GNU Radio

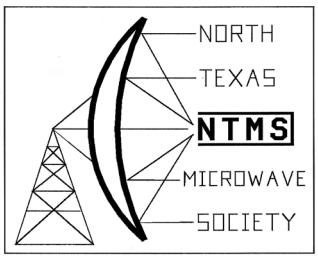
Workshop

Presented at Microwave Update

October 3 2019

Lewisville, Texas

Microwave Update 2019
Sponsored by:
North Texas Microwave Society



Publish by:

ARRL Logo

Printed in USA.

ISBN: xxxxxxxxxxxx

First Edition

Cover photo credit if any.

Table of Contents

Introduction

Slides presented at workshop

Additional Information supporting GNU Radio

References

Introduction

With the growing use of software defined radio (SDR) in communications and Amateur Radio this workshop is an effort to teach the application and approach which may be followed by people interested in developing their own working radio systems using GNU Radio. GNU Radio (GNR) is a tool developed for simulation and implementation of SDR concepts. GNR is dynamic as it is growing in its capability and expanding the number of physical hardware supported. This growth will likely continue especially as the supported hardware evolves.

GNR allows you to quickly get productive results and make changes in the operation or performance of your equipment. Its a powerful learning tool.

While the idea of learning about Digital Signal Processing (DSP) and the methods needed to implement a complete radio may seem overwhelming to many. GNR is just the tool to help you evolve into this space without too much pain.

We hope your participation in the workshop and for those of you that obtain the material later find this enjoyable and contribute to the continued development in your own way. You will be able to develop working systems and we hope you use them and share them with everyone.

Microwave Update 2019
Gnuradio Workshop

October 3, 2019

Tom McDermott, N5EG
Bob Stricklin, N5BRG
Jenner Lochridge, KK6RUM

NOTES:

Agenda

1. Install Gnuradio on Windows 10 64-bit OS.
2. Review of GNU Radio capabilities and core concepts.
3. Review of important GNU Radio modules, building a project, implementing and running projects involving hardware.
4. Use of Gnuradio Companion (GRC) graphical environment.
5. Demonstration of Gnuradio Companion (GRC) application with SDR Radio.
6. Ideas, Examples, and Hints
7. After Seminar – Breakout sessions with different radio demos.

NOTES:

Exercises & Demo

1. Load existing flowgraph from file, and run.
   - Real signals and Spectra
2. Modify flowgraph from Exercise #1.
   - Complex signals and Spectra.
3. Add noise and low pass filtering.
   - Tap spectrum
4. Create flowgraph to playback an existing recording from a file.
5. Demonstration – 2.3401 GHz SSB SDR radio.

NOTES:

Gnuradio on Windows 10 64-bit OS.

- Must be x86_64 type processor (not ARM).
- Prebuilt Windows Installer (MSI file) image.
  - You don't need to compile or build anything, just install.
- From MUD-workshop supplied USB stick.
- Latest version available at:
  http://www.gcndevelopment.com/gnuradio/downloads.htm
- Limitation: Pre-built image only supports radios that the maintainer chose to include.
  - Several widely popular radios are included.
  - You cannot (easily) add new modules or radios.

NOTES:

Gnuradio Core Concepts - 1

- Open Source software - Handles real-time DSP, Simulation, I/O, and buffer management.
- Based on Python and C++
  - Python for the interconnection, graphics, management.
  - C++ for high-speed DSP functions, buffers, and I/O.
  - The installer takes care of setting up everything for you.
- A flowgraph is a Python program:
  - A text file containing the Python source code.
  - Defines the DSP blocks, GUI blocks, Radio, Sources and Sinks.
  - Parameter values, how everything is wired up.
  - What your flowgraph looks like on the GUI display.
- You don't need to know Python or C++
  - Gnuradio Companion (GRC) is a graphical GUI.
    - Allows you to enter flowgraph modules, parameters, and wiring with a GUI tool.
    - Compiles your flowgraph into Python program and saves the file to disk.
    - Allows you to Start and Stop your flowgraph: Invokes Gnuradio for you.
- Create and run DSP without writing code.

NOTES:

Gnuradio Core Concepts - 2

Gnuradio handles real-time buffering.

There cannot be a loop in a flowgraph.

A 'throttle' is needed when there is no source of timing.

- A radio, or a soundcard (output or input) are all sources of real timing.
- The throttle prevents the CPU from working at 100%.
- GRC will warn you if it thinks you need one.
- It can be in any data path. If needed, use only one throttle.
- Windows audio sink can be a little goofy. Sometimes needs a throttle.

NOTES:

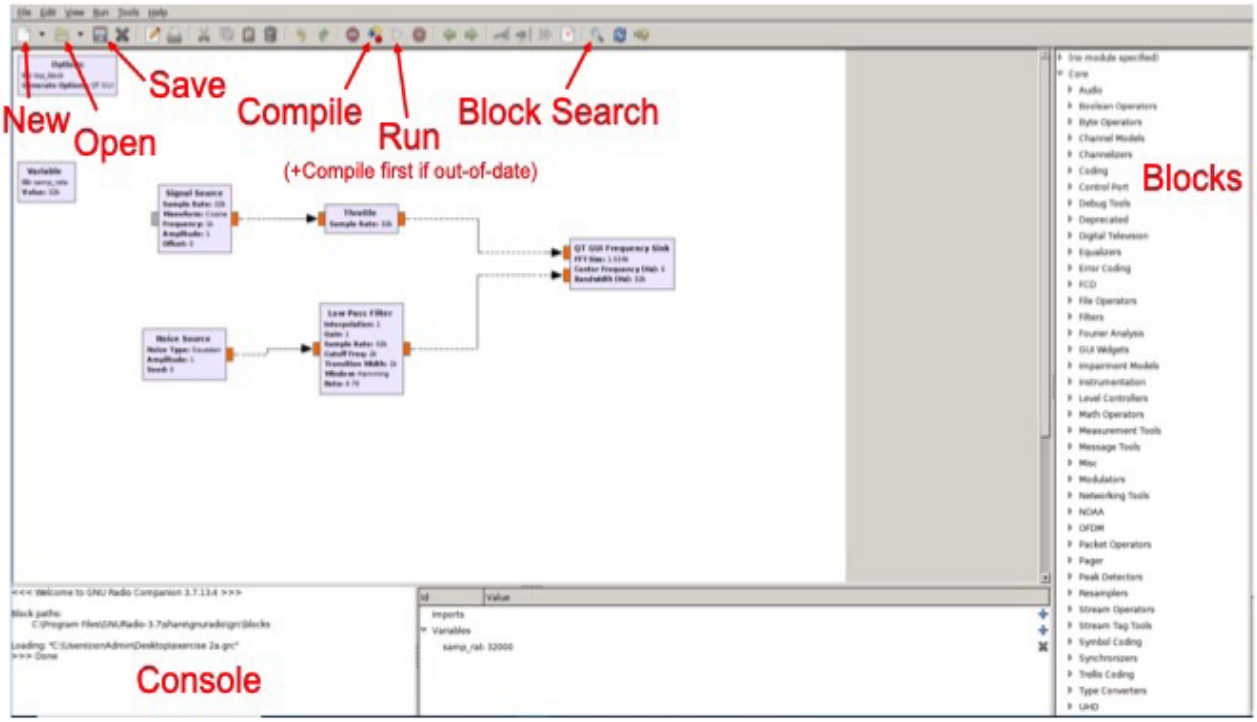
Windows Install Steps

1. Insert USB stick in computer.
2. Copy all the files to a folder on your desktop.
3. Click 'safely remove' then remove the USB stick.
4. Double-click the MSI file ICON to start the installation process.

- The installed software creates an ICON to start Gnuradio Companion each time you want to run.
  - Sets up the entire environment then Launches.
- Also installs the Gnuradio Manual.
  - No ICON, point your web browser to the index file and click to open with browser.
  - C:\Program Files\GNURadio-3.7\share\doc\gnuradio-3.7.xx.yy\html\index.html
  - xx, yy are the version numbers.

NOTES:

The GRC Main Window



NOTES:

Building a Flowgraph with GRC

Exercise #1

1. Open GRC by clicking the GRC ICON.
2. Open supplied existing flowgraph: Exercise_1.grc
3. Click the compile (build) icon.
   - Creates a flowgraph: top_block.py
   - You can rename this to anything Windows understands.
4. Click the run icon.
   - Flowgraph starts, displays flowgraph GUI.
   - If you forgot to build – no worries, GRC will rebuild automatically.
5. Close the flowgraph GUI to terminate the flowgraph.

