

Introductory Tutorial for SDR and GNU Radio Beginners

GNU Radio Conference 2023
Murat Sever

Outline

- About me
- About tutorial
- GNU Radio
- Lab: Digital Signal Processing (DSP)
- Lab: Software Defined Radio (SDR)
- Lab: Wide Band Frequency Modulation (WBFM)

About me

Part-Time Lecturer at TOBB ETU, Ankara,
Turkey

- ELE361L Course / Telecom Laboratory
 - Summer 2021
 - Fall 2022
 - *Fall 2023*

Project Owner & Manager

- SDR-Powered Education



AMATEUR RADIO DIGITAL COMMUNICATIONS

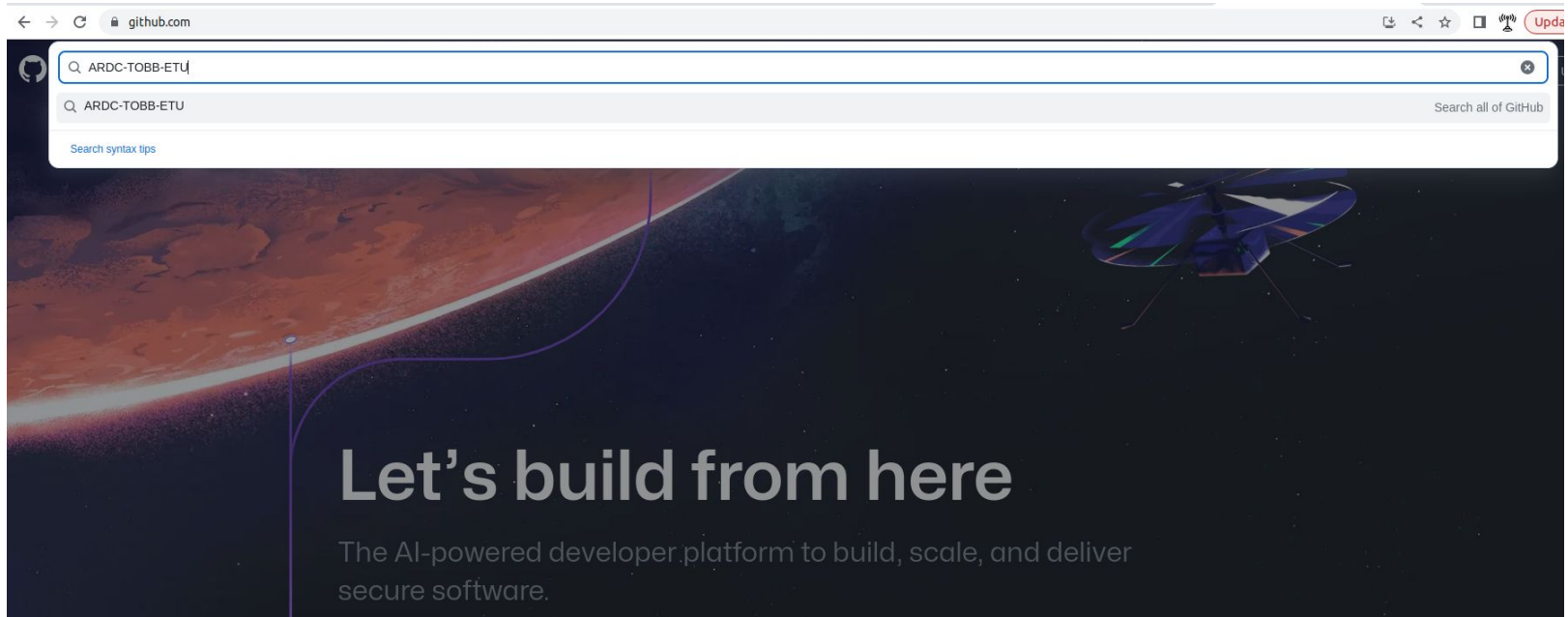


About tutorial

- Introduces fundamental DSP concepts and GNU Radio to new users
- Consists of the following lab modules based on Jupyter Notebooks
 - Lab DSP
 - Lab SDR
 - Lab WBFM
- We will use GNU Radio for
 - Exploring signals in simulation mode
 - Sound processing
 - Spectrum watching with RTL-SDR
 - Broadcast FM demodulation

Get the labs (if you haven't already)

- Labs available @ GitHub
 - <https://github.com/ARDC-TOBB-ETU/GRCon23Tutorial>



Download or clone

The screenshot shows the GitHub interface for the repository `ARDC-TOBB-ETU/GRCon23Tutorial`. The repository is public and has 1 branch and 0 tags. The file list includes:

File Name	Commit Message	Time
LabDSP	Initial content uploaded	
LabSDR	Initial content uploaded	
LabWBFM	Initial content uploaded	
.gitignore	Initial commit	
LICENSE	Initial commit	
README.md	updated README	2 days ago
install.sh	update installation	3 days ago
requirements.txt	added installation files	3 days ago

The 'Clone' dropdown menu is open, showing options for Local and Codespaces. The 'Clone' option is selected, and the 'HTTPS' method is chosen. The URL is `https://github.com/ARDC-TOBB-ETU/GRCon23T`. The 'Download ZIP' option is also visible.

The README content is as follows:

GRCon23Tutorial

Introductory Tutorial for SDR and GNU Radio Beginners

This course consists of many labs and each lab is built on Jupyter Notebook. Jupyter Notebook is chosen to present information and to guide participant what to do. Jupyter makes the course interactive and we will explore signals around in a hands-on fashion. Every notebook contains tasks that participant will attempt.

On the right side of the page, there are sections for 'About', 'Releases', 'Packages', and 'Languages'. The 'About' section includes: Introductory Tutorial for SDR and GNU Radio Beginners, Readme, CC0-1.0 license, Activity, 0 stars, 1 watching, 0 forks, and Report repository. The 'Releases' section shows 'No releases published'. The 'Packages' section shows 'No packages published'. The 'Languages' section shows 'Jupyter Notebook 100.0%'.

Use README to install

- Linux/Mac
 - Run `./install.sh`
- Windows
 - Install miniforge
 - Create a new environment
 - `conda config --append channels conda-forge`
 - `conda create --name GRCon23 --file requirements.txt`

Opening Jupyter Notebooks/GNU Radio

- Linux/Mac
 - Run `source "${HOME}/conda/etc/profile.d/conda.sh"`
 - **Activate the environment** `conda activate GRCon23`
 - Run `jupyter-lab`
 - Run `gnuradio-companion`
- Windows
 - Open a miniforge prompt
 - **Activate the environment** `conda activate GRCon23`
 - Run `jupyter-lab`
 - Run `gnuradio-companion`

Jupyter Notebook

localhost:8888/lab

File Edit View Run Kernel Tabs Settings Help

Filter files by name

Name	Last Modified
LabDSP	6 minutes ago
LabSDR	3 months ago
LabWBFM	3 months ago
install.sh	3 days ago
LICENSE	3 months ago
README.md	2 days ago
requirements.txt	3 days ago

Launcher

Notebook

Python 3 (ipykernel)

Console

Python 3 (ipykernel)

Other

Terminal Text File Markdown File Python File Show Contextual Help

Simple 0 1

Launcher

Outline

- About me
- About tutorial
- **GNU Radio (slides from a previous presentation)**
- Lab: Digital Signal Processing (DSP)
- Lab: Software Defined Radio (SDR)
- Lab: Wide Band Frequency Modulation (WBFM)

GNU Radio is...

- A signal processing library
- Designed for real-time
- The software part of an SDR
- Not a radio application
- The tool to **build your own** transceivers
- **FOSS**: Free and Open Source Software

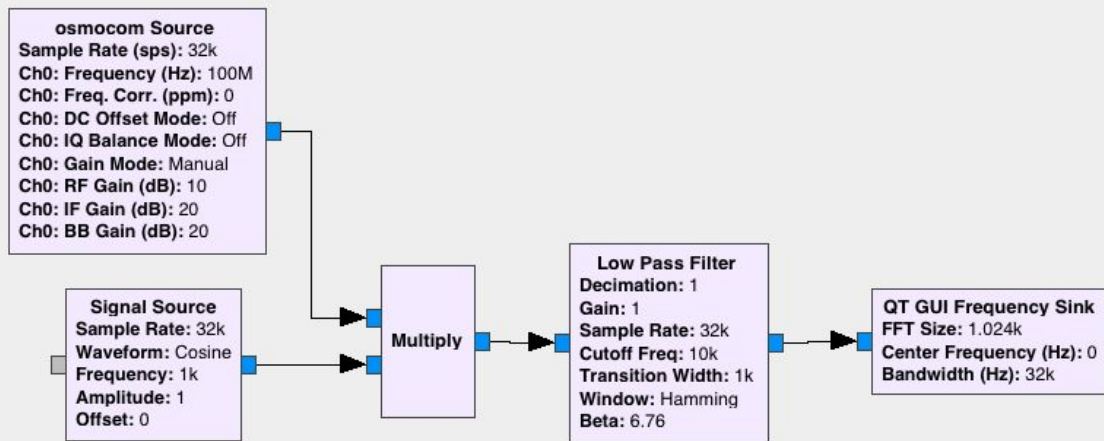


GNU Radio

- Open-source framework for SDR and signal processing
 - Founded by Eric Blossom in 2001
 - 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
-

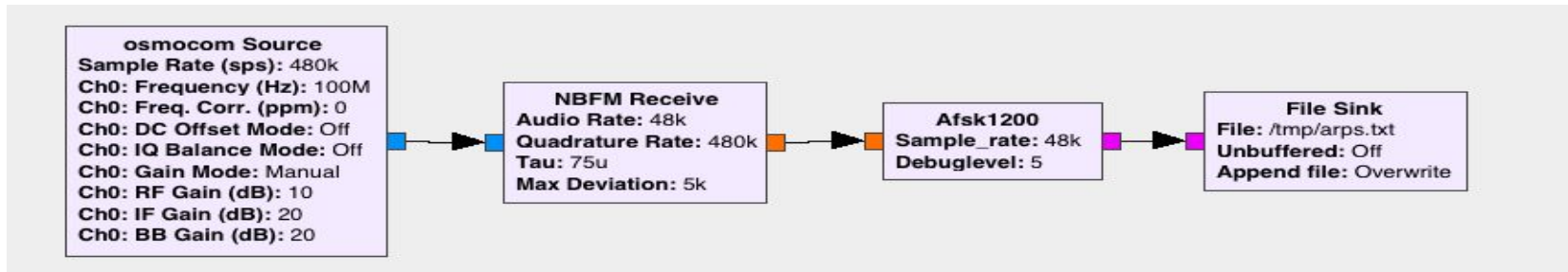
Basic Concept: Flow Graph

- Transceivers are implemented as *flow graphs*
- Similar to Simulink / schematics
- Define structure and parameters of *blocks*



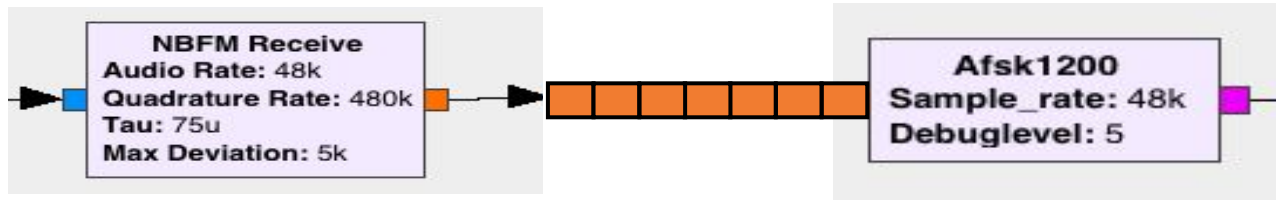
Basic Concept: Block

- Written in C++ or Python
- Implement one logical step
- Each block run in separate thread

