# SDR with GNU Radio

Ray Rischpater, KF6GPE
kf6gpe@arrl.net | kf6gpe.org

BayCon 2020, 8 February 2020

# What we're talking about today

- What's GNU Radio?
- A very basic introduction to SDR
- A word on hardware
- Getting GNU Radio
- GNU Radio Companion
- Using Python or C++ to write a module

# What’s GNU Radio?

- Free & open source software
- Provides signal-processing blocks to build software-defined radios
- Works with readily available hardware
- Runs on Linux (best), Windows and macOS
- Runs on both Intel & ARM (including Raspberry Pi!)
- Used in hobby and commercial applications

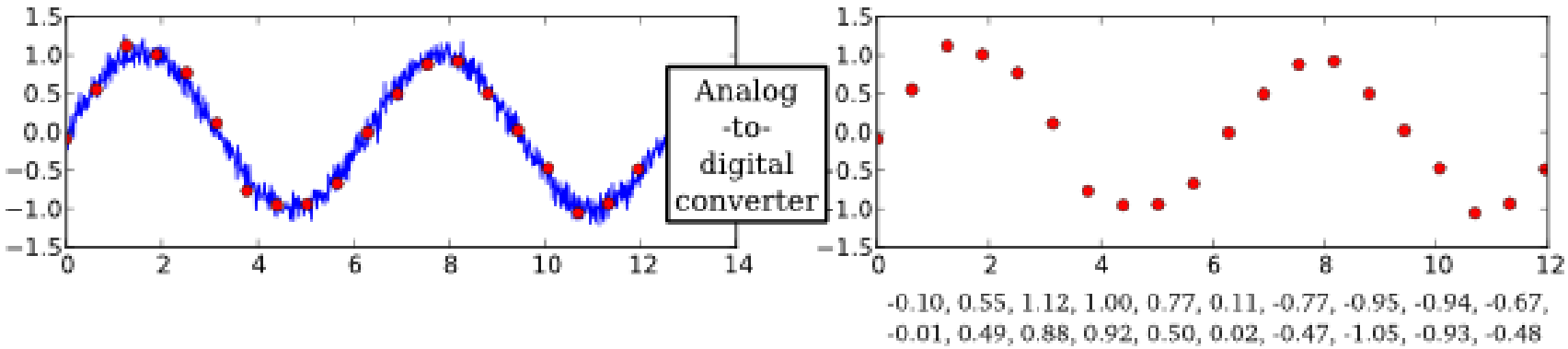
# A very basic introduction to SDR

- Uses digital representations of signals
- Uses lots of math to implement things like modulation & filters

# Sampling a signal

- Analog signals sampled periodically
- Sample rates should be at least twice maximum frequency (Nyquist theorem)

# Sampling a signal




source: Wikipedia; https://wiki.gnuradio.org/index.php/File:Cont_to_digital.png

# Sampling a signal - things to look out for

- Tradeoff between sample rate & computing requirements
- Tradeoff between cost, sample rate, and sample resolution
- Aliasing

