Software Defined Radio and Digital Signal Processing

Cardiff University - November 7, 2018

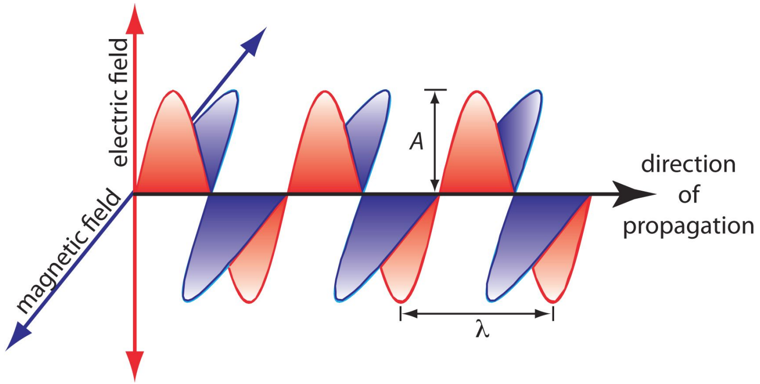
Derek Kozel

- Radio Amateur since second year of university
  - UK Advanced license MW0LNA, US Extra K0ZEL
- Moved from the San Francisco Bay Area to Cardiff in April 2017
- Bachelors and Masters in ECE & Public Policy at Carnegie Mellon University
- Worked at Range Networks, SpaceX, Ettus Research (NI)
- Currently a PhD at Cardiff University in the Centre for High Frequency Engineering
- GNU Radio Project Officer

Intro to Radio

Electromagnetic Waves

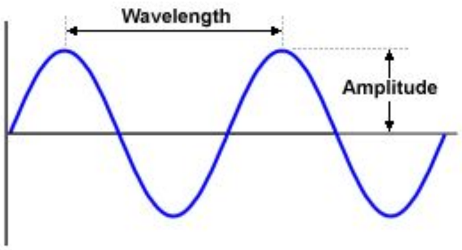
- Electric and Magnetic energy
- Can bounce, bend, and generally confuse

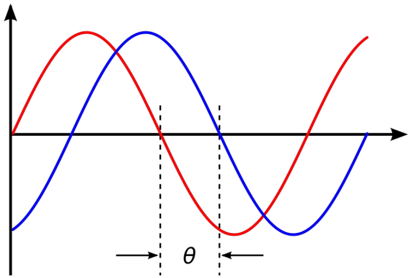


https://chem.libretexts.org/Textbook_Maps/Analytical_Chemistry_Textbook_Maps/Map%3A_Analytical_Chemistry_2.0_(Harvey)/10_Spectroscopic_Methods/10.1%3A_Overview_of_Spectroscopy

Properties of a wave

- Frequency
  - Number of cycles per second (Hertz)
- Wavelength
  - Distance between start and end of a cycle
- Amplitude
  - The magnitude or strength of the wave
- Phase
  - The offset of the wave with respect to another wave





https://en.wikipedia.org/wiki/Wavelength

https://en.wikipedia.org/wiki/File:Sine_voltage.svg

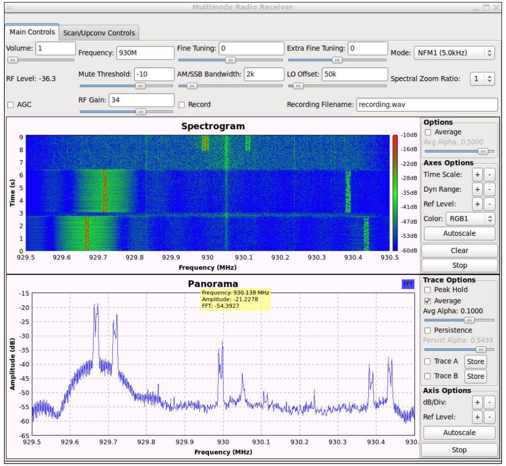
https://en.wikipedia.org/wiki/Phase_(waves)

Intro to GNU Radio

GNU Radio

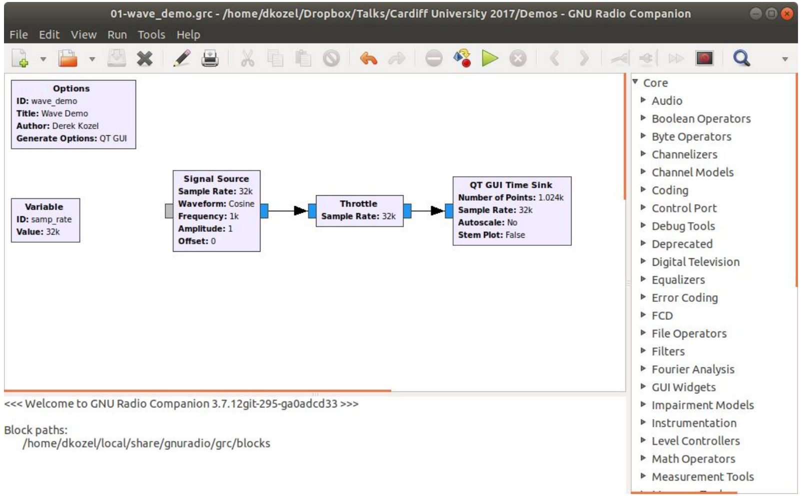
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

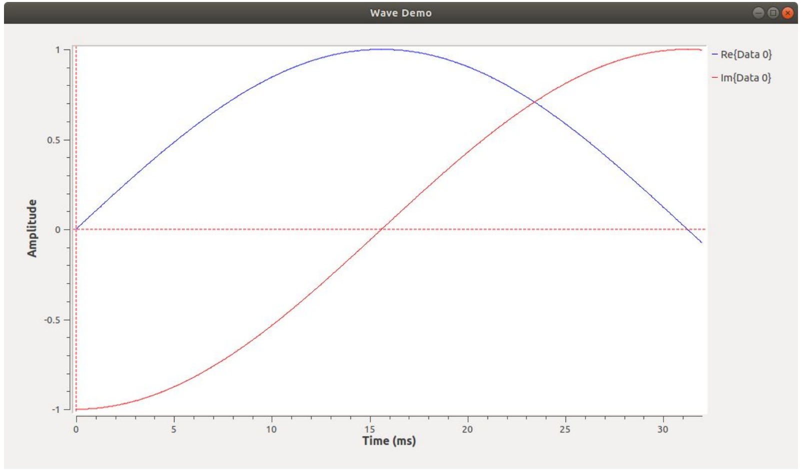
- A framework and set of libraries to build and run digital signal processing applications, primarily software defined radio ones
- Started in 2001
- Libre and open source
- Written in C++ and Python primarily
- Available on Linux, Windows, and Mac
- Used by a very wide variety of users
    - commerical, hobbyist, government



http://superkuh.com/rtlsdr.html

GNU Radio Companion



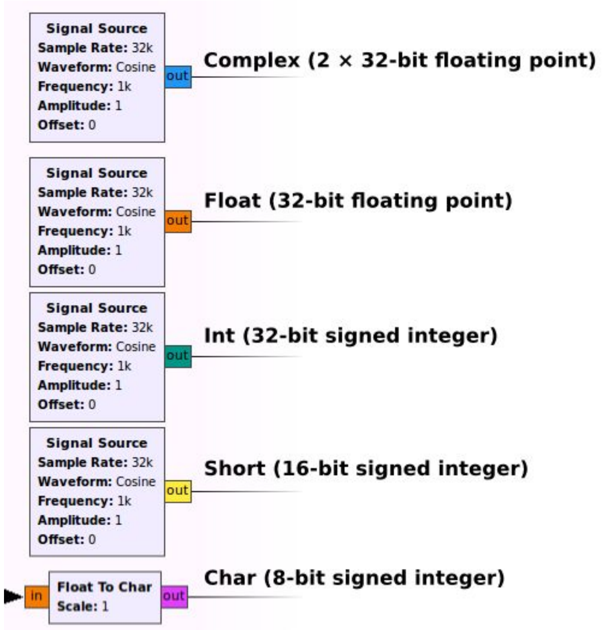


Automatic Code Generation

- The graphical UI is generating Python code
  - Or C++ in the latest version
- We’ll look quickly under the hood later

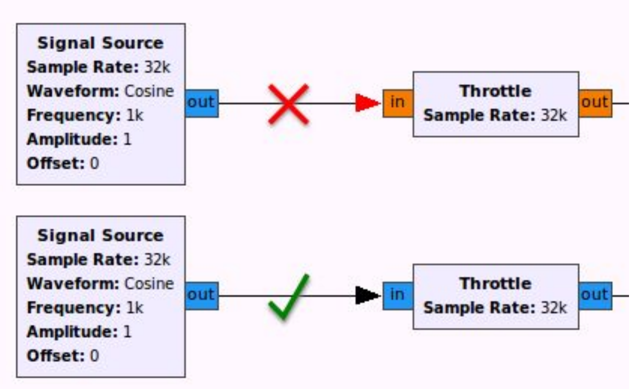
Data Types

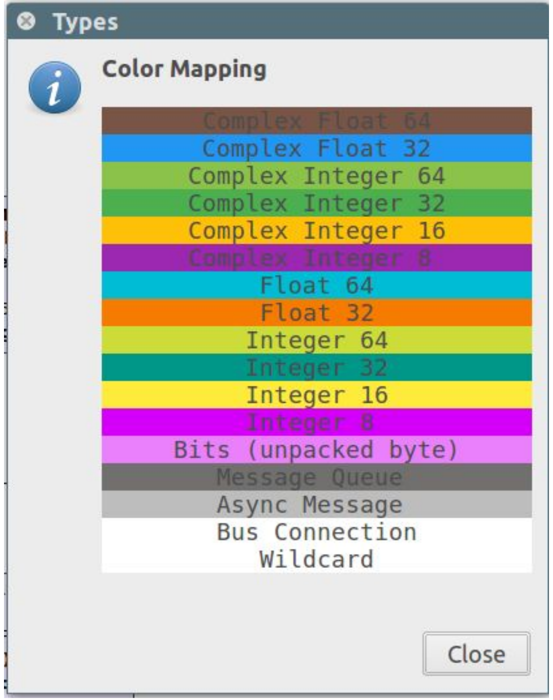
- Samples and data comes in different digital formats
- Semantic differences
  - Complex vs Real samples
  - Number vs Letters
- Size differences
  - 8 bits vs 32 bits