NOTES:

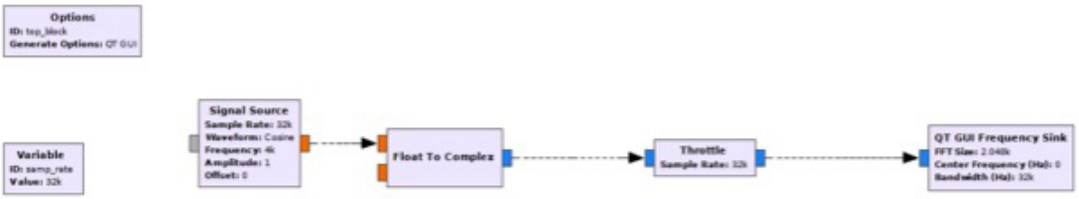
File Name Convention

- Created by graphically editing a flowgraph. File created when you click 'Save' or 'Save As"
- Describes:
  - Blocks, where they are on the layout sheet, rotation, orientation.
  - Block parameter values.
  - How blocks are wired together.
- Not executable, must create the Python executable.

- Auto-generated each time you build or run the flowgraph.
- Executable Python code.
- Default name: top_block.py
- Can be hand-edited. But rename the file or it will get overwritten!
- Name can be changed in Options block.
- Can also be executed from Gnuradio Python command line.

NOTES:

Exercise 1 Flowgraph



Defaults:

- QT GUI – provides graphical display
- samp_rate – sets the sample rate to 32 ksps. Easily changed.

Notes:

- Blue – type: complex Orange – type: float (real)
- GUI Frequency Sink – spectrum analyzer style display

NOTES:

Gnuradio Core Concepts - 3

- A 'source' is something that streams samples into your flowgraph.
  - Could be a radio, an audio soundcard source, a file, a TCP (connection…
- A 'sink' is something that removes samples from your flowgraph.
  - A radio, soundcard sink, a file, a TCP connection…
- General blocks receive samples, transform them, might do other things, then outputs modified (or not) samples.
  - Examples of general blocks:
    - Low Pass Filter
    - FM Demodulator
    - Delay block

NOTES:

Gnuradio Core Concepts - 4

Signal formats are color-coded.

- Blue: Complex single-precision float 32 (I + Q).
- Orange: Single-precision float 32.
- A bunch of others. Help→Types to display handy pop-up legend.

GUI will only wire together two pins if they are of the same type.

Select block then use arrow-up and arrow-down to scroll through the types supported by the block.

There are type-converter blocks available.

- Complex → Float has one blue input and two orange outputs.
- Can you explain why?

NOTES:

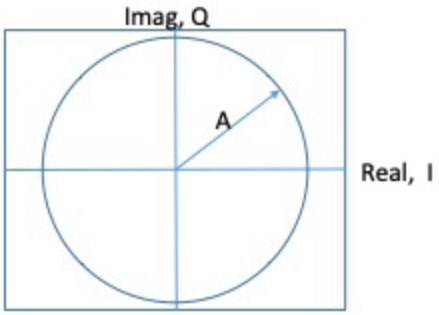
Complex vs. Real

- A complex signal contains both an In-Phase (I) signal and a Quadrature-Phase (Q) signal.
  - Real = one floating point number
  - Complex = a pair of floating point numbers.
  - Gnuradio keeps the complex number parts together as a pair.
- A complex number describes a vector on the complex plane.
  - CCW rotation once/sec = positive 1 Hertz.
  - CW rotation once/sec = negative 1 Hertz.
- A real number has no rotation – the vector exists only on the real axis.
  - Cannot differentiate negative frequency from positive frequency.
  - Therefore: it's both frequencies at the same time.

NOTES:

Complex notation

- Single frequency = A { cos (ωt) + i sin (ωt) } = A $e\uparrow i\omega t$
- Cos = I i Sin = Q A = Amplitude of the signal
- ω = frequency * 2π



NOTES:

Gnuradio Core Concepts - 5

- Each block must know about the sample rate of the samples coming into it.
- Some blocks can change the sample rate.
  - For example 'decimation'. 'Keep one-in-N'
  - Make sure you low pass filter before decimation or you violate Nyquist !
- Gnuradio creates a default variable samp_rate.
  - If you have interpolation or decimation, you may need multiple sample rate variables because you have several different sample rates at the same time in your flowgraph.
  - Many gnuradio blocks combine some function with decimation.
    - For example Low-Pass-Filter has adjustable decimation.
- Tip: Reduce the sample close to the source in your flowgraph to reduce CPU workload.

NOTES:

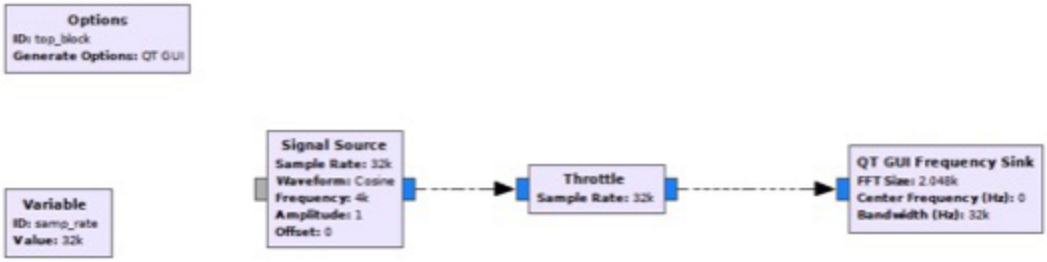
Modify a Flowgraph with GRC

Exercise #2

1. Launch GRC
2. If not already open, then Open the example from exercise #1.
3. Change the file name (otherwise will overwrite the previous file).
   1. Click File → Save As to save the grc file with a new name.
4. Modify the following blocks to your flowgraph to change format:
   1. Signal source
   2. Delete the Float to Complex converter
5. Run the flowgraph. What's different? Does it do what you expect?
6. Optional – Rename compiled Python filename:
   1. Defaults to top_block.py
   2. Double click on the Properties block, then modify the properties block.
   3. This changes the generated Python block name, but does not rename the flowgraph grc file.

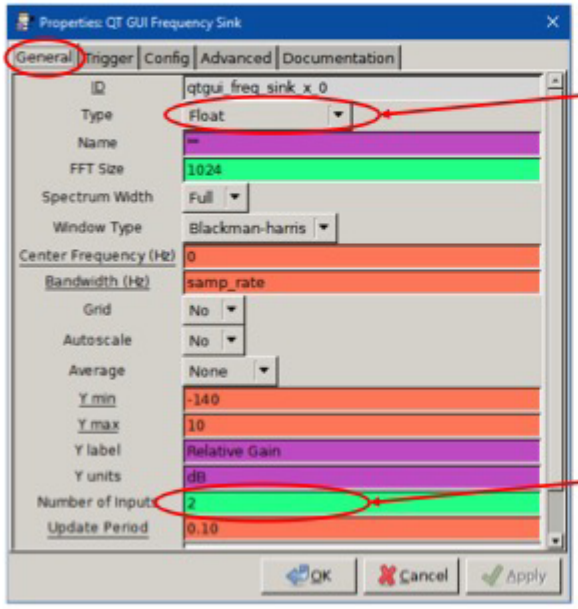
NOTES:

Exercise 2 Flowgraph



NOTES:

Modify the QT Frequency Sink



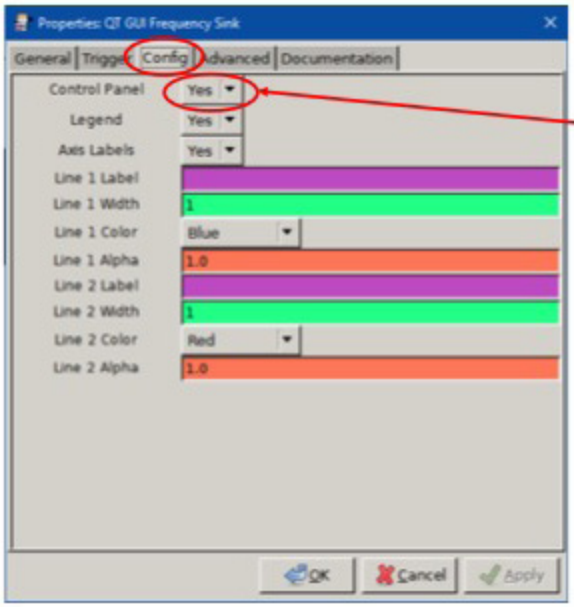
The data format type

- Parameters can be any valid Python expression evaluating to float, int, string, or boolean (as appropriate).
- Examples:
  - -140
  - float(x * 2) (x times 2)
  - int(10**y) (10^y power)

