Ettus Research™
A National Instruments Company

GNU Radio Workshop

The Open-Source Toolchain for the USRP

Version 2019-05-07

Nate Temple
`nate.temple@ettus.com`

Neel Pandeya
`neel.pandeya@ettus.com`

Agenda (1)

- Introduction to SDR concepts
- Overview of USRP product family / architecture
- Overview of Ettus Research / NI Product mapping
- Discussion of SDR toolchains
- Overview of Radio Transport Protocols
- Overview of Linux Installation
- Overview of using Git/Github
- Discussion of programming options (C++, Python, GRC, LabView, Matlab)
- Detailed overview of UHD and GNU Radio
- Building, installing, and configuring UHD on Linux
- Hands-on step-by-step discussion of the "Getting Started" procedure
- Connecting to, and communicating with, the USRP over USB and Ethernet
- Verifying the correct operation of the USRP device
- Overview of managing multiple UHD installations
- Brief discussion on Packet Flow Errors
- Using the UHD API from C++
- Using Wireshark for debugging
- Discussion on Motherboard and Daughterboard EEPROMs

Agenda (2)

- Detailed overview of GNU Radio
- Building, installing, and configuring GNU Radio on Linux
- Overview of managing multiple GNU Radio installations, and removing installations
- Overview of DTMF
- Brief example of DTMF with GNU Radio
- Verifying USRP function using GNU Radio
- Using GNU Radio and GRC
- Overview of the components of GNU Radio
- Creating and running flowgraphs
- Using GNU Radio from Python
- Hands-on demo of introduction flowgraphs, filters
- Overview of channelizing a signal with the Frequency Xlating FIR Filter
- Overview of transmitting a signal with GNU Radio
- Overview of GNU Radio Out-of-Tree Modules / CGRAN
- Creating a GNU Radio Block / OOT
- Implementing an FM receiver in GRC
- Implementing an FM transmitter in GRC
- Implementing an full duplex FM transceiver in GRC
- Implementing an dual channel FM receiver in GRC

Agenda (3)

Ettus Research™
A National Instruments Company

- Overview of gr-rds installation
- Implementing an FM+RDS receiver in GRC
- Implementing an FM+RDS transmitter in GRC
- Implementing an OFDM/BPSK receiver in GRC
- Implementing an OFDM/BPSK transmitter in GRC
- Overview of gr-osmosdr installation
- Overview of GQRX and installation
- Using GQRX
- Overview of gr-paint and installation
- Hands-on demo of gr-paint
- Overview of gr-fosphor and installation
- Brief examples of RFNoC based gr-fosphor
- Running gr-fosphor
- Overview of Inspectrum and installation
- Examples of Inspectrum with various signals
- Remote replay attack demo
- Hands-on demo of analyzing various signals with Inspectrum
- Overview of various Signal Identification resources

Agenda (5)

Ettus Research™
A National Instruments Company

- Overview of ADS-B
- Overview of gr-air-modes and installation
- Live demo of gr-air-modes
- Discussion, overview, and demonstration of E310/E312
- Walk through of using USB WiFi adapter with E3xx
- Discussion of embedded SDR workflow, embedded Linux environment (OE), and E310/E312 SDK
- Step-by-step walk through of cross compiling UHD for the E3xx
- Step-by-step walk through of cross compiling UHD C++ application for the E3xx
- Overview of useful E3xx linux commands
- Brief discussion on ARM NEON
- Implementing an FM transmitter in GRC for the E3xx
- Implementing an FM receiver on the E3xx and streaming processed data over ZMQ
- Background on FRS Radios
- Implementing an FRS transceiver in GRC
- Overview of Live SDR Environment
- Discussion on using the Live SDR Environment as a diagnostic and debugging tool
- Discussion of synchronizing multiple USRP devices for phase-synchronous and MIMO applications
- Discussion of the TwinRX daughterboard for MIMO applications
- Overview of the FPGA toolchain (Xilinx Vivado) and building FPGA images
- Overview of X300/X310 device recovery

Agenda (6)

- Introduction and overview of RFNoC
- Discussion of using the 10 Gb Ethernet interface
- Discussion of system performance tuning and optimization
- Detailed discussion of cellular applications:
  - (OpenBTS, srsLTE, OpenLTE, Eurecom OpenAirInterface, Amarisoft)
- Detailed discussion of GNSS applications (GNSS-SDR, Skydel Solutions, etc.)
- Overview and discussion on available learning resources
- Overview of Getting Help and Technical Support
- Upcoming Events

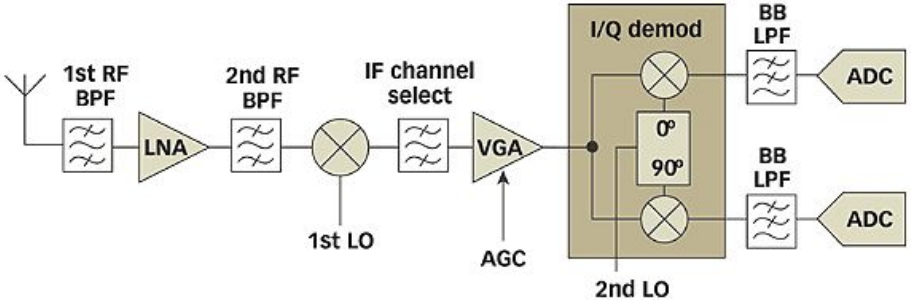
What is Software-Defined Radio (SDR)

Ettus Research™
A National Instruments Company

- A radio in which some or all of the physical-layer functions are implemented in software running on a microprocessor, and/or on an FPGA
- Algorithms from DSP and communications theory running as real-time software on a CPU and/or FPGA
- Software can be running 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 in 1991

SDR Architecture

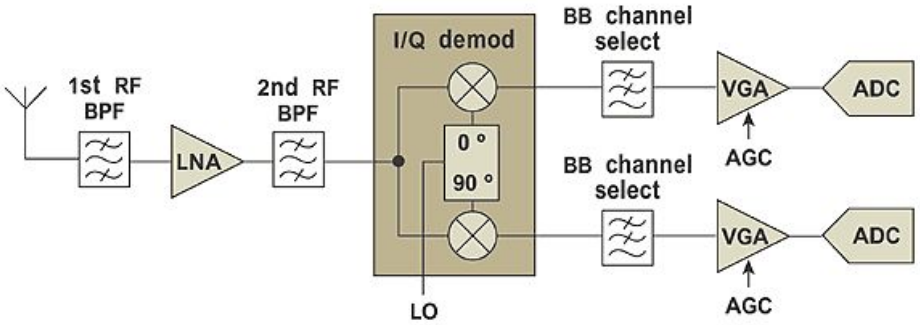
- Most radios use the classic superheterodyne receiver architecture
  - The RF signal from the antenna is mixed with a local oscillator to produce an intermediate frequency (IF) signal
  - The IF signal is a fixed lower-frequency signal which is then filtered and demodulated



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.

Why 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 Softmodems / WinModems in 1990s and 2000s
- Cellular handsets (baseband processors such as Qualcomm Snapdragon, MediaTek, etc.)
- Cellular basestations (OpenBTS, Osmocom, srsLTE, OpenLTE, Eurecom OAI, Amarisoft LTE)
- Cellular protocol stack emulation (GSM, WCDMA, LTE, 5G NR, UE/eNodeB)
- GPS Receivers and Simulators (GNSS-SDR, GPS-SDR-Sim, Skydel Solutions, Talen-X, Navigation Laboratories)
- Adaptive Radio and Cognitive Radio
- Satellite Communications (Ground Stations)
- Wireless Security Research
- Spectrum Monitoring
- Waveform Prototyping
- Wireless Systems Testing / Wireless Testbeds
- Public safety radio (Project 25)
- Radio Astronomy
- Drone Communications, Drone Detection, Drone Defense
- Direction Finding / Angle-of-Arrival
- Phased Arrays
- MIMO Systems
- Ad-hoc networks

Software Toolchains for SDR

Ettus Research™
A National Instruments Company

- Processing can be either real-time or off-line / post-processing
- C++ and the USRP Hardware Driver (UHD) API (open-source)
- GNU Radio (Python, NumPy, SciPy, Matplotlib, etc.) (open-source)
- LabVIEW™ (National Instruments)
- MATLAB™ (The MathWorks)
- Application-Specific:
    - Cellular: OpenBTS, OpenBTS-UMTS, OpenLTE, srsLTE, Amarisoft
    - GPS: GNSS-SDR, GPS-SDR-Sim, Skydel Solutions SDX
    - Amateur Radio: HDSDR, SDR#, SDR-Console, WinRAD (using ExtIO)

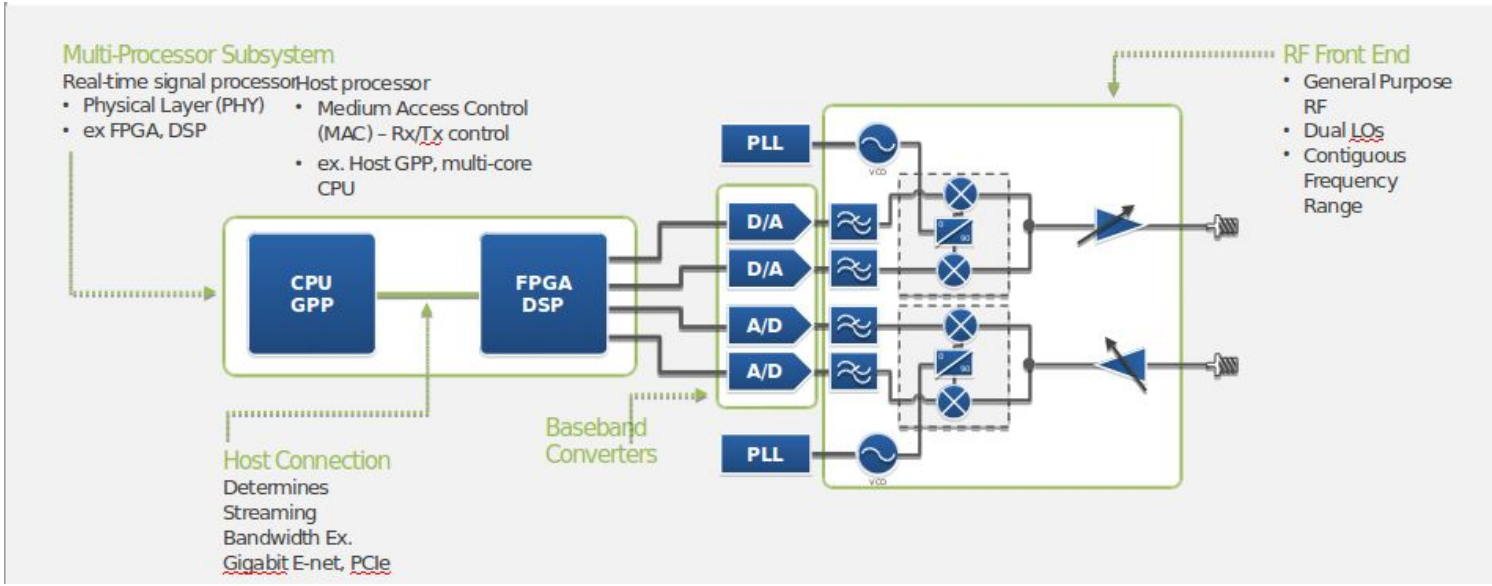
USRP™

- About Ettus Research
  - Founded in 2004 by Matt Ettus
  - Acquired by National Instruments in 2010
  - Located in Santa Clara, California, USA; Austin, Texas, USA; Dresden, Germany
- USRP Product Families:
  - B-series (B200, B210, B200mini): USB 3.0 host interface
  - N-series (N200, N210, N300/N310, N320/N321): 1 Gb Ethernet host interface
  - X-series (X300, X310): 1 and 10 Gb Ethernet host interface
  - E-series (E310, E312, E313, E320): Embedded stand-alone SDR with ARM CPU

USRP™

- USRP Daughterboards (for N-series and X-series only)
  - BasicTX and BasicRX: no tuner, 1 MHz to 250 MHz
  - LFTX and LFRX: no tuner, 0 Hz to 30 MHz
  - WBX: 50 MHz to 2200 MHz, 40 or 120 MHz capture bandwidth, 1 Tx and 1 Rx
  - SBX: 400 MHz to 4400 MHz, 40 or 120 MHz capture bandwidth, 1 Tx and 1 Rx
  - CBX: 1200 MHz to 6000 MHz, 40 or 120 MHz capture bandwidth, 1 Tx and 1 Rx
  - UBX: 10 MHz to 6000 MHz, 40 or 160 MHz capture bandwidth, 1 Tx and 1 Rx
  - TwinRX: 10 MHz to 6000 MHz, 80 MHz capture bandwidth, Dual Rx Only, No Tx

USRP Architecture



USRP Architecture

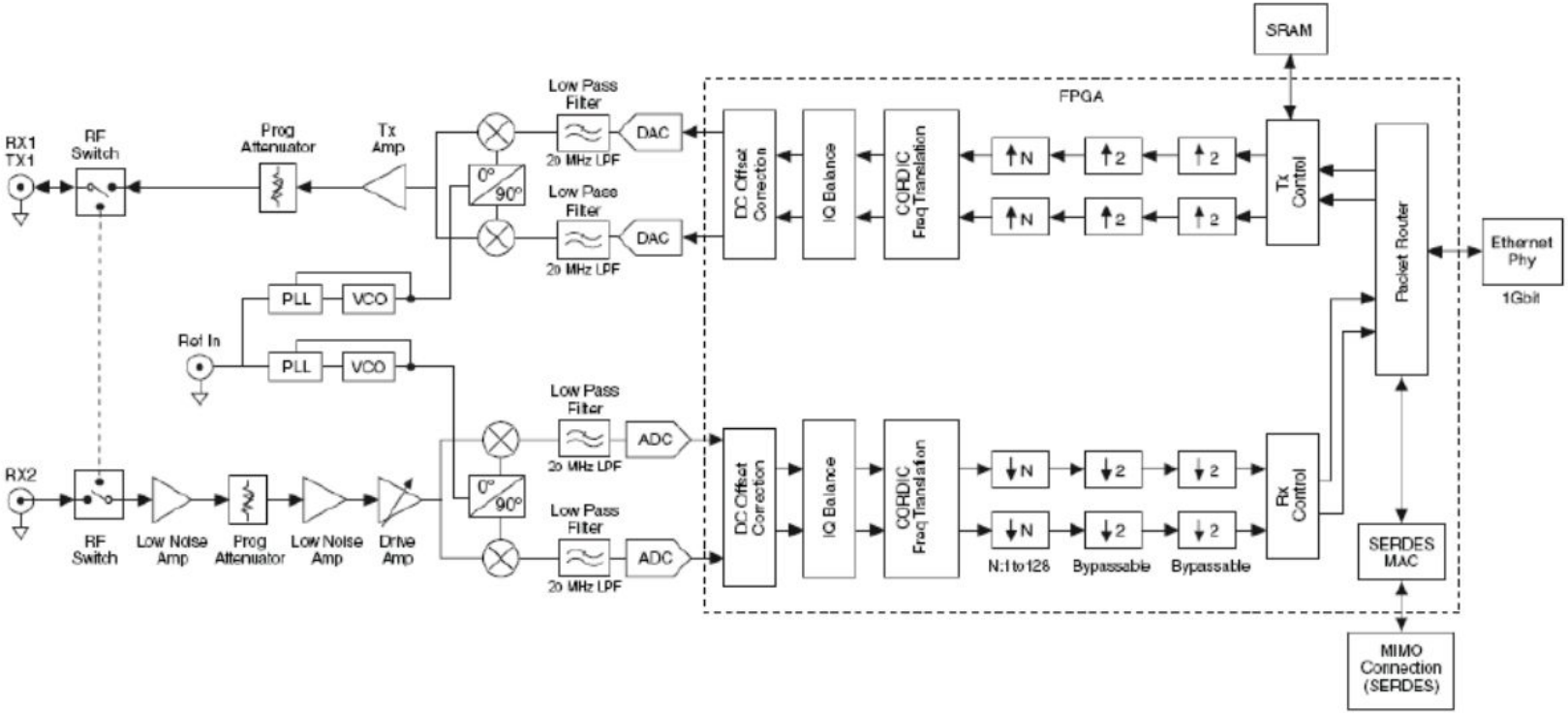
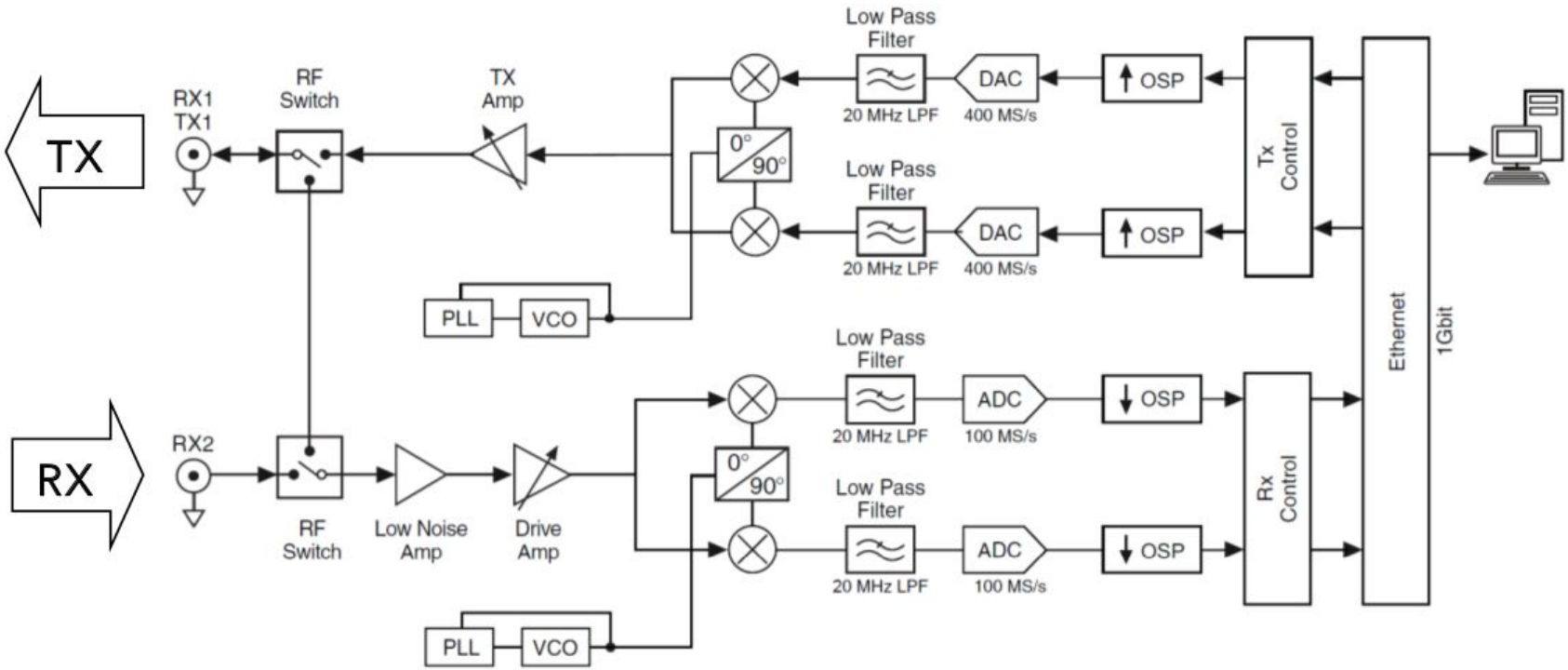


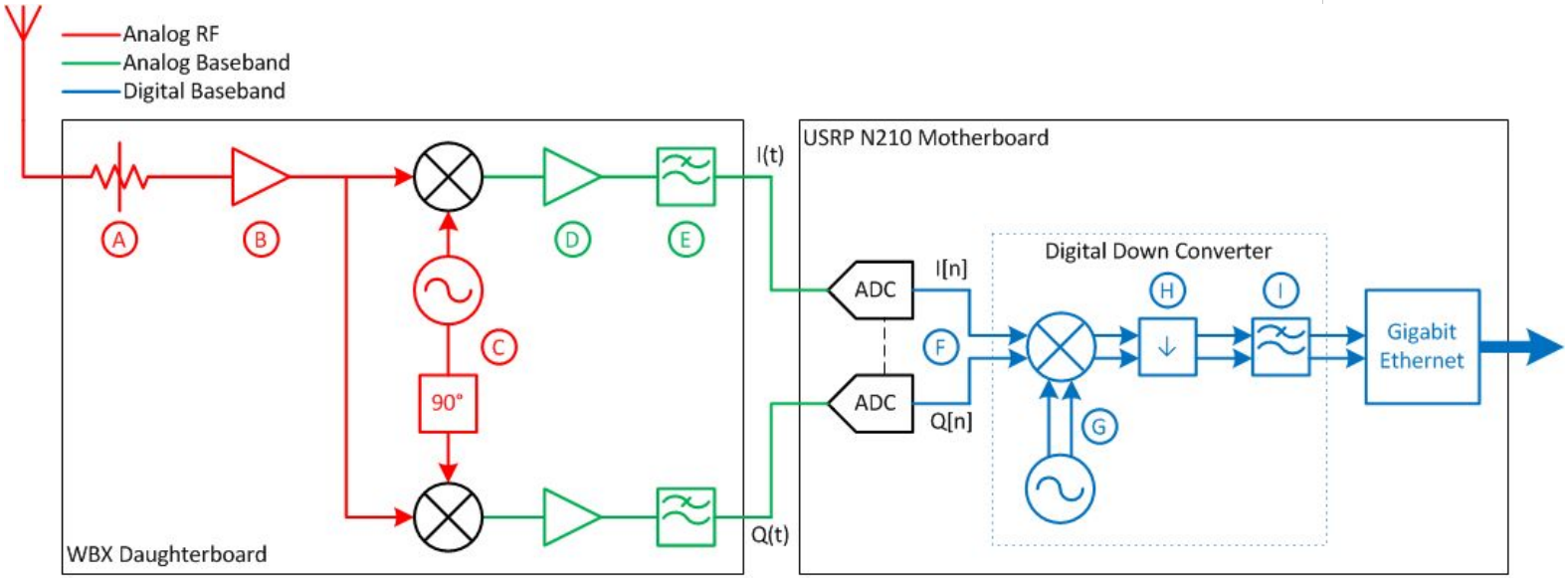
Figure 4 - Second-Generation USRP Architecture

USRP Architecture



Analog RF Transceiver | Fixed Function FPGA | PC

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

USRP Model Comparison

Ettus Research™
A National Instruments Company

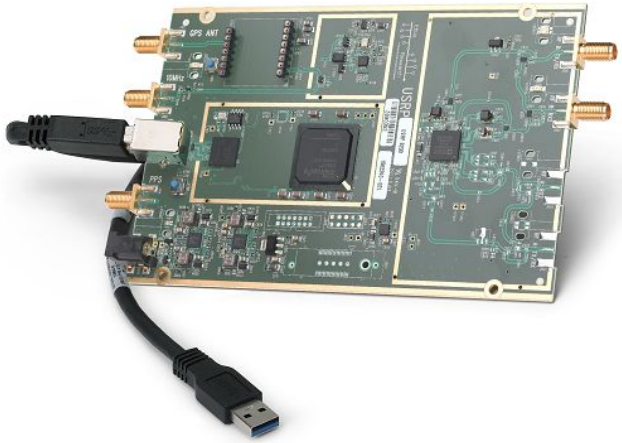



	Bus B2xx	Embedded E3xx	Networked N2xx	High Performance X3xx
Frequency (Hz)	70 M – 6 G	70 M – 6 G	DC-30M & 10M–6G	DC-30M & 10M–6G
Bandwidth	56MHz (32 MHz in 2x2)	56MHz (32 MHz in 2x2)	40 MHz	160 MHz
Channels	2 Tx, 2 Rx	2 Tx, 2 Rx w/ filter banks	1 Tx, 1Rx	2 Tx, 2 Rx
RF Performance	Good	Good	Better	Best
Architecture	Integrated RF	Integrated RF	RF Daughterboard	RF Daughterboards
Communication	USB	Embedded	1GbE	10GbE or PCIe
MIMO Capability	2x2	2x2	Up to 2x2	2x2 to 256x256
LabVIEW Support	Yes	No	Yes	Yes
NI Version	USRP-290x	None	USRP-292x USRP-293x	USRP-294x USRP295x
S/W Ecosystem	GNU Radio C++ MatLab Xilinx ISE	GNU Radio C++ Xilinx Vivado C Coder HDL Coder	GNU Radio C++ MatLab Xilinx ISE	GNU Radio C++ MatLab Xilinx Vivado Simulink C Coder HDL Coder

*Same Daughterboard in each slot

USRP B200

- Xilinx Spartan 6 XC6SLX75 FPGA
- Analog Devices AD9364 RFIC direct-conversion transceiver
- Frequency range: 70 MHz - 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- Full duplex, SISO (1 Tx & 1 Rx)
- Fast and convenient bus-powered USB 3.0 connectivity
- Optional Board Mounted GPSDO



USRP B210

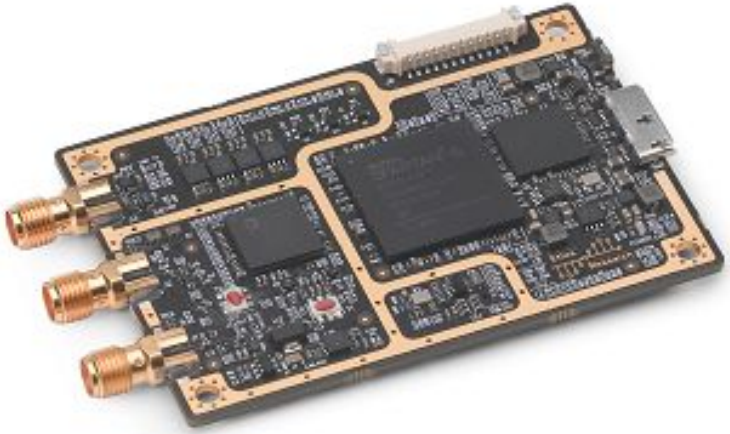
- Xilinx Spartan 6 XC6SLX150 FPGA
- Analog Devices AD9361 RFIC direct-conversion transceiver
- Frequency range: 70 MHz - 6 GHz
- Up to 56 MHz of instantaneous bandwidth (61.44MS/s quadrature)
- Full duplex, MIMO (2 Tx & 2 Rx)
- Fast and convenient bus-powered USB 3.0 connectivity
- Optional Board Mounted GPSDO

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
- Full duplex, SISO (1 Tx & 1 Rx)
- Power from the USB 3.0 bus
- Low SWaP (size of a business card)

- Optional configuration: B200mini-i
  - Industrial-grade XC6SLX75 FPGA

- Optional configuration: B205mini-i
  - Industrial-grade XC6SLX150 FPGA



USRP E310/E312/E313

- Wide frequency range: 70 MHz to 6 GHz
- Based on Analog Devices AD9361 Transceiver
- Up to 56 MHz of instantaneous bandwidth
- 2x2 MIMO transceiver
- RX and TX filter banks
- Xilinx Zynq 7020 FPGA
- ARM Cortex A9 866 MHz dual-core CPU
- Up to 10 MS/s sample data transfer rate to ARM processor
- Synchronization with PPS time reference
- Integrated GPS receiver (**not** GPSDO)
- 9-axis inertial measurement unit
- Rich set of peripherals such as host USB and 1 Gigabit Ethernet

USRP E310/E312/E313

FRONT

REAR

Front / Top Row:

- TRX-A – Channel A, Tx/Rx
- RX2-A – Channel A, Rx
- TRX-B – Channel B, Tx/Rx
- RX2-A – Channel B, Rx

Front / Bottom Row:

- Power Button
- microSD Card Slot
- GPS Antenna Port
- 1 PPS Sync

Rear:

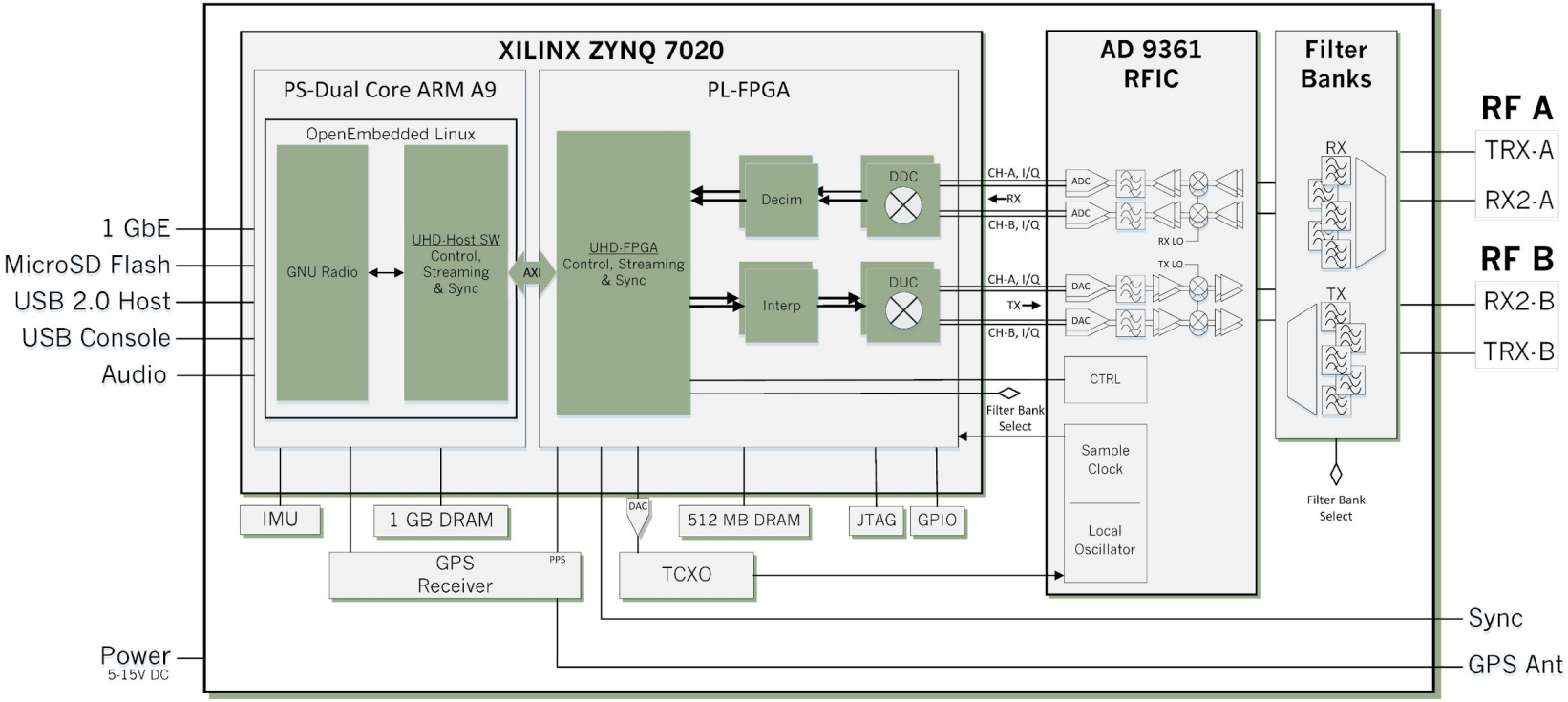
- Power Button
- microSD Card Slot
- GPS Antenna Port
- 1 PPS Sync

USRP E310/E312/E313

USRP E310/E312/E313

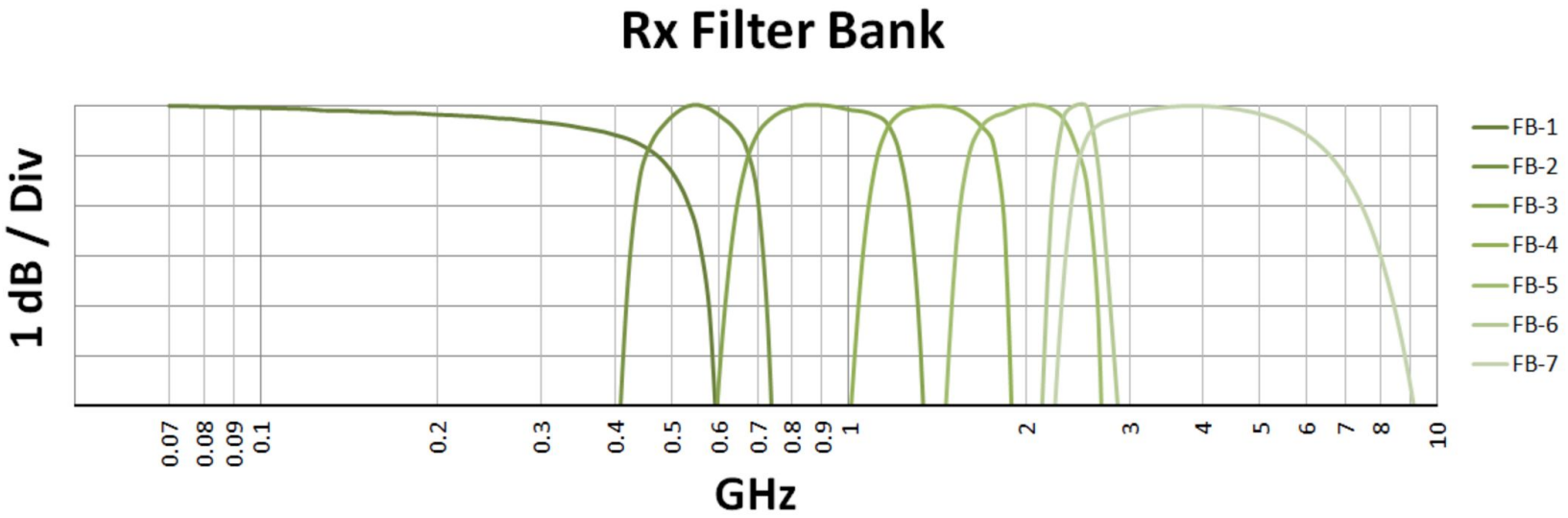
- OpenEmbedded build framework
- Supports UHD and RFNoC
- GNU Radio support maintained by Ettus Research
- Optional Battery (E312)
- Optional IP67/PoE Configuration (E313)

USRP E310/E312/E313

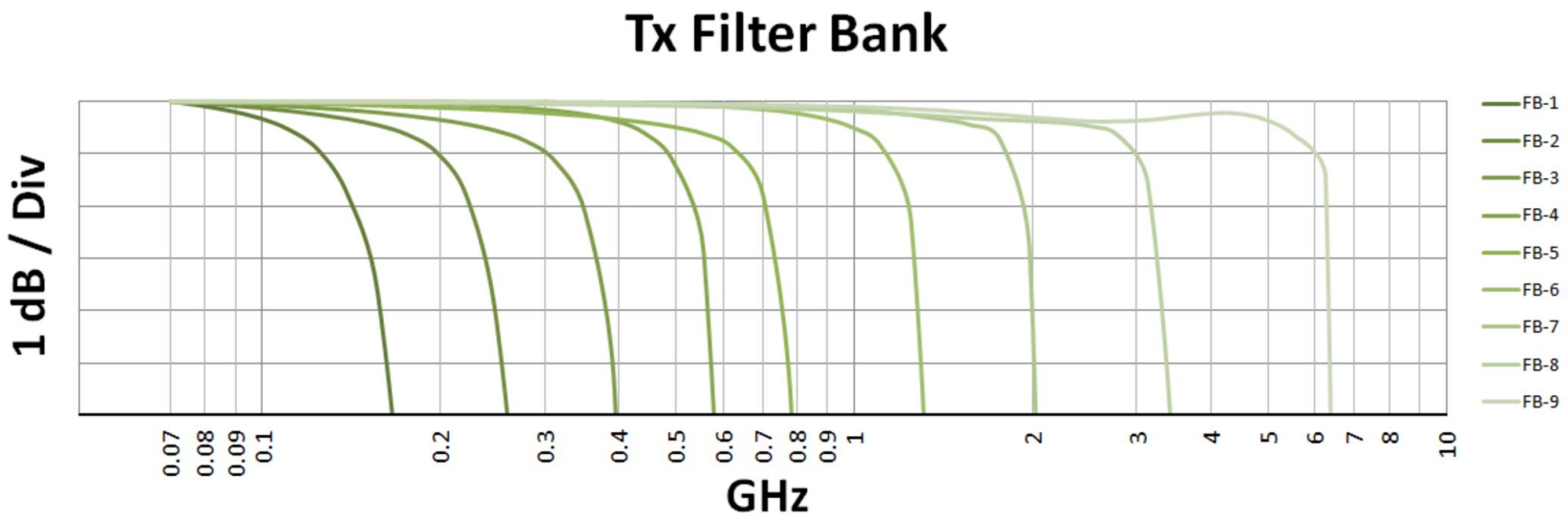


USRP E310 Block Diagram

USRP E310/E312/E313



USRP E310/E312/E313



USRP E320

Ettus Research™
A National Instruments Company

- Based on Analog Devices AD9361 (70 MHz to 6 GHz frequency range, 56 MHz bandwidth)
- Xilinx Z7045 FPGA *(3x to 4x larger than E310 FPGA)*
- Single SFP+ Port for 10 Gbps Ethernet streaming
- Support for 12.5 Gbps Aurora streaming
- Enclosure also acts as a passive heatsink
- Fan header and attach points for Zynq (for convection-cooled apps)
- Single PCB to make OEM integration easier
- Temperature sensors on AD9361 and Xilinx Zynq FPGA
- Battery connector for optional external battery to enable portability
- Low SWaP, 3U Eurocard Size (160 x 100 mm)
- Jackson Labs LTE-Lite GPSDO
- MEMS Gyroscope

USRP E320 Photo

USRP E320 Photo

USRP N200 / N210

- Xilinx Spartan 3A-DSP XC3SD1800A (N200)
- Xilinx Spartan 3A-DSP XC3SD3400A (N210)
- 50 MHz (sc8) / 25 MHz (sc16) RF bandwidth
- Frequency range: DC - 6 GHz with suitable daughterboard
- Gigabit Ethernet connectivity
- Onboard FPGA processing
- Ability to lock to external 5 or 10 MHz clock reference
- TCXO Frequency Reference (~2.5ppm)
- Optional internal GPS locked reference oscillator

USRP X300 / X310

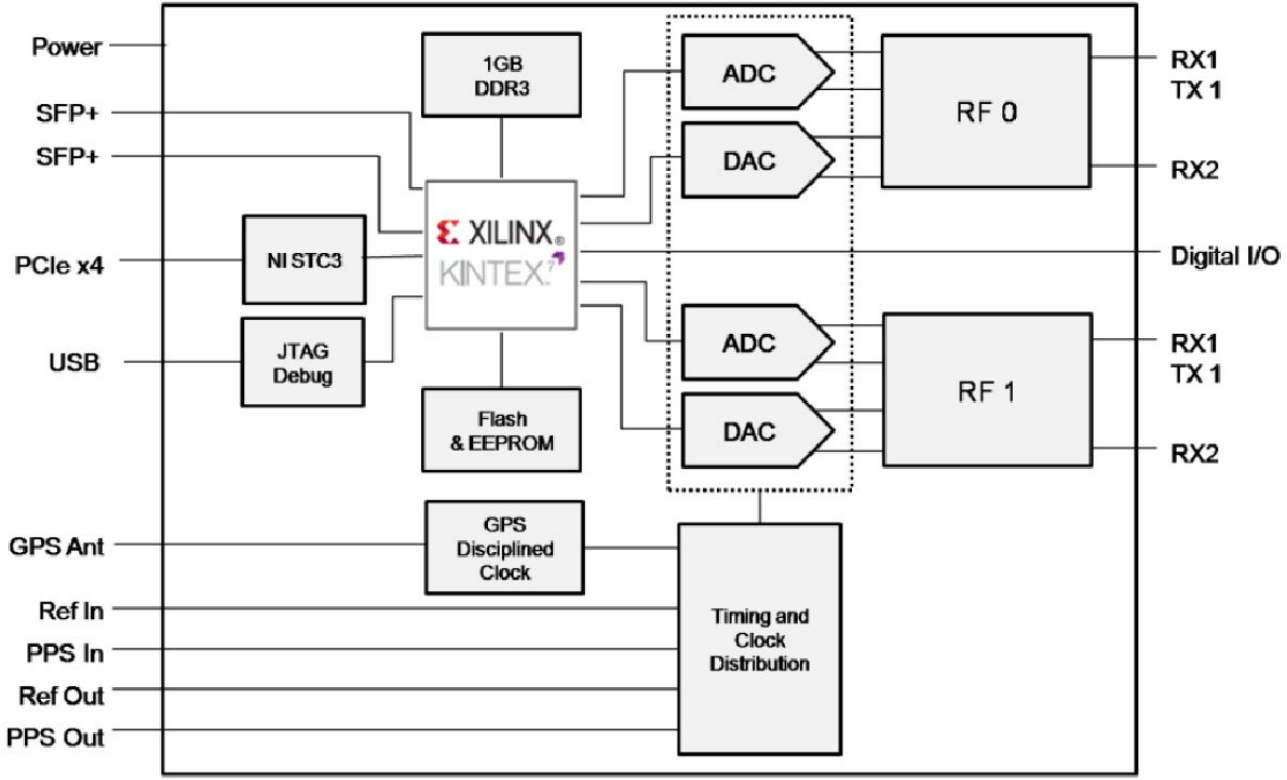
- Xilinx Kintex-7 XC7K325T FPGA (X300)
- Xilinx Kintex-7 XC7K410T FPGA (X310)
- Frequency range: DC to 6 GHz with daughterboard
- Up to 160 MHz bandwidth per channel
- Two wide-bandwidth RF daughterboard slots
- Optional internal GPSDO
- Multiple high-speed interfaces
    - Single 1 Gbps Ethernet
    - Single 10 Gbps Ethernet
    - Dual 10 Gbps Ethernet
    - PCIe Gen 2

USRP X300 / X310

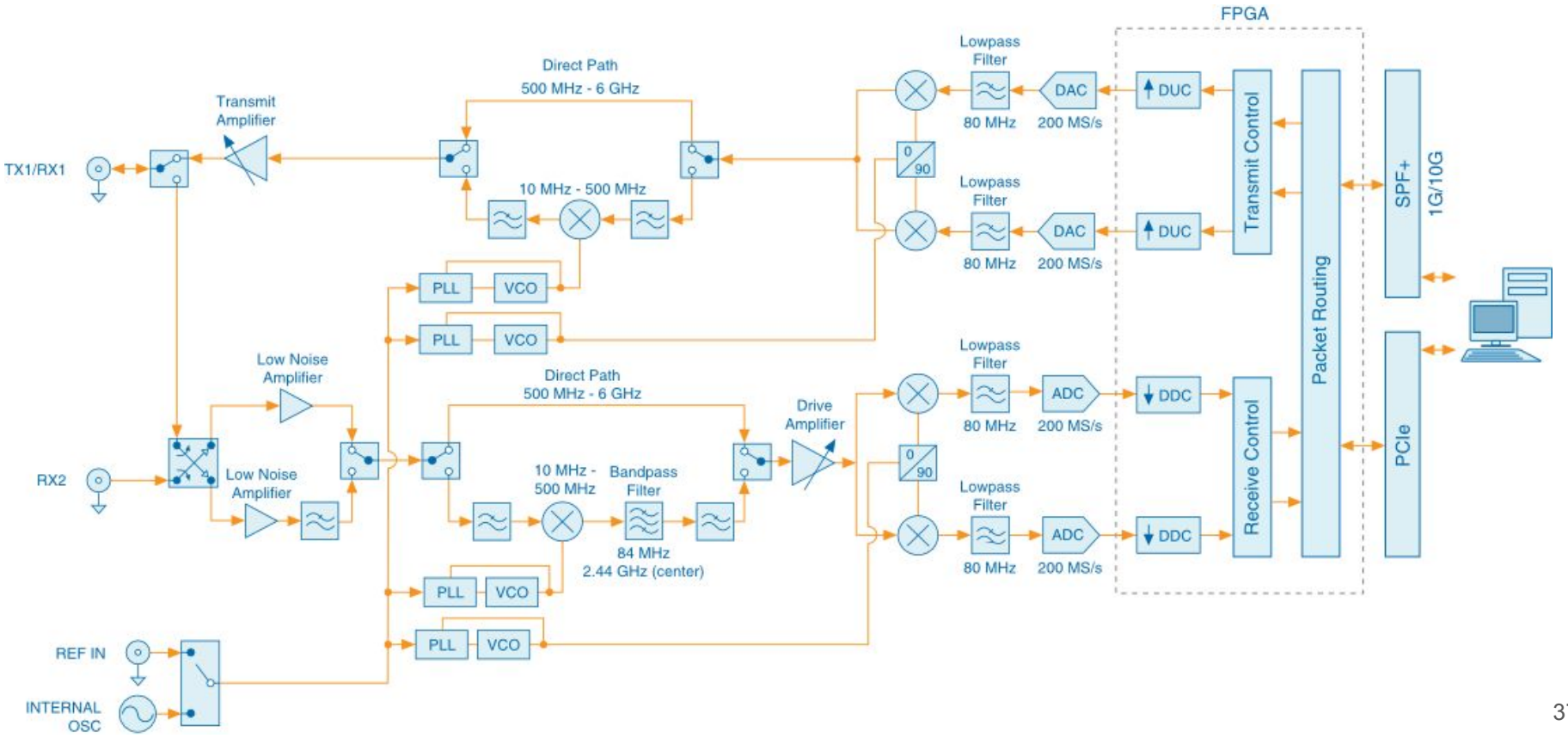
FRONT

REAR

USRP X300 / X310



USRP X300 / X310



USRP N300 / N310

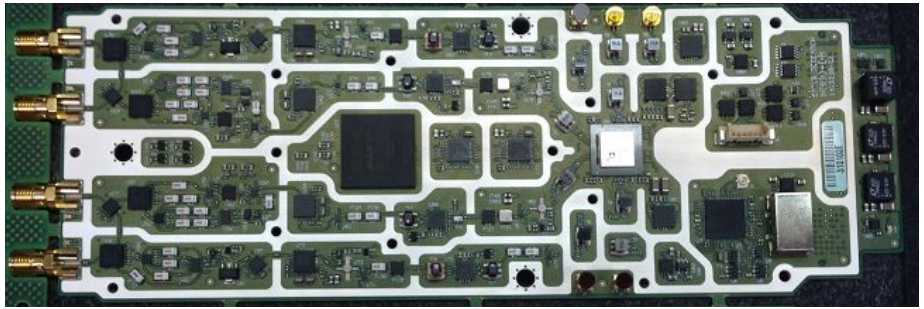
- Two Modes: Stand-Alone (embedded) and Host-Based (network streaming)
- Based on the Analog Devices AD9371
- Frequency Range 10 MHz to 6 GHz
- 100 MHz bandwidth per channel
- Channels: 4x4 (N310) or 2x2 (N300)
- 16-bit ADC and 14-bit DAC
- Master Clock Rates (MCR) of 122.88 MHz, 125 MHz, 153.6 MHz
- Xilinx Zynq 7100 (N310) or Zynq 7035 (N300) FPGA
- Dual 10 Gbps Ethernet port streaming support
- Rack-mountable, half wide, 1U
- Remote management support (remote firmware updates, remote OS updates, remote reboot, remote factory reset, remote diagnostics and system health)

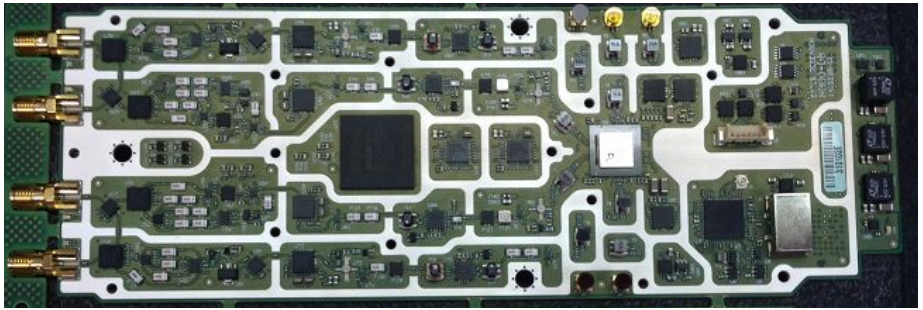
USRP N300 / N310 Photo

FRONT

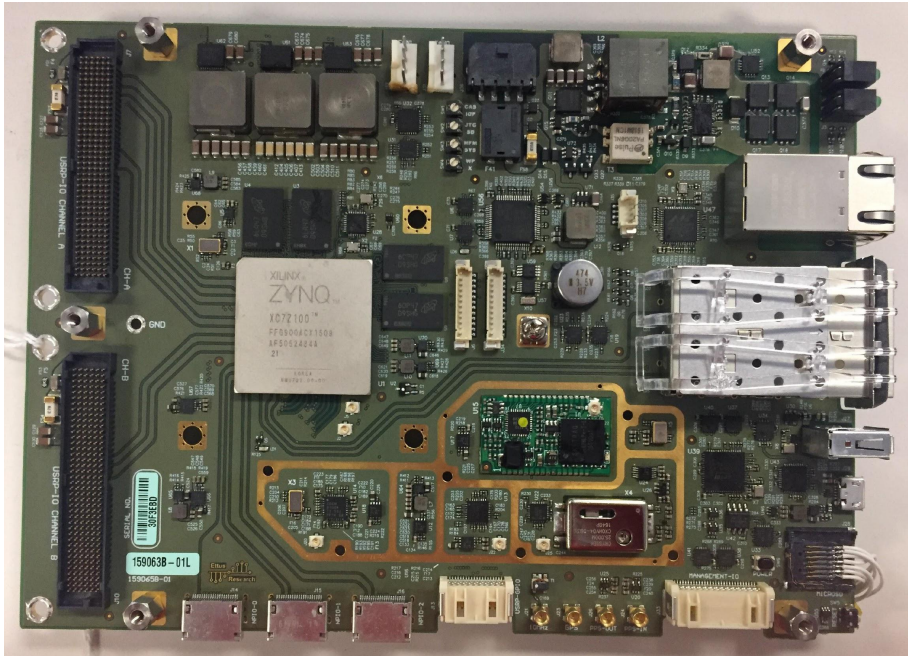
REAR

USRP N300 / N310 Photo



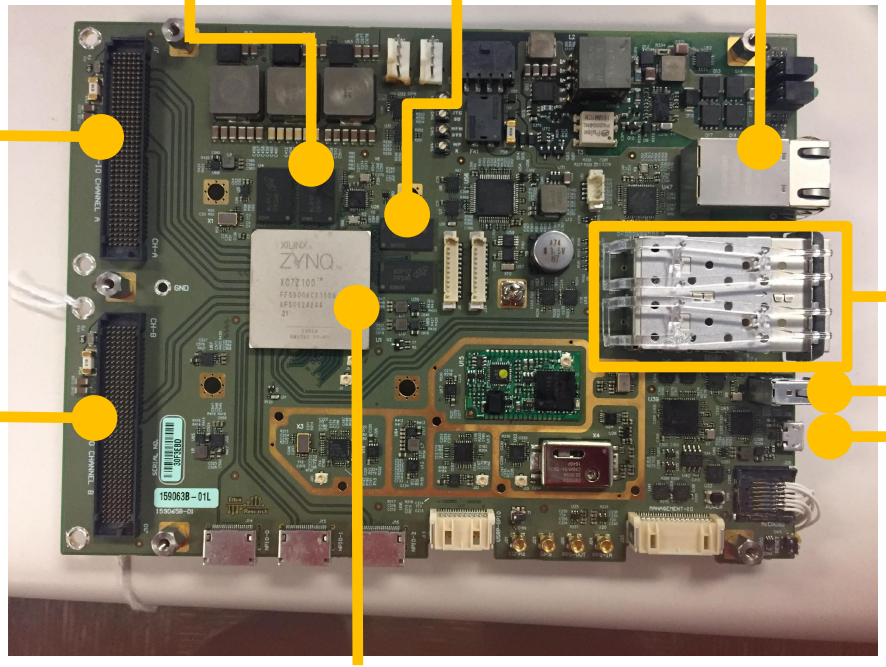


TWO DAUGHTERBOARDS



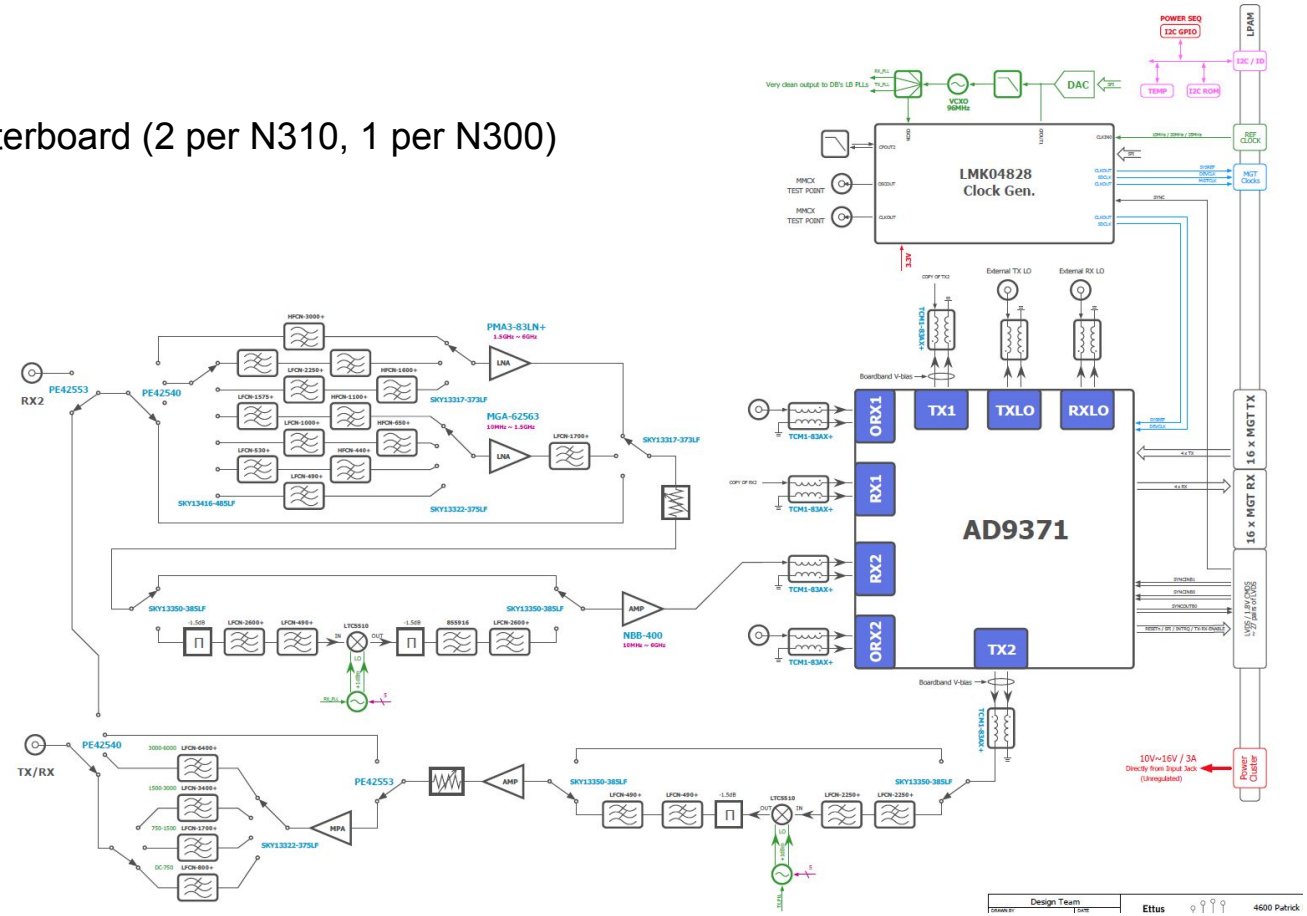
MOTHERBOARD

USRP N300 / N310



USRP N300 / N310

Daughterboard (2 per N310, 1 per N300)



USRP FPGA Resources

Ettus Research™
A National Instruments Company

	Zynq 7020 (E310)	Zynq 7035 (N300)	Zynq 7045 (E320)	Zynq 7100 (N310)	Kintex 7 410T (X310)
Logic Cells	85K	275K	350K	444K	406K
BRAM (MB)	4.9	17.6	19.1	26.5	28.6
DSP Slices	220	900	900	2020	1540
Flip-Flops	106K	343K	437K	554K	508K
LUTs	53K	171K	218K	277K	254K
GMACS	276	1334	1334	2622	2289

USRP N320 / N321

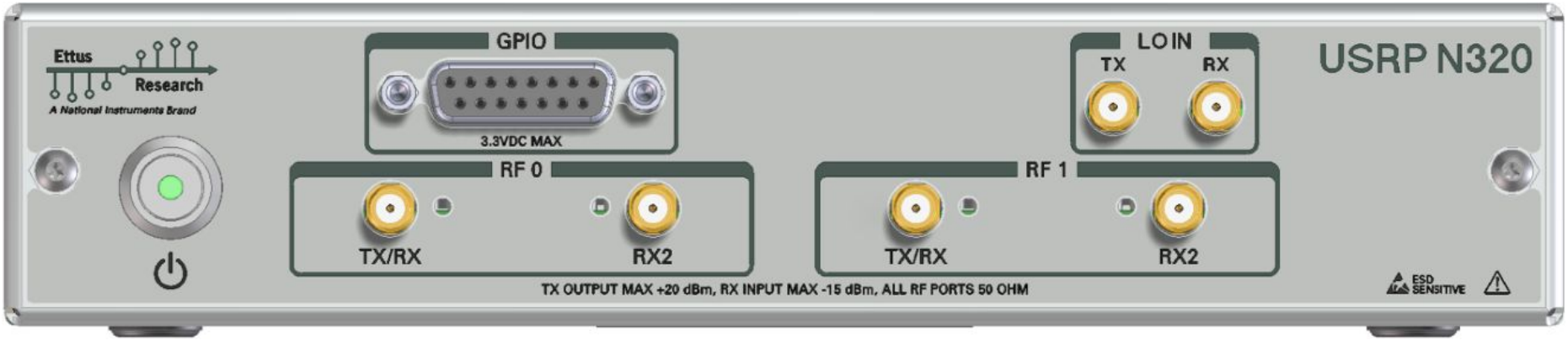
- Two Modes: Stand-alone (embedded) or host-based (network streaming) operation
- Discrete RF front-end (*not* based on AD9371) and 2x2 Channels
- Expanded frequency range from 3 MHz to 6 GHz *(covers the full HF band)*
- Up to 200 MHz of instantaneous bandwidth per channel
- Improved LO sharing, making it easier to build large phase coherent MIMO systems
- N320 and N321 have ability to import Tx and Rx LOs
- N321 has ability to export Tx and Rx LOs, allowing it to act as a master LO source
- Xilinx Zynq 7100 SoC with dual-core ARM Cortex A9 866 MHz CPU
- Two SFP+ ports and one QSFP+ port for full-rate sample streaming
- RJ-45 port (1 Gbps Ethernet) for remote management capability
- Independent 10 MHz clock and 1 PPS time references
- Includes internal GPSDO and supports White Rabbit synchronization
- 200, 245.76, 250 MHz Master Clock Rates, 14-bit ADC, 16-bit DAC

USRP N320 Photo

USRP N321 Photo

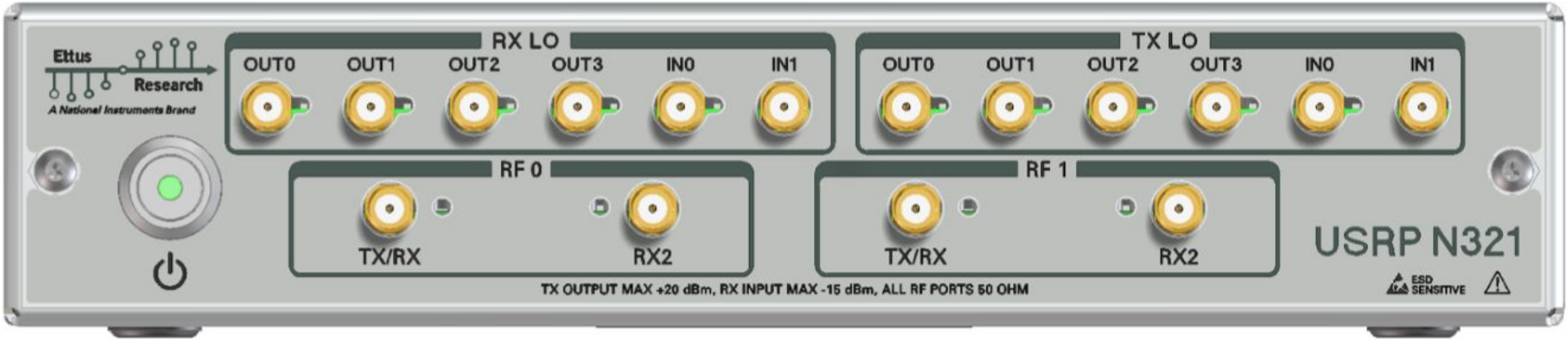
USRP N320 and N321 Photos

USRP N320 Front Panel



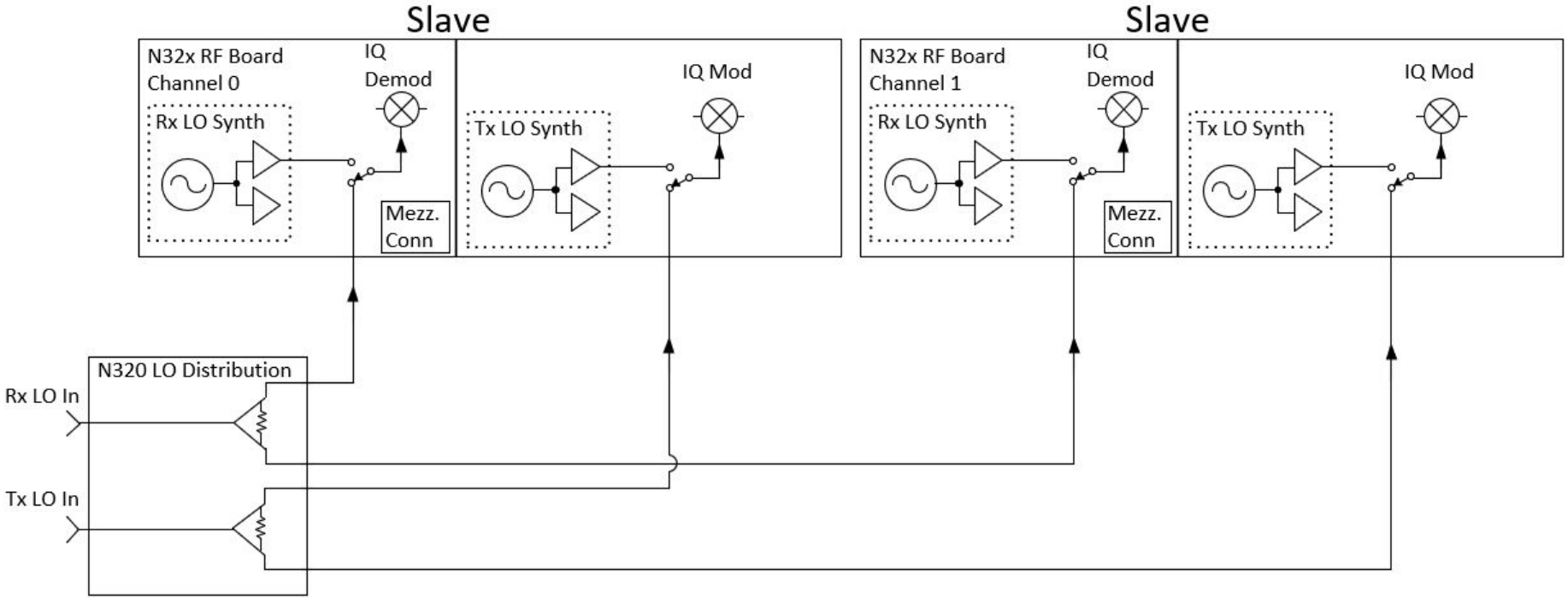
USRP N320 Rear Panel

USRP N321 Front Panel

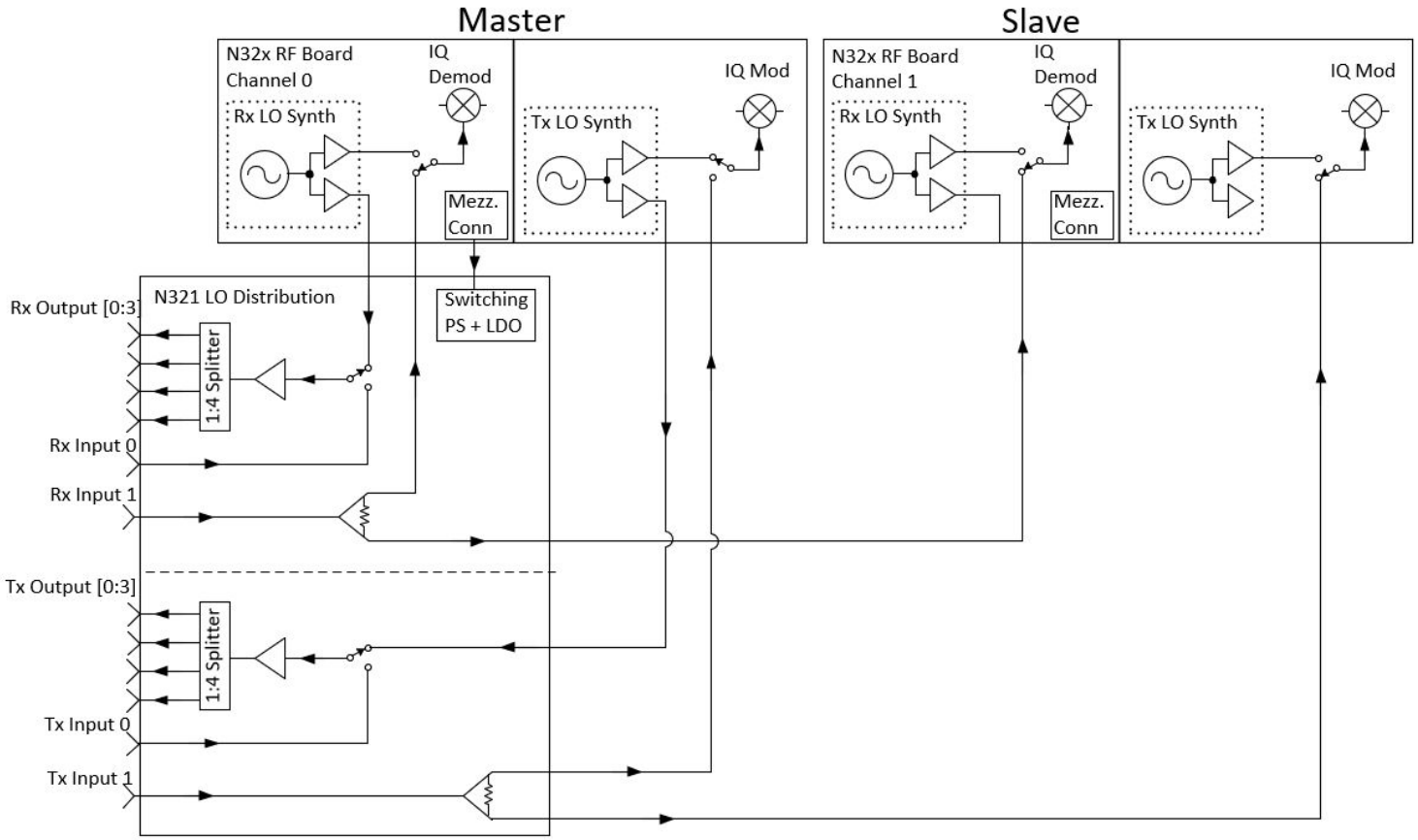


USRP N321 Rear Panel

USRP N320 LO Distribution Diagram



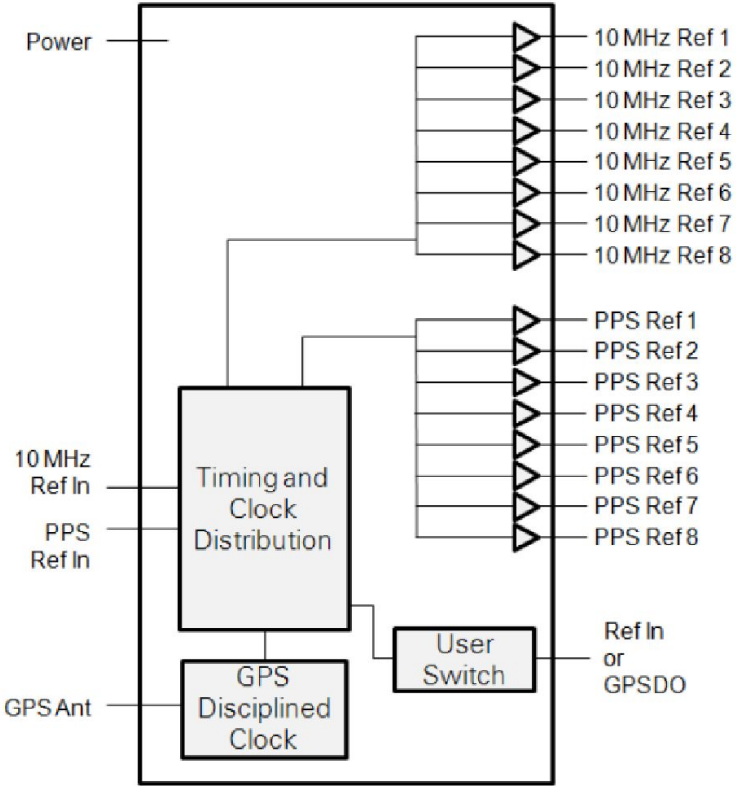
USRP N321 LO Distribution Diagram



OctoClock(-G) CDA-2990

- 8-Way Time and Frequency Distribution (1 PPS and 10 MHz)
- Convenient Solution for Multi-Channel Synchronization
- 19" Rackmount (1U)
- OctoClock distributes 10 MHz and 1 PPS signals from an external source
    - External 10 MHz/1 PPS Source Required
- OctoClock-G generates and distributes 10 MHz and 1 PPS signals
    - Contains internal GPSDO

OctoClock(-G) CDA-2990

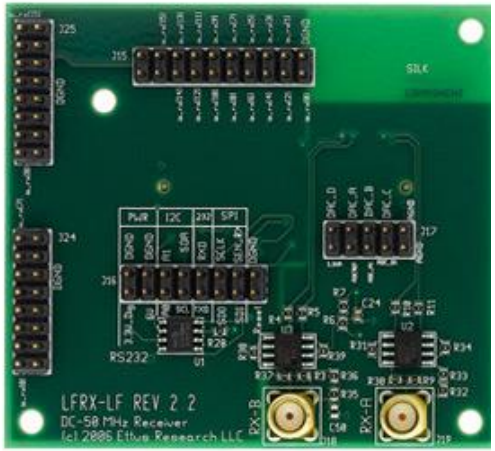


White Rabbit

- White Rabbit was developed at CERN
- Ethernet-based network for accurate time transfer and time distribution
- Aimed at geographically-distributed systems and GPS-denied environments
- Based on IEEE 1588 / PTP and Synchronous Ethernet (SyncE)
- Provides sub-nanosecond skew, and sub-picosecond jitter
- Supports connections of up to 10 km
- Scalable beyond 1000 nodes
- Open-standard and open-source implementations available
- Supported only on the USRP N300, N310, N320, N321
- **`https://ohwr.org/projects/white-rabbit`**