Data Types

- Have to connect matching types
- GRC will warn you if there's a mismatch
- In the end, bits are bits
    - Computer will interpret them as you tell it to

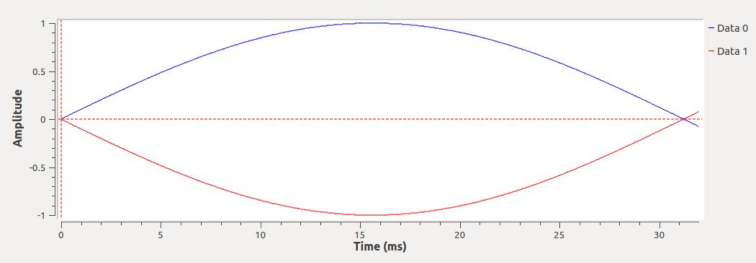




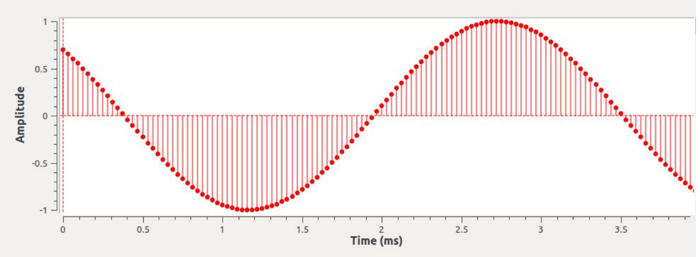
Digital Signal Processing

Time Domain

- Amplitude values over time
- Signals are continuous in the air or wire
- Signals are digitized by sampling the current value many times



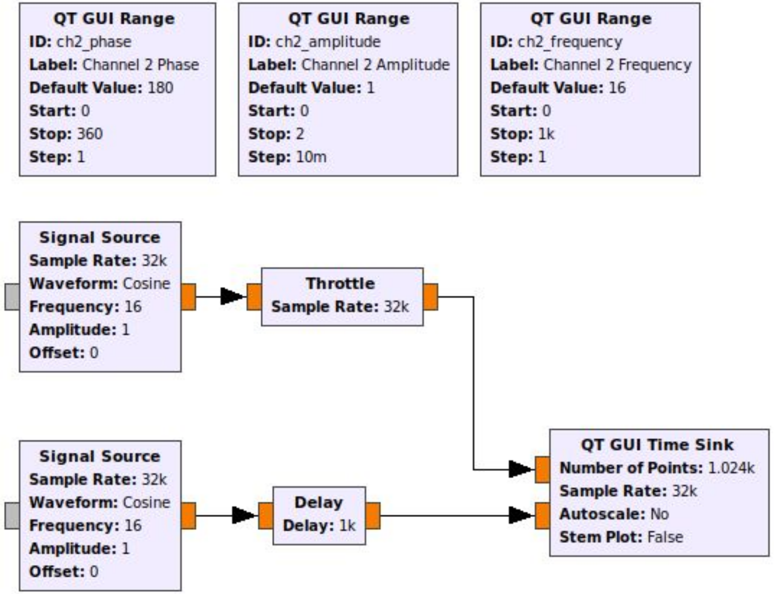
Continuous Signal (it’s a lie!)



Discrete (sampled) Signal

Exploring Waves

- Setup a flowgraph with controls for
  - Phase
  - Frequency
  - Amplitude



Delay value:

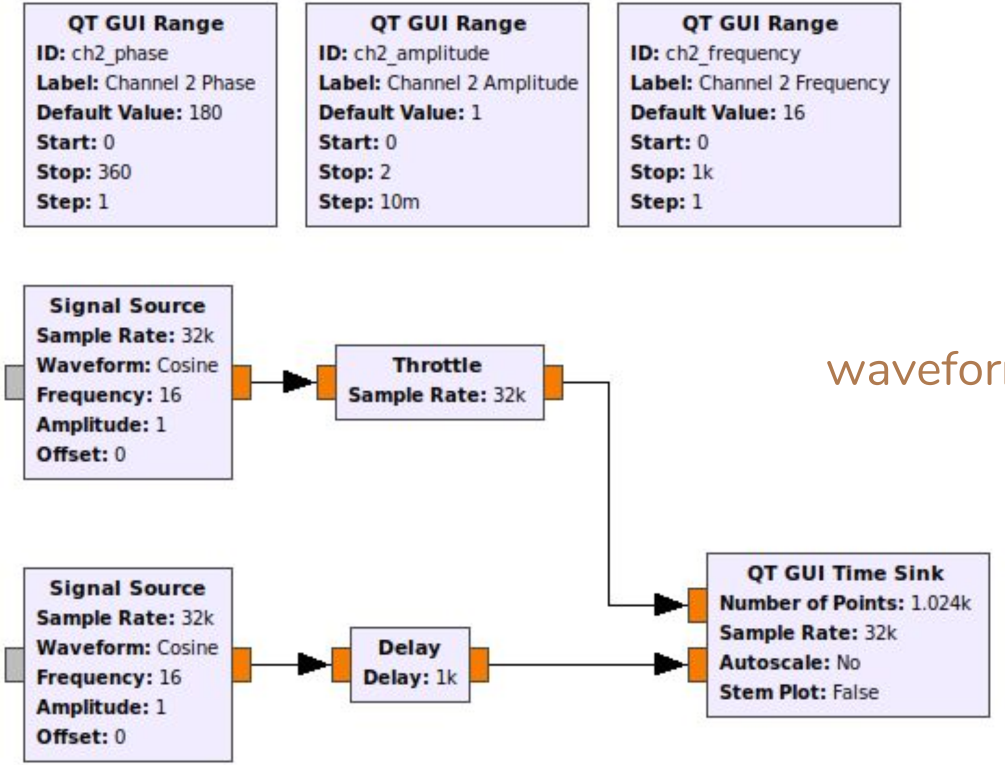
```
int((samp_rate/ch2_frequency) * (ch2_phase/360.0))
```

Throttle Block

- GNU Radio will process data as fast as possible
- Hardware (Analog to Digital or Digital to Analog converters) will have a set sample rate
- Simulation only doesn’t
- Add one (and only one!) throttle block to the flowgraph
    - Has a timer inside that tries to match the average throughput to the sample rate

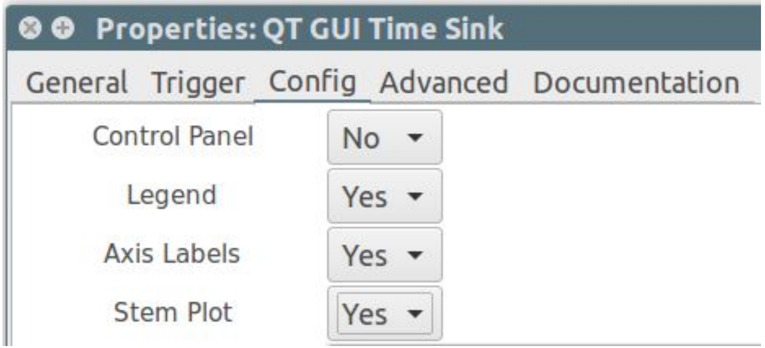
Delay value:

```
int((samp_rate/ch2_frequency) * (ch2_phase/360.0))
```

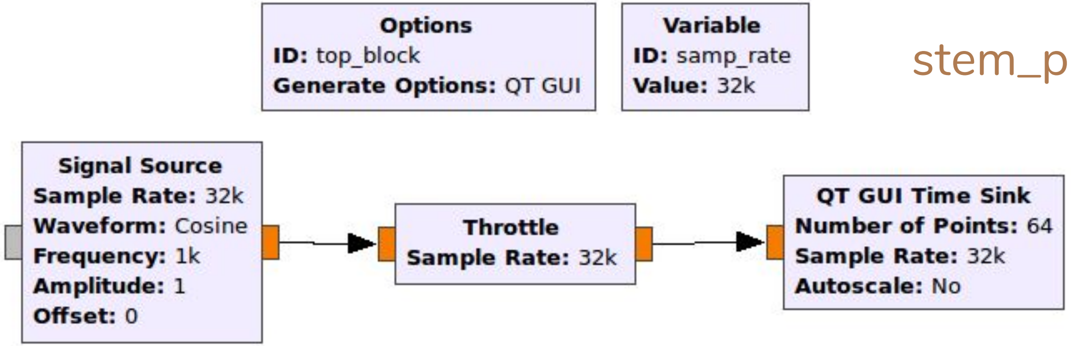


waveform_demo.grc

Discrete Sampling



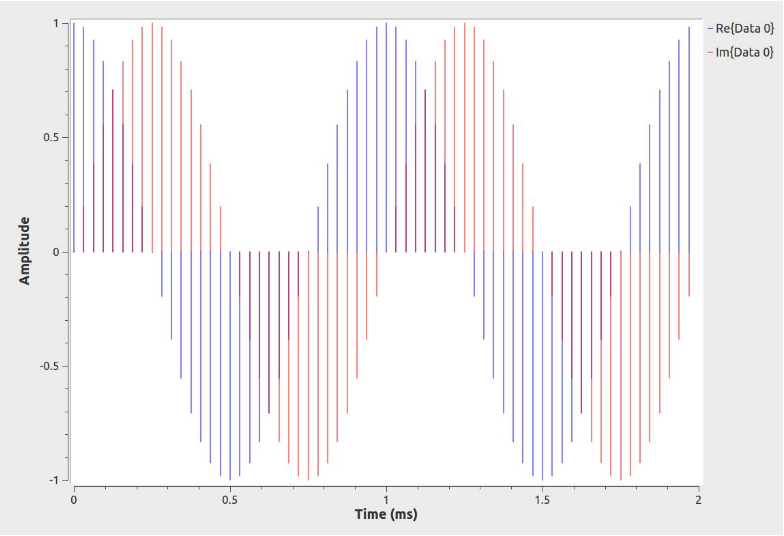
- Usually data is displayed as if it were continuous
  - Easier to visualize
  - Mostly accurate as long as you follow Nyquist's Sampling Law
- Can also display actual data points
  - Select Stem Plot under the Config menu in QT GUI Time Sink