Number of channels

NOTES:

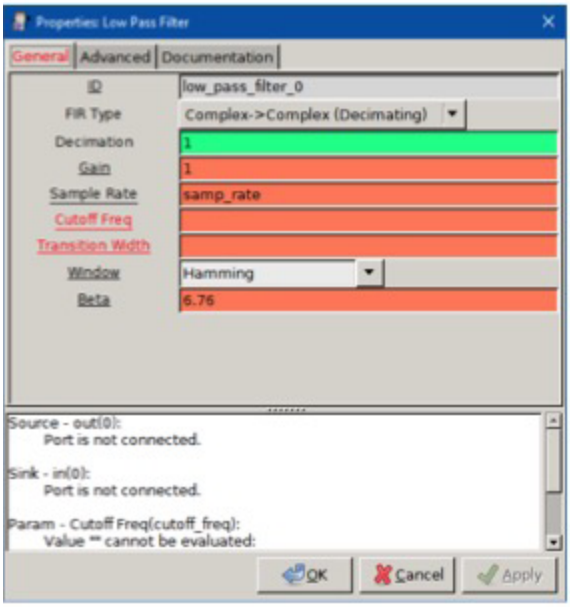
Additional QT Frequency Sink options



Turns on the run-time GUI Control Panel. Very useful.

NOTES:

Low Pass Filter



- Provides Common Filter Shapes.
- Complex → Complex (Input to Output).
- Decimation reduces output sample rate by value
- Gain.
- Sample Rate usually specified from flowgraph variable, but can be manually set.
- Cutoff Frequency: -6 dB point in Hertz, must be entered.
- Transition width: filter roll-off in Hertz, from -6 dB to stopband.

- LowPassFilter passband goes from –Cutoff to +Cutoff.
  - Real taps
  - This may be twice as wide as you are expecting!

NOTES:

Modifying Blocks

Exercise #3

Modifying blocks (double-click a block to modify):

- Can set the type in the windows, or
- Shortcut: single-click (select) the block, use ↑ and down ↓ keys to scroll through available formats.

Renaming

- File → Save saves the 'source code' of the flowgraph. Suffix: grc
- Compiling the file saves the compiled flowgraph. Suffix: .py
  - Default: top_block.py. Can rename the flowgraph by modifying the clicking on and modifying properties block.

Wire the blocks together per the diagram.

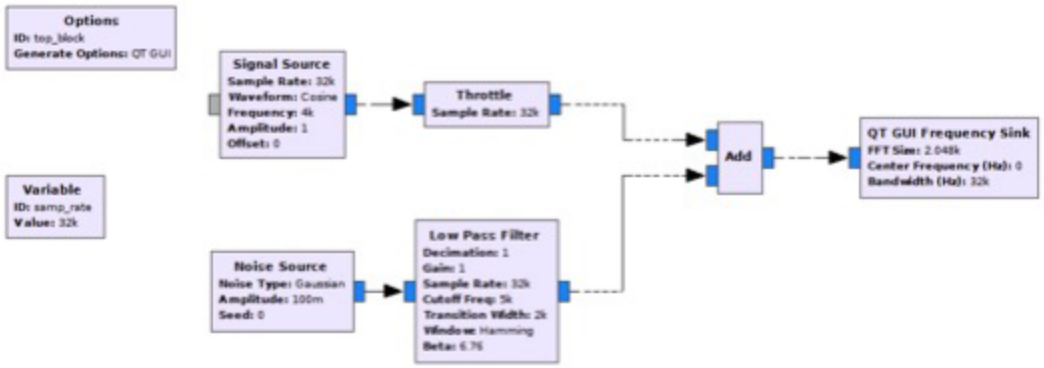
- Click on source port then click on destination port to wire together.
- A block with a **Red Title** has some kind of error – flowgraph won't run.
- Click a wire to highlight it, then hit Delete key to remove it.

Click run (will auto build).

- Will ask where to save it, and what to call it (only the first time).
- Note any errors. Fix, then try again (click run).

NOTES:

Exercise #3



- Rename the flowgraph.
- Add lowpass filter, Gaussian noise source.
- Run the flowgraph. What can you say about the low pass filter behavior?

NOTES:

Things to note about Exercise #3

Note the spectral response !

- What can we say about the spectrum of a complex sinusoid?
- What can we say about the Low Pass filter response?

What does negative frequency mean?

NOTES:

Some Key Gnuradio Modules - 1

- Filters. General types: Lowpass, Bandpass, Highpass.
  - Pre-made filter types available.
  - Can make custom filters if necessary.
- Transfer function defined by the taps.
  - Pre-defined filters use pre-created taps. Fast & easy way to implement standard filters.
  - GRC includes a filter designer GUI. Use it to create taps.
    - Tools → Filter Design Tool
  - Large # taps substantially increases computational resources.
  - Real taps → both positive and negative frequency response.
  - Complex taps → single-sided frequency response.

NOTES:

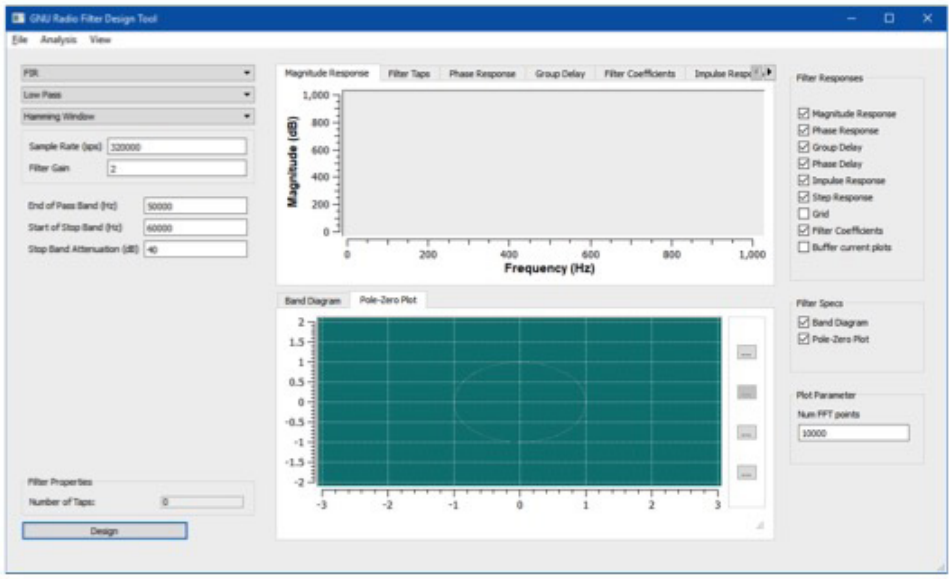
Some Key Gnuradio Modules - 2

- Add
  - Adds two signals sample-by-sample.
- Multiply
  - Multiply two signals sample-by-sample.
  - Implements mixer (frequency shifter) or Gain / Attenuate.
- Convertors
  - Float → Complex
  - Complex → Float
  - Stream → Vector
- GUI : scope, spectrum analyzer, constellation, waterfall.
- Many more. Explore on your own.

NOTES:

Filter Designer

Tools → Filter Design



- Use to create custom filters.
- Creates 'filter taps' file that can be read by filter block.
- Standard filter blocks don't need any of this.

NOTES:

Sample Rate & Decimation

- You must keep track of the sample rate throughout your flowgraph. Otherwise bad things happen.
  - Decimation & interpolation change the sample rate.
- Flowgraph creates a variable: samp_rate
  - You can use that name, or change it. You can make additional variables (for any purpose).
- Blocks that depend on the sample rate normally should use a variable.
  - Benefit: If you change the sample rate, then all blocks using that variable change along with it. Otherwise you have to hunt down and change all the rate-dependent blocks.

NOTES:

Decimation

- Normally the radio produces samples at too fast a rate – overwhelms CPU capability.
- Once you have isolated the frequencies of interest, frequency shift to zero Hertz, then decimate.
- Decimation reduces the sample rate by a factor of N.
- Nyquist criteria: You must low-pass-filter to ±Fs/2N or less before decimating.
  - Blocks downstream of the decimation thus process much fewer samples.
- Match sample rate between devices.
  - Example: Radio (perhaps 192 ksps) to Soundcard (perhaps 48 ksps).
    - LPF to less than 24 kHz (passband + roll off < 24k) Nyquist.
    - Then decimate by 192 / 48 (i.e. decimate by 4).