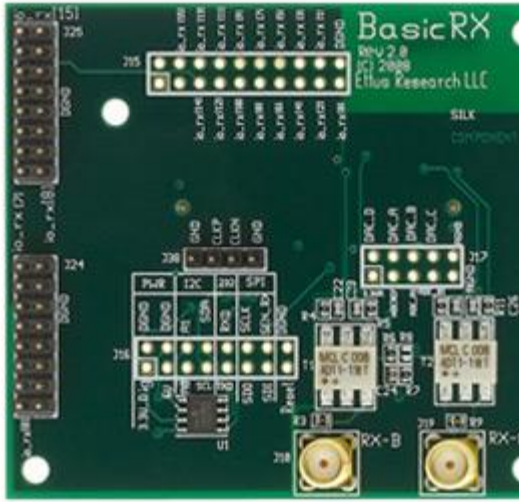
LFRX / LFTX Daughterboard

- Frequency Range: DC - 30 MHz
- Real or Complex Sampling



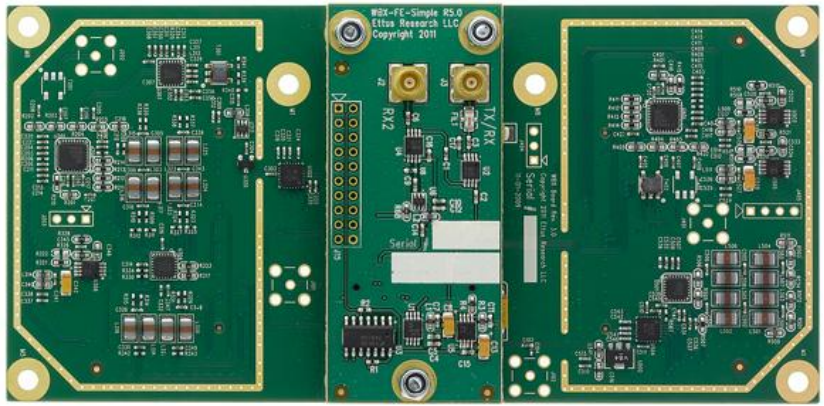
BasicRX / BasicTX Daughterboard

- Frequency Range: 1 MHz - 250 MHz
- Real or Complex Sampling
- Direct ADC / DAC inputs



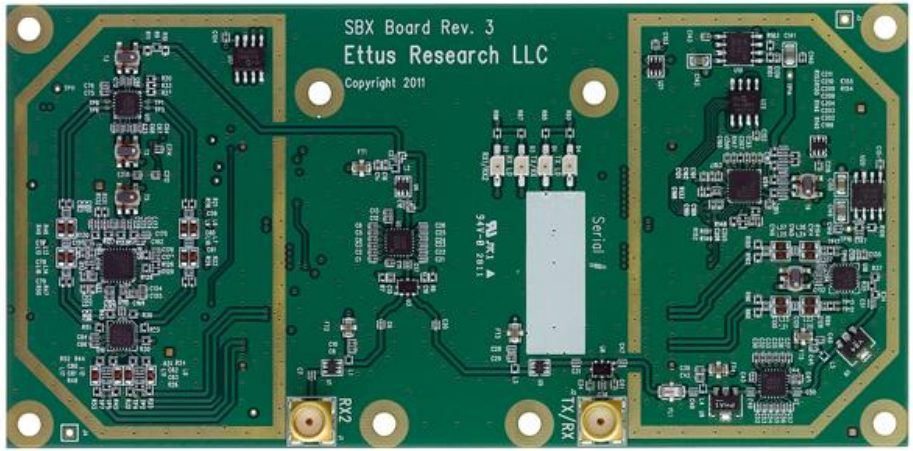
WBX Daughterboard

- Frequency Range: 50 MHz - 2.2 GHz
- Versions: 40 MHz / 120 MHz
- Full Duplex
- Phase sync with 180° ambiguity



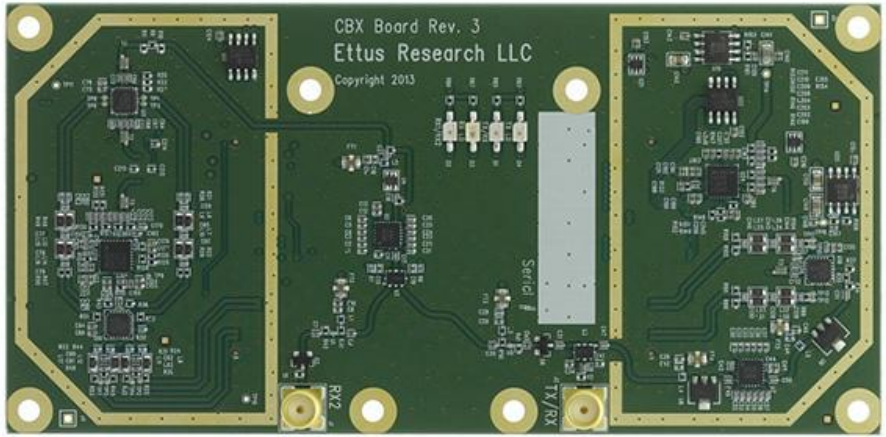
SBX Daughterboard

- Frequency Range: 400 MHz - 4.4GHz
- Versions: 40 MHz / 120 MHz
- Full Duplex
- Phase synchronization



CBX Daughterboard

- Frequency Range: 1.2 GHz - 6 GHz
- Versions: 40 MHz / 120 MHz
- Full Duplex



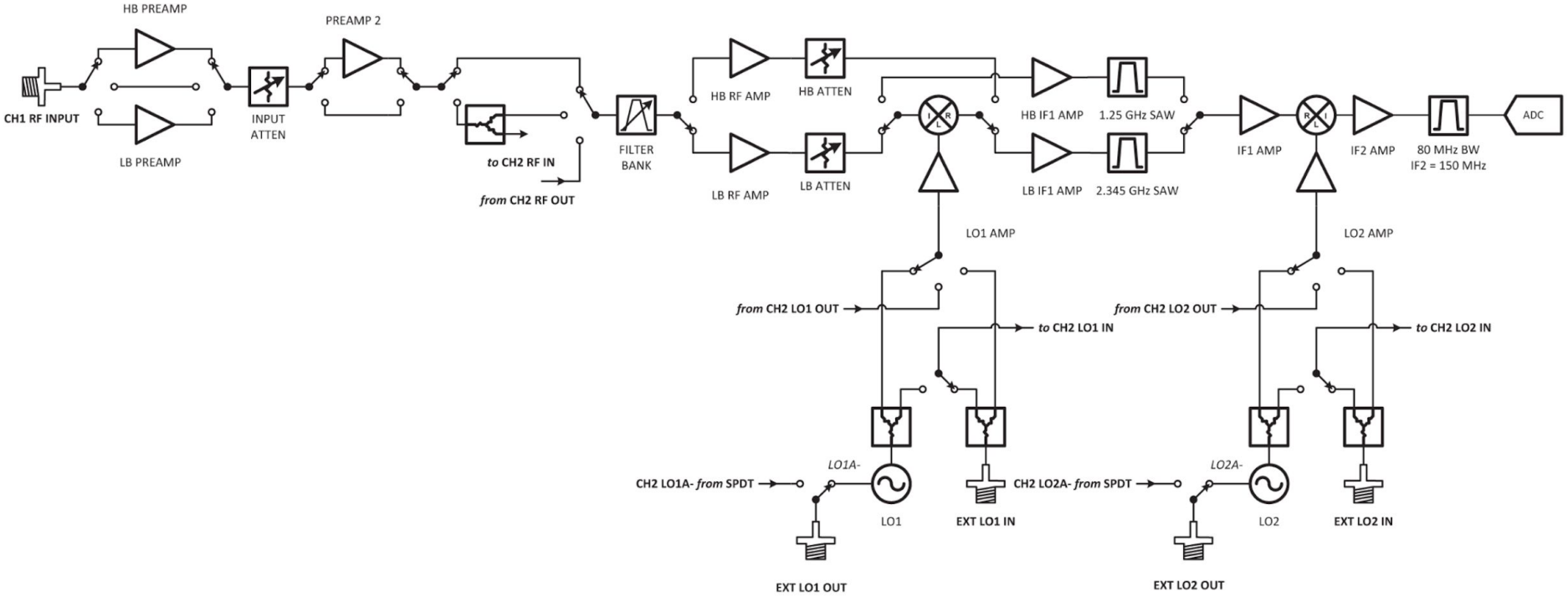
UBX Daughterboard

- Frequency Range: 10 MHz - 6 GHz
- Versions: 40 MHz / 160 MHz
- RF shielding
- Full duplex operation
- Independent TX and RX frequencies
- Synthesizer synchronization for applications requiring coherent or phase-aligned operation supported on USRP X Series motherboards only

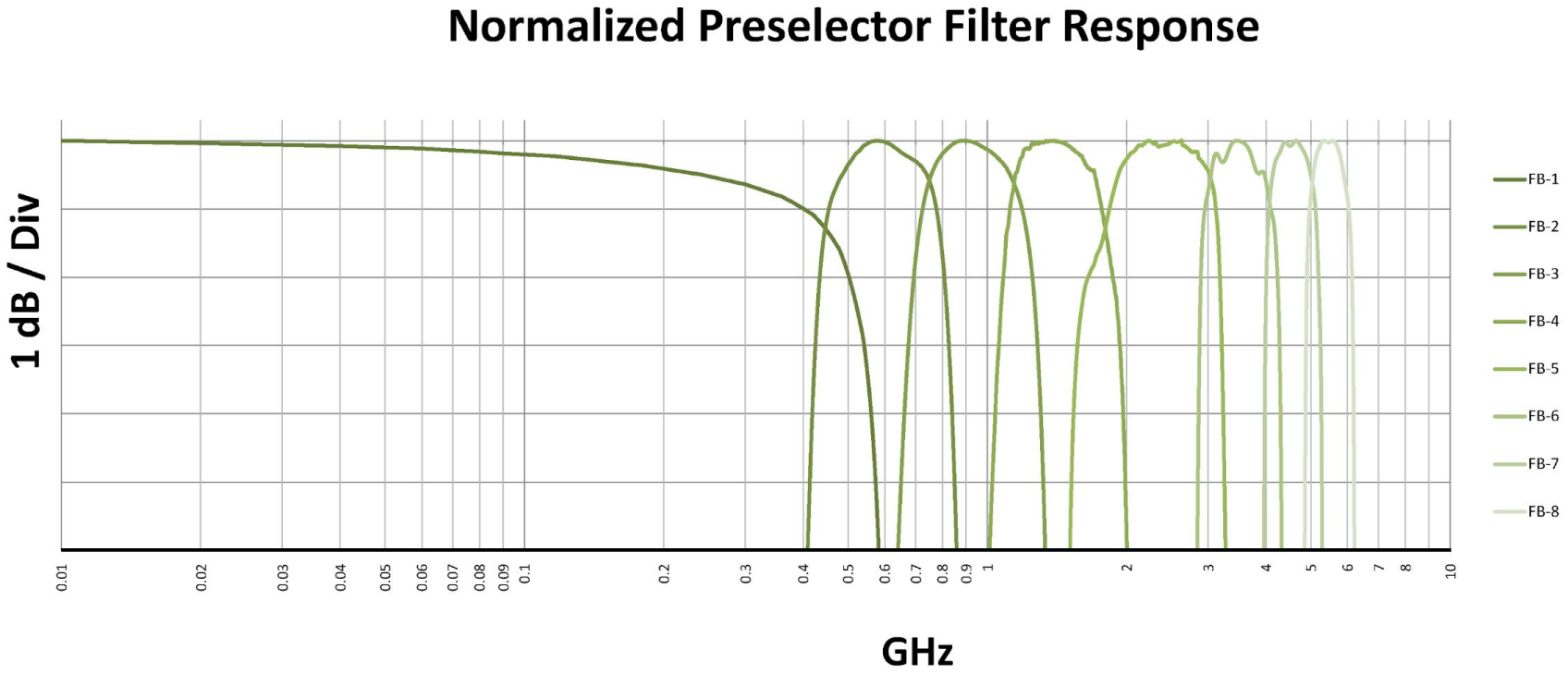
TwinRX Daughterboard

- Frequency range: 10 MHz - 6 GHz
- Bandwidth: 80 MHz per channel
- Channels: Two-stage superheterodyne
- 2 RX, Independent tuning
- LO sharing
- USRP compatibility: X Series
- RF shielding
- Coherent and phased aligned operation
- Spectrum Monitoring
- Direction Finding

TwinRX Daughterboard



TwinRX Daughterboard



TwinRX Daughterboard

Ettus Research™
A National Instruments Company

Frequency	Noise Figure[3] (dB)
10 MHz – 3 GHz	< 5
3 GHz – 5 GHz	< 4
5 GHz – 6 GHz	< 8

Frequency	Image Rejection[3] (dBc)
500 MHz – 6 GHz	-70

Phase Noise (dBc/Hz)			
Frequency Offset	0.9 GHz	2.4 GHz	5.8 GHz
10 kHz	-88	-86	-82
100 kHz	-105	-107	-103
1 MHz	-124	-127	-127

Third Order Intercept (dBm)			
Frequency	Full Scale = - 45 dBm	Full Scale = - 30 dBm	Full Scale = - 20 dBm
10 MHz - 1.8 GHz	-8	-2	16
1.8 GHz - 3 GHz	-10	-1	14
3 GHz - 6 GHz	-13	-1	12

Frequency	Non-Input Related (Residual) Spurs[2, 3] (dBm)
10 MHz – 3 GHz	< -95
3.2 GHz	-92
4.8 GHz	-98
5.4 GHz	-98

Sampling Rates

- Integer decimation of the Master Clock Rate (MCR)
  - Even decimation rate preferred
  - Odd decimation rate allowed but with CIC filter roll-off attenuation
  - For N200, N210:
    - MCR is 100 MHz
    - Decimation rates from 1 to 512
    - Sample rates: 100, 50, 25, 16.67, 12.5, 10, 8.33 MHz, … 195.31 KHz
  - For X300, X310:
    - MCR is 200 MHz
    - Decimation rates from 1 to 1024
    - Sample rates: 200, 100, 50, 33.33, 25, 20, 16.67, 14.29, 12.5 MHz, … 195.31 KHz
    - MCR 184.32 MHz also supported

Sampling Rates (continued)

- For N300, N310:
  - MCR are 122.88, 125, 153.6 MHz
  - Decimation rates from 1 to 1024
- For N320, N321:
  - MCR are 200, 245.76, 250 MHz
  - Decimation rates from 1 to 1024
- For B200, B210, B200mini, B205mini, E310, E312, E320:
  - 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

Host Interface Data Rates

- USB 2.0 uses 480 Mbits/sec (60 MB/sec) signalling rate
    - ~35 MB/sec practical, or ~8 Msps
- USB 3.0 uses 5 Gbits/sec (625 MB/sec) signalling rate
    - ~350 MB/sec practical, or ~80 Msps
- 1 GbE is 1000 Mbits/sec (125 MB/sec) theoretical
    - ~25 Msps (sc16), ~50 Msps (sc8), practical
- 10 GbE is 10000 Mbits/sec (1250 MB/sec) theoretical
    - ~250 Msps (sc16), ~500 Msps (sc8), practical

Over-the-wire (OTW) Formats

- Used to lessen the data widths of samples over the host computer interface,

  in order to enable higher sampling rates
- Trade-off is loss of resolution of the sample data
- One complex sample is nominally 16-bit I, 16-bit Q
- **`sc8`** is 8-bit I, 8-bit Q
- **`sc12`** is 12-bit I, 12-bit Q
- **`sc16`** is 16-bit I, 16-bit Q
- Specify with the **`--wirefmt`**, **`--tx_otw`**, **`--rx_otw`** options
- Device maximum sampling rate may exceed maximum interface data rate
  - Max sampling limited not by device but by interface
  - Also limited by CPU and disk I/O

USRP Calibration

- The ER-USRP are not calibrated devices
- Any mapping between physical input/output power levels and USRP gain value, signal levels, frequency setting must be measured empirically, and adjustments made in application software
- This mapping can change based on temperature, time, and sometimes gain setting
- This mapping can also vary unit-to-unit
- The NI-USRP have correction factors stored in the EEPROM
    - Calculated at the factory at the time of manufacture
    - Used by LabVIEW, and can be backed up
    - Not used by UHD, but UHD has software calibration utilities for DC offset and I/Q balance

NI-USRP & ER-USRP Branding

- Identical Hardware
  - Almost: No GPSDO on NI USRP B200, B210
- ER USRP used with open-source (C++ and GNU Radio) and Matlab
- NI USRP used with LabView
- NI USRP can be converted to ER USRP, and vice-versa
  - Application Note on KB explains process step-by-step
- NI only provides LabVIEW support for NI branded USRP
- NI-294x and NI-295x (X310) USRPs contain factory-installed correction values within EEPROM which will be lost if converting from NI-USRP to ER-USRP

NI-USRP & ER-USRP Product Mapping

Bus Series	
NI-USRP	ER-USRP
NI-2900	B200
NI-2901	B210

NI-USRP & ER-USRP Product Mapping

Network Series	
NI-USRP	ER-USRP
NI-2920	N210 + WBX
NI-2921	N210 + XCVR2450
NI-2922	N210 + SBX
NI-2930	N210 + WBX + GPSDO
NI-2932	N210 + SBX + GPSDO

NI-USRP & ER-USRP Product Mapping

X Series	
NI-USRP	ER-USRP
NI-2940R	X310 + WBX (x2)
NI-2942R	X310 + SBX (x2)
NI-2943R	X310 + CBX (x2)
NI-2944R	X310 + UBX (x2)
NI-2945R	X310 + TwinRX (x2)

NI-USRP & ER-USRP Product Mapping

X Series	
NI-USRP	ER-USRP
NI-2950R	X310 + WBX (x2) + GPSDO
NI-2952R	X310 + SBX (x2) + GPSDO
NI-2953R	X310 + CBX (x2) + GPSDO
NI-2954R	X310 + UBX (x2) + GPSDO
NI-2955R	X310 + TwinRX (x2) + GPSDO

USRP Hardware Driver (UHD)

- Provides a single, common interface to 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

<table>
<tr><td colspan="4">Application</td></tr>
<tr><td>LabVIEW</td><td>C++</td><td>GNU Radio
Python / GRC / C++</td><td>Matlab</td></tr>
<tr><td colspan="4">UHD Driver</td></tr>
<tr><td>Windows</td><td>OSX</td><td>Linux</td><td>Embedded Linux</td></tr>
<tr><td colspan="4">Hardware
Motherboard (FPGA)
Daughterboard
Antenna</td></tr>
</table>

UHD Licensing

- UHD and RFNoC are free software and open-source projects
  - UHD issued under GPLv3 license
  - RFNoC issued under LGPL license
  - All source code for host driver, MPM, FPGA, microcontroller firmware is available on GitHub
- Available:
  - Partial BoM (key components)
  - Schematics (PDF file)
  - 2D mechanical drawings (PDF file)
  - 3D CAD models (STP files)
  - Test validation data
- Not available:
  - Full BoM
  - Gerber files for PCBs

UHD Licensing (continued)

- ER also offers an alternative license for UHD and RFNoC
  - Enable non-FOSS designs and proprietary product distribution
  - Available only from Ettus Research
- Does not apply to GNU Radio
  - Only issued under GPLv3
  - Effort underway to issue GNU Radio under the LPGL
  - ER does not own the copyright

UHD Versioning

- Starting in UHD 3.10, the version numbering changed to quadruplets (`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
- When there is a significant change to the API, such as for adding N310 support, then the API is no longer the same as it was in the previous version, necessitating that the API component of the version number gets bumped up by one.
- The ABI pertains to how external applications communicate with (link to) the UHD library. When the ABI changes, the number gets bumped by one.
- The patch number is incremented when patches are made, typically for bug fixes.

Radio Transport Protocols

Radio transport protocols are used to exchange samples (or other items) between host and devices. If one were to sniff Ethernet traffic between a USRP and a PC, the packets would conform to a radio transport protocol.

For USRP devices, two radio transport protocols are relevant: **VRT** (the VITA Radio Transport protocol) and **CHDR** (compressed header, an Ettus-specific protocol).

Generation-3 devices and the B200 use **CHDR**, the rest use **VRT**.

VRT is an open protocol defined by the VITA-49 standard. It was designed for interoperability, and to allow different device types to work with different software stacks.

VRT is a very verbose standard, and only a subset is implemented in UHD/USRPs. The full standard is available from the VITA website: http://www.vita.com

Radio Transport Protocols

For the third generation of Ettus devices, a new type transport protocol was designed. It reduces the complexity of the original standard and uses a fixed-length 64-Bit header for everything except the timestamp. Because it is "compressed" into a 64-bit header, it was dubbed **CHDR** (pronounced like the cheese "cheddar").

By compressing all information into a 64-bit line, the header can efficiently be parsed in newer FPGAs, where the common streaming protocol is 64-Bit AXI. The first line in a packet already provides all necessary information to proceed.

Some CHDR-specific functions can be found in: **`uhd::transport::vrt::chdr`**.

Typical CHDR Packet form:

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

If there is no timestamp present, the data starts at address 8, otherwise, it starts at 16.

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)

Radio Transport Protocols

Tools
For CHDR, we provide a Wireshark dissector under `tools/chdr_dissector`. It can be used for Ethernet links as well as USB (e.g., for the B210).

Code
Relevant code sections for the radio transport layer are:
`uhd::transport::vrt`

Namespace for radio transport protocol related functions and definitions:
`uhd::transport::vrt::chdr`

Sub-namespace specifically for CHDR:
`uhd::sid_t`

Datatype to represent SIDs

Git and GitHub

- Git is a free, cross-platform, open-source distributed source code management system
- GitHub is a web-based hosting service for Git repositories
    - It’s not the only one, GitLab is also popular
- Both command-line-based and GUI-based clients for Windows, OS X, Linux
- Successor to CVS and Subversion (SVN), which are centralized systems
- Git was written by Linus Torvalds in 2005,
  and is now very widely used in the open-source community

Git and GitHub

Ettus Research™
A National Instruments Company

- Free books and resources:
  - http://git-scm.com/
  - https://progit.org/
  - https://try.github.io/
  - https://www.atlassian.com/git/tutorials
  - http://rypress.com/tutorials/git/index
  - https://www.git-tower.com/learn/git/ebook/en/command-line/introduction
  - http://www-cs-students.stanford.edu/%7Eblynn/gitmagic/

Git and GitHub - Git-Fu

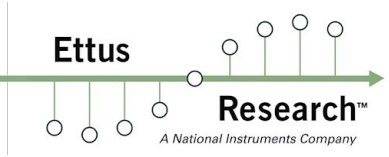
Install Git on your system, if it's not already installed:

- `sudo apt-get install git`

Most common Git commands when working with UHD and GNU Radio:

- `git clone <repository_url>`
- `git tag -l`
- `git checkout <tag|commit>`
- `git log`
- `git status`

Git and GitHub - Git-Fu



View your Git username and email configuration:

```
$ git config user.name
$ git config user.email
```

Set your Git username and email configuration:

```
$ git config user.name "username"
$ git config user.email "email@domain.com"
```

Validate your SSH key against Github.com:

```
$ ssh -T git@github.com
```

Git and GitHub - Git-Fu

Clone using SSH instead of HTTPS:

```
$ git clone git@github.com:EttusResearch/uhd.git
```

Clone and checkout a branch in a single command:

```
$ git clone -b release_003_009_005 git@github.com:EttusResearch/uhd.git
```

View commit history of a repo (from within a repo):

```
$ git log
```

See current status of your commits on your working repo:

```
$ git status
```

Git and GitHub - Git-Fu

Ettus Research™
A National Instruments Company

Create a local branch:

```
$ git checkout -b mynew/branch
```

Add files to be committed:

```
$ git add .
```

Commit files:

```
$ git commit -m “my commit message”
```

The UHD Repository on GitHub

`host/`
The source code for the host-side driver.

`firmware/`
The source code for all microcontrollers in USRP hardware.

`fpga-src/`
The source code for the UHD FPGA images. Note this is a git submodule, if you are cloning the repository and want to modify the FPGA code, you will need to run **'git clone --recursive'** to automatically populate this directory. Alternatively, you can run **'git submodule init'** followed by **'git submodule update'** to populate it after cloning the repository without **'--recursive'**. Note that this subdirectory is very large, and not necessary for building applications that link against UHD.

`mpm/`
The source code for the Module Peripheral Manager (MPM) for embedded USRP devices.

`images/`
This contains the package builder for FPGA and firmware images. We provide other tools to download image packages, the scripts in here are mainly relevant for UHD maintainers and -developers.

`tools/`
Additional tools and utilities, mainly for debugging purposes.

Git and GitHub - Creating Pull Requests

- Internal development and Pull Requests are always submitted against the *dev repositories
- PRs must be reviewed and pass BuildBot before merge
- Only the tech leads may merge into their specific repositories.

Git and GitHub - Creating Pull Requests

Start by cloning your target repository to your local machine:

```
$ git clone git@github.com:EttusResearch/uhddev.git

$ cd uhddev
```

Configure your username / email if not set globally:

```
$ git config user.name "your_username"
$ git config user.email "your_email@domain.com"
```

Create a new working branch. Standard naming is to use your username/new_branch_name:

```
$ git checkout -b myusername/branchname
```

**** Make your modifications to the files at this point ****

Git and GitHub - Creating Pull Requests

Ettus Research™
A National Instruments Company

After completing your modifications to the files, add them.

```
$ git add .
```

Next, commit the changes.

```
$ git commit -m "short message on changes made"
```

Next, push your changes to a remote branch:

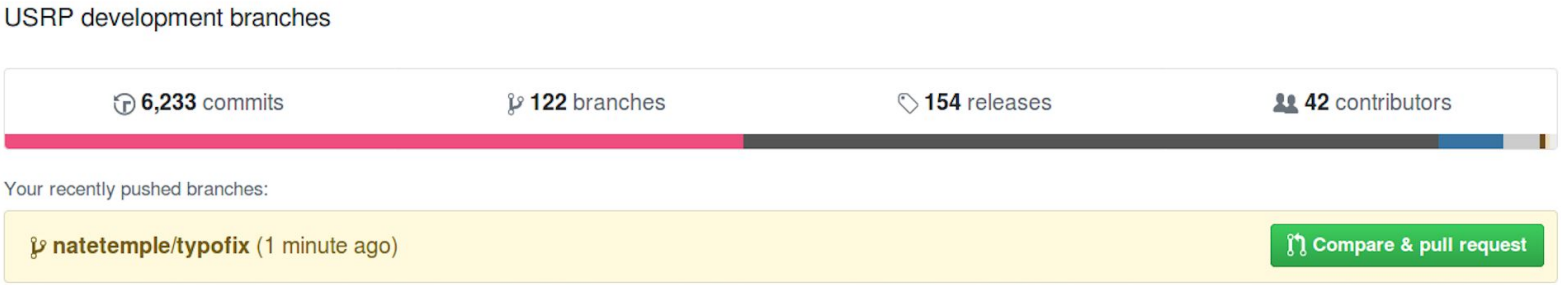
```
$ git push -u origin myusername/branchname
```

Your local branch has now been pushed to the remote repository.

Git and GitHub - Creating Pull Requests

Next, you will need to login to Github.com to submit a Pull Request.

Within the Repository that you have pushed, a popup will appear to create a Pull Request based on your recent push.



Click the “Compare & pull request” button.

Git and GitHub - Creating Pull Requests

Ettus Research™
A National Instruments Company

You will then be taken to the “Open a Pull Request” submission form. Within the comment space of the Pull Request form, provide any details of testing that you have performed, and if it is for “Review Only” or is “Ready to merge”.

After submitting the Pull Request, a BuildBot check will be ran.

The PR will then be reviewed either by the Tech Lead, or by the Assignees. Upon successful view, the PR will then be merged into the public repository.

Installing UHD, GNU Radio, etc

In this workshop, we will always clone into the **`/home/demo/workarea`** folder.

Create a **`/home/demo/workarea`** folder:

```
$ mkdir -p /home/demo/workarea
```

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. `cd ~/workarea`
3. `git clone git://github.com/EttusResearch/uhd.git`
4. `cd uhd/`
5. `git checkout release_003_009_005`
6. `cd host/`
7. `mkdir build && cd build`
8. `cmake ../`
9. `make -j4`
10. `make test`
11. `sudo make install`
12. `sudo ldconfig`

* Skip if using LiveSDR Environment

Reference: ~/ettus_workshop/instructions/install.html

Installing UHD from Binary Package

- From binary packages on Ubuntu Launchpad PPA
- Only for Ubuntu 14.04 LTS, 15.10, 16.04 LTS
- Only for UHD version 3.9.2 or newer
- See **`https://launchpad.net/~ettusresearch/+archive/ubuntu/uhd`**

```
sudo apt-add-repository ppa:ettusresearch/uhd

sudo apt-get update

sudo apt-get install uhd-host libuhd003 libuhd-dev
```

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.

Post-Installation Steps

Thread priority scheduling

Add “**usrp**” group and add your user to “**usrp**”:

```
$ groupadd usrp

$ usermod -aG usrp $USER
```

Append the line below to the file: **/etc/security/limits.conf**

```
@usrp   -       rtprio  99
```

Logout and log back in.

UHD Utilities - `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 Utilities - `uhd_images_downloader`

```
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 Utilities - `uhd_find_devices`

Uses broadcast packets for 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 Utilities - `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
|   |   |   |   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

--args , most UHD applications and examples make use of this parameter to focus on, or find specific devices.

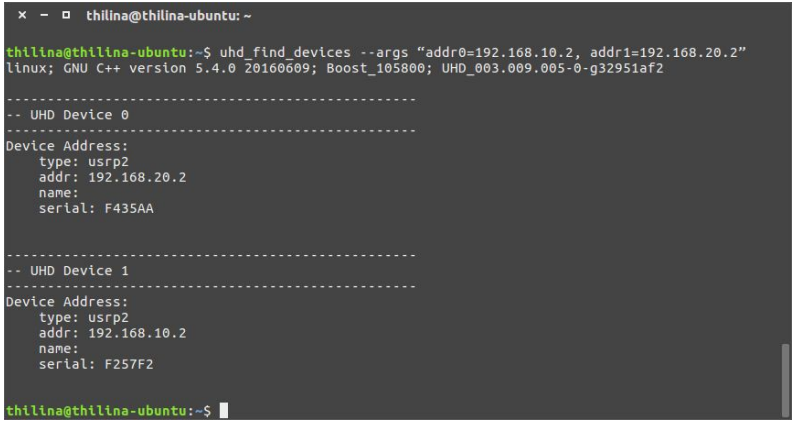
Common argument keys: **serial, addr, resource, name, type, vid/pid**.

`$ uhd_find_devices --args "addr=192.168.10.2"` ***(for N2xx / X3xx)***

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

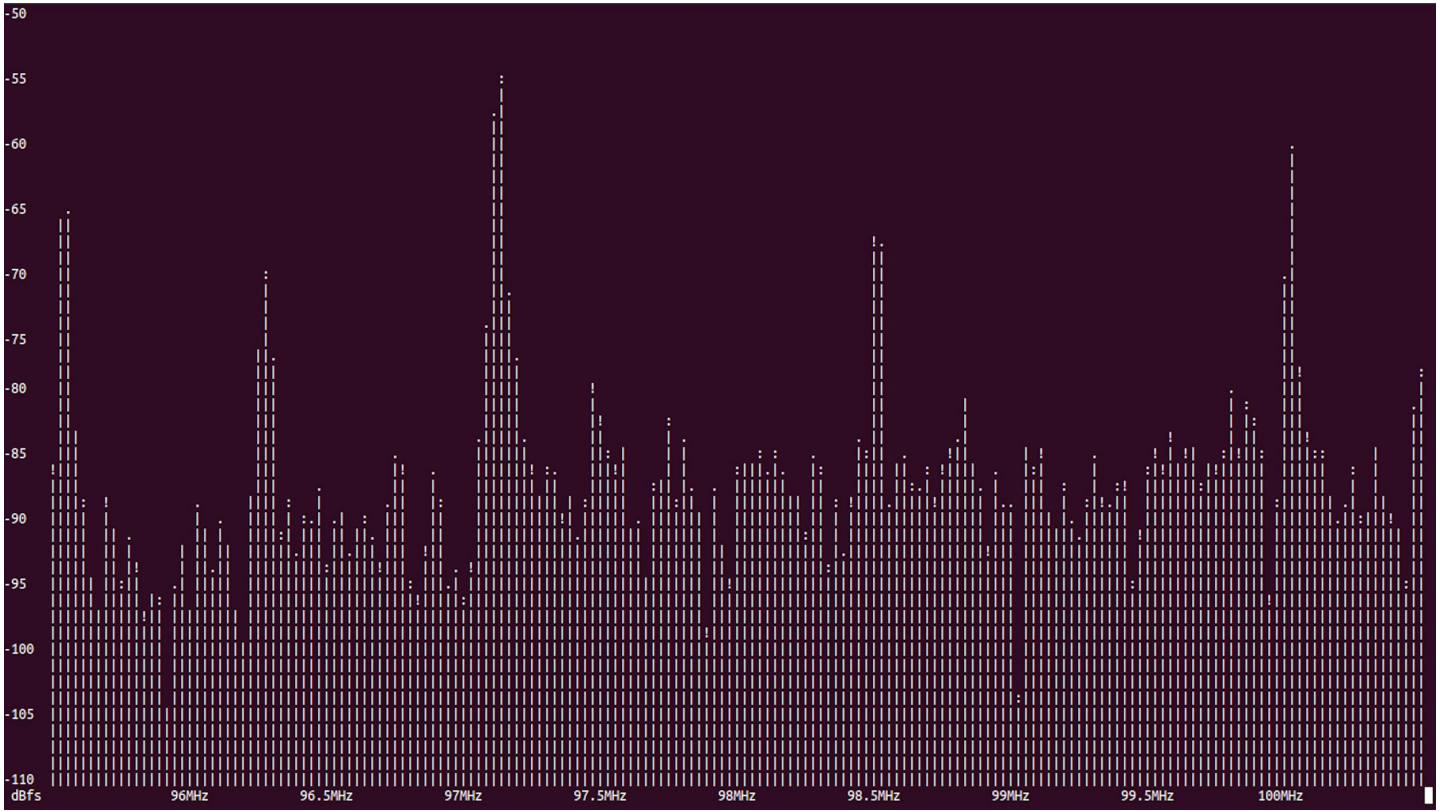
(Note multiple arguments are comma-delimited)

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 --args "addr=192.168.10.2" --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 --args=”addr=192.168.10.2” --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 --args="addr=192.168.10.2" --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 Utilities

- Default installation location is **`/usr/local/lib/uhd/utils`**
- **`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`**

UHD Managing Multiple Installations

It is possible to have multiple concurrent UHD / GR installations. This can be useful to quickly switch between versions and/or test different versions of UHD/GR.

Example: Create local (non-default) installation of UHD 3.9.6

First, create a directory for the installation location, and sources:

```
mkdir -p ~/installs/uhd_396/src
cd ~/installs/uhd_396/src
```

UHD Managing Multiple Installations

Next, clone UHD into the **`src`** directory and checkout the 3.9.6 release:

```
git clone git://github.com/EttusResearch/uhd.git
cd uhd
git checkout release_003_009_006
cd host
mkdir build
```

Next, we will pass the flag **`CMAKE_INSTALL_PREFIX`** to the **`cmake`** command. This will define which prefix UHD will be installed at.

```
cmake -DCMAKE_INSTALL_PREFIX=/home/demo/installs/uhd_396 ../
```

Next, we will compile and install UHD. Note, when performing a “local” installation, the **`“make install”`** command does not require **`“sudo”`**, because it is being installed to your local/home directory, and not the default **`/usr/local`**.

```
make -j4
make install
```

UHD Managing Multiple Installations

Next, we need to create an Environment Setup File. We will call this file **"setup_env.sh"**.

There are several important Linux Systems Variables that need to be configured to switch between UHD installations: **PATH, LD_LIBRARY_PATH, PYTHONPATH, PKG_CONFIG_PATH**

```
cd ~/installs/uhd_396
touch setup_env.sh
nano setup_env.sh
```

Add the following lines to **setup_env.sh**:

```
export BASE_PATH=/home/demo/installs/uhd_396
export PATH=$BASE_PATH/bin:$PATH

export LD_LIBRARY_PATH=$BASE_PATH/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$BASE_PATH/lib/python2.7/site-packages:$PYTHONPATH
export PYTHONPATH=$BASE_PATH/lib/python2.7/dist-packages:$PYTHONPATH
export PKG_CONFIG_PATH=$BASE_PATH/lib/pkgconfig:$PKG_CONFIG_PATH
```

UHD Managing Multiple Installations

If you're operating on a Fedora/CentOS/RHEL based OS which has a Library Suffix "64", you will need to modify the previously path commands to the paths below:

```
export LD_LIBRARY_PATH=$BASE_PATH/lib64:$LD_LIBRARY_PATH
export PYTHONPATH=$BASE_PATH/lib64/python2.7/site-packages:$PYTHONPATH
export PYTHONPATH=$BASE_PATH/lib64/python2.7/dist-packages:$PYTHONPATH
export PKG_CONFIG_PATH=$BASE_PATH/lib64/pkgconfig:$PKG_CONFIG_PATH
```

Note the addition of **"lib64".**

UHD Managing Multiple Installations

Before sourcing the "**setup_env.sh**" file, let's first check what UHD version is currently being called by the system. We will test this with two commands:

```
$ uhd_usrp_probe --version
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.005-0-g32951af2

003.009.005-0-g32951af2

$ which uhd_usrp_probe
/usr/local/bin/uhd_usrp_probe
```

UHD Managing Multiple Installations

Finally, we will now need to "**source**" the "**setup_env.sh**" file, which will execute and load the functions / variables into the current shell session.

```
$ source ~/installs/uhd_396/setup_env.sh
```

Now, verify that the local version of UHD is being referenced:

```
$ uhd_usrp_probe --version
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.006-0-g122d5f8e

003.009.006-0-g122d5f8e

$ which uhd_usrp_probe
/home/demo/installs/uhd_396/bin/uhd_usrp_probe
```

Note: The System Variables / Paths set by the "setup_env.sh" script will only be loaded into the current shell. If you close the terminal, or open a new terminal, you would need to rerun the "**source setup_env.sh**" command in order to use the locally installed version.

UHD Managing Multiple Installations

Next, run the "**`uhd_images_downloader`**" utility. This will download and install the FPGA image package to the local **`(~/installs/uhd_396/...)`** path.

```
$ uhd_images_downloader

Images destination:      /home/demo/installs/uhd_396/share/uhd/images
Downloading images from:
http://files.ettus.com/binaries/images/uhd-images_003.009.006-release.zip
Downloading images to:   /tmp/tmpOny8tE/uhd-images_003.009.006-release.zip
26269 kB / 26269 kB (100%)

Images successfully installed to: /home/demo/installs/uhd_396/share/uhd/images
```

You can now initialize the USRP with this local version of UHD by running:

```
$ uhd_usrp_probe
```

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.
- **`http://files.ettus.com/manual/page_usrp_x3x0_config.html`**

Using UHD from C++

Reference: ~/ettus_workshop/examples/usrp_basic/usrp_basic.cpp

```
#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 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 from C++

```
//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 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.
```

Debugging USRP with Wireshark

- **`sudo apt-get install wireshark`**
- In a terminal window, run as root: **`sudo wireshark`**
- Select appropriate network interface from list (usually **`eth0`**), and click **Start** icon
- **`uhd_find_devices`** uses broadcast packets to find all USRP devices on network
- Be careful, host firewall (iptables), switch, or router may block broadcast packets
- View status of your host firewall settings by running: **`sudo iptables -L`**
- X300 / X310 will emit gratuitous ARP packets showing IP address every 6 seconds
- N200 / N210 does not do this

Debugging USRP with Wireshark

Ettus Research™
A National Instruments Company

*eth1 [Wireshark 1.10.6 (v1.10.6 from master-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.10.3	192.168.10.255	ICMP	62	Information request id=0x0200, seq=438/46593, ttl=32
2	5.960641000	192.168.10.3	192.168.10.255	ICMP	62	Information request id=0x0200, seq=441/47361, ttl=32
3	8.430804000	192.168.10.253	192.168.10.255	BROWSER	272	Host Announcement NIPTON, Workstation, Server, Print Queue Server, Xenix Server, NT Workstatio
4	11.92133000(	192.168.10.3	192.168.10.255	ICMP	62	Information request id=0x0200, seq=444/48129, ttl=32
5	17.88199400(	192.168.10.3	192.168.10.255	ICMP	62	Information request id=0x0200, seq=447/48897, ttl=32
6	23.40970400(	National_0a:df:3e	Broadcast	ARP	62	Gratuitous ARP for 192.168.10.3 (Request)
7	23.84270700(	192.168.10.3	192.168.10.255	ICMP	62	Information request id=0x0200, seq=450/49665, ttl=32
8	29.80334900(	192.168.10.3	192.168.10.255	ICMP	62	Information request id=0x0200, seq=453/50433, ttl=32
9	35.76401300(	192.168.10.3	192.168.10.255	ICMP	62	Information request id=0x0200, seq=456/51201, ttl=32
10	41.72474400(	192.168.10.3	192.168.10.255	ICMP	62	Information request id=0x0200, seq=459/51969, ttl=32

```
▸ Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
▸ Ethernet II, Src: National_0a:df:3e (00:80:2f:0a:df:3e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Internet Protocol Version 4, Src: 192.168.10.3 (192.168.10.3), Dst: 192.168.10.255 (192.168.10.255)
▸ Internet Control Message Protocol
▸ VSS-Monitoring ethernet trailer, Source Port: 22852

0000  ff ff ff ff ff ff 00 80  2f 0a df 3e 08 00 45 00   ........ /..>..E.
0010  00 1c 00 00 40 00 20 01  c4 8e c0 a8 0a 03 c0 a8   ....@. . ........
0020  0a ff 0f 00 ed 49 02 00  01 b6 00 00 00 00 00 00   .....I.. ........
0030  00 00 00 00 00 00 00 00  00 00 00 00 59 44         ........ ....YD
```

File: "/tmp/wireshark_pcapng_eth1_... Packets: 10 · Displayed: 10 (100.0%) · Dropped: 0 (0.0%) Profile: Default

Debugging USRP with Wireshark

Installing the UHD / CHDR Dissector

Plugin source is located with UHD repository at: **`uhd/tools/dissectors`**

To install, the Wireshark Development files are required:

```
sudo apt-get install wireshark-dev
```

Next navigate to the dissectors directory:

```
cd ~/workarea/uhd/tools/dissectors
mkdir build
cd build
cmake ..
make
sudo make install
```

`*` Dissectors are also available for ZPU and OctoClock

Enable by passing "**`zpu`**" or "**`octoclock`**" option during **`cmake`** configuration step:

```
cmake -DETTUS_DISSECTOR_NAME=zpu ../
```

Debugging USRP with Wireshark

To sniff the USB bus, you must first enable the **USBMON** kernel module:

```
sudo modprobe usbmon
```

To give regular users privileges, make the usbmonX device(s) readable:

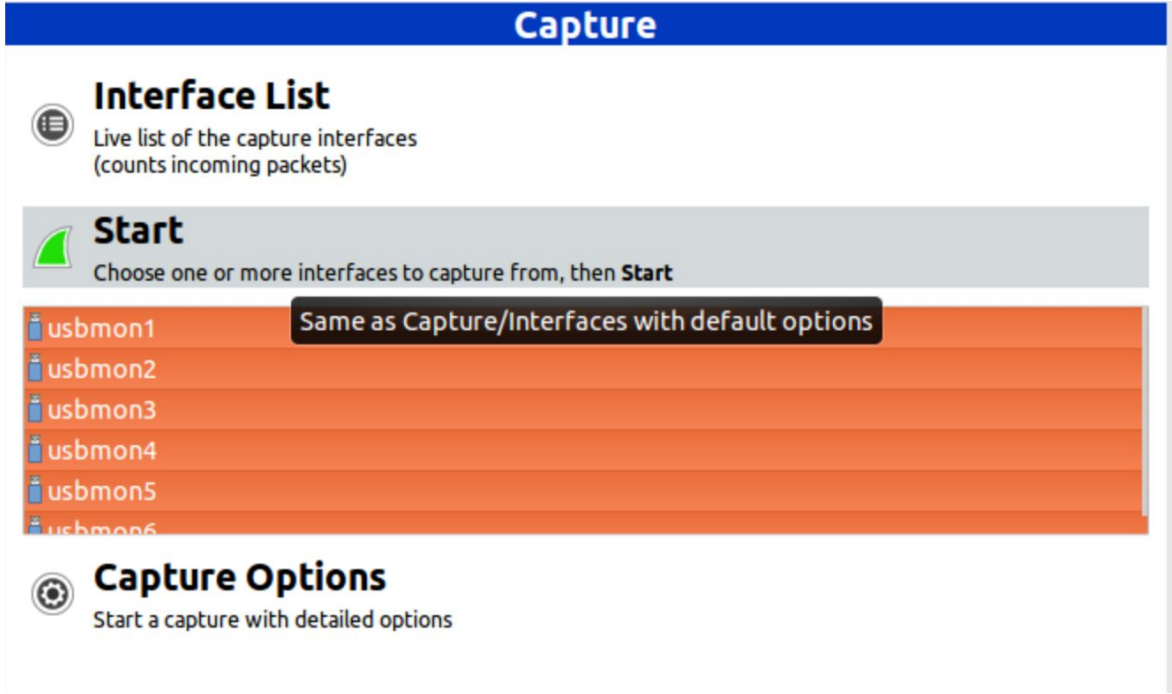
```
sudo setfacl -m u:$USER:r /dev/usbmon*
```

Next, run wireshark:

```
wireshark
```

Debugging USRP with Wireshark

Upon startup, select all (highlight one and press **`CTRL+A`**) of the **`usbmonX`** interfaces and click **`Start`**



Debugging USRP with Wireshark

Ettus Research™
A National Instruments Company

Filter: Expression... Clear Apply Save

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

Debugging USRP with 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   ....@...  .......
```

Debugging USRP with Wireshark

Ettus Research™
A National Instruments Company

No.	Time	Source	Destination	Protocol	Length	Info
32345	22.729325006	2.3	host	USB	66	URB_INTERRUPT in
32346	22.729326006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32347	22.729327006	host	2.3	USB	64	URB_INTERRUPT in
32348	22.729376006	host	2.6	USB	64	URB_BULK in
32349	22.731365006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32350	22.731427006	host	2.6	USB	64	URB_BULK in
32351	22.733409006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32352	22.733469006	host	2.6	USB	64	URB_BULK in
32353	22.735452006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32354	22.735480006	host	2.6	USB	64	URB_BULK in
32355	22.737497006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32356	22.737525006	host	2.6	USB	64	URB_BULK in
32357	22.739540006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32358	22.739578006	host	2.6	USB	64	URB_BULK in
32359	22.741585006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32360	22.741624006	host	2.6	USB	64	URB_BULK in
32361	22.743628006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32362	22.743656006	host	2.6	USB	64	URB_BULK in
32363	22.745673006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32364	22.745701006	host	2.6	USB	64	URB_BULK in
32365	22.747717006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32366	22.747756006	host	2.6	USB	64	URB_BULK in
32367	22.749761006	2.6	host	USB/CHDR	8256	URB_BULK in, CHDR
32368	22.749800006	host	2.6	USB	64	URB_BULK in

Debugging USRP with Wireshark

```
▼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: 0000ffff02000100ffffdff00000100feffeff0100ffff...

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%)
```

Debugging USRP with 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: 0000ffff02000100ffffdff00000100feffeff0100ffff...

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%)
```

Debugging USRP with 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: 0000ffff02000100ffffdff00000100feffeff0100ffff...

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   ........ ........

Time (chdr.time), 8 bytes          Packets: 60391 · Displayed: 60391 (100.0%)
```

Device EEPROM

- Motherboard EEPROM
  - Contains name, serial number, hardware revision number, compat number, product ID, MAC address, IP address, netmask
- Daughterboard EEPROM
  - Product name and ID (hex code), hardware revision number, serial number
- Two utilities for reading and writing EEPROMs
  - **`/usr/local/lib/uhd/utils/usrp_burn_db_eeprom`**
    - Specify slot and Tx/Rx
  - **`/usr/local/lib/uhd/utils/usrp_burn_mb_eeprom`**
    - Use the **`--read-all`** argument

MB EEPROM Output

Terminal

File Edit View Terminal Tabs Help

```
neel@nipton:~ >> /usr/local/lib/uhd/utils/usrp_burn_mb_eeprom --read-all --args="addr=192.168.10.3"
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.002-0-gf18abe54

Creating USRP device from address: addr=192.168.10.3
-- X300 initialization sequence...
-- Determining maximum frame size... 1472 bytes.
-- Setup basic communication...
-- Loading values from EEPROM...
-- Setup RF frontend clocking...
-- Radio 1x clock:200
-- Initialize Radio0 control...
-- Performing register loopback test... pass
-- Initialize Radio1 control...
-- Performing register loopback test... pass

Fetching current settings from EEPROM...
    EEPROM ["revision"] is "4"
    EEPROM ["revision_compat"] is "4"
    EEPROM ["product"] is "30518"
    EEPROM ["mac-addr0"] is "00:80:2f:0a:df:3e"
    EEPROM ["mac-addr1"] is "00:80:2f:0a:df:3f"
    EEPROM ["gateway"] is "192.168.10.1"
    EEPROM ["ip-addr0"] is "192.168.10.3"
    EEPROM ["subnet0"] is "255.255.255.0"
    EEPROM ["ip-addr1"] is "192.168.20.2"
    EEPROM ["subnet1"] is "255.255.255.0"
    EEPROM ["ip-addr2"] is "192.168.30.2"
    EEPROM ["subnet2"] is "255.255.255.0"
    EEPROM ["ip-addr3"] is "192.168.40.2"
    EEPROM ["subnet3"] is "255.255.255.0"
    EEPROM ["serial"] is "F43D1B"
    EEPROM ["name"] is ""

Power-cycle the USRP device for the changes to take effect.

Done
neel@nipton:~ >>
```

DB EEPROM Output

Ettus Research™
A National Instruments Company

Terminal

File Edit View Terminal Tabs Help

```
neel@nipton:~/Desktop >> /usr/local/lib/uhd/utils/usrp_burn_db_eeprom --args="addr=192.168.10.3" --slot B --unit RX
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.002-0-gf18abe54

-- X300 initialization sequence...
-- Determining maximum frame size... 1472 bytes.
-- Setup basic communication...
-- Loading values from EEPROM...
-- Setup RF frontend clocking...
-- Radio 1x clock:200
-- Initialize Radio0 control...
-- Performing register loopback test... pass
-- Initialize Radio1 control...
-- Performing register loopback test... pass
Reading RX EEPROM on B dboard...
  Current ID: WBX v3, WBX v3 + Simple GDB (0x0057)
  Current serial: "306E9E3"
  Current revision: ""
  Done

neel@nipton:~/Desktop >>
```

GNU Radio

- Open-source framework for SDR and signal processing
- Founded by Eric Blossom in 2001
- Project leader Ben Hilburn (formerly Tom Rondeau)
- 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 MathWorks 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/`**

GNU Radio Installation - Source Install

Ubuntu 18.04 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
```

Reference: ~/ettus_workshop/instructions/install.html

GNU Radio Installation - Source Install

Building GNU Radio

```
1.   cd ~/workarea
2.   git clone --recursive https://github.com/gnuradio/gnuradio.git
3.   cd gnuradio/
4.   git checkout v3.7.10.1
5.   mkdir build && cd build
6.   cmake ../
7.   make -j4
8.   sudo make install
9.   sudo ldconfig
```

Reference: ~/ettus_workshop/instructions/install.html

GNU Radio Installation - Other Methods

- PyBOMBS
  - Python Build Overlay Managed Bundle System
  - Package management tool for GNU Radio, similar to RPM, yum, apt-get, and pip
  - New version 2 just released
  - **`http://www.pybombs.info`**
- The **`build-gnuradio`** script
  - From Marcus Leech at CCERA (formerly SBRAC)
  - **`wget http://www.sbrac.org/files/build-gnuradio && chmod a+x ./build-gnuradio && ./build-gnuradio`**
- Launchpad PPA / Binary Packages
  - Often old versions are packages
  - Avoid installing from binary package
  - Ubuntu 14.04 packages GR 3.7.2.1
  - Ubuntu 15.10 packages GR 3.7.8

UHD / GNU Radio Installation App Notes

- Detailed step-by-step instructions for the following operating systems:
  - Ubuntu 18.04, 18.10, 19.04
  - Fedora 27, 28, 29
  - Windows 7, 8, 10
  - OS X
- Located in Ettus Research Knowledge Base
  - kb.ettus.com

- https://kb.ettus.com/Building_and_Installing_the_USRP_Open-Source_Toolchain_(UHD_and_GNU_Radio)_on_Linux
- https://kb.ettus.com/Building_and_Installing_the_USRP_Open_Source_Toolchain_(UHD_and_GNU_Radio)_on_Windows
- https://kb.ettus.com/Building_and_Installing_the_USRP_Open-Source_Toolchain_(UHD_and_GNU_Radio)_on_OS_X

GNU Radio Installation - Footnotes

- Always install UHD first, before installing GNU Radio
  - Otherwise, GNU Radio will not know to enable the gr-uhd component at build-time
  - Without gr-uhd, GNU Radio does not know how to use USRP devices
- Whenever UHD changes (upgrade or downgrade), you may need to re-build and re-install gr-uhd
  - Symptom: GR runs fine, but then suddenly crashes when you run with a USRP
  - This occurs because the UHD ABI, and sometimes API, can change from release to release
  - gr-uhd needs to be updated to accommodate such changes
  - The re-build and re-install of gr-uhd is quick
- If GR changes (upgrade or downgrade), you do <u>not</u> need to go back and re-build UHD

Parallel UHD and GR Installations

- Instead of switching versions, install multiple versions of UHD and GR in parallel
- Often done for testing, application development, debugging, running older legacy applications
- Tell CMake where to put the installation
  - Invoke CMake with:

    `-DCMAKE_INSTALL_PREFIX=<install-path>`
- The default location is **`/usr/local`**
- Commonly put parallel versions in **`/opt/uhd-x.y.z`** and **`/opt/gnuradio-x.y.z`**
- Switch between versions by setting environment variables appropriately:
  - **`$PATH`**
  - **`$LD_LIBRARY_PATH`**
  - **`$PYTHONPATH`**
  - **`$PKG_CONFIG_PATH`**

Switching Versions of UHD and GR

- Sometimes it is necessary to downgrade or upgrade UHD and/or GR
- Often done for testing, application development, debugging, running older legacy applications
- To switch versions:
  - go to the "**`build`**" folder in the repository
  - checkout a specific tagged release from Git
  - re-build
  - re-install

```
cd /home/user/uhd/host/build
git checkout release_003_008_004
make -j4
sudo make install
```

- Verify version switch by running **`uhd_find_devices`** and looking at the first line of output

GR Managing Multiple Installations

This slide continues upon the previously made parallel UHD installation.
This installation will reference the locally installed UHD, and install GNU Radio locally.

```
cd ~/installs/uhd_396/src
git clone --recursive -b v3.7.10.1 https://github.com/gnuradio/gnuradio.git
mkdir build && cd build

cmake -DCMAKE_INSTALL_PREFIX=/home/demo/installs/uhd_396/
-DUHD_DIR=/home/demo/installs/uhd_396/lib/cmake/uhd/
-DUHD_INCLUDE_DIRS=/home/demo/installs/uhd_396/include/
-DUHD_LIBRARIES=/home/demo/installs/uhd_396/lib/libuhd.so ../

make -j4
make install
```

Use this locally built version of GNU Radio by running "**`source setup_env.sh`**" against the previously created "**`setup_env.sh`**" script created under the "*UHD Managing Multiple Installations*" section.

Removing a UHD or GR Installation

- To remove an installation of UHD or GNU Radio:
  - Go into the “build” folder of the repository
  - Tell Make to do an uninstall
  - Optionally, remove the Git repository, if no longer needed

```
cd /home/user/uhd/host/build
make uninstall
rm -rf /home/user/uhd
```

  - Note that this procedure does not remove any GNU Radio OOT modules
    - Need to be removed separately and individually
    - Remove all OOT modules first, before removing GNU Radio itself

Testing the GNU Radio Installation

- Configuration utility:
  - `gnuradio-config-info --version` (or `-v`)
  - `gnuradio-config-info --prefix`
  - `gnuradio-config-info --enabled-components`
- In Python interpreter, run:

```
import gnuradio
```

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

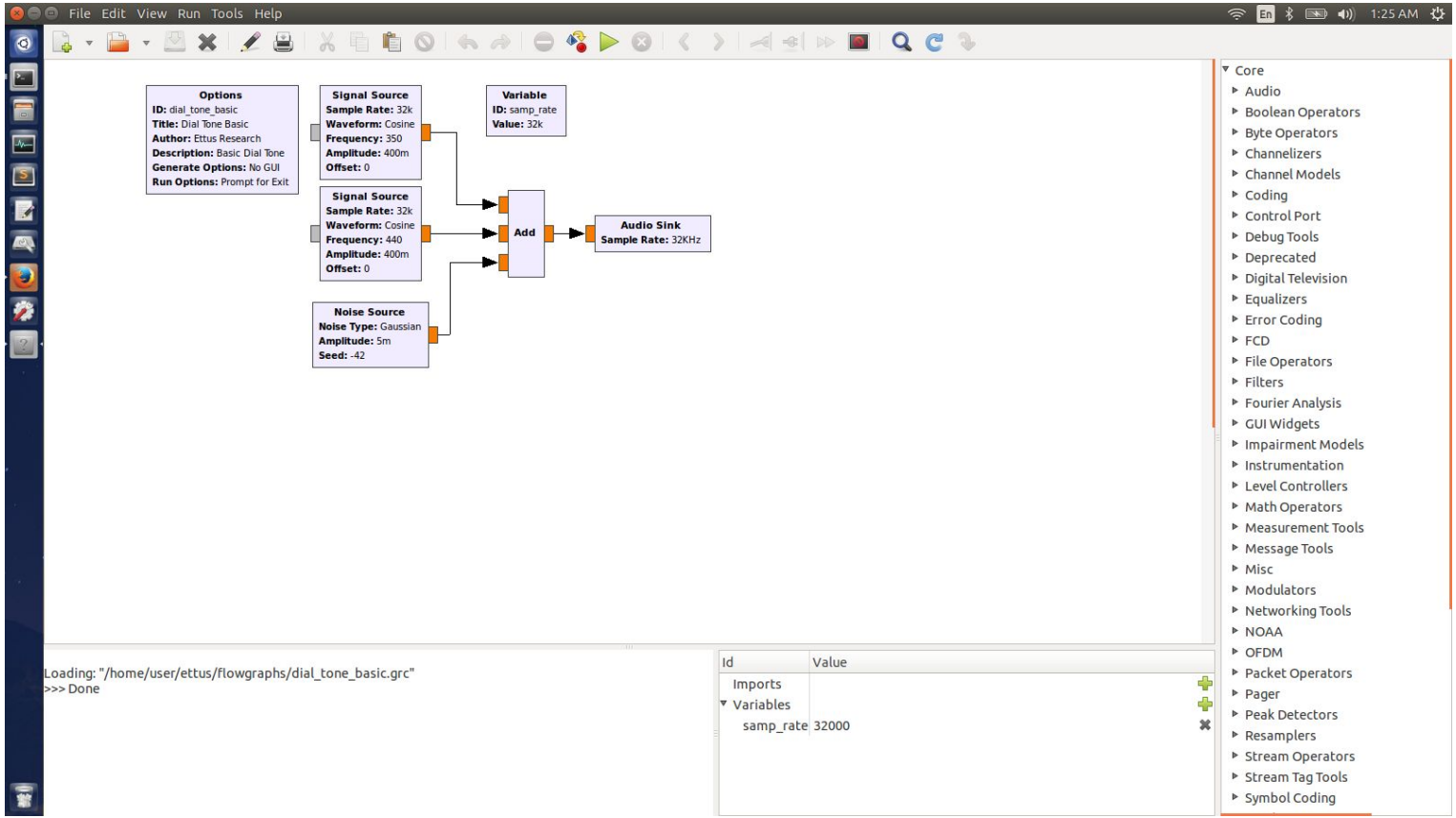
```
##################################################
# Blocks
##################################################
self.blocks_add_xx = blocks.add_vff(1)
self.audio_sink = audio.sink(32000, '', True)
self.analog_sig_source_x_1 = analog.sig_source_f(samp_rate, analog.GR_COS_WAVE, 440, .4, 0)
self.analog_sig_source_x_0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

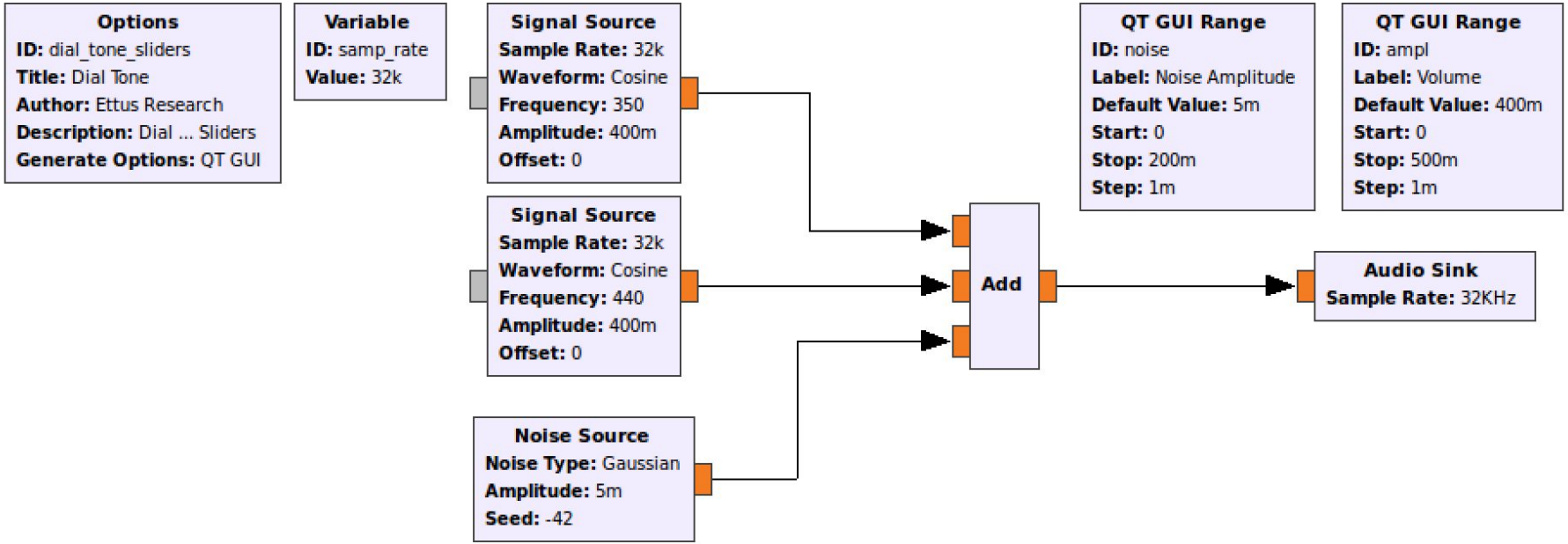
Dial Tone Example: Flowgraph

Location: ~/ettus_workshop/flowgraphs/dial_tone_basic.grc



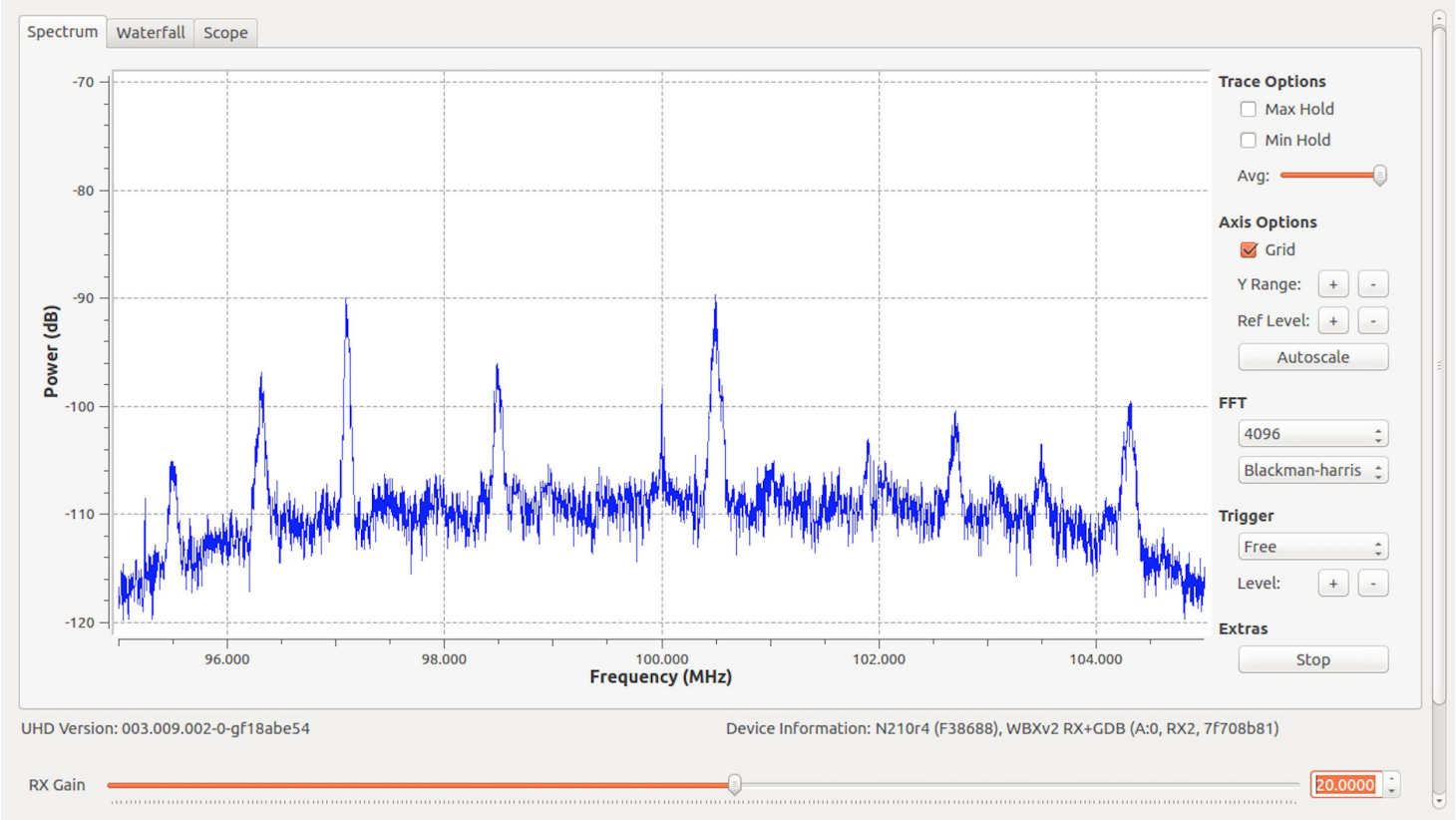
Example: Dial Tone with Slider Widgets

Location: ~/ettus_workshop/flowgraphs/dial_tone_sliders.grc



Verifying USRP using GNU Radio

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



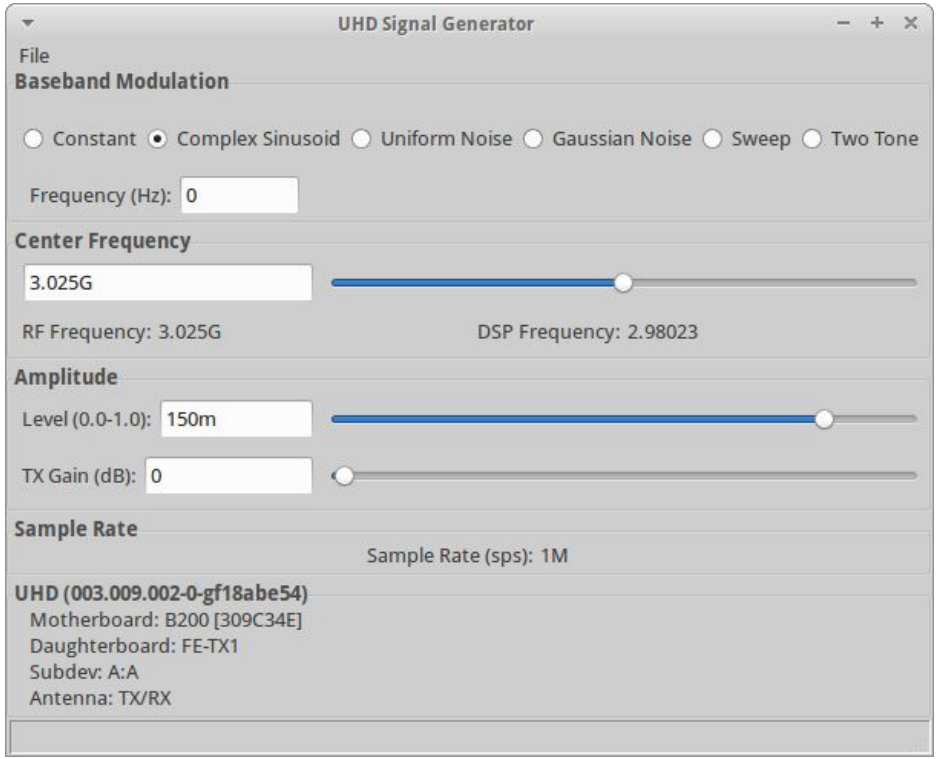
Verifying USRP using GNU Radio