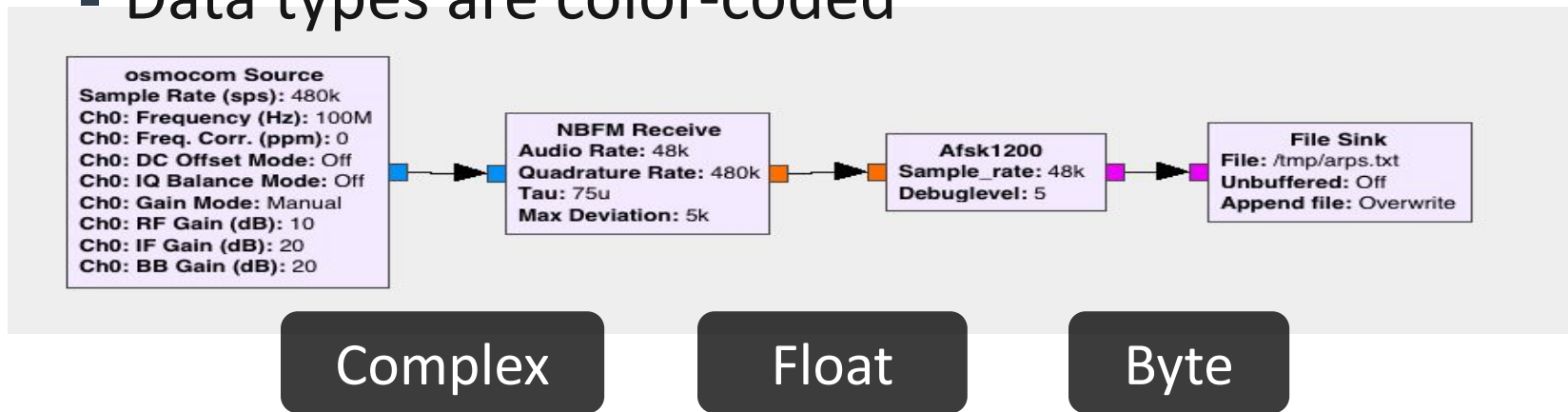


Data Streams

- Samples are buffered

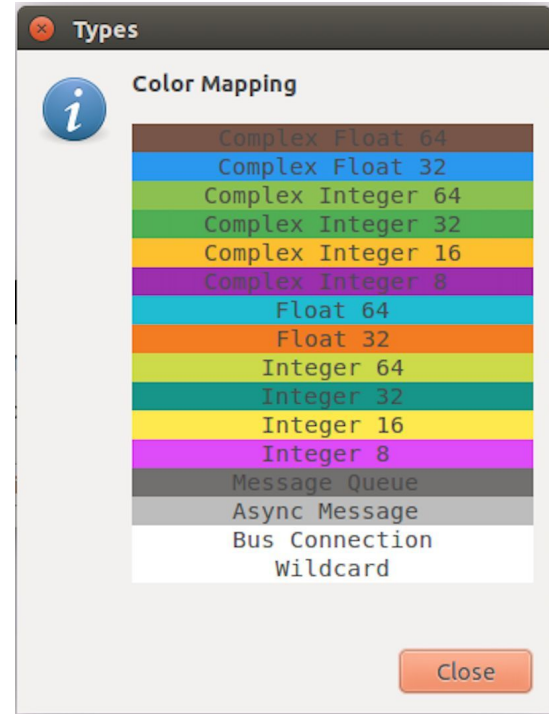


- Data types are color-coded



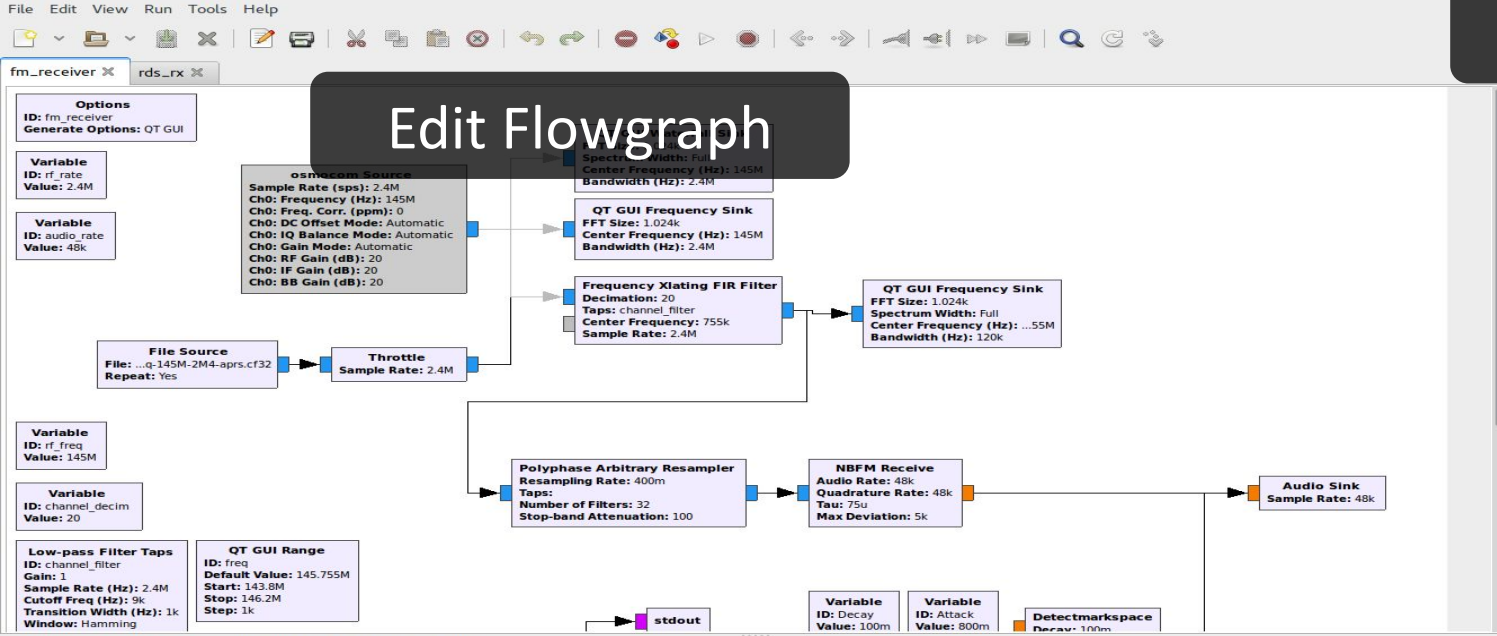
Color Types

Click on menu item Help->Types



GNU Radio Companion

Block Library



Edit Flowgraph

Console

Variables

```
<<< Welcome to GNU Radio Companion 3.7.12git-1109-gcbf30e9c >>>
```

Block paths:

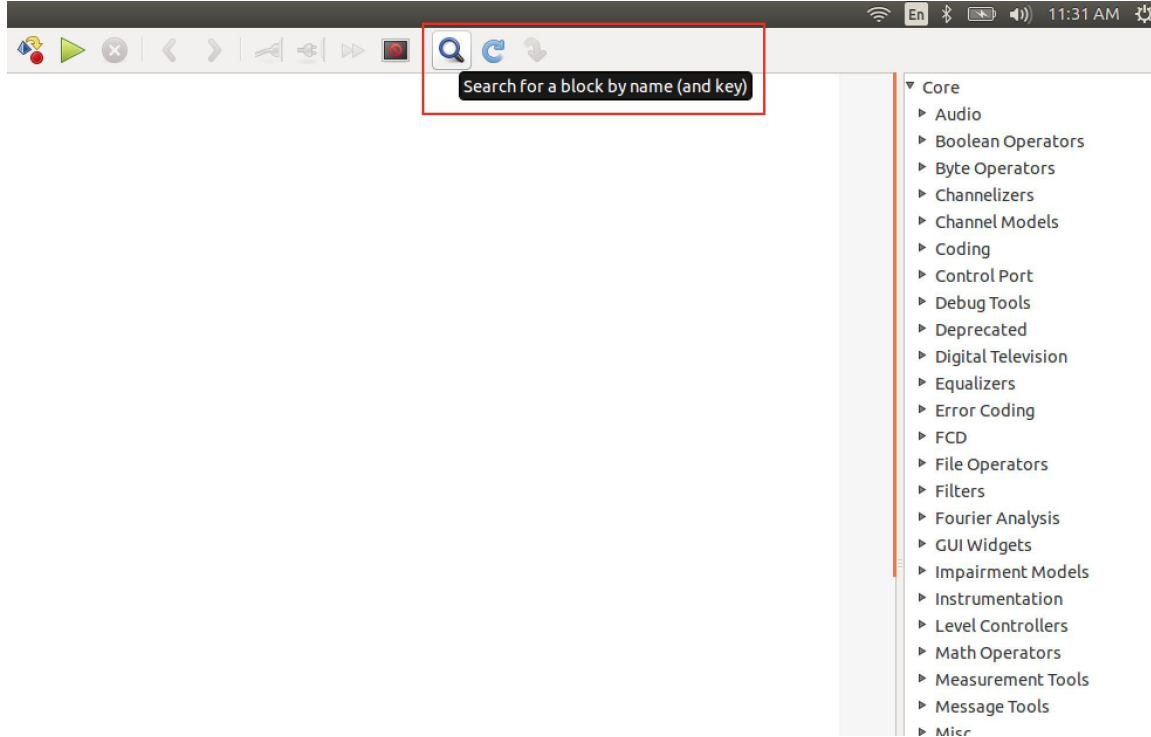
```
/home/basti/.gnc_gnuradio  
/home/basti/usr/gnuradio-next/share/gnuradio/grc/blocks
```

```
Loading: "/home/basti/src/gr-workshop/fm_r...  
>>> Done  
  
Loading: "/home/basti/src/gr-rds/apps/rds_rx.grc"  
>>> Done
```

Id	Value
Imports	
Variables	
Attack	0.8
audio_rate	48000
channel_fc	20
channel_rf	<Open Properties>
Decay	0.1
freq	<Open Properties>

- Core
- Audio
- Boolean Operators
- Byte Operators
- Channelizers
- Channel Models
- Coding
- Control Port
- Debug Tools
- Deprecated
- Digital Television
- Equalizers
- Error Coding
- File Operators
- Filters
- Fourier Analysis
- GUI Widgets
- Impairment Models
- Instrumentation
- Level Controllers
- Math Operators
- Measurement Tools
- Message Tools
- Misc
- Modulators
- Networking Tools
- OFDM
- Packet Operators
- Peak Detectors
- Resamplers
- Stream Operators
- Stream Tap Tools