NOTES:

Multiplier

- A Complex multiply produces a frequency shift.
- Complex multiply *does not* produce sum and difference frequencies – only produces the sum frequency.
  - Real multiply *does* produce sum and difference frequencies.

- Negative-frequency carrier used to down-convert. Two ways:
  - Enter frequency as a negative number.

    or
  - Take complex conjugate of positive frequency (negates the imaginary part).

NOTES:

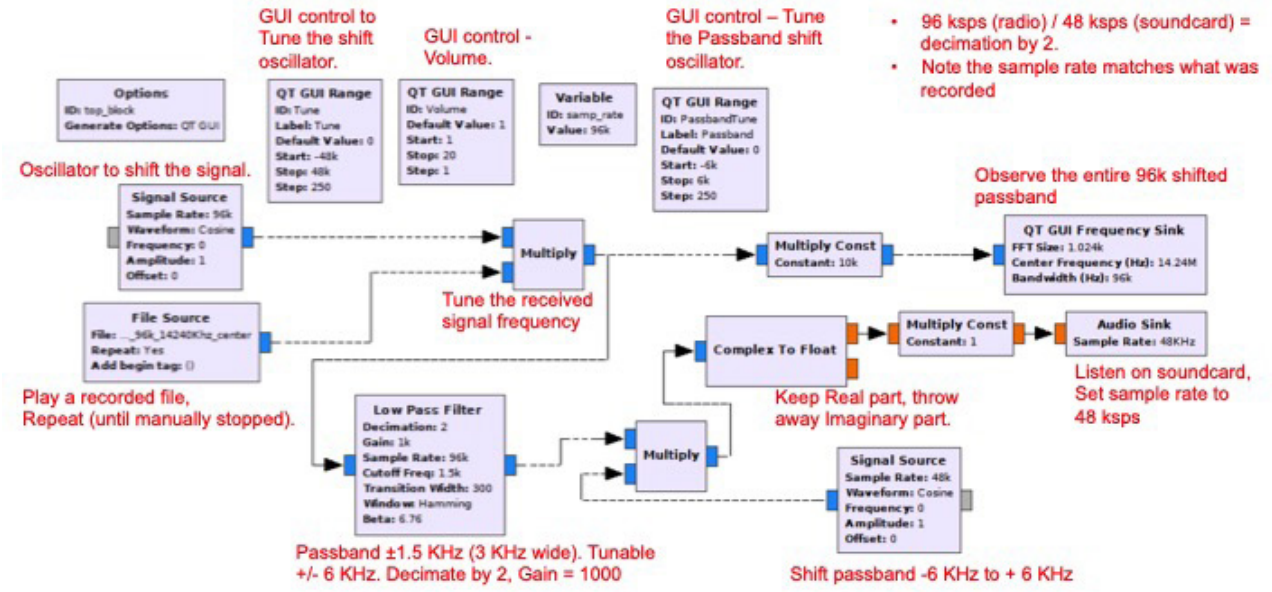
Down-convert + Filter + Listen

Exercise #4

- Stream samples from a file (pre-recorded).
  - "15mins_NAQP_SSB_96k_14240Khz_center" (*No .extension*)
- Visualize the spectra.
- Down-convert and Tune a selected channel.
- Tune the filter passband.
- Filter and decimate.
- Listen on your soundcard.

NOTES:

Exercise #4 Play pre-recorded file (SSB)



NOTES:

Radios

- A radio acts as source (receiver) and sink (transmitter).
  - Receiver source can replace the file source in the previous exercise or signal source in earlier exercises.
- Different radios have different source and sink blocks.
- A 'radio' tuning behavior will be slightly different than the previous 'file' tuning behavior.
  - Can you explain what that difference is?
- Some radios have separate source and sink blocks – even though they are connected to the same radio.
- Some radios combine source and sink all into one block.

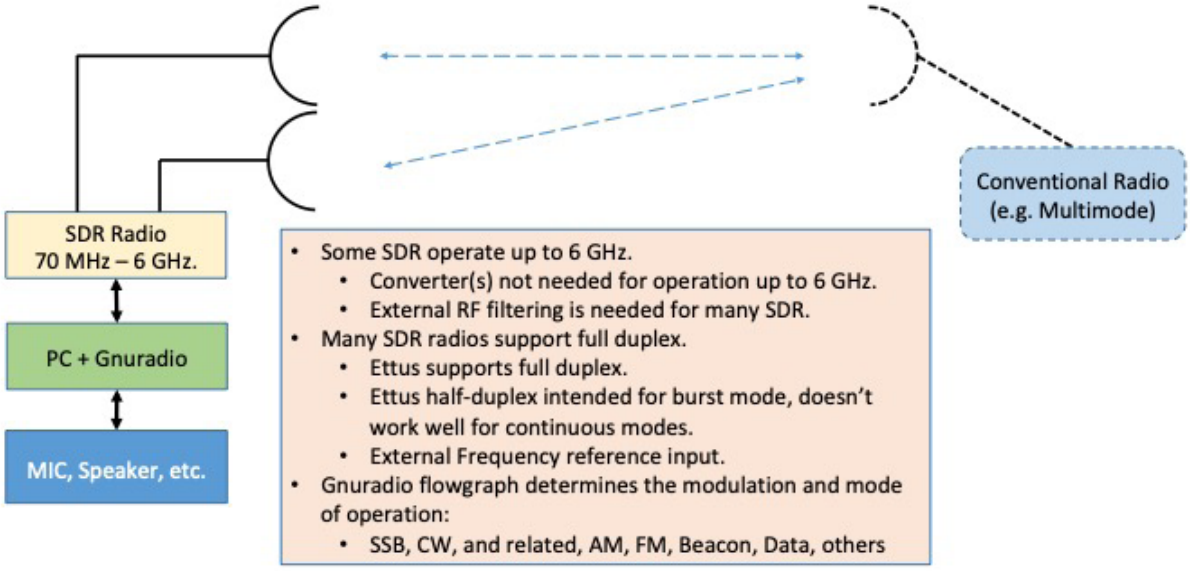
NOTES:

Microwave SDR Transceiver Demo

- Use Flexibility of GNR + SDR
  - Do I.F., filtering, modulation & demodulation in the SDR
  - Adjust LO and SDR operation to match what you have available
- Pick operation points for performance
  - Select LO to fit band plan – QRM input and output
  - Make filter easier to build/find
- Performance of components vs. frequency can drive design
- Gnuradio + SDR can reduce components in design and make changes easier.
- Some SDR can cover up to 6 GHz without additional conversions.
  - Multi-band radio can be simpler, smaller, and less expensive.
- 10+ GHz usually requires analog up & down conversion.

