with lighter loads and lower sampling rates)*
  - Apple M1 CPU in macOS 11, 12
    - Performance results pending
  - Raspberry Pi 3 and 4 on ARM CPU *(with lighter loads and lower sampling rates)*
- Supported Software
  - Linux is the de-facto “standard” operating system
    - Most development done on Linux, most users on Linux
    - Ubuntu and Fedora
    - Installation guides on Ettus and GNU Radio websites
  - Apple macOS also works well
  - Windows support lacking but improving, use scripts from Geof Nieboer
    - **`http://www.gcndevelopment.com/gnuradio/index.htm`**
    - **`https://github.com/gnieboer/GNURadio_Windows_Build_Scripts`**
  - Conda from Ryan Voltz at MIT Haystack
    - **`https://github.com/ryanvolz/radioconda`**
    - **`https://wiki.gnuradio.org/index.php/CondaInstall`**

Technical Resources

- GNU Radio Documentation and Wiki:
  - **`https://wiki.gnuradio.org/index.php/Main_Page`**
- Ettus Research Knowledge Base (KB) and Application Notes:
  - **`https://kb.ettus.com/`**
  - **`https://kb.ettus.com/Application_Notes`**
- USRP and UHD User Manual:
  - **`http://uhd.ettus.com/`**
  - **`http://files.ettus.com/manual/`**
- Additional Resources on the KB:
  - **`https://kb.ettus.com/Suggested_Videos`**
  - **`https://kb.ettus.com/Suggested_Reading`**
- PySDR by Dr Marc Lichtman:
  - **`https://pysdr.org/`**
- Wireless Pi Blog by Dr Qasim Chaudhari:
  - **`https://wirelesspi.com/`**
- "The Scientist and Engineer's Guide to DSP" by Dr Steven Smith:
  - **`http://www.dspguide.com/`**

Getting Help and Technical Support

- Direct email address
  - **`support@ettus.com`**
- Mailing lists **usrp-users** and **discuss-gnuradio**
  - **`https://kb.ettus.com/Mailing_Lists`**
- GNU Radio Matrix Chat Server
  - **`https://chat.gnuradio.org/`**

Upcoming SDR Events

- **GNU Radio Conference 2022 (GRCon 2022)**
  - Week of September 26 to 30
    - Tuesday, Wednesday, Thursday are the primary technical days
  - Venue is the Hilton Hotel in Washington DC
  - The videos for the 2015, 2016, 2017, 2018, 2019, 2020, 2021 events are archived online
  - **`https://www.youtube.com/c/GNURadioProject/playlists`**
  - **`https://events.gnuradio.org/e/grcon22`**
- Other relevant events:
  - **`https://kb.ettus.com/SDR_Events`**
  - **FOSDEM**: **`https://fosdem.org/2022/`**
  - **NEWSDR**: **`http://www.sdr-boston.org/`**
  - **Cyberspectrum**: **`https://www.meetup.com/Cyberspectrum/about/`**
  - **SDRA**: **`https://2022.sdra.io/`**