Search Blocks

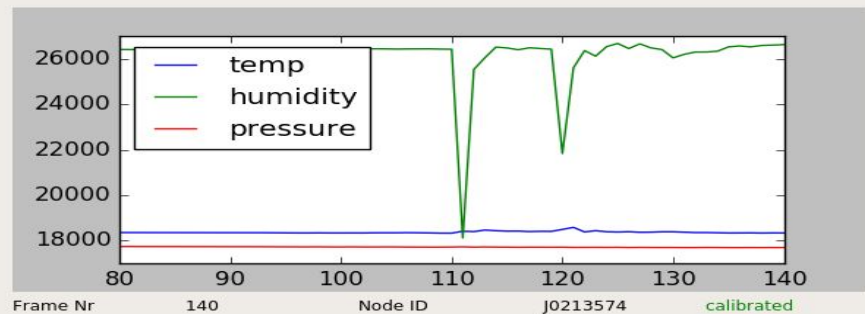
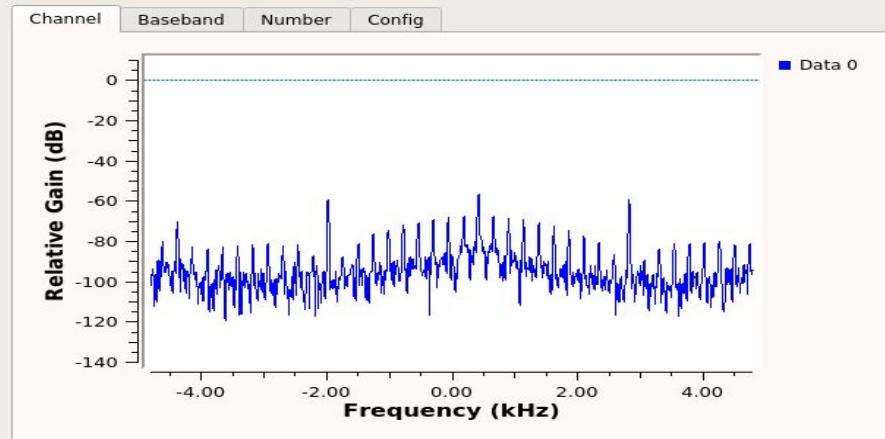
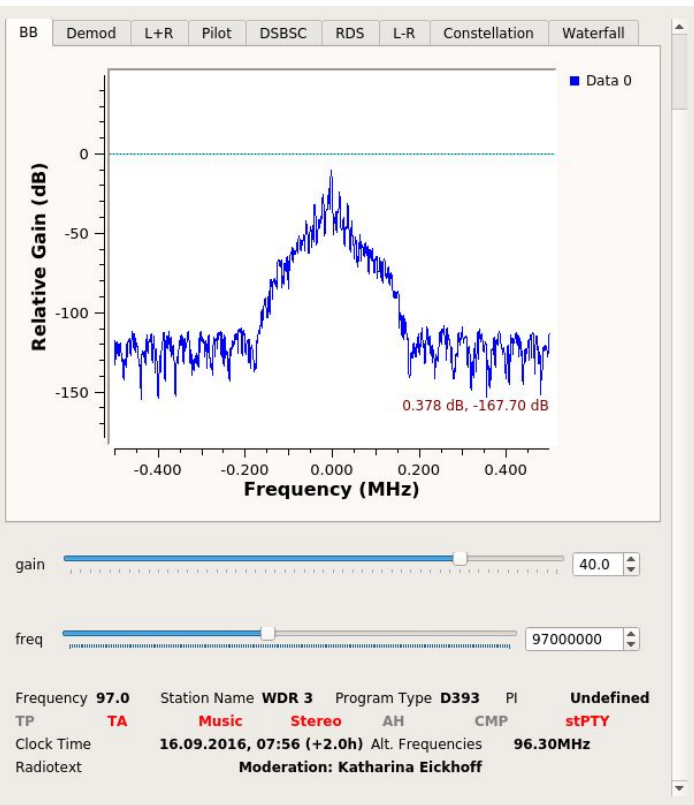


The screenshot shows a software interface with a search bar and a list of search results. The search bar is highlighted with a red box and contains the text "Search for a block by name (and key)". The search results are displayed in a list format, starting with "Core" and followed by various categories such as Audio, Boolean Operators, Byte Operators, Channelizers, Channel Models, Coding, Control Port, Debug Tools, Deprecated, Digital Television, Equalizers, Error Coding, FCD, File Operators, Filters, Fourier Analysis, GUI Widgets, Impairment Models, Instrumentation, Level Controllers, Math Operators, Measurement Tools, Message Tools, and Micr.

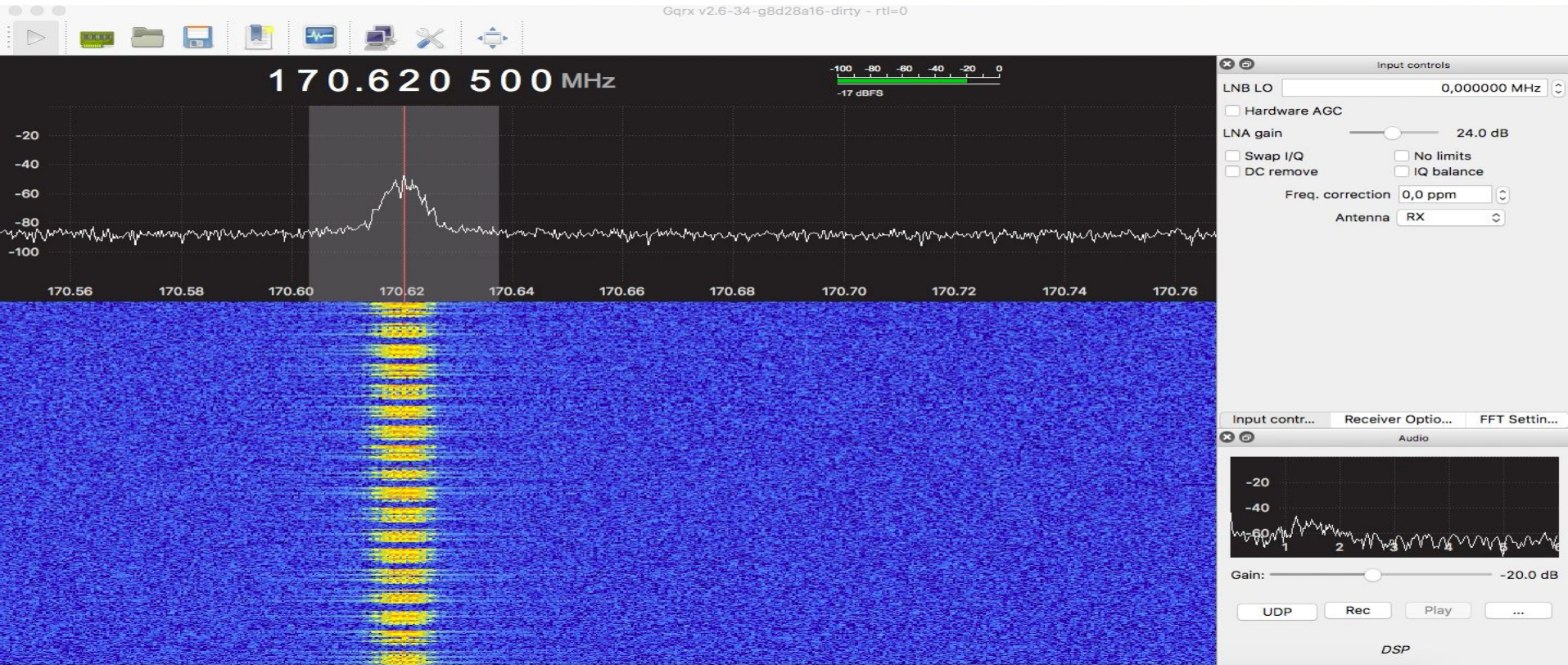
Search for a block by name (and key)

- ▼ Core
 - ▶ Audio
 - ▶ Boolean Operators
 - ▶ Byte Operators
 - ▶ Channelizers
 - ▶ Channel Models
 - ▶ Coding
 - ▶ Control Port
 - ▶ Debug Tools
 - ▶ Deprecated
 - ▶ Digital Television
 - ▶ Equalizers
 - ▶ Error Coding
 - ▶ FCD
 - ▶ File Operators
 - ▶ Filters
 - ▶ Fourier Analysis
 - ▶ GUI Widgets
 - ▶ Impairment Models
 - ▶ Instrumentation
 - ▶ Level Controllers
 - ▶ Math Operators
 - ▶ Measurement Tools
 - ▶ Message Tools
 - ▶ Micr

GUI Output and Instrumentation



GQRX - a GNU Radio Application



Out Of Tree Modules

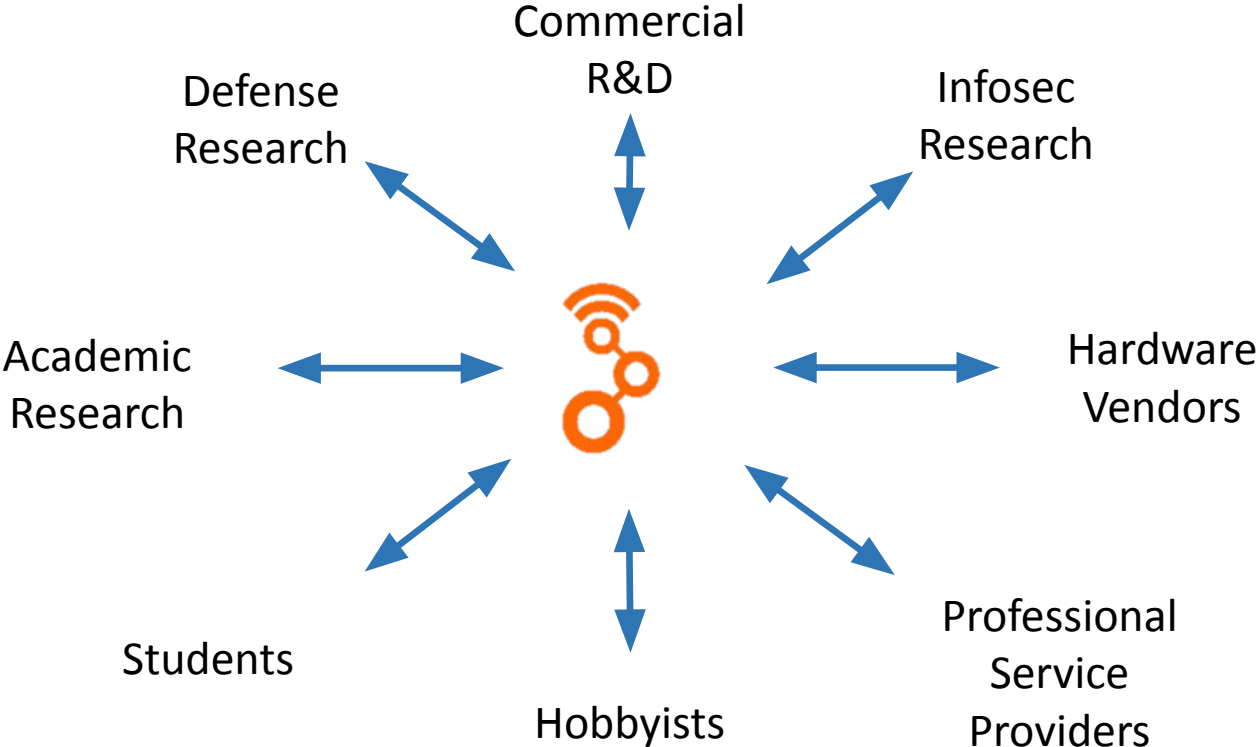
- GNU Radio can be extended with OOTs
- OOTs cover more specific functionality
- There is a large number available
- CGRAN is our central database



The screenshot shows the CGRAN Projects website. The header is orange with navigation links: CGRAN, Projects, Documentation, GNU Radio, and VOLK. Below the header is a large red circle containing a network diagram with nodes and connections. To the right of the diagram, the text reads: "The Comprehensive GNU Radio Archive Network" followed by a paragraph: "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." Below this text is a red circular button with a white downward arrow. Underneath the button is the text "Browse~Checkout~Hack". At the bottom of the page, there is a search bar and a table of projects.