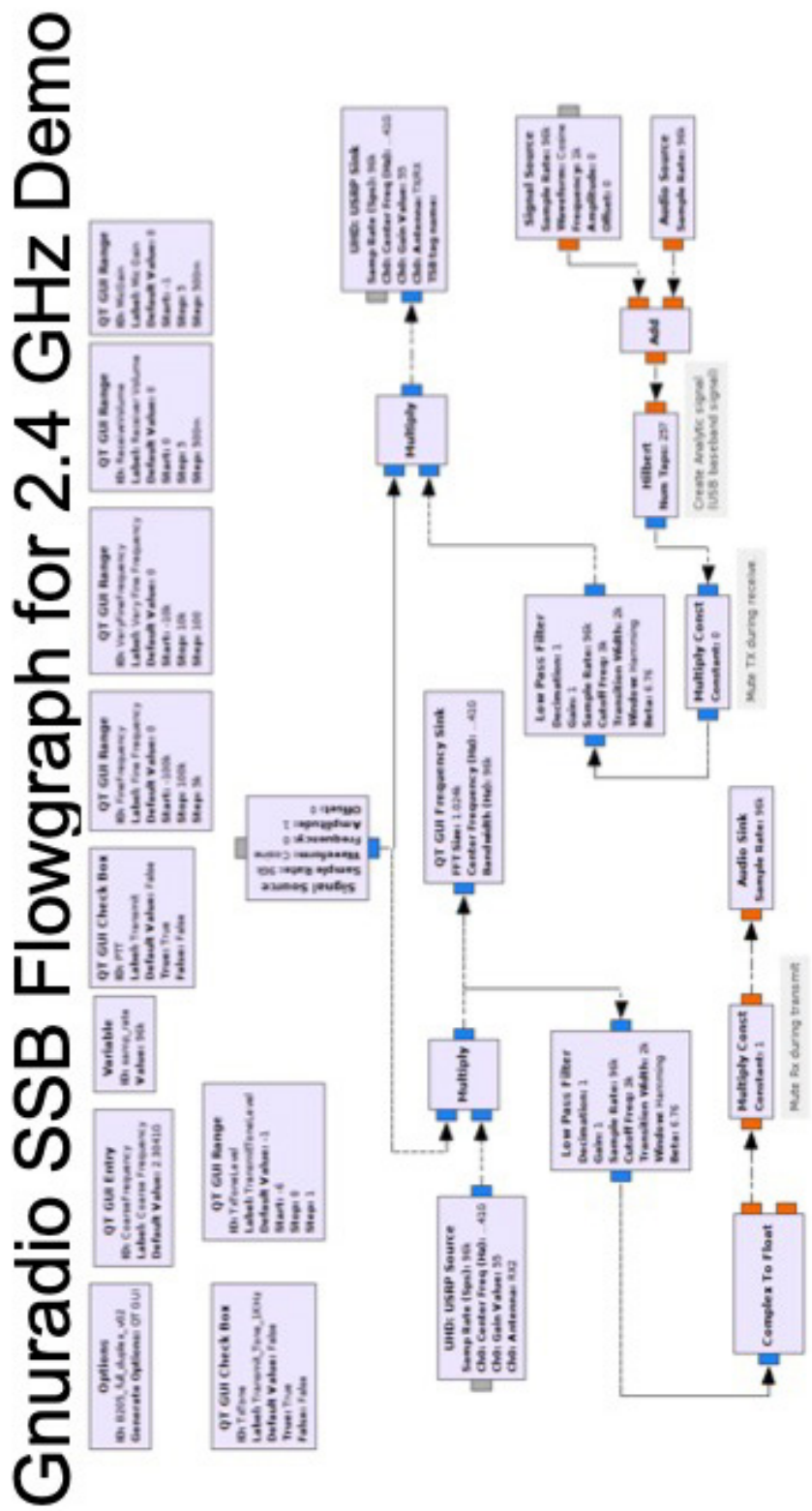
NOTES:

Microwave Demo Block Diagram: 2304.1 MHz SSB



- Some SDR operate up to 6 GHz.
  - Converter(s) not needed for operation up to 6 GHz.
  - External RF filtering is needed for many SDR.
- Many SDR radios support full duplex.
  - Ettus supports full duplex.
  - Ettus half-duplex intended for burst mode, doesn't work well for continuous modes.
  - External Frequency reference input.
- Gnuradio flowgraph determines the modulation and mode of operation:
  - SSB, CW, and related, AM, FM, Beacon, Data, others

NOTES:


Gnuradio SSB Flowgraph for 2.4 GHz Demo
Options
QT GUI Entry
Variable
QT GUI Check Box
QT GUI Range
QT GUI Check Box
QT GUI Range
QT GUI Range
QT GUI Range
QT GUI Range
QT GUI Range
Signal Source
QT GUI Frequency Sink
Multiply
Multiply
UHD: USRP Source
UHD: USRP Sink
Low Pass Filter
Low Pass Filter
Multiply Const
Multiply Const
Audio Sink
Complex To Float
Signal Source
Audio Source
Add
Hilbert
Create Analytic signal (USB baseband signal)
Mute TX during receive
Mute Rx during transmit

Post-Seminar Breakout

- There will be several different radios being demonstrated in this room for the next ~ hour.
- Find a demo or radio you are interested in and talk with the instructors.
- List of radio demos:
  - Pluto
  - RTL-SDR
  - Ettus – WiFi Spectrum Analyzer
  - Other?

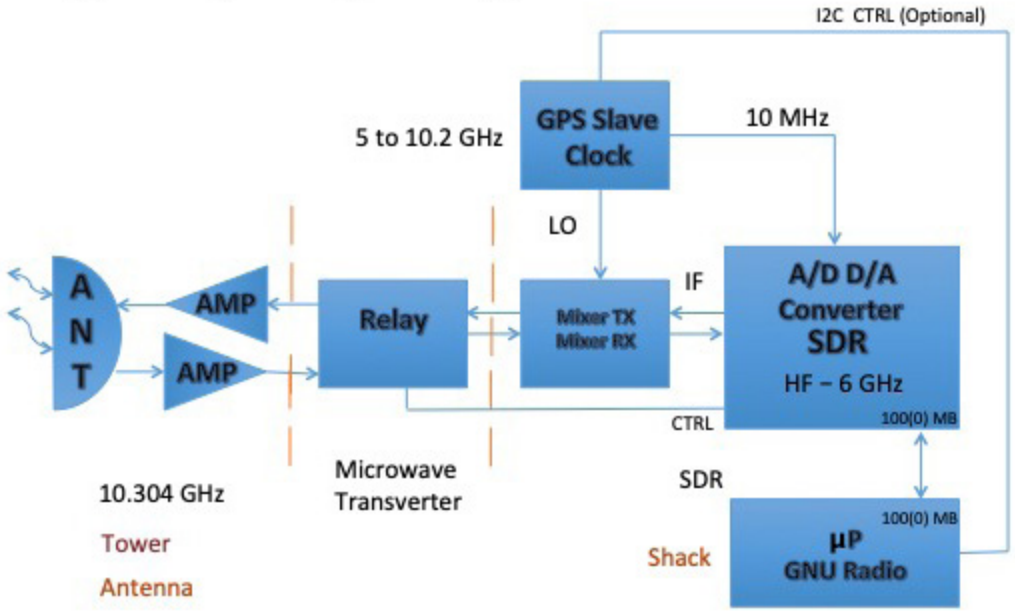
NOTES:

Example, Ideas, and Hints

- Extending frequency range beyond SDR capability.
- Measuring Cable / Other device attenuation at Microwave frequencies using SDR.
- Beacon

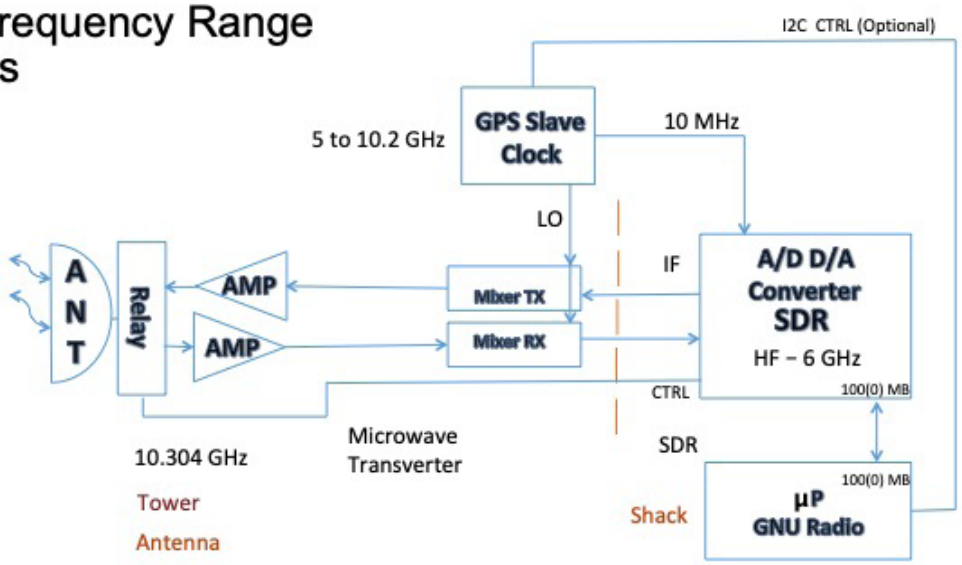
NOTES:

Extending Frequency Range



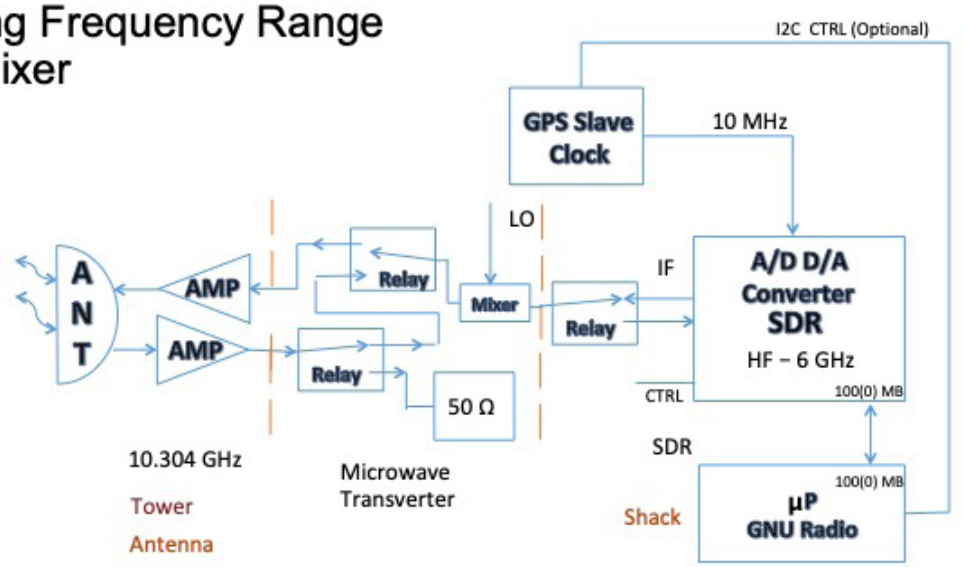
NOTES:

Extending Frequency Range – Two Mixers



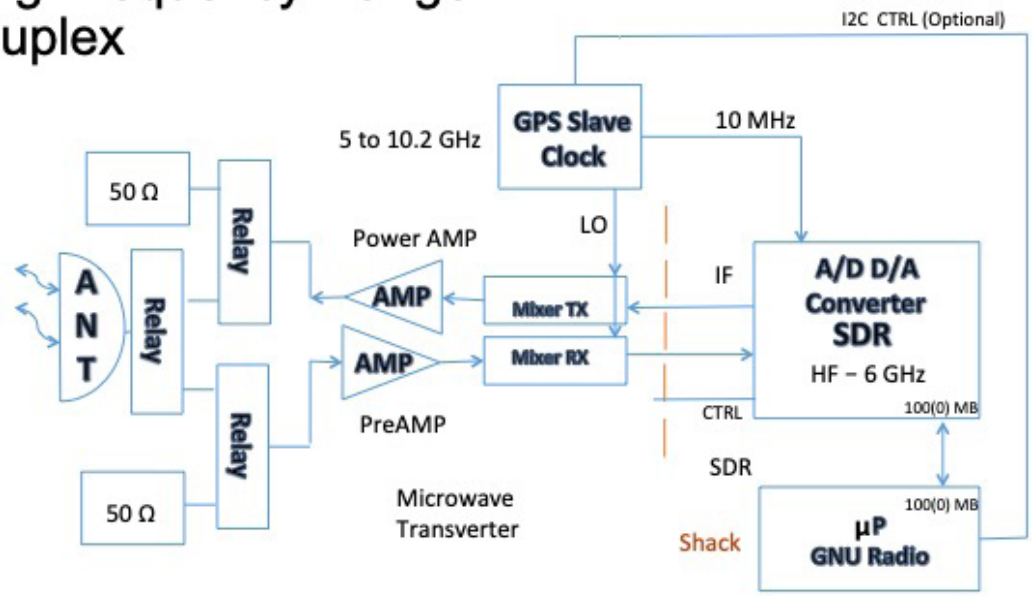
NOTES:

Extending Frequency Range – One Mixer



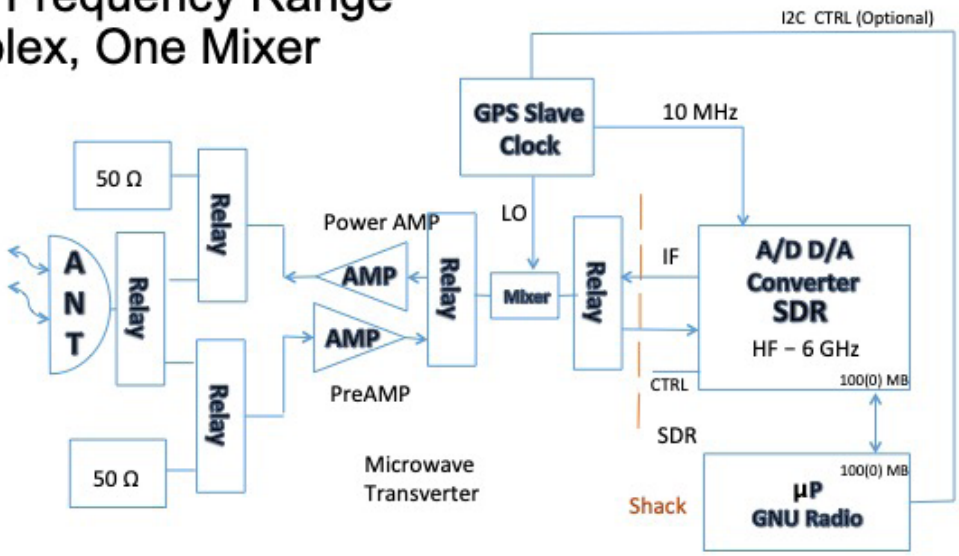
NOTES:

Extending Frequency Range – Half Duplex



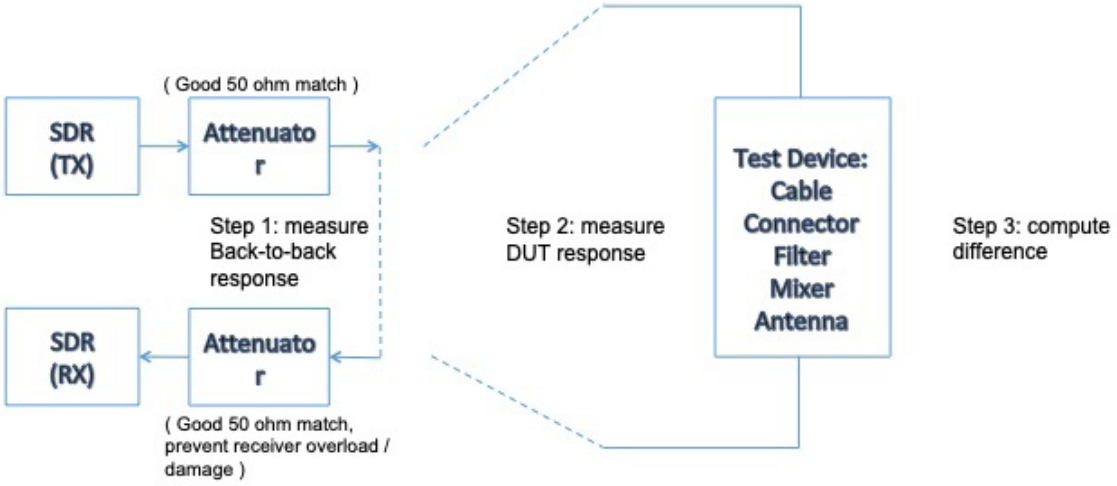
NOTES:

Extending Frequency Range – Half Duplex, One Mixer



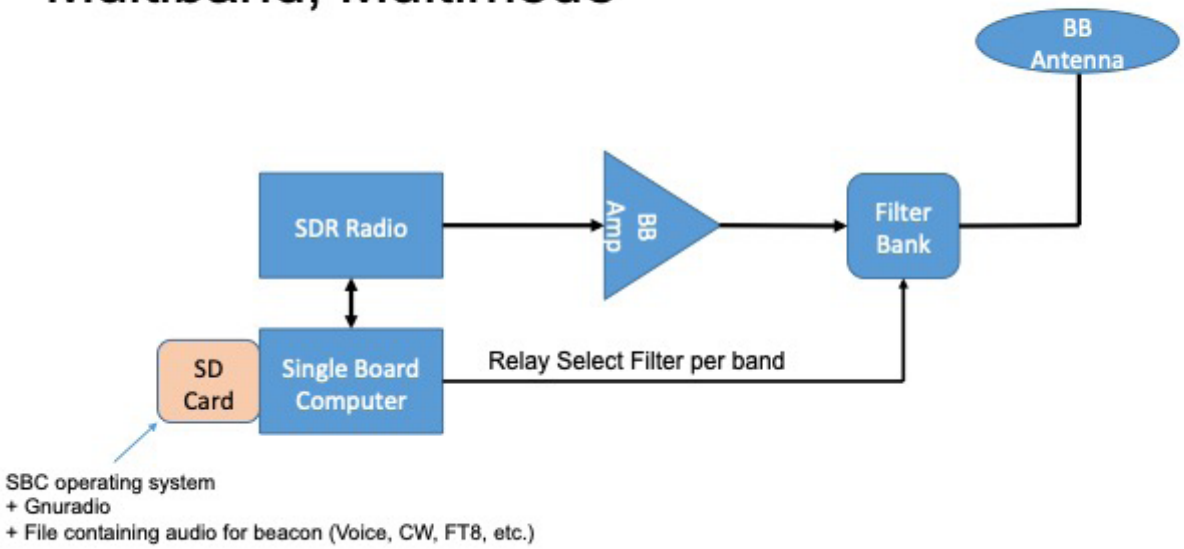
NOTES:

Measuring Loss vs. Frequency



NOTES:

Automated Low Cost Beacon: Multiband, Multimode



NOTES:

Help

- Gnuradio installs help on your PC:
  - C:\Program Files\GNURadio-3.7\share\doc\gnuradio-3.7.xx.yy\html\index.html
  - xx.yy corresponds to your version number (should be the only such folder).
  - Open with web browser. Maybe make a URL shortcut on your desktop?
  - Auto-generated: Help content thoroughness varies by module.
- Gnuradio Examples:
  - C:\Program Files\GNURadio-3.7\share\gnuradio\examples
- Main Page On Line:
  - https://wiki.gnuradio.org/index.php/Main_Page
- Gnuradio tutorials (excellent):
  - https://wiki.gnuradio.org/index.php/Tutorials
- Gnuradio mailing list (very busy):
  - https://wiki.gnuradio.org/index.php/MailingLists
- Ettus mailing list (Ettus radio only, busy):
  - http://lists.ettus.com/pipermail/usrp-users_lists.ettus.com/

NOTES:

Additional Information supporting GNU Radio

ADLM-Pluto Crystal Change

In the case of the ADLM-Pluto one possible improvement in performance would be to replace the crystal (Y3) used in manufacturing. I found some discussion about this doing a web search and the crystal shown below was suggested as a replacement. I collected the specifications on the two crystals and have them in Table 1.

Mfg	**ABRACON**		**Rakon**		
Part #	ASTX-13-C-40.000MHz-I05-T		RX03225M		
Duty cycle			45 to 55	%	
Output load	10	pf	15	pf	
Rise time/fall time			5 max	nS	
RMS phase Jitter	11.7	pS Calculated	0.3	pS	
Freq Stability	0.5	ppm	25 max	ppm	<Key
Operating Temp	-30 to 75	C	-55 ro 85	C	
Phase Noise @ 1 kHz	-130	dBc/Hz			
Vdd	1.8	V	3.3 max	V	
Id	2	mA	10 max	mA	

Table 1

The Frequency stability is the parameter that indicates the Abracon crystal is a better choice. I purchased some of these from the distributor Mouser Electronics. Using a hot air soldering station I removed the crystal on the ADLM-Pluto and replaced it with the Abrasion 40 MHz crystal. The original crystal seems to have been soldered on with a high temperature lead-free solder so it took a fair amount of heat to remove it. The pads on +1.8V and GND are probably also connected to copper planes which sink heat away. The replacement part is smaller so it gives you room to work with the pads and solder it in place.

If you do this pay attention to the part number because the C in the part number indicates the operating voltage of 1.8 V, which is what the ADLM-Pluto, is designed to use.

The location of the crystal (Y3) is shown in the photo of Figure 1.

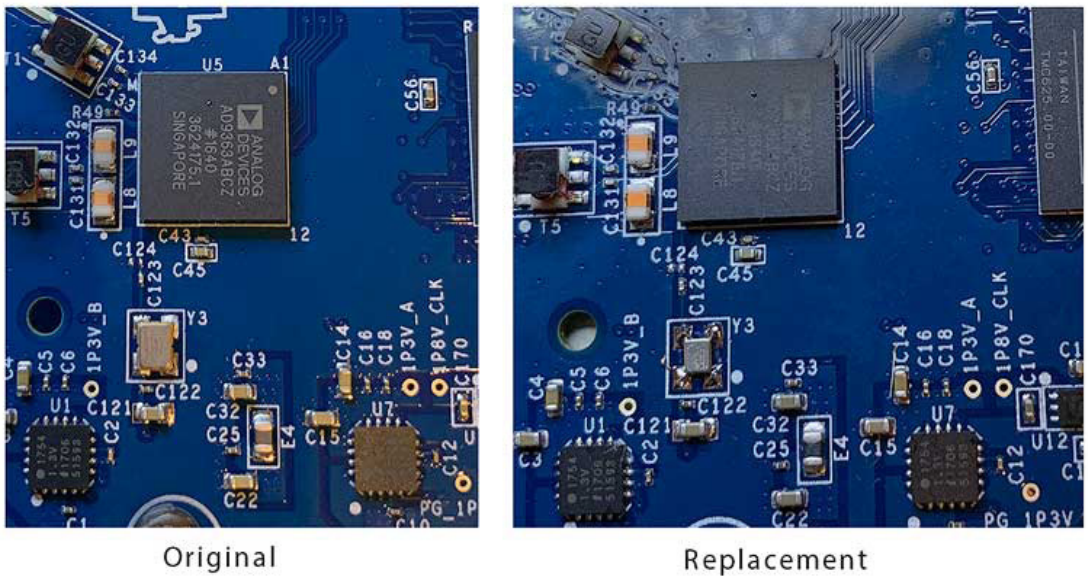


Figure 1.

Another option would be to remove the crystal and feed in an external LO reference. The follow wiki link discusses how to change the LO of your device if you have the ad9361 chip on the board.

https://wiki.analog.com/resources/tools-software/linux-drivers/iio-transceiver/ad9361#fast-lock_mode

GNU Radio is using the Libiio drivers to write values into the registers of the communication chip when the OOT module is present. So I assume you could look at all the registers and make changes to others values you want to modify.

Keep in mind the follow on PLL also plays a role in frequency stability. From some discussion I

heard at a recent workshop Analog Devices designers made an effort to mitigate the PLL issues.

This work would be best performed using a Linux interface for the Libiio support until the Windows platform is available to the ADLM-Pluto. There is a windows platform supporting the Libiio today but the interface is not through GNU Radio. This will likely change soon.

Speed Test

When running GNU radio and trying to use it as radio to communicate you may be pressed to find your best option on a computer to use. This information is intended to help you understand the difference in performance of a few available platforms.

I installed a package called sysbench on several computers. This package is an attempt at bench testing computers to see how fast they can complete a task. The version I used, sysbench -version:

```
sysbench 1.0.11 (using system LuaJIT 2.1.0-beta3)
```

The command line entry for the program I used was:

```
sysbench --test=cpu --num-threads=4 --cpu-max-prime=9999 run
```

In this case sysbench would execute with the following flags set:

--test=cpu Test the CPU in the computer

--num=threads=4 Run four different calculations at one time.

--cpu-max-prime=9999 Calculate the number of prim numbers found in 9999

run Execute the request and time the results

There are many other options available and some things are set at default. This command with the options seltected gives us some useful information.

The test results in Table 1 show us the time it takes each computer to complete the calculations.

Raspberry Pi	Model B+ V1.2	388	Seconds
Raspberry Pi	Rpi Zero W	42	Seconds
Raspberry Pi	Pi 4	29	Seconds
NVDIA	Jetson 2	10	Seconds
Intel	I7	10	Seconds

Table 1

I was surprised to see the Jetson 2 and the Intel I7 performed with the same results.