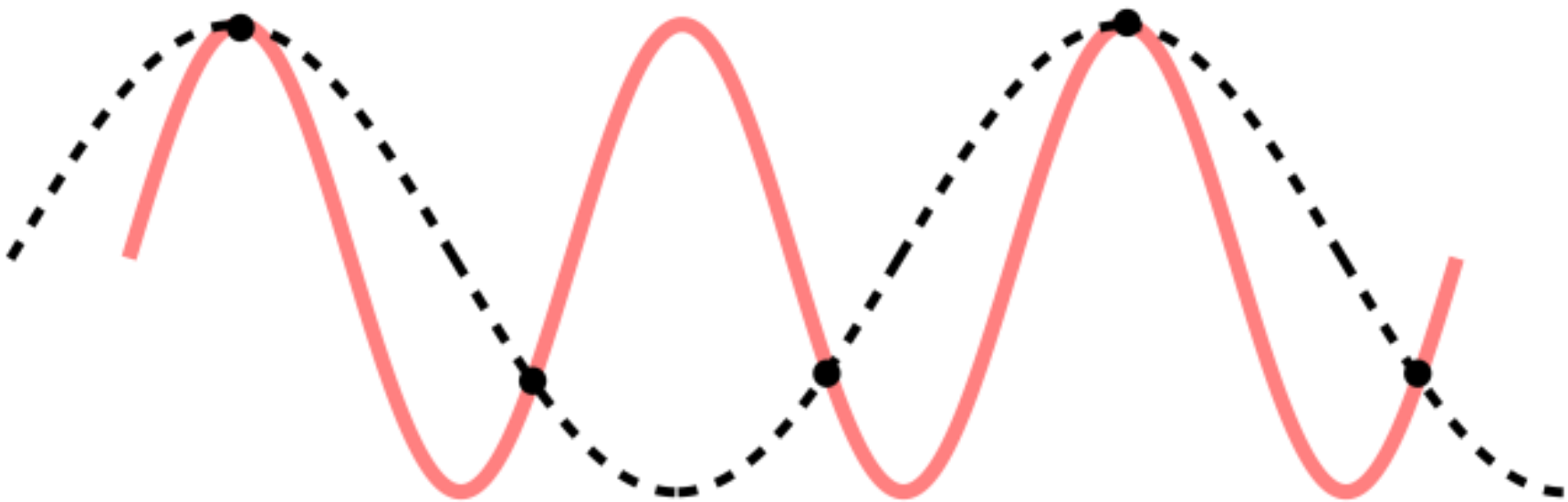
# A word on sampling

- More samples mean more expensive A/D converters
- More samples means more computing power to process the samples
- Faster sampling means higher cost

# A word on resolution

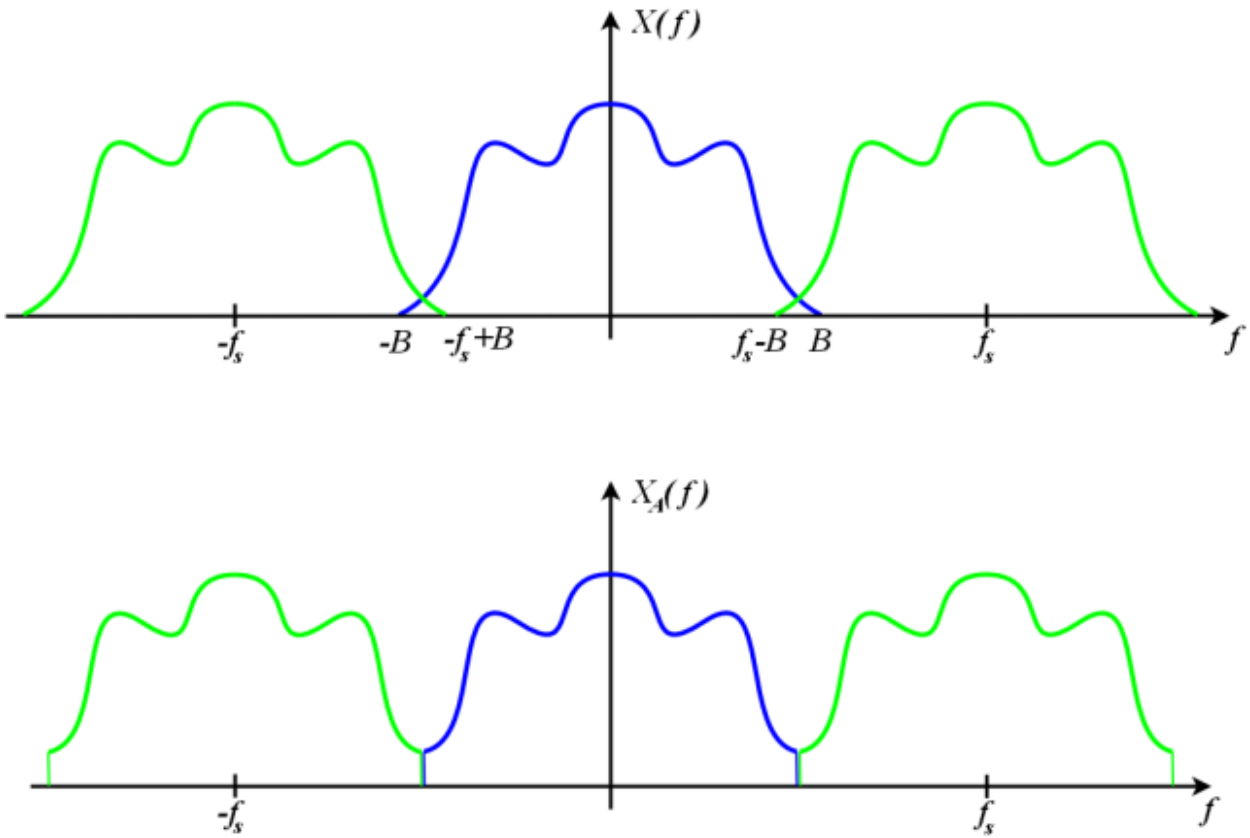
- The number of discrete steps measurable in the range of a signal
- Higher resolution generally means higher cost A/D converters
- Sometimes a lower noise lower-resolution A/D is better than a higher resolution A/D!