```
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:~$
```

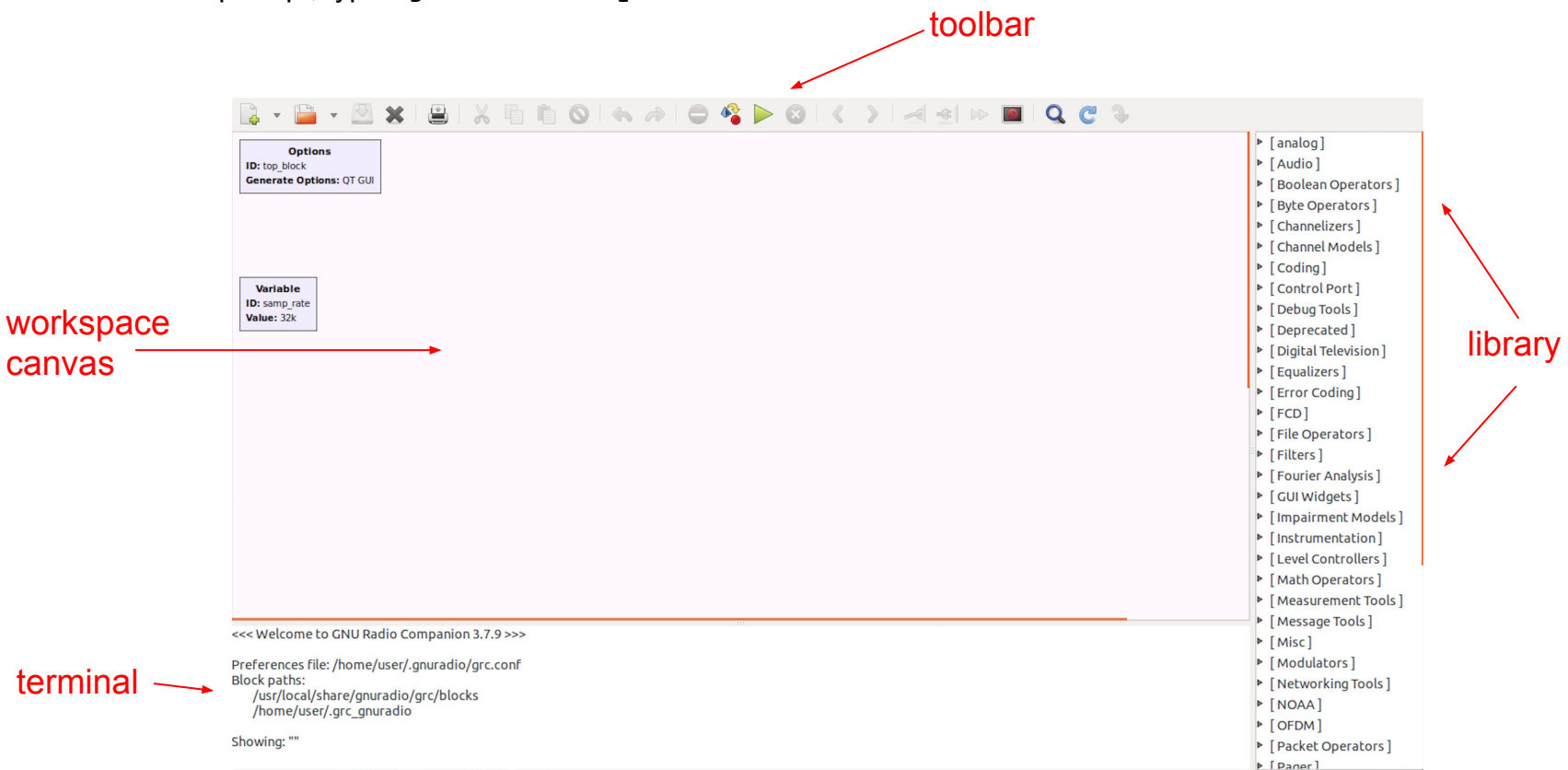
Verifying USRP using GNU Radio

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

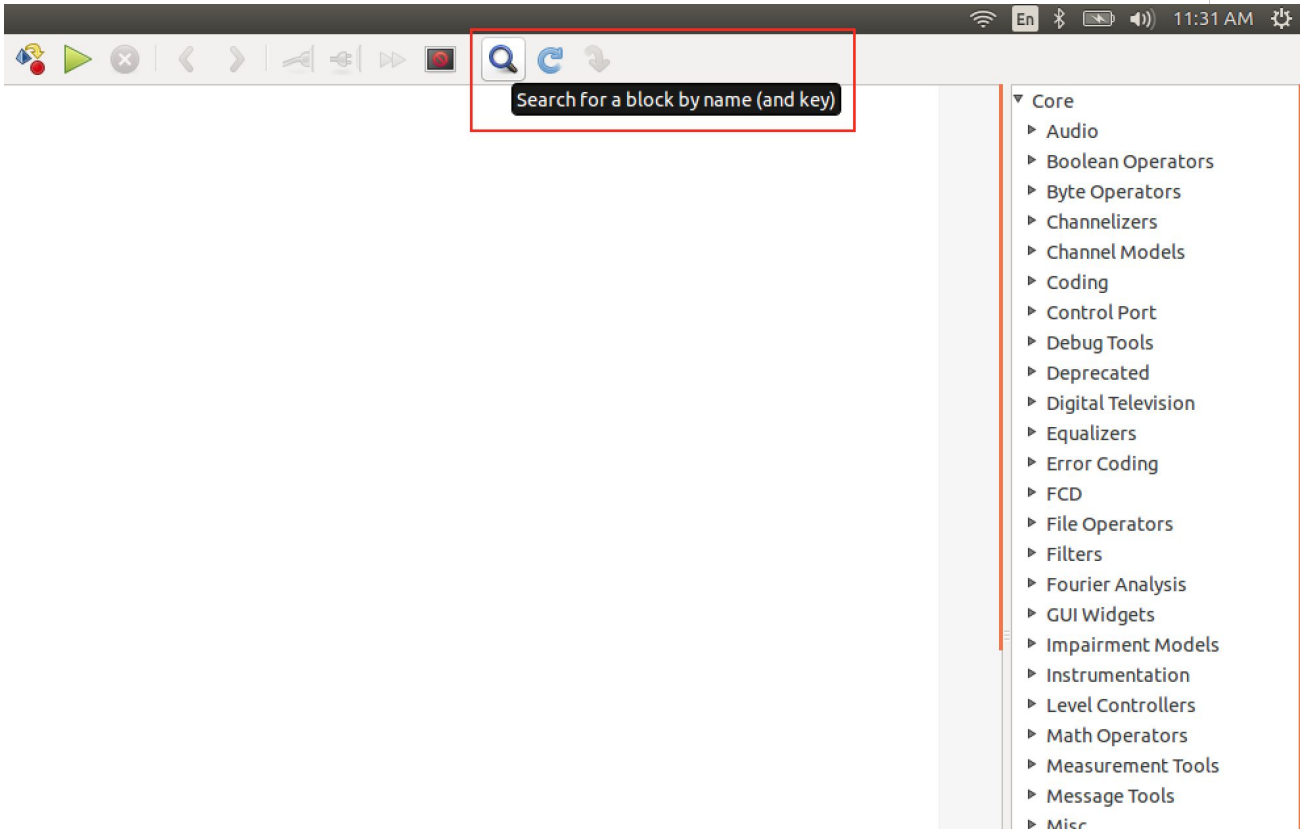


Using gnuradio-companion

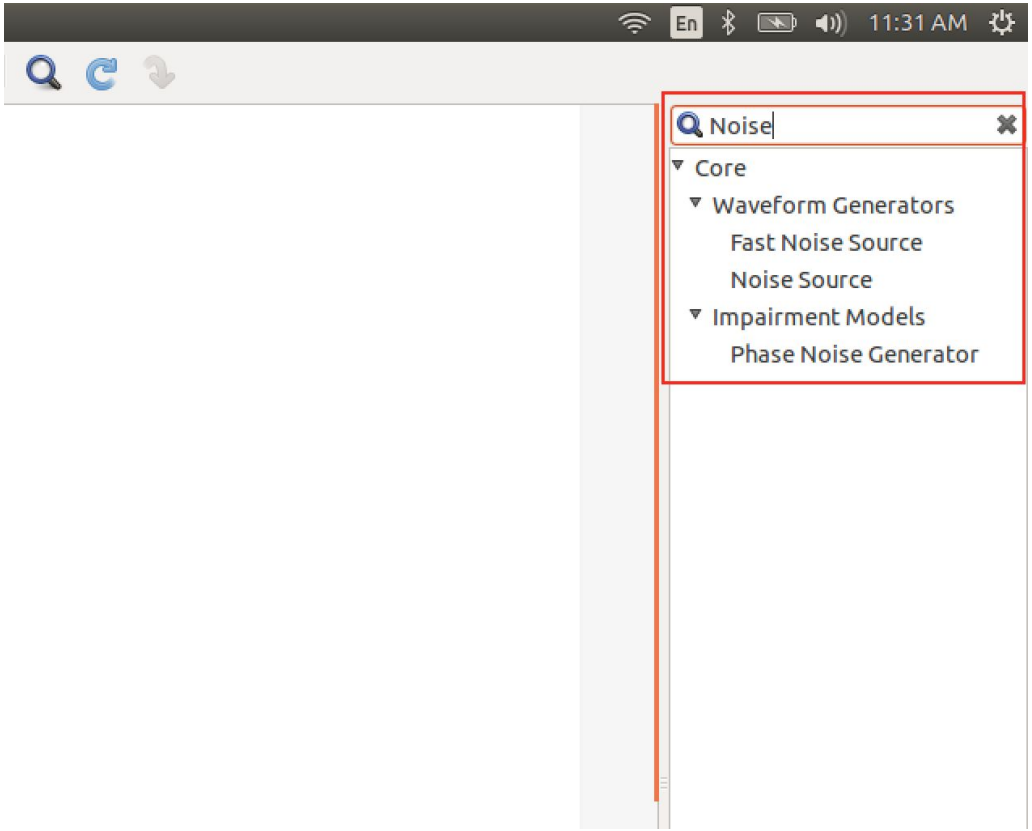
At a command prompt, type: **`gnuradio-companion`**



Using gnuradio-companion - Search



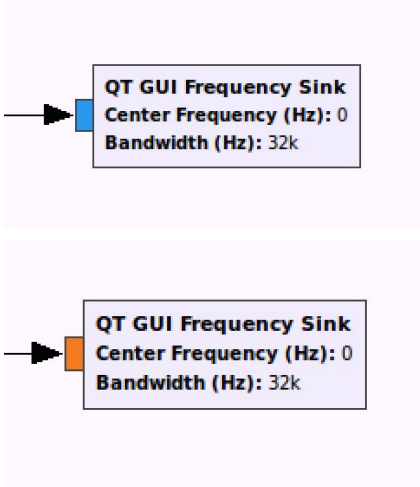
Using gnuradio-companion - Search



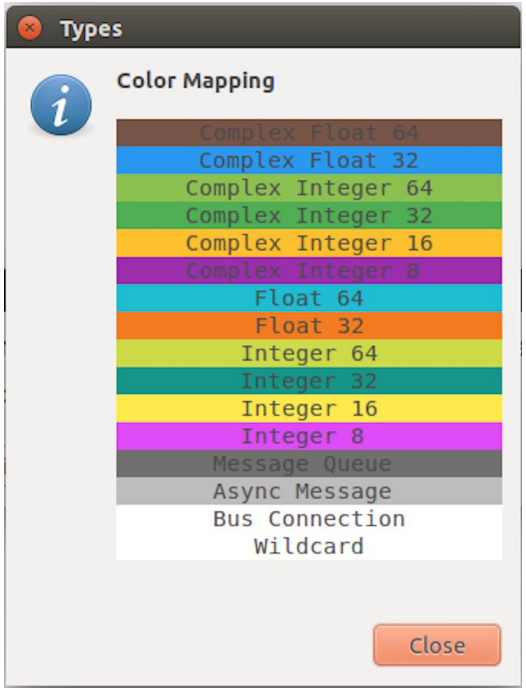
Using gnuradio-companion

Blocks have ports which input and output specific data types.

The color of the port indicates its data type.



Help -> Types

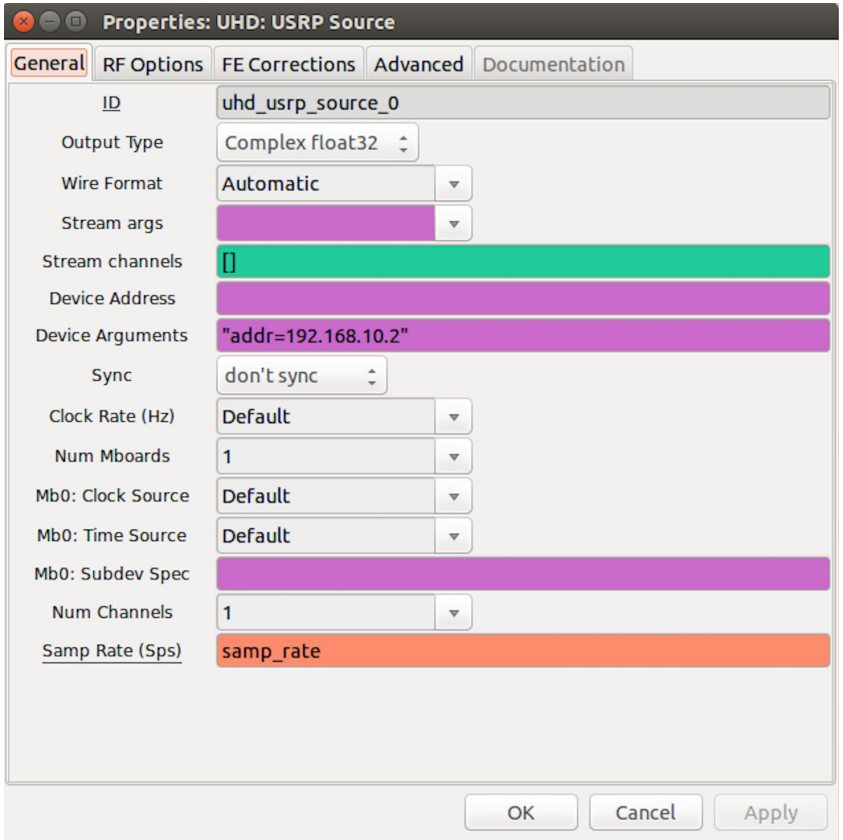


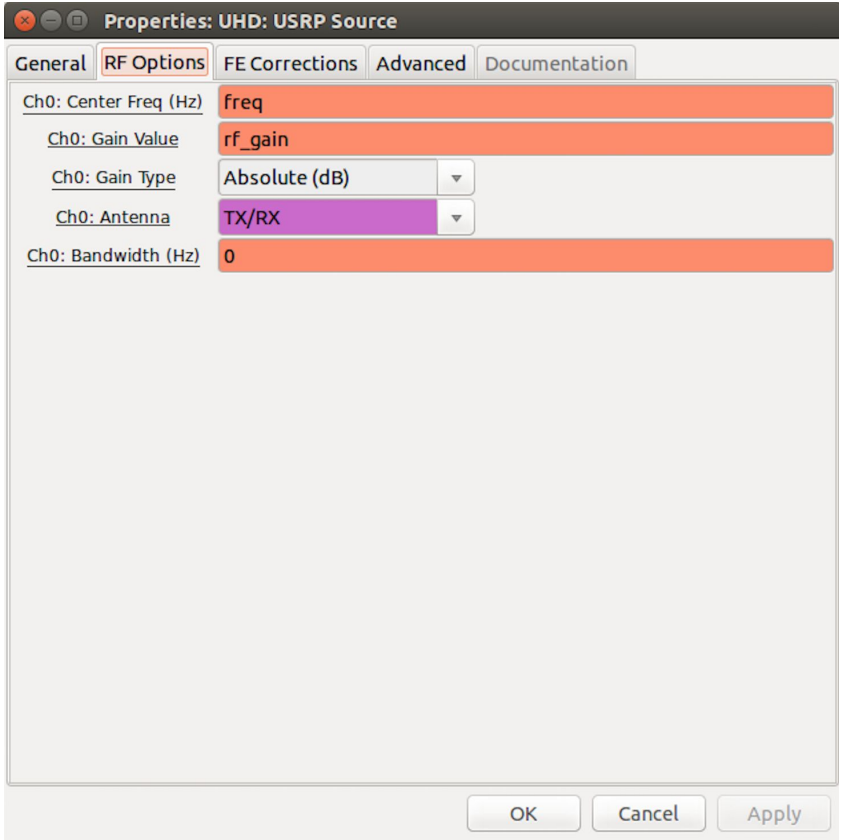
Hot keys:

- 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

Properties: UHD: USRP Sink

General | RF Options | Advanced | Documentation

ID	uhd_usrp_sink_0
Input Type	Complex float32
Wire Format	Automatic
Stream args	
Stream channels	[]
Device Address	""
Device Arguments	"addr=192.168.10.2"
Sync	don't sync
Clock Rate (Hz)	Default
Num Mboards	1
Mb0: Clock Source	Default
Mb0: Time Source	Default
Mb0: Subdev Spec	
Num Channels	1
Samp Rate (Sps)	samp_rate
TSB tag name	

OK | Cancel | Apply

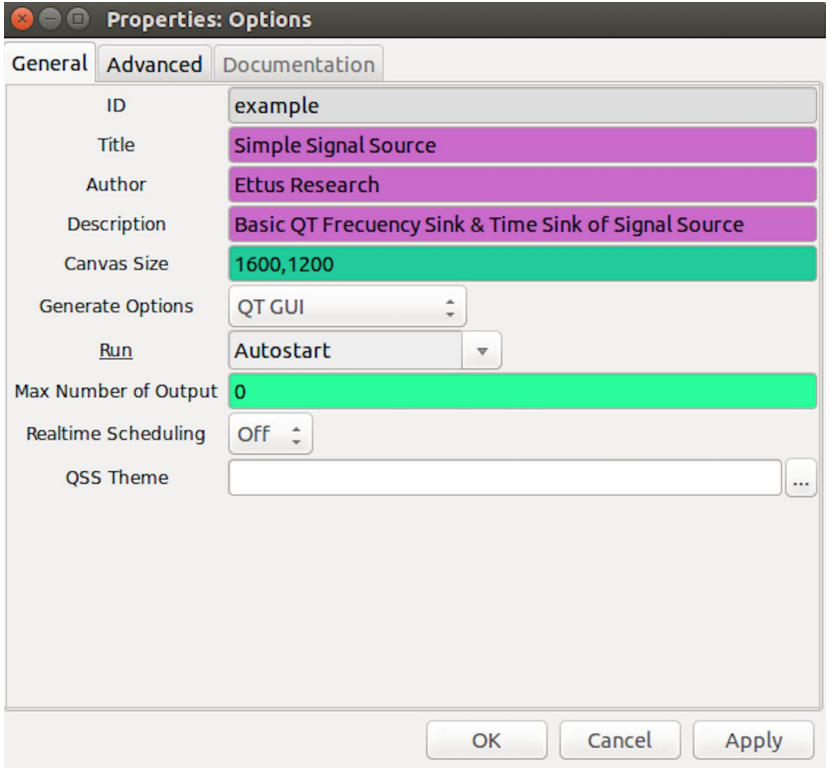
Properties: UHD: USRP Sink

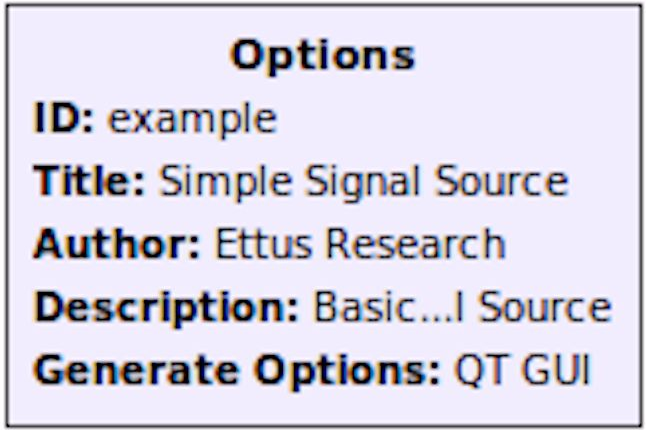
General | RF Options | Advanced | Documentation

Ch0: Center Freq (Hz)	center_freq
Ch0: Gain Value	rf_gain
Ch0: Gain Type	Absolute (dB)
Ch0: Antenna	TX/RX
Ch0: Bandwidth (Hz)	0

OK | Cancel | Apply

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 GUI Interface
  - QT is becoming the primary GUI toolkit for GNU Radio going forward
  - QT 4 currently, QT 5 coming soon

- **`gr-wxgui`**
  - WX GUI Interface
  - wxWidgets is being deprecated in GNU Radio

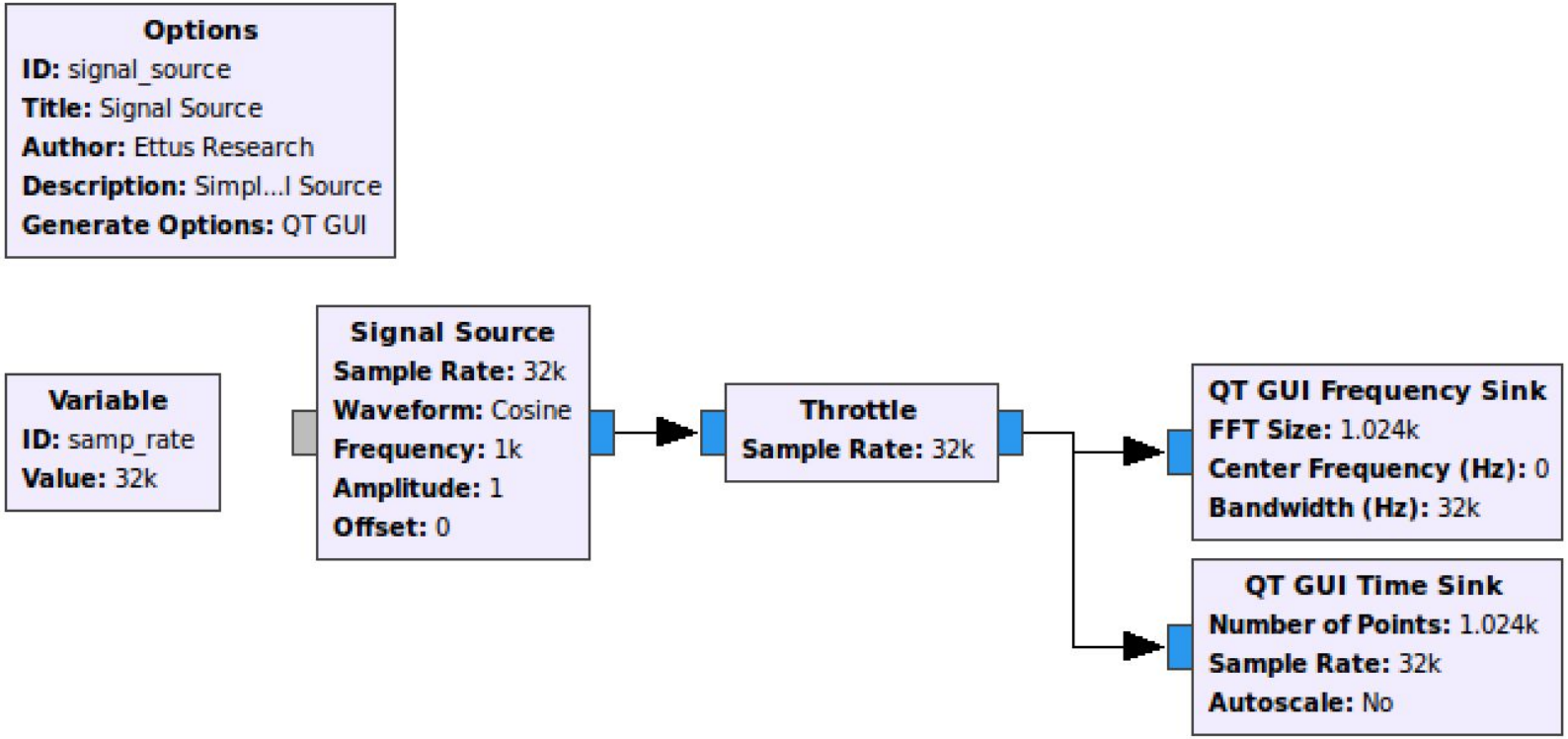
Components of GNU Radio

Hardware Interfaces

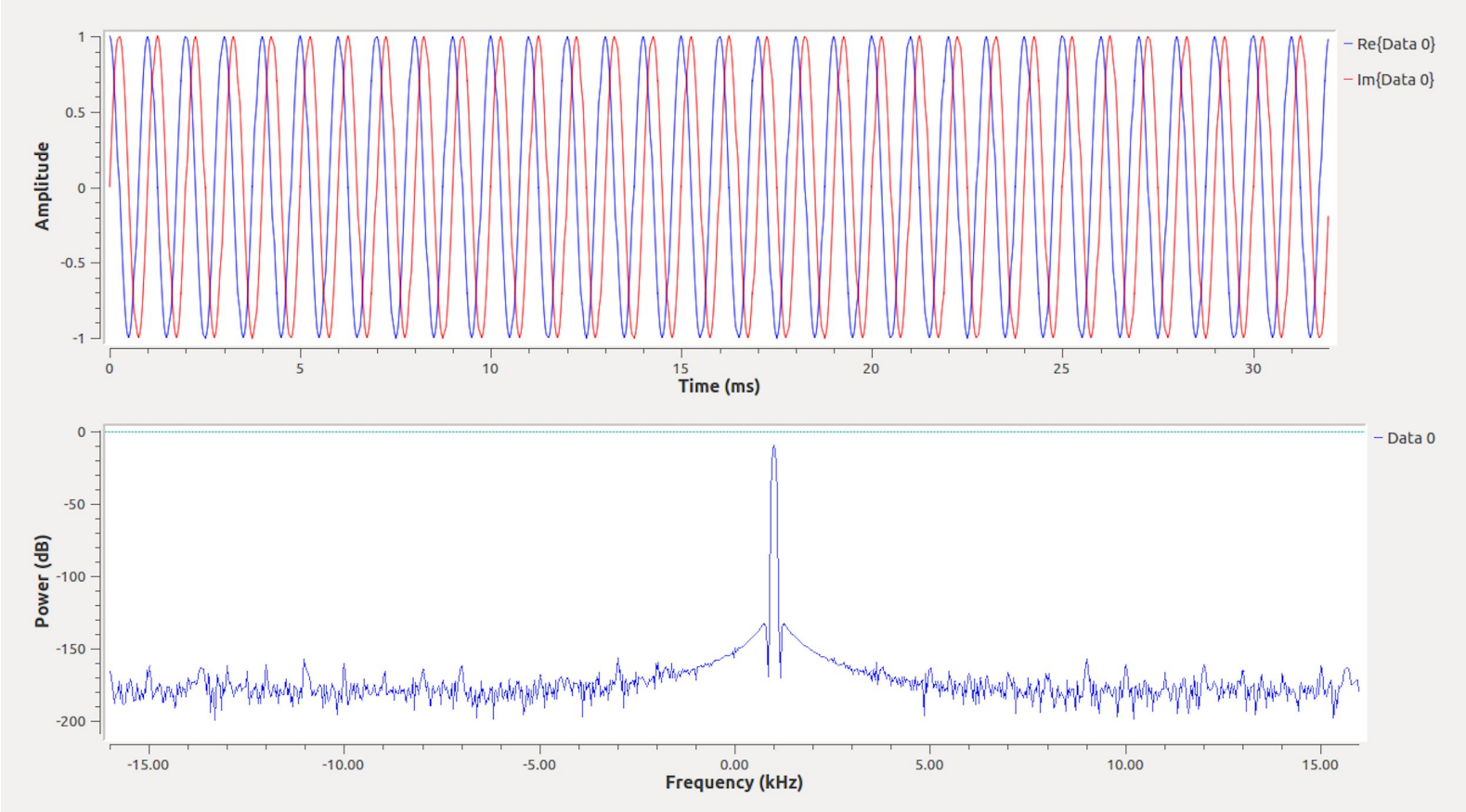
- **`gr-audio`**
  - Block for all supported audio sound systems
- **`gr-comedi`**
  - Blocks for the comedi library
- **`gr-fcd`**
  - Funcube Dongle source block for GNU Radio
- **`gr-shd`**
  - Blocks for the Simplex Hardware Driver (SHD)
- **`gr-uhd`**
  - Blocks to interface with USRP / UHD
- **`gr-osmocom`**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

Example: Signal Source

Location: ~/ettus_workshop/flowgraphs/signal_source.grc

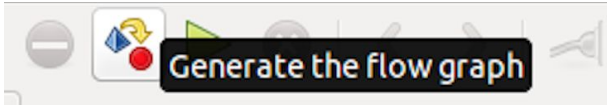


Example: Signal Source Running



Using GNU Radio from Python

Generate Python from GRC Flow graph



Invoke directly from the Linux command line:

```
$ python example_3.py
```

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

```
#!/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

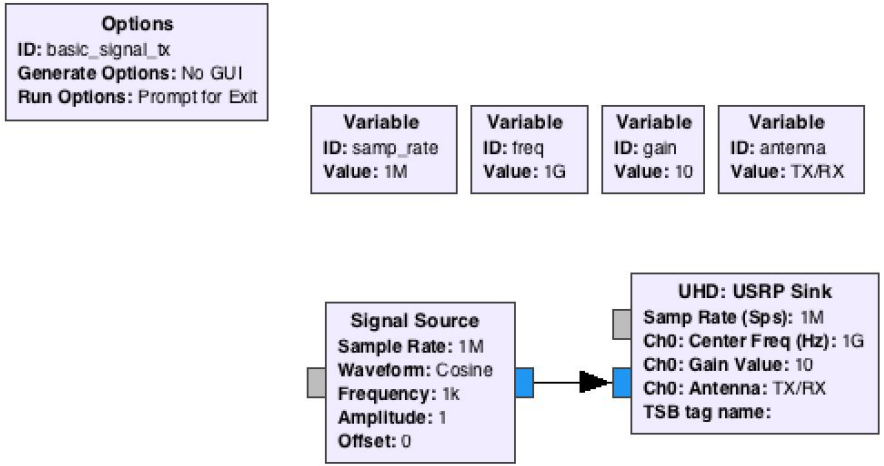
Ettus Research™
A National Instruments Company

```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
```

```python
    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

Ettus Research™
A National Instruments Company

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

Ettus Research™
A National Instruments Company

Location: ~/ettus_workshop/flowgraphs/basic_signal_tx.py

Passing of created class to main()

```
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()
```

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

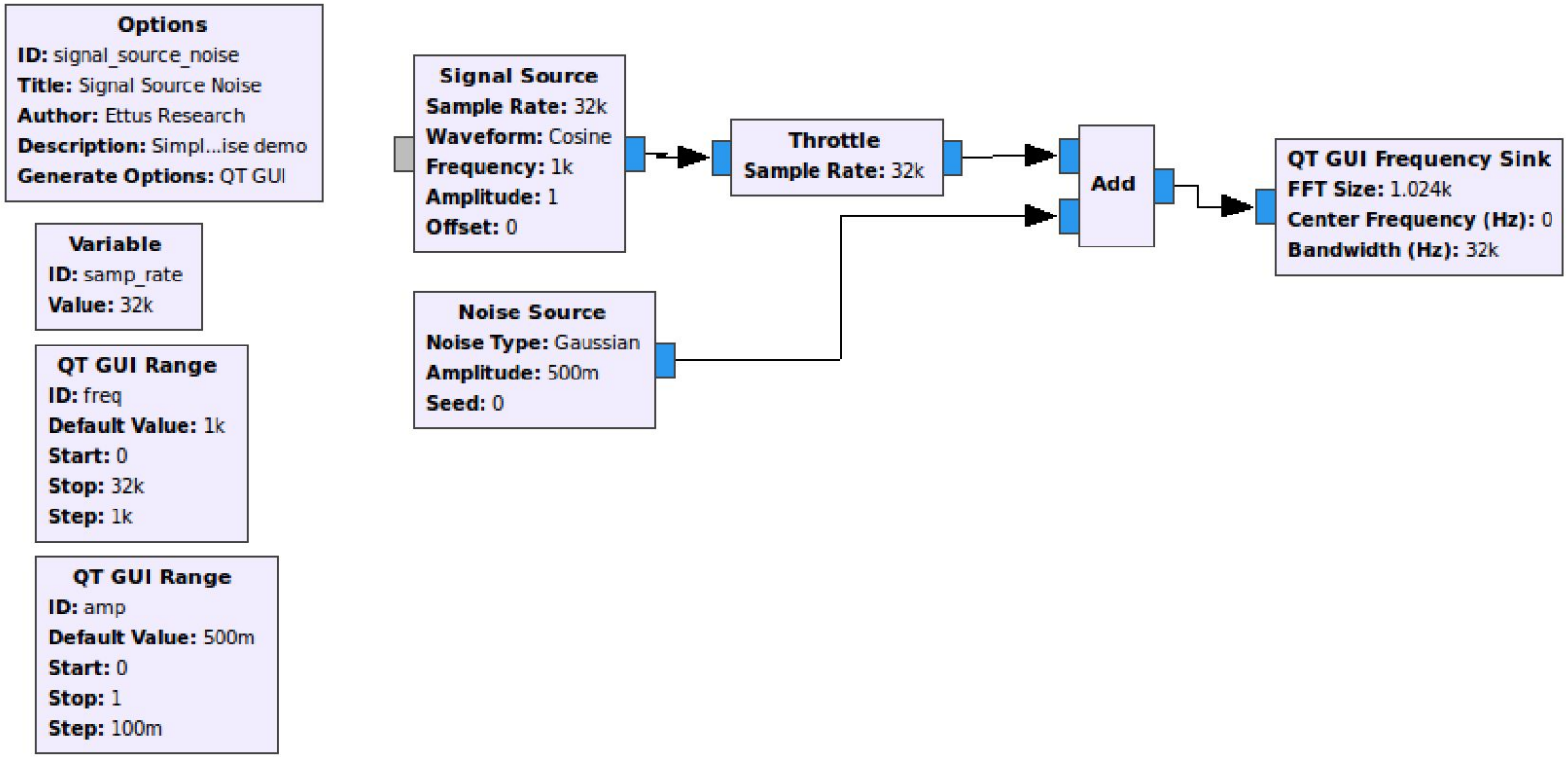
Example: Basic Signal Transmission

Additional details of Operating a Flowgraph:

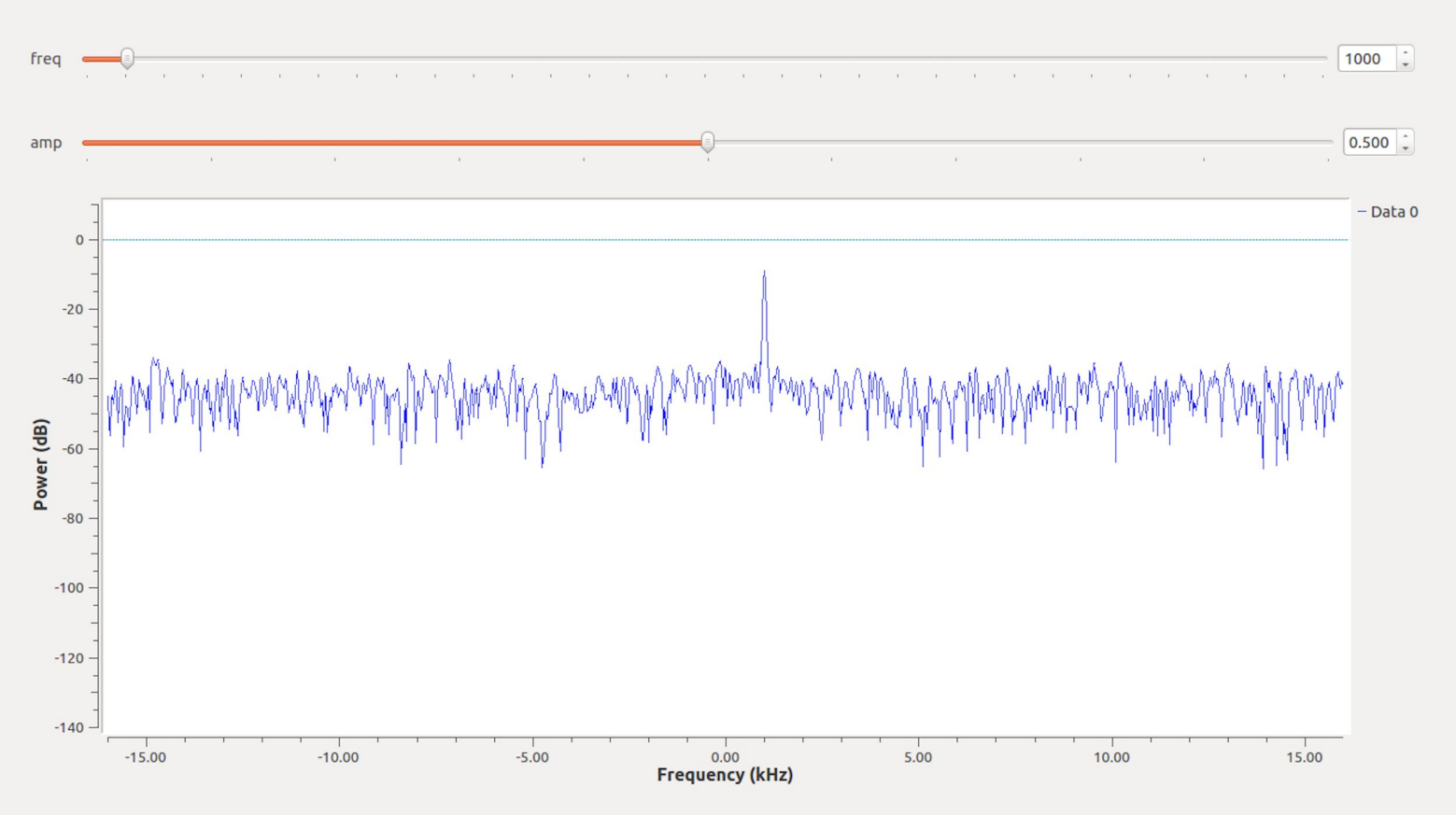
http://gnuradio.org/doc/doxygen/page_operating_fg.html

Example: Signal Source with Noise

Location: ~/ettus_workshop/flowgraphs/signal_source_noise.grc

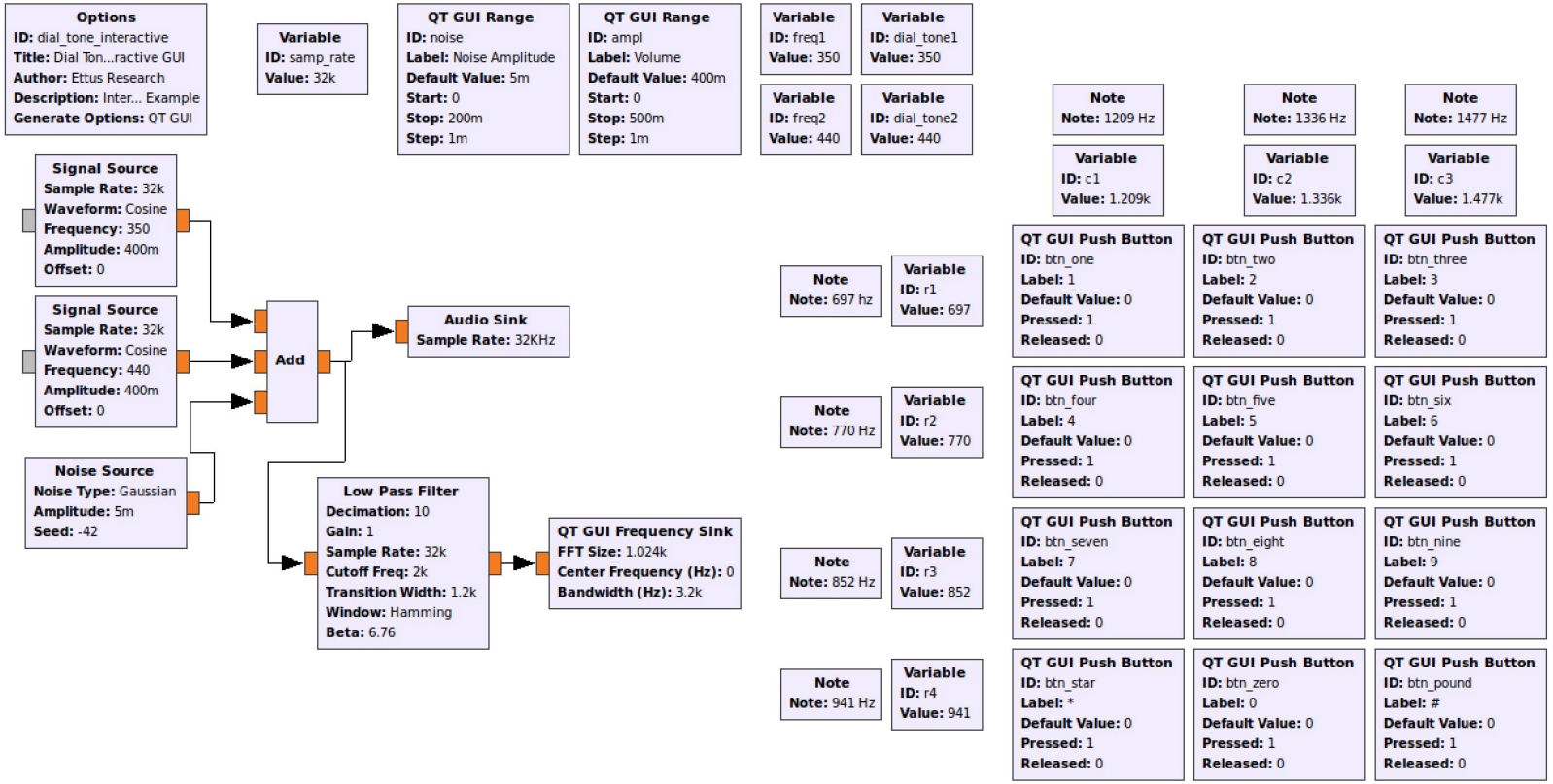


Example: Signal Source with Noise Running



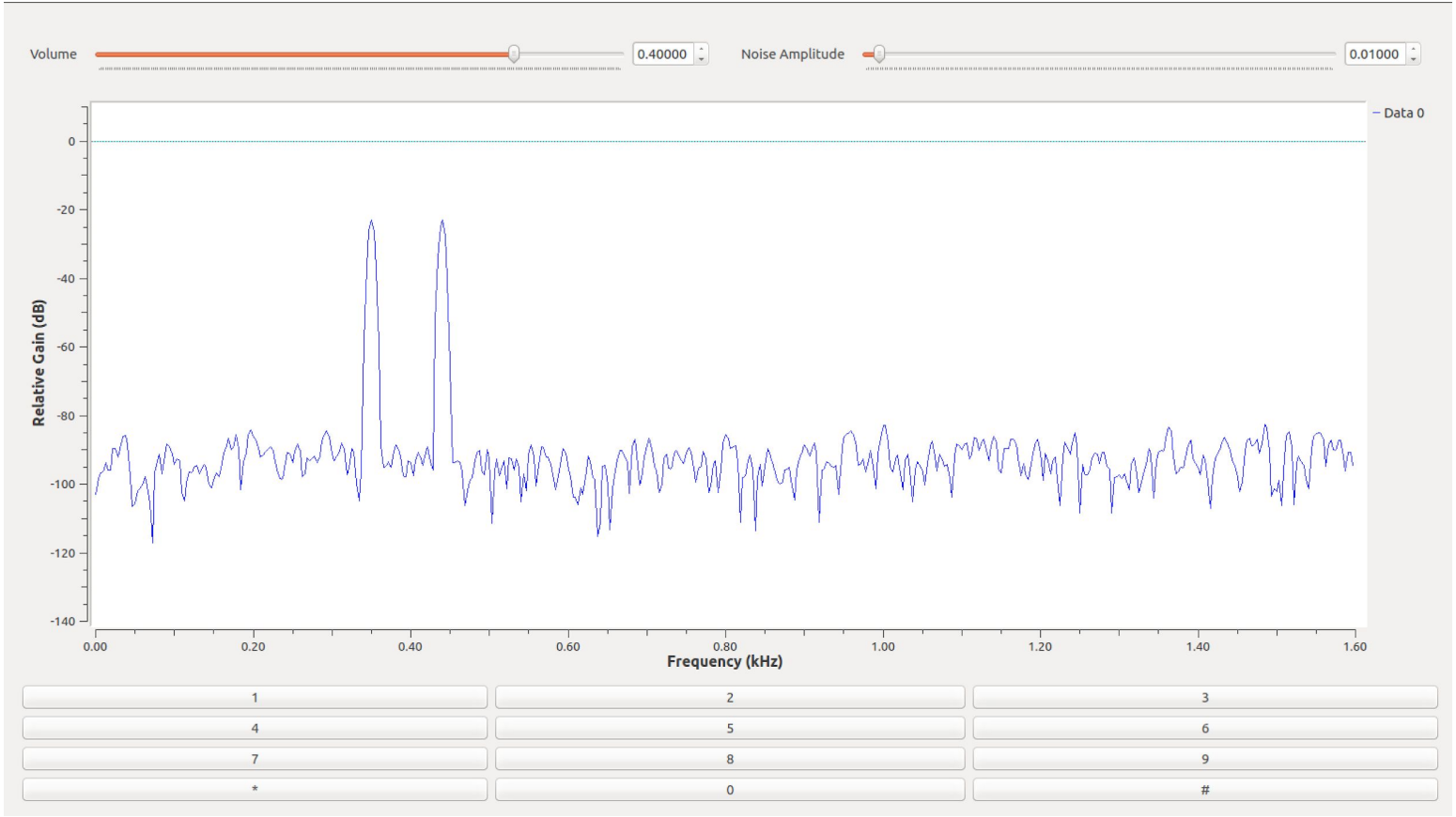
Example: Dial Tone / Touch Tone

Location: ~/ettus_workshop/flowgraphs/dial_tone_interactive.grc



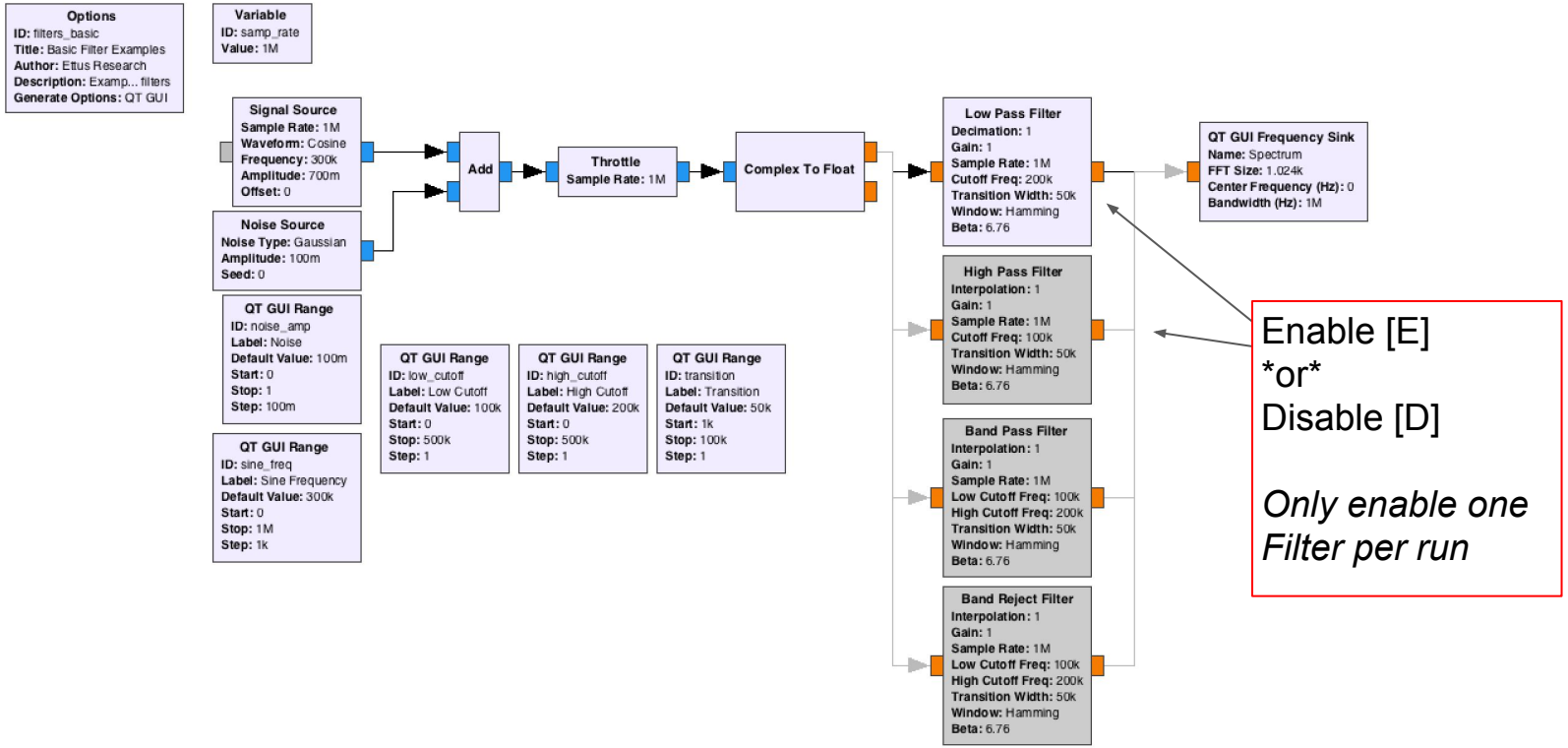
Example: Dial Tone / Touch Tone

Location: ~/ettus_workshop/flowgraphs/dial_tone_interactive.grc



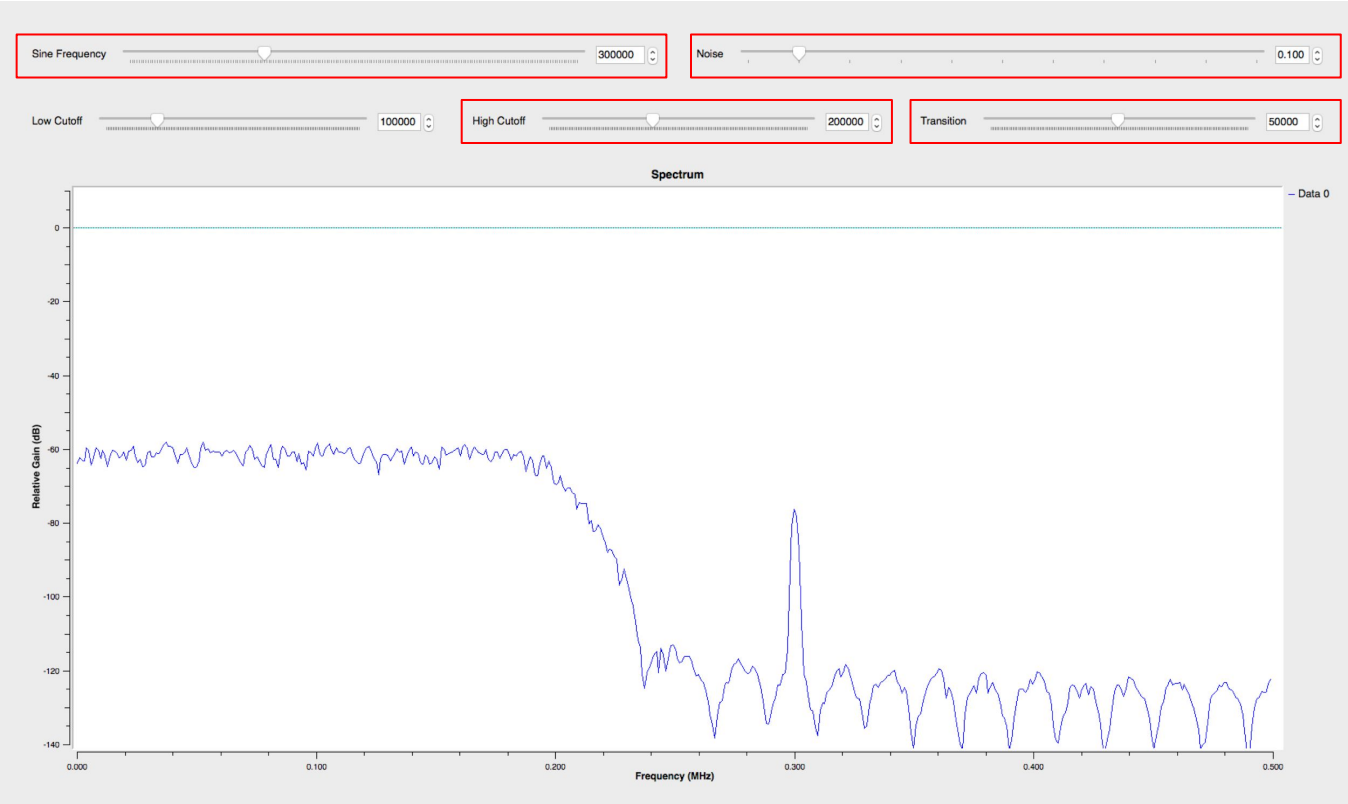
Example: Filters - Flowgraph

Location: ~/ettus_workshop/flowgraphs/filters_basic.grc



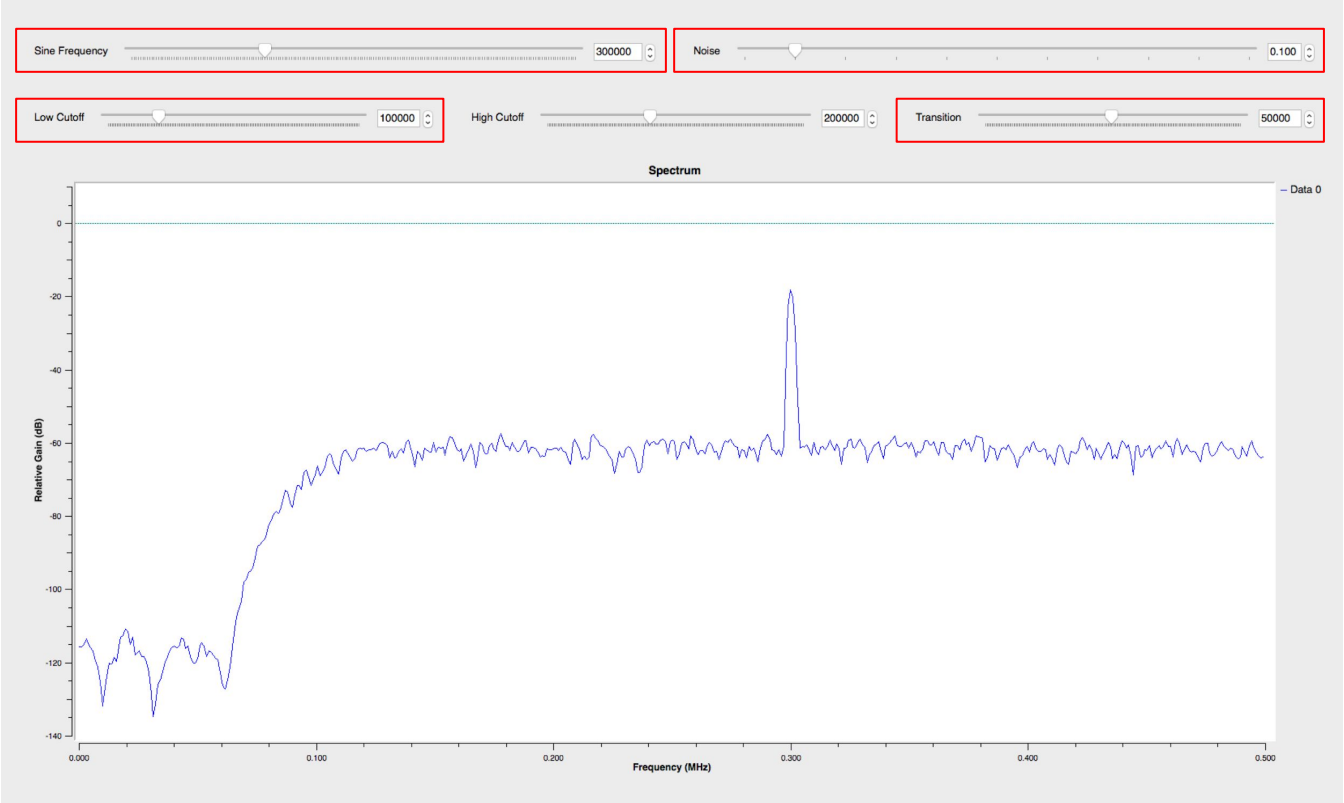
Example: Filters - Low Pass

Location: ~/ettus_workshop/flowgraphs/filters.grc



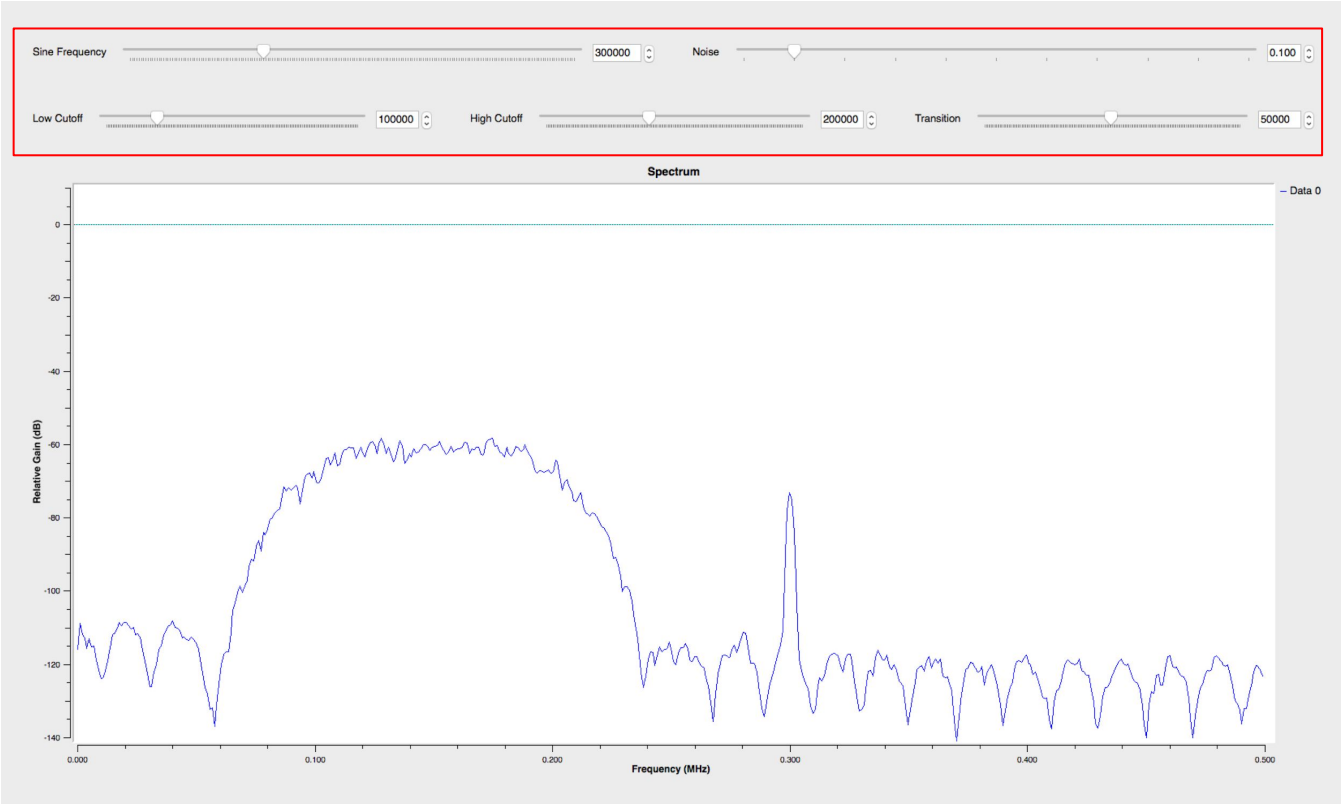
Example: Filters - High Pass

Location: ~/ettus_workshop/flowgraphs/filters.grc



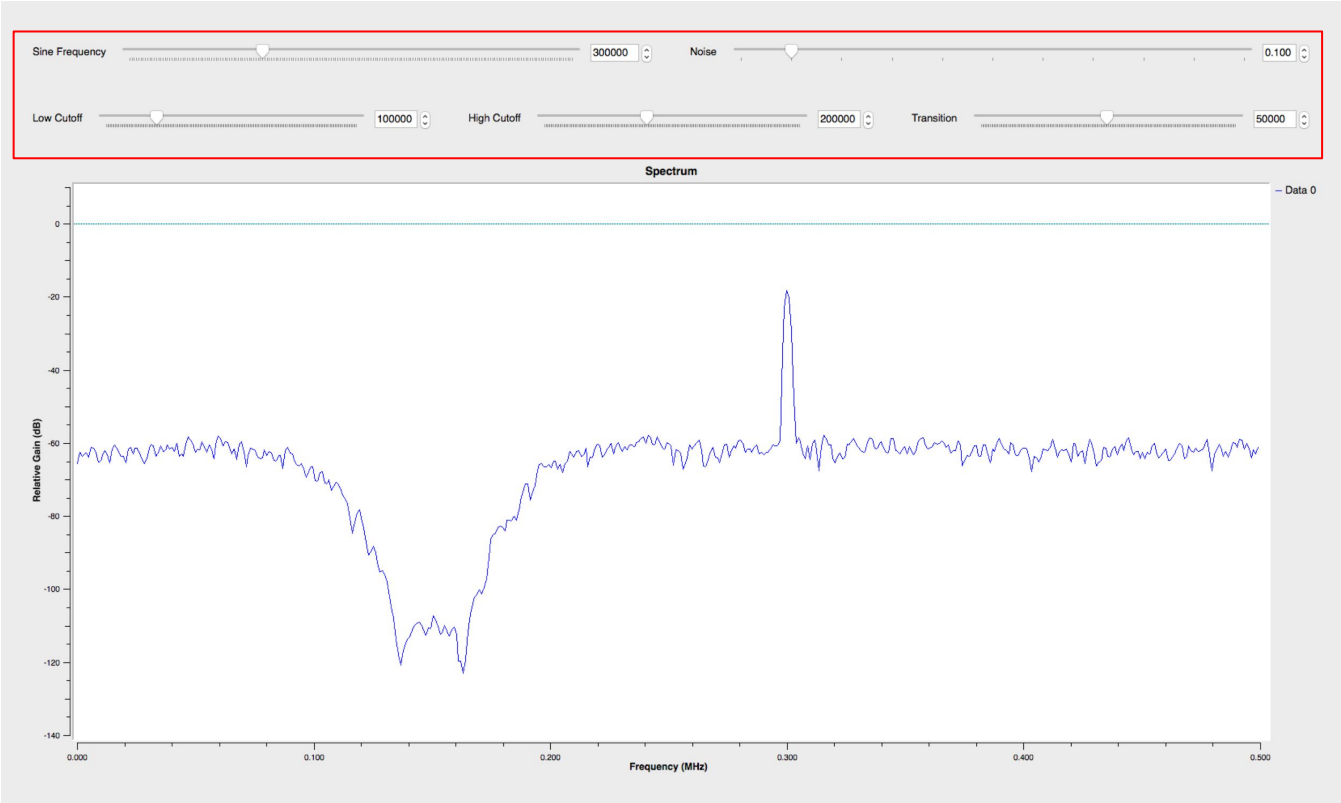
Example: Filters - Band Pass

Location: ~/ettus_workshop/flowgraphs/filters.grc



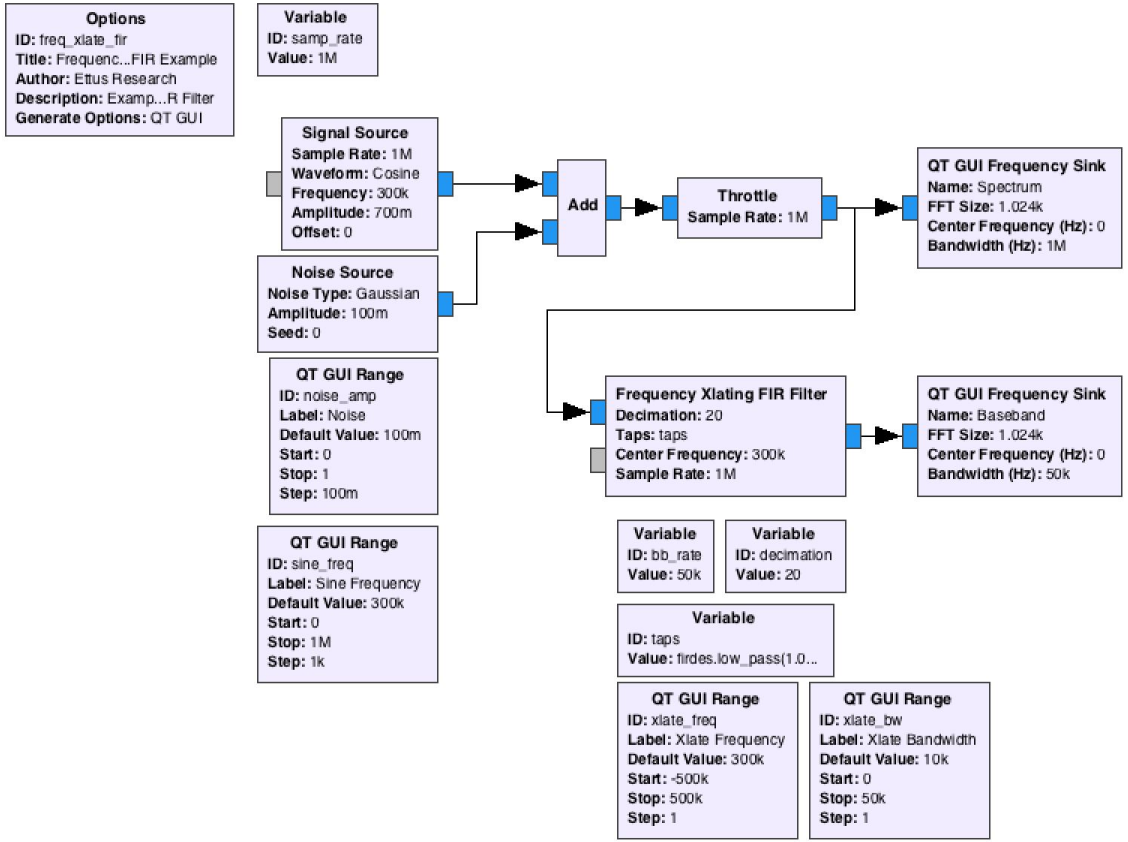
Example: Filters - Band Reject

Location: ~/ettus_workshop/flowgraphs/filters.grc



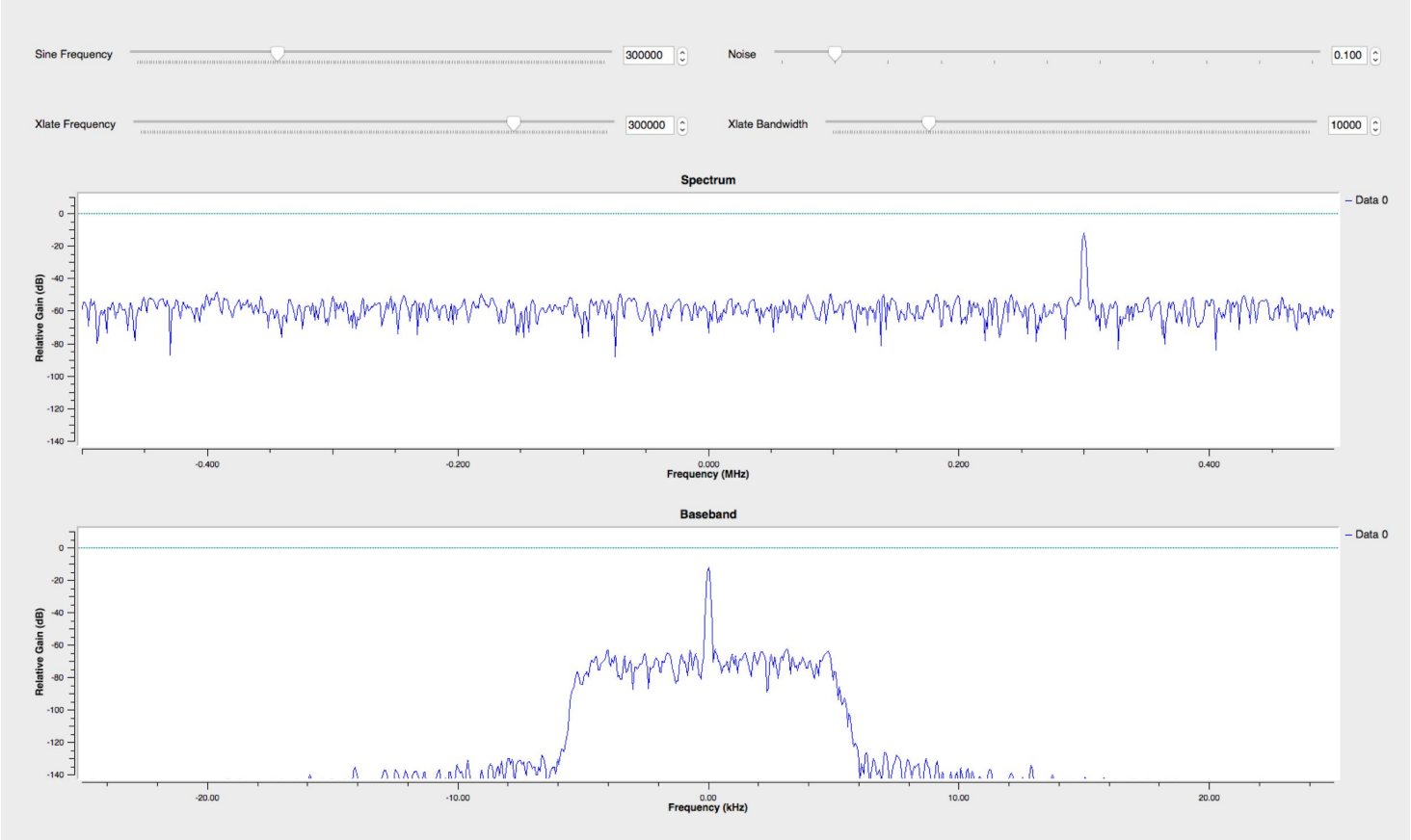
Example: Frequency Xlating FIR Filter

Location: ~/ettus_workshop/flowgraphs/freq_xlate_fir.grc



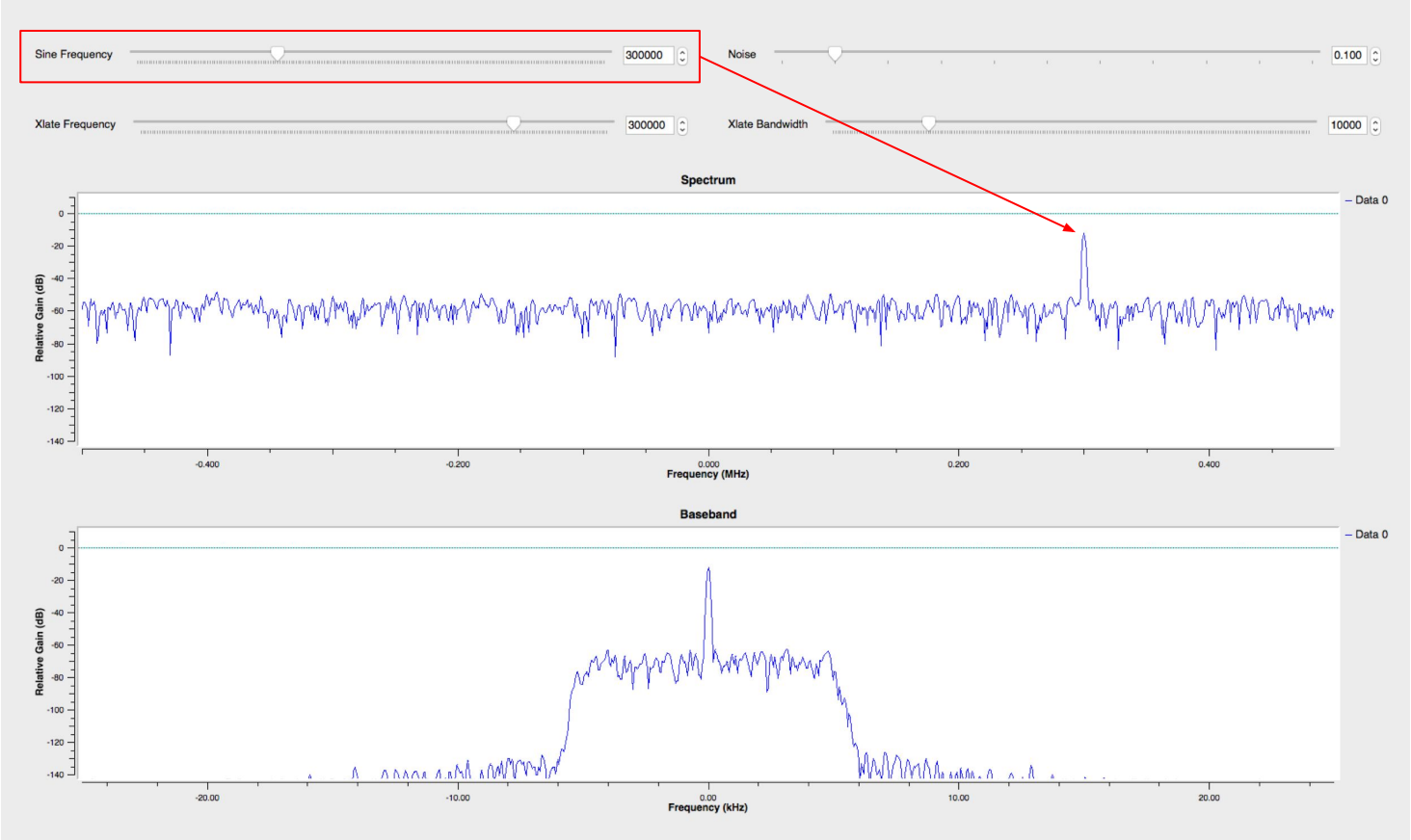
Example: Frequency Xlating FIR Filter

Location: ~/ettus_workshop/flowgraphs/freq_xlate_fir.grc



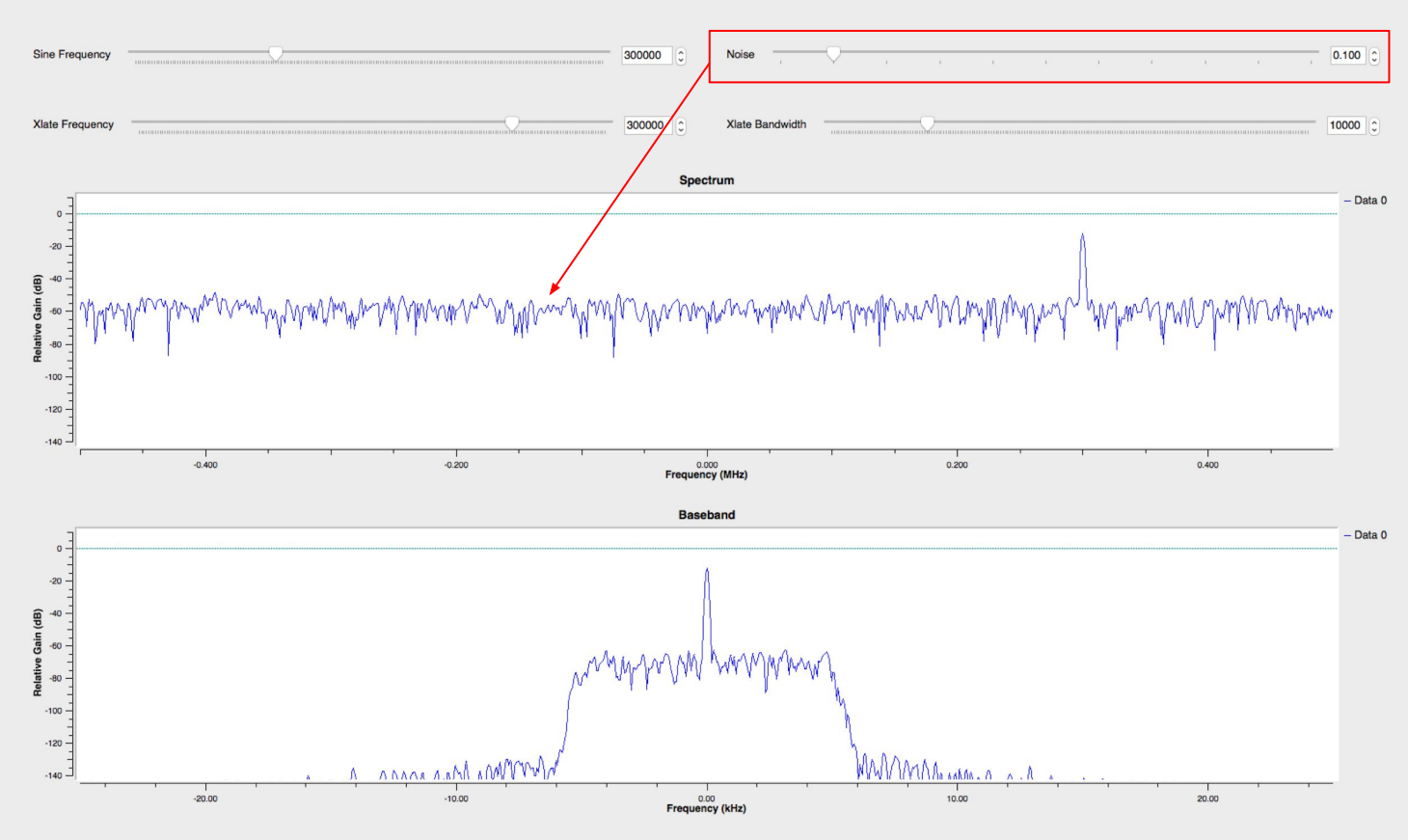
Example: Frequency Xlating FIR Filter

Location: ~/ettus_workshop/flowgraphs/freq_xlate_fir.grc



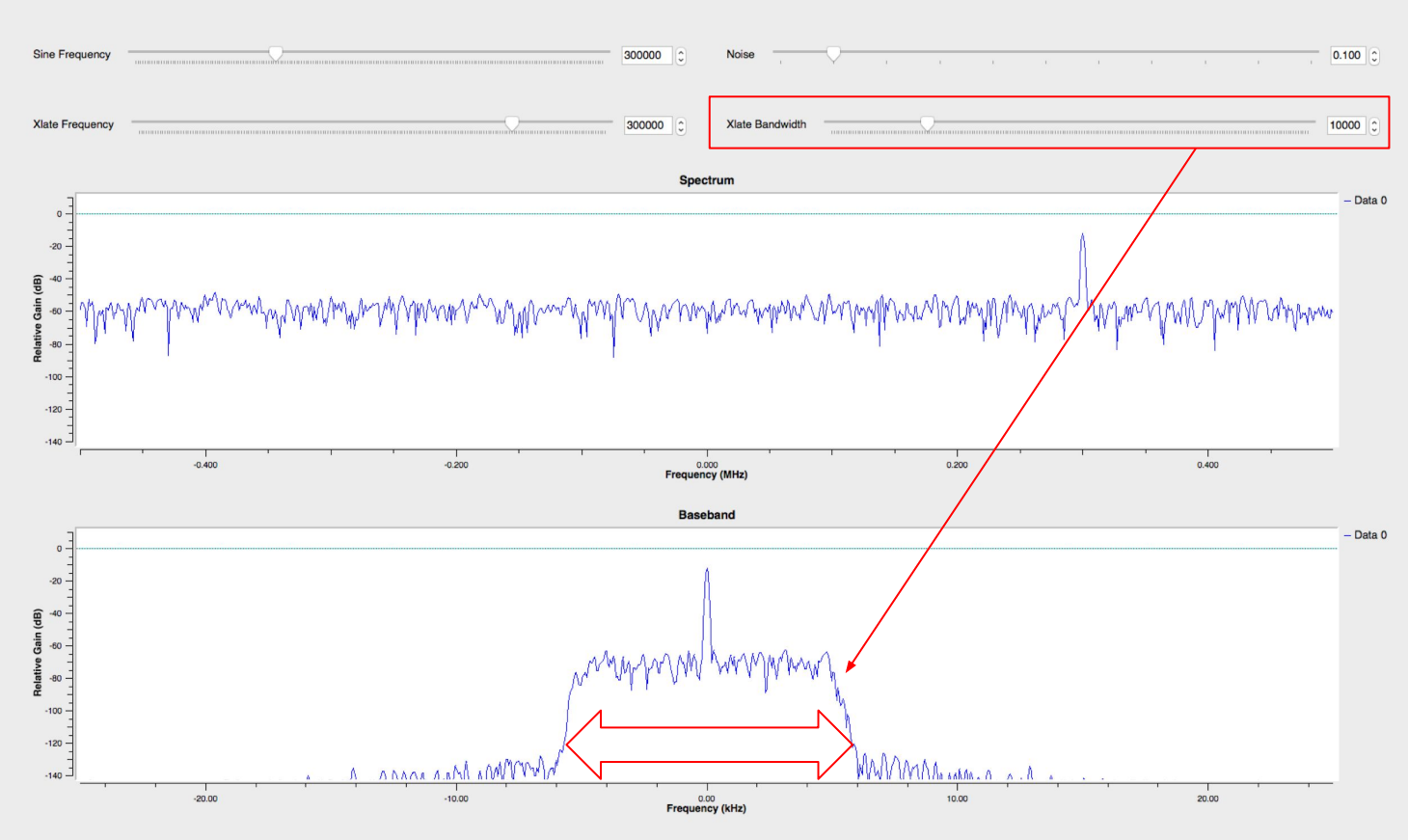
Example: Frequency Xlating FIR Filter

Location: ~/ettus_workshop/flowgraphs/freq_xlate_fir.grc



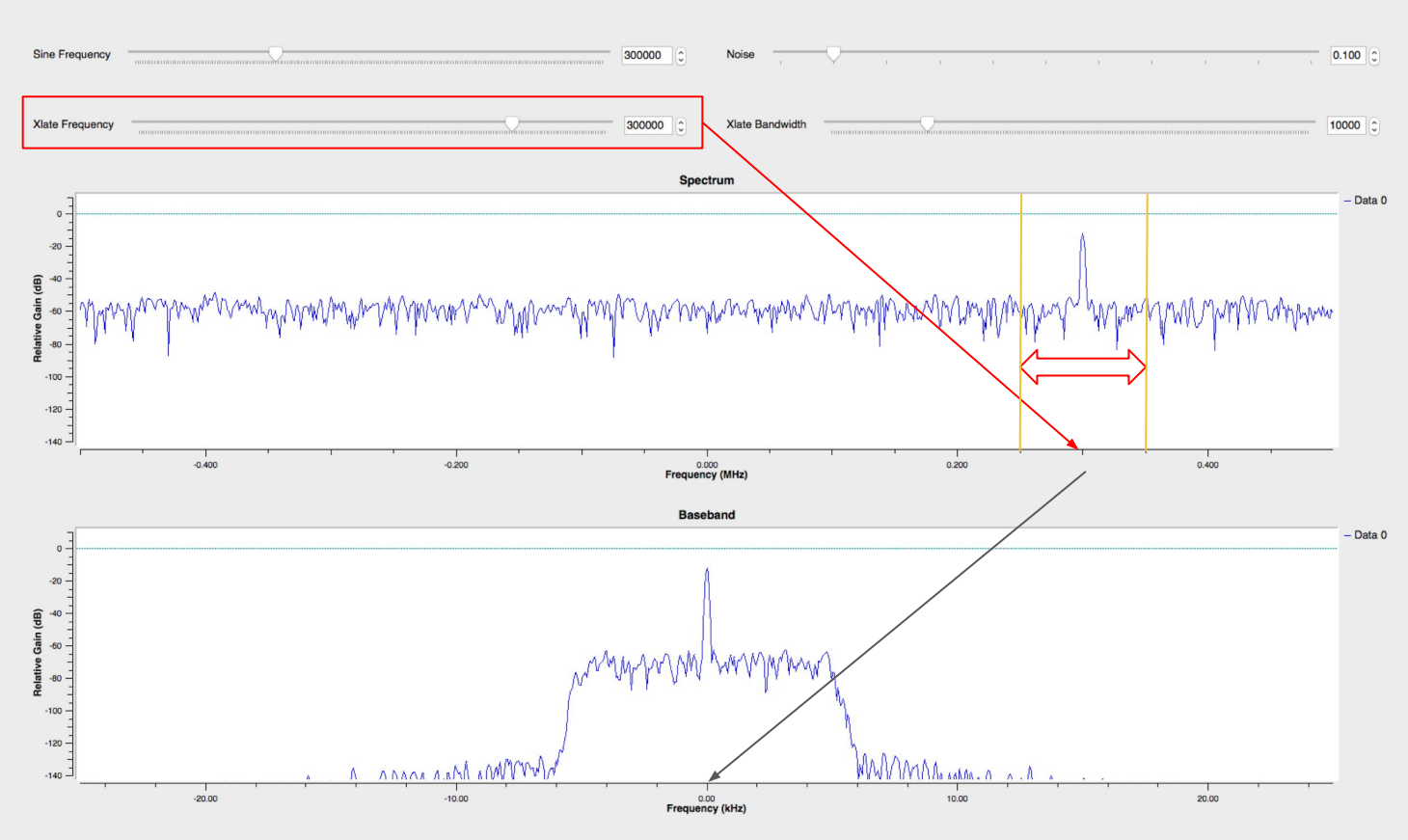
Example: Frequency Xlating FIR Filter

Location: ~/ettus_workshop/flowgraphs/freq_xlate_fir.grc



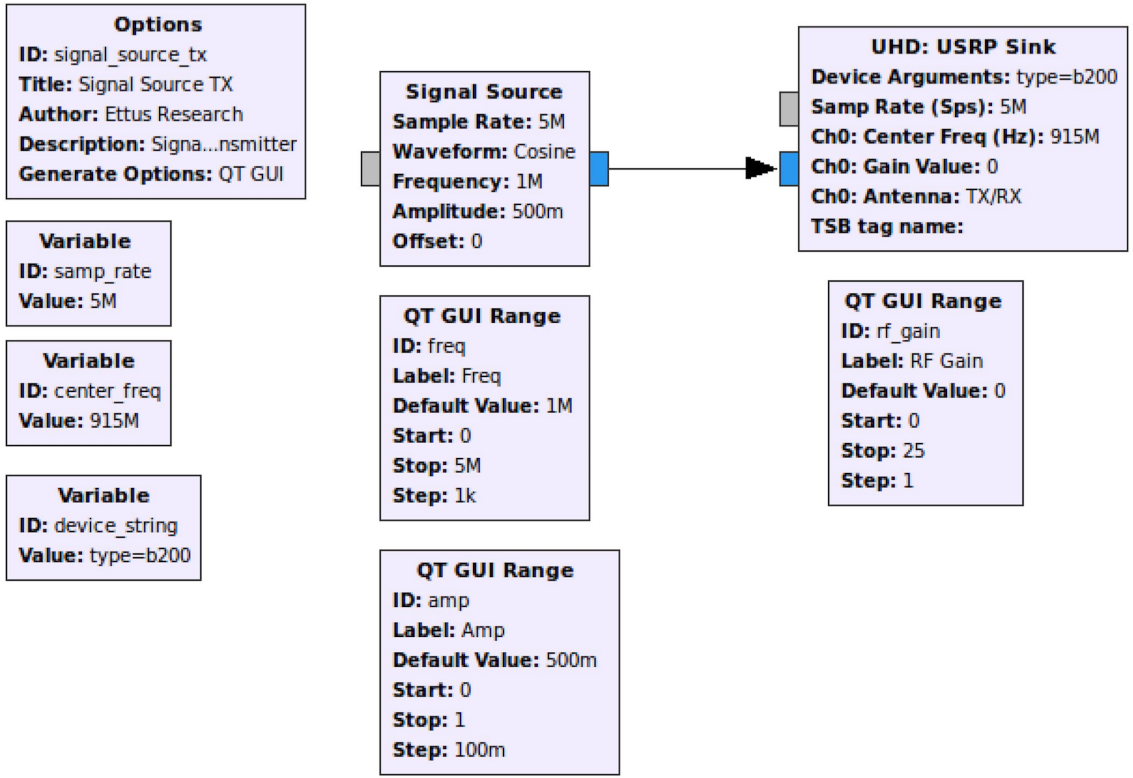
Example: Frequency Xlating FIR Filter

Location: ~/ettus_workshop/flowgraphs/freq_xlate_fir.grc



Example: Transmitting Signal Source

Location: ~/ettus_workshop/flowgraphs/signal_source_tx.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

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`

GNU Radio: Creating a Block / OOT

- Blocks can be written with Python or C++
- `gr_modtool` is used to create all of the boilerplate files
- Python or C++ QA
- For blocks with strict types, we use suffixes to declare the input and output types. This block operates on floats, so we give it the suffix _ff: Float in, float out. Other suffixes are _cc (complex in, complex out), or simply _f (a sink or source with no in- or outputs that uses floats).

- Standard Block Types:
  - **Synchronous** Blocks (1:1)
  - **Decimation** Blocks (N:1)
  - **Interpolation** Blocks (1:M)
  - **General** Blocks (N:M)
- Hierarchical Blocks
  - Hierarchical blocks are blocks that are made up of other blocks. They instantiate the other GNU Radio blocks (or other hierarchical blocks) and connect them together. A hierarchical block has a “connect” function for this purpose. Hierarchical blocks define an input and output stream much like normal blocks.
  -

More details see: https://wiki.gnuradio.org/index.php/BlocksCodingGuide

GNU Radio: Creating a Block