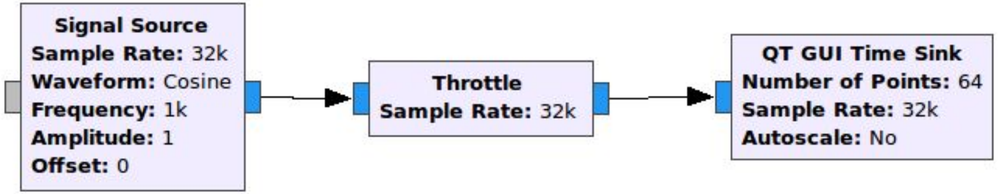
stem_plot.grc

Complex Sampling

- Hard to make fast ADC/DACs
- Also ambiguities in frequency are real(ly painful)
- When mixing a signal with a sinewave crossing zero you lose all the information!
- Solution: Split the signal in two, mix with a sine and cosine, sample each result at the same time
  - Twice the information, all of it useful
  - Not cheating Nyquist
  - Bandwidth = Sample Rate

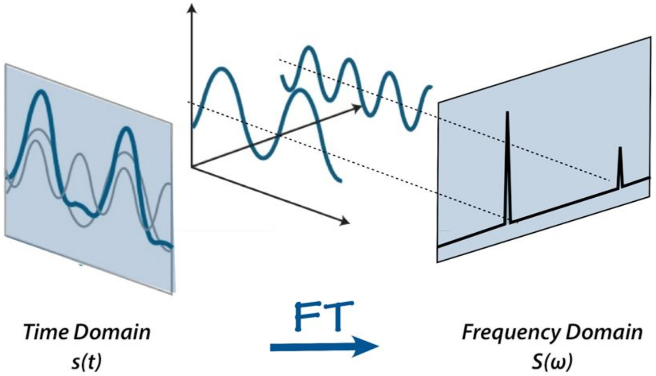


stem_plot.grc

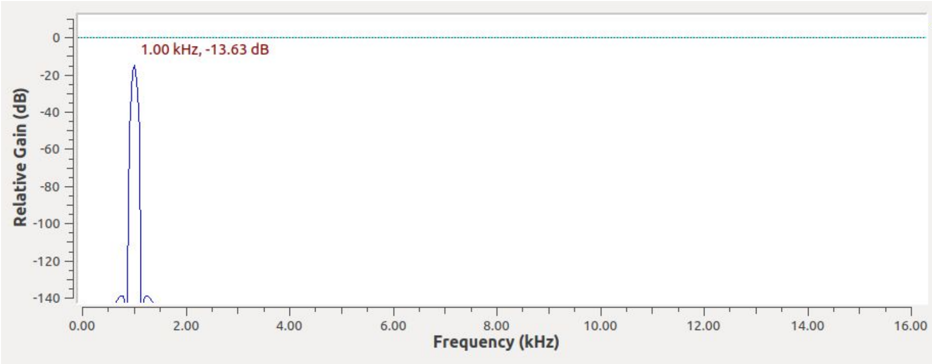


Frequency Domain

- Time domain signals contain energy at certain frequencies
- They can be decomposed into the sum of many sine waves with different amplitudes

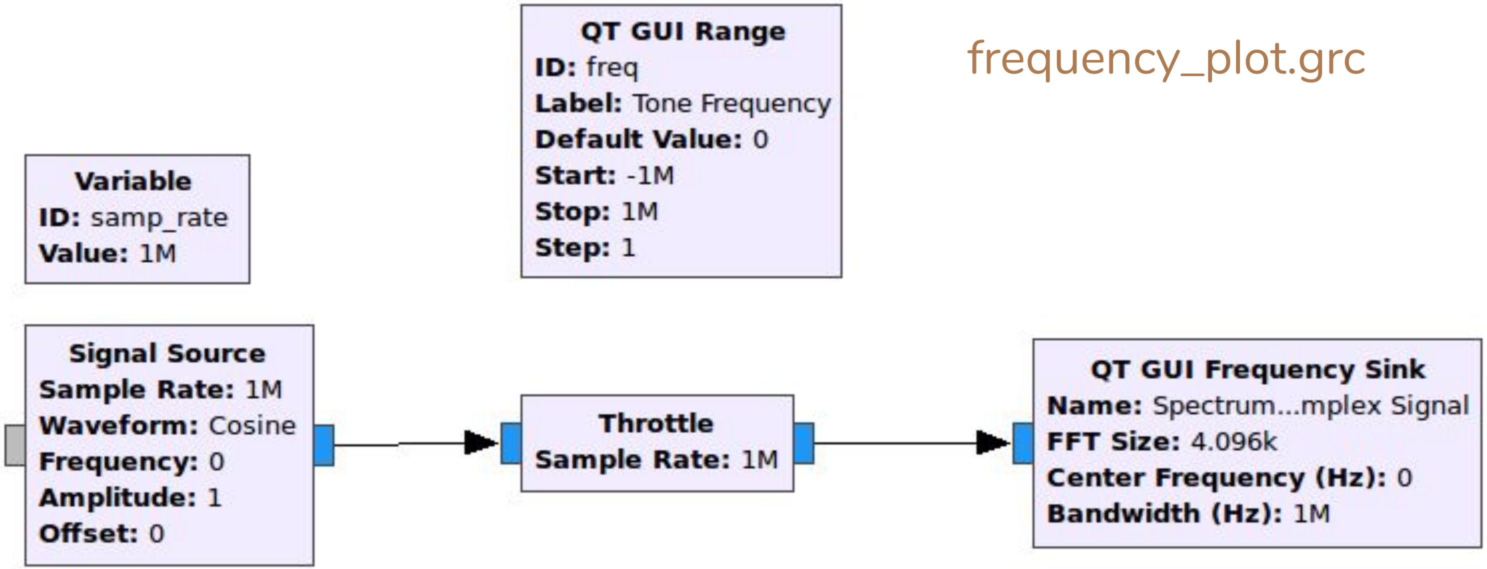


https://irenevigueguix.wordpress.com/2017/02/06/understanding-the-fourier-transform/