To identify the two fastest computers better I ran the command
With the following results:

sudo lshw –short
Intel:

```
Class       Description
=========================================================
system      System Product Name (SKU)
bus         PRIME Z370-A
memory      64KiB BIOS
memory      16GiB System Memory
memory      [empty]
memory      8GiB DIMM DDR4 Synchronous Unbuffered (Unregistered) 2400 MHz (0.4 ns)
memory      [empty]
memory      8GiB DIMM DDR4 Synchronous Unbuffered (Unregistered) 2400 MHz (0.4 ns)
memory      384KiB L1 cache
memory      1536KiB L2 cache
memory      12MiB L3 cache
processor   Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
bridge      8th Gen Core Processor Host Bridge/DRAM Registers
bridge      Xeon E3-1200 v5/E3-1500 v5/6th Gen Core Processor PCIe Controller (x16)
bridge      Xeon E3-1200 v5/E3-1500 v5/6th Gen Core Processor PCIe Controller (x8)
storage     88SE9230 PCIe SATA 6Gb/s Controller
display     Intel Corporation
bus         200 Series/Z370 Chipset Family USB 3.0 xHCI Controller
bus         xHCI Host Controller
```

sudo lshw
NVIDIA:

```
jetson-2
    description: Computer
    product: jetson-nano
    serial: 04211190063720c00501
    width: 64 bits
    capabilities: smp cp15_barrier setend swp
  *-core
       description: Motherboard
       physical id: 0
     *-cpu:0
          description: CPU
          product: cpu
          physical id: 0
          bus info: cpu@0
          size: 921MHz
          capacity: 1428MHz
          capabilities: fp asimd evtstrm aes pmull sha1 sha2 crc32 cpu-
freq
     *-cpu:1
          description: CPU
          product: cpu
          physical id: 1
          bus info: cpu@1
          size: 921MHz
          capacity: 1428MHz
          capabilities: fp asimd evtstrm aes pmull sha1 sha2 crc32 cpu-
freq
     *-cpu:2 DISABLED
          description: CPU
          product: cpu
          physical id: 3
          bus info: cpu@2
          size: 921MHz
          capacity: 1428MHz
          capabilities: cpufreq
     *-cpu:3 DISABLED
          description: CPU
          product: cpu
          physical id: 4
          bus info: cpu@3
          size: 921MHz
          capacity: 1428MHz
```

```
          capabilities: cpufreq
     *-cpu:4 DISABLED
          description: CPU
          product: idle-states
          physical id: 5
          bus info: cpu@4
     *-cpu:5 DISABLED
          description: CPU
          product: l2-cache
          physical id: 6
          bus info: cpu@5
     *-memory
          description: System memory
          physical id: 7
          size: 3956MiB
     *-pci
          description: PCI bridge
          product: NVIDIA Corporation
          vendor: NVIDIA Corporation
          physical id: 2
          bus info: pci@0000:00:02.0
          version: a1
          width: 32 bits
          clock: 33MHz
          capabilities: pci pm msi ht pciexpress normal_decode bus_master
cap_list
          configuration: driver=pcieport
          resources: irq:84 ioport:1000(size=4096) memo-
ry:13000000-130fffff
```

I plan to use the Jetson for my GNU Radio support, as it is small and fast. I purchased a fan to try and keep the CPU cooler. If the processor starts to warm up the clock rate will slow down without any indication to you and performance will degrade. This is a problem if you are a rover operating on a hot day. Some type of special cooling is needed or you need to operate inside an air-conditioned vehicle.

A Few Comments on FPGA operation vs. CPU code….

When working with an SDR radio many of them are Field Programmable Gate Array, FPGA based or have the option of using an FPGA. The FPGA is an IC, which is a larger collection of transistors that can be interconnected with a special connection map. The interconnect map is stored in a memory chip and loaded into the FPGA at power up typically but can be loaded or changed while the FPGA is in use.

This map to do the FPGA interconnect is called combinational logic map. An example of the some equivalent TTL logic of a 4 bit multiplier is shown in Figure 1. This multiplier would have the results of A X B in a few Nano seconds after either value changes states. Very fast and little effort after the FPGA is mapped and operational. The most difficult part of FPGA development can be establishing proper clocking across the chip. You must have all the logic switching properly or you may have logic levels in the wrong state when they are considered. There are tools to help with this but developers can spend days making sure they have good timing and still may have subtle issues that bite you later. This depends on the skill of the developer, the chips used, and the complexity of the design.

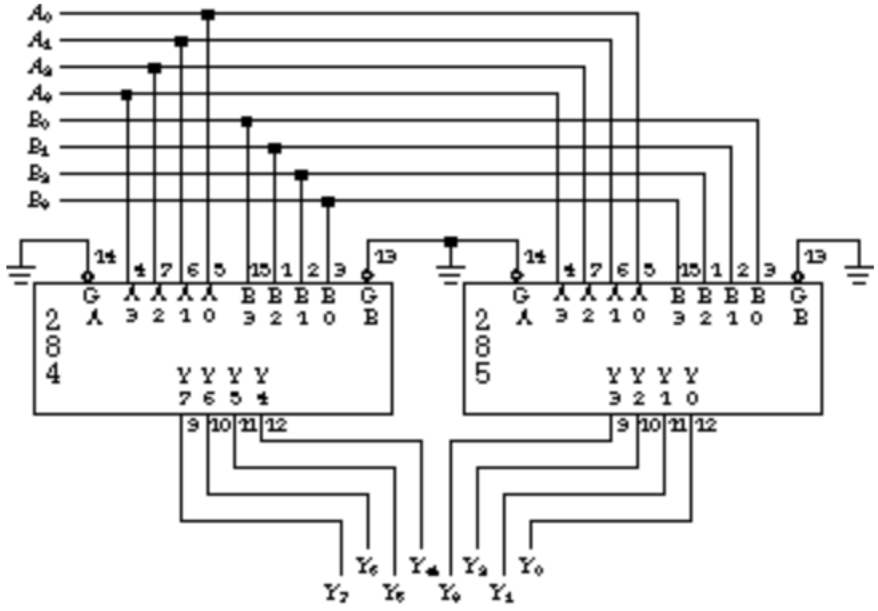


Figure 1

To do the same multiplication job in a processor you would write a hi level program in C , C++, or python which says Result = A * B. This would be assembled and compiled into a list of binary instructions for the CPU chip and would require many clock cycles to execute. The CPU can be an ARM processor like a Raspberry Pi or a high-end computer motherboard with a

water-cooled CPU. The time to complete the multiplication will depend on the clock frequency and the instruction set available.

If the needed DSP processing is not too complex the CPU approach will work. I remember Phil Karn, KA9Q pushing the idea of using an IBM desktop computer over a DSP based system 30 years ago.

The problem is implementation in both cases. In the case of the FPGA you need to know how to develop the mapping program using a language called Verilog typically and have the tools to do this. The tools are readily available almost free in many cases. A detailed understanding of the architecture of the FPGA selected and the tool set needed. Not a lot of people much less amateur radio people are available to do this.

In the case of the CPU there are more people with the know how to program and the tools are available for free. We are seeing more effort in this area but the progress is slow.

GNU Radio offers the amateur that is not a strong programmer an opportunity to build the needed firmware for radio projects using easy to understand flow charts. Learning the details of signal process is still a challenge but many already have some experience using analog processes.

GNU Radio use of Out Of Tree (OOT) modules allows a manufacture or a end user to create a OOT block that may call on routines buried in an FPGA. In some cases this is documented well enough that an armature can take advantage of writing there own OOT block to use an FPGA or other hardware available in a SDR platform. An example of this may be to obtain a single IO to do Radio Transmit/Receive switching.

You can usually find good documentation at the PCB level but when the signals go down into an IC this may be more difficult. The needed details do sometimes leak out of the manufacture over time. An example of this is the information needed to allow you to change the reference clocking frequency on the ADALM-Pluto by Analog Devices.

Some FPGA models are available in the public domain like the code used in the TAPR HPSDR and the Hermes Light for HF SDRs.

Hopefully this explains the difference in FPGA vs. CPU approaches to DSP.

Additional References

1. https://www.ettus.com/sdr-software/gnu-radio/ The GNU Radio web page with software, hardware and news about events.
2. https://greatscottgadgets.com/sdr/, Lessons on DSP and the Hack RF presented by Michael Ossmann of Great Scott Gadgets. A very nice easy to understand presentation especially lessons 6 and 7.
3. https://labsdl.wordpress.com/2018/08/30/gnu-radio-tips-how-to-convert-wav-iq-file-into-raw/ This is an utility use of a Flow Graph to convert a WAV file to a RAW file.
4. https://www.eevblog.com/forum/rf-microwave/adalm-pluto-as-vh-fuhf-spectrum-analyzer-and-a-tracking-generator/ A discussion on using a SDR (Pluto) as a tracking signal generator including flow charts.
5. http://aaronscher.com/GNU_Radio_Companion_Collection/GNU_Radio_Companion.html A collection of GNU Radio Flow Charts with documentation presented by Dr. Aaron Scher of Oregon Institute of Technology
6. https://sv1cal.com/usrp_map65/ Michael Margaras SV1CAL documents his use of the Ettus USRP for EME operations and other radio work.
7. https://stackoverflow.com/questions/33435642/programming-an-oot-module-with-variable-i-o-type Tip on writing OOT blocks.
8. https://github.com/gnuradio/gr-tutorial A tutorial on OOT module for GNU Radio
9. https://wiki.analog.com/resources/tools-software/linux-software/gnuradio Analog Devices Wiki on GU Radio
10. http://oz9aec.net/radios/gnu-radio/grc-examples Alexandru Csete OZ9AEC Example page