```
$ gr_modtool help
Usage:
gr_modtool <command> [options] -- Run <command> with the given options.
gr_modtool help -- Show a list of commands.
gr_modtool help <command> -- Shows the help for a given command.

List of possible commands:

Name      Aliases          Description
=====================================================================
disable   dis              Disable block (comments out CMake entries for files)
info      getinfo,inf      Return information about a given module
remove    rm,del           Remove block (delete files and remove Makefile entries)
makexml   mx               Make XML file for GRC block bindings
add       insert           Add block to the out-of-tree module.
newmod    nm,create        Create a new out-of-tree module
rename    mv               Rename a block in the out-of-tree module.
```

GNU Radio: Creating a Block

```
$ gr_modtool help newmod
Usage: gr_modtool nm [options].
 Call gr_modtool without any options to run it interactively.

Options:
  General options:
    -h, --help          Displays this help message.
    -d DIRECTORY, --directory=DIRECTORY
                        Base directory of the module. Defaults to the cwd.
    -n MODULE_NAME, --module-name=MODULE_NAME
                        Use this to override the current module's name (is
                        normally autodetected).
    -N BLOCK_NAME, --block-name=BLOCK_NAME
                        Name of the block, where applicable.
    --skip-lib          Don't do anything in the lib/ subdirectory.
    --skip-swig         Don't do anything in the swig/ subdirectory.
    --skip-python       Don't do anything in the python/ subdirectory.
    --skip-grc          Don't do anything in the grc/ subdirectory.
    --scm-mode=SCM_MODE
                        Use source control management (yes, no or auto).
    -y, --yes           Answer all questions with 'yes'. This can overwrite
                        and delete your files, so be careful.

  New out-of-tree module options:
    --srcdir=SRCDIR     Source directory for the module template.
```

GNU Radio: Creating a Block

Create your first Out-Of-Tree Module with the gr_modtool:

```
$ gr_modtool newmod workshop
Creating out-of-tree module in ./ gr-workshop... Done.
Use 'gr_modtool add' to add a new block to this currently empty module.

$ ls
gr-workshop

$ cd gr-workshop/
$ ls
CMakeLists.txt   MANIFEST.md  apps     cmake        docs     examples grc
    include      lib      python       swig
```

GNU Radio: Creating a Block

Since we are dealing with Python in this tutorial we only need to concern ourselves with the **`Python`** folder and the **`grc`** folder.

```
$ gr_modtool add -t sync -l python
GNU Radio module name identified: workshop
Language: Python
Enter name of block/code (without module name prefix):  my_multiply_py_ff
Block/code identifier: my_multiply_py_ff
Enter valid argument list, including default arguments:  multiple
Add Python QA code? [Y/n]  y
Adding file 'python/my_multiply_py_ff.py'...
Adding file 'python/qa_my_multiply_py_ff.py'...
Editing python/CMakeLists.txt...
Adding file 'grc/workshop_my_multiply_py_ff.xml'...
Editing grc/CMakeLists.txt...
```

GNU Radio: Creating a Block

First let's take a look at the file: **`python/my_multiply_py_ff.py`**

```
21
22  import numpy
23  from gnuradio import gr
24
25  class my_multiply_py_ff(gr.sync_block):
26      """
27      docstring for block my_multiply_py_ff
28      """
29      def __init__(self, multiple):
30          gr.sync_block.__init__(self,
31              name="my_multiply_py_ff",
32              in_sig=[<+numpy.float+>],
33              out_sig=[<+numpy.float+>])
34
35
36      def work(self, input_items, output_items):
37          in0 = input_items[0]
38          out = output_items[0]
39          # <+signal processing here+>
40          out[:] = in0
41          return len(output_items[0])
42
43
```

Notice that there are "<...>" scattered in many places. These placeholders are from **`gr_modtool`** and tell us where we need to alter things.

The **gr.sync_block.init** takes in 4 inputs: self, name, and the size/type of the input and output vectors. First, we want to make the item size a single precision float or numpy.float32 by removing the "<" and the ">".

GNU Radio: Creating a Block

python/my_multiply_py_ff.py

```python

import numpy
from gnuradio import gr

class my_multiply_py_ff(gr.sync_block):
    """
    docstring for block my_multiply_py_ff
    """
    def __init__(self, multiple):
        gr.sync_block.__init__(self,
            name="my_multiply_py_ff",
            in_sig=[numpy.float32],
            out_sig=[numpy.float32])


    def work(self, input_items, output_items):
        in0 = input_items[0]
        out = output_items[0]
        # <+signal processing here+>
        out[:] = in0
        return len(output_items[0])


```

The other piece of code that has the placeholders is in the **`work()`** function but let us first get a better understanding of the **`work()`** function.

The **`work()`** function is where the actual processing happens, where we want our code to be. Because this is a **`sync`** block, the number of input items always equals the number of output items because synchronous block ensures a fixed output to input rate. There are also decimation and interpolation blocks where the number of output items are a user specified multiple of the number of input items.

GNU Radio: Creating a Block

python/my_multiply_py_ff.py

```
21
22  import numpy
23  from gnuradio import gr
24
25  class my_multiply_py_ff(gr.sync_block):
26      """
27      docstring for block my_multiply_py_ff
28      """
29      def __init__(self, multiple):
30          gr.sync_block.__init__(self,
31              name="my_multiply_py_ff",
32              in_sig=[numpy.float32],
33              out_sig=[numpy.float32])
34
35
36      def work(self, input_items, output_items):
37          in0 = input_items[0]
38          out = output_items[0]
39          # <+signal processing here+>
40          out[:] = in0
41          return len(output_items[0])
42
43
```

The **"in0"** and **"out"** simply store the input and output in a variable to make the block easier to write.

The signal processing can be anything including if statements, loops, function calls but for this example we only need to modify the **out[:] = in0** line so that our input signal is multiplied by our variable multiple.

What do we need to add to make the **in0** multiply by our **multiple**?

out[:] = in0*self.multiple

GNU Radio: Creating a Block

python/my_multiply_py_ff.py

```
21
22  import numpy
23  from gnuradio import gr
24
25  class my_multiply_py_ff(gr.sync_block):
26      """
27      docstring for block my_multiply_py_ff
28      """
29      def __init__(self, multiple):
30          gr.sync_block.__init__(self,
31              name="my_multiply_py_ff",
32              in_sig=[numpy.float32],
33              out_sig=[numpy.float32])
34
35      def work(self, input_items, output_items):
36          in0 = input_items[0]
37          out = output_items[0]
38          # <+signal processing here+>
39          out[:] = in0*self.multiple
40          return len(output_items[0])
41
42
```

The last item to modify within the new block is to assign the input value of "**multiple**" to **self.multiple** within the **init()** function.

self.multiple = multiple

```
29      def __init__(self, multiple):
30          gr.sync_block.__init__(self,
31              name="my_multiply_py_ff",
32              in_sig=[numpy.float32],
33              out_sig=[numpy.float32])
34          self.multiple = multiple
```

GNU Radio: Creating a Block

Completed **`python/my_multiply_py_ff.py`**

```
22  import numpy
23  from gnuradio import gr
24
25  class my_multiply_py_ff(gr.sync_block):
26      """
27      docstring for block my_multiply_py_ff
28      """
29      def __init__(self, multiple):
30          gr.sync_block.__init__(self,
31              name="my_multiply_py_ff",
32              in_sig=[numpy.float32],
33              out_sig=[numpy.float32])
34          self.multiple = multiple
35
36      def work(self, input_items, output_items):
37          in0 = input_items[0]
38          out = output_items[0]
39          # <+signal processing here+>
40          out[:] = in0*self.multiple
41          return len(output_items[0])
42
43
```

GNU Radio: Creating a Block

Ettus Research™
A National Instruments Company

Next, we will create the QA (Quality Assurance) tests.

Open the file: **`python/qa_my_multiply_py_ff.py`**

```
21
22  from gnuradio import gr, gr_unittest
23  from gnuradio import blocks
24  from my_multiply_py_ff import my_multiply_py_ff
25
26  class qa_my_multiply_py_ff (gr_unittest.TestCase):
27
28      def setUp (self):
29          self.tb = gr.top_block ()
30
31      def tearDown (self):
32          self.tb = None
33
34      def test_001_t (self):
35          # set up fg
36          self.tb.run ()
37          # check data
38
39
40  if __name__ == '__main__':
41      gr_unittest.run(qa_my_multiply_py_ff, "qa_my_multiply_py_ff.xml")
42
```

`gr_unittest` adds support for checking approximate equality of tuples of float and complex numbers. The only part we need to worry about is the def **`test_001_t`** function. We know we need input data so let us create data. We want it to be in the form of a vector so that we can test multiple values at once.

GNU Radio: Creating a Block

`python/qa_my_multiply_py_ff.py`

Create a vector of floats within the **`test_001_t`** function:

```
src_data = (0, 1, -2, 5.5, -0.5)
```

We also need output data so we can compare the input of the block to ensure that it is doing what we expect it to do. Let us multiply by 2 for simplicity.

```
expected_result = (0, 2, -4, 11, -1)
```

Next, create a flowgraph with the **`src_data`**:

```
src = blocks.vector_source_f(src_data)
```

GNU Radio: Creating a Block

```
python/qa_my_multiply_py_ff.py
```

Next, we will call our created function with a value of 2.

```
mult = multiply_py_ff(2)
```

Next, we will create a Vector Sink block.

```
snk = blocks.vector_sink_f()
```

Next, connect the blocks:

```
self.tb.connect (src, mult)
self.tb.connect (mult, snk)
```

Next, run the flowgraph and capture the resulting data from the Sink.

```
self.tb.run ()
result_data = snk.data ()
```

GNU Radio: Creating a Block

python/qa_my_multiply_py_ff.py

The last step is to compare the expected result and actual result. The last value, “6” in the function call below will compare the results to decimal places

self.assertFloatTuplesAlmostEqual (expected_result, result_data, 6)

The completed **test_001_t()** function should match below:

```
34      def test_001_t (self):
35          src_data = (0, 1, -2, 5.5, -0.5)
36          expected_result = (0, 2, -4, 11, -1)
37          src = blocks.vector_source_f(src_data)
38          mult = my_multiply_py_ff(2)
39          snk = blocks.vector_sink_f()
40          self.tb.connect(src, mult)
41          self.tb.connect(mult, snk)
42          self.tb.run()
43          result_data = snk.data()
44          self.assertFloatTuplesAlmostEqual(expected_result, result_data, 6)
45
```

GNU Radio: Creating a Block

Next, we will test the QA code. Run it with:

```
$ python python/qa_my_multiply_py_ff.py
.
----------------------------------------------------------------------
Ran 1 test in 0.003s

OK
```

GNU Radio: Creating a Block

Open the file **`grc/workshop_my_multiply_py_ff.xml`**

```xml
<?xml version="1.0"?>
<block>
  <name>my_multiply_py_ff</name>
  <key>workshop_my_multiply_py_ff</key>
  <category>[workshop]</category>
  <import>import workshop</import>
  <make>workshop.my_multiply_py_ff($multiple)</make>
  <!-- Make one 'param' node for every Parameter you want settable from the GUI.
       Sub-nodes:
       * name
       * key (makes the value accessible as $keyname, e.g. in the make node)
       * type -->
  <param>
    <name>...</name>
    <key>...</key>
    <type>...</type>
  </param>

  <!-- Make one 'sink' node per input. Sub-nodes:
       * name (an identifier for the GUI)
       * type
       * vlen
       * optional (set to 1 for optional inputs) -->
  <sink>
    <name>in</name>
    <type><!-- e.g. int, float, complex, byte, short, xxx_vector, ...--></type>
  </sink>

  <!-- Make one 'source' node per output. Sub-nodes:
       * name (an identifier for the GUI)
       * type
       * vlen
       * optional (set to 1 for optional inputs) -->
  <source>
    <name>out</name>
    <type><!-- e.g. int, float, complex, byte, short, xxx_vector, ...--></type>
  </source>
</block>
```

The last step to create a block is to modify the XML file.

GRC uses the XML files for all the options we see.

GNU Radio: Creating a Block

Boilerplate / default: **`grc/workshop_my_multiply_py_ff.xml`**

```xml
<?xml version="1.0"?>
<block>
  <name>my_multiply_py_ff</name>
  <key>workshop_my_multiply_py_ff</key>
  <category>[workshop]</category>
  <import>import workshop</import>
  <make>workshop.my_multiply_py_ff($multiple)</make>
  <!-- Make one 'param' node for every Parameter you want settable from the GUI.
       Sub-nodes:
       * name
       * key (makes the value accessible as $keyname, e.g. in the make node)
       * type -->
  <param>
    <name>...</name>
    <key>...</key>
    <type>...</type>
  </param>

  <!-- Make one 'sink' node per input. Sub-nodes:
       * name (an identifier for the GUI)
       * type
       * vlen
       * optional (set to 1 for optional inputs) -->
  <sink>
    <name>in</name>
    <type><!-- e.g. int, float, complex, byte, short, xxx_vector, ...--></type>
  </sink>

  <!-- Make one 'source' node per output. Sub-nodes:
       * name (an identifier for the GUI)
       * type
       * vlen
       * optional (set to 1 for optional inputs) -->
  <source>
    <name>out</name>
    <type><!-- e.g. int, float, complex, byte, short, xxx_vector, ...--></type>
  </source>
</block>
```

GNU Radio: Creating a Block

File: **`grc/workshop_my_multiply_py_ff.xml`**

We can change the name that appears and the category it will appear in GRC. The category is where the block will be found in GRC. Examples of categories tag are Audio and Waveform Generators used in previous examples. Examples of names tag are the QT GUI Time Sink or the Audio Sink.

We will leave the default values for this section.

```
3   <name>my_multiply_py_ff</name>
4   <key>workshop_my_multiply_py_ff</key>
5   <category>[workshop]</category>
6   <import>import workshop</import>
7   <make>workshop.my_multiply_py_ff($multiple)</make>
```

GNU Radio: Creating a Block

File: **`grc/workshop_my_multiply_py_ff.xml`**

```
8   <!-- Make one 'param' node for every Parameter you want settable from the GUI.
9        Sub-nodes:
10       * name
11       * key (makes the value accessible as $keyname, e.g. in the make node)
12       * type -->
13  <param>
14    <name>...</name>
15    <key>...</key>
16    <type>...</type>
17  </param>
18
```

This is referring to the parameter that we used in the very beginning when creating our block: the variable called **`"multiple"`**. Fill it in as shown below:

```
13  <param>
14    <name>Multiple</name>
15    <key>multiple</key>
16    <type>float</type>
17  </param>
```

GNU Radio: Creating a Block

File: **`grc/workshop_my_multiply_py_ff.xml`**

The next placeholder can be found in the sink and source tags. We can see that it is asking for a type so we can simply erase everything in the tag and replace it with **`"float"`** for both the **`source`** and the **`sink`** blocks. That should do it for this block. The best way to get more experience writing XML files is to look at the source code of previously made blocks such as the existing multiple block.

```
    <sink>
      <name>in</name>
      <type>float</type>
    </sink>

    <source>
      <name>out</name>
      <type>float</type>
    </source>
  </block>
```

GNU Radio: Creating a Block

Now we will install our block into GNU Radio Companion.

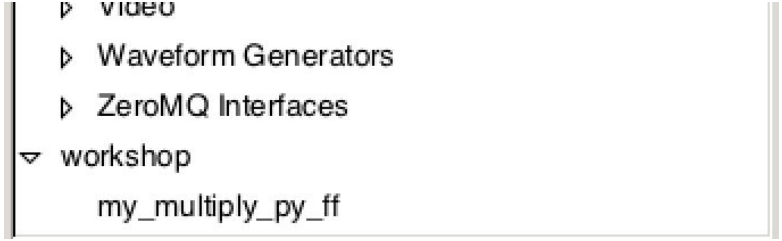
```
cd ~/workarea/gr-workshop
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

Now return to GNU Radio Companion and click the **"Reload Blocks"** button:

GNU Radio: Creating a Block

In the Block listing on the right side of GRC, at the very bottom, you should note a new category, **"workshop"**.

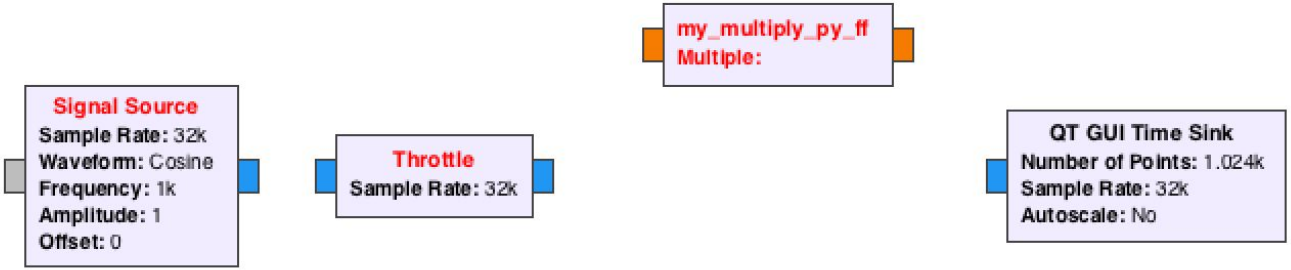
Clicking on **"workshop"**, will reveal our newly created block.



GNU Radio: Creating a Block

Next, we will create a flowgraph to demonstrate our newly created block.

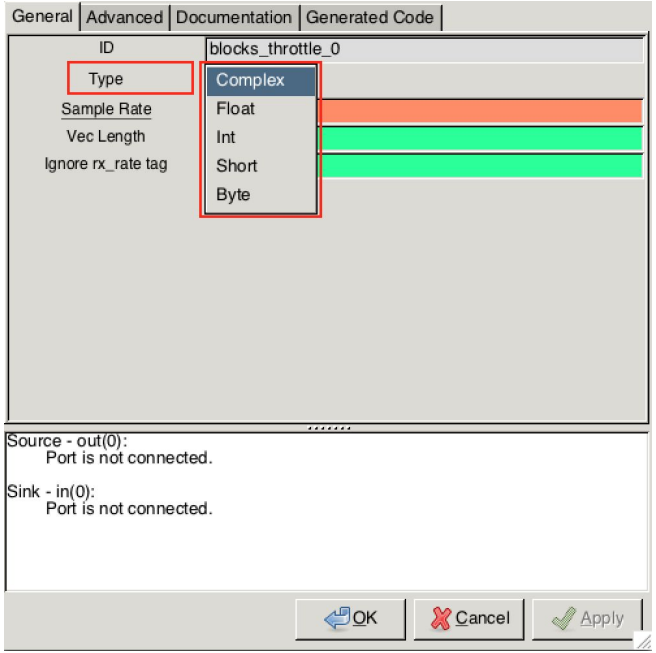
Start with dragging a **`Signal Source`**, **`Throttle`**, **`my_multiply_py_ff`**, and **`QT Time Sink`** block onto the canvas.



GNU Radio: Creating a Block

First, we need to adjust the **`Data Types`** of the blocks with the **`Blue`** (Complex) data types.

Select each block and press the **`"Down"`** arrow, which will change the **`Type`**. Modify each block so all inputs are **`Orange`** (Float). Data types can also be changed by double clicking on each block and adjusting the option within the pulldown menu.



GNU Radio: Creating a Block

Next, we need to add two inputs to the **`QT Time Sink`** block.

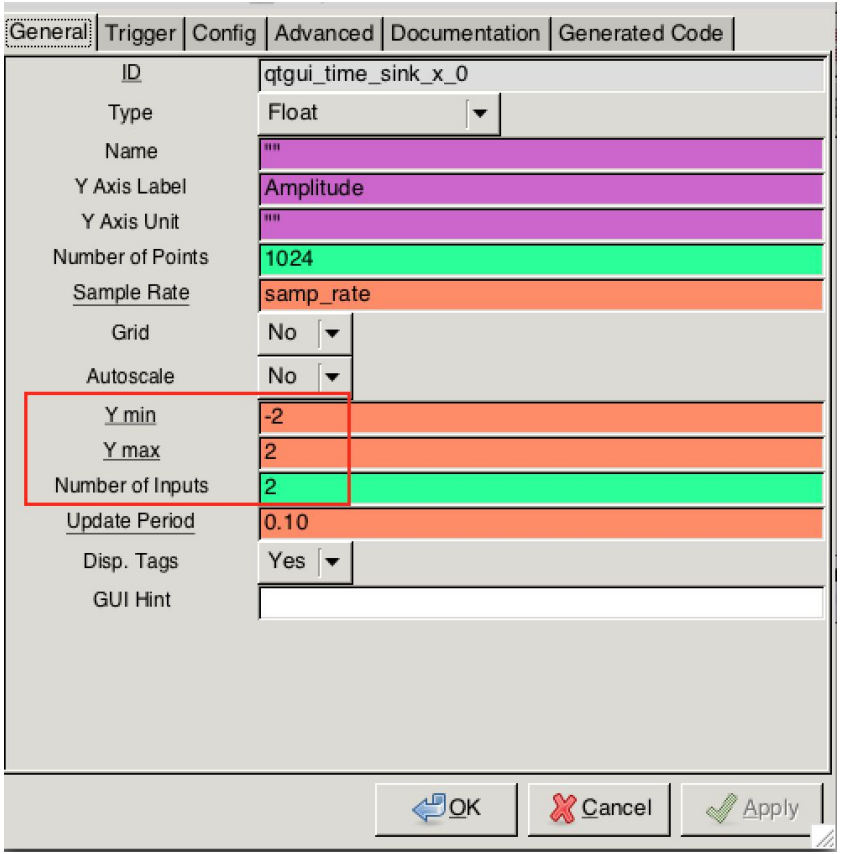
Double click on the **`QT Time Sink`** block to display its options.

Modify the field **"`Number of Inputs`"** to be a value of **"`2`"**.

Modify **"`Y min`"** to be **"`-2`"**
Modify **"`Y max`"** to be **"`2`"**

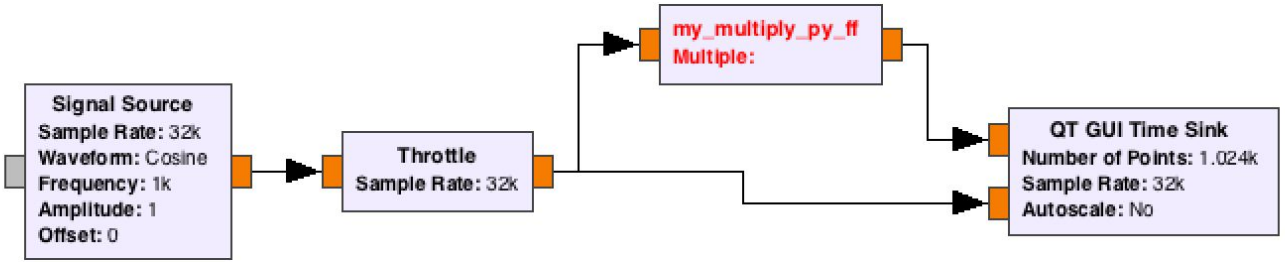
Under the **"`Config`"** tab,
adjust the value for **"`Control Panel`"** to **"`Yes`"**

Click **"`OK`"** to close the window.



GNU Radio: Creating a Block

Next, connect all of the blocks as shown below:



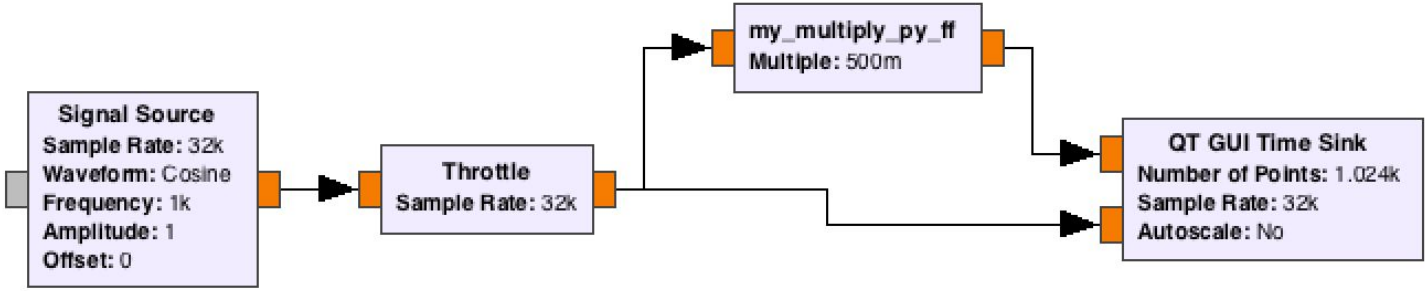
Note, our block "**my_multipl_py_ff**" is still highlighted Red. We need to add a value to it for our variable "**Multiple**".

Double click on the block "**my_multipl_py_ff**" and enter a value of "**0.5**" for "**Multiple**".

Click "**OK**" to close the Block Options window.

GNU Radio: Creating a Block

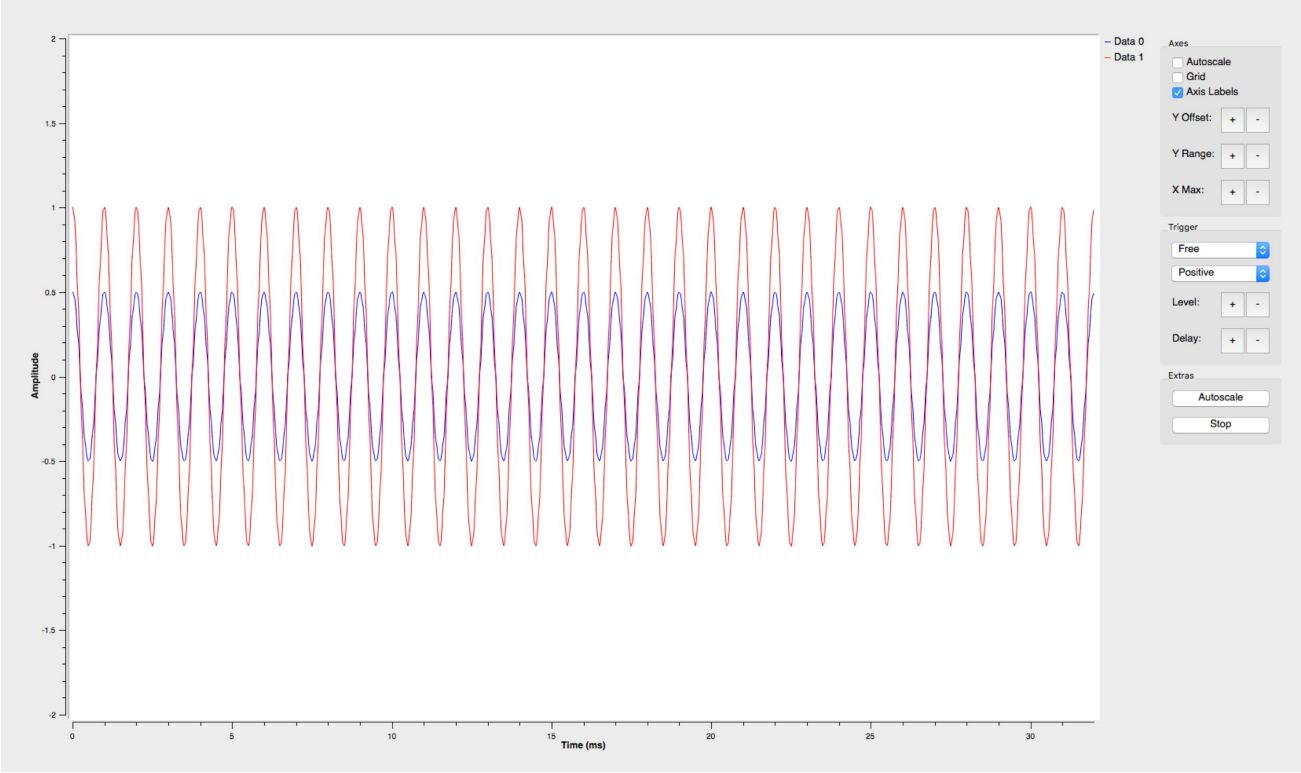
Your flowgraph should match as below:



GNU Radio: Creating a Block

Next, save the flowgraph from the top menu **"File" -> "Save As"**. Enter any filename.

Finally we will run the flowgraph. Click the **"Execute the flowgraph"** button in the menu.



Note: The Blue line is the samples which are being passed through our **"my_multiple_py_ff"** block, which are being multiplied by **0.5**, resulting in a signal 50% smaller than the original (Red) signal.

Try adjusting the **"Y Offset"**, **"Y Range"**, and **"X Max"** settings by clicking the **"+/-"** buttons within the **Control Panel** of the **QT Time Sink**.

GNU Radio: Creating a Block

Stop the flowgraph and try modifying the **`"Multiple"`** value within the newly created block.

Try values such as 0.1, 1.5, 2.0 or 3.0, and then run the flowgraph again.

GNU Radio: Creating a Block

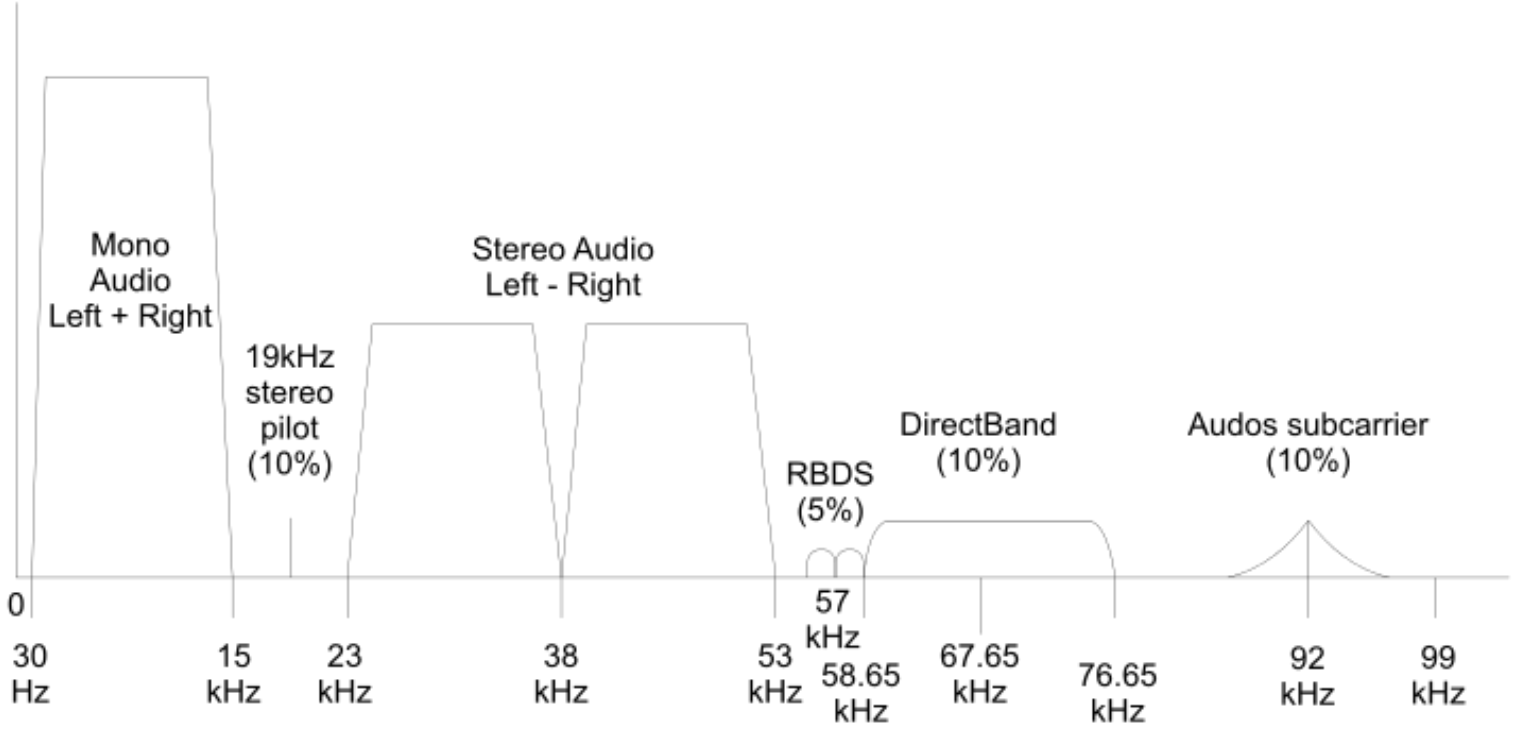
Official GNU Radio Creating a block in Python Tutorial
https://wiki.gnuradio.org/index.php/Guided_Tutorial_GNU_Radio_in_Python

Official GNU Radio Creating a block in C++ Tutorial
https://wiki.gnuradio.org/index.php/Guided_Tutorial_GNU_Radio_in_C%2B%2B

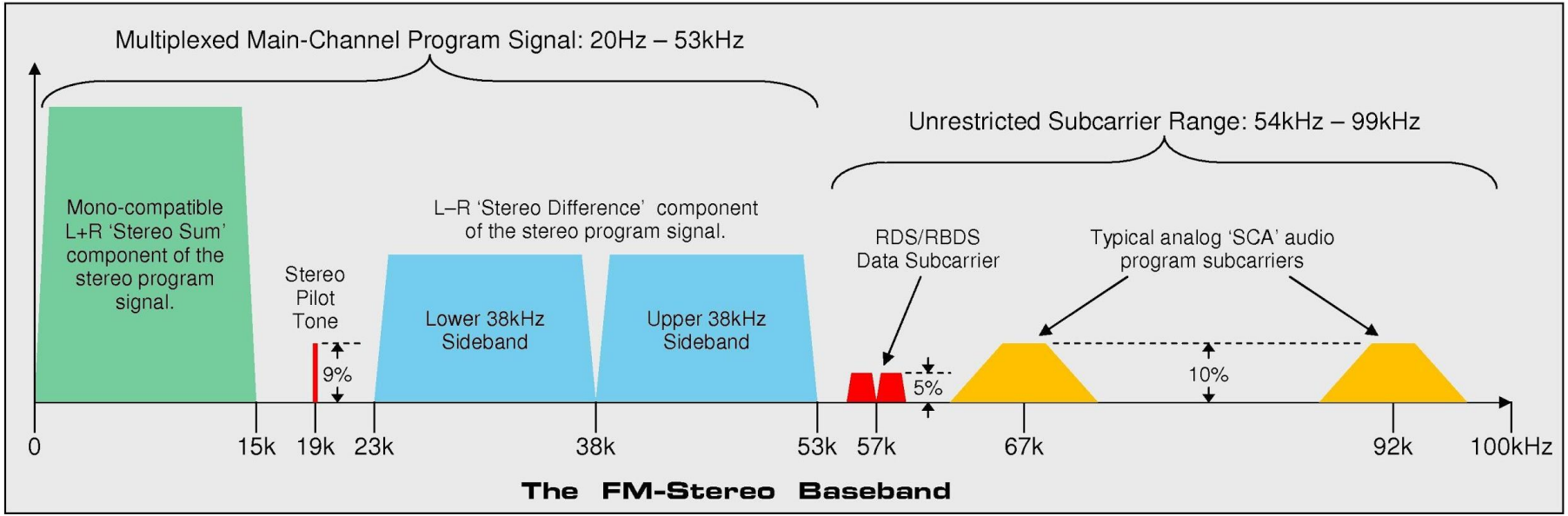
Full GNU Radio Tutorial (Tutorial links above are contained within this multi-part series)
https://wiki.gnuradio.org/index.php/Guided_Tutorial_Introduction

GNU Radio Block Coding Guide
https://wiki.gnuradio.org/index.php/BlocksCodingGuide

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 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.

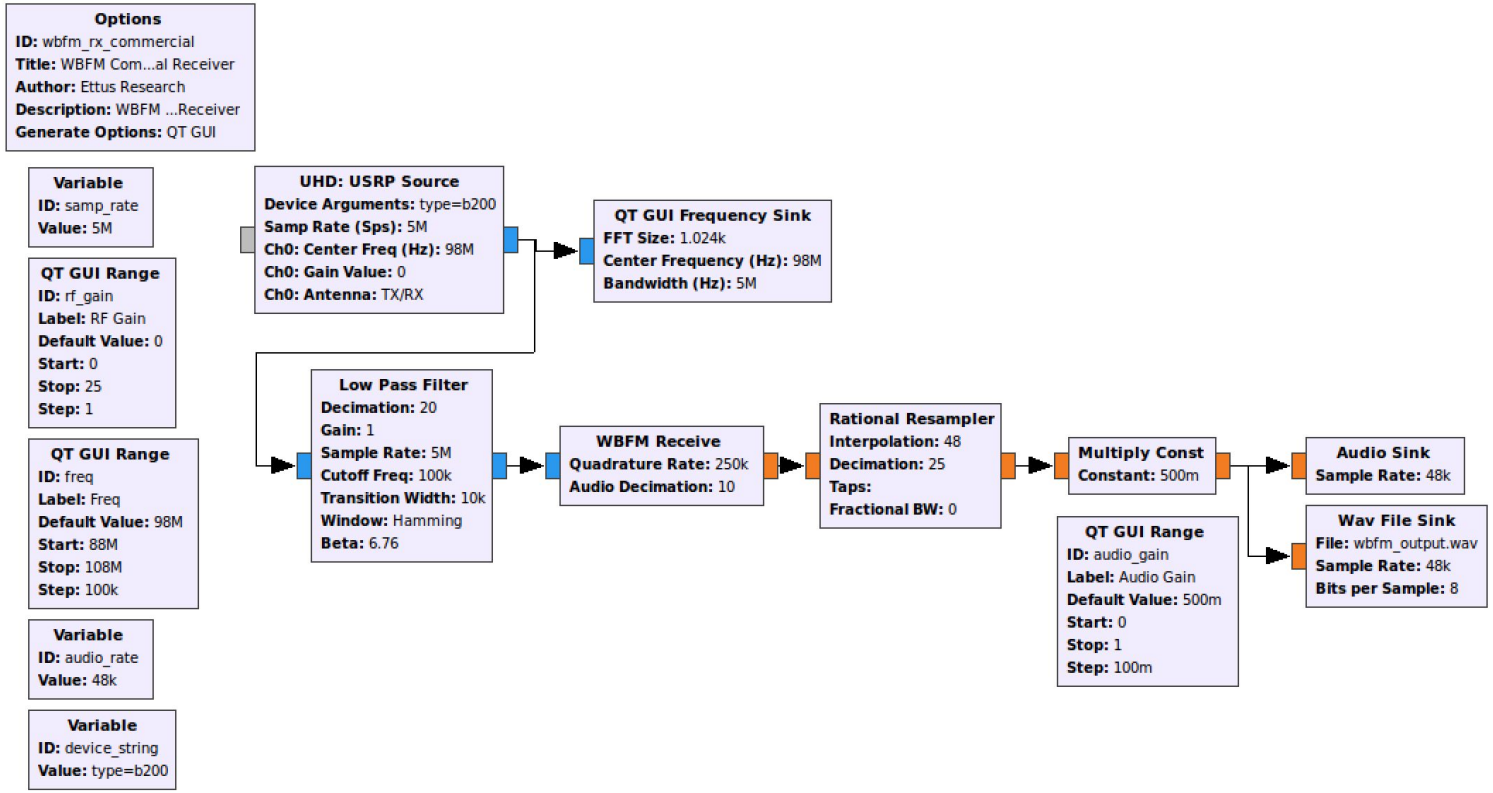
- **RT** (radio text) -- This function allows a radio station to transmit a 64-character free-form text that can be either static (such as station slogans) or in sync with the programming (such as the title and artist of the currently playing song).

- **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.

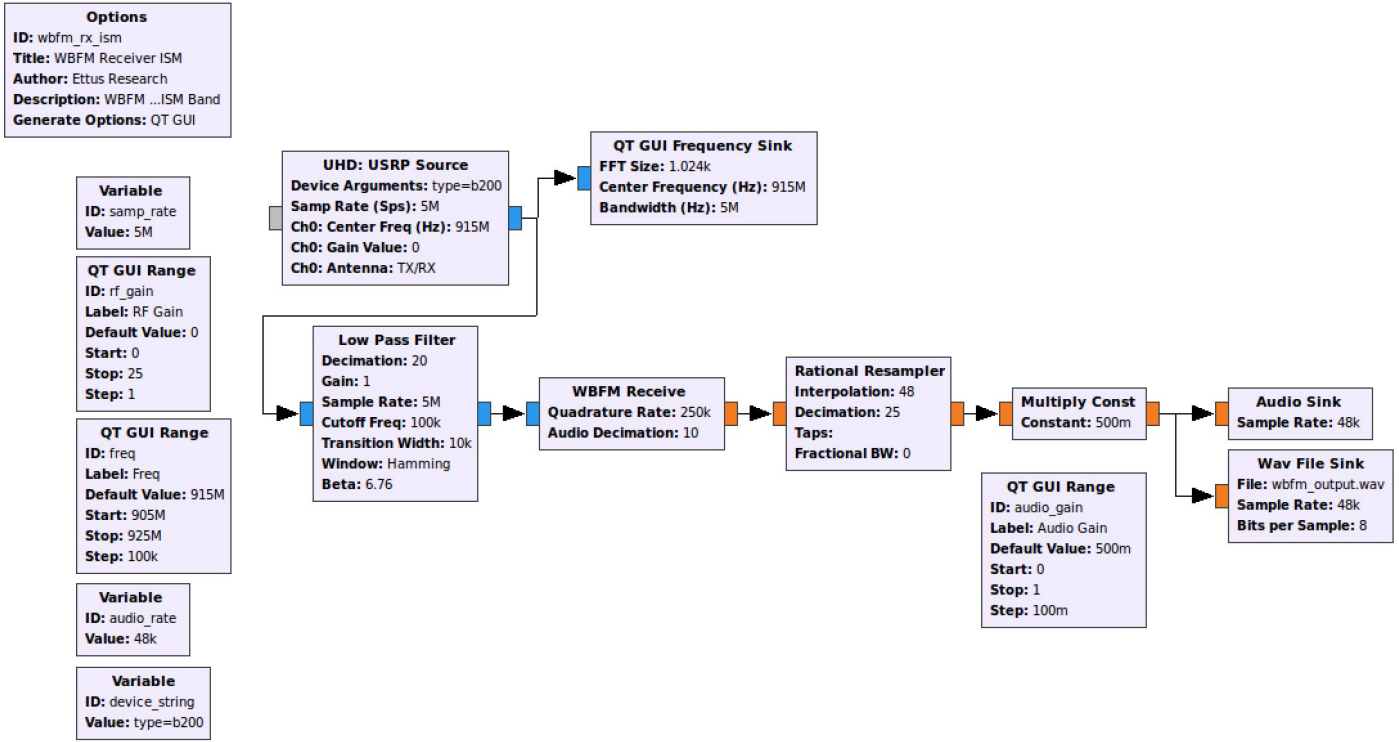
FM Receiver in GRC

Location: ~/ettus_workshop/flowgraphs/wbfm_rx_commercial.grc



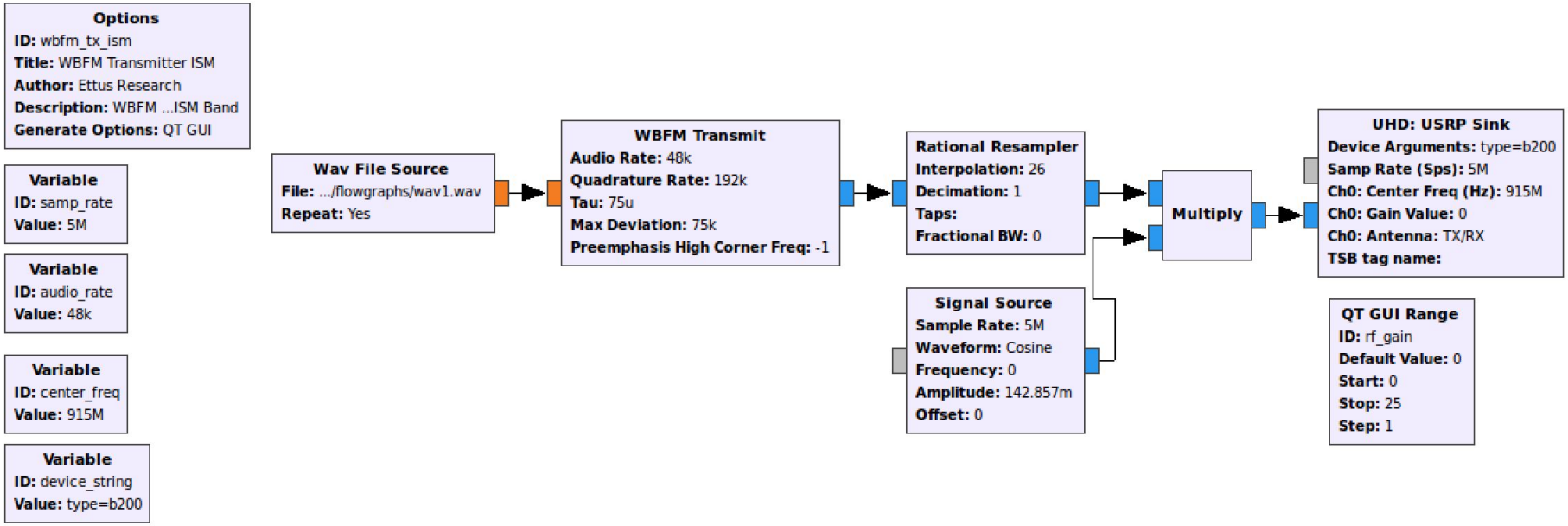
FM Receiver in GRC (ISM)

Location: ~/ettus_workshop/flowgraphs/wbfm_rx_ism.grc



FM Transmitter in GRC (ISM)

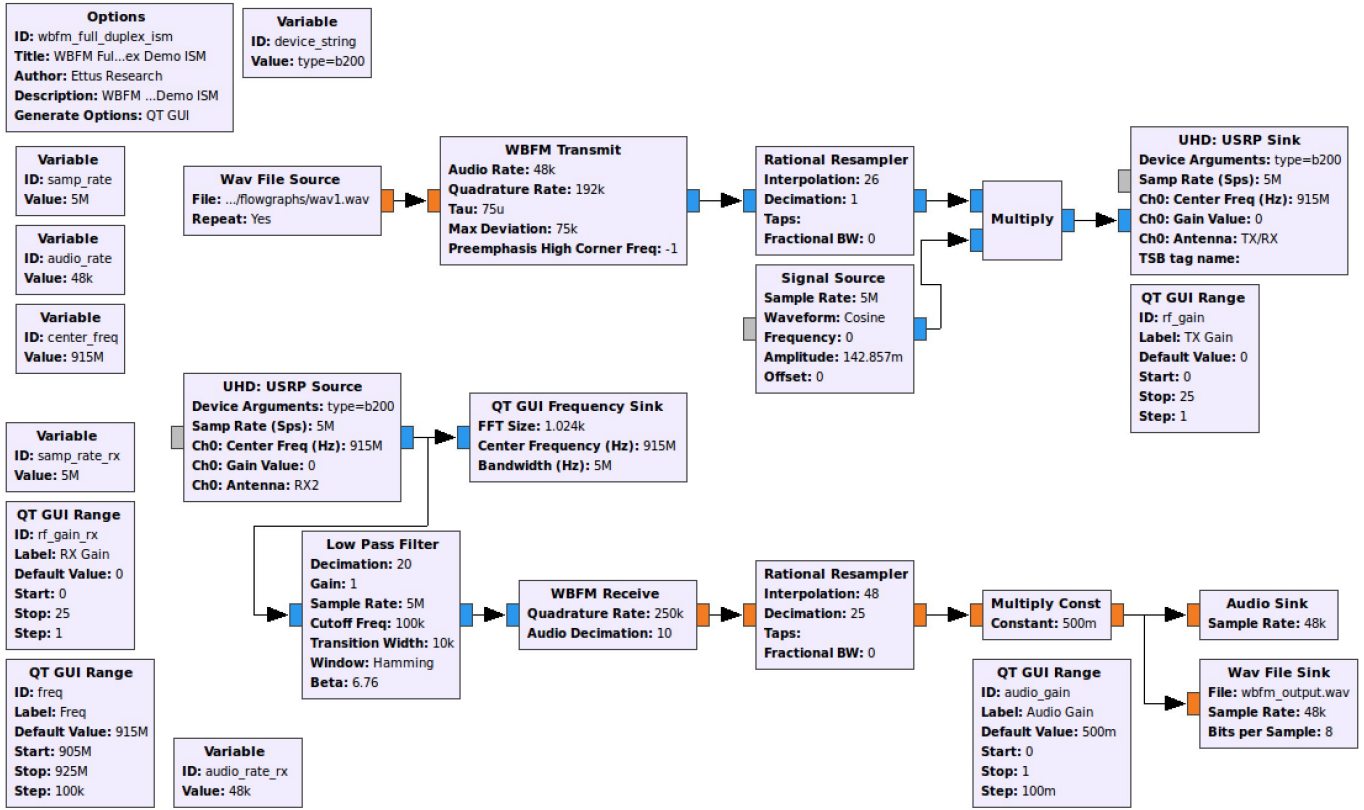
Location: ~/ettus_workshop/flowgraphs/wbfm_tx_ism.grc



FM TX/RX Full Duplex in GRC (ISM)

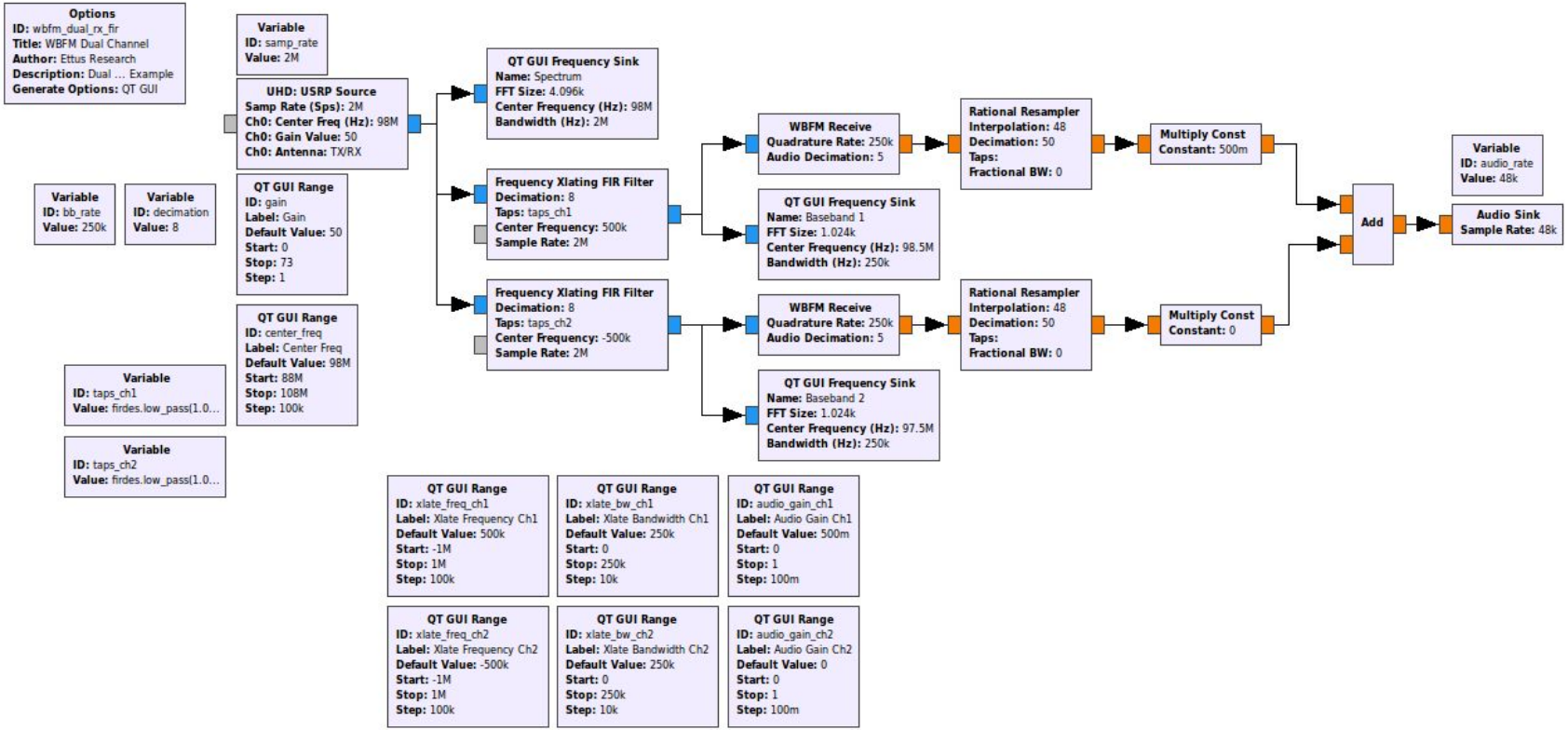
Ettus Research™
A National Instruments Company

Location: ~/ettus_workshop/flowgraphs/wbfm_full_duplex_ism.grc



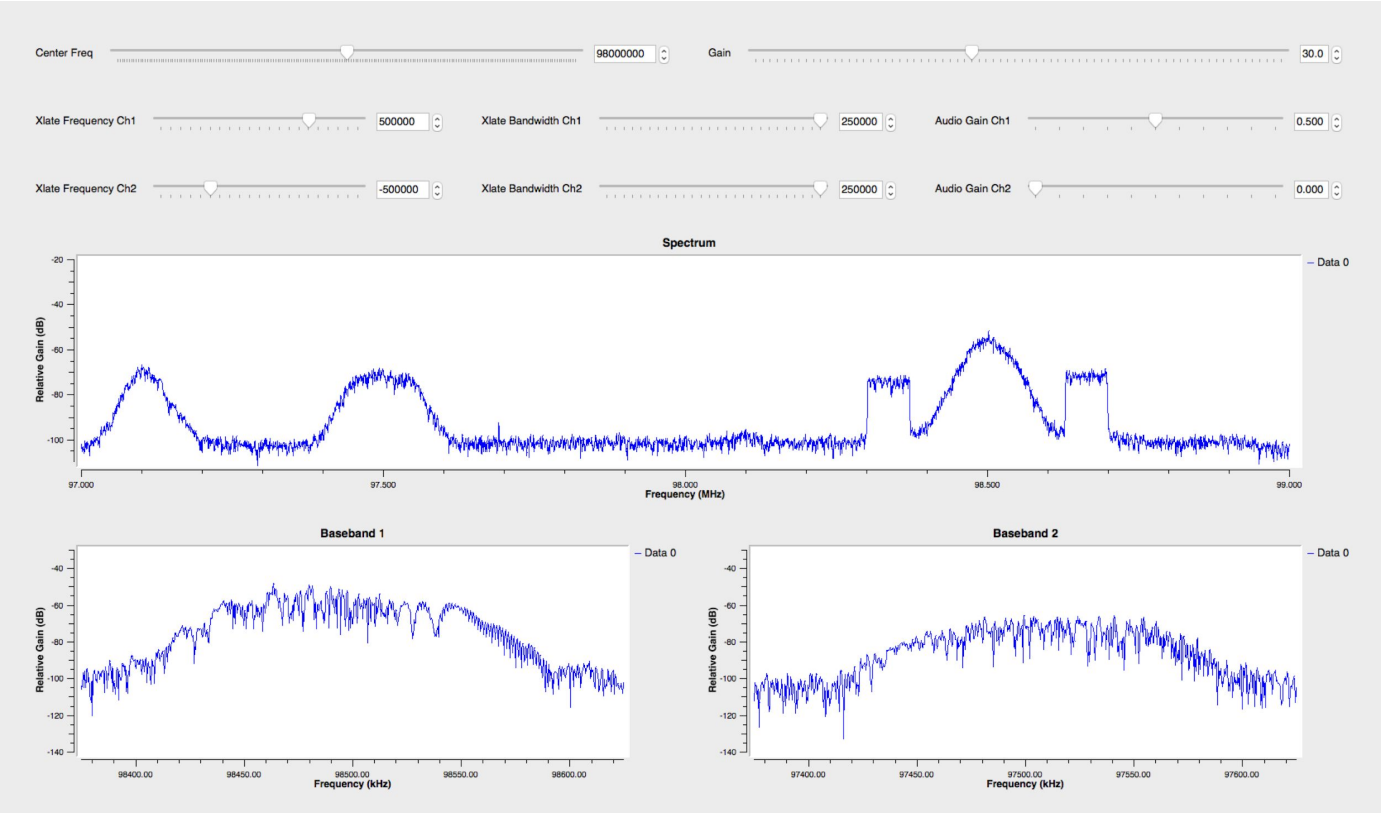
FM Receiver in GRC (Dual Channel FIR)