# Aliasing in sampling



source: Wikipedia; https://en.wikipedia.org/wiki/Nyquist–Shannon_sampling_theorem#/media/File:CPT-sound-nyquist-thereom-1.5percycle.svg

# Aliasing in the frequency domain




source: Wikpedia; https://en.wikipedia.org/wiki/Nyquist–Shannon_sampling_theorem

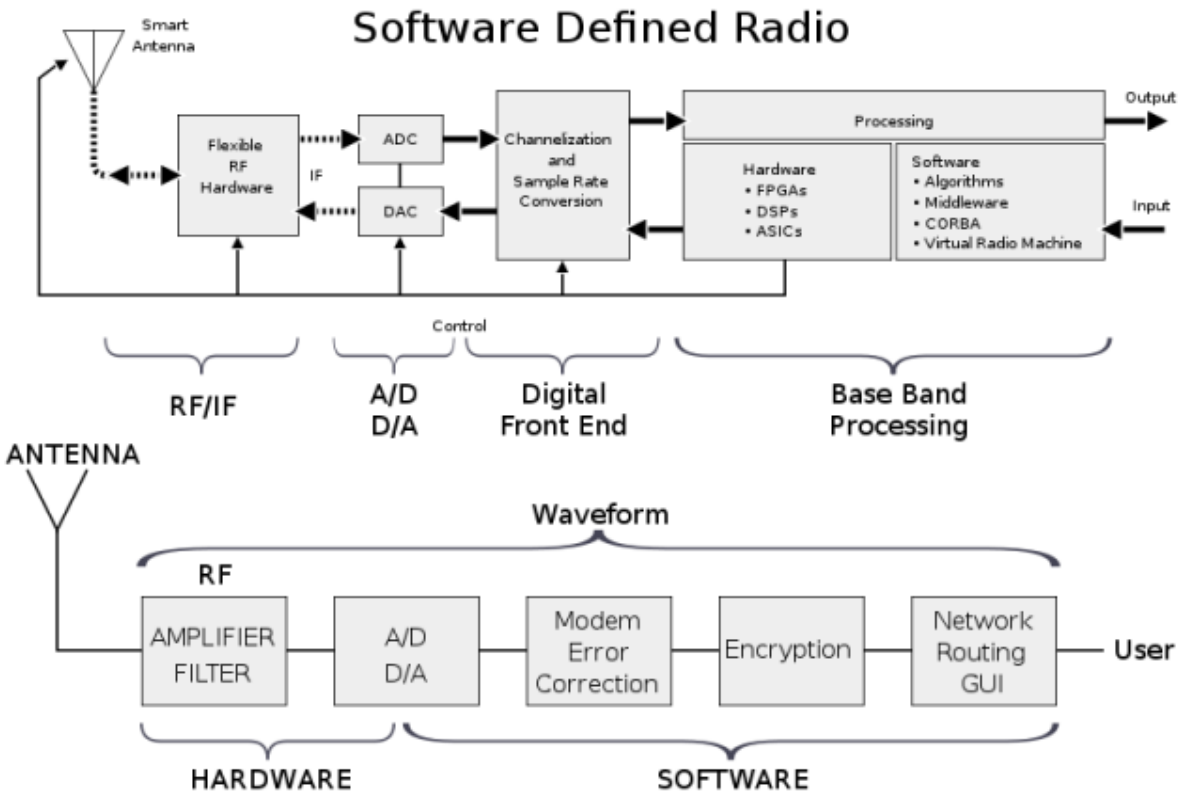
# Digital Filtering

- Math!
- Uses signal processing on the digitally sampled signal
- Requires fast processsing for additions & multiplications
- Digital filters can often create far steeper cutoffs for the same component cost

# Modulation & demodulation

- More math!
- AM can be accomplished multiplying the carrier by the audio signal
- BPSK can be accomplished by changing the phase of the carrier
- Other modulations possible, too!