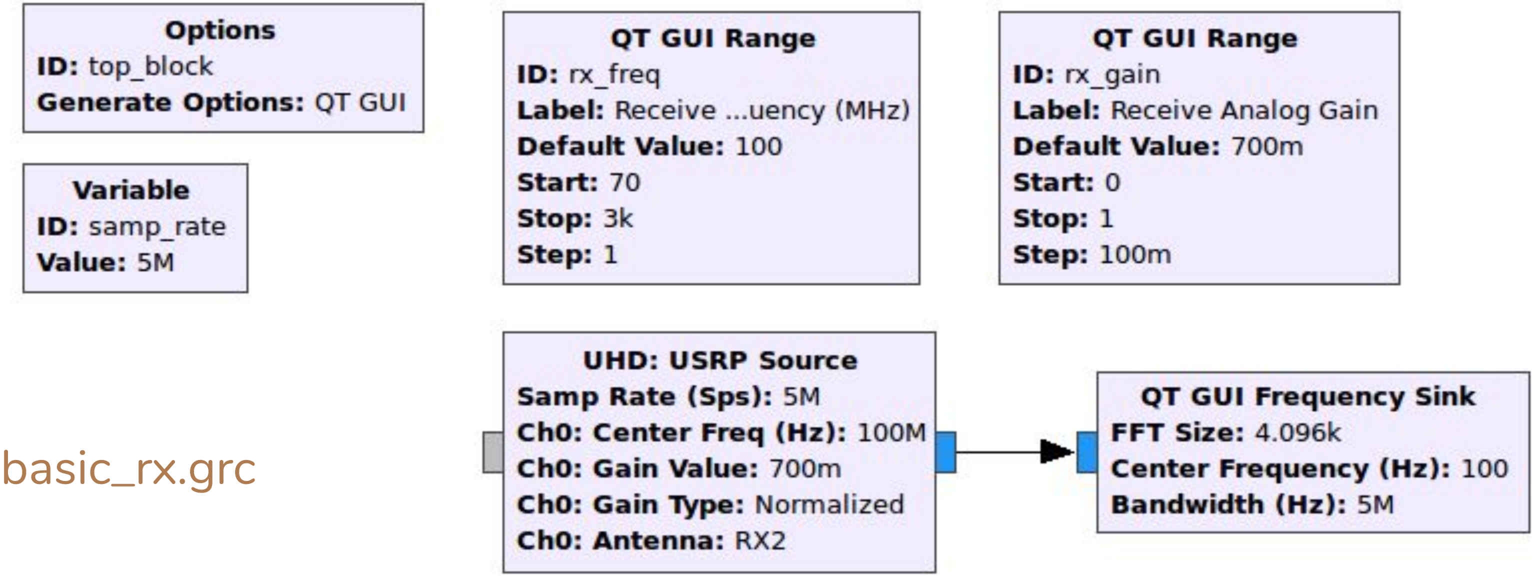


Frequency Sink

frequency_plot.grc



Simple Receiver



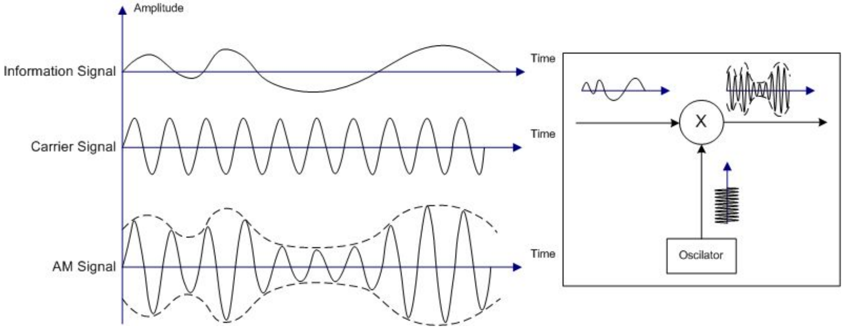
basic_rx.grc

Carrying Information

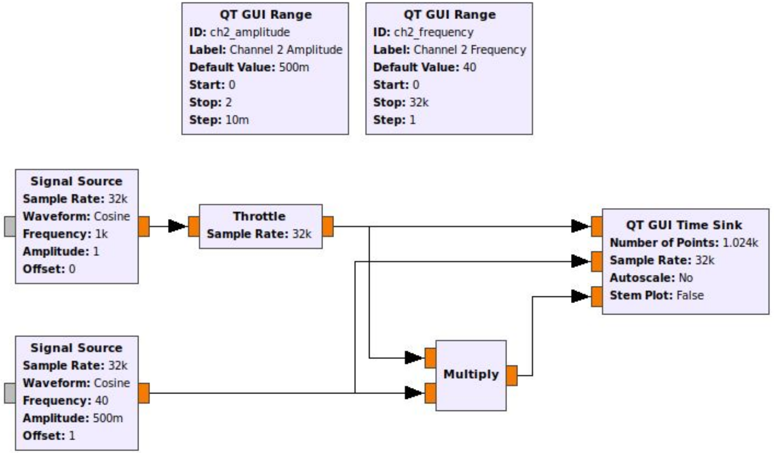
- Frequency, Amplitude, and Phase can all be changed over time
  - This change change of the signal is Modulation

Amplitude Modulation

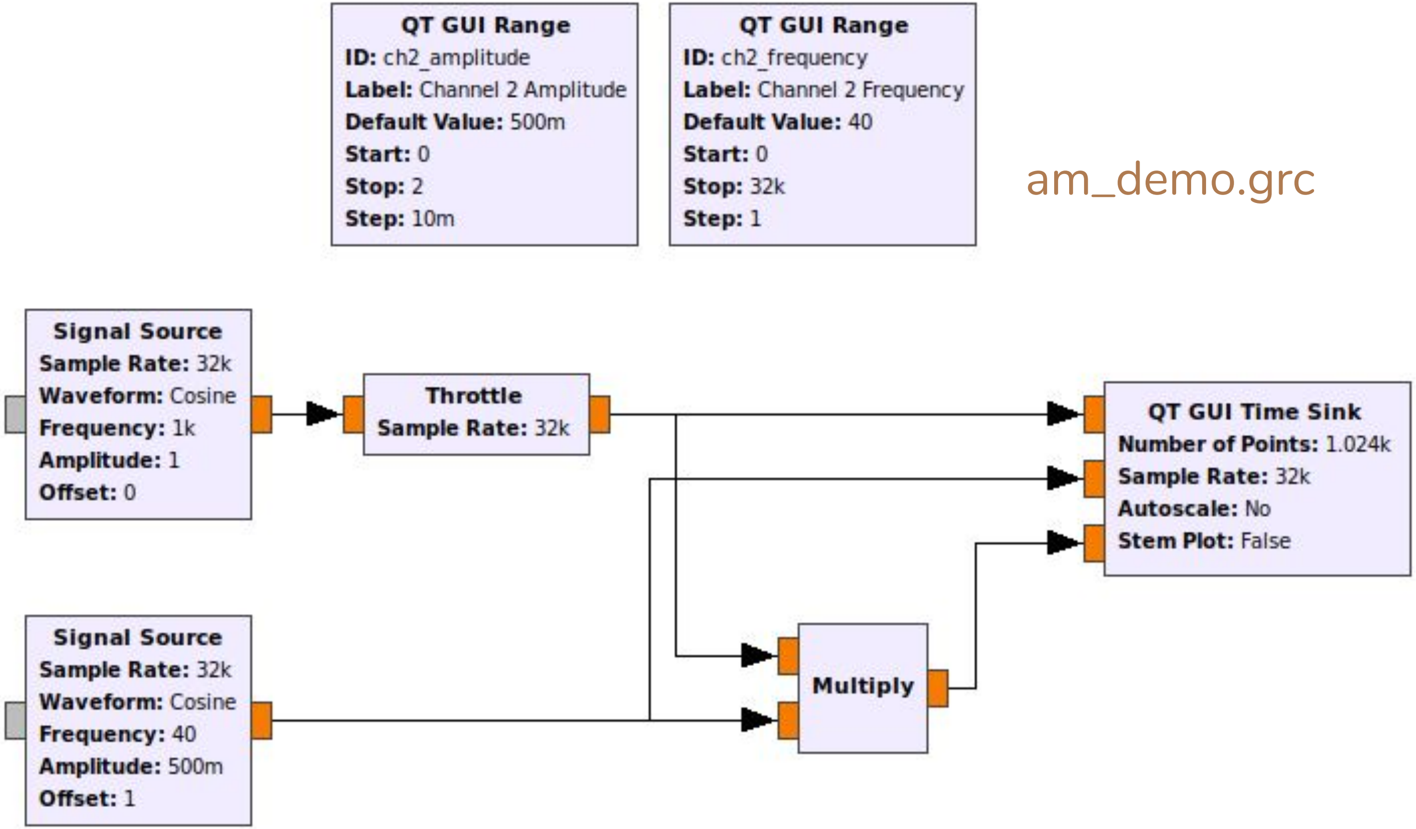
- Changing the amplitude of a “carrier” wave at a fixed frequency



https://en.wikipedia.org/wiki/Amplitude_modulation

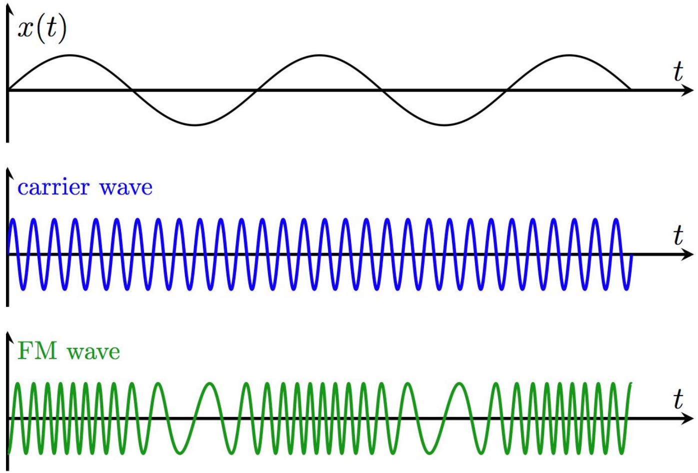


am_demo.grc

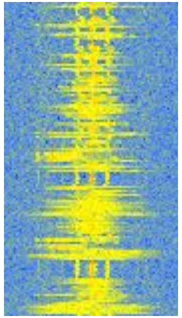


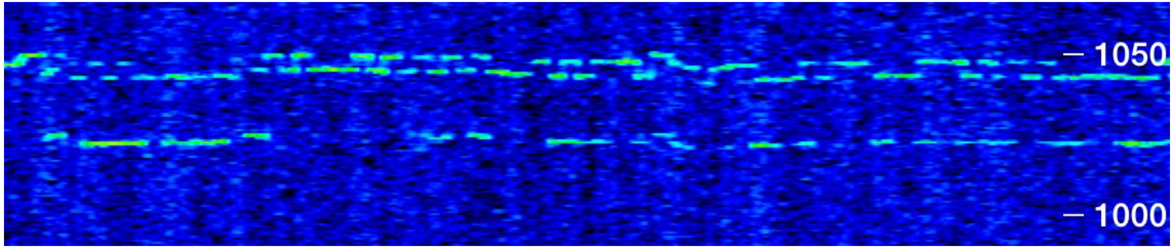
https://en.wikipedia.org/wiki/Amplitude_modulation