Location: ~/ettus_workshop/flowgraphs/wbfm_dual_rx_fir.grc



FM Receiver in GRC (Dual Channel FIR)

Location: ~/ettus_workshop/flowgraphs/wbfm_dual_rx_fir.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`

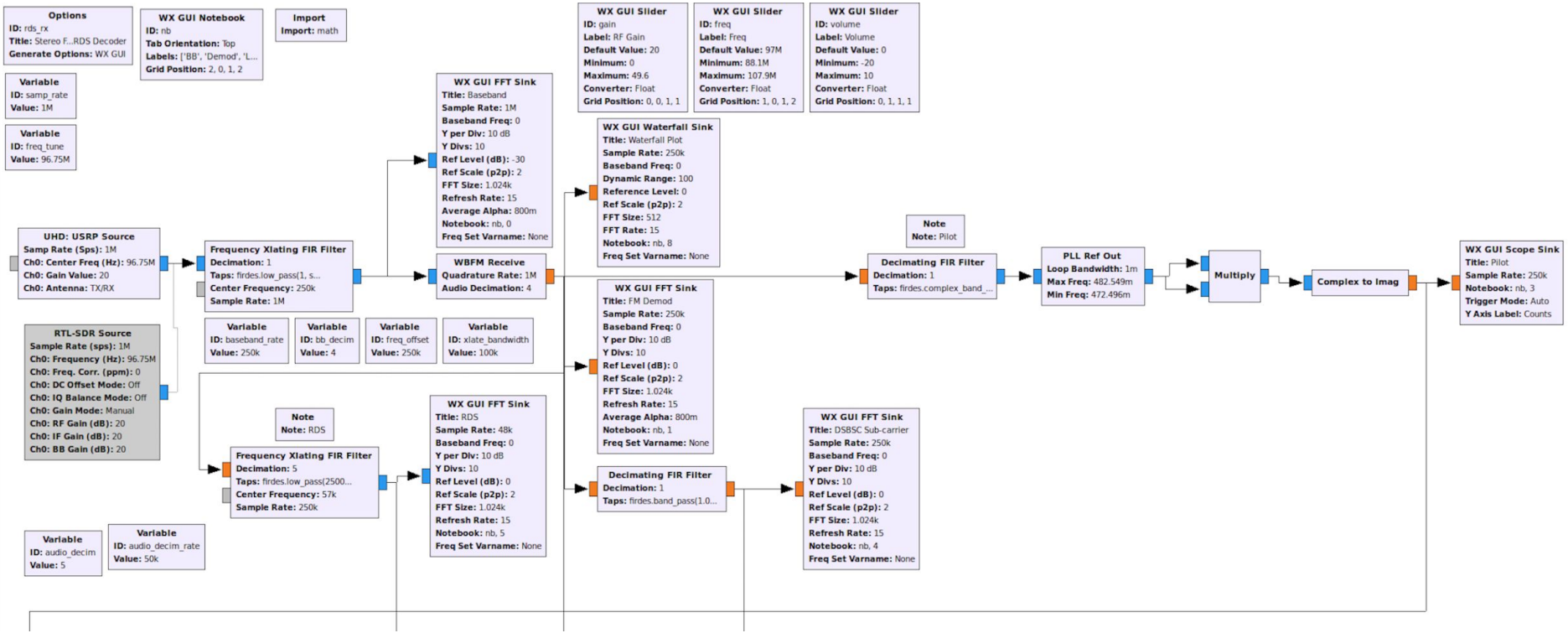
- In GRC, open: **`~/ettus_workshop/flowgraphs/rds_rx.grc`**
- Verify correct antenna under **`Options`** Block
- Start flowgraph
- Tune to strong station with RDS
- Adjust Gain slider if needed

Reference: ~/ettus_workshop/instructions/install.html

* Skip if using LiveSDR Environment

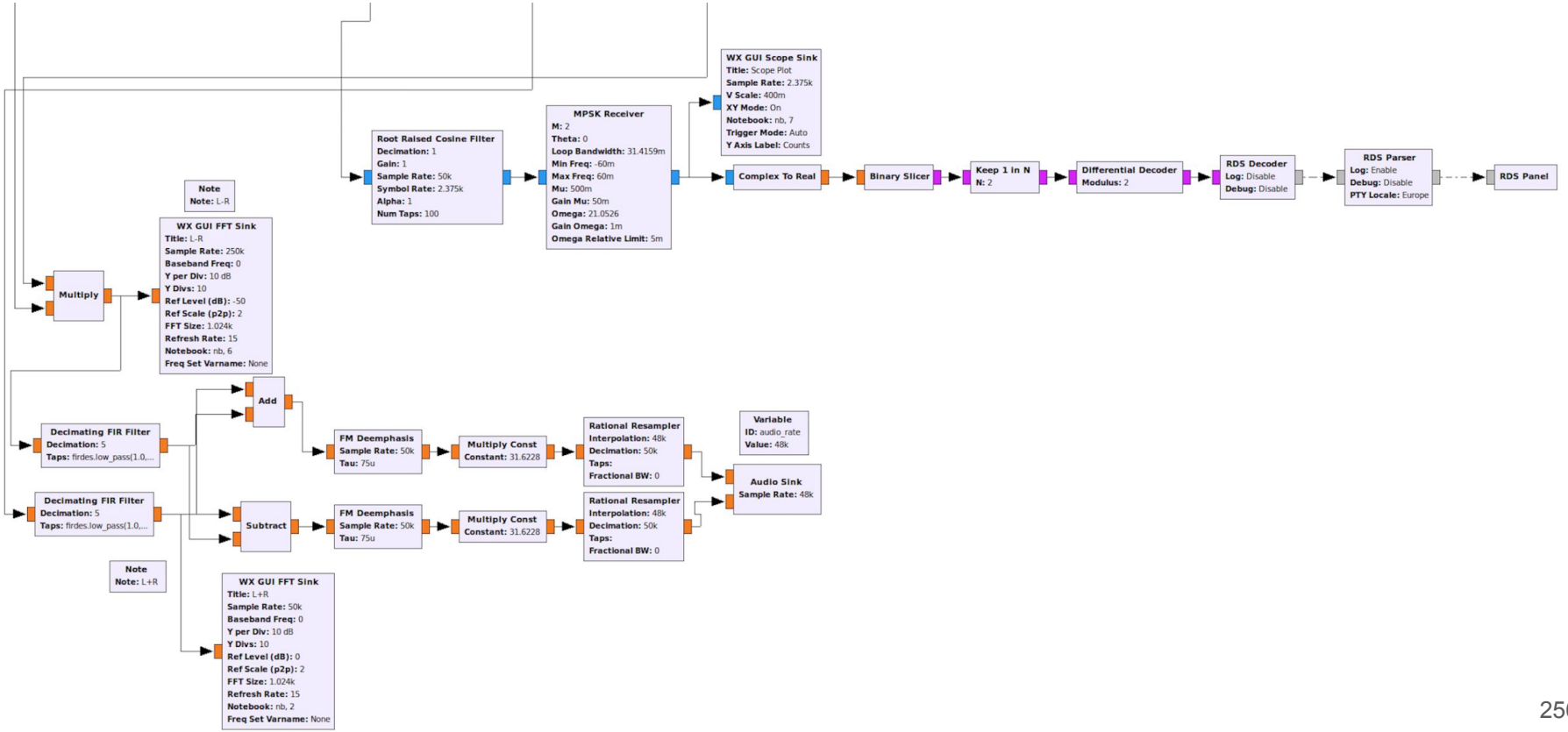
FM RDS Receiver in GRC - Part 1

Location: ~/ettus_workshop/flowgraphs/rds_rx.grc

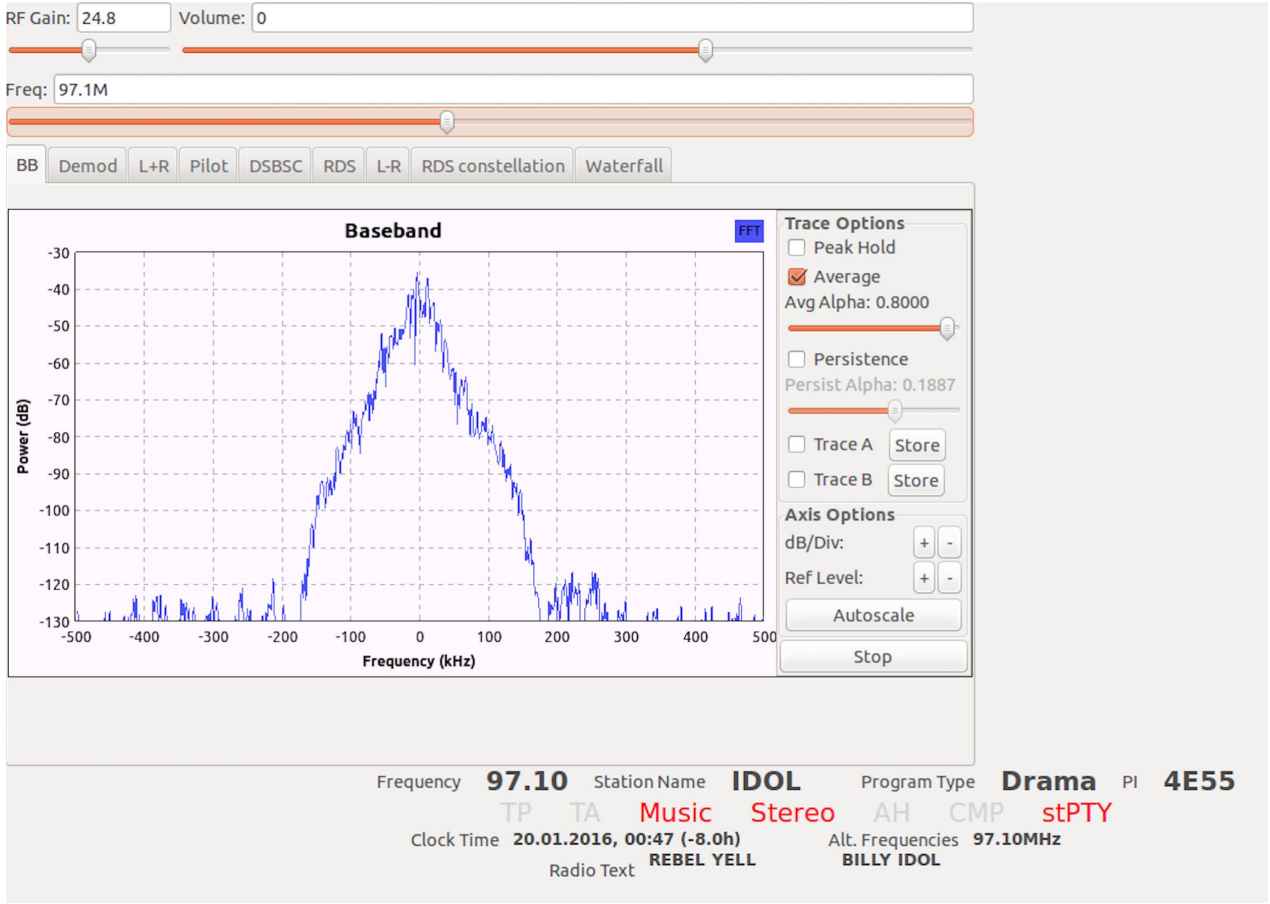


FM RDS Receiver in GRC - Part 2

Location: ~/ettus_workshop/flowgraphs/rds_rx.grc

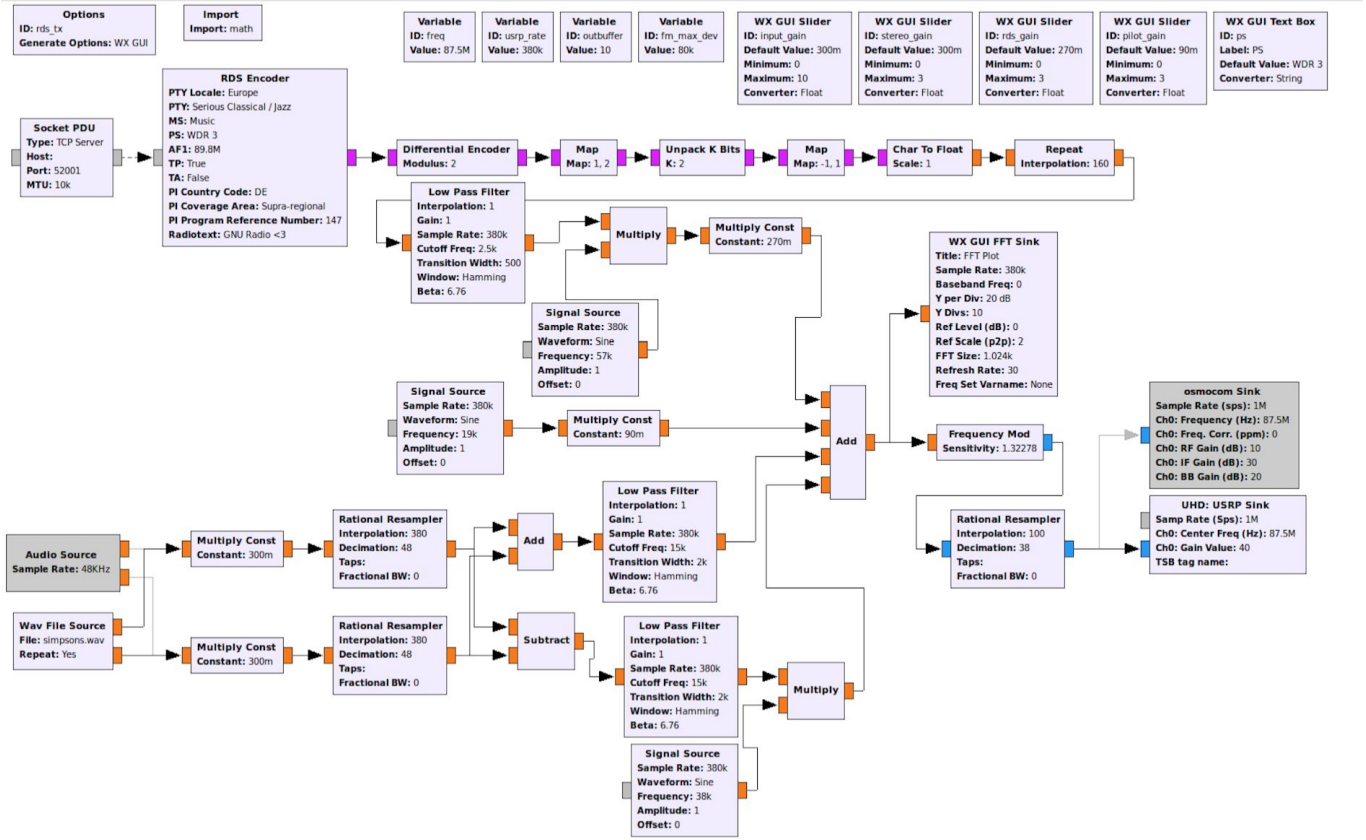


Out-of-Tree Module Installation: gr-rds



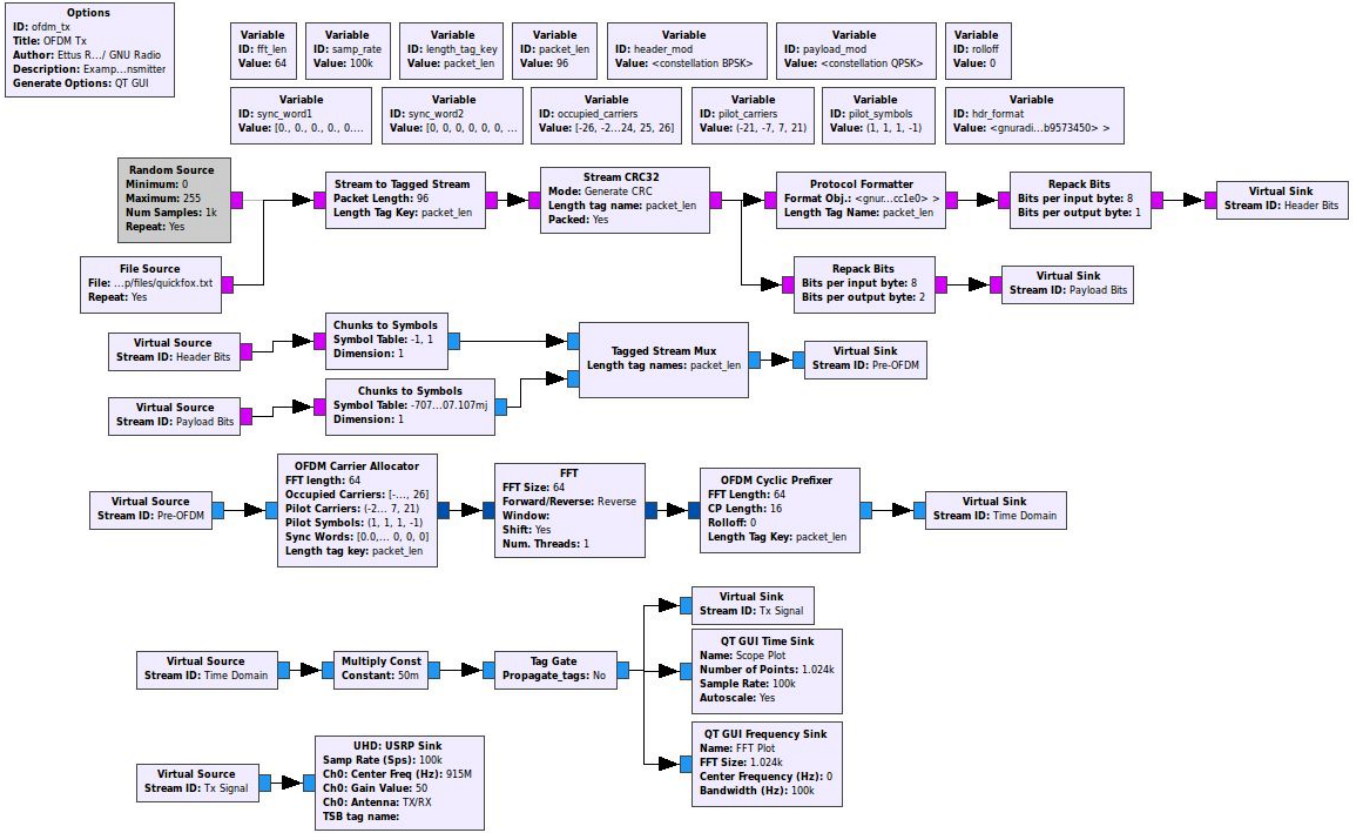
FM RDS Transmitter in GRC

Location: ~/ettus_workshop/flowgraphs/rds_tx.grc



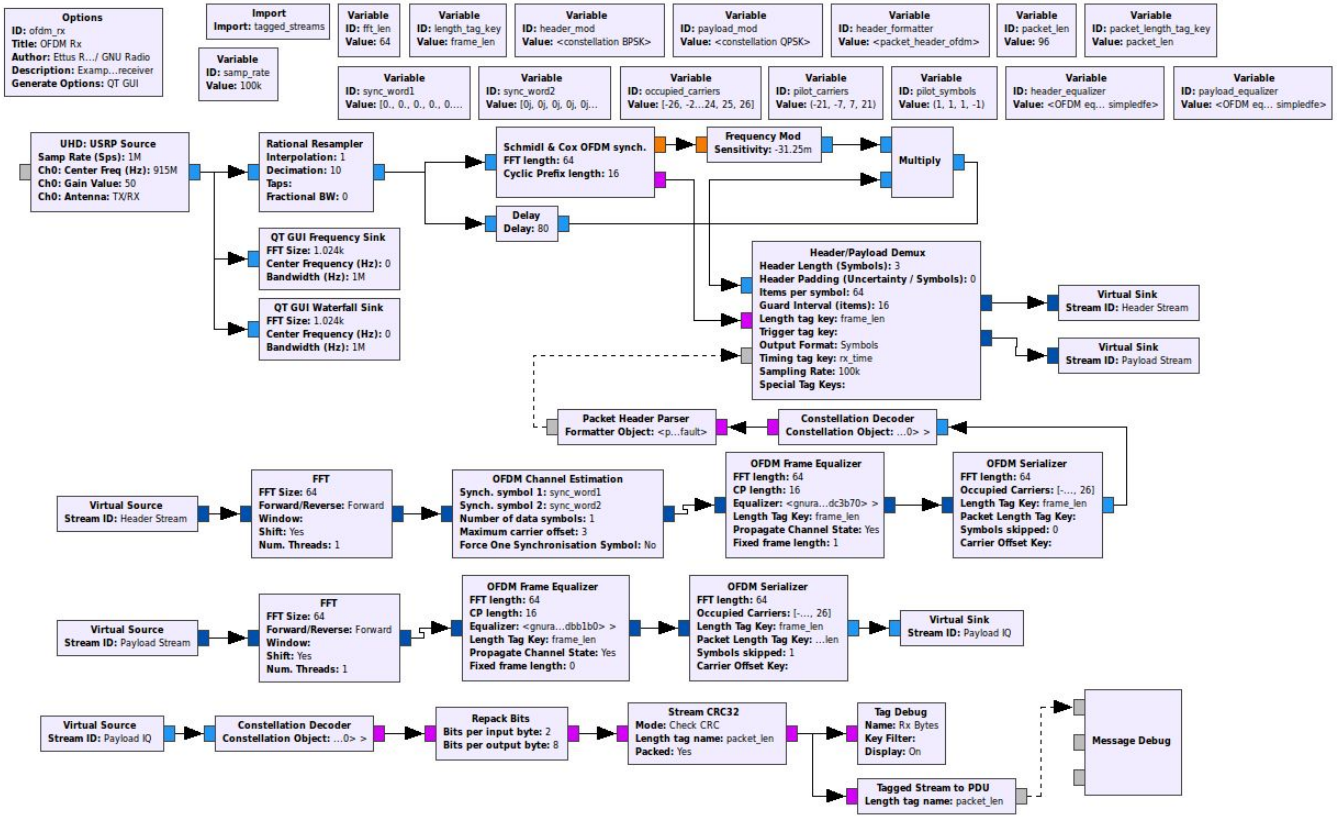
OFDM BPSK Transmitter and Receiver

Location: ~/ettus_workshop/flowgraphs/ofdm_tx.grc



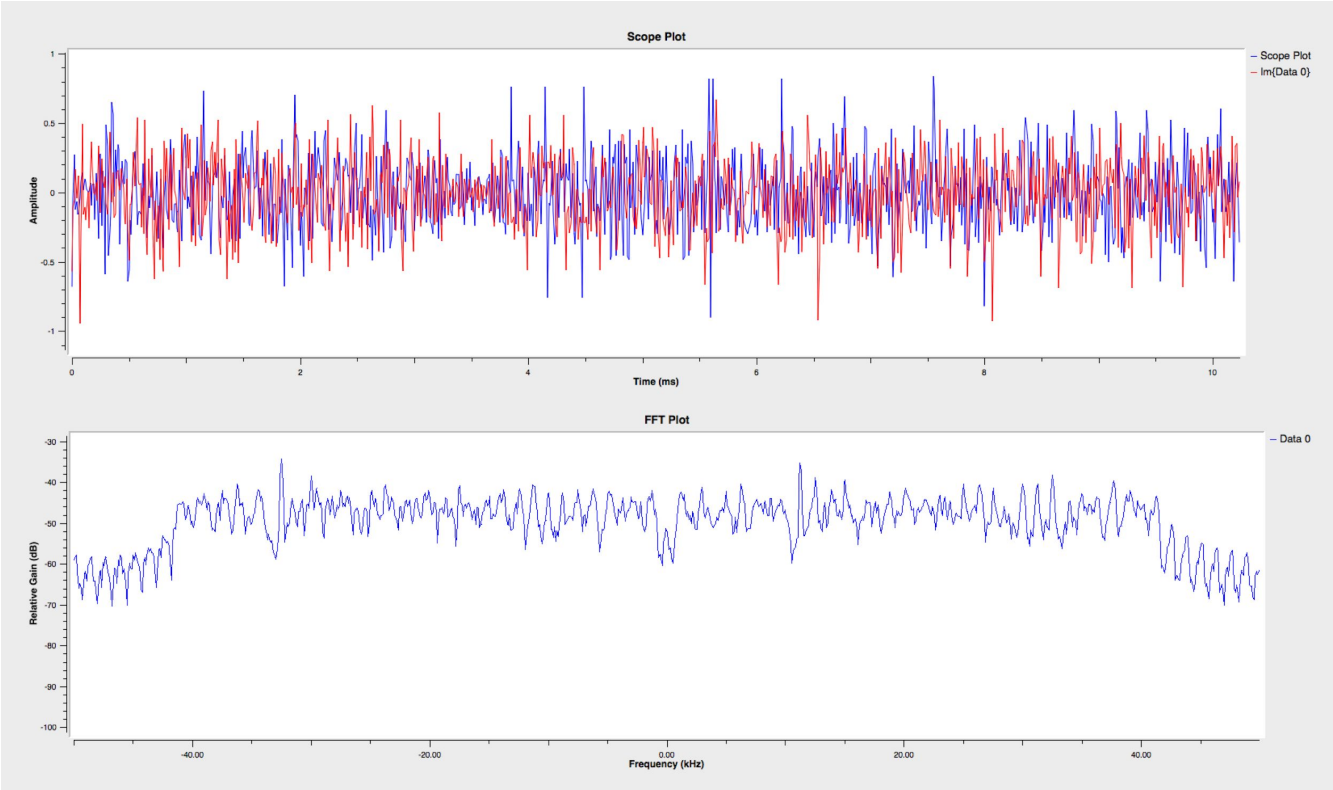
OFDM BPSK Transmitter and Receiver

Location: ~/ettus_workshop/flowgraphs/ofdm_rx.grc



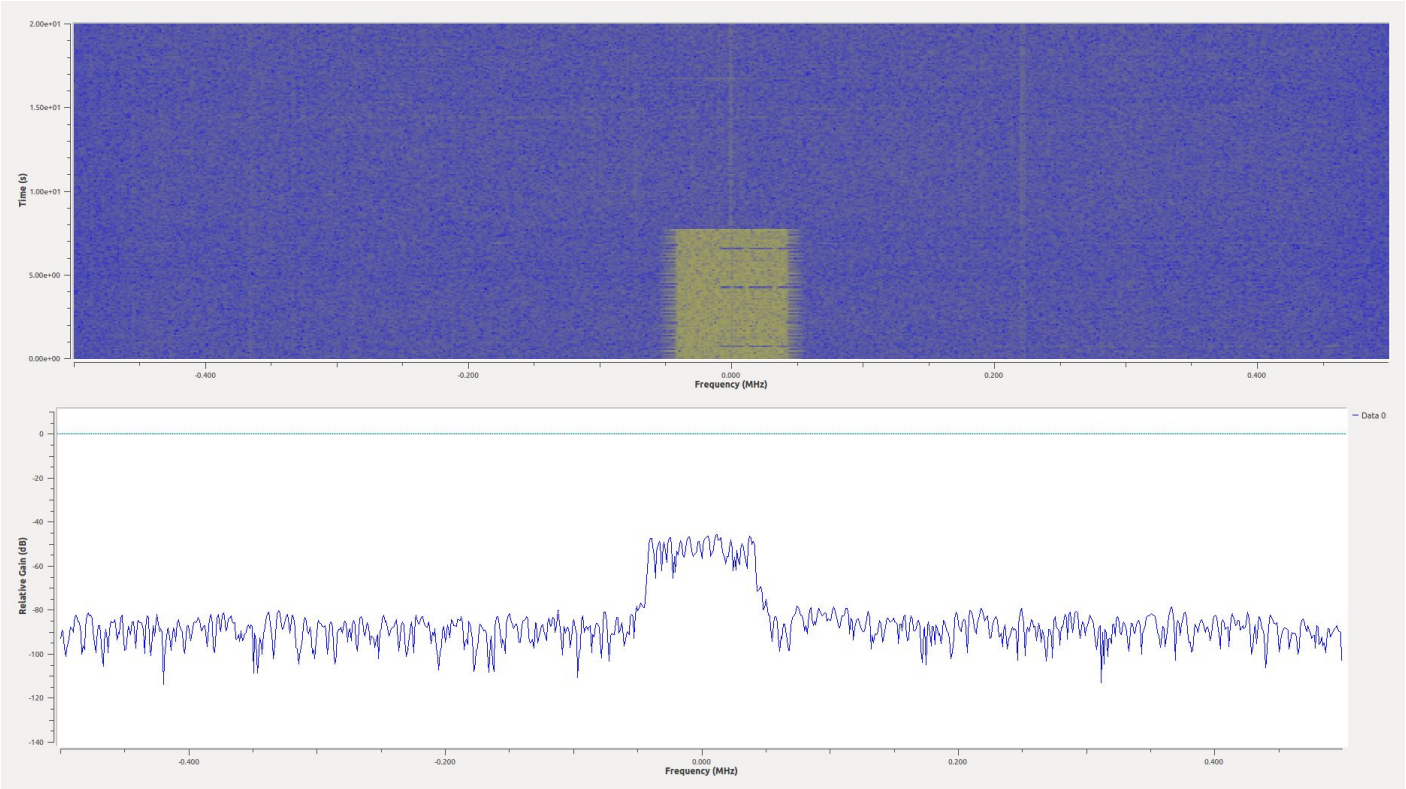
OFDM BPSK Transmitter and Receiver

Location: ~/ettus_workshop/flowgraphs/ofdm_tx.grc



OFDM BPSK Transmitter and Receiver

Location: ~/ettus_workshop/flowgraphs/ofdm_rx.grc



OFDM BPSK Transmitter and Receiver

Location: ~/ettus_workshop/flowgraphs/ofdm_rx.grc

```
--------------------------------------------
******* MESSAGE DEBUG PRINT ********

--------------------------------------------
Tag Debug: Rx Bytes
Input Stream: 00
  Offset: 374592  Source: n/a     Key: packet_num   Value: 2754
  Offset: 374592  Source: n/a     Key: rx_time   Value: {60 0.627483}
  Offset: 374592  Source: n/a     Key: ofdm_sync_carr_offset   Value: 0
  Offset: 374592  Source: n/a     Key: ofdm_sync_chan_taps   Value: #[(0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0.50317,0.0583197) (0.406829,-0.332531) (0.0768853,-0.539353)
(-0.299765,-0.444107) (-0.520336,-0.0964112) (-0.411394,0.348308) (-0.128469,0.540099) (0.279397,0.469937) (0.534734,0.15615) (0.480536,-0.246443) (0.178435,-0.523367)
(-0.226049,-0.509302) (-0.515899,-0.20567) (-0.522154,0.20976) (-0.220224,0.522914) (0.187443,0.529673) (0.501944,0.252847) (0.0605469,-0.0266685) (0.033371,-0.0526452)
(-0.210673,-0.511695) (-0.0497782,-0.030815) (-0.0602623,0.0131421) (-0.0364705,0.0539207) (0.017276,0.0577378) (0.0501564,0.0438867) (0.0615571,-0.0230514) (0,0)
(0.00217673,-0.0826737) (-0.0379996,-0.0474547) (-0.0737203,0.00126344) (-0.0434138,0.0377747) (-0.011016,0.0682483) (0.0452003,0.0569671) (0.575386,0.00141254)
(0.0476044,-0.0396964) (0.00239569,-0.0669852) (-0.0393086,-0.0498288) (-0.0584019,-0.0133017) (-0.0510127,0.0308333) (-0.0163422,0.0580334) (0.0344359,0.0508854)
(0.0552663,0.0193386) (0.0510031,-0.0260913) (0.0268287,-0.0578494) (-0.0235808,-0.0534187) (-0.0557805,-0.0215279) (-0.0561213,0.0219966) (-0.16399,0.514263)
(0.0205786,0.0569654) (0.0546058,0.0231806) (0.0529967,-0.0169914) (0.0283085,-0.0498383) (-0.0176685,-0.0542161) (0,0) (0,0) (0,0) (0,0) (0,0)]
  Offset: 374592  Source: n/a     Key: packet_len   Value: 96
--------------------------------------------
(((ofdm_sync_chan_taps . #[(0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (0.50317,0.0583197) (0.406829,-0.332531) (0.0768853,-0.539353) (-0.299765,-0.444107) (-0.520336,-0.0964112)
(-0.411394,0.348308) (-0.128469,0.540099) (0.279397,0.469937) (0.534734,0.15615) (0.480536,-0.246443) (0.178435,-0.523367) (-0.226049,-0.509302) (-0.515899,-0.20567)
(-0.522154,0.20976) (-0.220224,0.522914) (0.187443,0.529673) (0.501944,0.252847) (0.0605469,-0.0266685) (0.033371,-0.0526452) (-0.210673,-0.511695) (-0.0497782,-0.030815)
(-0.0602623,0.0131421) (-0.0364705,0.0539207) (0.017276,0.0577378) (0.0501564,0.0438867) (0.0615571,-0.0230514) (0,0) (0.00217673,-0.0826737) (-0.0379996,-0.0474547)
(-0.0737203,0.00126344) (-0.0434138,0.0377747) (-0.011016,0.0682483) (0.0452003,0.0569671) (0.575386,0.00141254) (0.0476044,-0.0396964) (0.00239569,-0.0669852)
(-0.0393086,-0.0498288) (-0.0584019,-0.0133017) (-0.0510127,0.0308333) (-0.0163422,0.0580334) (0.0344359,0.0508854) (0.0552663,0.0193386) (0.0510031,-0.0260913)
(0.0268287,-0.0578494) (-0.0235808,-0.0534187) (-0.0557805,-0.0215279) (-0.0561213,0.0219966) (-0.16399,0.514263) (0.0205786,0.0569654) (0.0546058,0.0231806)
(0.0529967,-0.0169914) (0.0283085,-0.0498383) (-0.0176685,-0.0542161) (0,0) (0,0) (0,0) (0,0) (0,0)]) (ofdm_sync_carr_offset . 0) (rx_time . {60 0.627483}) (packet_num . 2754)) . #[  5
6  7  8  9
 the quick brown fox jumped over the lazy dog 0 1 2 3 4 5 6 7 8 9
```

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
```

* Skip if using LiveSDR Environment

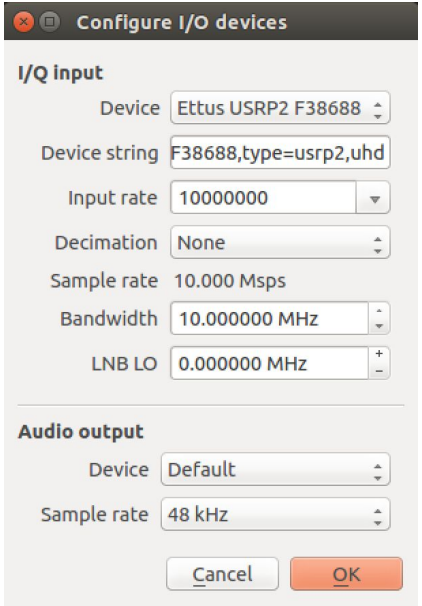
Reference: ~/ettus_workshop/instructions/install.html

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

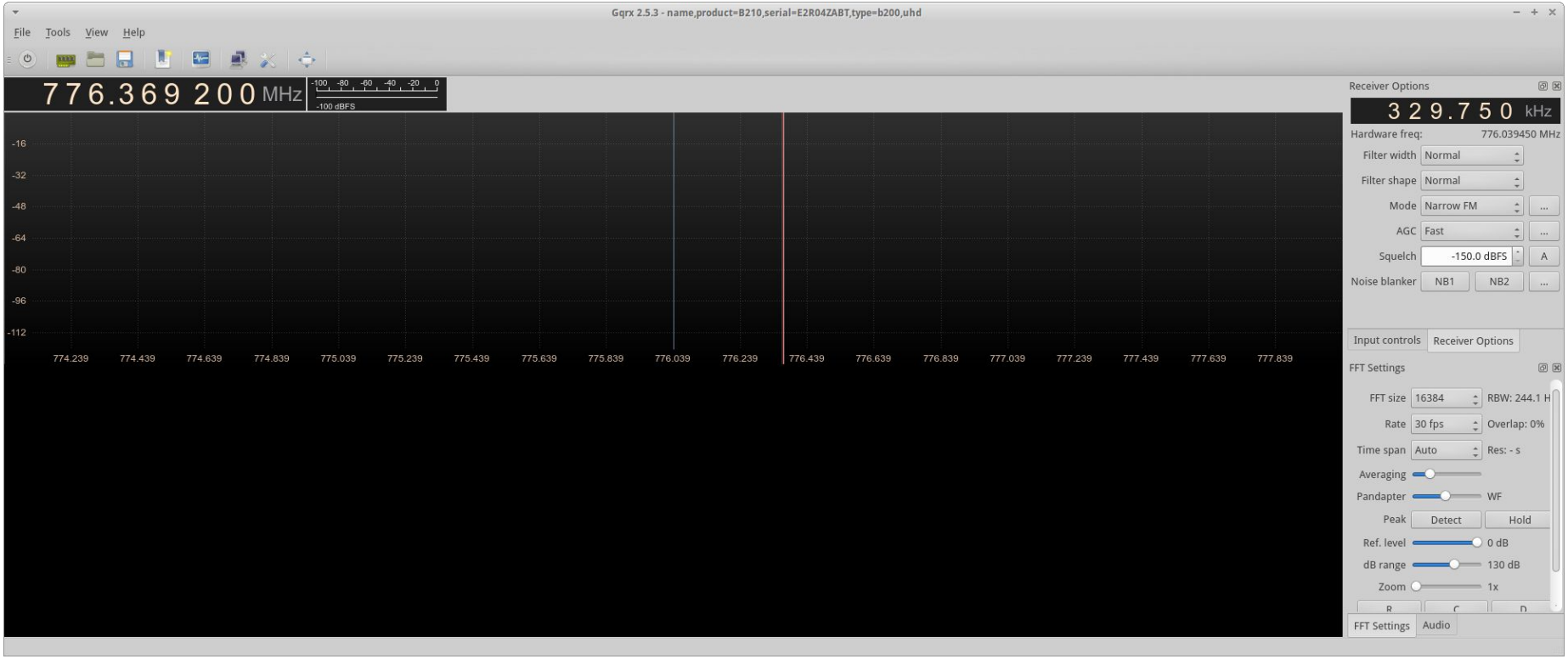
1. `sudo apt-get install qt5-default qttools5-dev-tools libqt5svg5 libqt5svg5-dev`
2. `git clone https://github.com/csete/gqrx.git`
3. `cd gqrx`
4. `mkdir build && cd build`
5. `cmake ../`
6. `make -j4`
7. `sudo make install`
8. `sudo ldconfig`



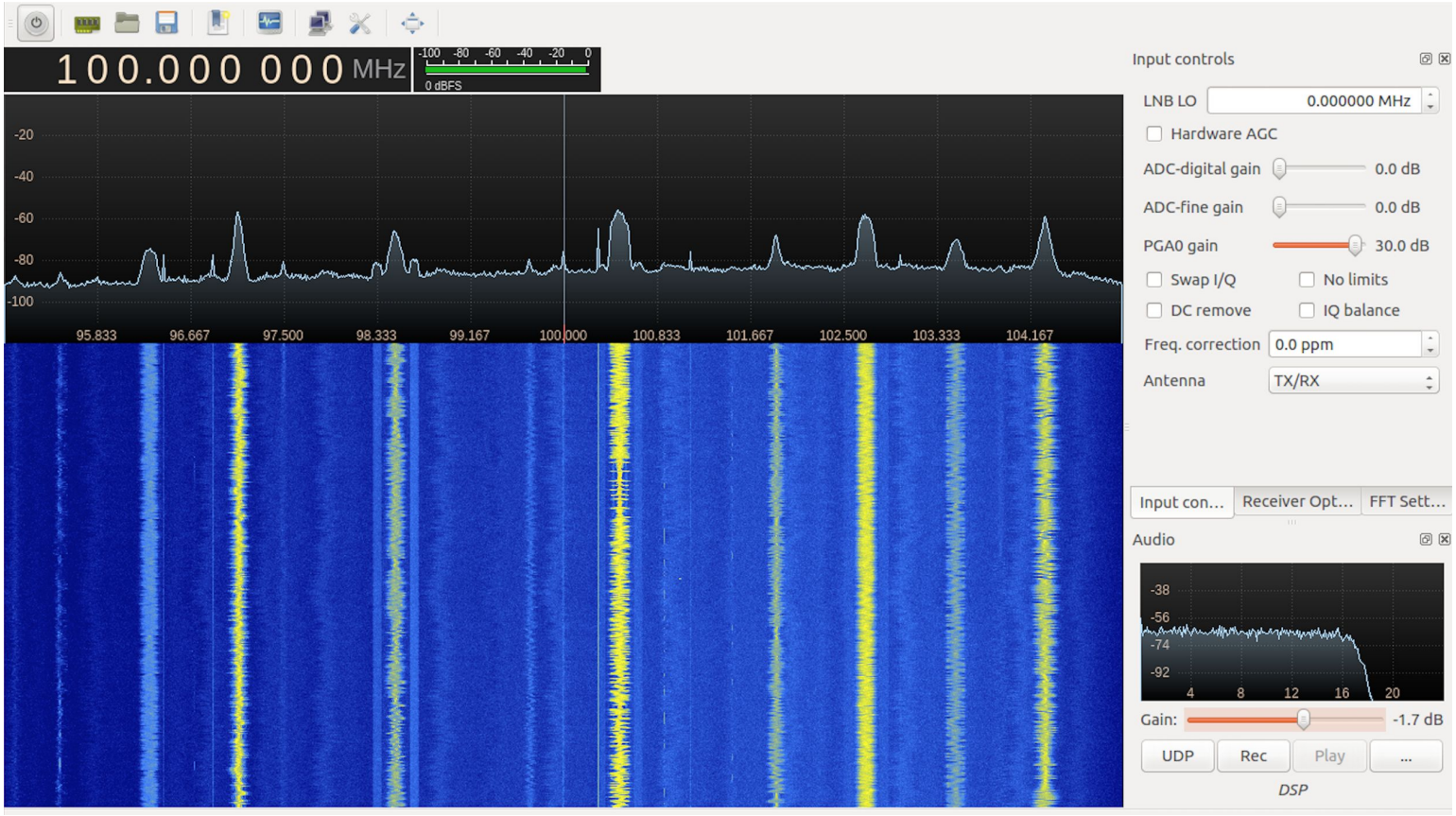
- To start, run at command prompt: **`gqrx`**
- Select Device, Set Input Rate, Decimation and Bandwidth

Reference: ~/ettus_workshop/instructions/install.html

GQRX Screenshot

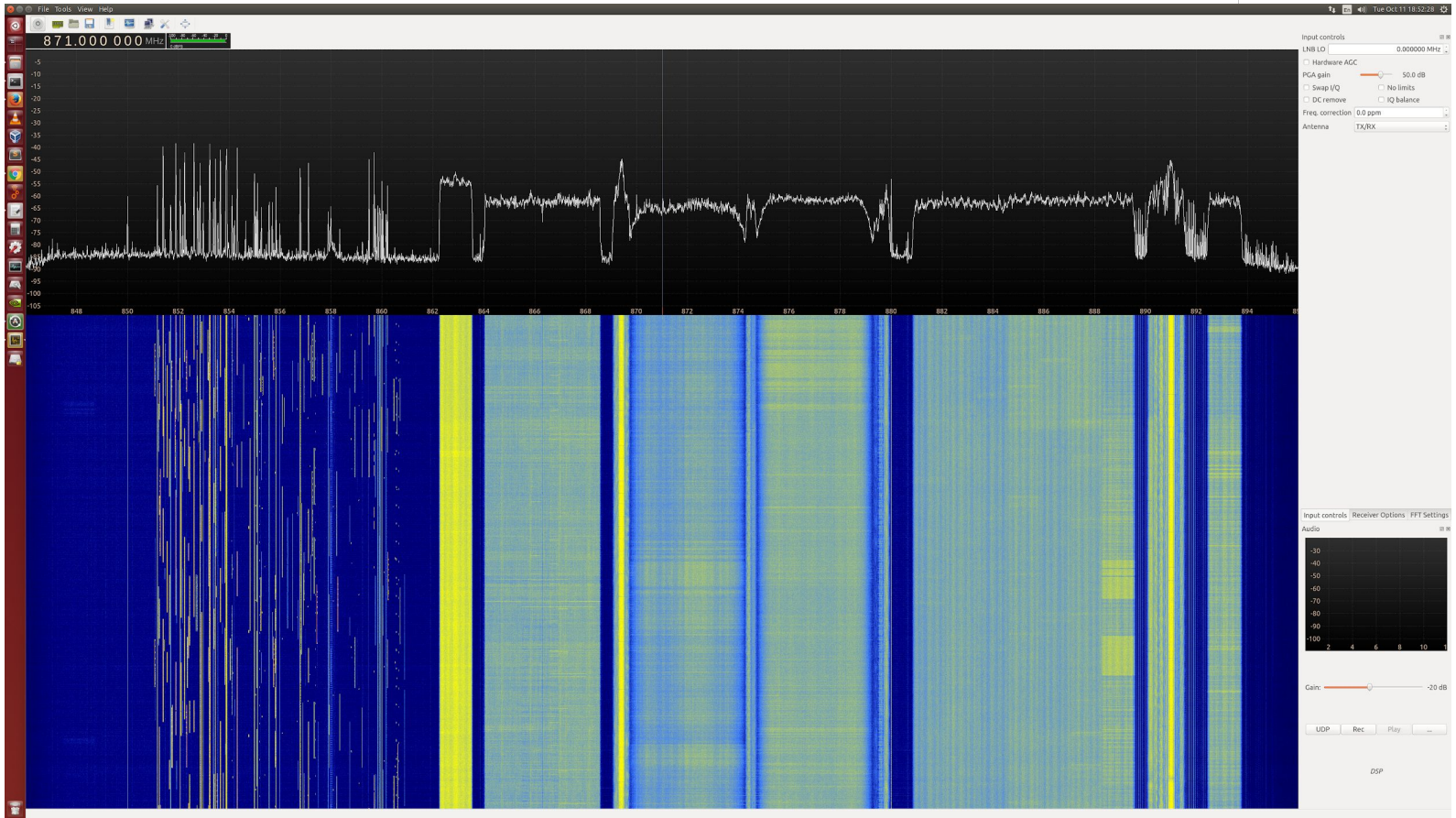


GQRX Screenshot



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

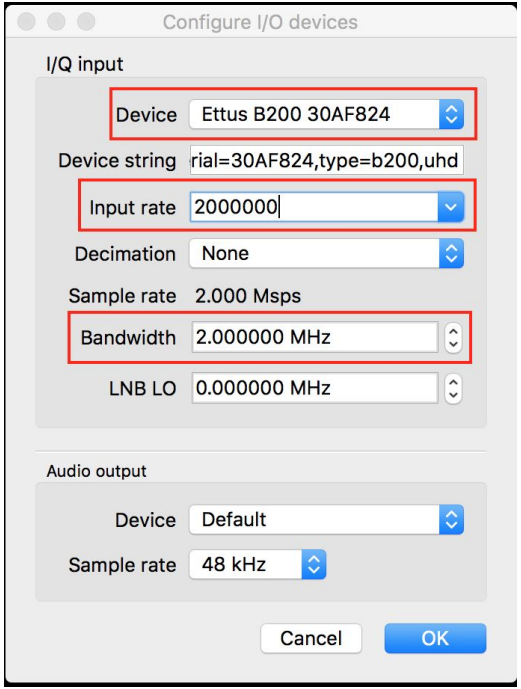
1. `git clone https://github.com/drmpeg/gr-paint.git`
2. `cd gr-paint`
3. `mkdir build`
4. `cd build`
5. `cmake ..`
6. `make`
7. `sudo make install`
8. `sudo ldconfig`

Reference: ~/ettus_workshop/instructions/install.html

* Skip if using LiveSDR Environment

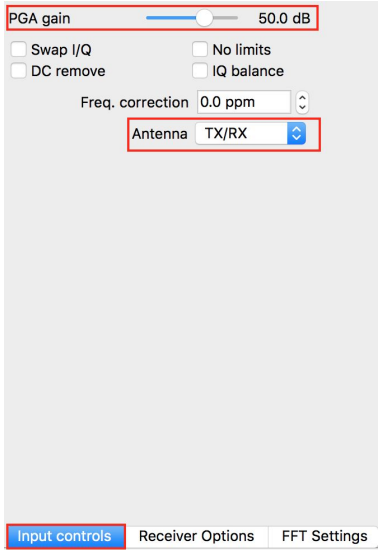
gr-paint - RX demo

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



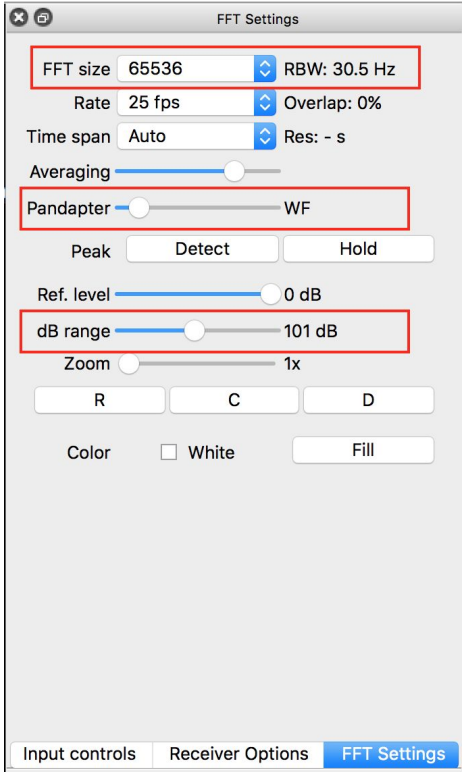
gr-paint - RX demo

1. Under "FFT Settings" tab set:

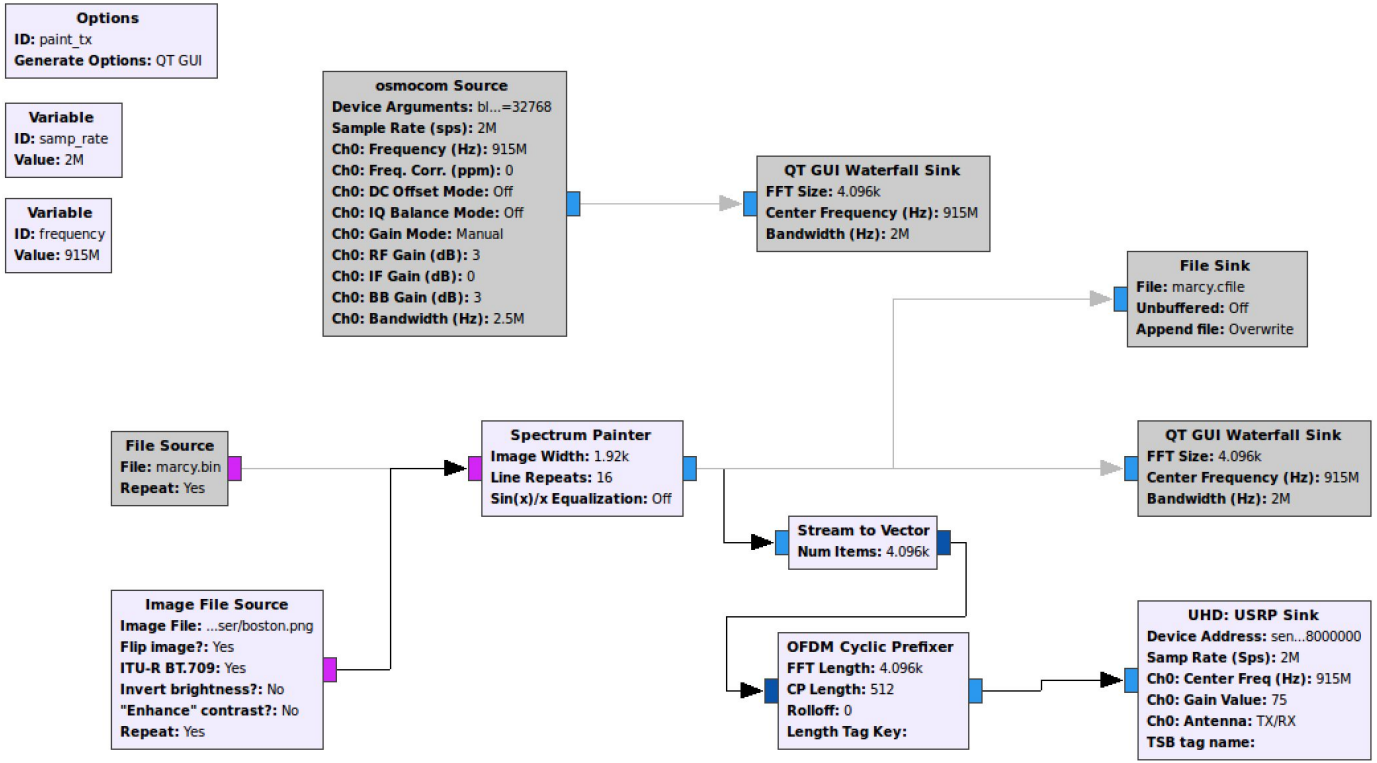
   FFT Size: 65536

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

   dB Range to ~ 100 dB



Demo - gr-paint



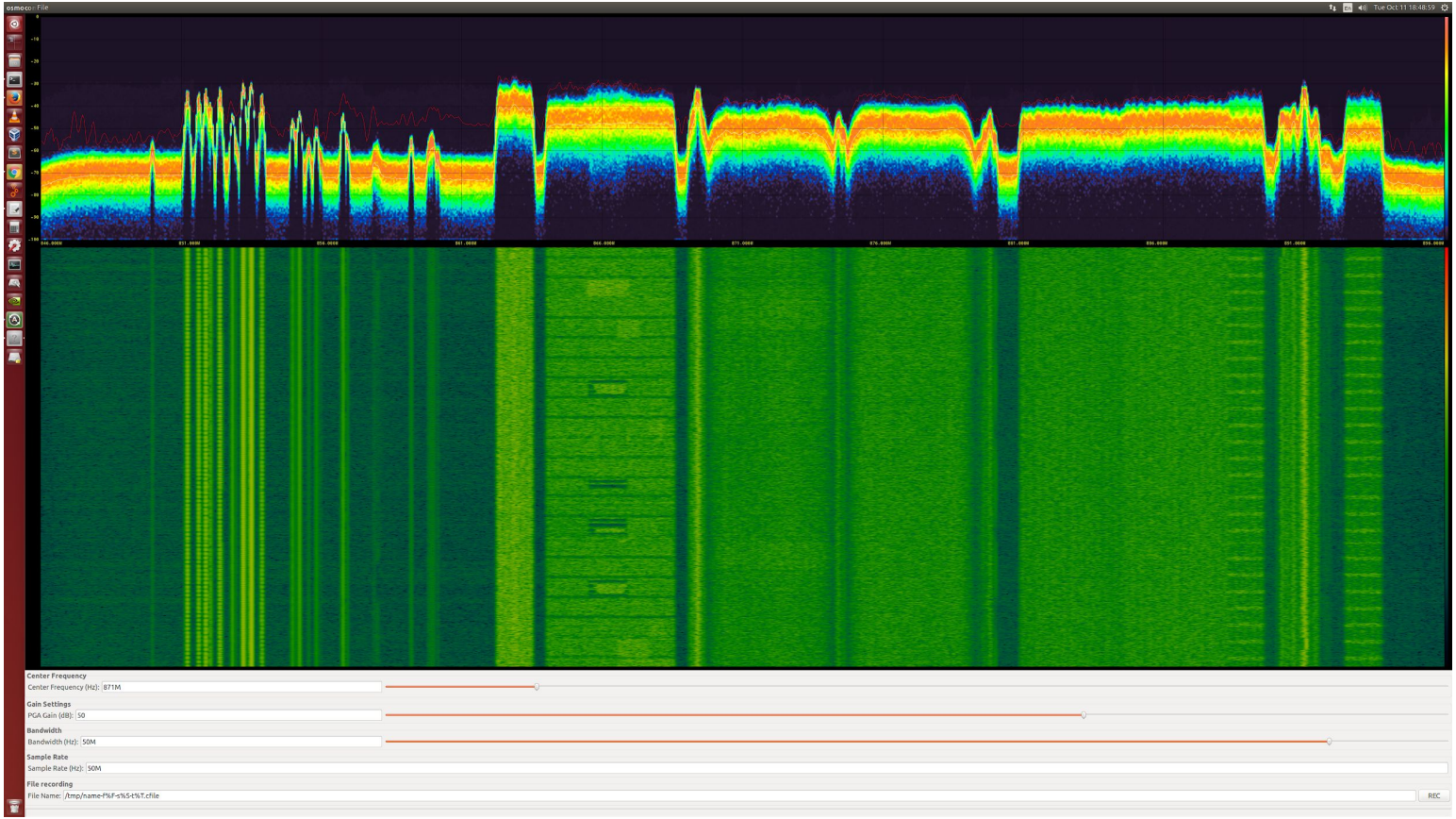
Demo - gr-paint

gr-fosphor

- Open-source, GPU-accelerated FFT and Waterfall display tool
- GNU Radio block for RTSA-like spectrum visualization
- Uses OpenCL and OpenGL for acceleration
- High frame rate, great for visualizing fast-changing signals
- RFNoC variant for X3xx, E3xx USRPs - FPGA accelerated FFT/Waterfall
- Created by Sylvain Munaut, **`@tnt`**, **`https://github.com/smunaut`**
- Homepage **`http://sdr.osmocom.org/trac/wiki/fosphor`**
- Video demo **`https://www.youtube.com/watch?v=mjD-l3GAghU`**

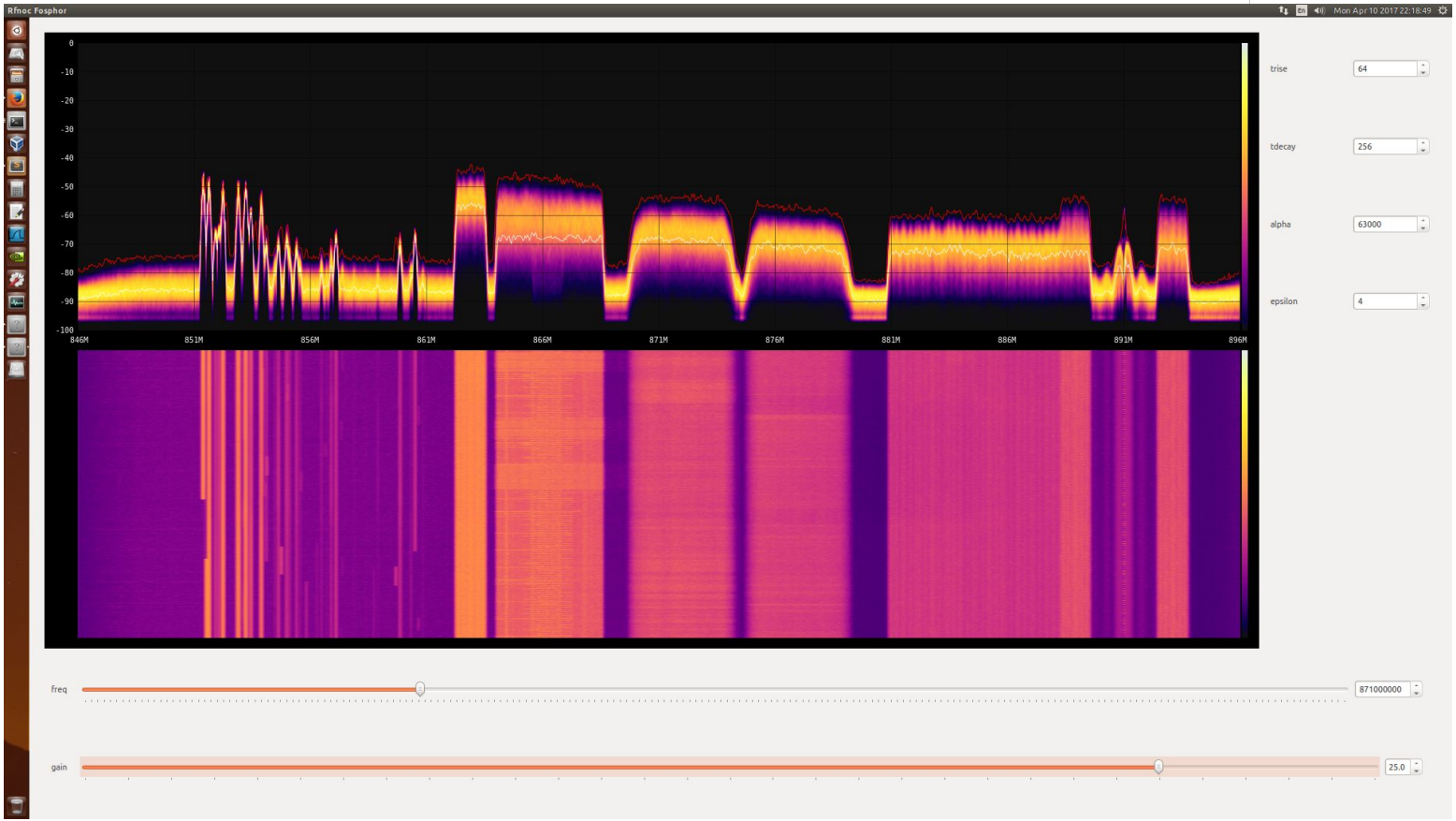
gr-fosphor - GPU - Screenshot 50 MS/s

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



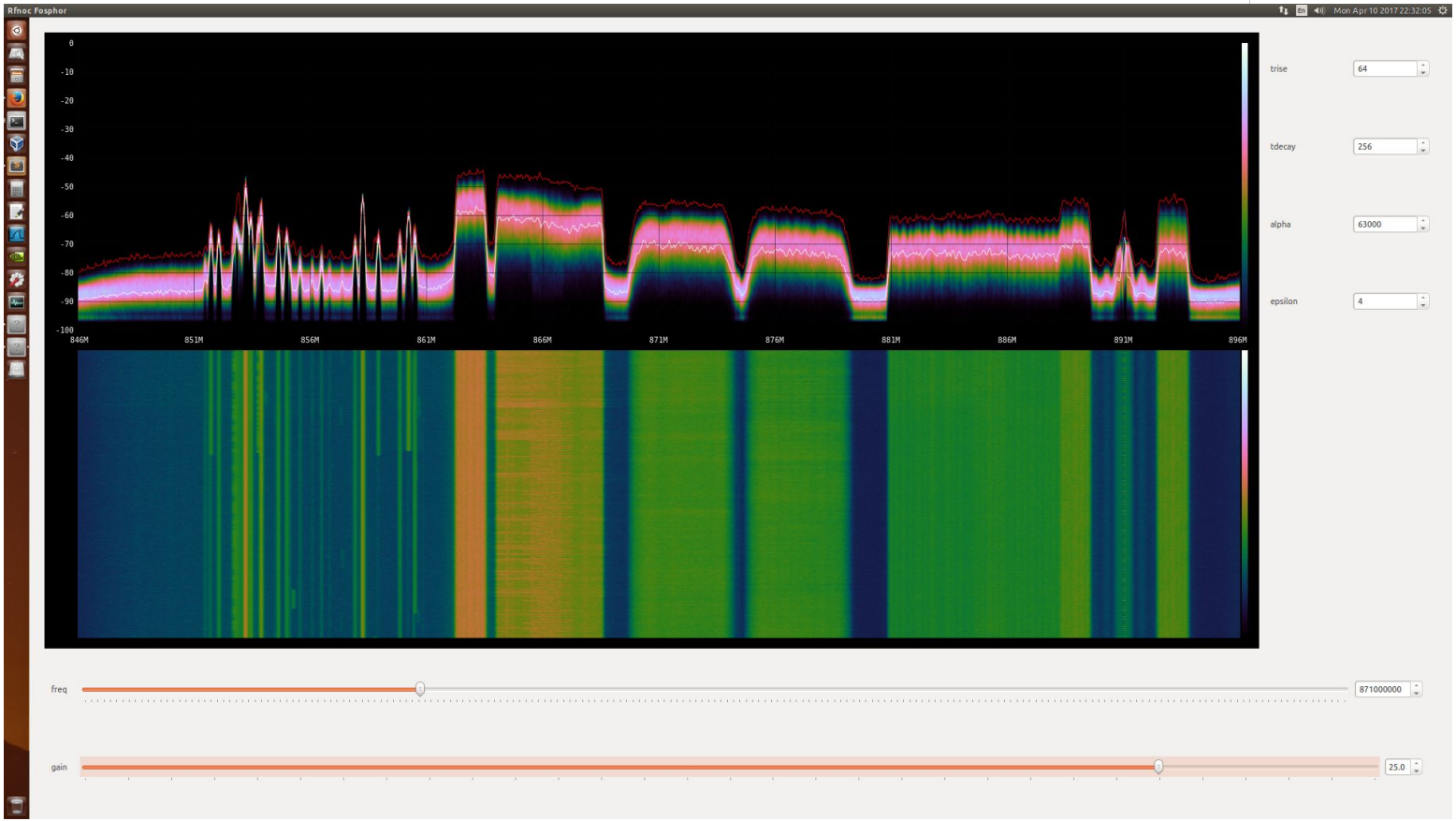
RFNoC gr-fosphor - Screenshot 50 MS/s

Frequency 871 MHz - NFM, P25, LTE, WCDMA



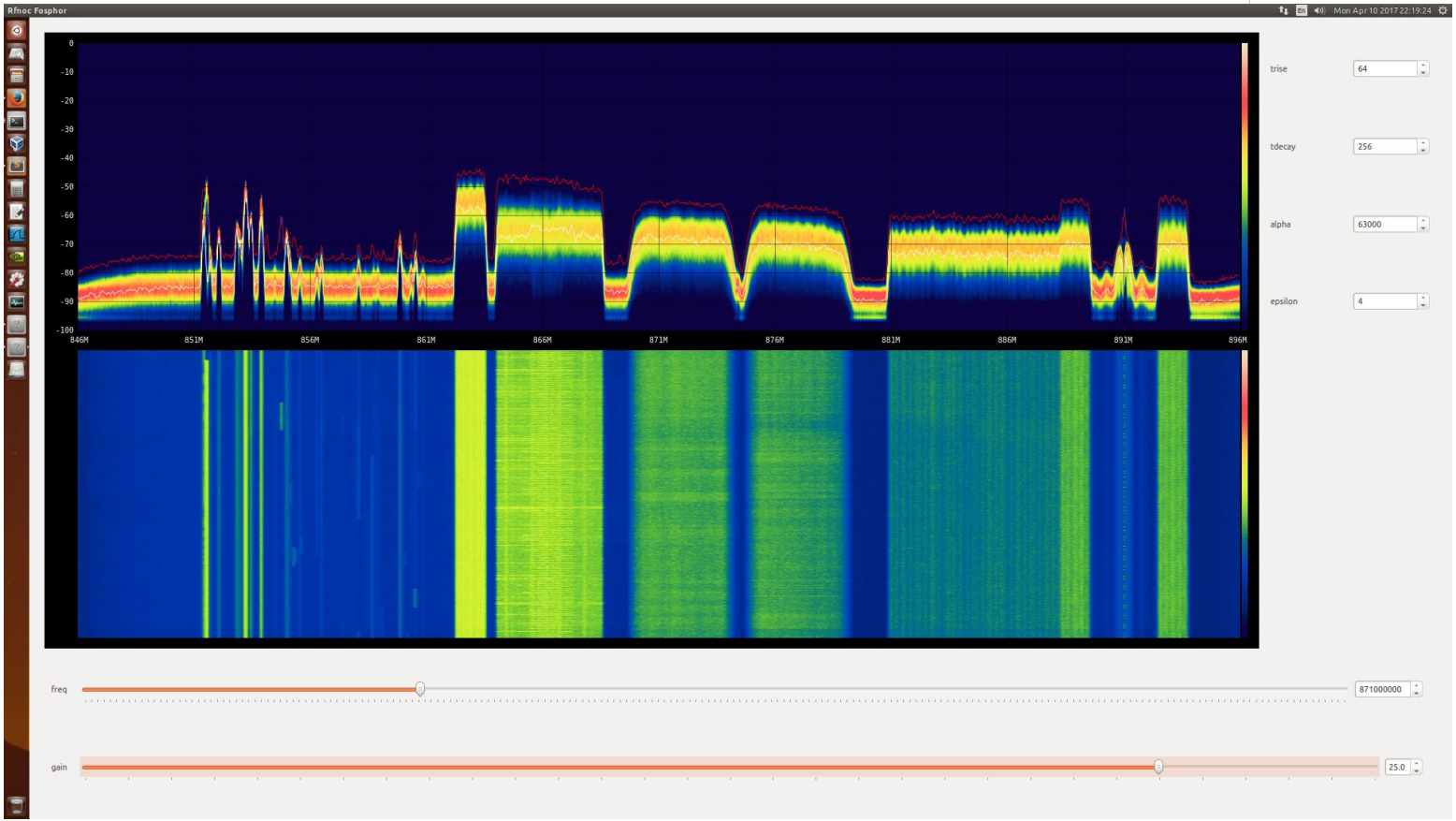
RFNoC gr-fosphor - Screenshot 50 MS/s

Frequency 871 MHz - NFM, P25, LTE, WCDMA



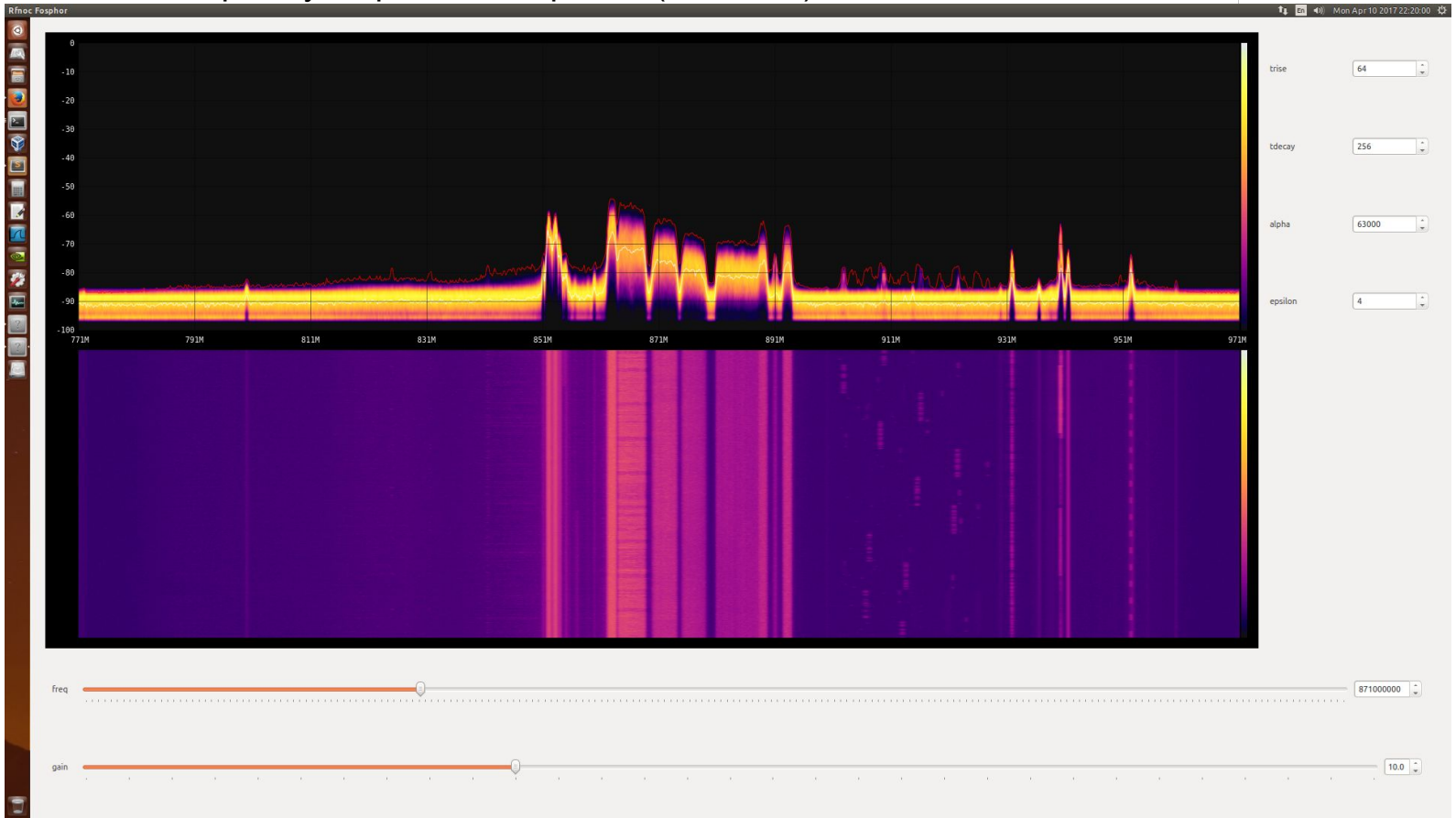
RFNoC gr-fosphor - Screenshot 50 MS/s

Frequency 871 MHz - NFM, P25, LTE, WCDMA



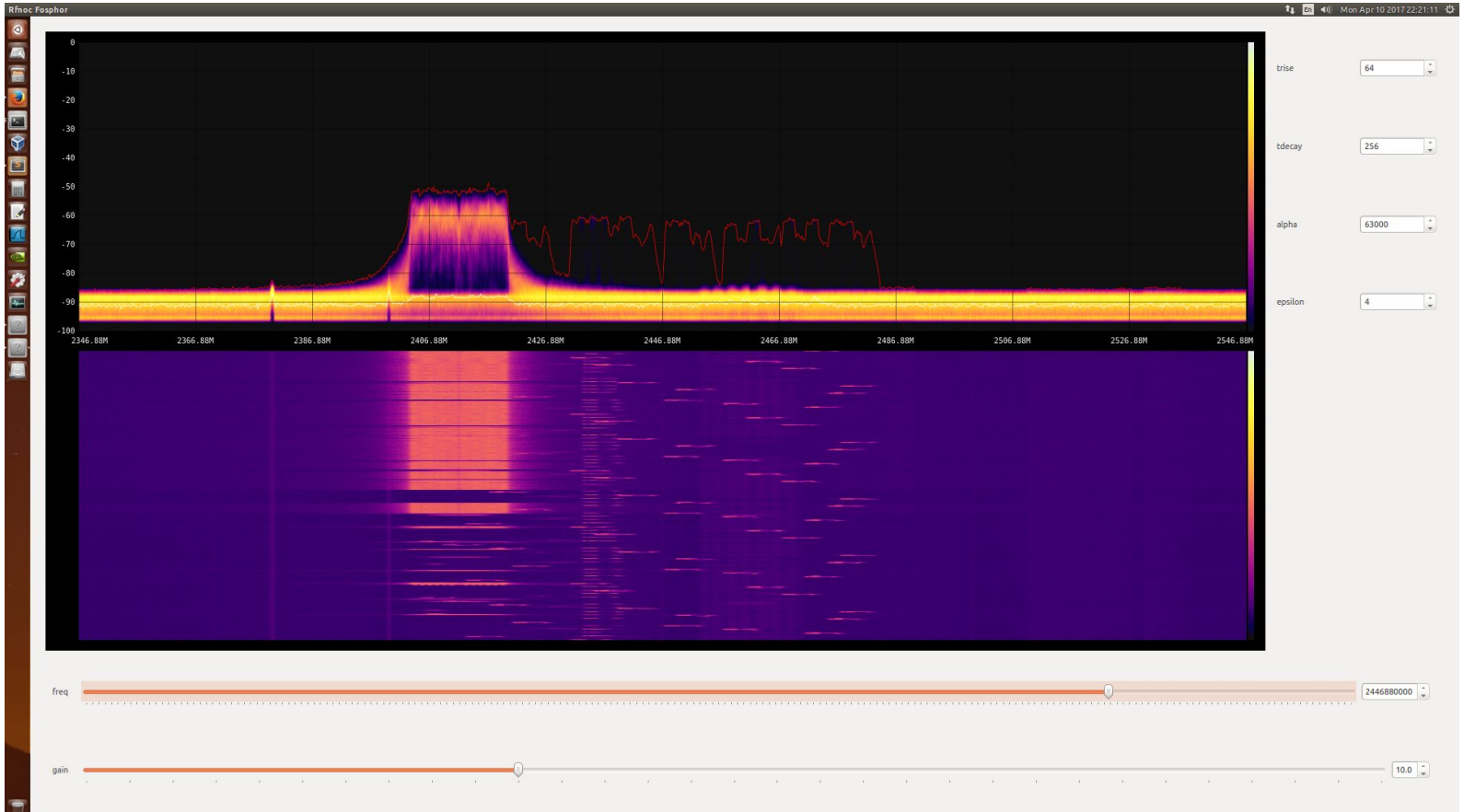
RFNoC gr-fosphor - Screenshot 200 MS/s

* Same Center Frequency as previous captures (871 MHz)



RFNoC gr-fosphor - Screenshot 200 MS/s

2.4 GHz WiFi



gr-fosphor - Installation - Intel OpenCL

Install Dependencies:

```
sudo apt-get install cmake xorg-dev libglu1-mesa-dev opencl-headers ocl-icd-opencl-dev alien clinfo
```

Install GLFW3:

```
cd ~/workarea
git clone https://github.com/glfw/glfw
cd glfw
mkdir build
cd build
cmake ../ -DBUILD_SHARED_LIBS=true
make
sudo make install
sudo ldconfig
```

* Skip if using LiveSDR Environment

Reference: ~/ettus_workshop/instructions/install.html

gr-fosphor - Installation - Intel OpenCL

Installing Intel OpenCL

```
mkdir $HOME/tmp

cd $HOME/tmp

cp ~/ettus_workshop/files/opencl_runtime_14.2_x64_4.5.0.8.tgz .