Name	Tags	Description	Repository
gr-eventstream	scheduler, streams, bursty	The event stream scheduler	Github

GNU Radio is used by



GNU Radio is an Ecosystem

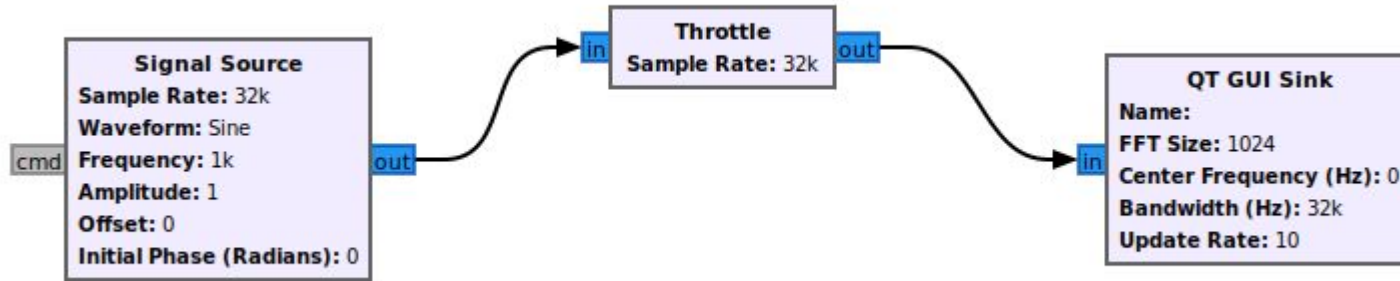
- Active Open Source community since 2001
- PyBombs, OOTs
- GRCon since 2011
- GNU Radio Foundation
- FOSDEM SDR DevRoom
- GSoC, SoCIS, R&S Competition, SDR Academy
- GNU Radio Europe



Outline

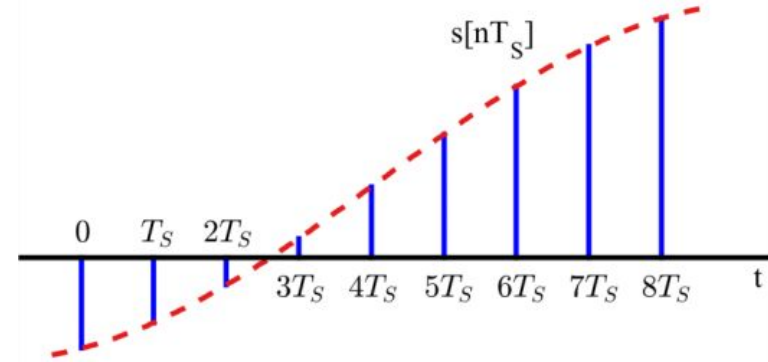
- About me
- About tutorial
- GNU Radio
- **Lab: Digital Signal Processing (DSP)**
- Lab: Software Defined Radio (SDR)
- Lab: Wide Band Frequency Modulation (WBFM)

Exploration of Signals in Frequency Domain



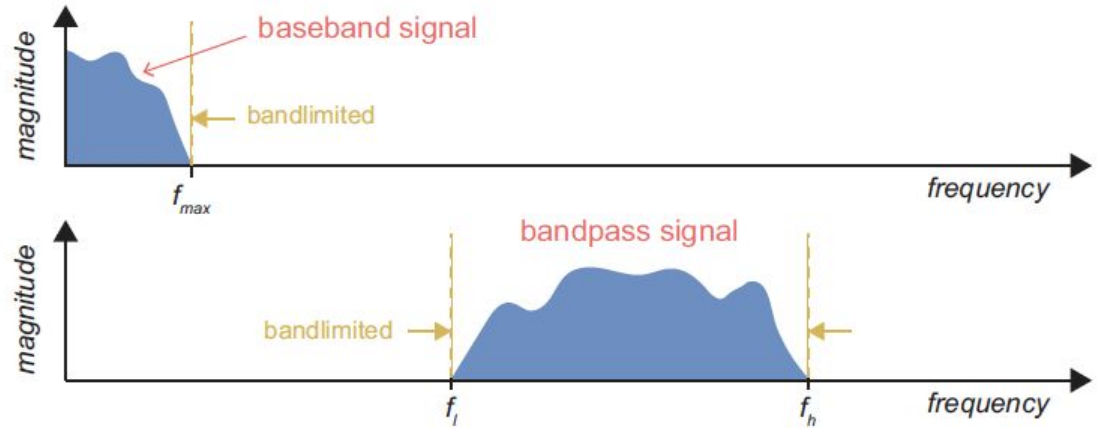
Sampling

- Communication signals are continuous-time
- We (ADCs) take samples at regular times
- T_s is sampling period
- F_s is sampling frequency



Baseband & Bandpass

- Baseband: Information signal
- Bandpass: Communication signal

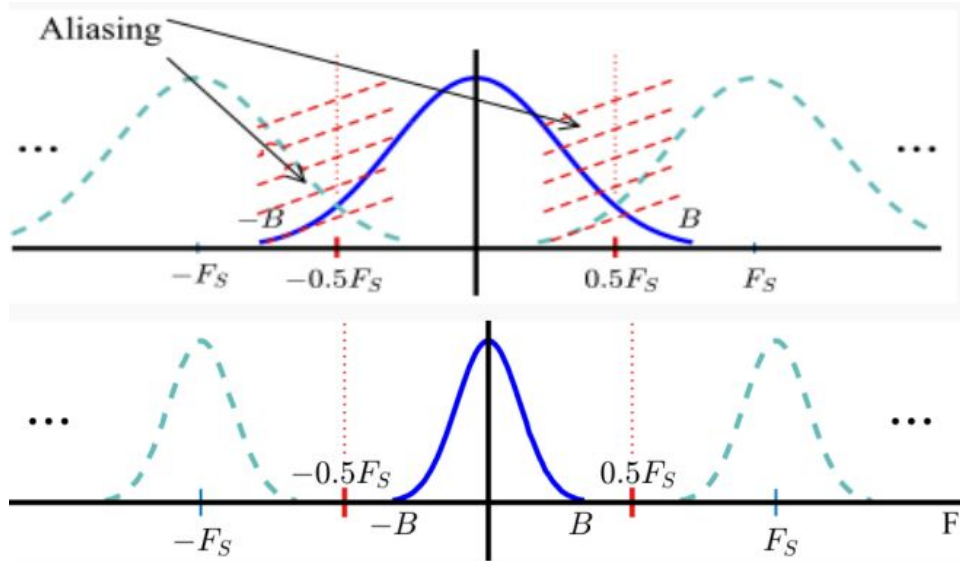


Nyquist Sampling Theorem

- The **Nyquist Sampling Theorem** states that a baseband, bandlimited signal must be sampled at **greater than twice the bandwidth** present in the signal, i.e.
 - $f_s > 2 * f_{max}$
 - $f_s > 2 * (f_{high} - f_{low})$

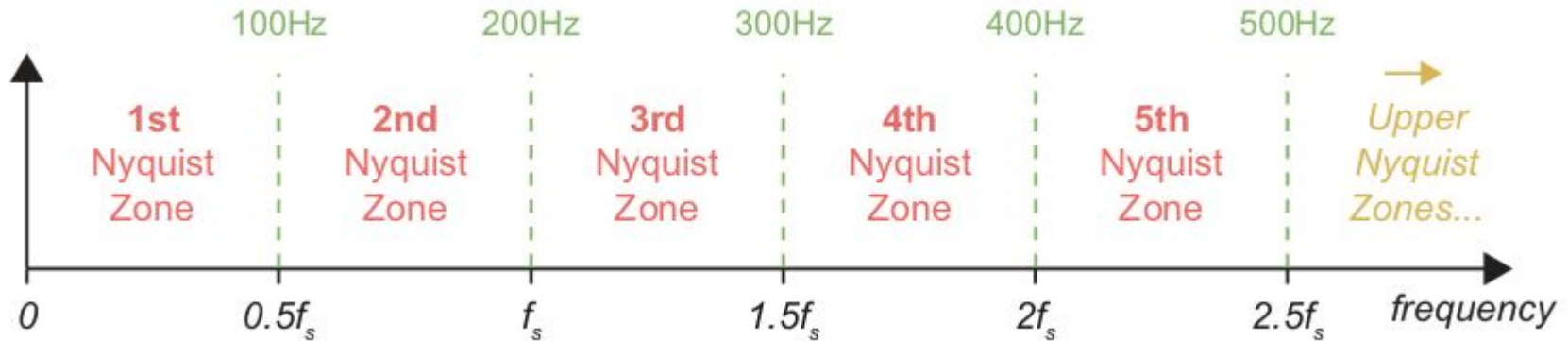
Aliasing

- Sampling produces aliases (spectral replicas)
- To prevent aliasing F_s must satisfy $F_s > 2 * BW$

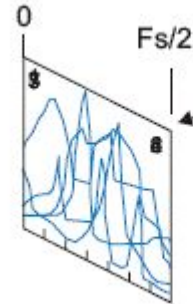
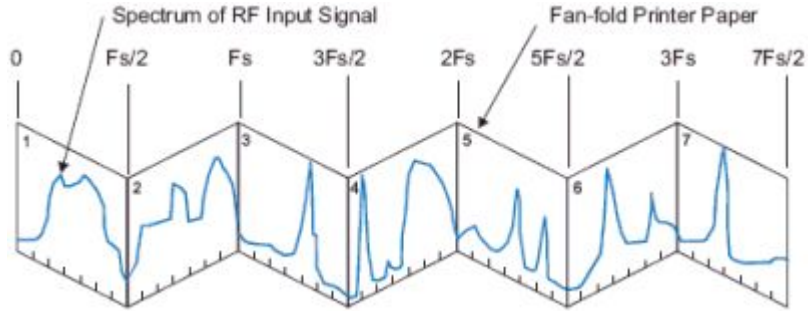


Nyquist Zones

- Partitions of bandwidth $0.5f_s$ in the frequency domain
- Any signal components present in higher Nyquist Zones are 'folded' down into the 1st Nyquist Zone as a result of aliasing