Frequency Modulation



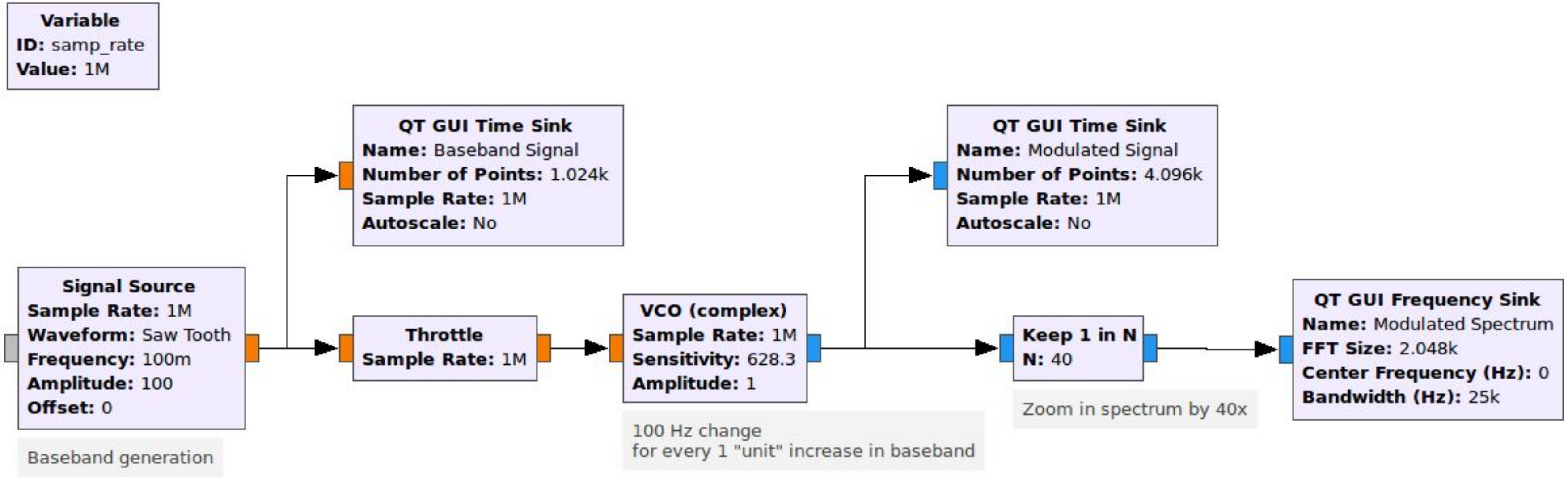
- Changing the frequency of a carrier wave
- Either discrete steps (Frequency Shift Keying)
- Or continuous (Broadcast FM)





https://irenevigueguix.wordpress.com/2017/02/06/understanding-the-fourier-transform/

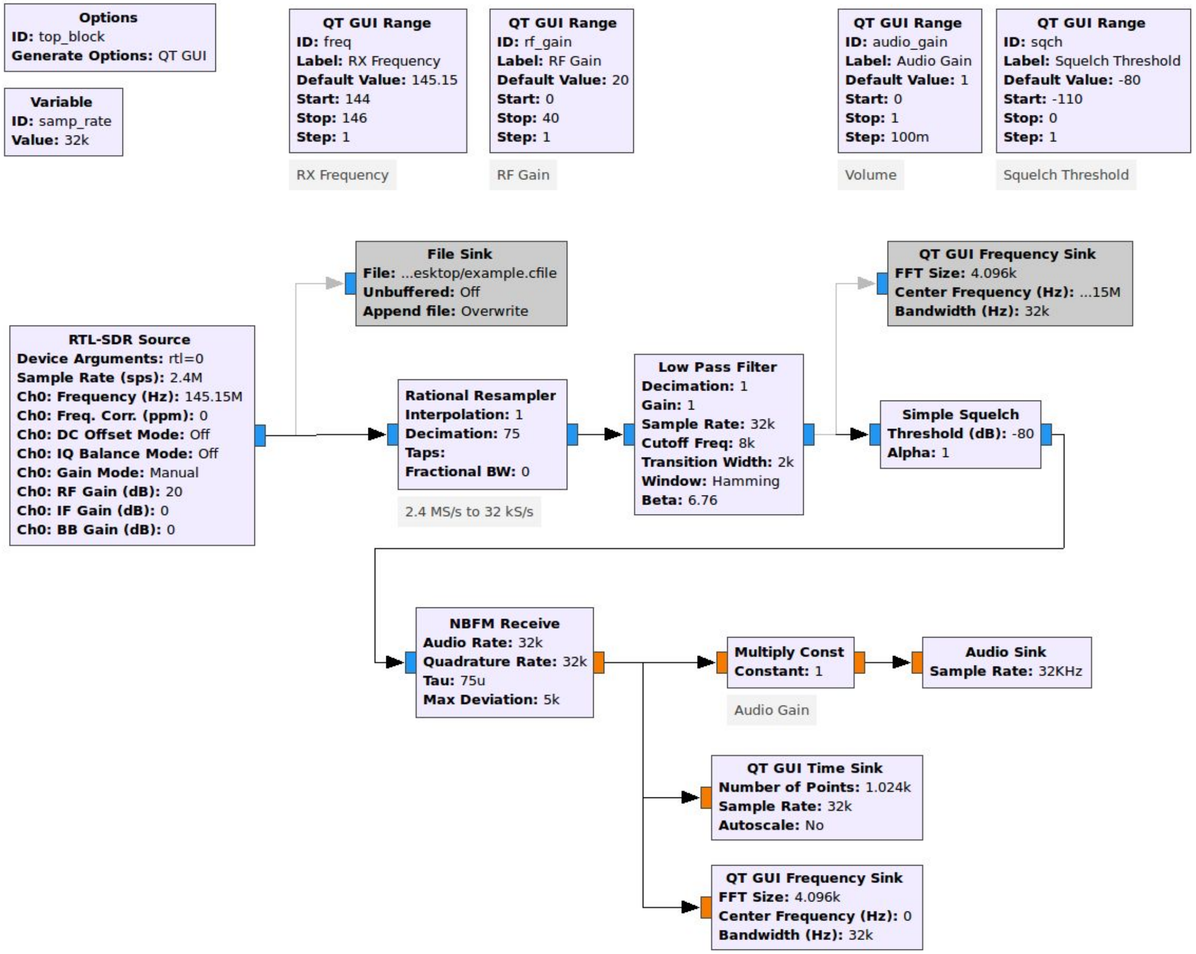
FM Modulation Example



Narrow Band FM

- Popular analog modulation scheme for voice transmission
    - Walkie Talkies, Land Mobile Radio
- Could implement each step of the modulation and demodulation
- GNU Radio already has it packaged

Narrow Band FM Receiver



NBFM Receiver - Notes

- Soundcards will support different rates, 32 and 44.1 kHz pretty universal
- Thoughtful selection of SDR sampling rate makes decimation simple (1/75)
  - Avoid large fractions (i.e. 1023/127) as they require LOTs of computation
- Squelch is in dB Full Scale, not dBm or dbW
  - GNU Radio has no way of knowing an absolute power level
- NBFM block
  - Can decimate, but usually set output and input sample rates to the same
  - Deviation and pre-emphasis (tau) are dependent on the transmitter, default values will work in most cases

Underruns

- Soundcards and transmitters are hard-realtime systems, you must supply enough data to keep them always running
  - Failing to do so will cause an “underrun”
  - In RF will produce gaps in the transmission and splatter
  - In Audio will produce gaps and clicks
- GNU Radio will print “U” for underruns with USRPs and “aU” for soundcards (audio Underrun)