# Block diagram of a typical SDR




source: Wikipedia; https://commons.wikimedia.org/wiki/File:SDR_et_WF.svg

# A word on hardware

- You need a moderately fast PC or Raspberry Pi running Linux, Windows, or macOS
- For RF experiments, you need some sort of SDR hardware
- RTL-SDR is a cheap receiver (check the raffle table!)
- hackRF is a popular transceiver
- LimeSDR is another newer transceiver
- Lots of other choices; just check for GNU Radio compatibility first

# What does GNU Radio give me?

- Over 400 DSP blocks, including filters, modulators, demodulators, visualization tools
- Flow-based construction of radios & subsystems
- The ability to add new blocks with code in Python or C++

# Getting GNU Radio

- On Linux, use your package manager
- On Windows, use an installer
- On macOS, use MacPorts
- You can also build from the source code
- See the GNU Radio Wiki for more details

# GNU Radio Companion

- Drag and drop construction of SDRs using modules
- Uses a data-flow paradigm for connecting modules
- Provides controls and visualization tools

# Modeling SDRs using data flow diagrams

- Radio components connected by signal paths
- Each signal is a stream of numbers
- Modules transform the number streams in some way
- Sources provide data, sinks acccept it

# Modules and variables

- Most elements of your radio are functional blocks that do something
- Variables are named containers that hold numbers (frequencies, sample rates, etc)
- Modules get connections, variables just sit there.

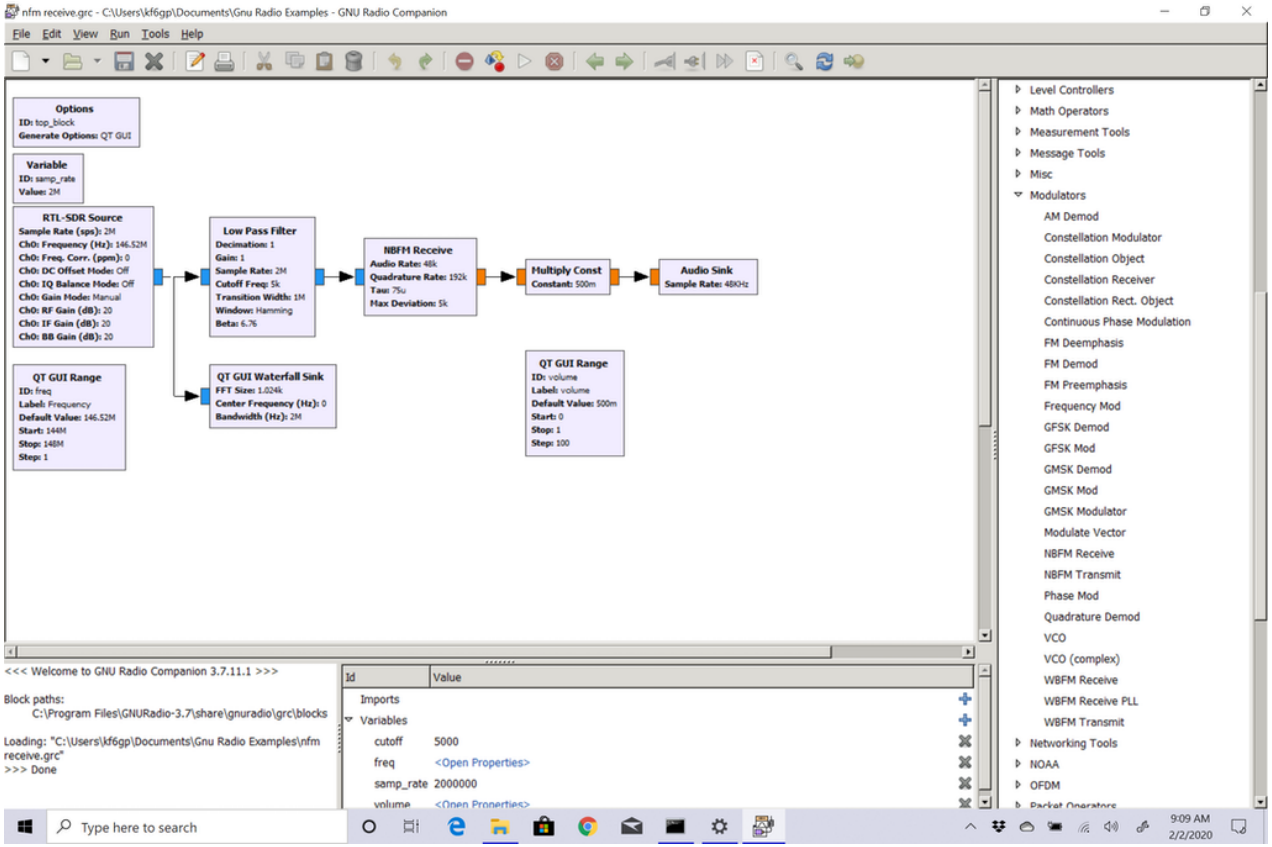
# Graphical User Interface components

- Some sinks display data (e.g., waterfall)
- Some GUI controls to set variables (e.g., knobs)