## or ##

wget http://registrationcenter.intel.com/irc_nas/4181/opencl_runtime_14.2_x64_4.5.0.8.tgz


tar xf opencl_runtime_14.2_x64_4.5.0.8.tgz

cd pset_opencl_runtime_14.1_x64_4.5.0.8/rpm
```

* Skip if using LiveSDR Environment

Reference: ~/ettus_workshop/instructions/install.html

gr-fosphor - Installation - Intel OpenCL

Ettus Research™
A National Instruments Company

Installing Intel OpenCL

```
alien --to-tgz opencl-1.2-base-pset-4.5.0.8-1.noarch.rpm
tar xf opencl-1.2-base-4.5.0.8.tgz
sudo mv opt/intel /opt
rm -rf opt

alien --to-tgz opencl-1.2-intel-cpu-4.5.0.8-1.x86_64.rpm
tar xf opencl-1.2-intel-cpu-4.5.0.8.tgz

sudo mkdir -p /etc/OpenCL/vendors

sudo mv opt/intel/opencl-1.2-4.5.0.8/etc/intel64.icd /etc/OpenCL/vendors/

sudo mkdir -p /opt/intel/opencl-1.2-4.5.0.8/lib64/

sudo mv opt/intel/opencl-1.2-4.5.0.8/lib64/* /opt/intel/opencl-1.2-4.5.0.8/lib64/
rm -rf opt
```

* Skip if using LiveSDR Environment

Reference: ~/ettus_workshop/instructions/install.html

gr-fosphor - Installation

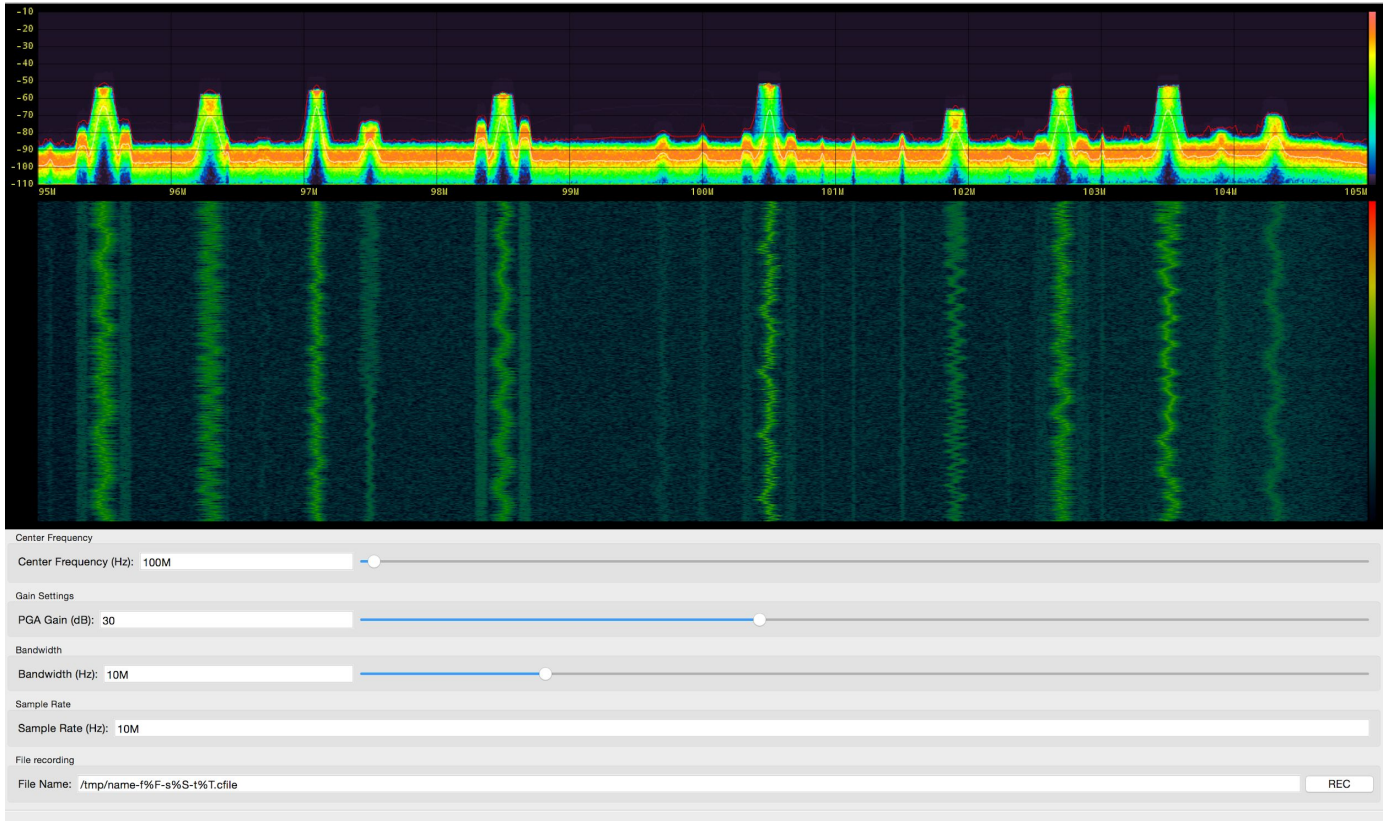
```
cd ~/workarea

git clone git://git.osmocom.org/gr-fosphor
cd gr-fosphor
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

* Skip if using LiveSDR Environment

Reference: ~/ettus_workshop/instructions/install.html

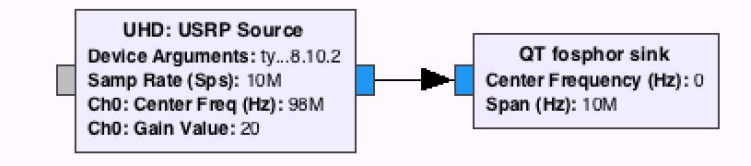
gr-fosphor - Demo with osmocom_fft

```
$ osmocom_fft -f 98e6 -A TX/RX -g 50 -s 10e6 -F
```



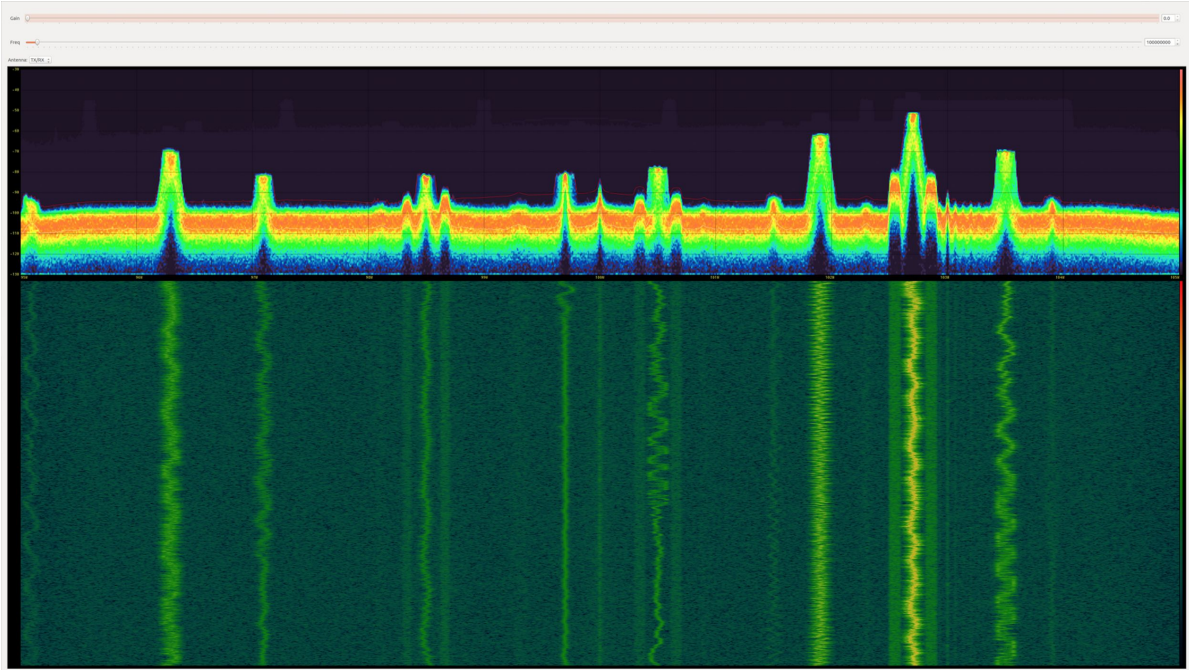
gr-fosphor - GNU Radio Block / Shortcuts

- Block within GNU Radio



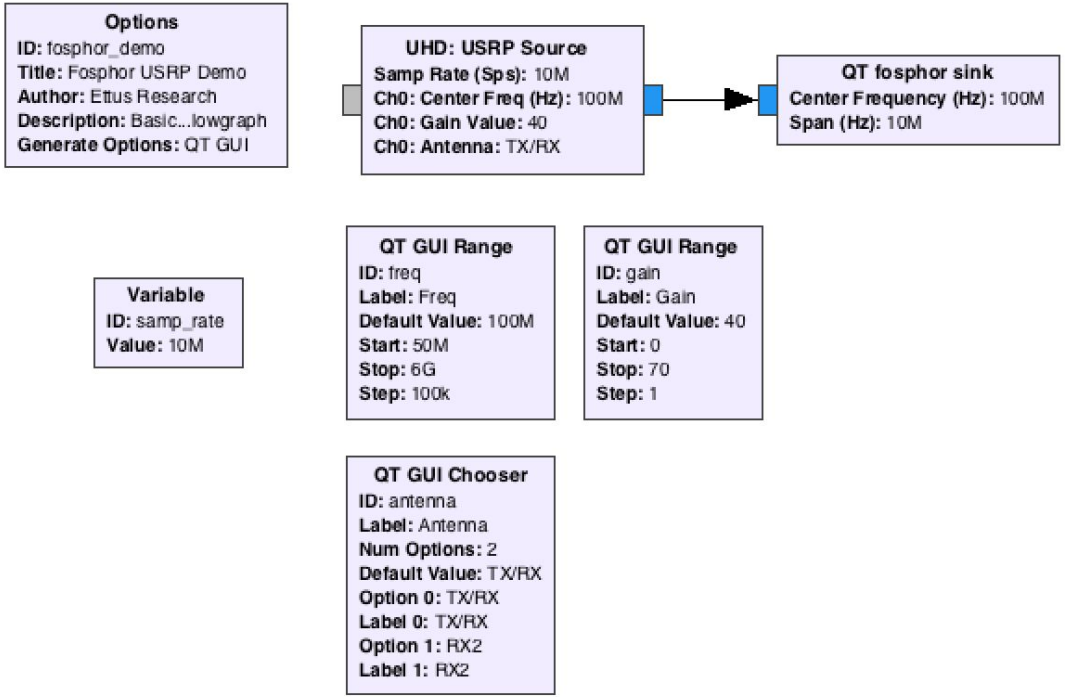
Shortcut Key Bindings for gr-fosphor

```
z:        toggle zoom mode
a/d:      move zoom frequency down/up
s/w:      adjust zoom width
q/e:      adjust screen split
           between waterfall and fft
space:    pause display

(left)/(right)   adjust dB/div
(up)/(down)      adjust ref level
```



gr-fosphor - Example Flowgraph

Location: ~/ettus_workshop/flowgraphs/fosphor.grc



Inspectrum

- Off-line spectrum analysis tool
- `https://github.com/miek/inspectrum`
- Spectrogram with zoom/pan
- Large (multi-gigabyte) file support

Install dependency: **`liquid-dsp`**

```
cd ~/workarea
git clone git://github.com/jgaeddert/liquid-dsp.git
cd liquid-dsp
./bootstrap.sh

# note Ubuntu 16.04 requires additional ./configure flags
CFLAGS="-march=native" ./configure --enable-fftoverride

# Ubuntu 14.04
./configure

make
sudo make install
sudo ldconfig
```

Reference: ~/ettus_workshop/instructions/install.html

Inspectrum

- Off-line spectrum analysis tool
- `https://github.com/miek/inspectrum`
- Spectrogram with zoom/pan
- Large (multi-gigabyte) file support

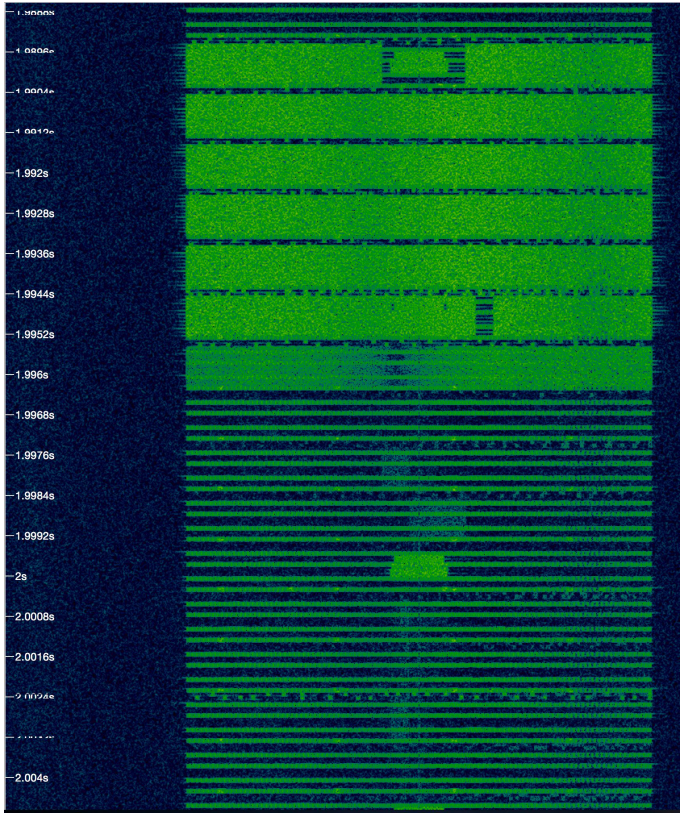
```
cd ~/workarea
sudo apt-get install qt5-default libfftw3-dev cmake pkg-config
git clone https://github.com/miek/inspectrum.git
cd inspectrum
mkdir build
cd build
cmake ..
make
sudo make install
```

Reference: ~/ettus_workshop/instructions/install.html

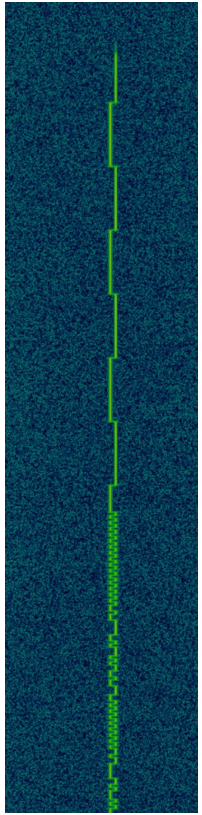
Inspectrum

(screenshots taken from older Inspectrum version which used vertical time display)

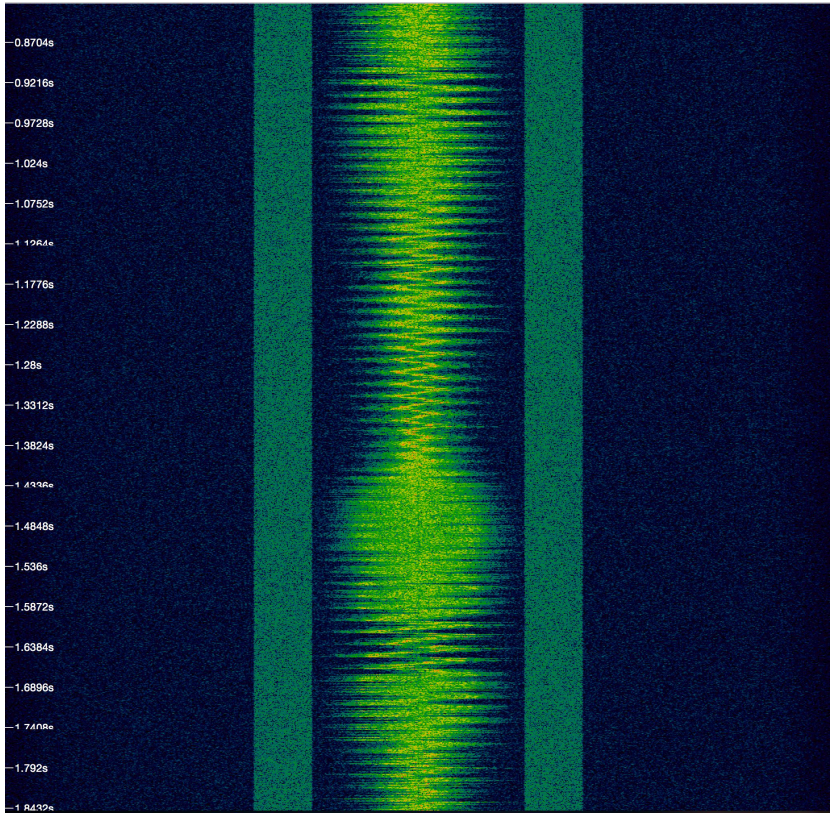
LTE



FSK



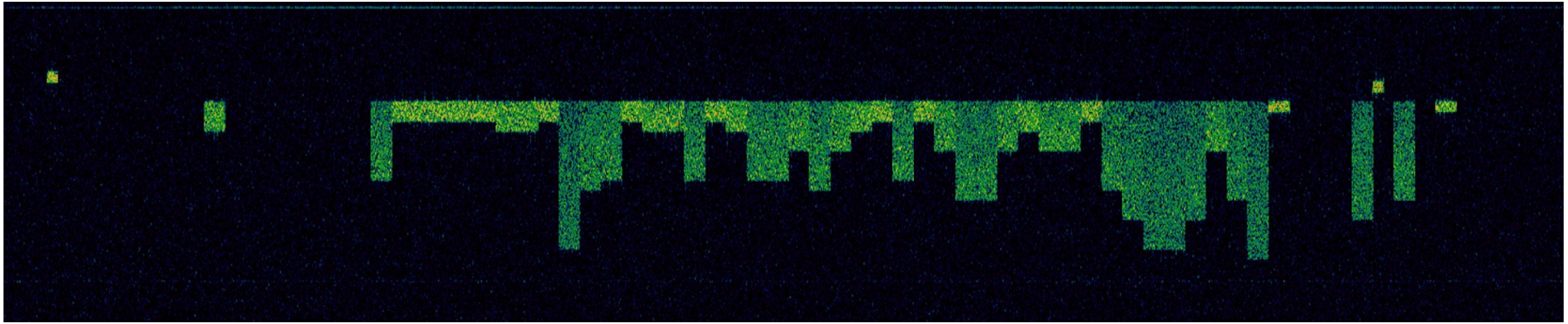
WBFM with OFDM HD Sidebands



Inspectrum

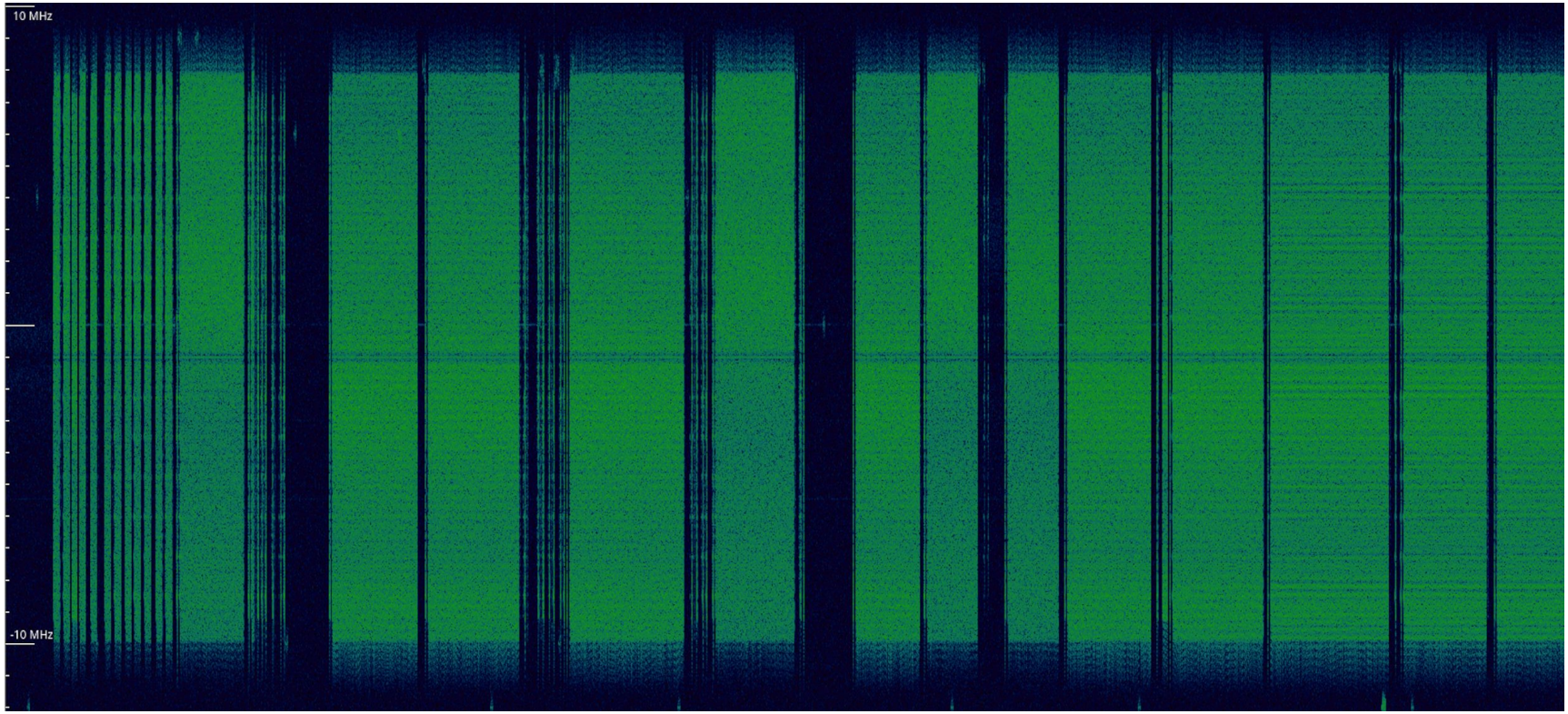
FSK

LTE



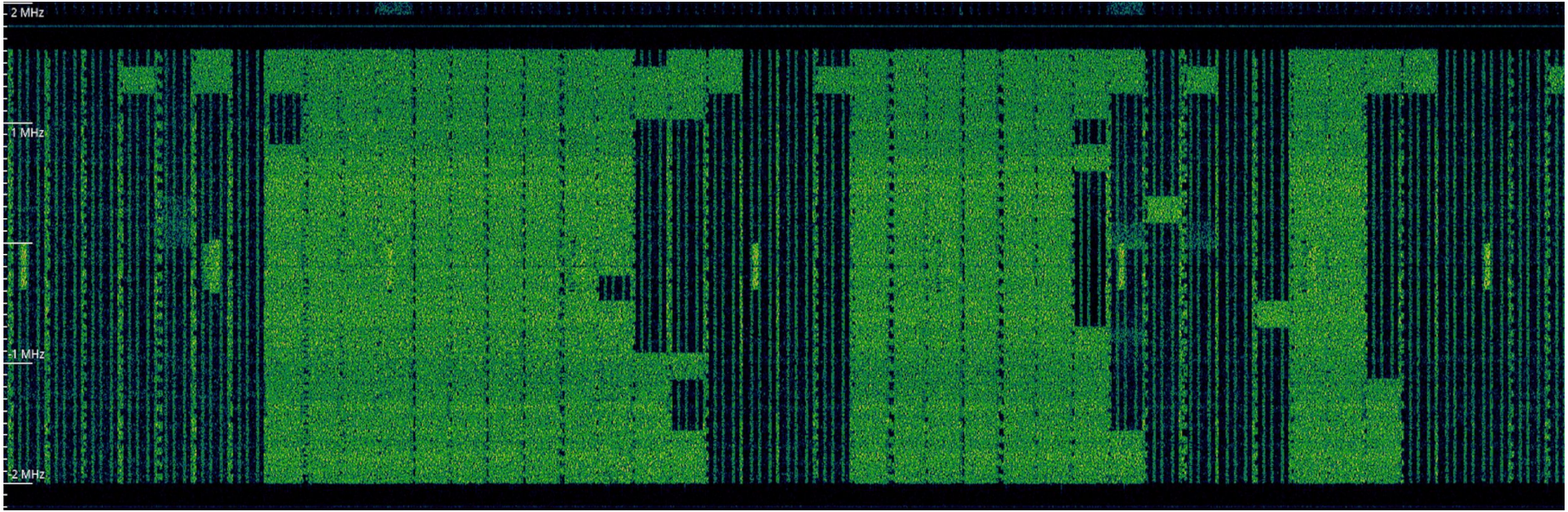
Inspectrum

802.11b WiFi (20 MHz)



Inspectrum

LTE



Demo - Remote Replay Attack

- Common in inexpensive, low power RF modules.
  - Wireless light switches / outlets, wireless thermometers, wireless doorbells, older/inexpensive garage door openers, wireless outdoor motion alarms, inexpensive RC toys, older keyfobs, shock collars, etc.
- Will typically run in 70cm ISM band (433.92 MHz) or 315MHz *(to avoid DC spike, use offset tuning and tune 200 KHz low)*
- Common modulations: ASK/OOK, FSK, some PSK
- Most newer keyfobs/garage door openers will use rolling codes, one time codes, or challenge-response authentication to mitigate

- First step is to find the signal.

```
$ osmocom_fft --args "type=b200" -A TX/RX -s 1e6 -f 433.72e6 -g 50 -F
```

Or use GQRX and tune to 433.72 MHz

Demo - Remote Replay Attack

- Recording Signal:

```
$ /usr/local/lib/uhd/examples/rx_samples_to_file \
   --args "type=b200" \
   --type float \
   --freq 433.72e6 \
   --rate 1e6 \
   --gain 0 \ # adjust gain based on distance from remote
   --ant TX/RX \
   --bw 1e6 \
   --file on.f32
```

Demo - Remote Replay Attack

- Replaying Signal:

```
$ /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 on.f32
```

Analyzing Captures - Unpacking Tarball

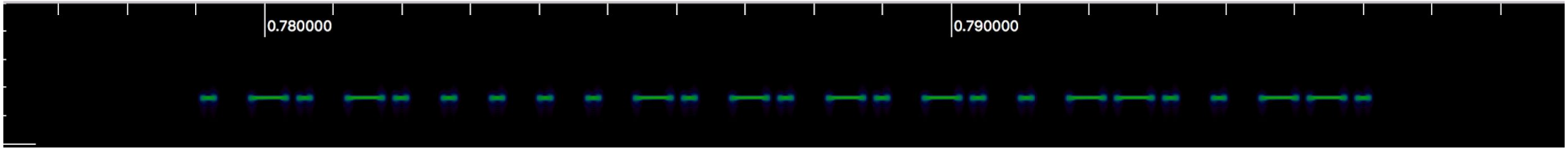
```
$ cd ~/ettus_workshop/captures/

$ ./unpack_tarballs.sh
```

Terminal

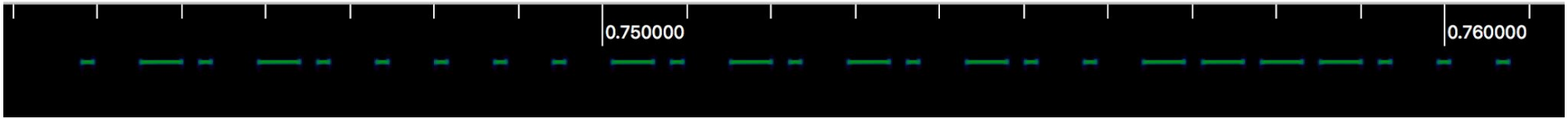
```
user@host:~/product-docs-master/ettus_workshop/captures$ ./unpack_tarballs.sh
fsk_250e3.f32
iridium_10e6.f32
LTE_12e6_filtered.f32
mexsat_600e3.f32
switch_off.f32
switch_on.f32
wifi_30e6.f32
user@host:~/product-docs-master/ettus_workshop/captures$
```

Inspectrum - Analyzing Captures

~/ettus_workshop/captures/switch_on.f32



~/ettus_workshop/captures/switch_off.f32



Inspectrum - Analyzing Captures

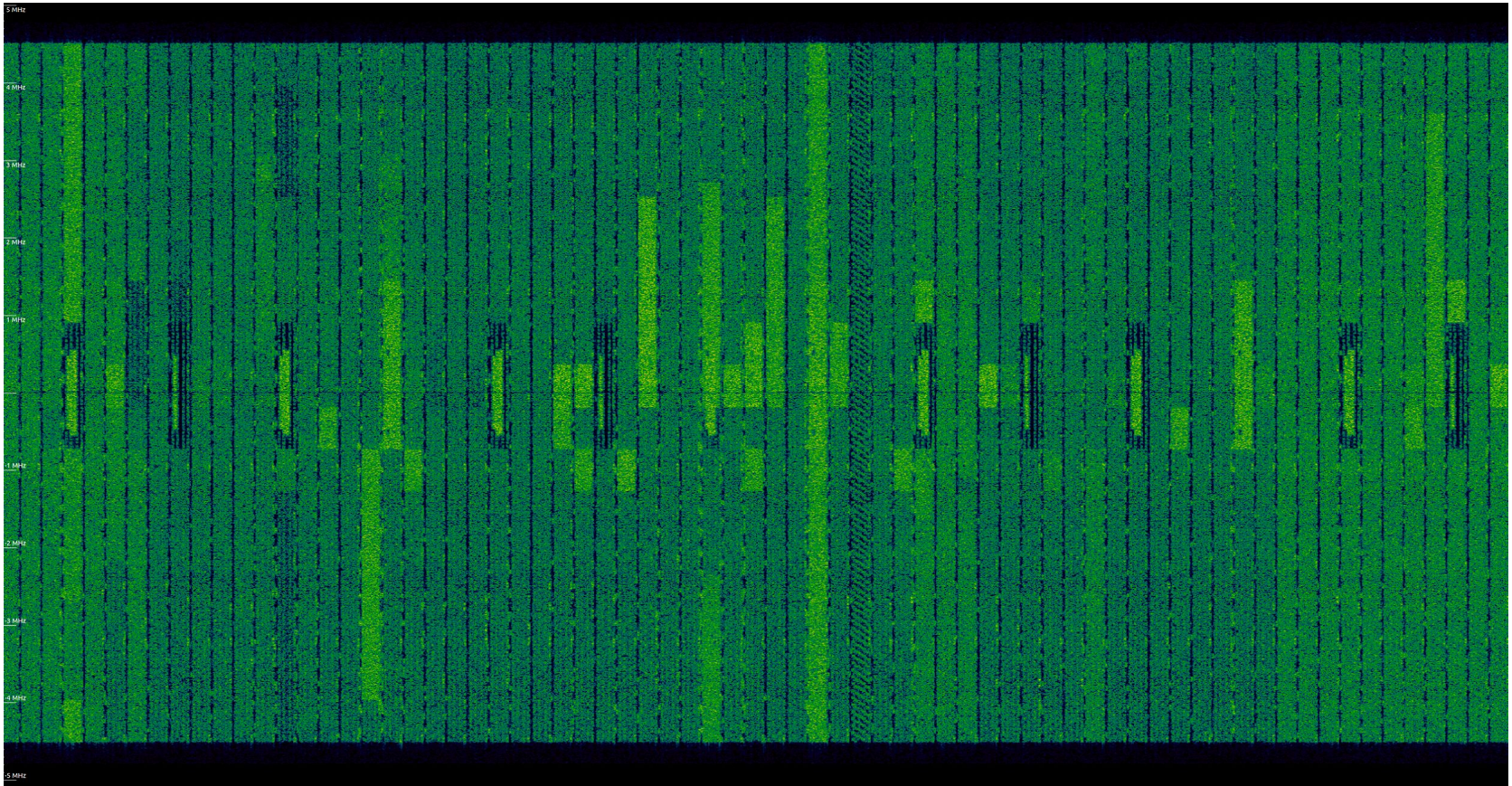
```
$ inspectrum --help
Usage: inspectrum [options] file
spectrum viewer

Options:
  -h, --help        Displays this help.
  -r, --rate <Hz>   Set sample rate.

Arguments:
  file              File to view.
```

```
$ inspectrum -r 1e6 ~/ettus_workshop/captures/turning_on.f32
$ inspectrum -r 1e6 ~/ettus_workshop/captures/turning_off.f32
```

Additional interesting capture files located in **~/ettus_workshop/captures/*.f32**
LTE_20e6.f32, wifi_30e6.f32, fsk_1e6.f32, wbfm_2e6.f32, Iridium_10e6.f32, mexsat_600e3.f32

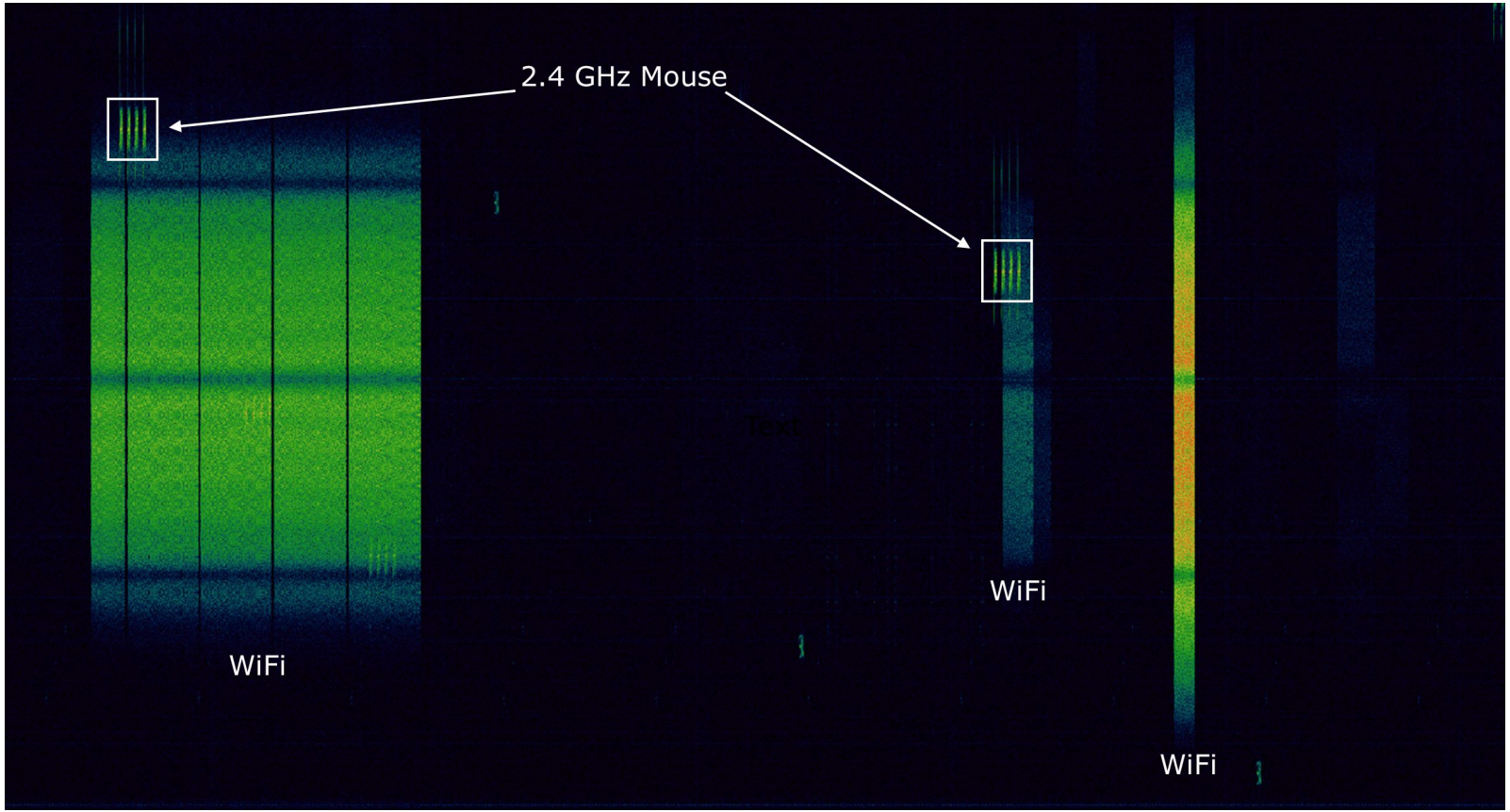
Inspectrum - Analyzing Captures

~/ettus_workshop/captures/LTE_12e6.f32



Inspectrum - Analyzing Captures

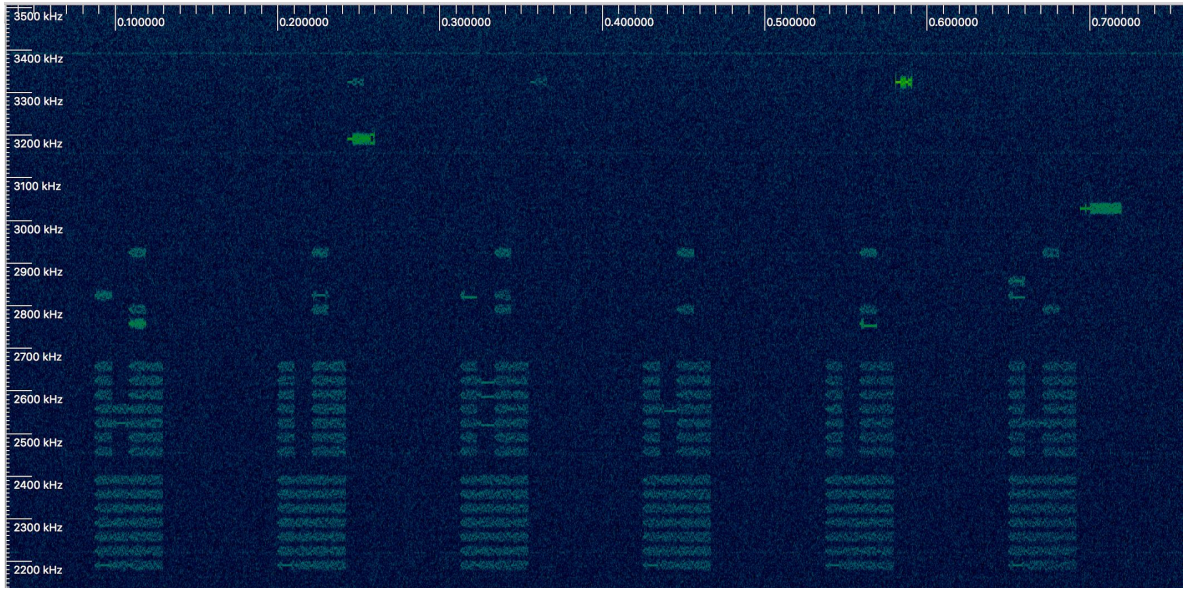
~/ettus_workshop/captures/wifi_30e6.f32

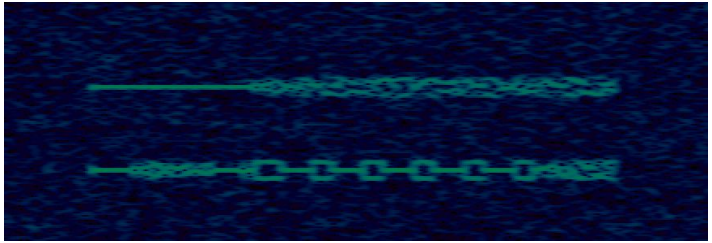


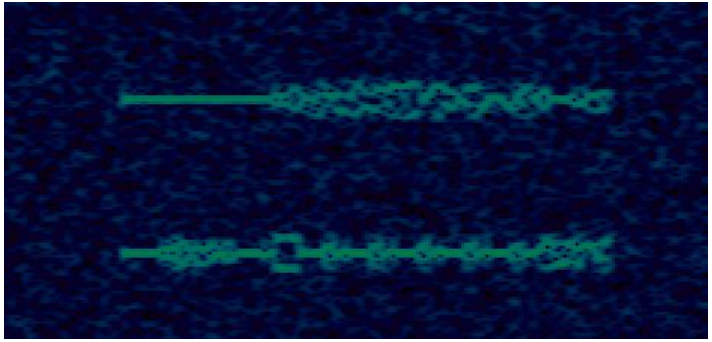
Inspectrum - Analyzing Captures

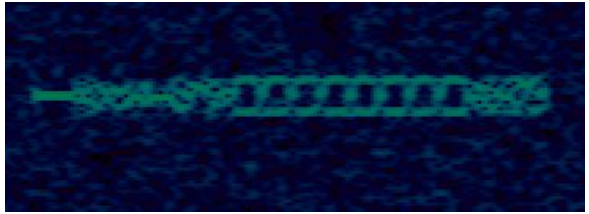
Ettus Research™
A National Instruments Company

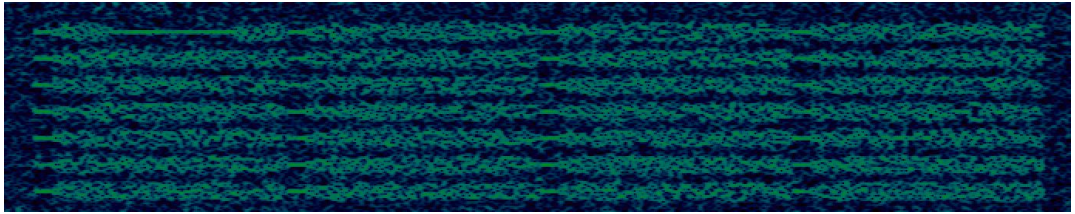
~/ettus_workshop/captures/iridium_10e6.f32





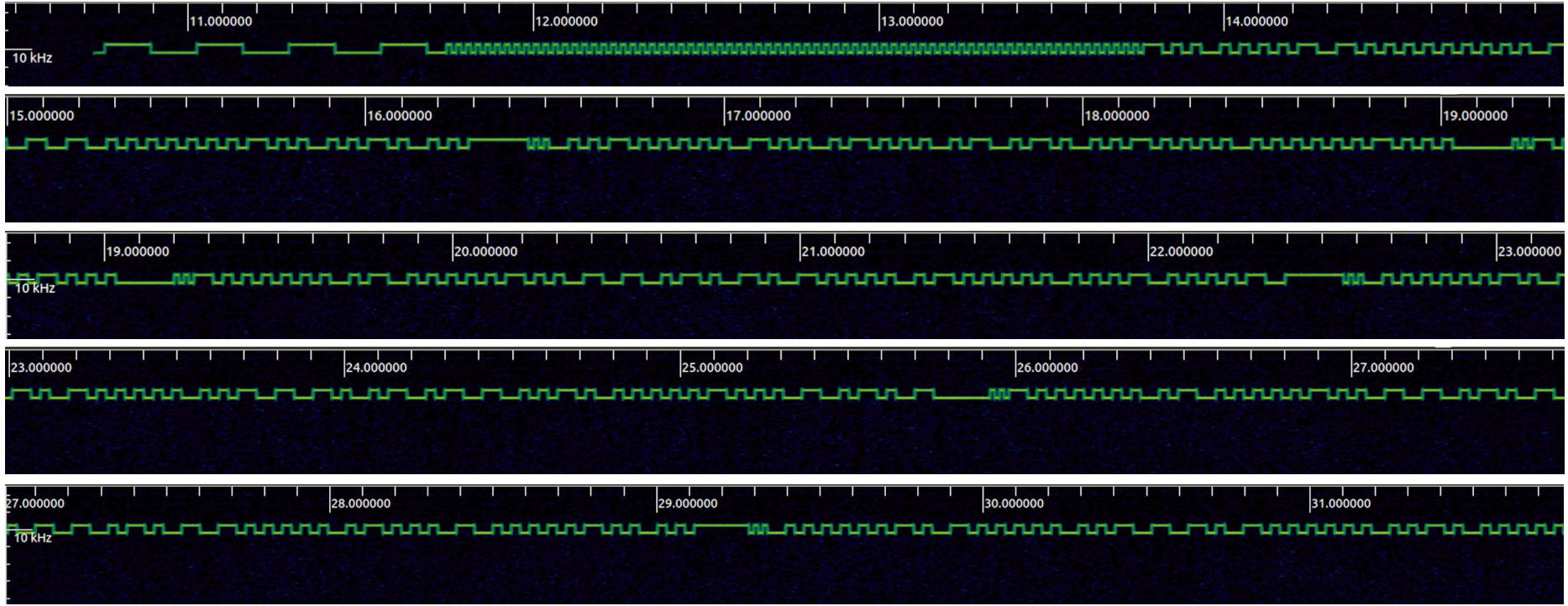






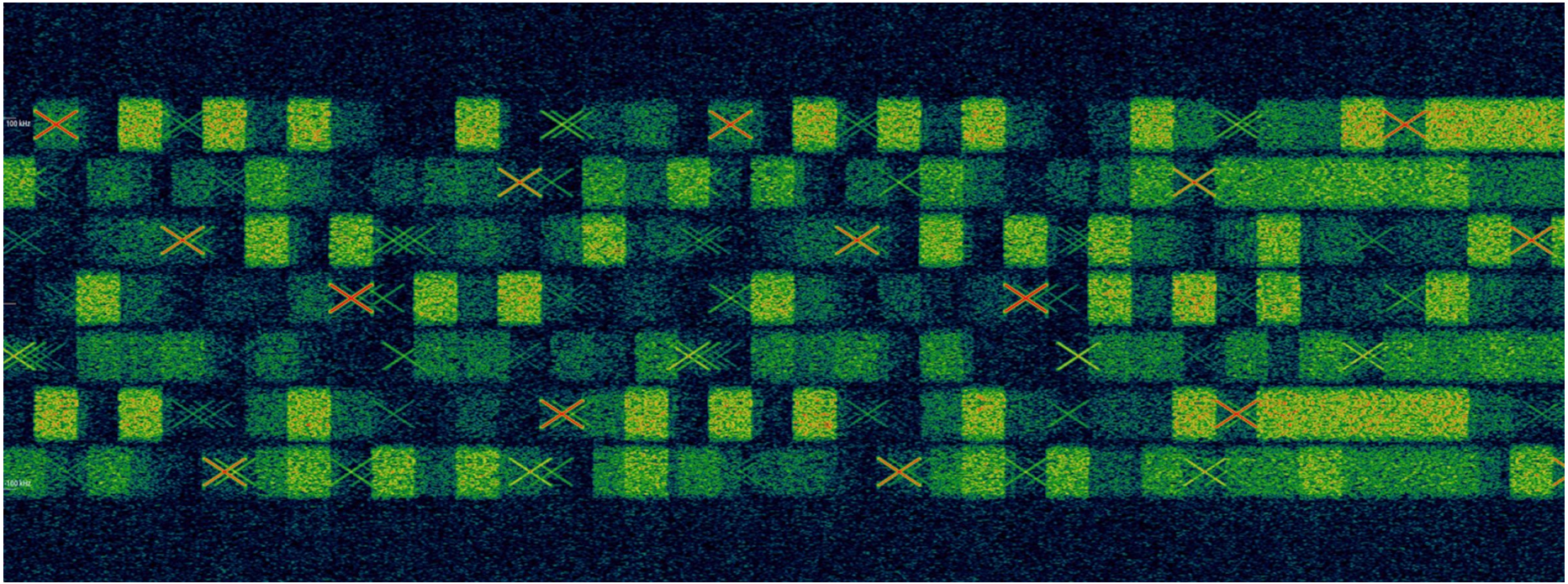
Inspectrum - Analyzing Captures

~/ettus_workshop/captures/fsk_250e3.f32



Inspectrum - Analyzing Captures

```
~/ettus_workshop/captures/mexsat_600e3.f32
"X" is Dual Chirp waveform for Synchronization
Overlap due to seeing multiple spot beams (each sat has 128 steerable beams)
```



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

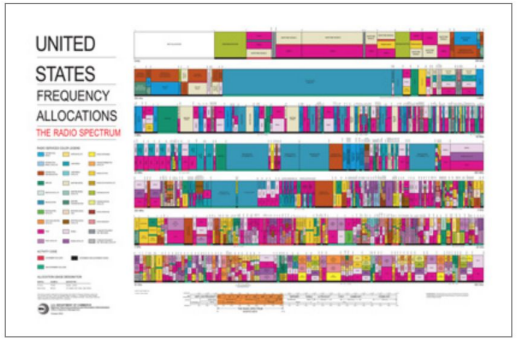
WELCOME TO SPECTRUMWIKI

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.

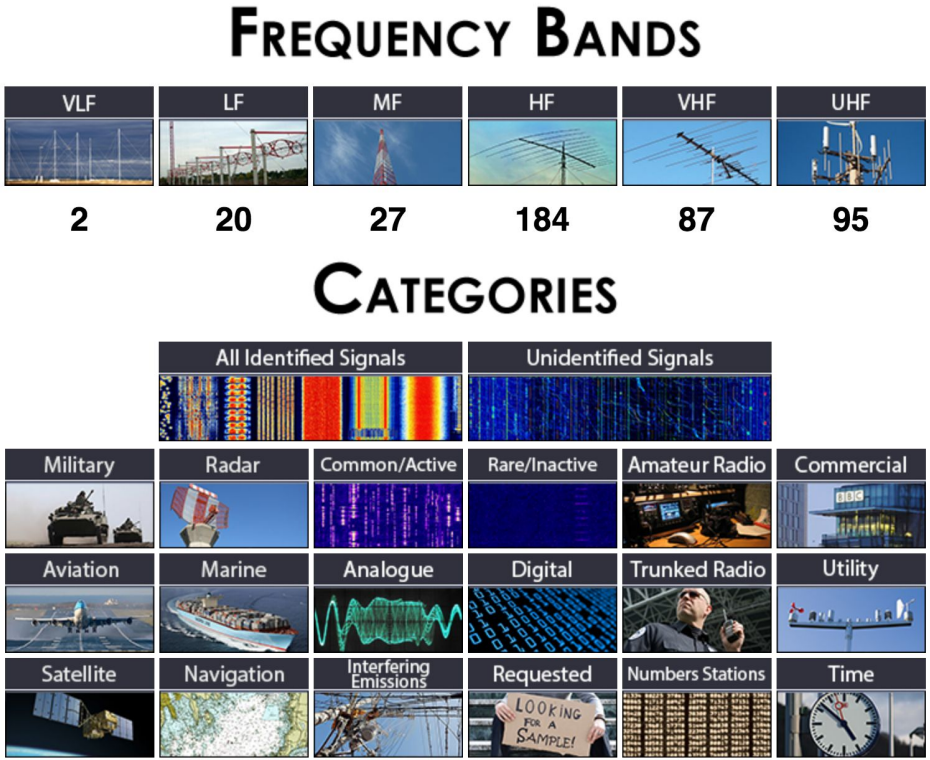
LATEST WIKI ENTRIES

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**



ADS-B / Mode-S Overview

Ettus Research™
A National Instruments Company

Automatic Dependent Surveillance – Broadcast (ADS–B) is a surveillance technology in which an aircraft determines its position via satellite navigation and periodically broadcasts it, enabling it to be tracked. The information can be received by air traffic control ground stations as a replacement for secondary radar. It can also be received by other aircraft to provide situational awareness and allow self separation.

ADS–B is "automatic" in that it requires no pilot or external input. It is "dependent" in that it depends on data from the aircraft's navigation system.

Implementation of ADS-B is mandatory in European airspace as well as in Australia. North American implementation is still voluntary, with a mandate arriving in 2020 via the FAA's "NextGen" program.

ADS-B-equipped aircraft broadcast ("squitter") their position, velocity, flight number, and other interesting information to any receiver within range of the aircraft. Position reports are typically generated once per second and flight identification every five seconds.

Mode S is the transponder protocol used in modern commercial aircraft. A Mode S-equipped aircraft replies to radar interrogation by either ground radar (secondary surveillance) or other aircraft ("Traffic Collision Avoidance System", or TCAS).

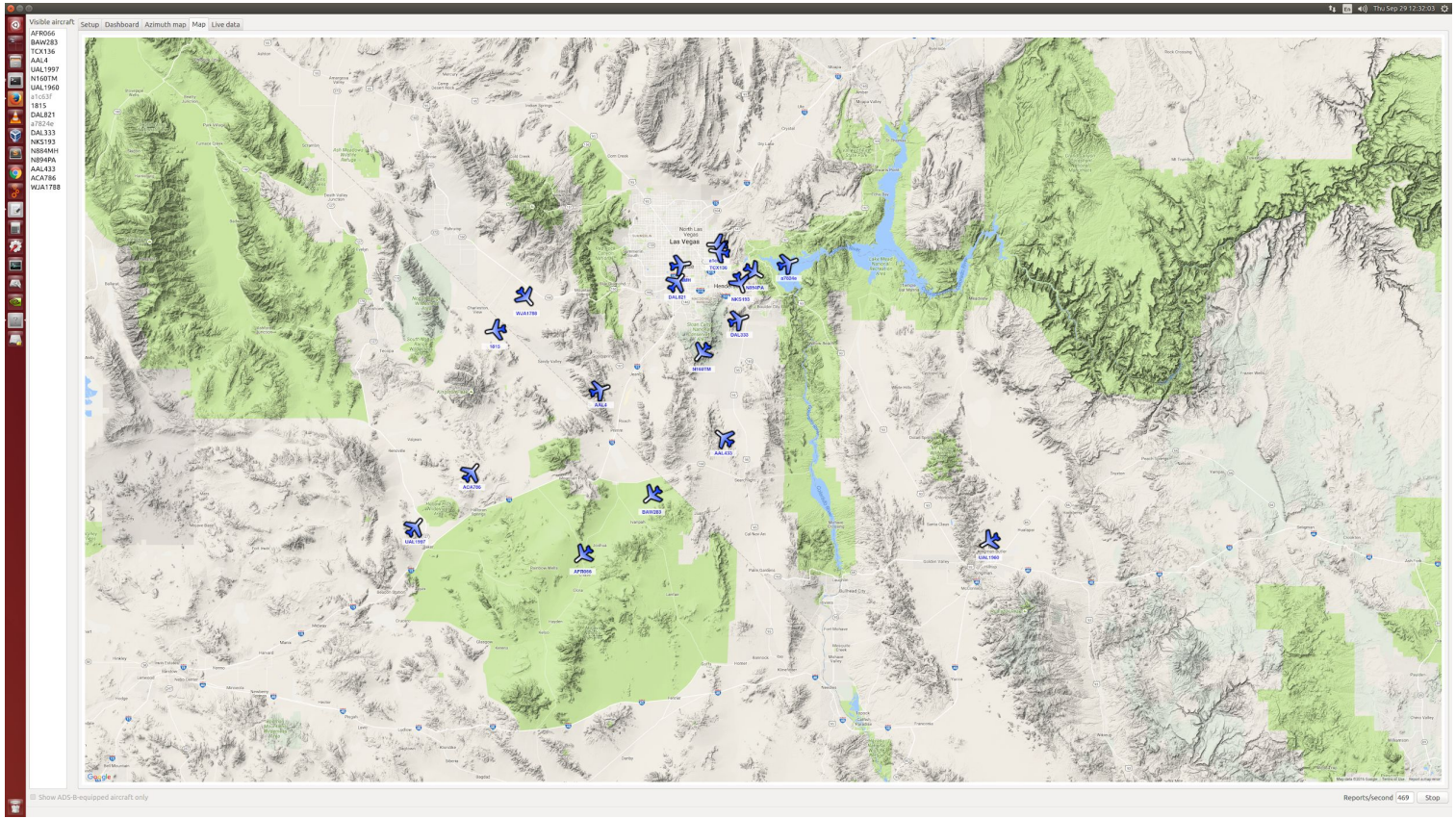
ADS-B / Mode-S Overview

- Operates at 1090 MHz
- PPM Modulation (Pulse Position Modulation)
- Packets are 10 bytes long
- Packets may contain:
    - GPS position (latitude, longitude)
    - Pressure
    - Altitude
    - Callsign
    - Ground track
    - Ground speed

gr-air-modes

- Created by Nick Foster
- https://github.com/bistromath/gr-air-modes
- Raw (or minimally processed) output of packet data ADS-B / Mode-S / TCAS
- Parsed text
- SQLite database backend
- Built-in Google Maps Display
- KML for use with Google Earth
- SBS-1-compatible output for use with e.g. PlanePlotter or Virtual
- Radar Server
- FlightGear multiplayer interface for real-time display of traffic within the simulator

Demo - gr-air-modes



E310, E312, E313 - Device Overview

- Embedded Architecture
- Open Embedded Linux OS
- Zynq 7020 FPGA with ARM Cortex A9 dual-core CPU
- Based on Analog Devices AD9361 Transceiver
    - 2x2 MIMO, 50 MHz - 6 GHz, 61.44 MS/s, 56 MHz bandwidth
- TX and RX filter banks
- Integrated uBlox GPS receiver
- Supports UHD, GNU Radio, RFNoC
- E312 is an E310 with battery
- E313 is an E310 with IP67 chassis and PoE

E310, E312, E313 - Embedded Architecture

The USRP E3xx is designed to process data locally on the Zynq 7020 FPGA and Cortex ARM CPU.

- Data rate from the AD9361 transceiver to FPGA is capable of 61.44 MS/s
- Data rate into Cortex ARM CPU is about 12 MS/s
- Data rate over Ethernet interface to external host is about 2 or 3 MS/s

Network Mode

- Streams complex samples directly over Ethernet interface to external host, similar to N2xx and X3xx
- Should be used as "debugging mode", and not meant for production use

E310, E312, E313 - SD Card Images

- Contains the Linux OpenEmbedded OS and FPGA image
- **`http://files.ettus.com/e3xx_images/`**
- Release 4 is the current release (includes UHD 3.9.2 and GNU Radio 3.7.9)
- Release 5 is coming soon (includes UHD 3.10.1.1 and GNU Radio 3.7.10.2)
- SG1 = Speed Grade 1 (667 MHz ARM CPU)
- SG3 = Speed Grade 3 (866 MHz ARM CPU)

- Images are compressed with “xz”. Use either **“`xz`”** or **“`xzdec`”** utilities to decompress

```
$ xzdec e300.direct.xz
```

- Write image to SD card either with **“`dd`”** or **“`bmaptool`”** utilities

```
$ sudo dd if=e300.direct of=/dev/sdX
$ sudo bmaptool copy e300.direct.xz /dev/sdX --nobmap
```

Additional details can be found in the on the USRP E300 Embedded Hardware Resources page within the Ettus Research Knowledge Base.

`https://kb.ettus.com/Ettus_USRP_E300_Embedded_Family_Hardware_Resources#SD_Card_Images`

E310, E312, E313 - WiFi USB

- A USB WiFi dongle may be used with the E310
- The Edimax EW-7811Un N150 works out-of-the-box
- Any WiFi dongle with kernel support should work out-of-the-box
- Generate PSK for wireless network:

```
$ wpa_passphrase <SSID> >> /etc/wpa_supplicant.conf
```

- Edit the file **/etc/wpa_supplicant.conf** to match below, updating any missing values:

```
network={
        ssid="YOUR_SSID"
        psk=HASH_VALUE
        key_mgmt=WPA-PSK
        proto=RSN WPA
        pairwise=CCMP TKIP
        group=CCMP TKIP
}
```

E310, E312, E313 - WiFi USB

- Start **`wpa_supplicant`** with the specified configuration file:

```
$ wpa_supplicant -B -D nl80211 -i wlan0 -c /etc/wpa_supplicant.conf
```

- Request DHCP address with the command below:

```
$ udhcpc -i wlan0
```

- Verify an IP address has been assigned:

```
$ ip a
```

E310, E312, E313 - Development

Notes on Cross-Compiling and SDK (Software Development Kit)

- SDKs contain the compiler toolchain and libraries for the embedded device.
- The SDK enables you to compile and link on one platform, but target another platform
- What you build on the external host (development machine) will run on the embedded E310 host (production machine)
- Cross-compiling is recommended due to the limited CPU processing power and memory on E3xx
- PyBOMBS (Python Build Overlay Managed Bundle System) includes a recipe that automates the installation of all the elements necessary to cross-compile for the E3XX.

Additional details and step-by-step Application Note within the Ettus Research Knowledge Base:

`https://kb.ettus.com/Software_Development_on_the_E310_and_E312`

E310, E312, E313 - Development

- SSHFS (SSH Filesystem) is a filesystem client to mount and interact with directories and files located on a remote server or workstation over a normal ssh connection.

- Example:

```
$ sshfs user@host:/mountpoint/ local_mount_point/
```

- Useful to use to transfer files to/from E31x from host computer

E310, E312, E313 - Cross Compiling

On Host Machine:

Ensure “Bash” is set as your default shell. Some version of Ubuntu will default to “Dash”

```
sudo dpkg-reconfigure dash
```

"Select No"

Verify **`/bin/sh`** is pointing to **`/bin/bash`**

```
ll /bin/sh
```

Create working directory on Host machine:

```
mkdir -p ~/e300
```

E310, E312, E313 - Cross Compiling

Download Open Embedded SDK

- Example name: "**oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh**"
- Located in the **/e3xx-release-xx/** directories on **files.ettus.com**
- **http://files.ettus.com/e3xx_images/e3xx-release-4/**

```
wget http://files.ettus.com/e3xx_images/e3xx-release-4/oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
```

Install the Open Embedded SDK

```
bash oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
```

Enter your working directory "**~/e300**" for the target directory when prompted.

SDK installation may take a few minutes.

When completed, change into the **~/e300** directory, and **source** the SDK Environment file:

```
cd ~/e300
source ./environment-setup-armv7ahf-vfp-neon-oe-linux-gnueabi
```

E310, E312, E313 - Cross Compiling

Verify compiler is setup correctly:

```
echo $CC
```

Example output:

```
arm-oe-linux-gnueabi-gcc -march=armv7-a -mfloat-abi=hard -mfpu=neon
--sysroot=/home/demo/e300/sysroots/armv7ahf-vfp-neon-oe-linux-gnueabi
```

Next, create a **`src/`** directory within the **`~/e300`** directory:

```
mkdir src
cd src
```

Verify you're in the correct directory with **`pwd`** command:

```
pwd
```

Expected output:

```
/home/demo/e300/src
```

E310, E312, E313 - Cross Compiling

Next download the UHD source code:

```
git clone https://github.com/EttusResearch/uhd.git
```

Change into the cloned directory:

```
cd uhd
```

Optional: Checkout specific version of UHD:

```
git checkout release_003_009_005
```

Change into the `host` directory:

```
cd host/
```

Create `build` directory and change into it:

```
mkdir build && cd build
```

E310, E312, E313 - Cross Compiling

Ettus Research™
A National Instruments Company

Next, run the CMake command:

```
cmake -DCMAKE_TOOLCHAIN_FILE=../host/cmake/Toolchains/oe-sdk_cross.cmake
-DCMAKE_INSTALL_PREFIX=/usr -DENABLE_E300=ON ..
```

Note the additional **`CMAKE_TOOLCHAIN_FILE`** option.

Note: **`CMAKE_INSTALL_PREFIX`** points to **`/usr`**

- This references the **`$DESTDIR/usr`** location on the E3xx, not the host machine.

Note: **`ENABLE_E300=ON`**

- This enabled E3xx specific functionality within UHD

Next, run the **`make`** command to build. This example uses 4 cores of the host machine to build.

```
make -j4
```

Next, install to the destination directory, note the **`DESTDIR`** flag, this will install it into our working directory.

```
make install DESTDIR=~/e300
```

E310, E312, E313 - Cross Compiling

Next you will need to create an environment setup file. This will point important system variables to the correct path.

Create the file **`setup_env`** within the **`~/e300`** folder on your host machine:

```
cd ~/e300
touch setup_env
nano setup_env
```

File contents, note the **`LOCALPREFIX`** is set to **`~/newinstall/usr`**:

```
LOCALPREFIX=~/newinstall/usr
export PATH=$LOCALPREFIX/bin:$PATH
export LD_LOAD_LIBRARY=$LOCALPREFIX/lib:$LD_LOAD_LIBRARY
export LD_LIBRARY_PATH=$LOCALPREFIX/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$LOCALPREFIX/lib/python2.7/site-packages:$PYTHONPATH
export PKG_CONFIG_PATH=$LOCALPREFIX/lib/pkgconfig:$PKG_CONFIG_PATH
export GRC_BLOCKS_PATH=$LOCALPREFIX/share/gnuradio/grc/blocks:$GRC_BLOCKS_PATH
export UHD_RFNOC_DIR=$LOCALPREFIX/share/uhd/rfnoc/
export UHD_IMAGES_DIR=$LOCALPREFIX/share/uhd/images
```

E310, E312, E313 - Cross Compiling

Open a new Terminal window, and SSH into the E3xx:

```
ssh root@192.168.10.2
```

Verify you're within the root home directory:

```
root@ettus-e3xx-sg3:~# pwd
```

Expected output:
/home/root

Create **~/newinstall** directory on the E3xx:

```
root@ettus-e3xx-sg3:~# mkdir newinstall
```

Next, mount the **~/e300** directory [from your host machine] on to the E3xx with SSHFS, to the **~/newinstall** location. Update the username and IP address to match the host configuration:

```
root@ettus-e3xx-sg3:~# sshfs username@192.168.10.5:e300/ newinstall/
```

E310, E312, E313 - Cross Compiling

Verify that the **~/e300** on the host machine has been successfully mounted on the E3xx:

```
root@ettus-e3xx-sg3:~# ls ~/newinstall
environment-setup-armv7ahf-vfp-neon-oe-linux-gnueabi  src
version-armv7ahf-vfp-neon-oe-linux-gnueabi
setup_env                                  sysroots
site-config-armv7ahf-vfp-neon-oe-linux-gnueabi       usr
```

Next, determine the current UHD which is being used:

```
root@ettus-e3xx-sg3:~# which uhd_find_devices
/usr/bin/uhd_find_devices
```

Next, **source** the **setup_env** file to re-configure the E3xx to use the newly built UHD version:

```
root@ettus-e3xx-sg3:~# cd ~/newinstall
root@ettus-e3xx-sg3:~# source ./setup_env
```

E310, E312, E313 - Cross Compiling

Verify that you're now using the newly built UHD version:

```
root@ettus-e3xx-sg3:~/newinstall#  which uhd_find_devices
/home/root/newinstall/usr/bin/uhd_find_devices
```

Next, you will need to download the corresponding FPGA images for the newly built UHD. If your E3xx is not on an internet connected network, this can be a multiple step process.