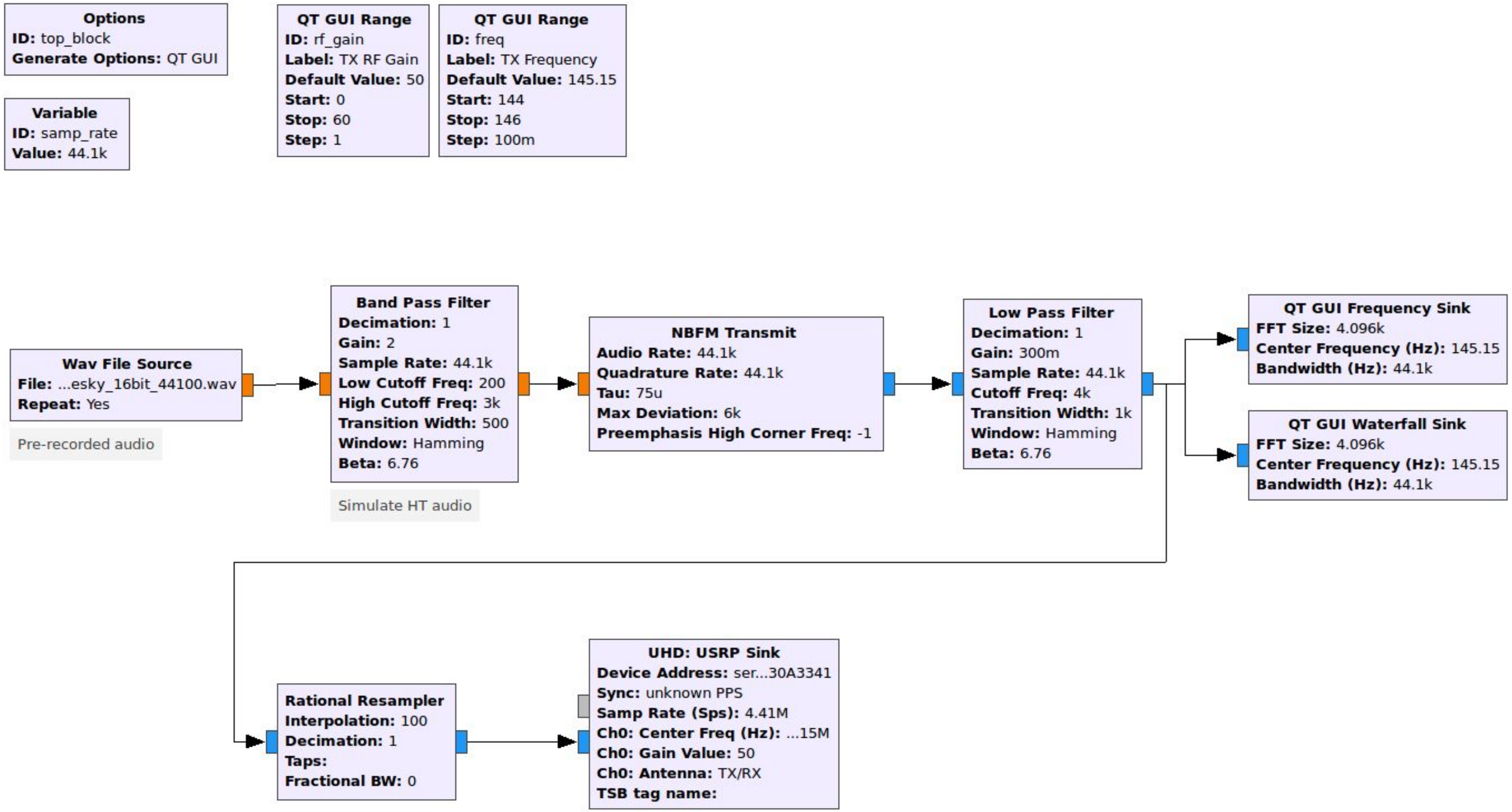
Two Clock Problem

- SDR Transmitter or receiver has an internal reference oscillator, so does a soundcard
- If the two references are not EXACTLY the same there's a problem
  - Source (producer) frequency > Sink (consumer) means too many samples are available, will build up a backlog of data to handle
    - In to Out delay will increase (Audio will lag)
  - Source < Sink means not enough data is available, underruns will occur

Mitigating the Two-Clock Problem

- Use the same reference oscillator for source and sink sample clocks (ADCs & DACs)
  - Great answer if using the same hardware for both, difficult (or impossible) with an SDR and soundcard
- Increase buffer sizes
  - Store more data before telling output to start
  - Reduces how often underruns occur
    - I.E. run out of data once a minute rather than 0.1 seconds

NBFM Transmitter



NBFM Transmitter - Notes

- USRP hardware sink sets transmit frequency, RF gain, and expected sample rate
    - USRP B200 (my demo hardware) is very flexible in sample rates, usually hardware will support specific rates
- Software interpolation/decimation will have sharper (better) filtering than FPGA or analog filters
    - This is a generalization but usually true
    - Interpolating by 100x means we have a clean signal but still very manageable sample rate (4.41 MS/s, easy for USB)
- Use the time and frequency sinks to plot signals at different points (think spectrum analyzer and oscilloscopes when debugging)
- Confirm functionality off the air before including hardware (simulation)
- FM is forgiving with filtering
    - Accidentally generated 6 kHz deviation, filtered to 4k Hz, received with 5 kHz, still works
    - Partially thanks to filter transition bandwidth

Useful Tips

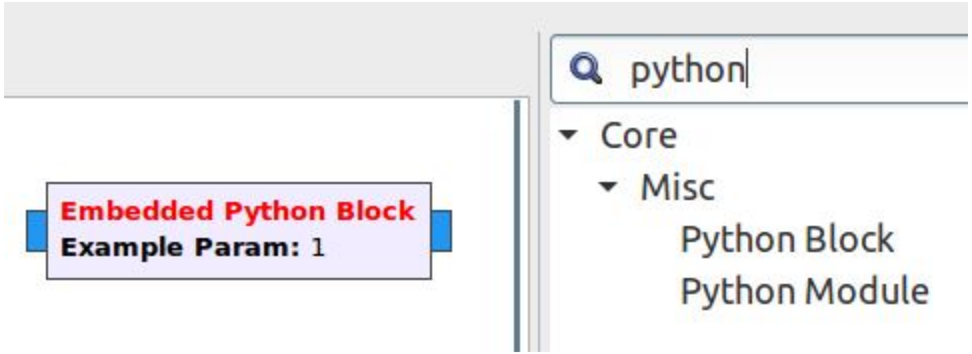
- Test/develop using a pre-recorded audio file
  - Expected format is 16 bit real valued samples
  - Sample rate chosen as 32 kHz to match what a soundcard (Mic in) would likely generate
- Add comments
  - Text box in the “Advanced” tab of each block
- Use variables and sliders (“Range” in QT”
  - Lets you experiment quickly with values to hand tune performance

Programming Languages

- GNU Radio has a core written in C++
  - The main engine and all default blocks are C++
- Python is wrapped around the C++
  - Generally considered more experimenter friendly
  - Only small performance hit as main work is done in C++ land
- GRC is entirely written in Python
  - But again, the engine is C++, so best of both worlds

Python Block

- Lets draw back the curtain and peek at the insides
- The “Embedded Python Block” lets you add custom code to a GRC flowgraph very easily
  - Code is stored in the .grc file
  - Default template supplies basic features



Embedded Python Block

- Add a “Python Block” to the flowgraph, open it and click “Open in Editor” and use the Default
- The template has all the main features of a GNU Radio block setup already

```python
"""
Embedded Python Blocks:

Each time this file is saved, GRC will instantiate the first class it finds
to get ports and parameters of your block. The arguments to __init__  will
be the parameters. All of them are required to have default values!
"""

import numpy as np
from gnuradio import gr


class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def __init__(self, example_param=1.0):  # only default arguments here
        """arguments to this function show up as parameters in GRC"""
        gr.sync_block.__init__(
            self,
            name='Embedded Python Block',   # will show up in GRC
            in_sig=[np.complex64],
            out_sig=[np.complex64]
        )
        # if an attribute with the same name as a parameter is found,
        # a callback is registered (properties work, too).
        self.example_param = example_param

    def work(self, input_items, output_items):
        """example: multiply with constant"""
        output_items[0][:] = input_items[0] * self.example_param
        return len(output_items[0])
```

Headers and Includes

```
"""
Embedded Python Blocks:

Each time this file is saved, GRC will instantiate the first class it finds
to get ports and parameters of your block. The arguments to __init__ will
be the parameters. All of them are required to have default values!
"""

import numpy as np
from gnuradio import gr
```

- The red text surrounded by quotes is a comment explaining how the template works
- The import lines pull in code from gnuradio and numpy
  - numpy is a Python library of math functions that GNU Radio uses extensively
- You could add more imports to use other libraries

Class and Initialization

```
class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def __init__(self, example_param=1.0):  # only default arguments here
        """arguments to this function show up as parameters in GRC"""
        gr.sync_block.__init__(
            self,
            name='Embedded Python Block',   # will show up in GRC
            in_sig=[np.complex64],
            out_sig=[np.complex64]
        )
```

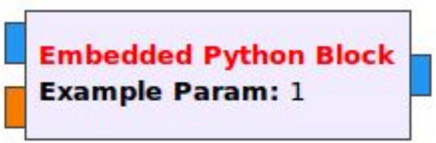
- GNU Radio has several types (or “classes”) of blocks
  - We’re using a sync block since input and output rates are the same (synchronous)
- The next comment will appear in the block documentation tab
- The “__init__” function setups (initializes) our block
  - We have one parameter called example_param with a default value of 1.0

Block Initialization

```
class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def __init__(self, example_param=1.0):  # only default arguments here
        """arguments to this function show up as parameters in GRC"""
        gr.sync_block.__init__(
            self,
            name='Embedded Python Block',   # will show up in GRC
            in_sig=[np.complex64],
            out_sig=[np.complex64]
        )
```

- GNU Radio already knows a lot about blocks. We just have to fill in the specific details by calling gr.sync_block.__init(….)
    - name is just for humans
- in_sig/out_sig is the “signature” of the input/output
    - How many channels, what type of data (1 channel of complex data)
    - The data types are numpy since this is Python

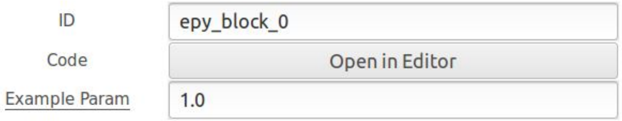
Block Initialization - Continued



- in_sig=[np.complex64, np.float32] would be 1 channel complex and 1 channel real floats

- If you want to be able to change a value while the flowgraph is running (with a Range slider for instance) then create a “class attribute” like the following:

```
# if an attribute with the same name as a parameter is found,
# a callback is registered (properties work, too).
self.example_param = example_param
```



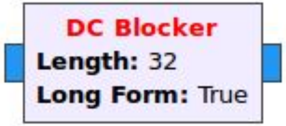
- GRC will automatically add code to update the value correctly
    - Only values with an underline in GRC can be changed at runtime

Doing Work on Samples

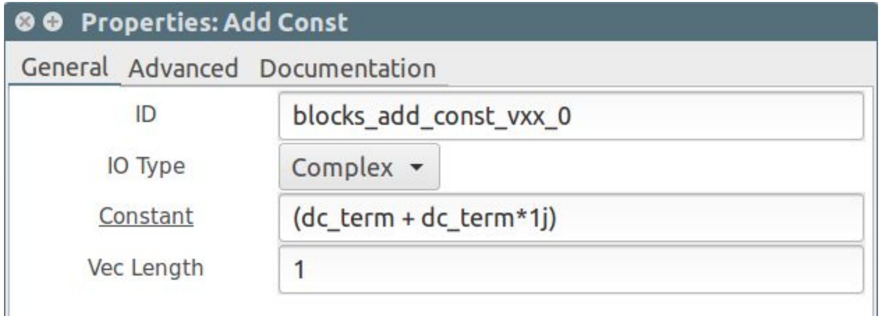
```
def work(self, input_items, output_items):
    """example: multiply with constant"""
    output_items[0][:] = input_items[0] * self.example_param
    return len(output_items[0])
```