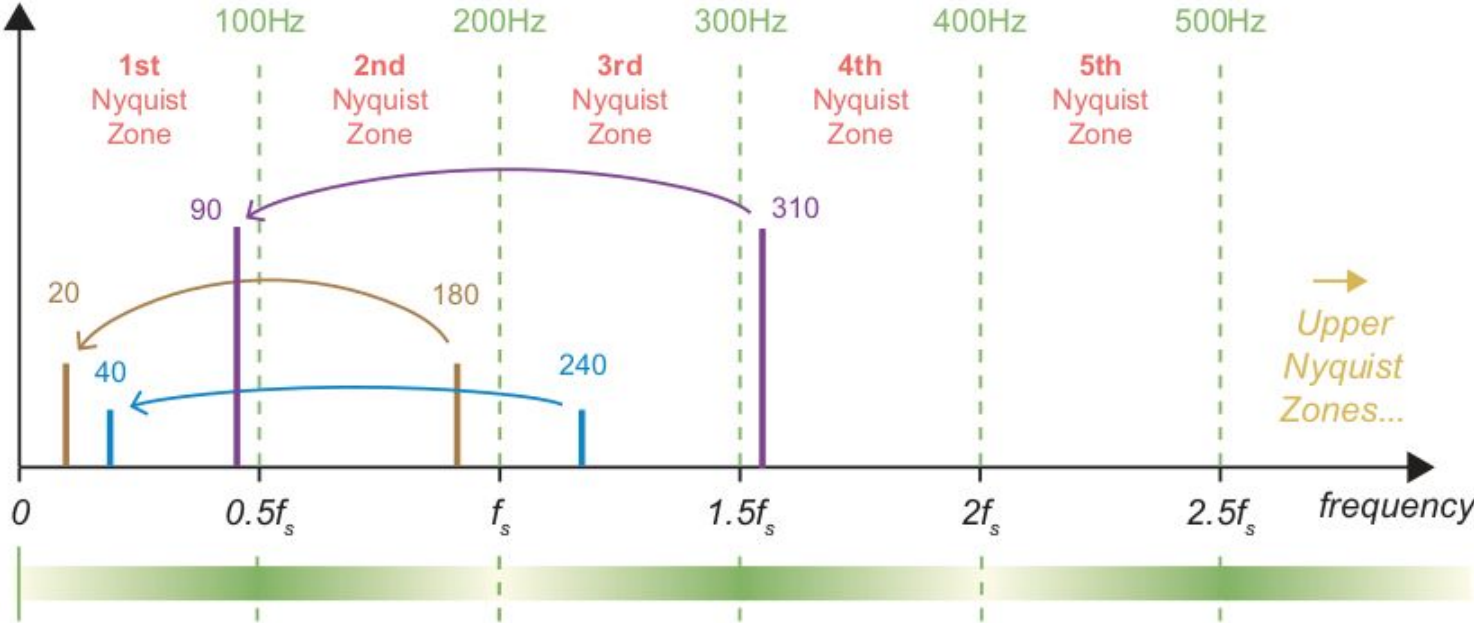
Folded Spectrum View



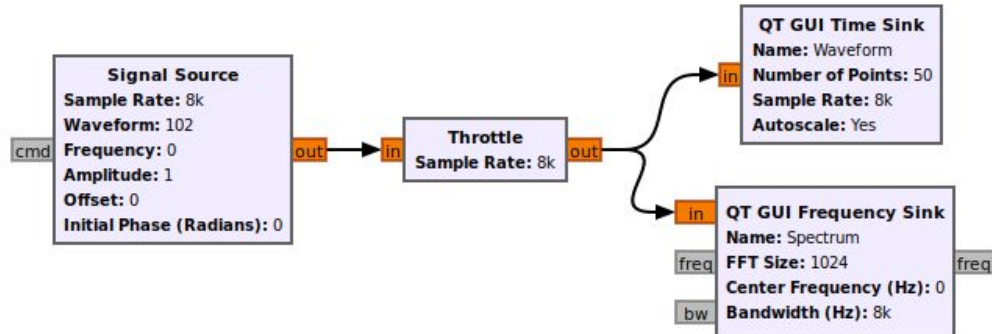
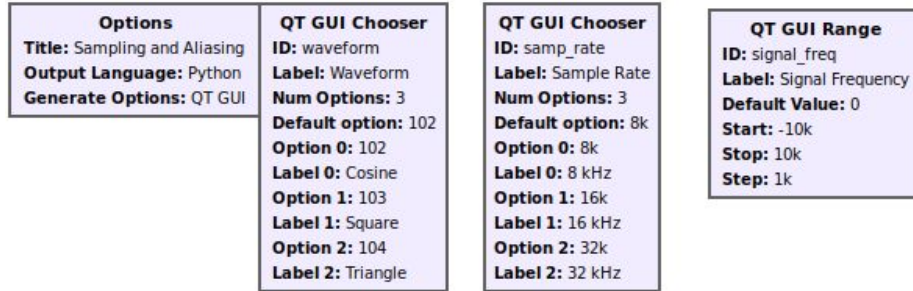
After sampling, all out of band signals and noise are folded into the band between 0 and $F_s/2$



Examples of aliasing with reference to Nyquist Zones

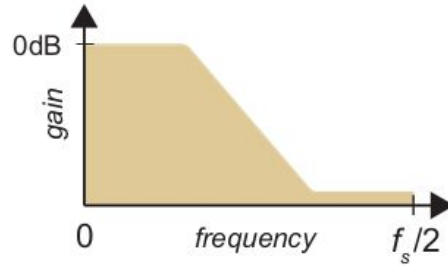


Sampling and Aliasing

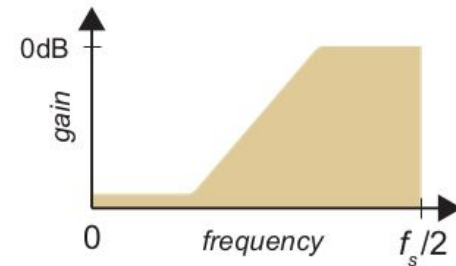


Digital Filters

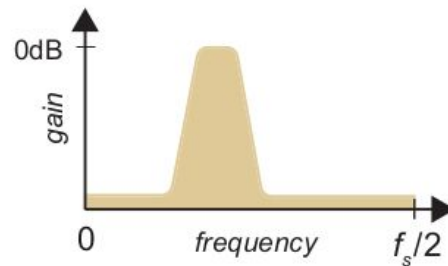
- A filter modifies the frequency contents of an input signal
- Types
 - LPF
 - HPF
 - BPF
 - Notch



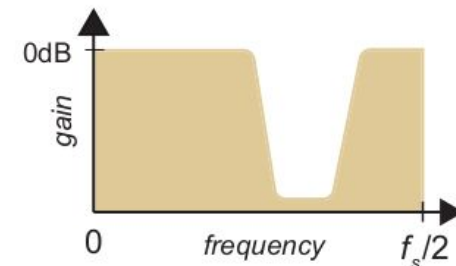
(a) lowpass



(b) highpass

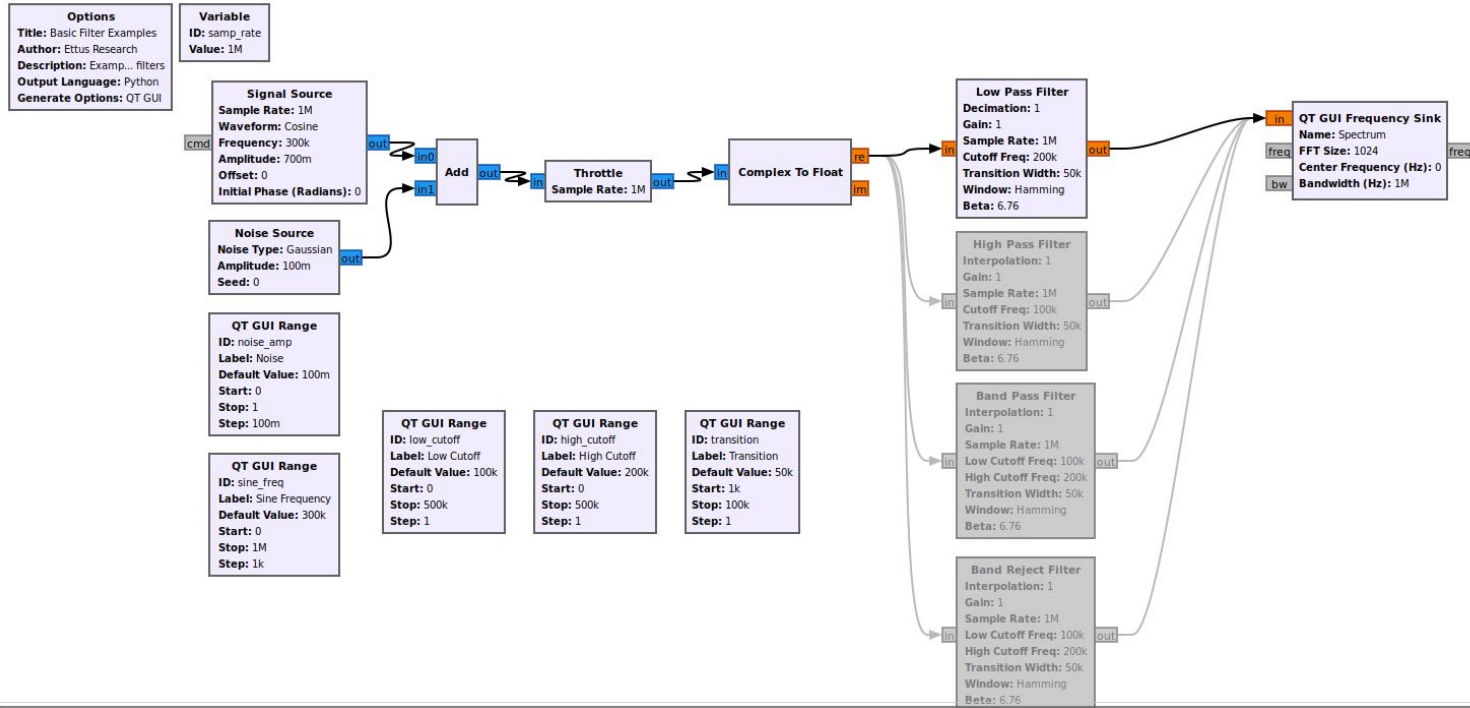


(c) bandpass



(d) bandstop

Filters Using GNU Radio

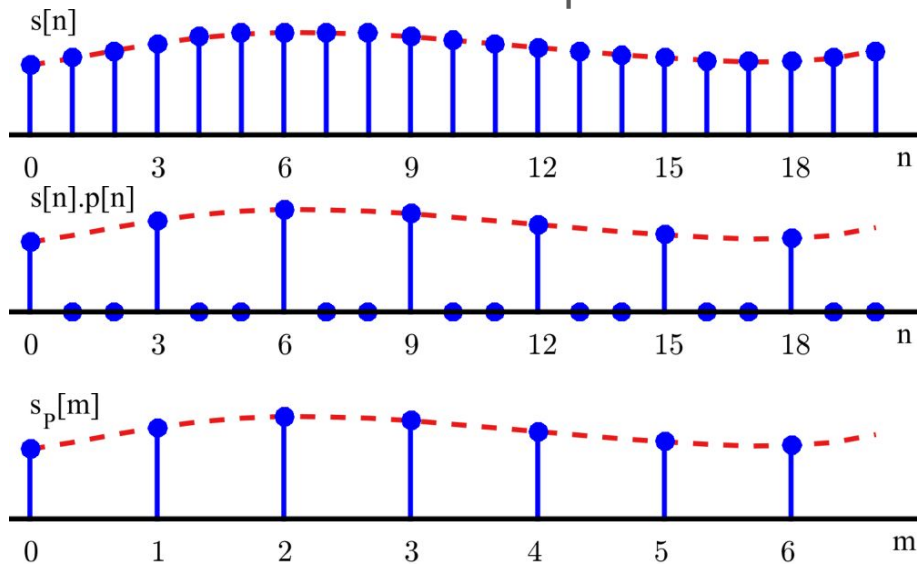


Multirate Signal Processing

- Multirate operations are required to change the sampling rate in a DSP system to optimise computational efficiency
- Some example scenarios
 - To match the sampling rates of two signal paths that will be combined
 - To adjust the sampling rate closer to Nyquist when the signal bandwidth changes
 - To match the sampling rate of an external interface, such as a DAC
 - To ease analogue anti-alias or image-rejection filter requirements

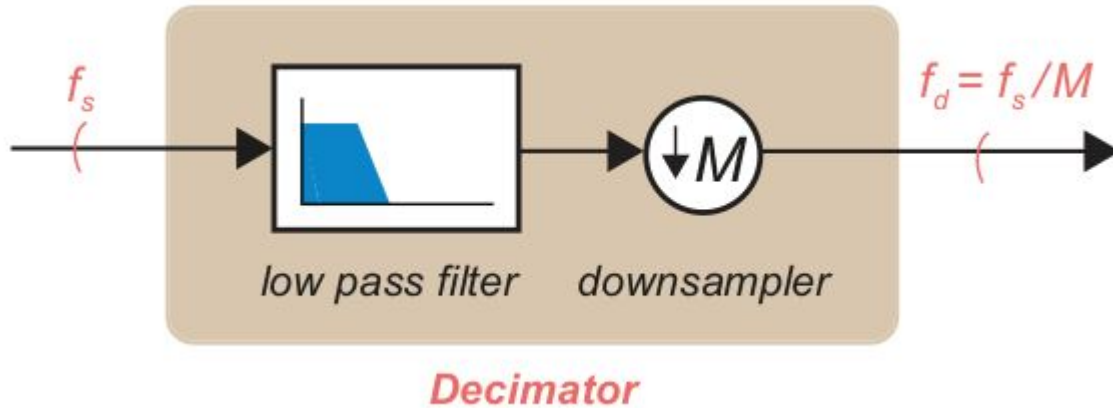
Decimation

- Reducing the sample rate by an integer factor
- Retain every P th sample and discard the remaining samples
- The new slower sample rate is $1/P$ of the original faster sample rate