Start with trying to run the **`uhd_images_downloader`** utility:

```
root@ettus-e3xx-sg3:~/newinstall#  cd ~/
root@ettus-e3xx-sg3:~#  uhd_images_downloader
```

Note: If you have internet access configured on the network your E3xx is on, this command will be successful. If it is not connected to a network with internet it will fail.

E310, E312, E313 - Cross Compiling

Example of run without internet access:

```
root@ettus-e3xx-sg3:~# uhd_images_downloader
UHD_IMAGES_DIR environment variable is set.
Default install location: /home/root/newinstall/usr/share/uhd/images
Images destination:      /home/root/newinstall/usr/share/uhd/images
Downloading images from: http://files.ettus.com/binaries/images/uhd-images_003.009.005-release.zip
Downloading images to:   /tmp/tmptpuiEg/uhd-images_003.009.005-release.zip
Downloader raised an unhandled exception: ('Connection aborted.', gaierror(-2, 'Name or service
not known'))
You can run this again with the '--verbose' flag to see more information
If the problem persists, please email the output to: support@ettus.com
```

Note: The URL of the UHD FPGA Zip file it is attempting to download.
Copy this URL and fetch it on your host machine (which should have internet access):

* Note this command is ran on your host machine:

```
$ cd ~/e300/src
$ wget http://files.ettus.com/binaries/images/uhd-images_003.009.005-release.zip
```

E310, E312, E313 - Cross Compiling

Ettus Research™
A National Instruments Company

Next you will need to decompress the FPGA archive.

```
$ unzip uhd-images_003.009.005-release.zip
```

Decompressing the FPGA Zip file will create a multi-level folder structure. The contents of the images/ folder need to be moved to the images/ location of your new UHD build. Note: You may need to create the **`uhd/images/`** folder.

```
$ cd ~/e300/usr/share
$ mkdir -p uhd/images
$ cd uhd/images
```

Verify you're in the correct directory:

```
$ pwd
/home/demo/e300/usr/share/uhd/images
```

Move the decompressed files to your current directory:

```
$ mv -v ~/e300/src/uhd-images_003.009.005-release/share/uhd/images/* .
```

E310, E312, E313 - Cross Compiling

Next, return to your E3xx terminal window, and verify that you’re able to use the FPGA images by running:

```
root@ettus-e3xx-sg3:~#  uhd_usrp_probe
linux; GNU C++ version 4.9.2; Boost_105700; UHD_003.009.005-0-g32951af2

-- Loading FPGA image: /home/root/newinstall/usr/share/uhd/images/usrp_e310_fpga_sg3.bit... done
-- Initializing core control...
-- Performing register loopback test... pass
-- Performing register loopback test... pass
-- Performing register loopback test... pass
-- Performing CODEC loopback test... pass
-- Performing CODEC loopback test... pass
```

Within the output you should note the path of the FPGA image that is being loaded is from your new installation.

E310, E312, E313 - Cross Compiling

Upon running **`uhd_usrp_probe`** successfully, you can then run the compiled UHD example programs by navigating to the **`examples/`** directory of the new build on the E3xx:

```
root@ettus-e3xx-sg3:~# cd ~/newinstall/usr/lib/uhd/examples/

./rx_samples_to_file --freq 100e6 --gain 0 --ant TX/RX --rate 1e6 --null
```

Press **`CTRL+C`** to stop the program after it has ran for a moment.

Note the version string that is printed, this should match the version you have checked out and compiled.

At this point, if you created a tarball of the **`~/e300/usr`** directory and transfer it to the E3xx, untar it locally, and then update the **`setup_env`** file's paths, you will have a freshly created UHD installation without the need to use SSHFS. SSHFS is useful for quick testing and debugging while building.

E310, E312, E313 - Cross Compiling

Next we will cross compile an independent version of the UHD example program **`'rx_samples_to_file'`** for the E3xx.

First start by navigating to the **`~/e300`** directory on your host machine and create a working space:

```
cd ~/e300
mkdir rx_samples
cd rx_samples
```

Next, we will need to copy files from the UHD repo and create a Toolchain folder for the OE SDK Cross Compile CMake instructions file.

```
mkdir -p cmake/Toolchains
cp ~/e300/src/uhd/host/cmake/Toolchains/oe-sdk_cross.cmake cmake/Toolchains/
cp ~/e300/src/uhd/host/examples/rx_samples_to_file.cpp my_rx_samples_to_file.cpp
cp ~/e300/src/uhd/host/examples/init_usrp/CMakeLists.txt .
```

E310, E312, E313 - Cross Compiling

Verify your directory structure matches by running the **`'tree'`** command:

```
$ tree
.
├── cmake
│   └── Toolchains
│       └── oe-sdk_cross.cmake
├── CMakeLists.txt
└── my_rx_samples_to_file.cpp

2 directories, 3 files
```

E310, E312, E313 - Cross Compiling

The first action item is to modify the **`CMakeLists.txt`** file to reference **`'my_rx_samples_to_file'`**

```
nano CMakeLists.txt
```

Update all occurrences of **`'init_usrp'`** to be **`'my_rx_samples_to_file'`**

Quick tip, use the Linux utility **`'sed'`** to replace text inline of a file:

```
sed -i "s/init_usrp/my_rx_samples_to_file/g" CMakeLists.txt
```

```
### 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.
```

E310, E312, E313 - Cross Compiling

Next, modify the **`my_rx_samples_to_file.cpp`** file to add additional text to display that it is our locally built version:

```
nano my_rx_samples_to_file.cpp
```

Locate the section (Near Line 209):

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

After the **`uhd::set_thread_priority_safe();`** line, add the code:

```
std::cout << std::endl << "This is my locally built version of
rx_samples_to_file!!" << std::endl;
```

Exit **`nano`** and save the modified file.

E310, E312, E313 - Cross Compiling

Next, we will build this version of **`my_rx_samples_to_file`**.

On the host machine, run the commands:

```
mkdir build-arm
cd build-arm

cmake -Wno-dev -DCMAKE_TOOLCHAIN_FILE=../cmake/Toolchains/oe-sdk_cross.cmake
-DCMAKE_INSTALL_PREFIX=/usr -DUHD_DIR=~/e300/usr/lib/cmake/uhd/
-DUHD_INCLUDE_DIRS=~/e300/usr/include -DUHD_LIBRARIES=~/e300/usr/lib/libuhd.so
../

make
```

Note: If you encounter an error:

```
/home/demo/e300/usr/lib/libuhd.so: error adding symbols: File in wrong format
```

Make sure you have sourced the SDK Environment file:

```
source ~/e300/environment-setup-armv7ahf-vfp-neon-oe-linux-gnueabi
```

E310, E312, E313 - Cross Compiling

Next, return to the terminal window connect to your E3xx. Navigate to the **`build-arm`** directory:

```
root@ettus-e3xx-sg3:~#  cd ~/newinstall/rx_samples/build-arm/
```

Run your **`my_rx_samples_to_file`** program. Note, you will see the additional text that you added in your **`stdout`**:

```
# ./my_rx_samples_to_file --freq 1e9 --rate 1e6 --gain 0 --ant TX/RX --null
linux; GNU C++ version 4.9.2; Boost_105700; UHD_003.009.005-0-g32951af2

This is my locally built version of rx_samples_to_file!!

Creating the usrp device with: ...
-- Loading FPGA image:
/home/root/newinstall/usr/share/uhd/images/usrp_e310_fpga_sg3.bit... done
-- Initializing core control...
-- Performing register loopback test... pass
```

E310, E312, E313 - Linux Commands

To unmount an SSHFS attached folder:

```
root@ettus-e3xx-sg3:~# umount ~/newinstall/
```

Find Release information:

```
root@ettus-e3xx-sg3:~# cat /etc/version
201601141734

root@ettus-e3xx-sg3:~# cat /etc/version-image
Timestamp: 201601150140
Release: Release-4
Image: gnuradio-demo-image
```

E310, E312, E313 - Linux Commands

To configure a static IP address edit

/etc/network/interfaces

to look like

```
auto eth0
iface eth0 inet static
    address your-ip
    netmask your-netmask
    gateway your-gateway
```

E310, E312, E313 - Linux Commands

```
root@ettus-e3xx-sg3:~# cat /proc/cpuinfo
processor: 0
model name    : ARMv7 Processor rev 0 (v7l)
Features  : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpd32
CPU implementer    : 0x41
CPU architecture: 7
CPU variant   : 0x3
CPU part  : 0xc09
CPU revision  : 0

processor: 1
model name    : ARMv7 Processor rev 0 (v7l)
Features  : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpd32
CPU implementer    : 0x41
CPU architecture: 7
CPU variant   : 0x3
CPU part  : 0xc09
CPU revision  : 0

Hardware  : Xilinx Zynq Platform
Revision  : 0000
Serial        : 0000000000000000
```

E310, E312, E313 - ARM NEON

The E310/E312 ARM CPU includes NEON support

ARM NEON is a general-purpose SIMD engine that efficiently processes current and future multimedia formats, enhancing the user experience.

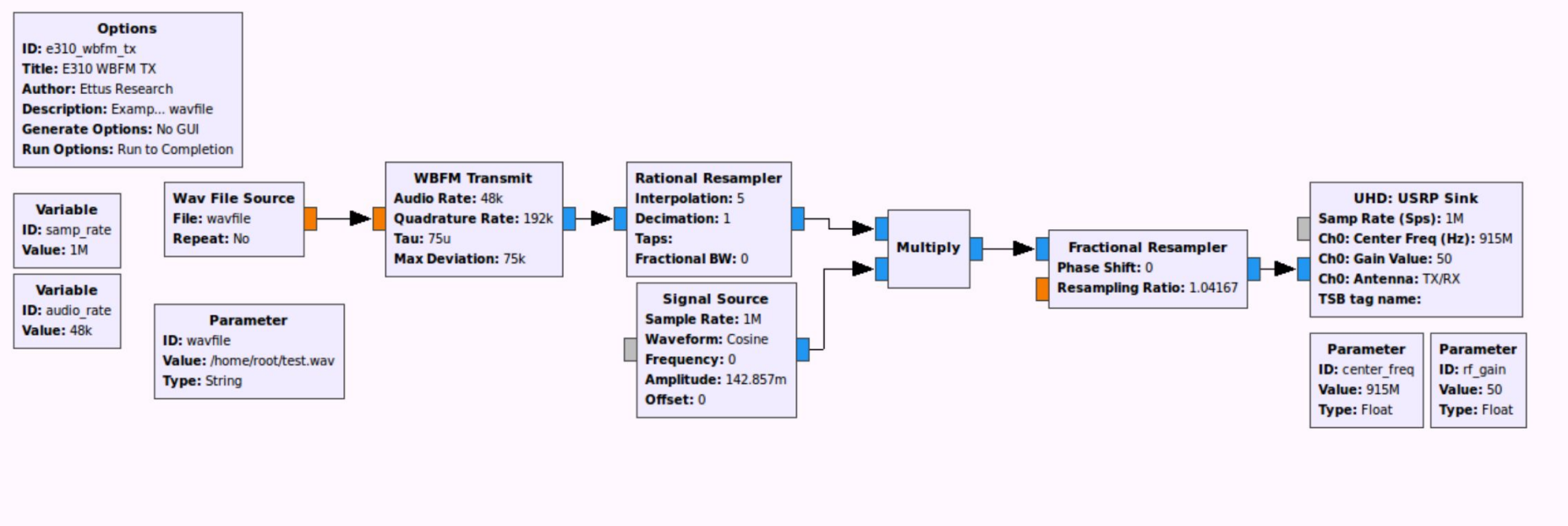
NEON technology can accelerate multimedia and signal processing algorithms such as video encode/decode, 2D/3D graphics, gaming, audio and speech processing, image processing, telephony, and sound synthesis by at least 3x the performance of ARMv5 and at least 2x the performance of ARMv6 SIMD.

NEON technology is a 128-bit SIMD (Single Instruction, Multiple Data) architecture extension for the ARM Cortex-A series processors, designed to provide flexible and powerful acceleration for consumer multimedia applications, delivering a significantly enhanced user experience. It has 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide.

Additional Information on ARM NEON:
https://www.arm.com/products/processors/technologies/neon.php

E31x - WBFM TX Demo

Location: ~/ettus_workshop/flowgraphs/e310/e310_wbfm_tx.grc



E31x - WBFM TX Demo

Generate Python file from flowgraph

Copy Python file to E31x

```
$ scp ~/ettus_workshop/flowgraphs/e310_wbfm_tx.py root@192.168.10.2:~/.
```

Copy audio file to E31x

```
$ scp ~/ettus_workshop/sources/audio2.wav root@192.168.10.2:~/.
```

SSH to E31x

```
$ ssh root@192.168.10.2
```

E31x - WBFM TX Demo

```
# ./e310_wbfm_tx.py --help
linux; GNU C++ version 4.9.2; Boost_105700; UHD_003.009.002-0-unknown

Usage: e310_wbfm_tx.py: [options]

Options:
  -h, --help            show this help message and exit
  --wavfile=WAVFILE     Set wavfile [default=/home/root/test.wav]
  --center-freq=CENTER_FREQ
                        Set center_freq [default=915M]
  --rf-gain=RF_GAIN     Set rf_gain [default=50]
```

E31x - WBFM TX Demo

Run Python Flowgraph on E31x

```
# ./e310_wbfm_tx.py --wavfile /home/root/audio2.wav
```

Tune with another radio to 915 MHz

E31x - WBFM TX Demo

Ettus Research™
A National Instruments Company

Potential Error: **`TypeError: __init__() got an unexpected keyword argument 'fh'`**

Due to mismatch of GNU Radio versions running on E31x and Host

Terminal

```
root@ettus-e3xx-sg3:~# ./E310_WBFM_TX.py --wavfile wav1.wav
linux; GNU C++ version 4.9.2; Boost_105700; UHD_003.009.002-0-unknown

-- Loading FPGA image: /usr/share/uhd/images/usrp_e310_fpga_sg3.bit... done
-- Detecting internal GPSDO .... found
-- Initializing core control...
-- Performing register loopback test... pass
-- Performing register loopback test... pass
-- Performing register loopback test... pass
-- Performing CODEC loopback test... pass
-- Performing CODEC loopback test... pass
-- Setting time source to internal
-- Asking for clock rate 16 MHz
-- Actually got clock rate 16 MHz
-- Performing timer loopback test... pass
-- Performing timer loopback test... pass
Using Volk machine: neon_hardfp
Traceback (most recent call last):
  File "./E310_WBFM_TX.py", line 143, in <module>
    main()
  File "./E310_WBFM_TX.py", line 137, in main
    tb = top_block_cls(center_freq=options.center_freq, rf_gain=options.rf_gain, wavfile=options.wavfi
le)
  File "./E310_WBFM_TX.py", line 69, in __init__
    fh=-1.0,
TypeError: __init__() got an unexpected keyword argument 'fh'
-- Loading FPGA image: /usr/share/uhd/images/usrp_e3xx_fpga_idle_sg3.bit... done
```

E31x - WBFM TX Demo

Resolve by editing generated Python file.

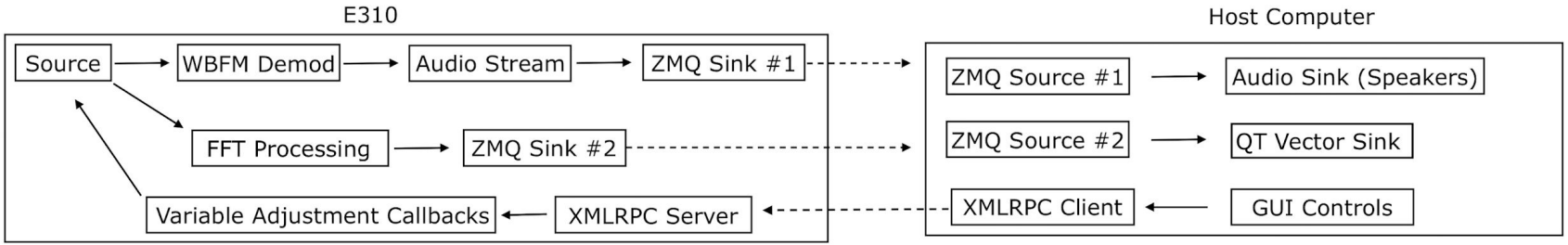
```
61          self.fractional_resampler_xx_0 = filter.fractional_resampler_cc(0, samp_rate/(audio_rate*4*5))
62          self.blocks_wavfile_source_0 = blocks.wavfile_source(wavfile, False)
63          self.blocks_multiply_xx_0 = blocks.multiply_vcc(1)
64          self.analog_wfm_tx_0 = analog.wfm_tx(
65              audio_rate=audio_rate,
66              quad_rate=audio_rate*4,
67              tau=75e-6,
68              max_dev=75e3,
69              fh=-1.0,
70          )
```

Remove line 69:

```
fh=-1.0,
```

E31x - WBFM ZMQ Demo

- Performs DSP operations to demodulate commercial WBFM radio station on E310
- Performs DSP operations to calculate FFT of received spectrum
- Streams demodulated Audio and FFT bins to remote host via ZeroMQ (ZMQ) sockets
- GUI controls run on host, commands are sent to E31x via XMLRPC interface

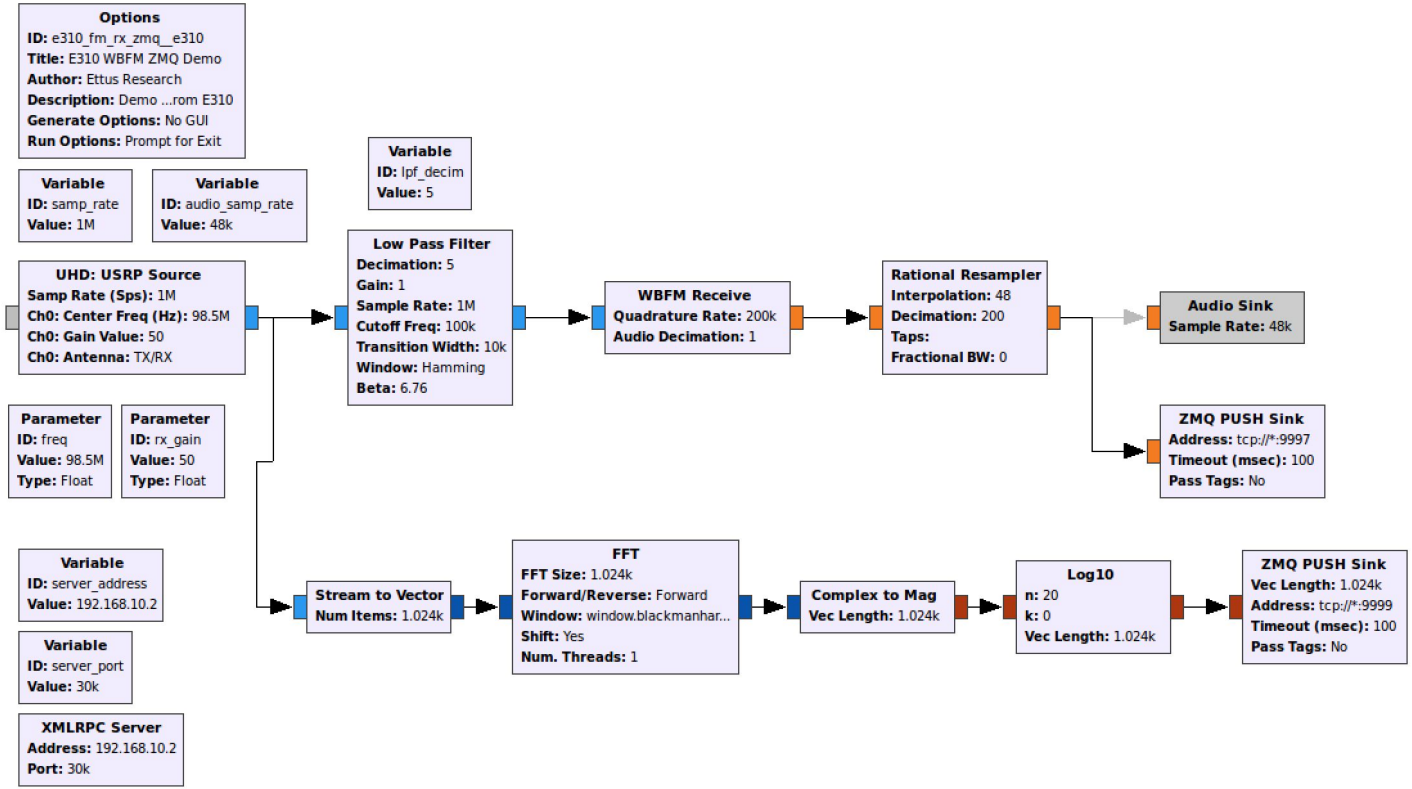


Application Note “Streaming processed data from the E31x with GNU Radio and ZMQ”
https://kb.ettus.com/Streaming_processed_data_from_the_E31x_with_GNU_Radio_and_ZMQ

E31x - WBFM ZMQ Demo

E31x Device Flowgraph

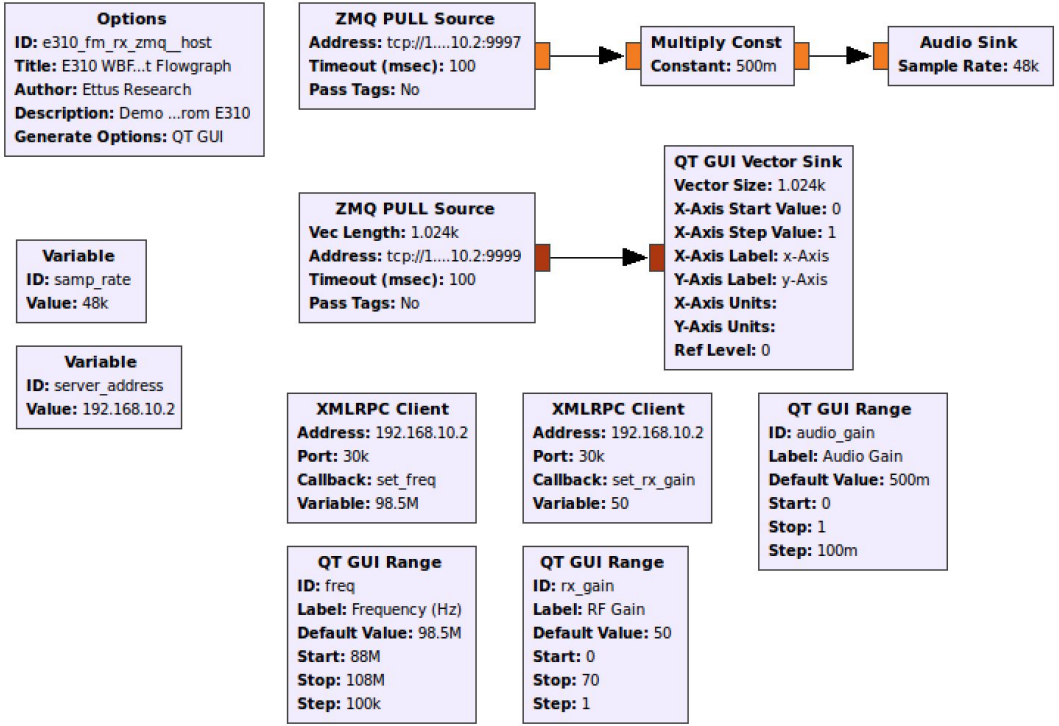
Location: ~/ettus_workshop/flowgraphs/e310/e310_fm_rx_zmq__e310.grc



E31x - WBFM ZMQ Demo

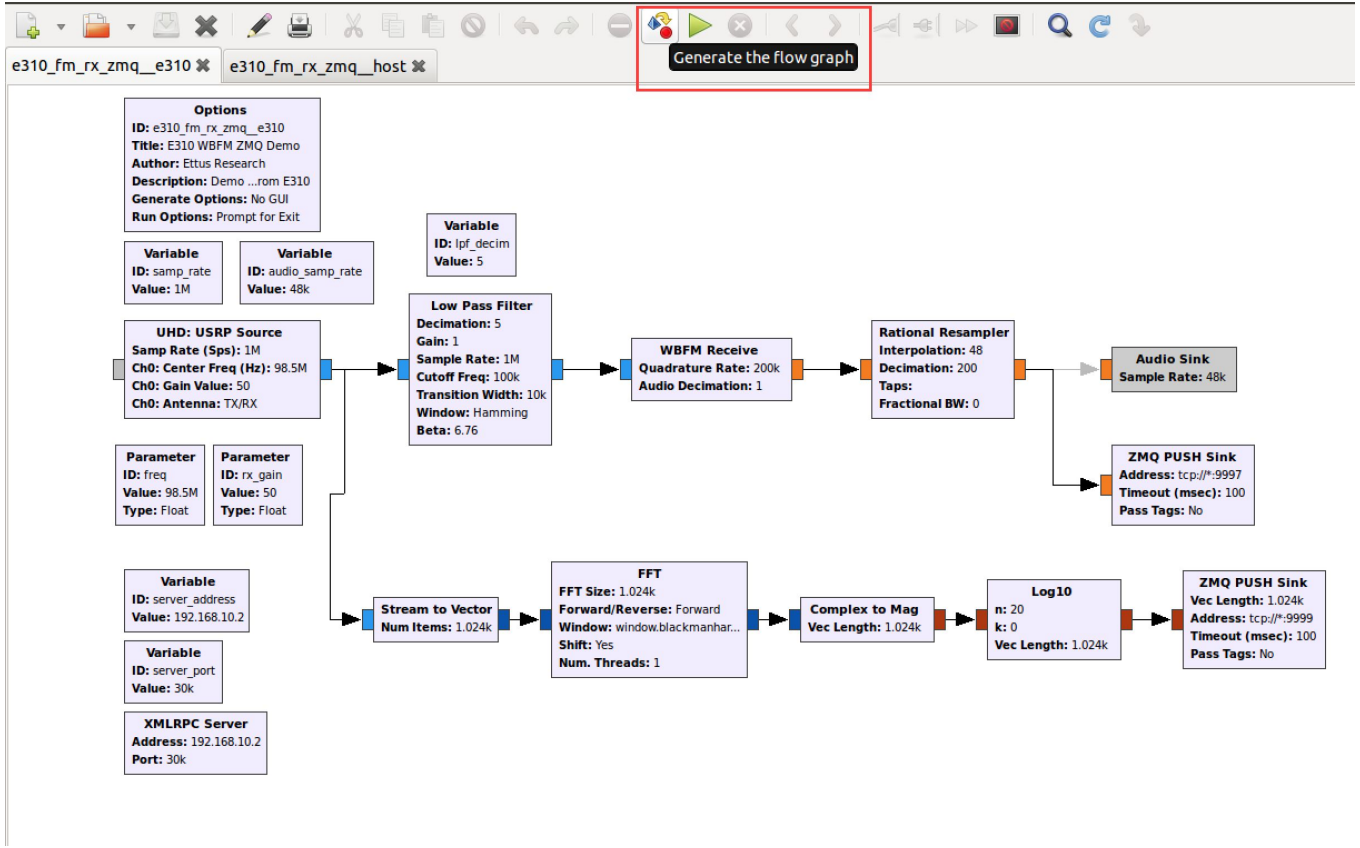
E31x Host Flowgraph

Location: ~/ettus_workshop/flowgraphs/e310/e310_fm_rx_zmq__host.grc



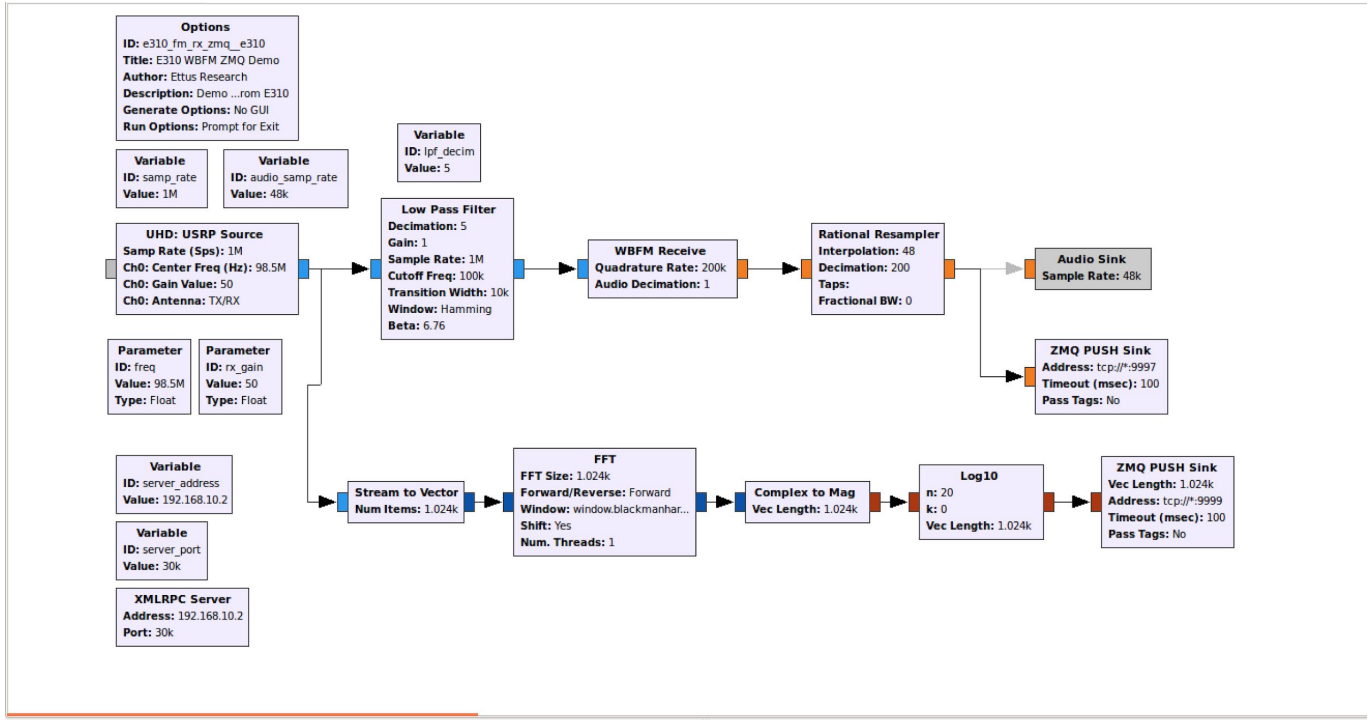
E31x - WBFM ZMQ Demo

Generate Python from Flowgraph within GRC for **`fm_receiver_zmq.grc`**



E31x - WBFM ZMQ Demo

Note file name and location in console where it was Generated



E31x - WBFM ZMQ Demo

Check connectivity to E31x by pinging.

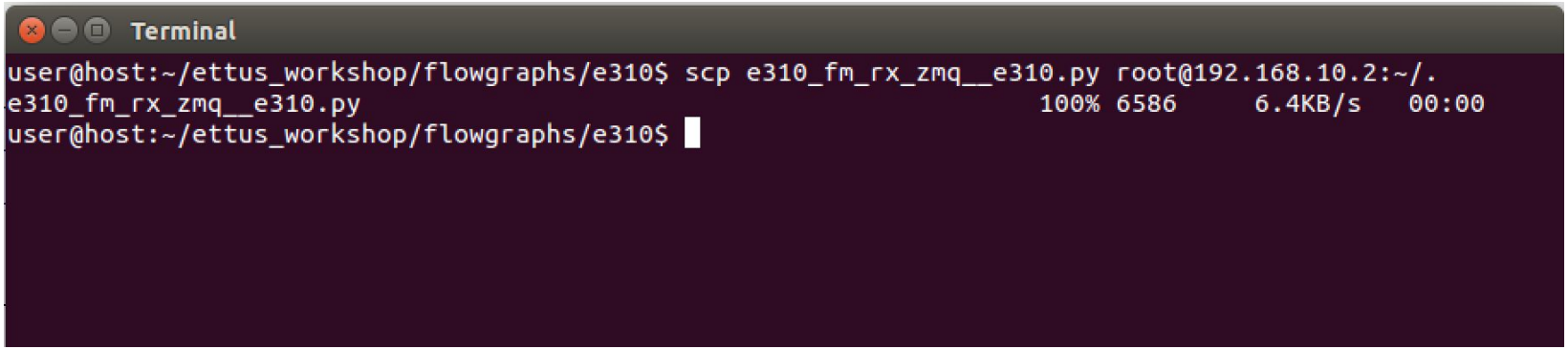
```
$ ping 192.168.10.2
```

```
ubuntu@ubuntu: ~/flowgraphs
ubuntu@ubuntu:~/flowgraphs$ ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=0.175 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.144 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=0.170 ms
^C
--- 192.168.10.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.144/0.163/0.175/0.013 ms
ubuntu@ubuntu:~/flowgraphs$
```

E31x - WBFM ZMQ Demo

Copy generated `.py` file to E31x

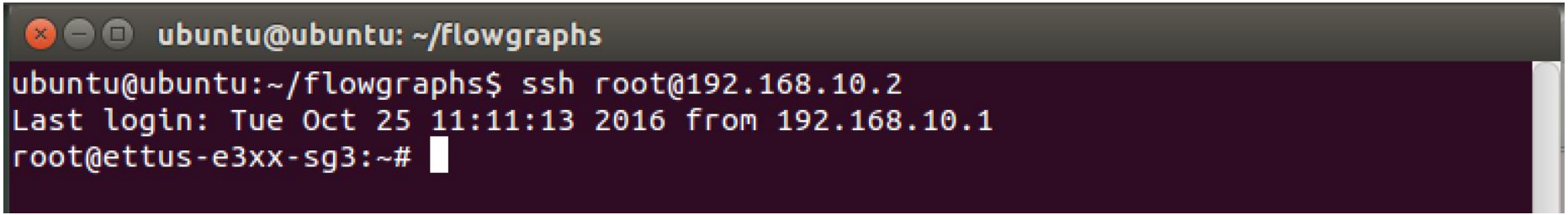
```
$ scp ~/ettus_workshop/flowgraphs/e310_fm_rx_zmq__e310.py root@192.168.10.2:~/.
```



E31x - WBFM ZMQ Demo

SSH to E31x

```
$ ssh root@192.168.10.2
```



E31x - WBFM ZMQ Demo

Verify you're running on E310

```
# uhd_find_devices
```

```
Terminal
root@ettus-e3xx-sg3:~# uhd_find_devices
linux; GNU C++ version 4.9.2; Boost_105700; UHD_003.009.002-0-unknown

--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    type: e3x0
    node: /dev/axi_fpga
    name:
    serial: 30D2812
    product: 30675
```

```
# uname -a
```

```
Terminal
root@ettus-e3xx-sg3:~# uname -a
Linux ettus-e3xx-sg3 3.14.2-xilinx #1 SMP PREEMPT Thu Jan 7 14:49:20 PST 2016 armv7l GNU/Linux
root@ettus-e3xx-sg3:~#
```

E31x - WBFM ZMQ Demo

Ettus Research™
A National Instruments Company

Run Flowgraph on E310 (Python)

```
# ./e310_fm_rx_zmq__e310.py
```

Terminal

```
root@ettus-e3xx-sg3:~# ./e310_fm_rx_zmq__e310.py
linux; GNU C++ version 4.9.2; Boost_105700; UHD_003.009.002-0-unknown

-- Loading FPGA image: /usr/share/uhd/images/usrp_e310_fpga_sg3.bit... done
-- Detecting internal GPSDO .... found
-- Initializing core control...
-- Performing register loopback test... pass
-- Performing register loopback test... pass
-- Performing register loopback test... pass
-- Performing CODEC loopback test... pass
-- Performing CODEC loopback test... pass
-- Setting time source to internal
-- Asking for clock rate 16 MHz
-- Actually got clock rate 16 MHz
-- Performing timer loopback test... pass
-- Performing timer loopback test... pass
Using Volk machine: neon_hardfp
Press Enter to quit:
```

E31x - WBFM ZMQ Demo

Potential Error:

`TypeError: push_sink_make() takes at most 5 arguments (6 given)`

Due to mismatch of GNU Radio versions running on E31x and Host

Terminal

```
root@ettus-e3xx-sg3:~# ./e310_fm_rx_zmq__e310.py
linux; GNU C++ version 4.9.2; Boost_105700; UHD_003.009.002-0-unknown

Traceback (most recent call last):
  File "./e310_fm_rx_zmq__e310.py", line 174, in <module>
    main()
  File "./e310_fm_rx_zmq__e310.py", line 163, in main
    tb = top_block_cls(freq=options.freq, rx_gain=options.rx_gain)
  File "./e310_fm_rx_zmq__e310.py", line 51, in __init__
    self.zeromq_push_sink_0_0_0 = zeromq.push_sink(gr.sizeof_float, 1024, 'tcp://
/*:9999', 100, False, -1)
  File "/usr/lib/python2.7/site-packages/gnuradio/zeromq/zeromq_swig.py", line 1
43, in make
    return _zeromq_swig.push_sink_make(*args, **kwargs)
TypeError: push_sink_make() takes at most 5 arguments (6 given)
root@ettus-e3xx-sg3:~#
```

E31x - WBFM ZMQ Demo

Resolve by editing generated Python file.

```
48          ##################################################
49          # Blocks
50          ##################################################
51          self.zeromq_push_sink_0_0_0 = zeromq.push_sink(gr.sizeof_float, 1024, 'tcp://*:9999', 100, False, -1)
52          self.zeromq_push_sink_0 = zeromq.push_sink(gr.sizeof_float, 1, 'tcp://*:9997', 100, False, -1)
53          self.xmlrpc_server_0 = SimpleXMLRPCServer.SimpleXMLRPCServer((str(server_address), int(server_port)), allow_none=True)
54          self.xmlrpc_server_0.register_instance(self)
55          self.xmlrpc_server_0_thread = threading.Thread(target=self.xmlrpc_server_0.serve_forever)
```

Edit lines near 51 & 52 (ZeroMQ Push Sink lines) by removing the last argument: **", -1"**

```
self.zeromq_push_sink_0_0_0 = zeromq.push_sink(gr.sizeof_float, 1024, 'tcp://*:9999', 100, False  ,
-1)
self.zeromq_push_sink_0 = zeromq.push_sink(gr.sizeof_float, 1, 'tcp://*:9997', 100, False  , -1)
```

Final Result:

```
self.zeromq_push_sink_0_0_0 = zeromq.push_sink(gr.sizeof_float, 1024, 'tcp://*:9999', 100, False)
self.zeromq_push_sink_0 = zeromq.push_sink(gr.sizeof_float, 1, 'tcp://*:9997', 100, False)
```

E31x - WBFM ZMQ Demo

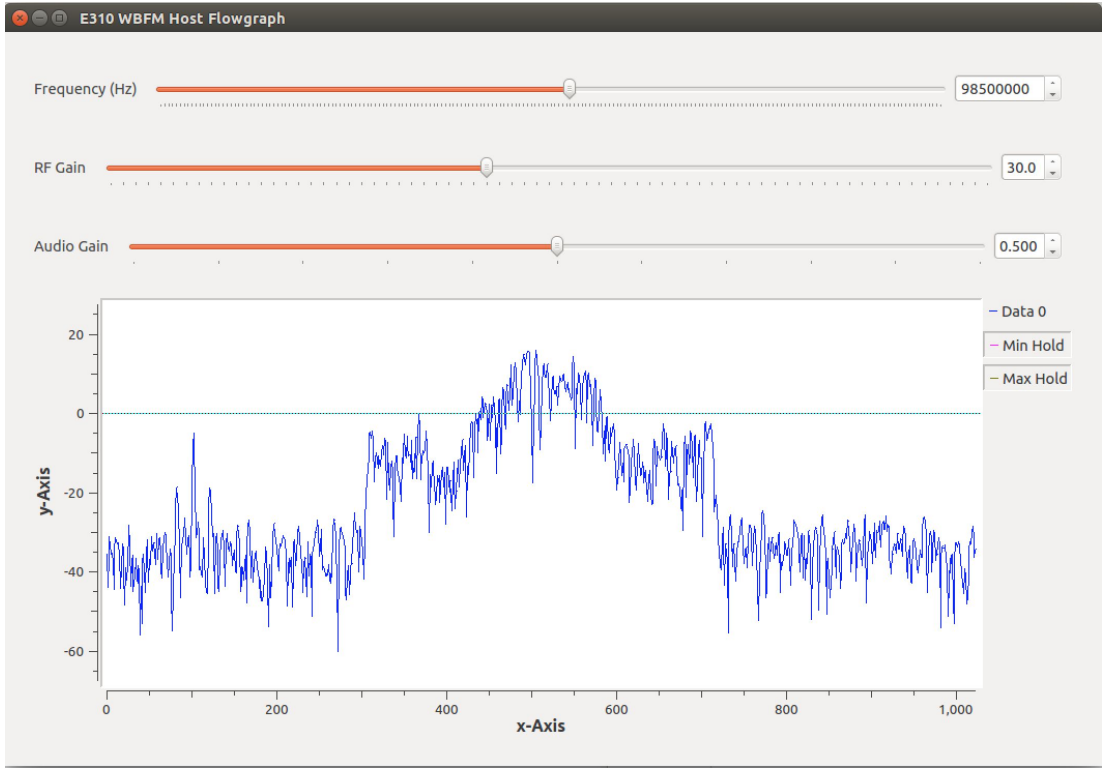
Return to GRC on Host Machine

Start "**`e310_fm_rx_zmq__host.grc`**" FG

Tune to strong local FM station

Adjust RF Gain

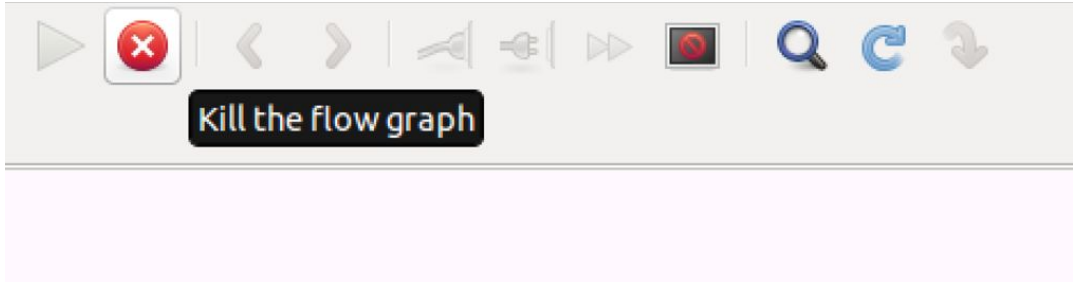
Adjust Audio Gain



E31x - WBFM ZMQ Demo

Stopping Flowgraphs

Kill flowgraph on host by either hitting “X” in corner of window, or “Stop” symbol within GRC



In Terminal which you’re connect to the E310 via SSH, hit “Enter” to stop the flowgraph.

Always shut down the E310 safely to avoid SD card corruption.

```
# shutdown -h now
```

FRS Background

The Family Radio Service (FRS) is an improved walkie-talkie radio system authorized in the United States since 1996. This personal radio service uses channelized frequencies around 462 and 467 MHz in the ultra high frequency (UHF) band.

- Common “bubble-pack” type walkie-talkies
- Frequency Modulation (FM)
- 14 Channels
- Average range 0.5 to 1.5km in urban environment
  - Under exceptional conditions up to 60 km (LOS, hilltop to hilltop)
- Some channels shared with General Mobile Radio Service (GMRS)
- GMRS requires license, allow for better antennas, higher power, repeaters

Channel	Frequency (MHz)	Notes
1	462.5625	Shared with GMRS
2	462.5875	Shared with GMRS
3	462.6125	Shared with GMRS
4	462.6375	Shared with GMRS
5	462.6625	Shared with GMRS
6	462.6875	Shared with GMRS
7	462.7125	Shared with GMRS
8	467.5625	FRS use only
9	467.5875	FRS use only
10	467.6125	FRS use only
11	467.6375	FRS use only
12	467.6625	FRS use only
13	467.6875	FRS use only
14	467.7125	FRS use only

FRS CTCSS / DCS Codes

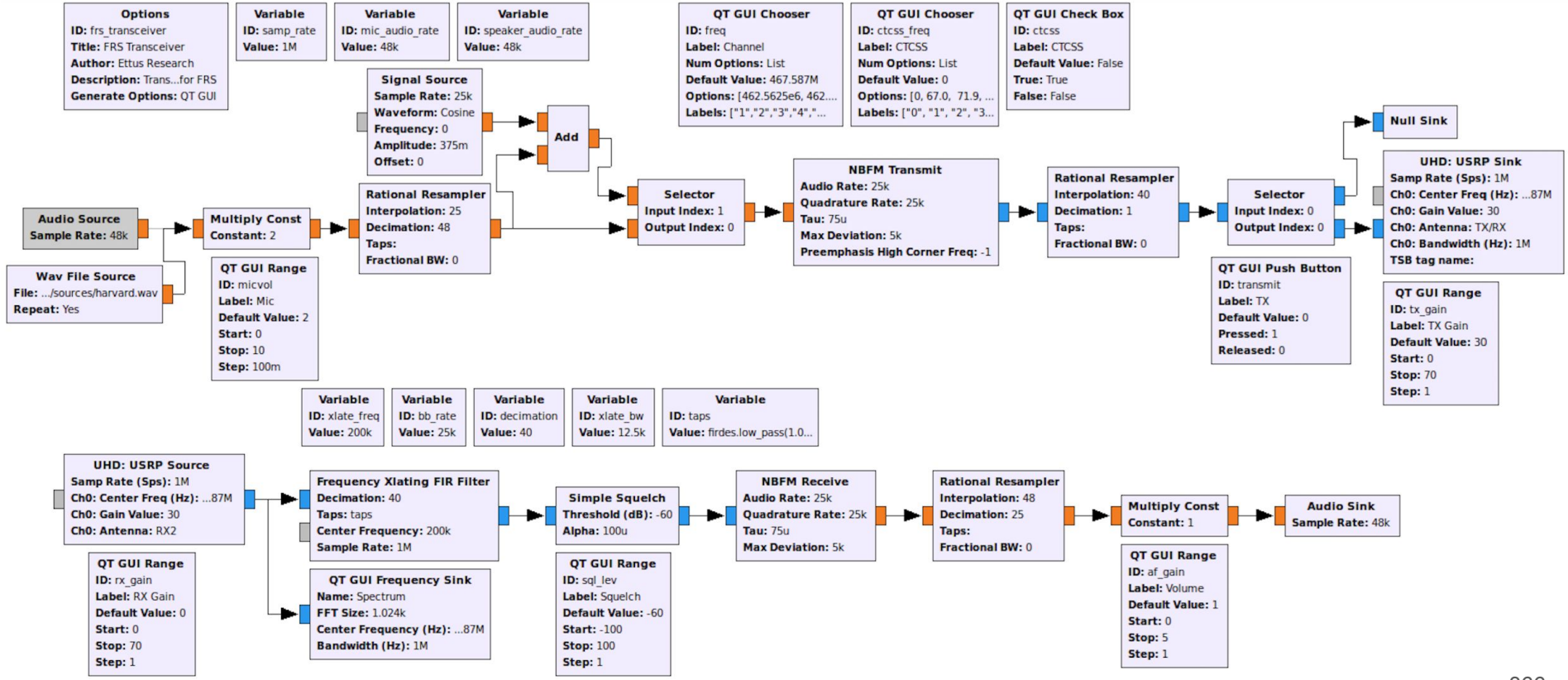
CTCSS (Continuous Tone-Coded Squelch System)

- Continuously superimposes sub-audible tone on the transmitted signal, ranging from 67 to 254 Hz.
- Filters out unwanted chatter from other users on the same frequency
- Sometimes called "privacy codes" or "private line codes" (PL codes)
  - Offer no protection from eavesdropping
  - Only intended to help share busy channels
- Do nothing to prevent desired transmissions from being jammed by stronger signals having a different code.

DCS (Digital-Coded Squelch)

- Digital replacement for CTCSS
- Trademark of Motorola
- Adds 134.4 bps (sub-audible) bitstream to the transmitted audio

FRS Transceiver

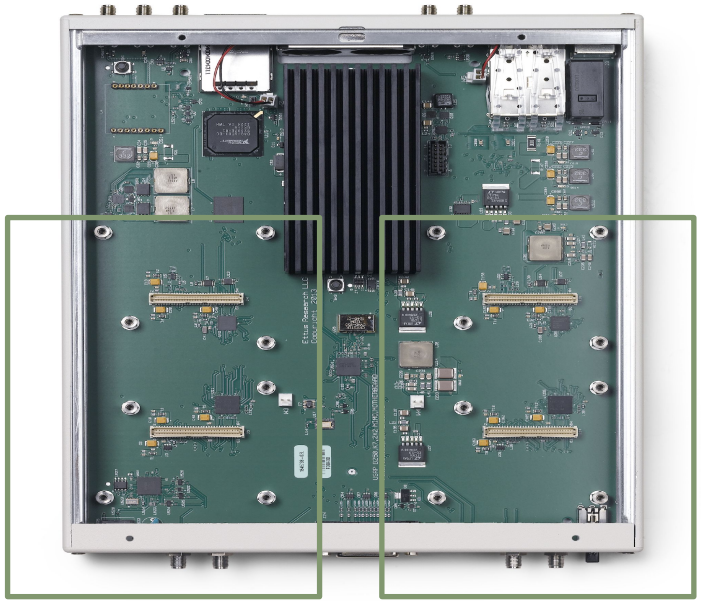


Multi-USRP Synchronization

- How to synchronize multiple X300 / X310 devices together for applications such as:
  - MIMO
  - Distributed MIMO
  - Phased Arrays
  - Beam-Forming (BF)
  - Direction-Finding (DF)
- The X300 / X310 have 2x2 MIMO capability out-of-the-box
- Data streaming requirements

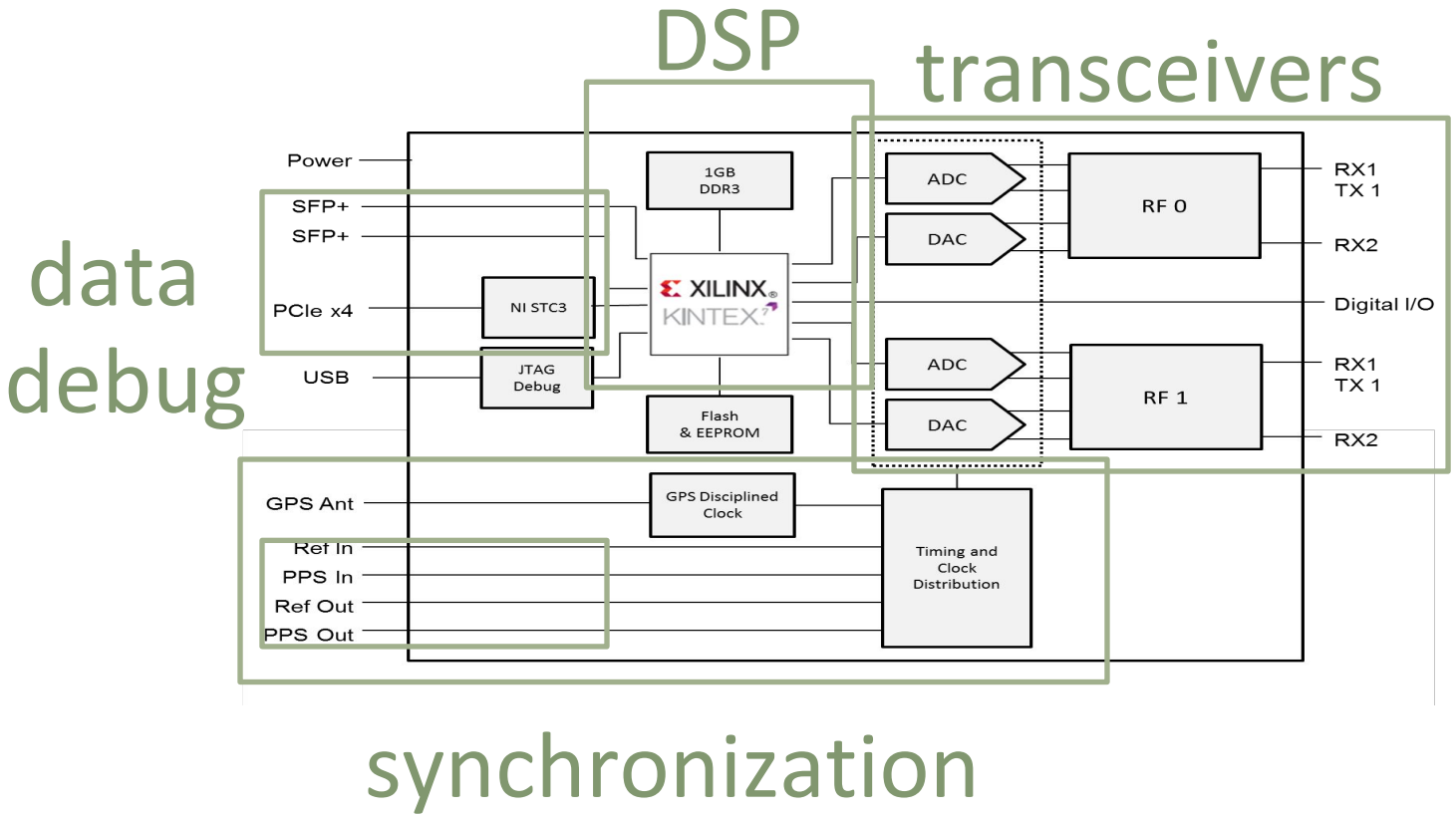
X300/X310 Inputs and Outputs

X300/X310 Motherboard



2 interchangeable RF daughterboards

X300/X310 Block Diagram



RF Front-End

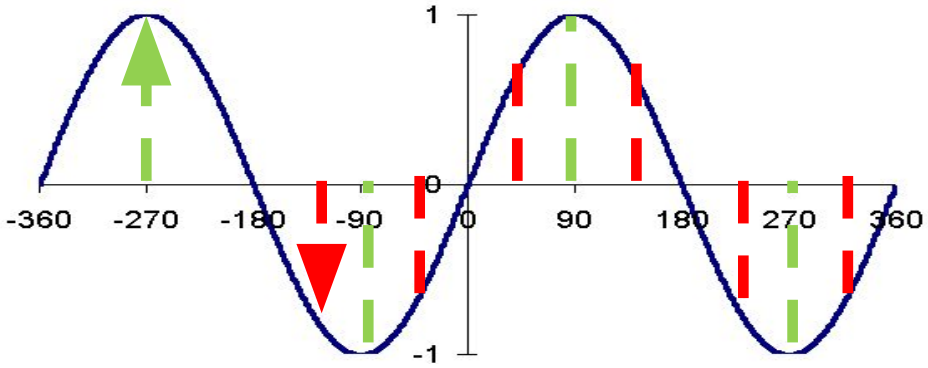
- Daughterboards serve as interchangeable RF front ends (two slots per motherboard)
- Direct conversion transceivers: low noise amplifier (LNA), mixer, local oscillator (LO), quadrature analog/digital converters (ADC, DAC)
- One LO per daughterboard.

Name	Bandwidth	Frequency
TwinRX	80 MHz per channel (160 MHz total)	10 MHz – 6 GHz
UBX	40 MHz, 160 MHz	10 MHz – 6 GHz
CBX	40 MHz, 120 MHz	1.2 GHz – 6 GHz
SBX	40 MHz, 120 MHz	400 MHz – 4.4 GHz
WBX	40 MHz, 120 MHz	50 MHz – 2.2 GHz
LFRX, LFTX	30 MHz	0 MHz - 30 MHz
BasicRX, BasicTX	250 MHz	1 MHz – 250 MHz

High Channel Count MIMO

Transceivers capable of MIMO (Multiple Input Multiple Output) operation must meet two basic requirements:

- **Sample clock synchronization** - sample clocks are synchronized and aligned
- **Device time synchronization** - DSP operations are performed on samples aligned in time (from the same sample clock edge)
- Sample clocks must have common frequency reference, and edges must be aligned
- All channels must acquire and process samples that represent the same temporal event



Distributed MIMO

Distributed MIMO systems require synchronization over wide geographical regions:

- **GPS Disciplined Oscillator (GPSDO)**
  - 10 MHz oven controlled crystal oscillator (OCXO) with 20 ppb frequency accuracy
  - Improve accuracy to 0.01 ppb when GPS receiver locks to satellite constellation
  - Time synchronization within 50 ns
  - at 900 MHz, 20 ppb is 18 Hz

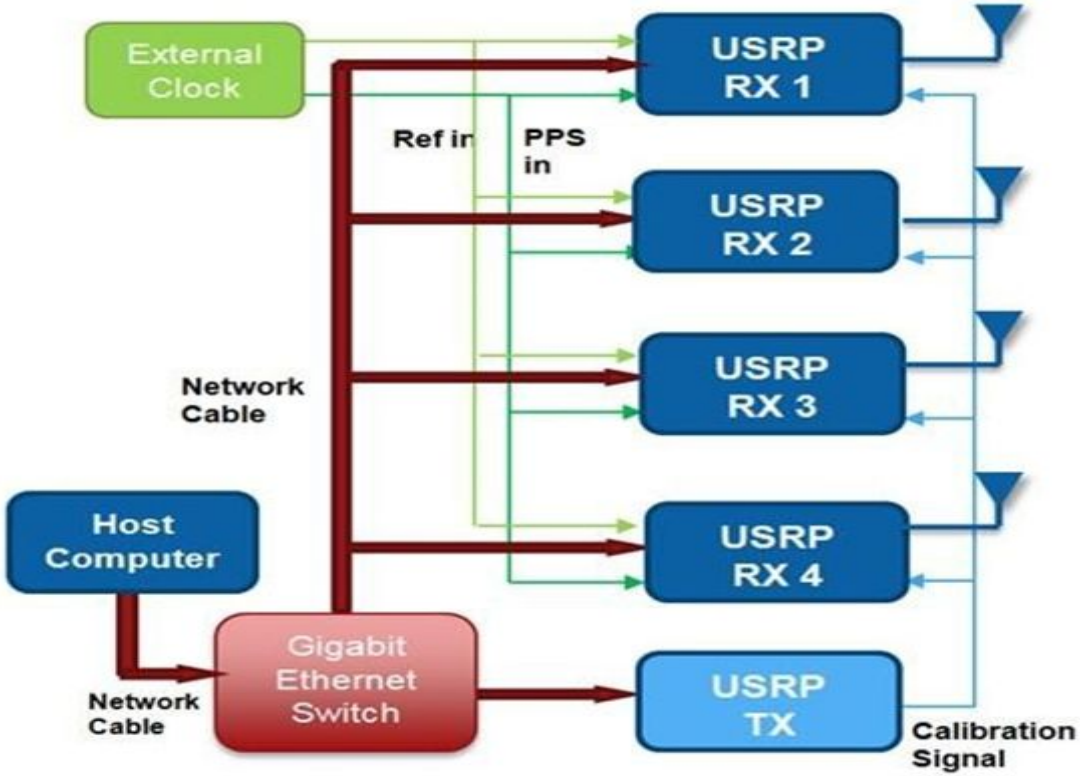
Jackson Labs LC_XO

Phased Arrays

Phased array applications like beamforming and direction-finding have additional requirements:

- **Channel phase synchronization**:
  - known phase relationships between RF inputs and outputs
- **Periodic Calibration:**
  - remove phase errors due to phase-locked loops (PLLs), mixers, amplifiers, and filter, which vary with time, temperature, mechanical condition, etc
- System-specific phase errors are constant and can be removed with calibration

Application Requirements



Synchronization Signals

The X300/X310 has three synchronization signals:

- **10 MHz Reference** – Frequency alignment of PLLs. Frequency alignment means all PLLs in system are derived from a common (master) reference. The slave PLLs inherit frequency accuracy of the master reference. The PLLs do not all need to output the same frequency (i.e., ADC sample clock fixed at 200 MHz, LOs are controlled by user)
- **Pulse-per-second (PPS) Trigger** – Time alignment of DSP operations, sample clocks, and hardware control commands within and between devices. Time alignment means actions are performed on samples of the same temporal event on all channels. (e.g., person A and person B have watches that are off. A and B are told to look outside at exactly noon to see if person C walks by.) The PPS ensures all devices have same notation of time.
- **GPS Antenna** – Lock GPSDO module to satellite constellation. GPS provides highly accurate 10 MHz reference (0.1 ppb) and PPS signals.

Phase-Locked Loops

- PLL – feedback circuit, output signal frequency is derived from input (reference) signal frequency
- Multiple PLLs used in USRP to generate master reference, sample clock, and LOs
- Each PLL has its own frequency and phase that is independent of others
- Analogy: multiple runners moving at different speeds (frequency) and starting at different times (phase)
- Frequency alignment - all PLLs must derive frequency from a common 10 MHz reference
- Frequency accuracy depends on the type of oscillator in reference PLL

Frequency Accuracy

Ettus Research™
A National Instruments Company

	Internal TCXO	GPS-Disciplined Clock
Frequency Reference	TCXO	OCXO
Frequency Accuracy	± 2.5ppm *± 2,500 Hz @ 1 GHz*	± 20 ppb *± 20 Hz @ 1 GHz*
Frequency Accuracy (GPS-Disciplined)		± 0.01ppb *~ ± 0.01 Hz @ 1 GHz*
GPS Time Sync Accuracy		±50ns to UTC Time**
10 MHz Reference Phase Drift with GPS Sync		<±20ns After 1 Hour**

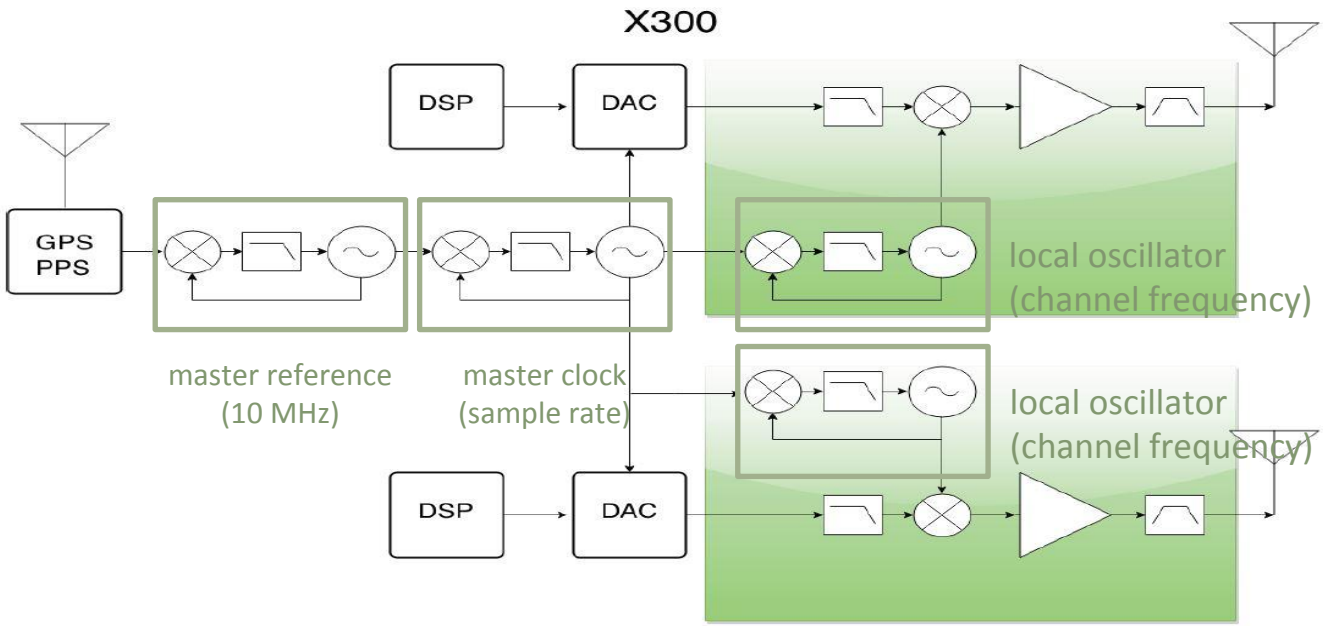
Device Time

- Recall that MIMO systems require clock synchronization and time synchronization
- One sample clock is shared between two daughterboards – frequency alignment
- 10 MHz reference ensures frequency alignment of sample clocks across devices
- Samples must be acquired and operated on at the same time across all channels
- Counter on each device (in FPGA) keeps track of time, driven by master sample clock
  - 64-bit unsigned integer
- PPS signal used to synchronize device time

Single Device

Constant phase offset is due to independent dividers in LOs, and phase errors introduced by other RF components. Master reference only aligns LO frequency, does not correct phase errors.

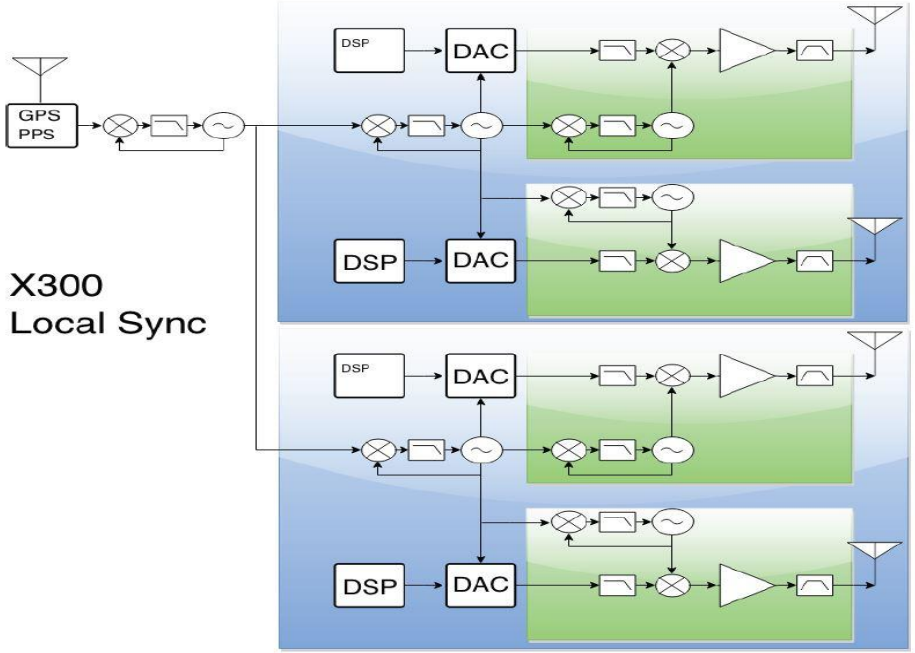
2x2 MIMO capable out-of-box



Multiple Devices

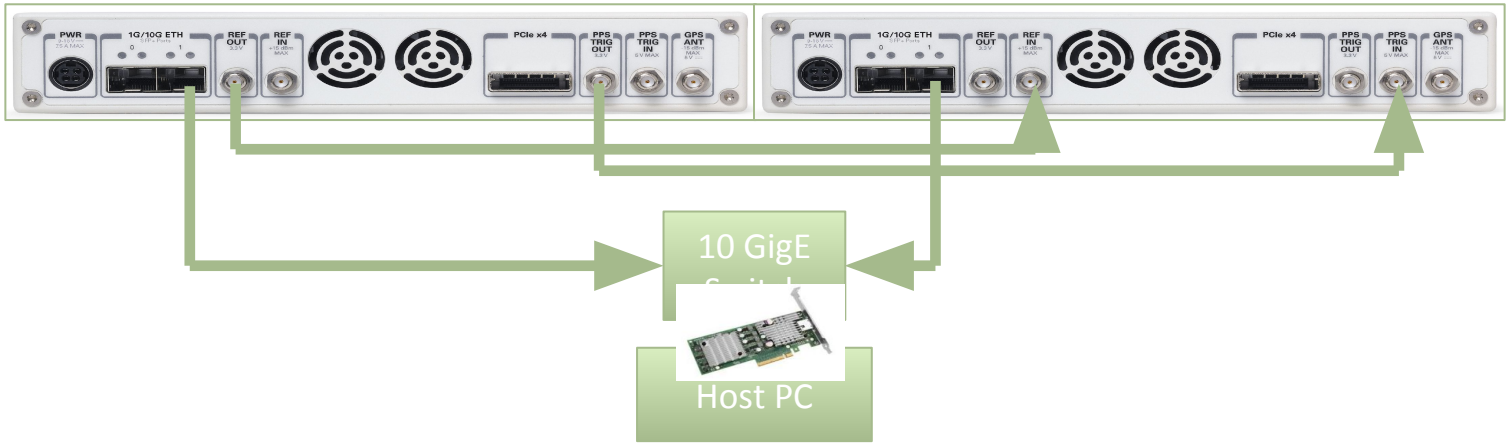
Requires common, external reference and PPS signal to be supplied to each device.

All LOs exhibit constant phase offset relative to each other.