- The main purpose of most blocks is to do something with or to samples
  - GNU Radio will call the work function with a bunch of input samples and a place to put the output samples
- The default template multiplies each sample by a value (example_param)
- We need to tell GNU Radio how many samples we've produced
  - In this case we've used all the input to make the same number of output samples
  - The len function gives the length of the output_items array, so we return that number to GNU Radio's engine
- Clearly some Python knowledge is needed, but most of the heavy lifting already done

DC Offset Example



- Same template but cleaned up
- Let's introduce a DC component to the signal
  - Usually a terrible idea
  - Could have used an Add Const block

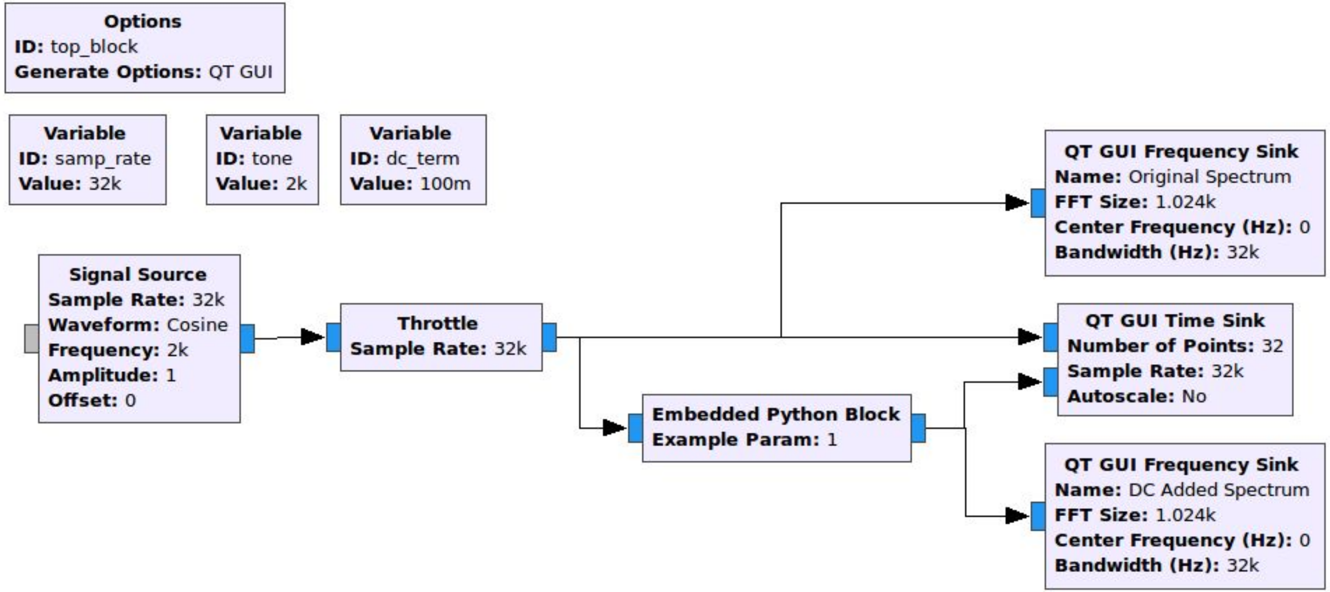


Trivia:

Can remove a DC offset using the DC Blocker

DC Offset Test Setup

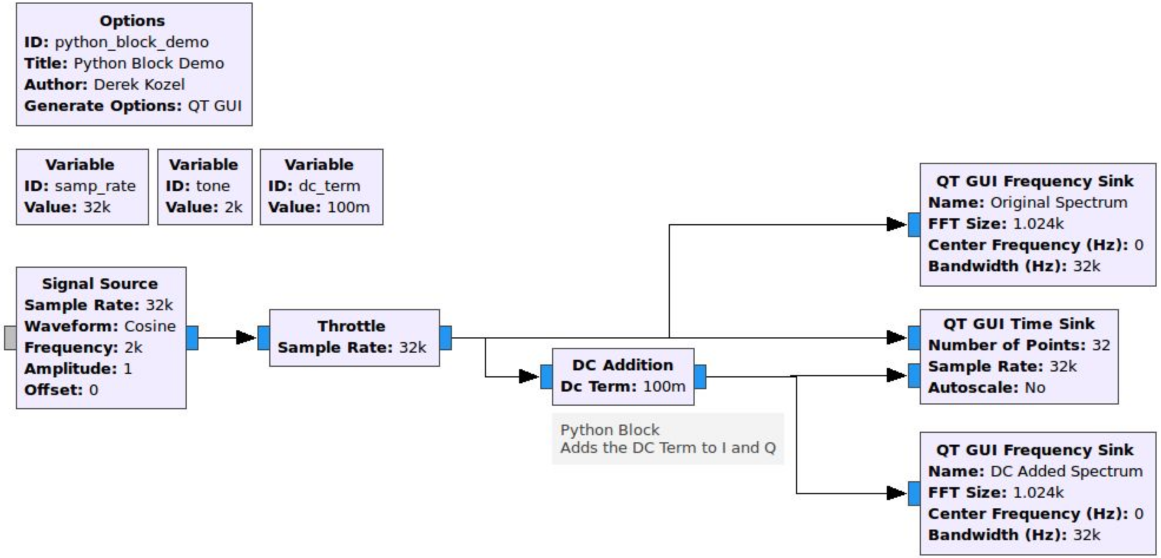
- Basic testing setup with an Embedded Python Block



```python
import numpy as np
from gnuradio import gr

class blk(gr.sync_block):
    # Block Documentation
    """DC Addition Block - Surely more is better!"""

    def __init__(self, dc_term=0.1):  # One parameter

        gr.sync_block.__init__(
            self,
            name='DC Addition',    # Will show up in GRC
            in_sig=[np.complex64], # Complex float 32 bit pairs
            out_sig=[np.complex64] # Complex float 32 bit pairs
        )

        self.dc_term = dc_term

    def work(self, input_items, output_items):

        # Add the value of "dc_term" to the I and Q parts of the signal
        # For example: output = input + (0.1 + j0.1)
        output_items[0][:] = input_items[0] + np.complex64(self.dc_term+self.dc_term*1j)

        # Tell GNU Radio's scheduler how many samples we are outputting
        return len(output_items[0])
```

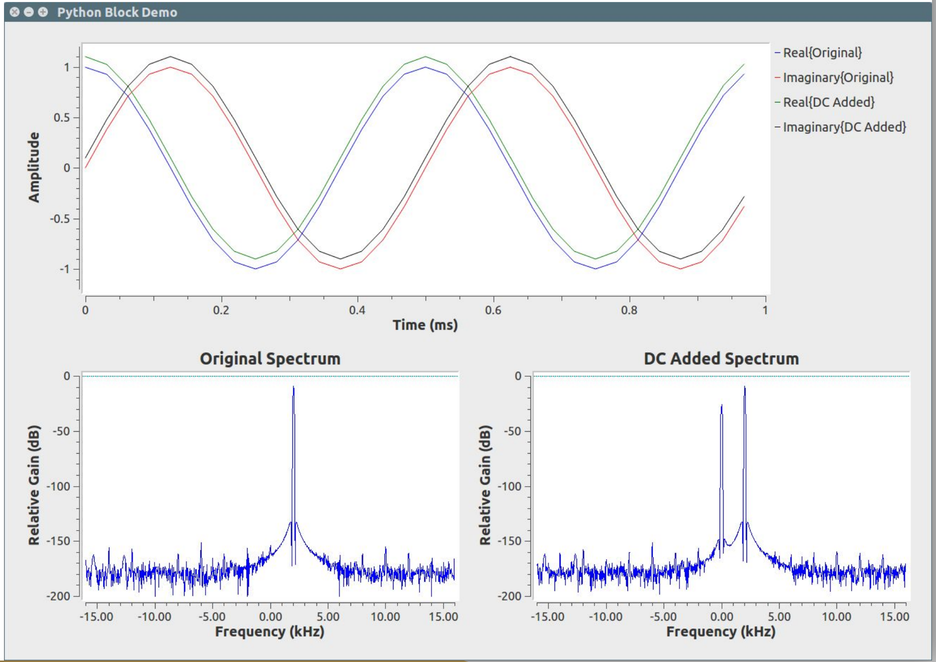
DC Offset Results

- Looks like a real block!