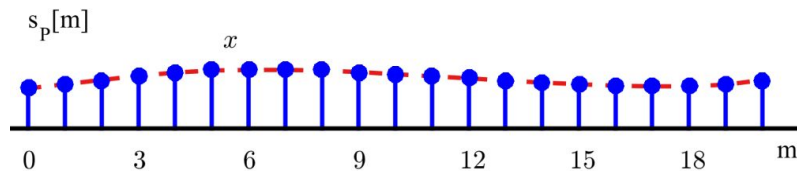
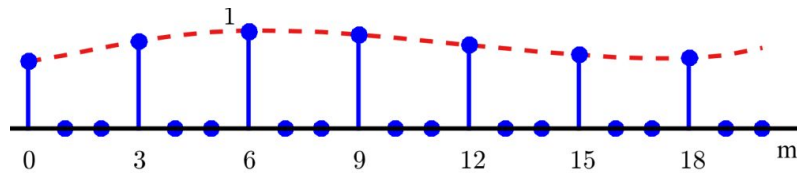
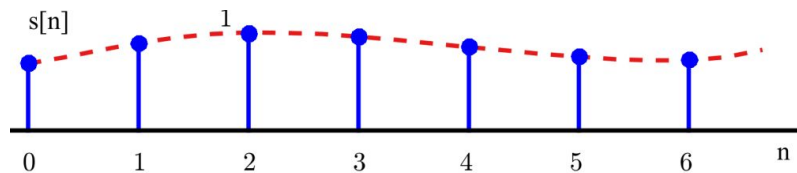
Decimation

- Decimation involves two processes:
 - anti-alias low pass filtering, followed by
 - downsampling



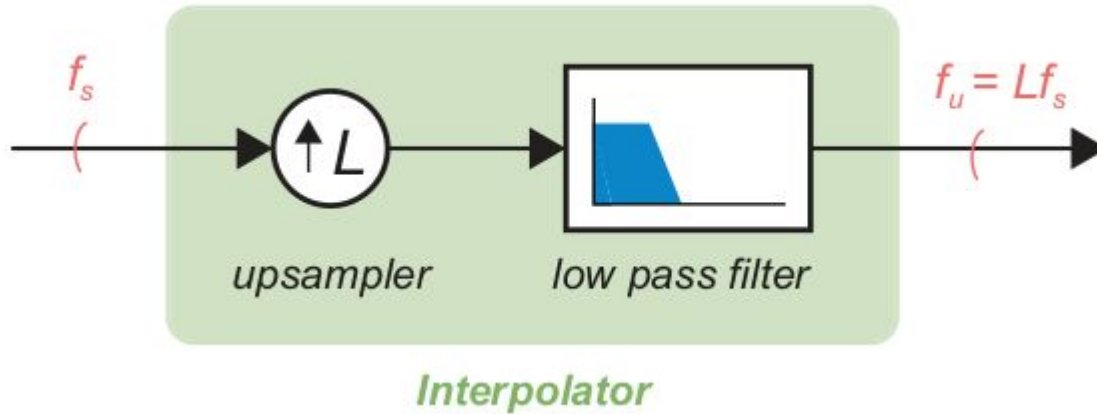
Interpolation

- Increasing the sample rate by an integer factor
- Insert $P - 1$ zeros between the original input samples and interpolate
- The new faster sample rate is P times the original slower sample rate



Interpolation

- An interpolator is composed of
 - an upsampling operation, followed by
 - a low pass image rejection filter

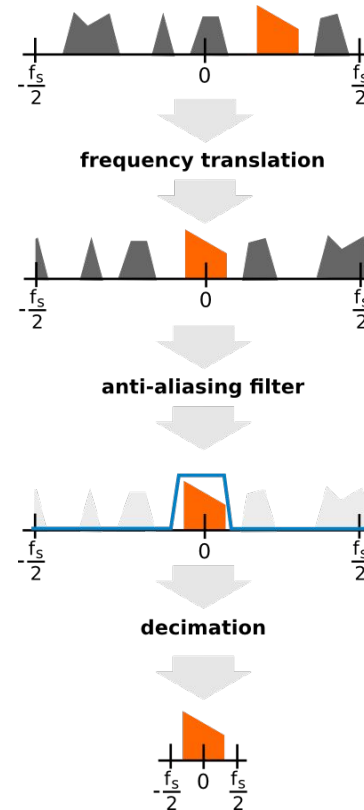


Other Multirate Operations

- There are other types of operation to be aware of, beyond simple decimation and interpolation by integer factors
- Resampling a signal by a **rational fraction**
 - If the sampling rate is to be changed by the ratio of two integers, e.g. a rate change from 100 MHz to 150 MHz could be expressed as $R = 3 / 2$. Rational fractional rate changes can be achieved using a **cascade** of an interpolator and decimator, e.g. $L = 3$ and $M = 2$ in this example. The resulting structure can be optimised using polyphase methods.
- Resampling a signal by an **irrational fraction**, or by a factor that changes over time
 - Where there is no convenient integer-based expression for the resampling ratio, or where it is dynamic, a different type of approach is required. Popular methods include highly oversampled polyphase filters, and Farrow structures.

Frequency Xlating FIR Filter

- Frequency Xlating FIR Filter is a block that:
 - performs **frequency translation** on the signal,
 - **downsamples** the signal by running a **decimating FIR filter** on it.
- It can be used as a **channelizer**:
 - it can select a narrow bandwidth channel from the wideband receiver input.



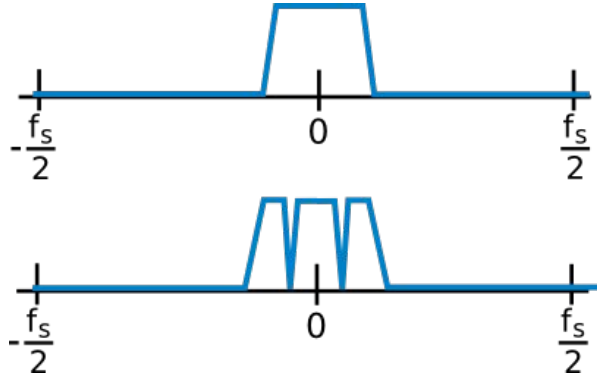
Suppose this is the stations in FM radio example!

Our aim is to select only one channel

Frequency Xlating FIR Filter

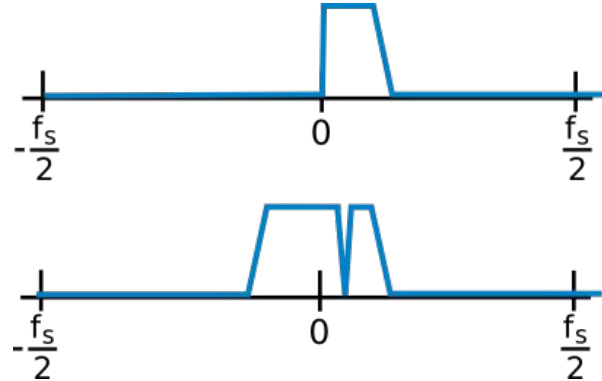
- If you have Real taps, then your FIR filter will be symmetric in the frequency domain.

```
firdes.low_pass(1, samp_rate, samp_rate/(2*decimation), transition_bw)
```



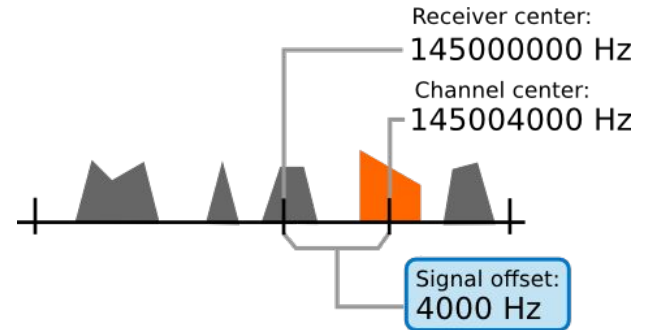
- If you have Complex taps, then your FIR filter will not have to be symmetric in the frequency domain.

```
firdes.complex_band_pass(1, samp_rate, -samp_rate/(2*decimation), samp_rate/(2*decimation), transition_bw)
```



Frequency Xlating FIR Filter

- **Decimation**: the integer ratio between the input and the output signal's sampling rate.
- Example:
 - Input sample rate = 240000
 - Decimation factor = 5
 - Output sample rate = $240000 \div 5 = 48000$
- **Center frequency**: the frequency translation offset frequency.
- In practice, it is the frequency offset of the signal of interest to be selected from the input.



Frequency Xlating FIR Filter

Options
Title: Frequenc...FIR Example
Author: Ettus Research
Description: Examp...R Filter
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 1M

QT GUI Range
ID: noise_amp
Label: Noise
Default Value: 100m
Start: 0
Stop: 1
Step: 100m

QT GUI Range
ID: sine_freq
Label: Sine Frequency
Default Value: 300k
Start: 0
Stop: 1M
Step: 1k

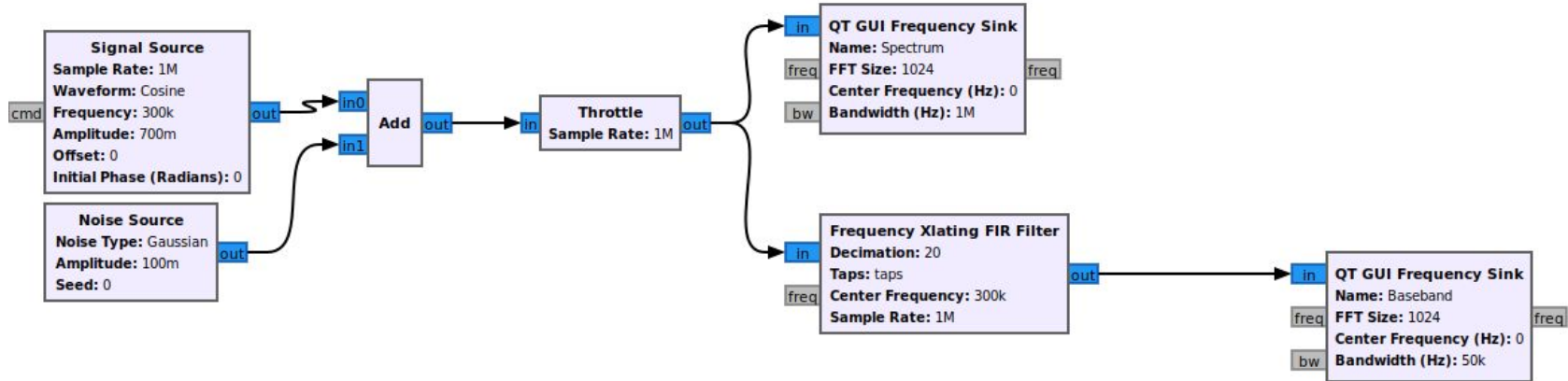
QT GUI Range
ID: xlate_freq
Label: Xlate Frequency
Default Value: 300k
Start: -500k
Stop: 500k
Step: 1

QT GUI Range
ID: xlate_bw
Label: Xlate Bandwidth
Default Value: 10k
Start: 0
Stop: 50k
Step: 1

Variable
ID: bb_rate
Value: 50k

Variable
ID: decimation
Value: 20

Variable
ID: taps
Value: firdes.low_pass(1.0...