# Basic use

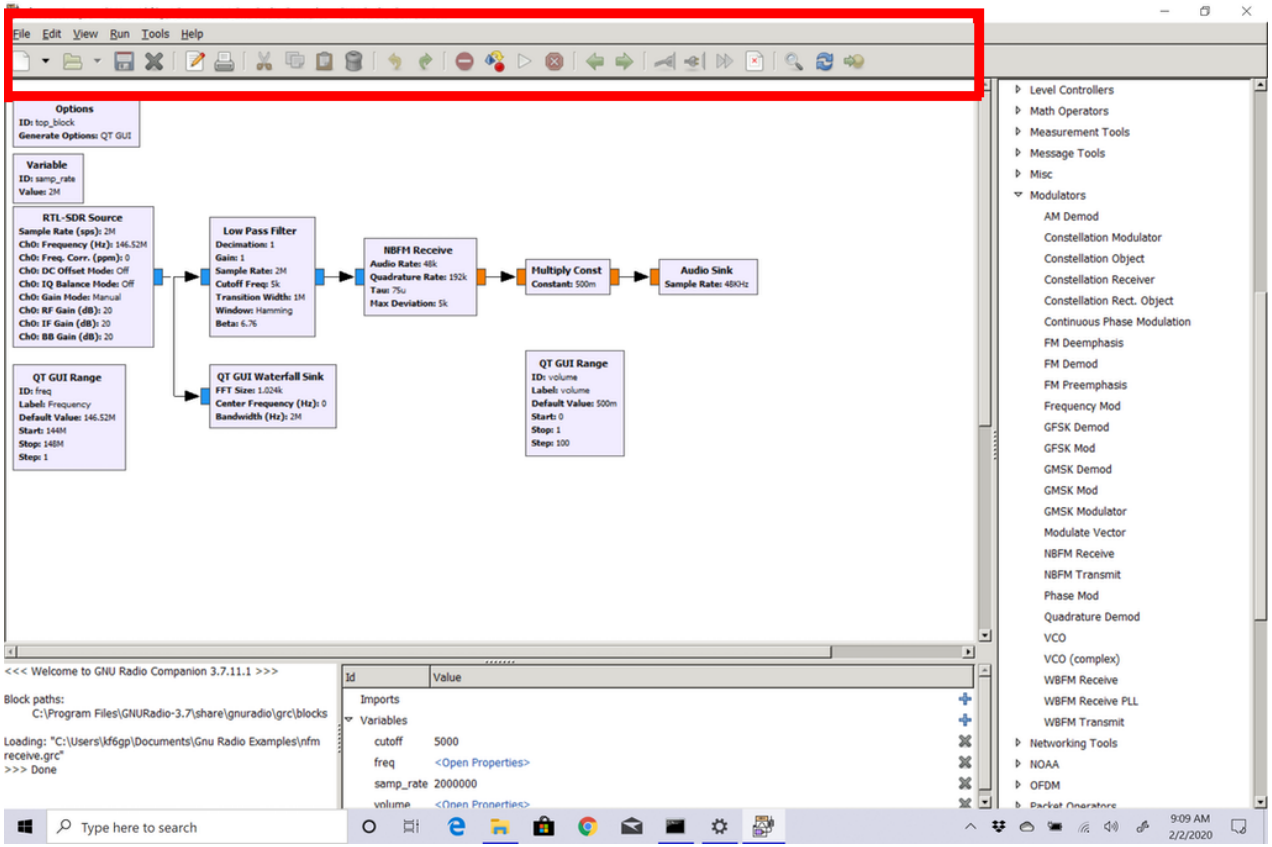
- Drag and drop modules
- Click to create data flow connections
- Control-F is your friend (finds modules)