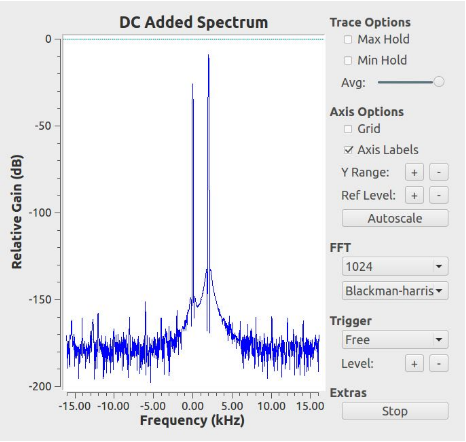
DC Offset Results

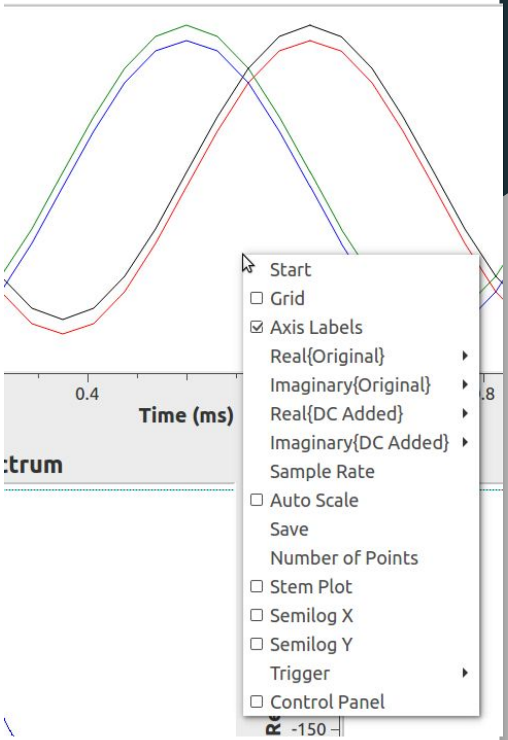
- DC Offset clearly visible in time and frequency



Quick Tips

- Click on the line labels in the Time plot to hide or show a particular line
    - Works on other visual sinks too
- Middle mouse click on a QT plot to bring up a menu of options.
- Enable a Control Panel in the Advanced Tab





User Manual and Documentation

- A bit spread out and wanting in depth in spots
- User Manual: www.gnuradio.org/doc/doxygen
  - Generated from the C++
  - Useful for finding out more about blocks
  - Talks about the design of the core engine and code
- Python Manual: www.gnuradio.org/doc/sphinx
  - Generated from the Python
  - Does not cover many of the topics in main manual
  - Likely to be combined with the C++ in the next year

User Manual and Documentation

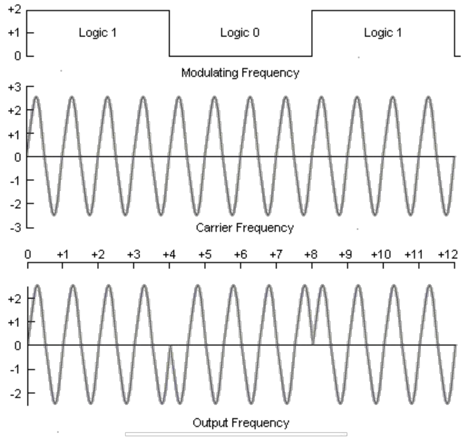
- Wiki: http://wiki.gnuradio.org
  - Several sets of tutorials
  - Presentations from other classes and events
  - Working groups and developer info
  - GNU Radio Conference info
    - Links to videos and slides from the talks
  - Lots of outdated pages, getting cleaner over time

Main Website

- www.gnuradio.org
- Blog
  - Short and long posts about significant events
- Releases
  - Description of changes in new versions
- Links to everything on the previous page

Phase Modulation

- Introduce changes in the carrier's phase to signal information



Introduction to Amateur Radio

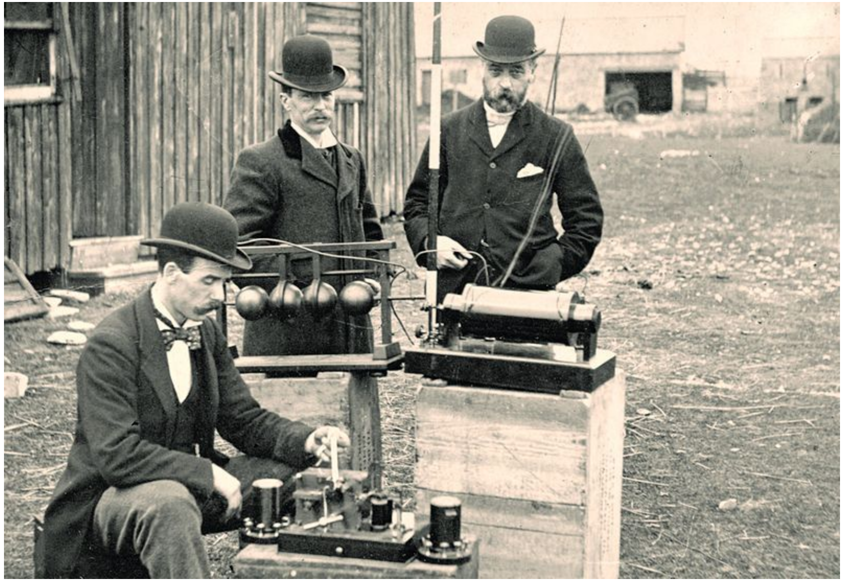
Amateur Radio History

1896 - Marconi transmits and receives Morse Code over 2 km on Salisbury Plain

1897 - Marconi transmits to Ireland from Lavernock Point just south of Cardiff

1913 - The London Wireless Club is founded, becomes the Radio Society of Great Britain later

1923 - First two way contact between UK and US



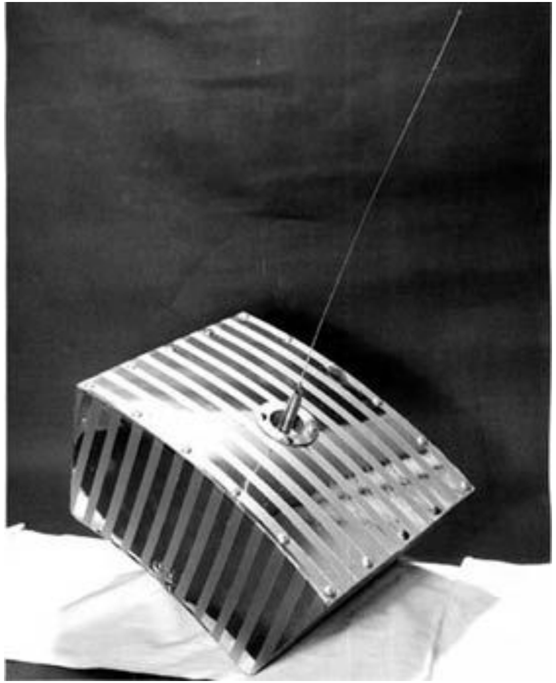
Amateur Radio History

1961 - OSCAR 1 launched, the first Orbiting Satellite Carrying Amateur Radio

2003 - Morse code requirement dropped in the UK

2009 - AMSAT-DL successfully bounces signal off Venus

2018 - DSLWP satellite launched into moon orbit



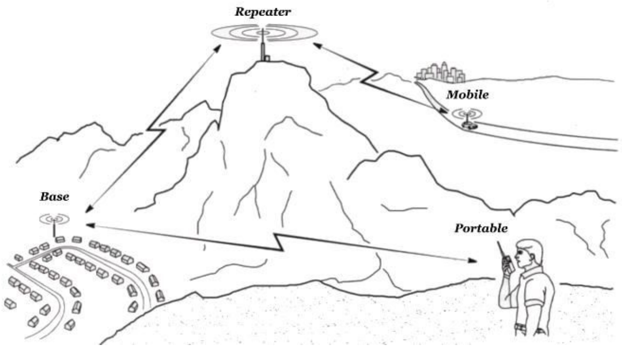
https://en.wikipedia.org/wiki/Amateur_radio_satellite

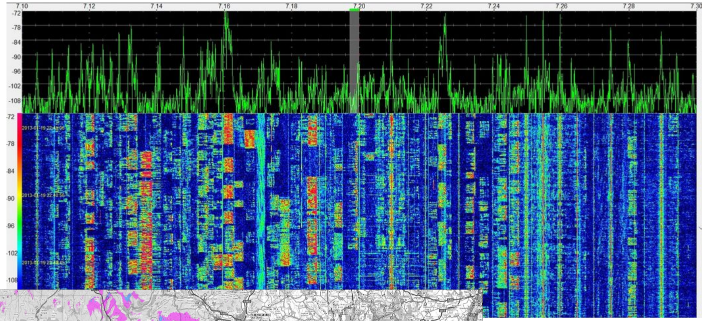
Spectrum Access

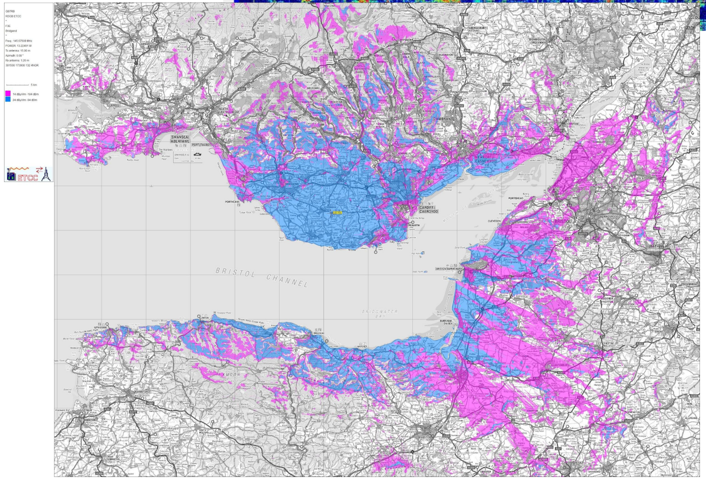
- Licenses granted by Ofcom (National Regulator) give access to 100s of MHz of spectrum
- Bands from 135 kHz to 250 GHz
- Up to 400 Watts of output power (+antenna gain!)
- Intended for experimental use, to promote radio skills, and international good will
  - Commercial applications not allowed

https://en.wikipedia.org/wiki/Amateur_radio_satellite

Social - On the air







http://www.gb7cd.co.uk/Coverage.html

Social - Clubs/Societies

Social/Learning - Conventions

- Numerous events in the UK and around the world
  - Radio Society of Great Britain Convention - Milton Keynes - October 11-13
  - HAMRADIO - Friedrichshafen, Germany - June 21-23
- UK Microwave Group
  - Regular Roundtables around the UK
  - Probable event here in Cardiff in March

Wrapping Up

Thanks for Coming

- Questions?
- The latest version of these slides can always be found at

  http://www.derekkozel.com/talks

- Twitter: @derekkozel
- Email: derek@bitstovolts.com