LabDSP.ipynb

The screenshot shows a JupyterLab environment. On the left is a file browser for the directory `/ LabDSP /`. It contains a table of files:

Name	Last Modified
BandPassPropertie...	4 months ago
CaptureStereoTon...	4 months ago
FilterLengths.png	4 months ago
lab_dsp_noise_ba...	4 months ago
lab_dsp_noise_ba...	4 months ago
lab_dsp_tone_deci...	4 months ago
lab_dsp_tone_deci...	4 months ago
lab_dsp_tone_deci...	4 months ago
lab_dsp_tone_deci...	4 months ago
lab_dsp_tone_deci...	4 months ago
lab_dsp_tone_deci...	4 months ago
lab_dsp_tone_deci...	4 months ago
lab_dsp_tone_deci...	4 months ago
lab_dsp_tone_ster...	4 months ago
LabDSP.ipynb	24 minutes ago
pentek_folded_spe...	4 months ago
soundcontrol.png	4 months ago

The main notebook editor displays the following content:

LabDSP: Hands-on DSP using GNU Radio and Sound Card ¶

GNU Radio

GNU Radio is software radio toolkit that is free and open-source (**FOSS**). It has many DSP blocks which are mostly written in **C++**. You can also write blocks in **Python** though their execution will be slower. GNU Radio also has a built-in **scheduler** to provide blocks with data they need. Data flow is handled by scheduler. Each block has a specific function.

GNU Radio has no GUI but it comes with an application called **GNURadio Companion (GRC)** so you can add and connect blocks together to build what is called a **flowgraph**. You can find a block by searching from available blocks. After you find your block you can add blocks either by double clicking on it or dragging and dropping it onto the surface of GRC. You can also connect blocks in a Python script if you wish. Actually this is what GRC does! It generates a script file to do all connections and start the flowgraph. Every flowgraph starts with a **source** block and terminates with a **sink** block.

Task1: Characterizing a real bandpass filter

Below is the first flowgraph we will create. In this example we will examine filter characteristics of a bandpass filter. Filters are fundamental building blocks of DSP. Our flowgraph has one source and two sink blocks. We feed the

At the bottom of the interface, the status bar shows: `Simple` (toggle), `0` (CPU usage), `1` (memory usage), `Python 3 (ipykernel) | Idle`, `Mode: Command`, `Ln 1, Col 1`, and `LabDSP.ipynb`.

Outline

- About me
- About tutorial
- GNU Radio
- Lab: Digital Signal Processing (DSP)
- **Lab: Software Defined Radio (SDR)**
- Lab: Wide Band Frequency Modulation (WBFM)

What is Software Defined Radio (SDR)?

“A radio in which aspects of functionality are implemented in, or controlled by, software.”

- Flexible functionality
 - the operation of a radio can be changed without making any physical alterations to the device
- Algorithms from DSP and communications theory running as real-time software on a CPU, GPU and/or FPGA
- Joe Mitola first coined the term in 1991

Why SDR?

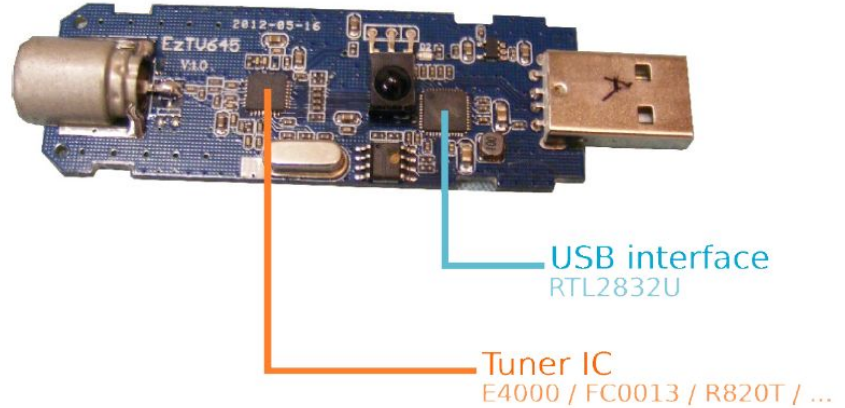
- Traditional radios are hard-wired to specific frequency bands and communication protocols
 - Fixed-function, Black Box
 - Can't be easily modified, can't easily access internal values and states
- SDR provides:
 - Flexibility
 - Upgradability
 - Reconfigurability
 - Lower Cost

Key SDR Parameters (Features)

- Frequency (Tuning) Range
- Instantaneous Bandwidth
- Bit resolution
- Interface (USB, Ethernet, PCIe)
- Rx/Tx, half-duplex, full-duplex, MIMO
- Preselectors
- Budget: 50\$-...k\$

RTL-SDR

- “I smell a very cheap poor man’s SDR here 😊”
- Cheap man’s radio since 2012
- Hams, DIY, hackers, makers, students,...
- Demodulator
 - Named by RTL2832U chip, DVB-T
- Tuner
 - **R820T**: 24-1766MHz
 - **E4000**: 52-2200MHz

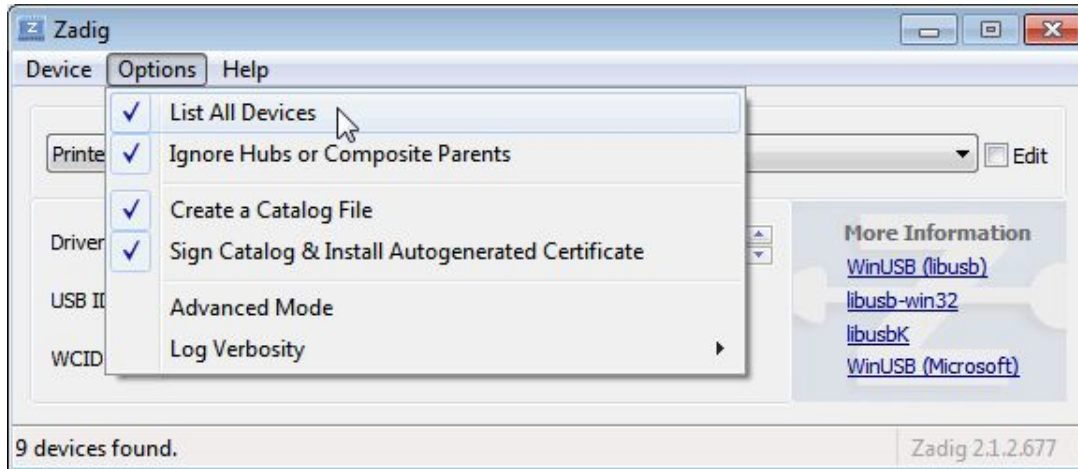


RTL-SDR

- Receive-only
- 8-bit ADC
- 24MHz-1.75GHz (depends on tuner chip)
- 2.4MSPS BW (stable) upto 3.2M
- “*HamItUp*” upconverter
 - HF coverage

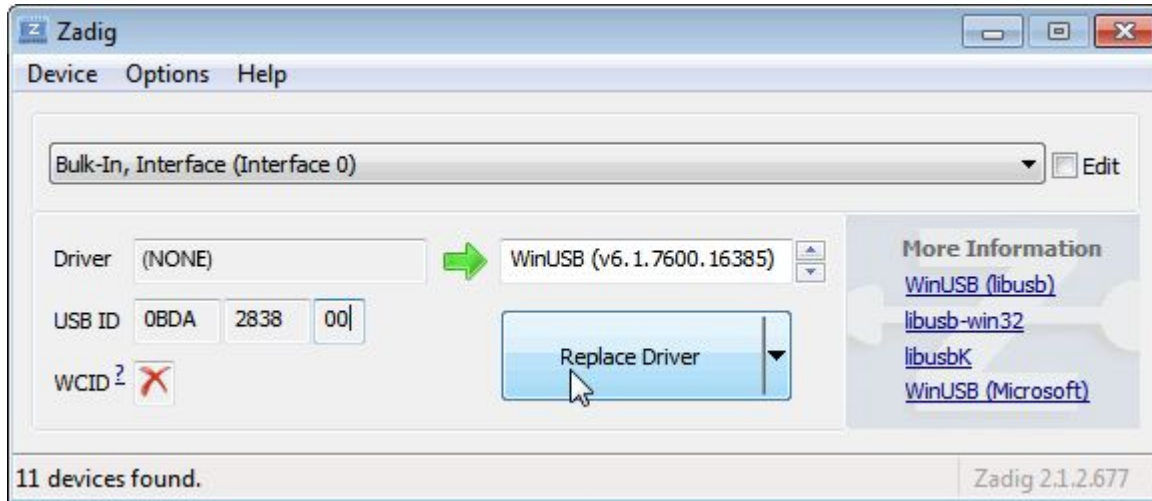
RTL-SDR Driver Installation#1 - Windows

- Plug in your dongle
- Right click zadig.exe file and select "Run as administrator".
- In Zadig, go to "Options->List All Devices" and make sure this option is checked. If you are using Windows 10 or 11, in some cases you may need to also uncheck "Ignore Hubs or Composite Parents".



RTL-SDR Driver Installation#2 - Windows

- Select "Bulk-In, Interface (Interface 0)" from the drop down list. Make sure it is Interface 0 (ZERO), and not "1".



RTL-SDR Driver Installation - Linux

- Linux users may blacklist RTL so that default DVB-T driver is not loaded when dongle is plugged in.
 - `# cd /etc/modprobe.d/`
 - `# sudo gedit blacklist-rtl.conf`
 - `# append: blacklist dvb_usb_rtl28xxu`
 - OR
 - `# echo "blacklist dvb_usb_rtl28xxu" >> /etc/modprobe.d/blacklist.conf`

LabSDR.ipynb

The screenshot shows a JupyterLab interface. On the left is a file browser with a search bar and a table of files. The main area displays a notebook cell with the following content:

▼ LabSDR: Introduction to Software Defined Radio (SDR)

▼ What is an SDR? ¶

Software Defined Radio (SDR) is a radio communication system where components that have been traditionally implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are instead implemented by means of software on a personal computer or embedded system [wikipedia](#). Computation platform can be anything from general purpose **CPUs to FPGAs, from GPUs to DSP** chips.

With SDR one can access some part of electromagnetic spectrum, monitor, capture, demodulate it. Width of the RF signal depends on the capability of RF front-end. Another important factor is resolution.

SDR has been an important tool in education, industry for years. It will provide us over-the-air signals easily so that we can make sure our DSP algorithms behave the same as they do in the simulation.

In this lab we will learn about our SDR hardware and look into SDR tools giving us access to electromagnetic spectrum.