Daisy Chain

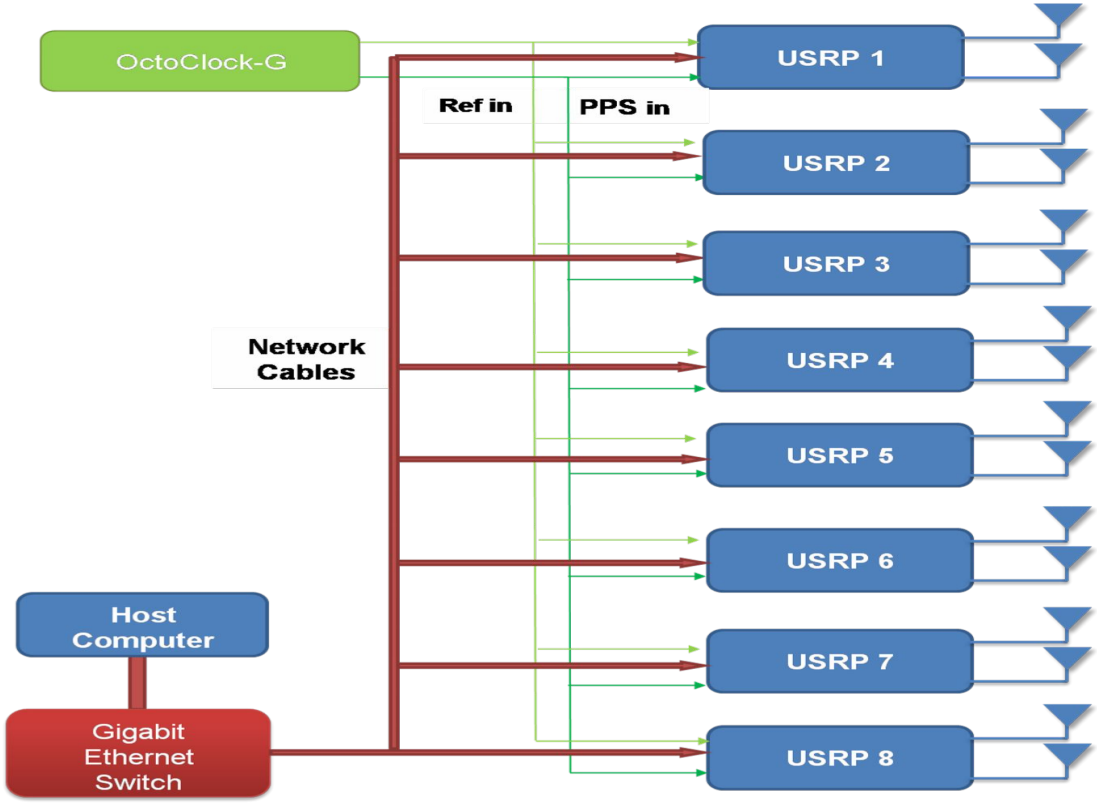
- Daisy chain two devices for 4x4 MIMO
- Master exports internal 10 MHz reference (TCXO) and 1 PPS to slave
- Caution: propagation delay increases with channel count



OctoClock CDA-2990

- Use OctoClock or OctoClock-G to build large, scalable MIMO systems
- 8-channel clock distribution
- For more than 8 channels, cascade multiple OctoClocks into a tree-structure
- OctoClock requires external 10 MHz reference and 1 PPS signal, which gets distributed
- OctoClock-G contains integrated GPSDO module, generates its own 10 MHz and 1 PPS
- Use matched-length cables

Large, Scalable MIMO Systems



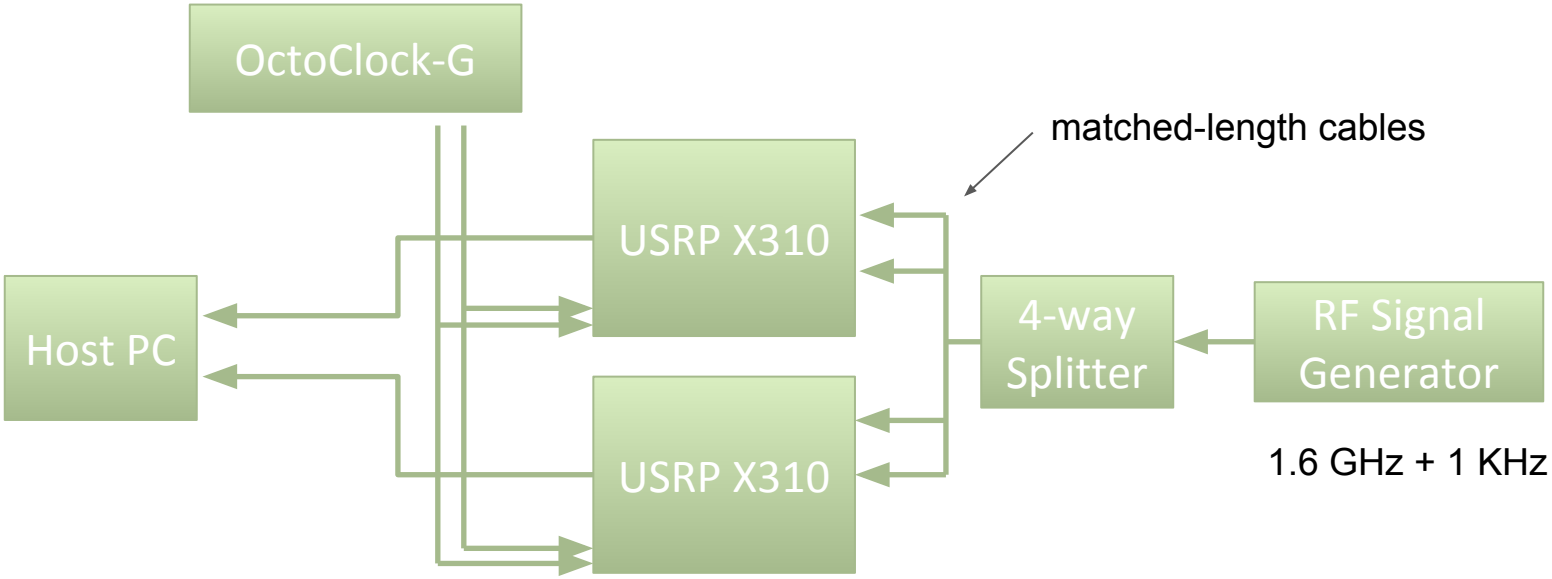
Daughterboards

- Recall: phased arrays require known phase relationship between RF inputs and outputs
- PLL on each daughterboard generates LO signal independently on each channel
- 10 MHz reference ensures frequency alignment of LOs across all channels
- Fractional-N dividers in PLLs introduce random phase offset after each LO retune
- Some daughterboards have resync capability after retune
- Phase ambiguity on WBX is due to external divider at output of PLL

Name	Phase Sync
TwinRX	Yes
UBX	Yes
CBX	No
SBX	Yes
WBX	Yes, with 180° ambiguity
LFRX, LFTX, BasicRX, BasicTX	No LOs

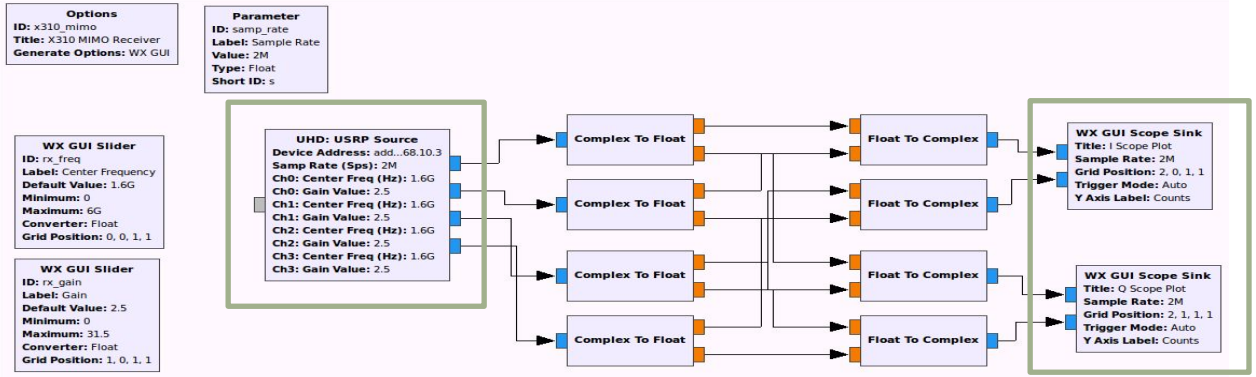
MIMO Test System Diagram

4-channel MIMO receiver with the X310 and GNU Radio



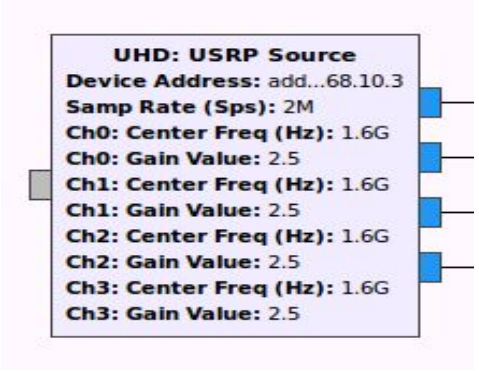
4-Channel Receiver Flowgraph

- Use a single USRP Source block to configure multiple devices in a MIMO system
- Specify more than one motherboard
- Display I and Q waveforms on separate Scope Sinks for clarity
- Set TX frequency to 1 KHz + RX frequency for 1 KHz tone

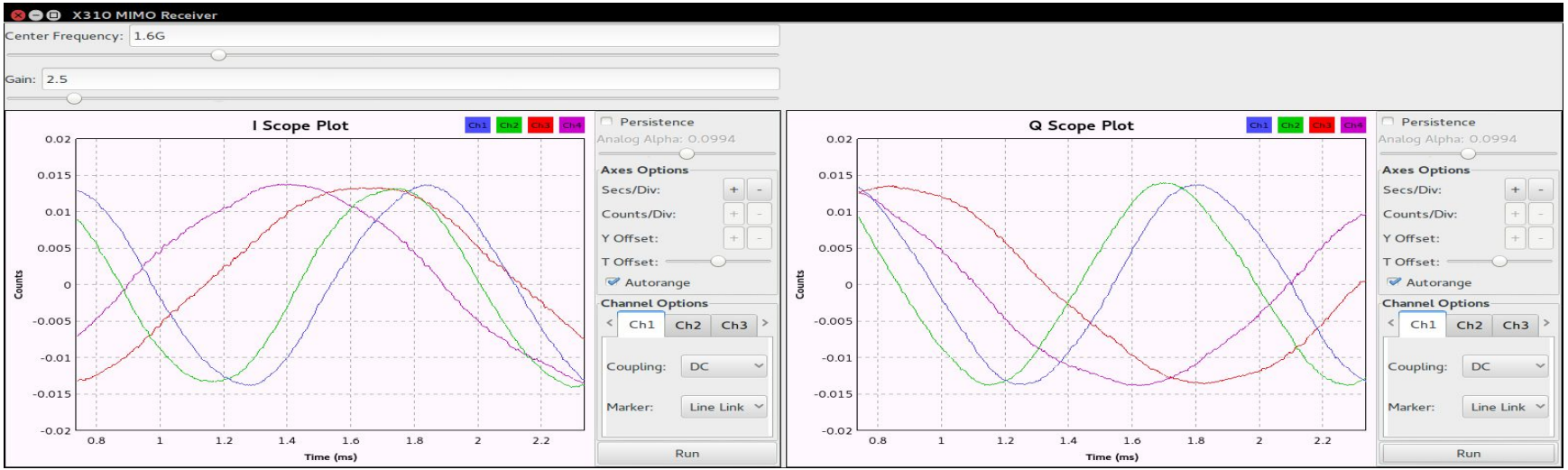


No Synchronization

- Each device uses its own internal 10 MHz reference (TCXO, 2.5 ppm)
- No common PPS signal, device time is not synchronized across devices
- Daughterboards on the same device have the same frequency and constant phase offset (out-of-the-box MIMO capability)
- Daughterboards across devices have <u>different frequencies</u> and <u>phase drift</u>



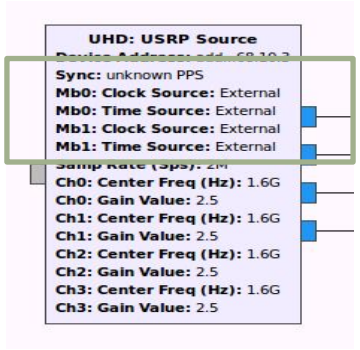
No Synchronization



Ch1 and **Ch2** are from first X310
Ch3 and **Ch4** are from second X310

Clock and Time Synchronization

- OctoClock-G 10 MHz reference (OCXO, 20 ppb) distributed to all devices
- OctoClock-G PPS trigger distributed to all devices
- All daughterboards have the <u>same frequency</u> and <u>constant phase offset</u>
- System meets sample clock and device time synchronization requirement for MIMO
- Synchronize channel phase by correcting constant phase offset in software

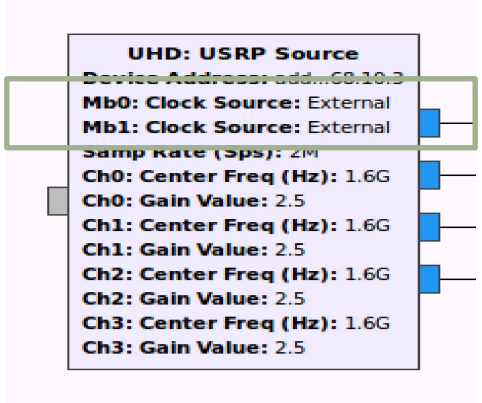


Clock and Time Synchronization

Frequency increased on both devices and is closer to 1 KHz, due to more accurate reference signal

Clock Synchronization Only

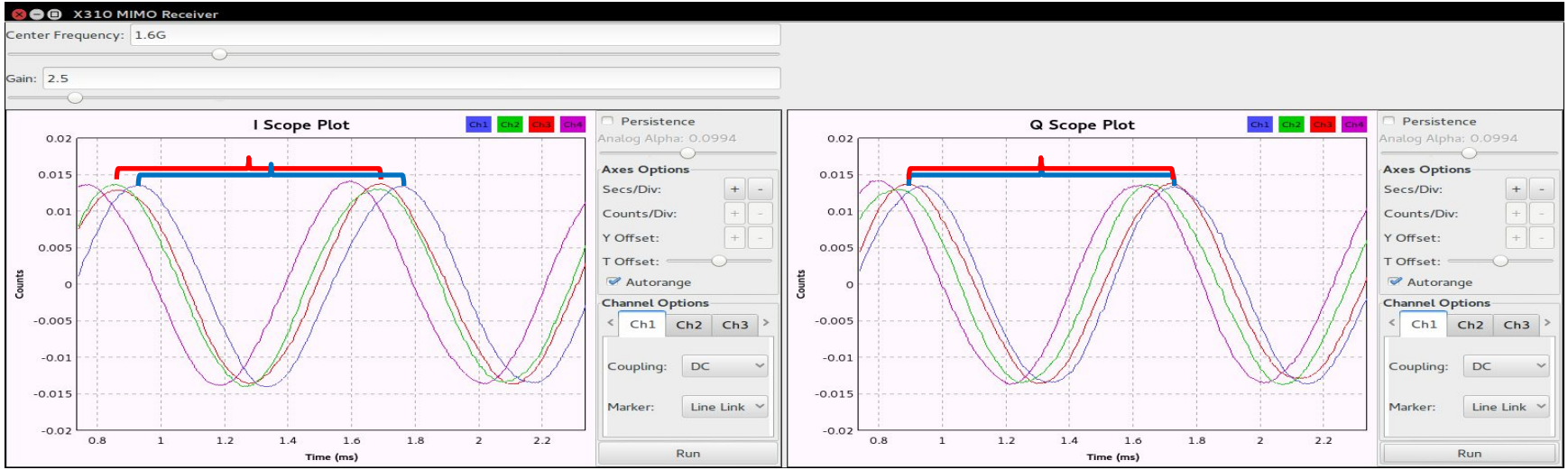
- OctoClock-G 10 MHz reference (OCXO, 20 ppb) distributed to all devices
- No common PPS signal, device time is not synchronized across devices
- All daughterboards have the <u>same frequency</u>
- Daughterboards across devices experience <u>phase drift</u>



Clock Synchronization Only

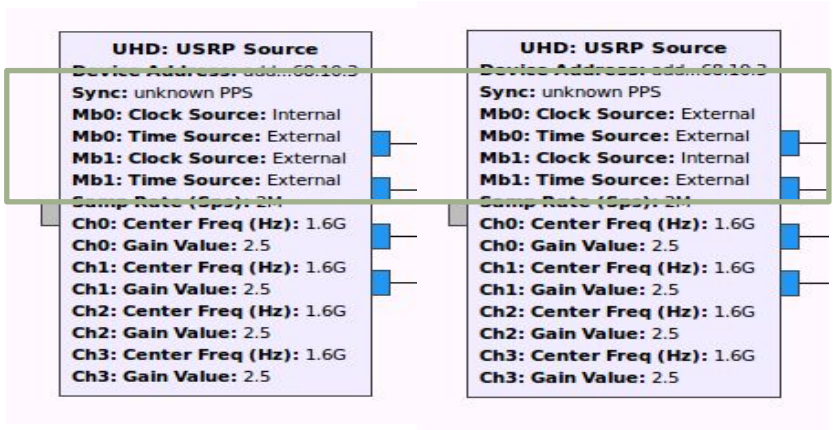
Ettus Research™
A National Instruments Company

Ch1 and Ch3 are from different devices and drift closer together



Frequency Accuracy

- OctoClock-G 10 MHz reference on one device only
- OctoClock-G PPS trigger distributed to all devices
- Daughterboards across devices have <u>different frequencies</u>
- Daughterboards across devices have <u>phase drift</u> due to difference in frequency
- Frequency error (difference from 1 kHz tone) depends on master oscillator



Frequency Accuracy

Device 0 – internal TCXO (2.5 ppm)

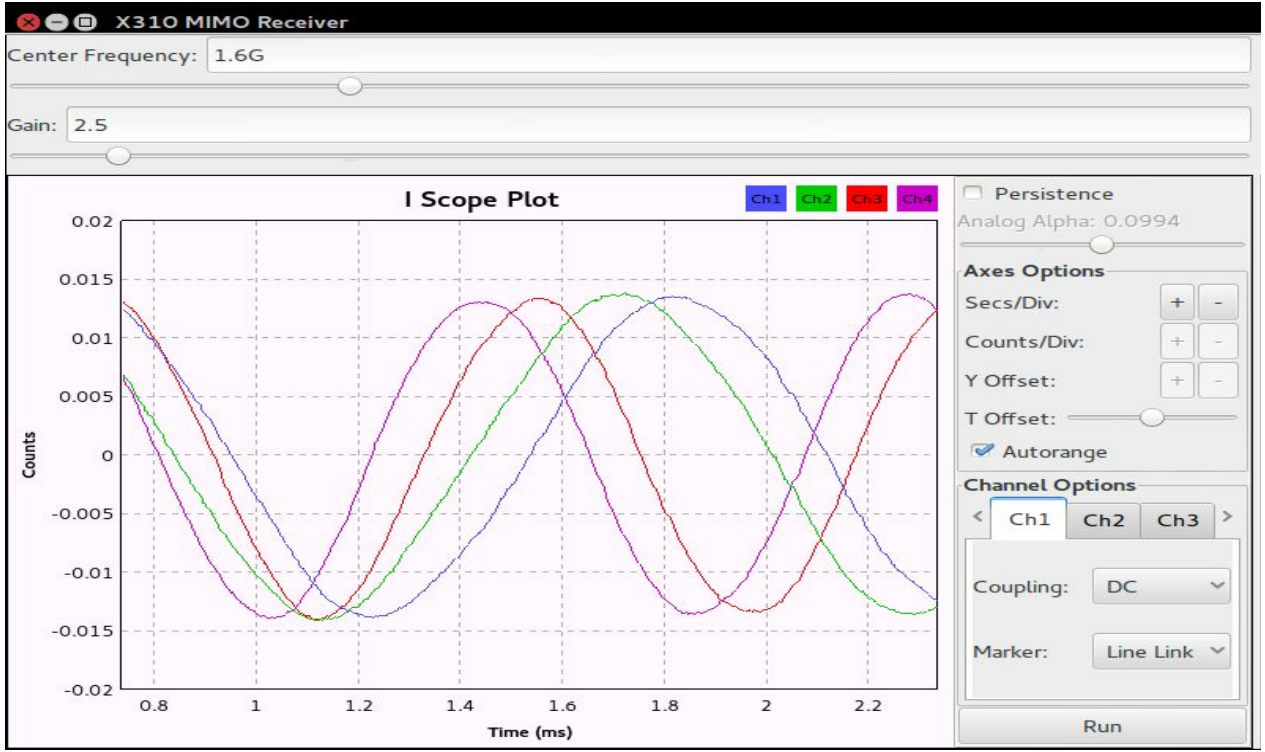
Device 1 – external OCXO reference (20 ppb)

Oscillators on Device 0, Device 1, and OctoClock all have different frequency accuracies

Frequency alignment affects how close the demodulated tone is to the desired 1 KHz source

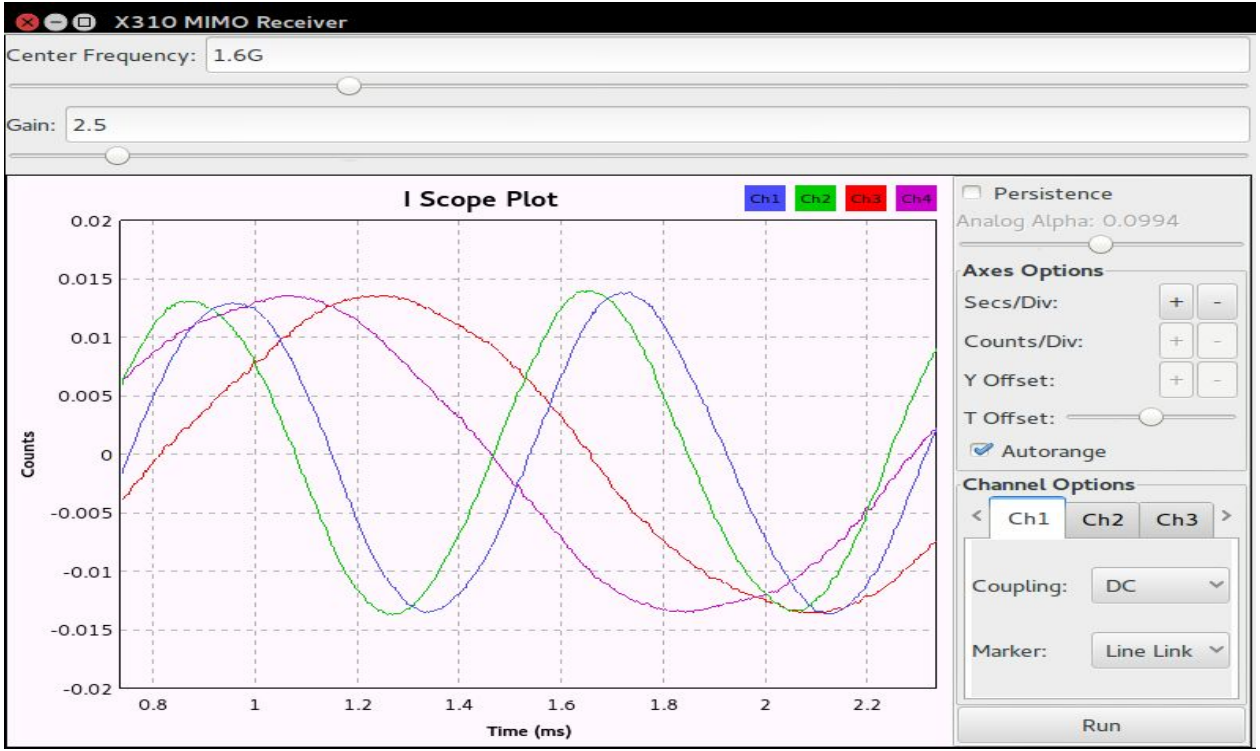
Signal source will contribute frequency error

Compare and contrast following slides with each other



Frequency Accuracy

Device 0 – external OCXO reference (20 ppb), Device 1 – internal TCXO (2.5 ppm)



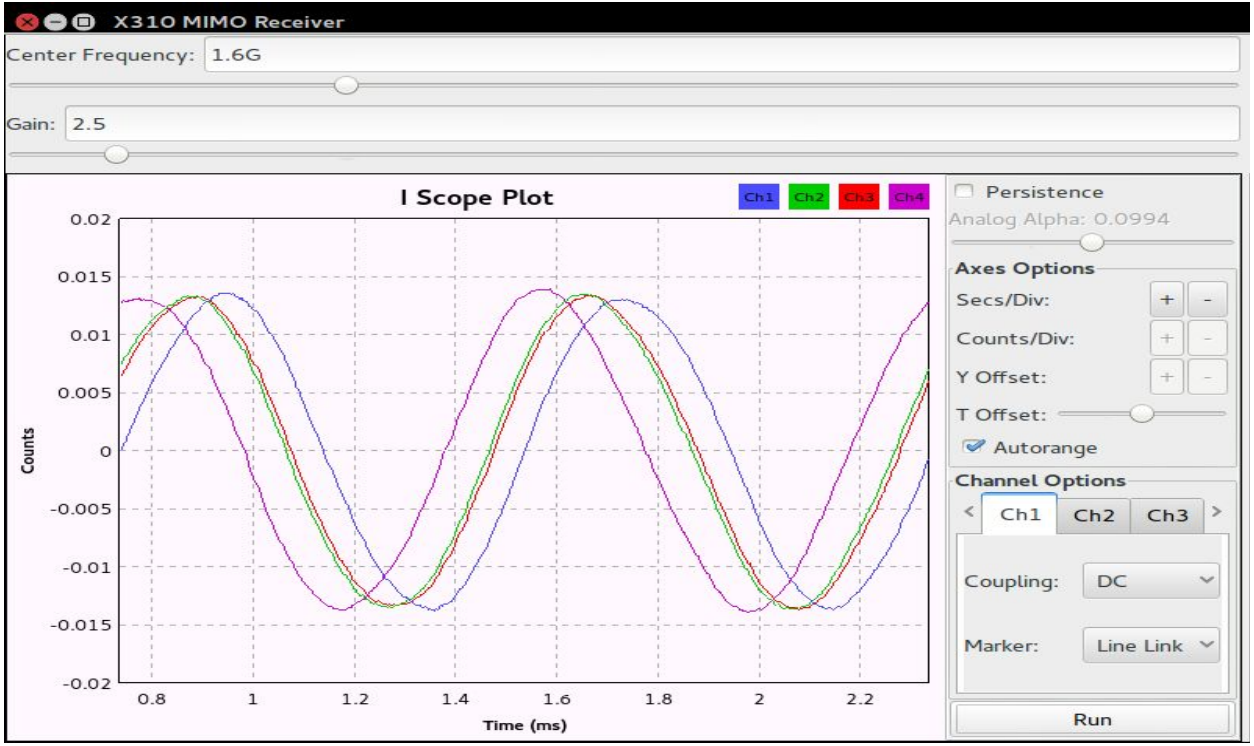
Frequency Accuracy

Both devices use internal TCXO reference (2.5 ppm)

Frequency Accuracy

Ettus Research™
A National Instruments Company

Both devices use external OCXO reference (20 ppb)



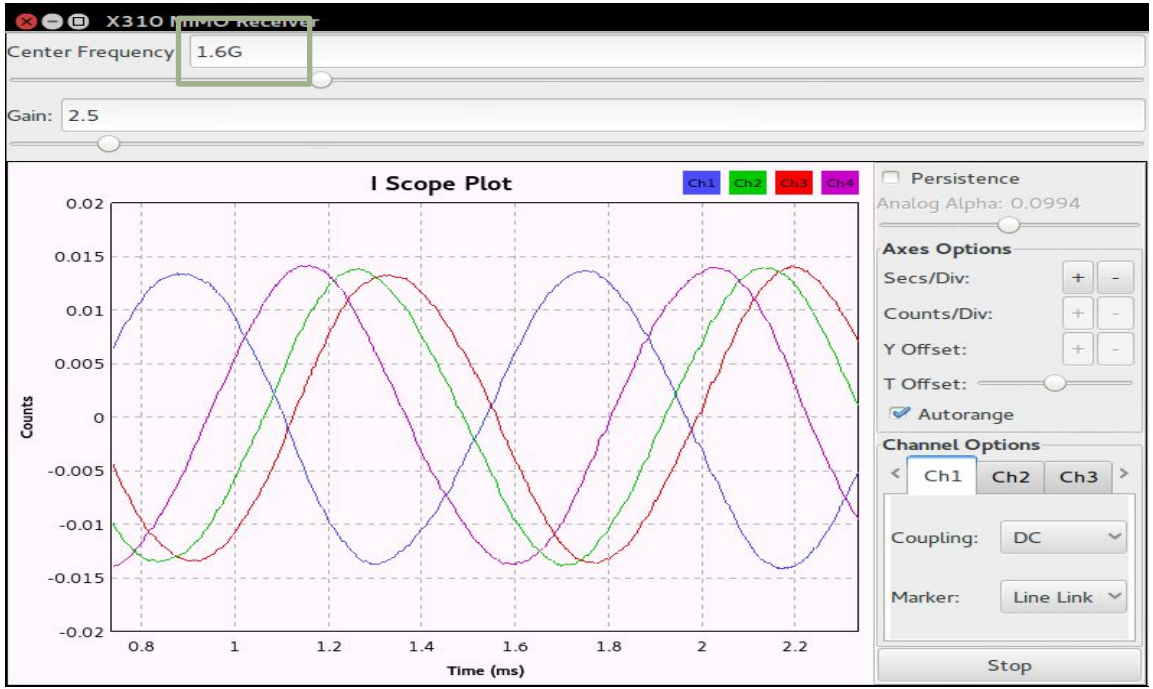
Resync LO Phase

- Phase offsets change after LO is re-tuned
- UBX and SBX will resync LO phase if re-tune command is synchronized to device time
- WBX will resync LO phase with 180 degree phase ambiguity due to external divider at output of PLL
- CBX does not have resync LO phase capability, re-calibration required after each re-tune

Resync LO Phase

Resync LO Phase

Resync LO Phase



Resync LO Phase

- Re-tuning with timed commands is not exposed through USRP Sink and Source blocks in GNU Radio (gr-uhd)
- User must edit the generated Python code for the flowgraph
- Must set timed command for each device individually

```
)
self.uhd_usrp_source_0.set_clock_source("external", 0)
self.uhd_usrp_source_0.set_time_source("external", 0)
self.uhd_usrp_source_0.set_clock_source("external", 1)
self.uhd_usrp_source_0.set_time_source("external", 1)
self.uhd_usrp_source_0.set_time_unknown_pps(uhd.time_spec())
self.uhd_usrp_source_0.set_samp_rate(samp_rate)

t = self.uhd_usrp_source_0.get_time_now() + uhd.time_spec(0.1)  # Added
self.uhd_usrp_source_0.set_command_time(t, 0)   # Added
self.uhd_usrp_source_0.set_command_time(t, 1)   # Added
self.uhd_usrp_source_0.set_center_freq(rx_freq, 0)
self.uhd_usrp_source_0.set_gain(rx_gain, 0)
self.uhd_usrp_source_0.set_center_freq(rx_freq, 1)
self.uhd_usrp_source_0.set_gain(rx_gain, 1)
self.uhd_usrp_source_0.set_center_freq(rx_freq, 2)
self.uhd_usrp_source_0.set_gain(rx_gain, 2)
self.uhd_usrp_source_0.set_center_freq(rx_freq, 3)
self.uhd_usrp_source_0.set_gain(rx_gain, 3)
self.uhd_usrp_source_0.clear_command_time(0)    # Added
self.uhd_usrp_source_0.clear_command_time(1)    # Added
self.q_scope_plot = scopesink2.scope_sink_c(
        self.GetWin(),
```

```
def set_rx_freq(self, rx_freq):
    self.rx_freq = rx_freq
    self._rx_freq_slider.set_value(self.rx_freq)
    self._rx_freq_text_box.set_value(self.rx_freq)
    t = self.uhd_usrp_source_0.get_time_now() + uhd.time_spec(0.1)  # Added
    self.uhd_usrp_source_0.set_command_time(t, 0)   # Added
    self.uhd_usrp_source_0.set_command_time(t, 1)   # Added
    self.uhd_usrp_source_0.set_center_freq(self.rx_freq, 0)
    self.uhd_usrp_source_0.set_center_freq(self.rx_freq, 1)
    self.uhd_usrp_source_0.set_center_freq(self.rx_freq, 2)
    self.uhd_usrp_source_0.set_center_freq(self.rx_freq, 3)
    self.uhd_usrp_source_0.clear_command_time(0)    # Added
    self.uhd_usrp_source_0.clear_command_time(1)    # Added
```

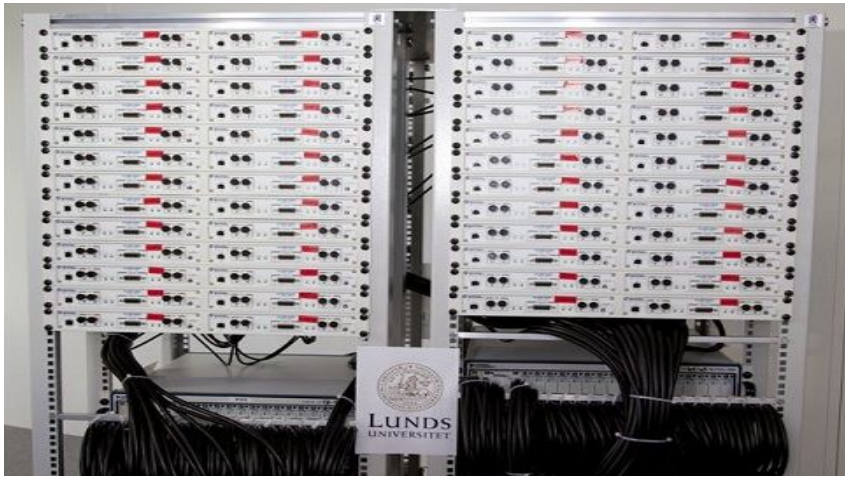
Resync LO Phase

Resync LO Phase

Resync LO Phase

MIMO Data Transfer

- Scalable MIMO systems must sustain fast data transfer rates
- More channels requires more streaming bandwidth
- X300/X310 provides multiple high-speed interface options (1 GbE, 10 GbE, PCIe)
- Consider FPGA processing to reduce data rate



MIMO Data Transfer

What is the data rate of a 4-channel MIMO receiver with 10 MHz signal bandwidth?

- 10 MHz BW requires quadrature (IQ) sample rate of 10 Msps
- 14-bit quadrature ADC requires:
    - 32 bits per sample, 4 bytes per sample, using the sc16 OTW format
- Each channel acquires 320 Mbps
- Full system data rate is 1280 Mbps

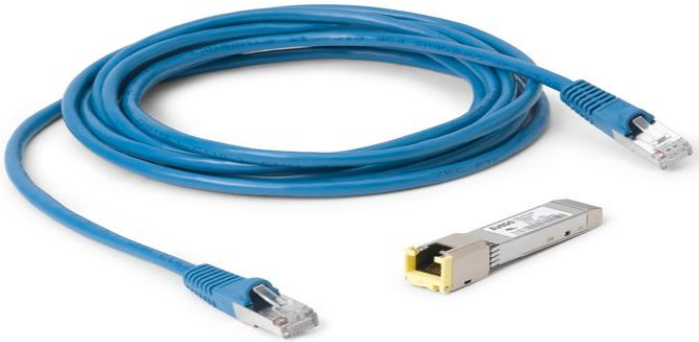
MIMO Data Transfer

Interface	USRP ™ Devices	Host Sample Rate (MS/s @ 16-bit I/Q)	Half/Full Duplex
USB 2.0	USRP ™ 1, B100	8	Half Duplex
USB 3.0	B200/B210	61.44	Half Duplex
Gigabit Ethernet	N200/N210	25	Full Duplex
10 Gigabit Ethernet	X300/X310	200	Full Duplex
PCI-Express (4-lane PCIe card)	X300/X310	200	Full Duplex
PCI-Express (1-lane ExpressCard)	X300/X310	50	Full Duplex
OMAP GPMC	E100/E110	4	Half Duplex

MIMO Data Transfer

Using the 1 GbE interface:

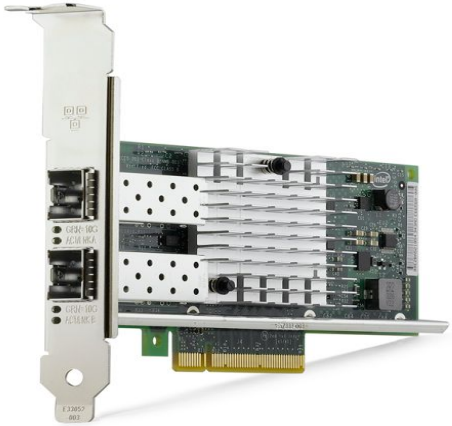
- Maximum streaming bandwidth is 25 Msps
- 4-channel receiver requires two X300 devices
- Requires host PC with multiple 1 GbE ports (one per X300 device)
- Will NOT work through Ethernet switch



MIMO Data Transfer

Using the 10 GbE interface:

- Realistic streaming bandwidth is 200 Msps
- Known-good 10 Gbps Ethernet cards:
  - Intel X520-DA2, Intel X540-T2, Intel X550-TA2, Intel X710-DA2
- Can the host keep up with the data rate?



MIMO Data Transfer

Using the PCIe interface:

- PCI-Express Kit for desktop computers is capable of 200 Msps
- ExpressCard Kit for laptop computers is capable of 50 Msps
- There are other limitations with PCI-Express, no advantages over 10 GbE

USRP FPGA

- Performs high-rate processing:
  - front-end filtering (CIC and half-band), DUC, DDC, interfaces to ADC and DAC
- Manages communication with the host computer
- Provides a register mapping for UHD to all devices on the MD and DB
- Open-source and hosted on GitHub
- Hosted as a Git submodule in UHD repository
- Entirely written in Verilog
- UHD stores FPGA images by default in the folder `/usr/local/share/uhd/images`
- `https://github.com/EttusResearch/fpga/tree/UHD-3.9.2`

USRP FPGA Devices

- USRP FPGA Devices:
  - N200: Xilinx® Spartan® 3A-DSP 1800
  - N210: Xilinx Spartan 3A-DSP 3400
  - B200: Xilinx Spartan 6 XC6SLX75
  - B210: Xilinx Spartan 6 XC6SLX150
  - E310: Xilinx Zynq XC7Z020
  - X300: Xilinx Kintex-7 XC7K325T
  - X310: Xilinx Kintex-7 XC7K410T

USRP FPGA Toolchains

- Toolchains:
  - N200, N210, B200, B210: Xilinx ISE, System Edition, version 14.7
  - E310, X300, X310: Xilinx Vivado
    - May use free WebPack Edition for E310
    - Must use non-free Design Edition or System Edition for X300 / X310
    - Version 2014.4 for UHD 3.9.x
    - Version 2015.4 for UHD 3.10.x
  - X300 / X310 used Xilinx ISE 14.7 prior to UHD 3.9.0
  - `http://www.xilinx.com/products/design-tools/vivado.html`
  - Simulation with Xilinx XSim and ModelSim PE, DE, SE

Building and Modifying USRP FPGA

- FPGA image builds are done with Makefiles from Linux (Ubuntu 14.04/16.04) command line
- Should also work under RHEL/CentOS 7 (command-line) and Microsoft Windows (GUI and project file)
- To build X300/X310 FPGA image, invoke Makefile with specific target:

```
make X300_XG
```

- Option to create project file:

```
make X300_XG GUI=1
```

- Two ways to modify and add custom functionality to FPGA:
  - Edit Verilog code directly (harder)
  - Use RFNoC (easier)

Building and Modifying USRP FPGA

- FPGA configuration targets for X-series:
  - **`X300_HG`** & **`X310_HG`**
    - Port 0: 1 GbE / Port 1: 10 GbE
    - DRAM FIFO
    - Only UHD 3.10
  - **`X300_XG`** & **`X300_XG`**
    - Port 0: 10 GbE / Port 1: 10 GbE (Dual 10 GbE)
    - DRAM FIFO
    - Only UHD 3.10
  - **`X300_HGS`** & **`X300_HGS`**
    - Port 0: 1 GbE / Port 1: 10 Gb
    - SRAM FIFO
    - Only UHD 3.8 and 3.9
  - **`X300_XGS`** & **`X300_XGS`**
    - Port 0: 10 GbE / Port 1: 10 Gb (not Dual 10 GbE)
    - SRAM FIFO
    - Only UHD 3.8 and 3.9

Building and Modifying USRP FPGA

- FPGA configuration targets for X-series:
  - **`X300_HA`** & **`X310_HA`**
    - Port 0: 1 GbE / Port 1: Aurora
    - DRAM FIFO
    - Xilinx Aurora interface
    - Only UHD 3.10
  - **`X300_XA`** & **`X300_XA`**
    - Port 0: 10 GbE / Port 1: Aurora
    - DRAM FIFO
    - Xilinx Aurora interface
    - Only UHD 3.10

Flashing USRP FPGA for X300 / X310

- UHD stores FPGA images by default in the folder **`/usr/local/share/uhd/images`**
- FPGA images are tied to a specific version of UHD
- The `003.009.005.tag` file in the images folder indicates the corresponding UHD version
- Run the **`uhd_images_downloader`** utility to obtain FPGA images for your current version of UHD
- Run the **`usrp_x3xx_fpga_burner`** utility to write the FPGA image onto the flash memory
- JTAG interface can be used for recovery from bricking
    - Loaded image will not persist across power-cycles, so you must use **`usrp_x3xx_fpga_burner`** after JTAG'ing to make the image persist
    - Use free Xilinx iMPACT or Xilinx Vivado Lab Edition to perform JTAG

X300 / X310 Device Recovery via JTAG

1. Download Xilinx Vivado Lab 2015.4. Must be Vivado 2015.4.

2. Install it, by default it ends up in **`/opt/Xilinx`**

3. Install cable driver:
```
sudo /opt/Xilinx/Vivado_lab/2015.4/data/xicom/cable_drivers/lin64/install_script/install_drivers/install_digilent.sh
```

4. To reload the new udev rules that tell your linux system to make the JTAG adapter available to normal users

```
sudo udevadm control --reload
```

5. Run Vivado, go to Hardware Manager, connect your X310 via USB JTAG and power it up

6. Within Hardware Manager, **`Tools -> Autoconnect`**

7. Right-click on the FPGA in the hardware "list", program device, select the appropriate .bit from UHD typically:
   `/usr/share/uhd/images/usrp_x310_fpga_HG.bit`

After you have flashed the FPGA image via JTAG, you will then need to connect it via ethernet, and reflash the image before power cycling the device with the **`"uhd_image_loader"`** utility, which will write a new FPGA image to the EEPROM.

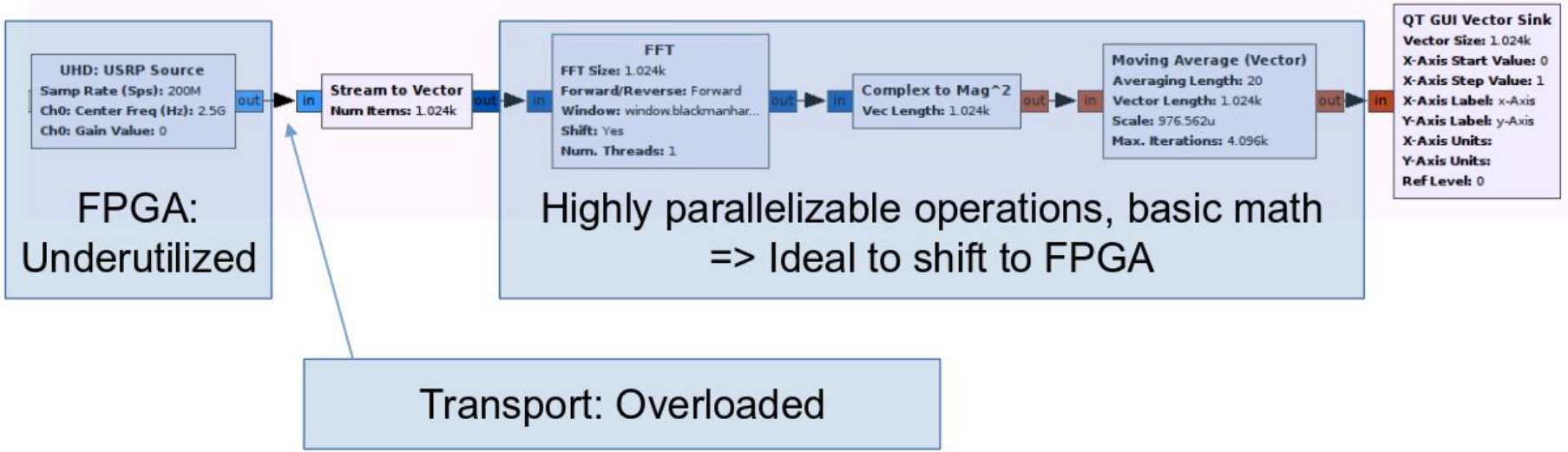
* Annotated step-by-step Application Note can be found in the Ettus Research Knowledge Base (kb.ettus.com)

RFNoC

- RF Network-on-Chip (RFNoC) is a technology that enables modular SDR development on FPGAs
- The goal is to make FPGA computing more accessible, and automatically manage the integration and implementation details, and let the user focus more on their algorithm and application
- Many applications underutilize the FPGA, but would benefit from moving some processing from the CPU to the FPGA
    - parallelizable algorithms
    - large amounts of data, high sampling rates
    - low-latency signal processing

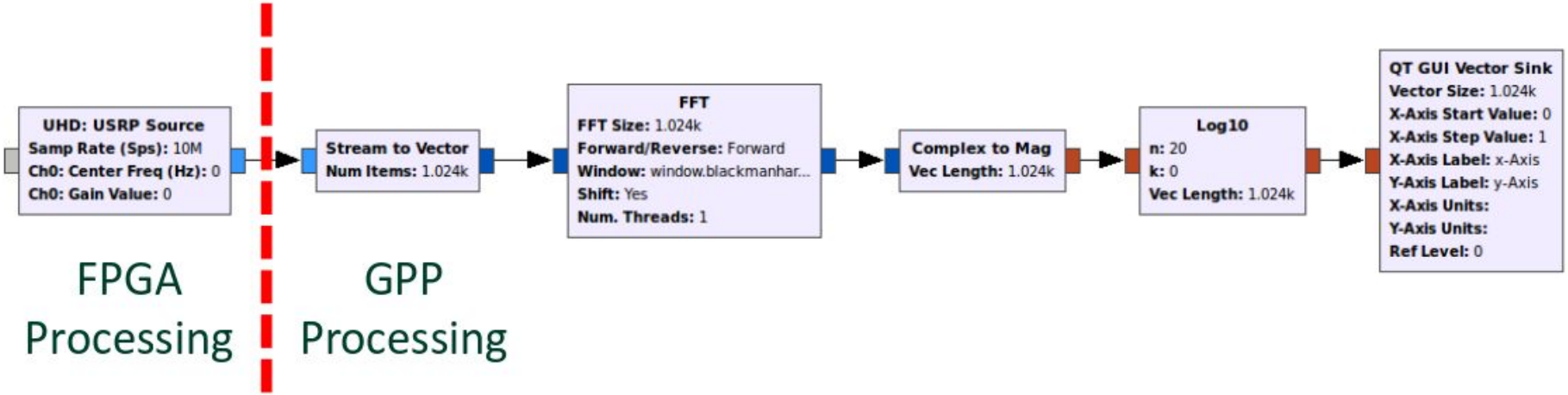
RFNoC (cont'd)

- Example application: 200 MHz real-time Welch's Algorithm for spectral density estimation



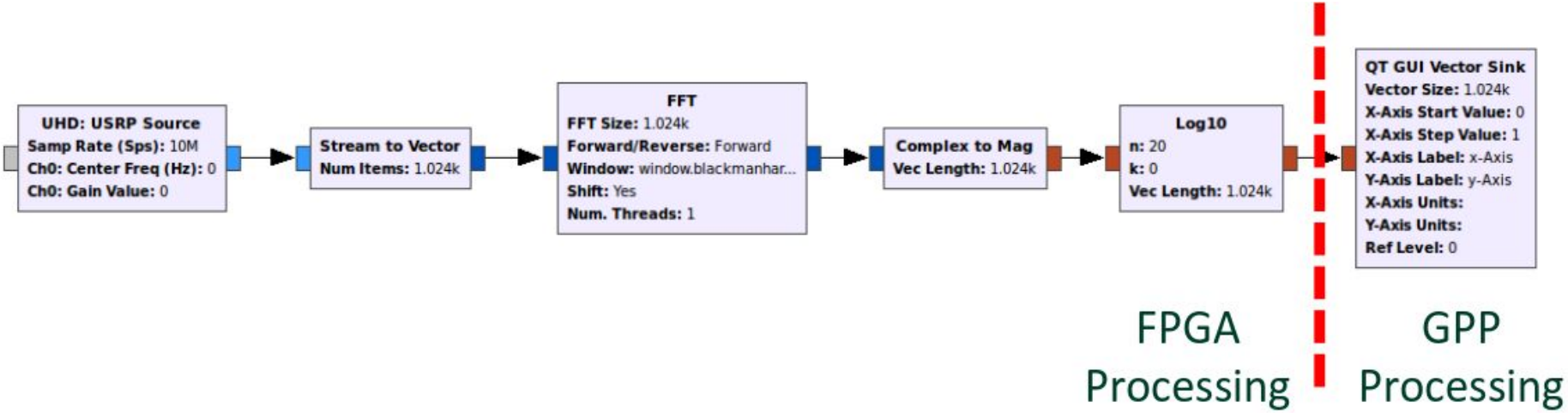
RFNoC (cont'd)

- Goals:
  - Heterogeneous Processing
  - Support composable and modular designs using CPU, FPGA, and beyond
  - Maintain ease-of-use, and make FPGA acceleration easier on USRP devices
  - Provide tight integration with popular SDR frameworks (i.e., GNU Radio)



RFNoC (cont'd)

- Goals:
  - Heterogeneous Processing
  - Support composable and modular designs using CPU, FPGA, and beyond
  - Maintain ease-of-use, and make FPGA acceleration easier on USRP devices
  - Provide tight integration with popular SDR frameworks (i.e., GNU Radio)



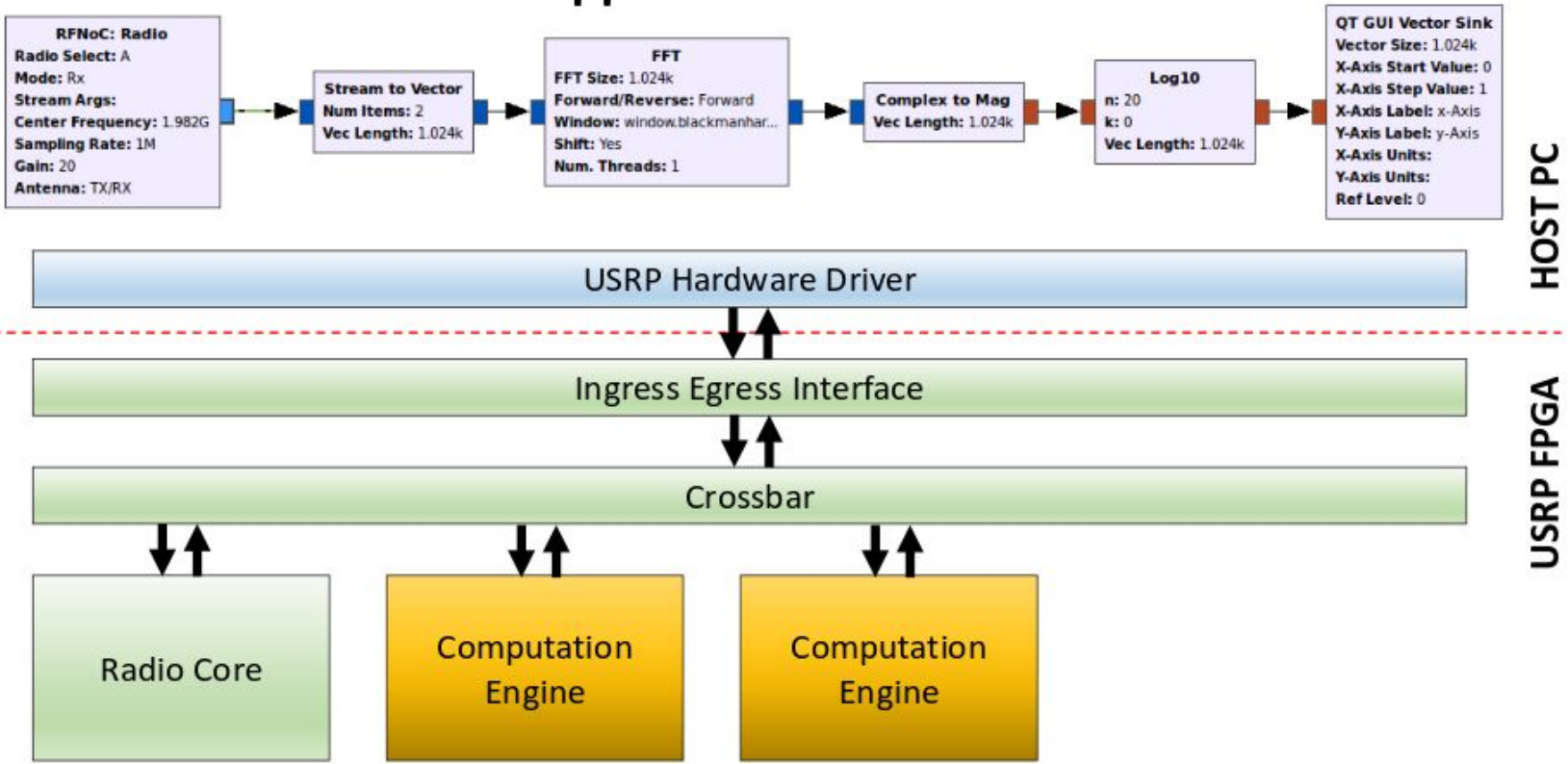
RFNoC (cont'd)

- RFNoC:
  - Provides a software API with an FPGA infrastructure
  - Manages communication and data flow between host and FPGA
  - Provides simple software and HDL interfaces to the user
  - Scalable design for massive distributed processing
  - Fully supported by UHD API, and in GNU Radio and GRC
  - Switching fabric is industry-standard AXI Crossbar
  - Provides space for user code in Computation Engines (CE)
    - many types of CE: FFT, FIR filter, cryptography, compression, etc.
    - CEs may be interconnected together in almost any order to create specific data flows
    - these data flows may cross the FPGA-host boundary
    - messages can be passed between the host and CEs

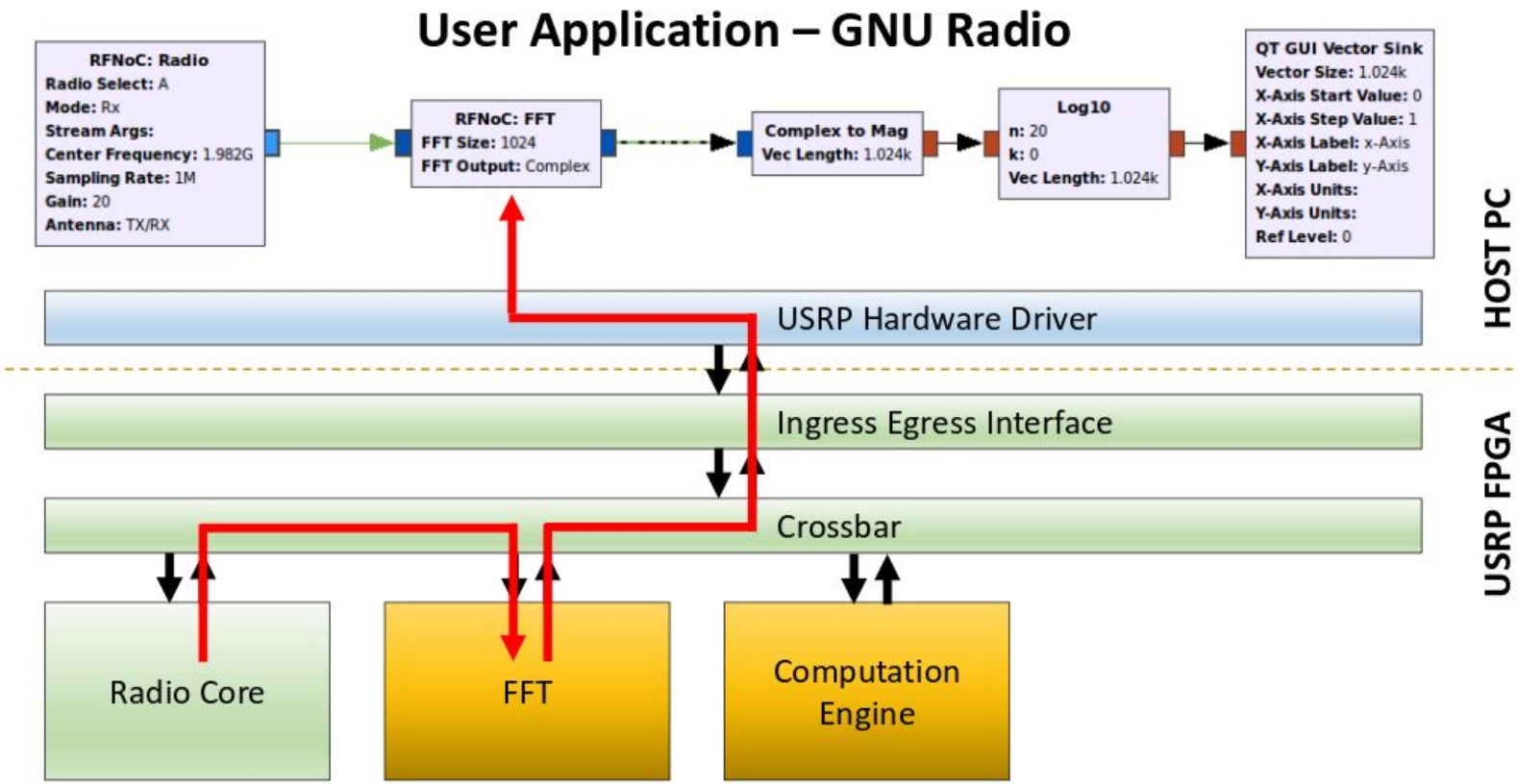
RFNoC (cont'd)

Ettus Research™
A National Instruments Company

User Application – GNU Radio



RFNoC (cont'd)


User Application – GNU Radio
RFNoC: Radio
Radio Select: A
Mode: Rx
Stream Args:
Center Frequency: 1.982G
Sampling Rate: 1M
Gain: 20
Antenna: TX/RX
RFNoC: FFT
FFT Size: 1024
FFT Output: Complex
Complex to Mag
Vec Length: 1.024k
Log10
n: 20
k: 0
Vec Length: 1.024k
QT GUI Vector Sink
Vector Size: 1.024k
X-Axis Start Value: 0
X-Axis Step Value: 1
X-Axis Label: x-Axis
Y-Axis Label: y-Axis
X-Axis Units:
Y-Axis Units:
Ref Level: 0
HOST PC
USRP Hardware Driver
USRP FPGA
Ingress Egress Interface
Crossbar
Radio Core
FFT
Computation Engine

RFNoC - Computation Engines

- CEs can be any mix of:
  - Blocks from the Ettus Research library (open-source)
  - User-defined blocks
  - Third-party Xilinx IP (closed-source)

RFNoC - CEs by Ettus Research

- Ettus Research provides a default FPGA image and a library of CE:
  - FIFO
  - FFT
  - FIR filter
  - window
  - vector IIR
  - keep-one-in-N
  - add/subtract stream
  - null source
  - null sink
  - split stream

RFNoC Availability and Status

- RFNoC is part of UHD, and is free and open-source
- Currently lives in its own UHD branch, but will be moved in the main UHD branch later this year
- Supported on E310, E312, E313, X300, X310
- Requires the Xilinx Vivado toolchain to build an FPGA image
- Available now, you can start using it today
- Requirements:
    - Use rfnoc-devel branch of UHD
    - Use GNU Radio 3.7.6 or newer
    - Use gr-ettus component of GNU Radio
    - Use gr-uhd component of GNU Radio
    - Xilinx Vivado, Design Edition or System Edition, version 2014.4
        - only needed if you want to build your own RFNoC-enabled FPGA images

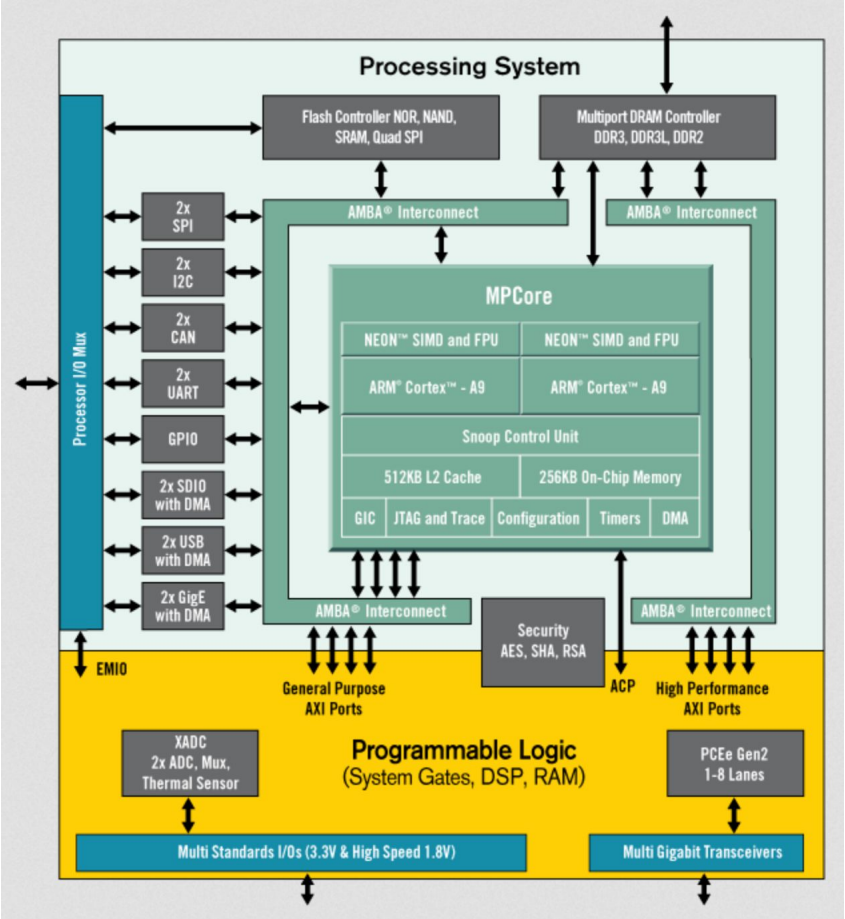
RFNoC Resources

- https://kb.ettus.com/RFNoC
- https://kb.ettus.com/Getting_Started_with_RFNoC_Development
- http://www.ettus.com/blog/2015/06/rfnoc-for-high-performance-sdr
- RFNoC presented at Wireless @ Virginia Tech, 2015
  - https://www.youtube.com/watch?v=8cPd3t88djE
- http://conferences.sigcomm.org/sigcomm/2013/papers/srif/p45.pdf
- http://www-int.etec.uni-karlsruhe.de/seiten/conferences/past/WSR2014/Papers/wsr14_21.pdf
- GRCon16 - It's the RFNoC Life for Us, Martin Braun
  - https://www.youtube.com/watch?v=51rpjJ2W0Qs

RFNoC - Zynq Family

- E31x Based on Zynq 7020
- Equipped with dual-core ARM Cortex-A9 processor (667 MHz SG1 / 866 MHz SG3)
- 85k Logic Cells
- 53,200 Loop-Up Tables (LUTs)
- 106,400 Flip-Flops
- 4.9 Mb Block RAM
- 220 DSP Slices
- 200 I/O Pins
- Compatible with the free Xilinx Vivado WebPACK Edition

RFNoC - Zynq Family



RFNoC - E3xx Linux

- E3xx runs customized Open Embedded (OE) Linux Distribution

"OpenEmbedded is a software framework used for creating Linux distributions aimed for, but not restricted to, embedded devices. The build system is based on BitBake recipes, which behave like Gentoo Linux ebuilds."

- Requires cross-compiling for ARM
- Ettus Research offers a free SDK which runs on Ubuntu/Fedora Linux
- Embedded CPU, local processing
- PyBOMBs recipes required to build
- No **`apt-get`**, uses "**`opkg`**"

RFNoC - AXI

- Advanced eXtensible Interface (AXI)
- Third generation of AMBA interface defined in the AMBA 3 specification
- Open Standard, on-chip interconnect specification
- Facilitates development of multi-processor designs with large numbers of controllers and peripherals
- Targeted at high performance, high clock frequency system designs and includes features that make it suitable for high speed sub-micrometer interconnect
- Separate address/control and data phases
- Support for unaligned data transfers using byte strobes
- Burst based transactions with only start address issued
- Issuing of multiple outstanding addresses with out of order responses
- Easy addition of register stages to provide timing closure

Using 10 Gigabit Ethernet

- X300 and X310 support both 1 and 10 Gigabit Ethernet interface
- Port 0 must be 1 GbE, and Port 1 must be 10 GbE
- Set the MTU to 9000 on 10 GbE:

```
sudo ifconfig eth0 mtu 9000
```

- Some motherboards do not provide enough PCI Express (PCIe) bus bandwidth to support higher sample rates. Motherboards with PCIe 3.0 are required, and the PCIe architecture of the motherboard should be carefully considered. Slots with dedicated PCIe lanes should be used for 10 GbE cards that will be connected to the X300.
- Intel and Myricom 10 GbE cards are recommended. Mellanox, SolarFlare, and Chelsio 10 GbE cards are not currently recommended. The Ethernet card should be plugged into the slot that has the most direct connection with the CPU (PCIe lanes are not shared with another slot).
- The Intel X520 card works very well out-of-the-box. This is the card that is sold on the company website.
- The Intel X710 card is the next generation 10 GbE card, and stable Linux support is not yet 100%, but will be soon
- Maximum suggested length for 10 GbE SFP+ copper cable is 3 m (10 ft)
    - Much longer ranges possible with optical fiber

Host System Performance Tuning

- The User Manual contains some information about performance tuning:

  `http://files.ettus.com/manual/page_usrp_x3x0_config.html`
- Many tools to monitor system performance: **`top`** **`htop`** **`iotop`** **`iostat`** **`sar`**
- Use an SSD drive, or even a RAM disk. Avoid spinning mechanical drives. Use the SATA-III interface.
- Use an up-to-date kernel. Install kernel 3.19 on Ubuntu 14.04 with:

  `sudo apt-get install linux-generic-lts-vivid`
- Power management (ACPI) on the host system attempts to save power by reducing clock frequencies or powering off devices while not in use. This can lead to significant performance issues when trying to operate at high sample rates. We strongly recommend disabling all power management. This may need to be done in the BIOS.
- Monitor the Ethernet interface for problems and errors with the command listed below. The output is driver-specific, but may give important clues as to what may be happening. For example, a high value on rx_missed_errors for an Intel NIC indicates that the bus (i.e., PCIe) is not keeping up.

  `ethtool -S eth0`

Host System Performance Tuning (cont'd)

- Allow UHD to set thread priority. When UHD spawns a new thread, it may try to boost the thread's scheduling priority. If setting the new priority fails, UHD prints a warning to the console, as shown below. This warning is harmless; it simply means that the thread will retain a normal or default scheduling priority.

```
UHD Warning:
        Unable to set the thread priority. Performance may be negatively affected.
        Please see the general application notes in the manual for instructions.
        EnvironmentError: OSError: error in pthread_setschedparam
```

- Non-root users need special permission to change the scheduling priority. Add the following line to the **`/etc/security/limits.conf`** file:

```
@GROUP    - rtprio  99
```

- Replace **`GROUP`** with a group in which your user is a member. You may need to logout and log back into the account for the settings to take effect. See the **`/etc/group`** file for a list of groups and group members.

Host System Performance Tuning (cont'd)

- The CPU governors dictate the frequency at which the CPU operates and attempt to reduce the CPU frequencies at certain times to save power. When running at high sample rates, reduction of CPU frequencies can cause significant performance issues. To prevent those issues, set the governor to the **`performance`** policy setting.
- Some documentation at:

  `http://files.ettus.com/manual/page_usrp_x3x0_config.html#x3x0cfg_hostpc_pwr_cpugov`
- On Ubuntu systems:
  - Install the cpufrequtils package: **`sudo apt-get install cpufrequtils`**
  - Edit file **`/etc/init.d/cpufrequtils`** and set governor policy setting on the appropriate line (run as root):

    ```
    sudo sed s/^GOVERNOR=.*$/GOVERNOR=\"performance\\"/g /etc/init.d/cpufrequtils > /etc/init.d/cpufrequtils
    ```
  - Restart cpufrequtils: **`sudo /etc/init.d/cpufrequtils restart`**
  - You may also need to run the following in order to keep your system from changing the governor:

    `sudo update-rc.d ondemand disable`
- On Fedora systems, change the CPU governor:

  `sudo cpupower frequency-set -g performance`

Host System Performance Tuning (cont'd)

- Interrupt coalescing, although it reduces CPU loading, can cause extra latency resulting in packet flow problems. To disable it, run:

  ```
  sudo ethtool -C eth0 adaptive-tx off
  ```

- Increase the number of descriptors used by the Ethernet driver, at the cost of higher memory usage:

  ```
  ethtool -G eth0 rx 4096 tx 4096
  ```

- Increase the maximum size of the kernel socket buffers to avoid potential overruns and underruns at high sample rates. Add the following entries into the `/etc/sysctl.conf` file (root privileges required):

  ```
  net.core.rmem_max=33554432
  net.core.wmem_max=33554432
  ```

  Either restart the system, or issue the following commands:

  ```
  sudo sysctl -w net.core.rmem_max=33554432
  sudo sysctl -w net.core.wmem_max=33554432
  ```

Host System Performance Tuning (cont'd)

- Use the function **`pthread_setaffinity_np()`** to dedicate a single core to each thread that calls **`multi_usrp::recv()`**, and use the same function on all the other threads in the program to avoid these cores
- Specify which cores should, and should not, process interrupts
  - Be sure to have at least of version 1.0.9 of irqbalance installed
  - Add the following line to the file **`/etc/defaults/grub`**

    `GRUB_CMDLINE_LINUX_DEFAULT="isolcpus=8,24"`
  - Add the following line to the file **`/etc/defaults/irqbalance`**

    `IRQBALANCE_BANNED_CPUS=1000100`
  - The IRQBALANCE_BANNED_CPUS is a bitmask to enable interrupt processing for selected cores.
  - Find your Ethernet interface in the file **`/proc/interrupts`**, take the first argument (the interrupt ID), and edit all files **`/proc/irq/<interrupt-id>/smp_affinity`** and write **`1000100`** to them

Applications - Cellular

- OpenBTS and OpenBTS-UMTS (Harvind Samra, David Burgess, Range Networks)
  - GSM, GPRS, WCDMA implementation
  - EDGE implementation (commercial)
  - **http://openbts.org/**
- srsLTE and srsUE (Paul Sutton, Linda Doyle, Trinity College)
  - **http://www.softwareradiosystems.com/**
  - **https://github.com/srsLTE/srsLTE**
- OpenLTE (Ben Wojtowicz, was Motorola, now Google)
  - **http://openlte.sourceforge.net/**
- Eurecom's OpenAirInterface (OAI)
  - **http://www.openairinterface.org/**
- Amarisoft (Fabrice Bellard)
  - Full LTE Release 12 eNodeB implementation
  - Commercial, not open-source
  - **http://www.amarisoft.com/**
  - **http://bellard.org/lte/**

Applications - GNSS Rx and Tx

- GNSS-SDR (Rx) and GPS-SDR-Sim (Tx)
  - Open-source, SDR-based GPS receiver and transmitter
  - `http://gnss-sdr.org`
  - `https://github.com/gnss-sdr/gnss-sdr`
  - `https://github.com/osqzss/gps-sdr-sim`
- Navigation Laboratories (NavLabs)
  - GNSS Simulator, computation performed on FPGA
  - Based in San Diego, California
  - `http://www.navlabs.com/`
- Skydel Solutions / Talen-X
  - GNSS Simulator, computation performed on GPU
  - Supports GPS (C/A code, P(Y) code, M code), Galileo, GLONASS, BeiDou
  - Based in Montreal, Quebec (Skydel) and Dayton, Ohio (Talen-X)
  - `http://www.skydelsolutions.com/`

Additional Resources

- GNU Radio Documentation and Wiki:
  - **`https://wiki.gnuradio.org/index.php/Main_Page`**
  - Locally at **`/usr/local/share/doc/gnuradio-3.7.9/html/index.html`**
- Ettus Research Knowledge Base (KB):
  - **`https://kb.ettus.com/`**
- USRP and UHD User Manual:
  - **`http://uhd.ettus.com/`**
  - **`http://files.ettus.com/manual/`**
  - Locally at **`/usr/local/share/doc/uhd/doxygen/html/index.html`**
- Additional Resources on the KB:
  - **`https://kb.ettus.com/Suggested_Videos`**
  - **`https://kb.ettus.com/Suggested_Reading`**

Getting Help and Technical Support

- Direct email address
  - `support@ettus.com`
- Mailing list **discuss-gnuradio**
  - `https://lists.gnu.org/mailman/listinfo/discuss-gnuradio`
- Mailing list **usrp-users**
  - `http://lists.ettus.com/mailman/listinfo/usrp-users_lists.ettus.com`
- Slack Workspace for GNU Radio
  - `https://slack.gnuradio.org/` *(to sign up)*
  - `https://gnuradio.slack.com/` *(to sign in)*
- IRC Channels (`irc.freenode.net`)
  - `#usrp`
  - `#gnuradio`

Upcoming SDR Events

- **Cyberspectrum Meetup**
  - Monthly meetup in the Bay Area / Silicon Valley
  - All 25 past events are archived online
  - **`http://www.meetup.com/Cyberspectrum/`**
- **NEWSDR 2019**
  - Thursday June 13 & Friday June 14
  - Boston, Massachusetts
  - Free registration ($0)
  - **`http://www.sdr-boston.org/`**
- **GNU Radio Conference 2019**
  - Week of September 16 to 20
  - Huntsville, Alabama
  - Registration is ~$500
  - The 2015, 2016, 2017, 2018 events are archived online
  - **`http://gnuradio.org/grcon-2019`**
- **DEFCON 27 & The Wireless Village**
  - Thursday August 8 through Sunday August 11
  - Las Vegas, Nevada
  - Registration is $280
  - Past events are archived online
  - **`https://www.defcon.org/`**
  - **`https://www.wirelessvillage.ninja/`**