# GNU Radio Companion UI



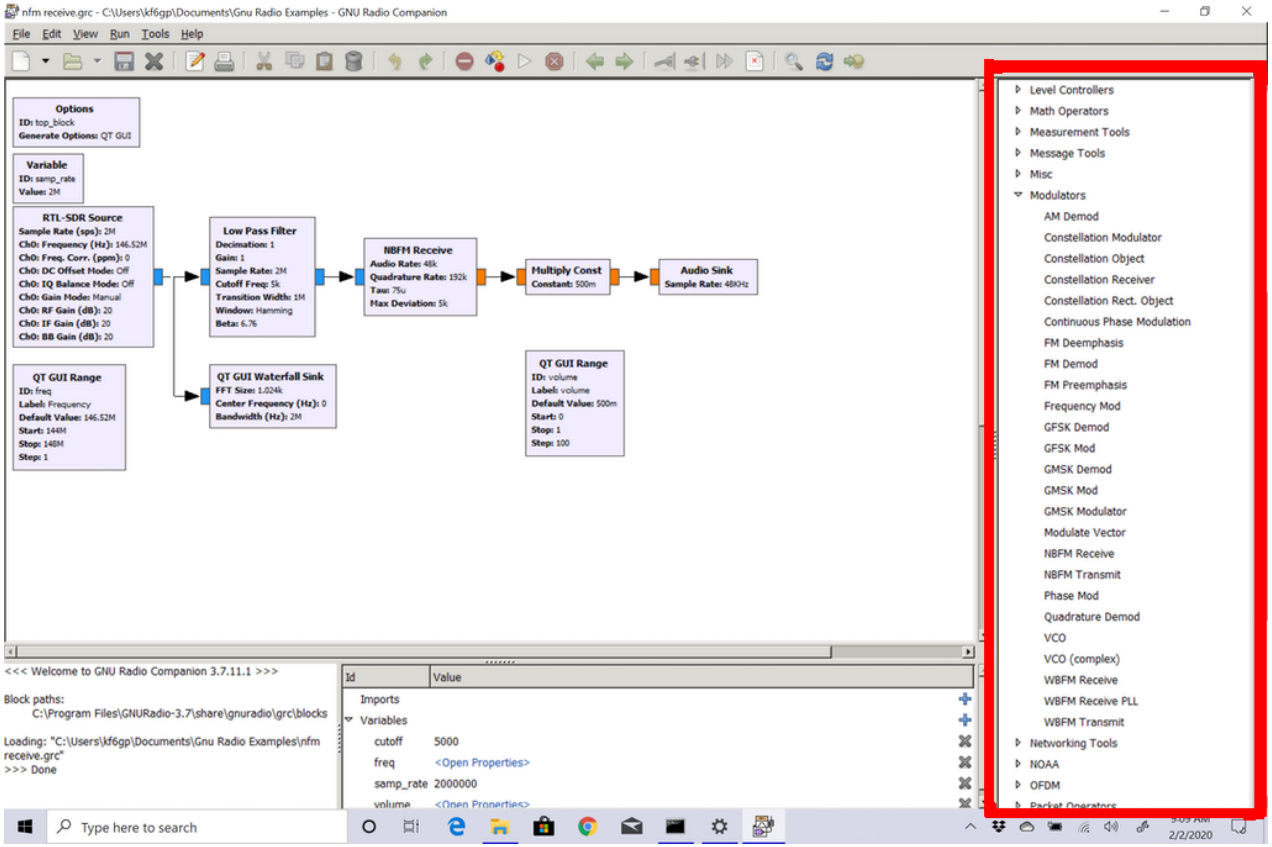

source: by the author

# Toolbar



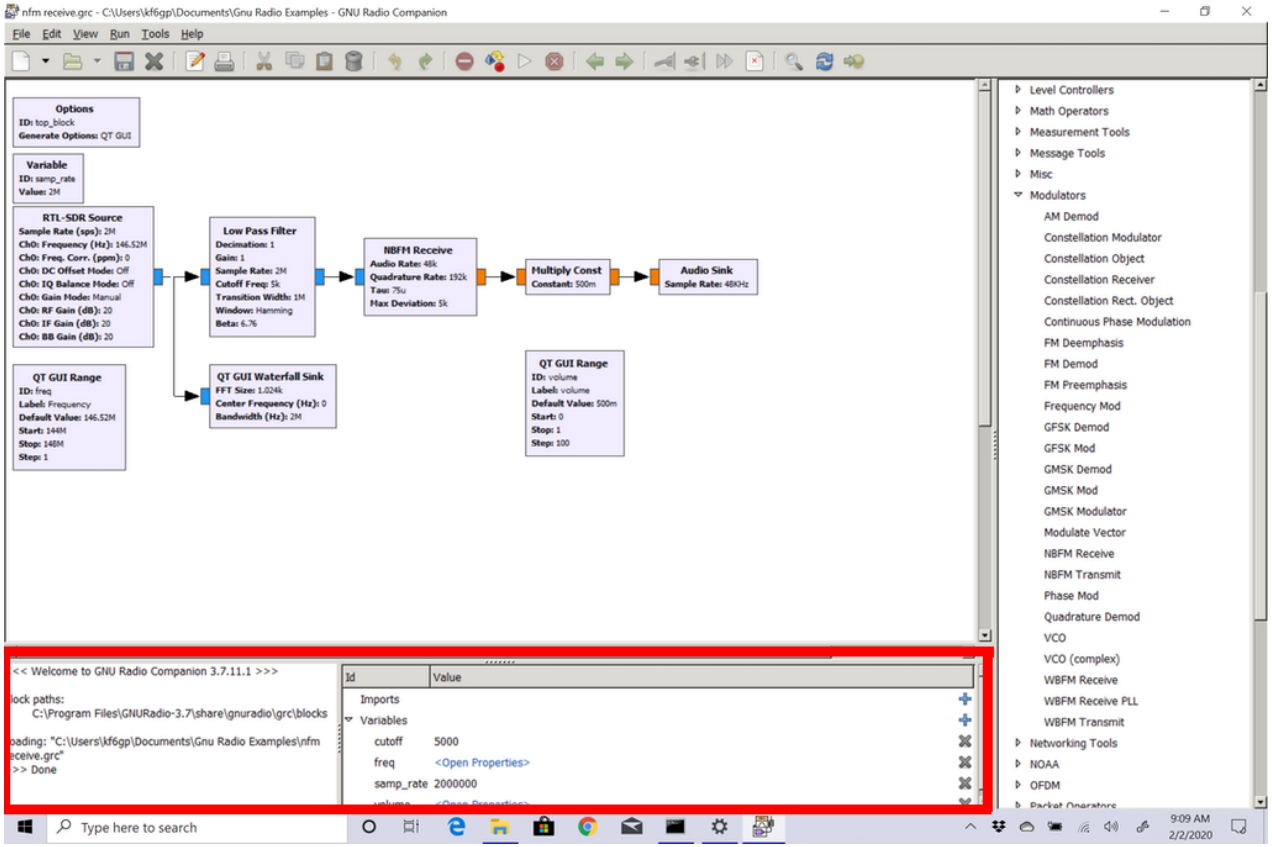

source: by the author

# Modules



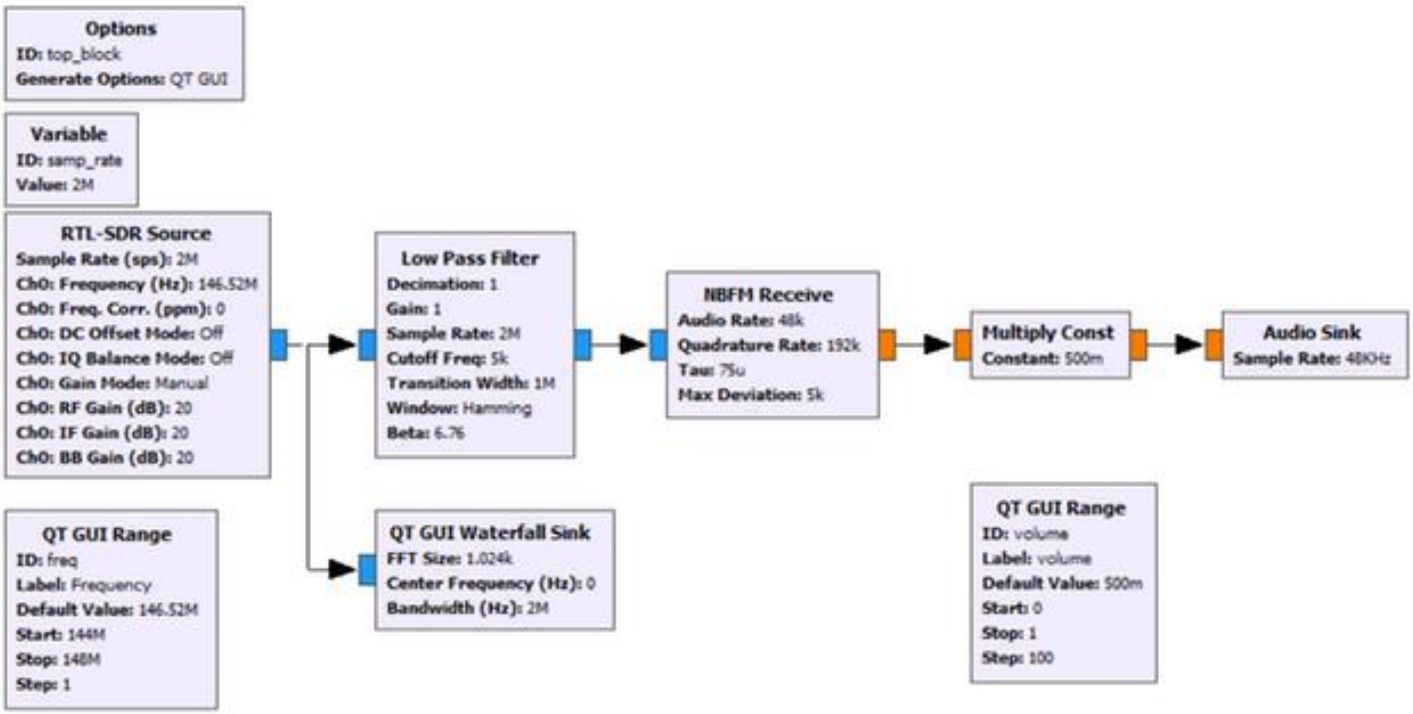

source: by the author

# Console & variable inspector




source: by the author

# A simple NBFM receiver flow graph

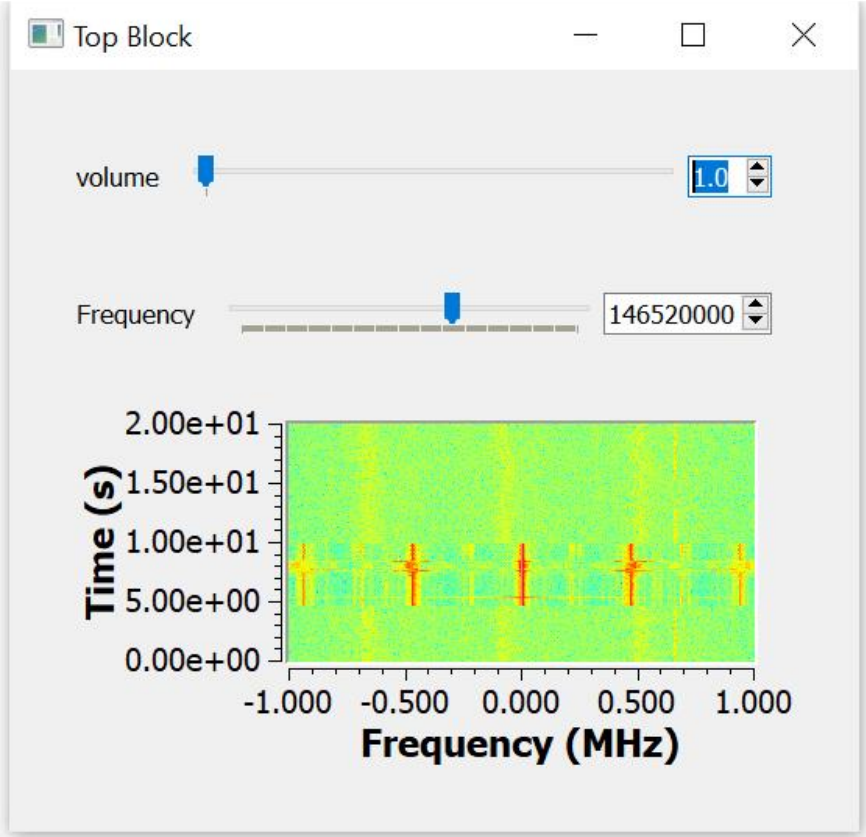



source: by the author

# Running the flow graph

- First GRC creates a Python file of your flow graph
- The Python file uses code from GNU Radio
- Then GRC invokes GNU Radio with your Python file

# A simple NBFM receiver UI




source: by the author

# Creating modules & radios

- Uses Python or C++.
- Define modules using XML
- Implement them using Python or C++
- Uses the same data-flow paradigm, just in code.

# Modules have

- An initialization block
- A message handler
- A work block that does the math

# Connecting modules in code

- Create the flowgraph in Python or C++
- Connect modules using the `connect` method

# Defining the module

- Modules require a definition
- Definitions are in YAML or XML
- This enables GNU Radio to “see” a module

# Tips

- Work through the tutorial
- Look at error messages closely
- Windows and macOS ports are flakier than Linux
- Get your SDR working with the software that comes with it first.

# Resources - Hardware

- RTL-SDR receiver, frequencies vary
- hackRF transceiver, 1 MHz-6 GHz
- LimeSDR transceiver, 100 kHz-3.8 GHz

# Resources - Software

- GNU Radio
- GNU Radio Live SDR Environment

# Resources - Reading

- Recent editions of the *ARRL Handbook*
- "Digital Signal Processing and GNU Radio Companion", *QEX* July/August 2014
- GNU Radio Wiki