RTL-SDR

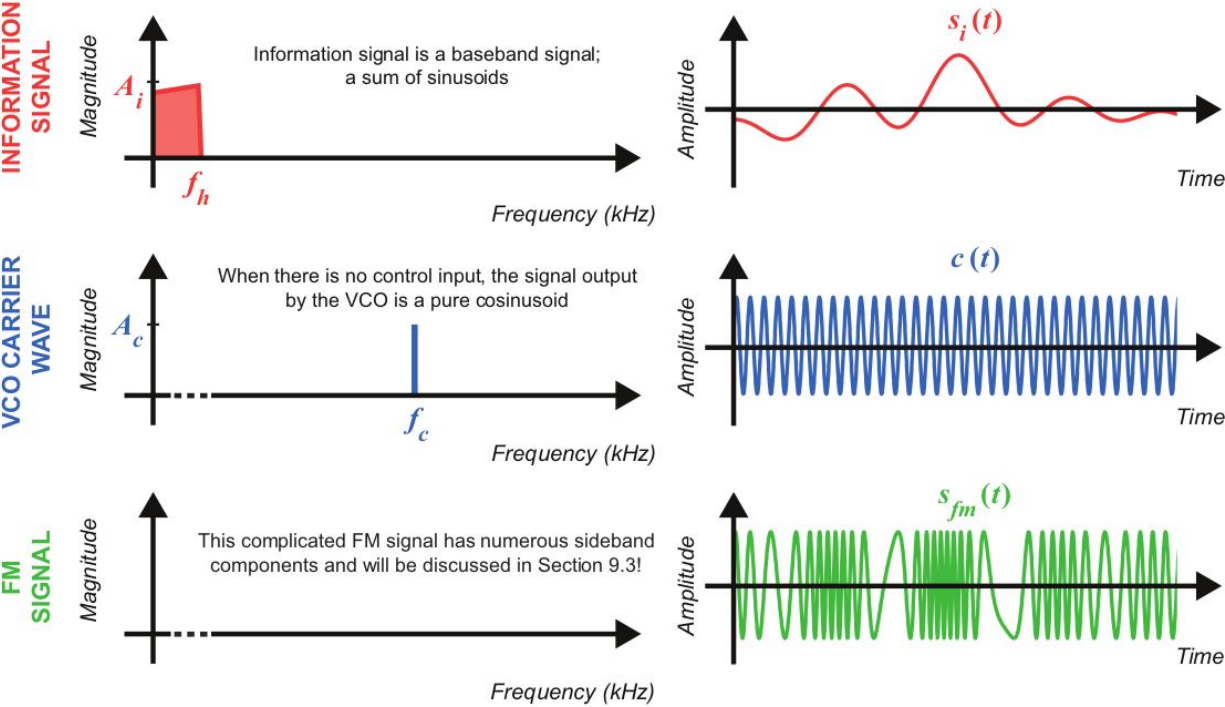
Previously in DSP lab we have only used our PC with sound-card to implement some basic DSP functionalities.

The interface also shows a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help), a toolbar, and a status bar at the bottom indicating the current mode (Command) and location (Ln 1, Col 1).

Outline

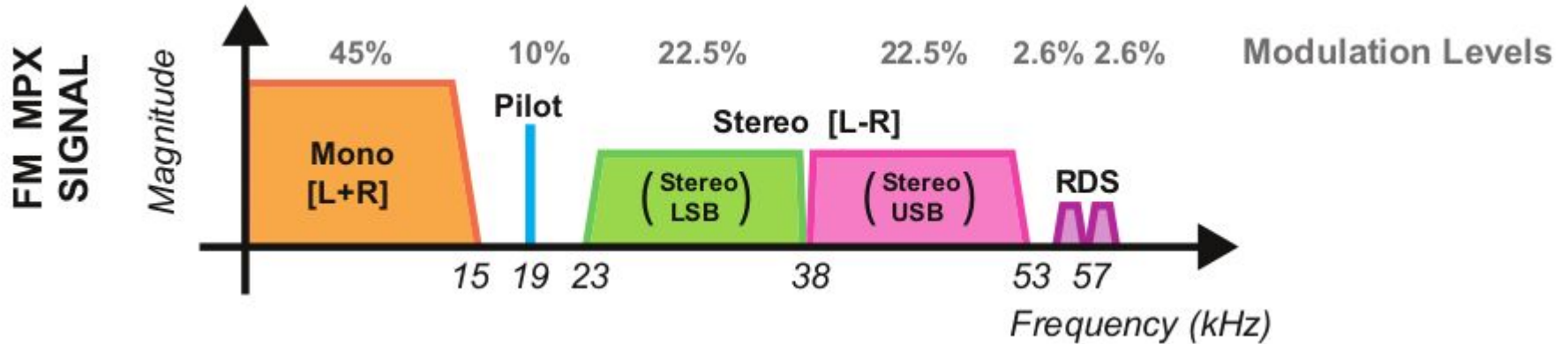
- About me
- About tutorial
- GNU Radio
- Lab: Digital Signal Processing (DSP)
- Lab: Software Defined Radio (SDR)
- **Lab: Wide Band Frequency Modulation (WBFM)**

Frequency Modulation (FM)

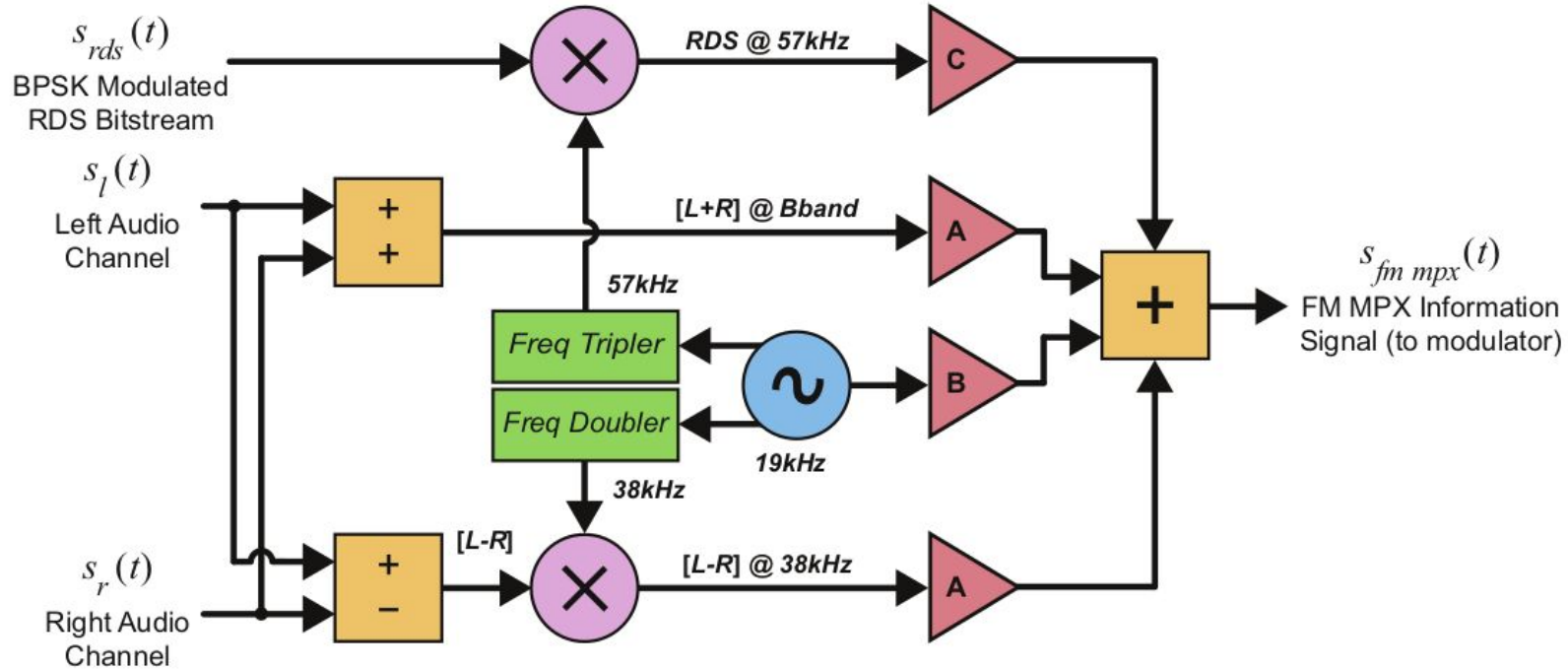


FM Radio Multiplex

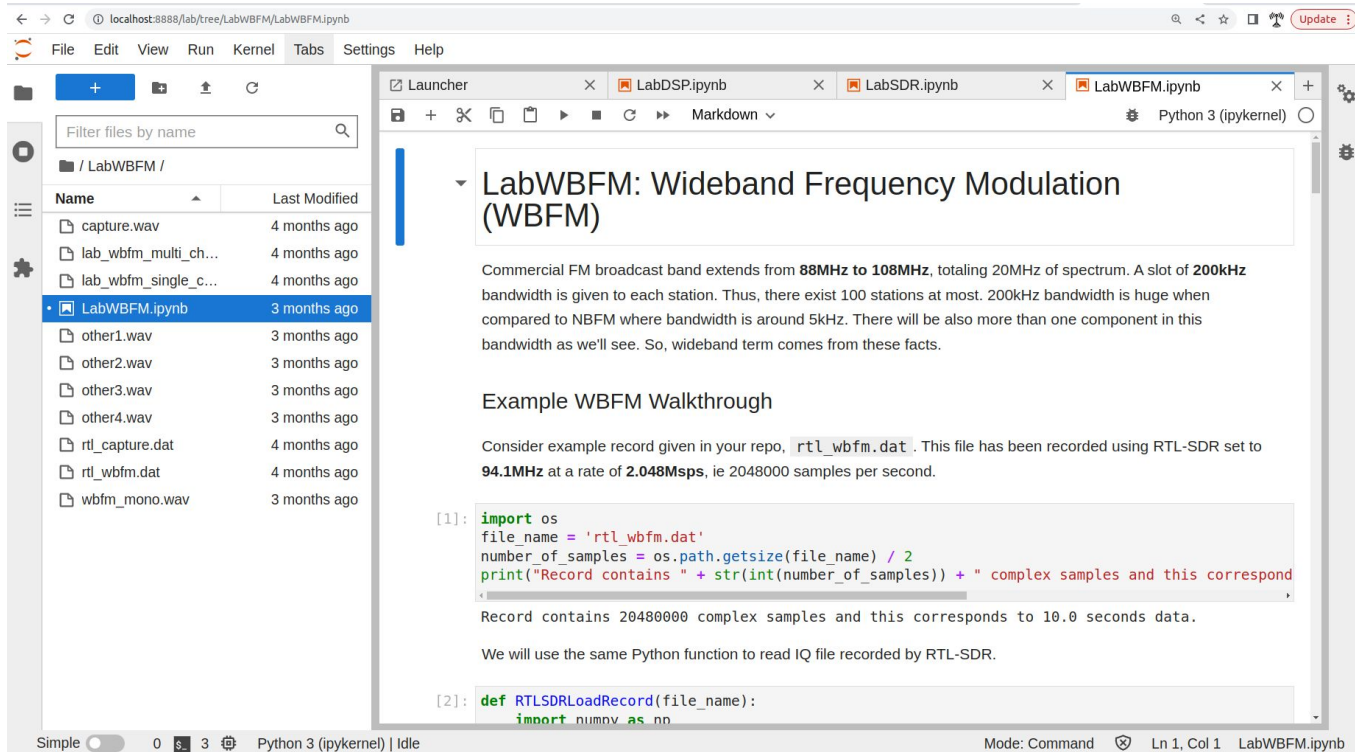
- It is common practice to multiplex multiple information signals together before performing modulation, as this allows for multi-channel transmission using one carrier.



Broadcast FM (WBFBM)



LabWBFM.ipynb



The screenshot shows a JupyterLab interface with a file browser on the left and a notebook cell on the right. The file browser shows a directory named 'LabWBFM' containing several files, including 'LabWBFM.ipynb'. The notebook cell contains the following text:

LabWBFM: Wideband Frequency Modulation (WBFM)

Commercial FM broadcast band extends from **88MHz to 108MHz**, totaling 20MHz of spectrum. A slot of **200kHz** bandwidth is given to each station. Thus, there exist 100 stations at most. 200kHz bandwidth is huge when compared to NBFM where bandwidth is around 5kHz. There will be also more than one component in this bandwidth as we'll see. So, wideband term comes from these facts.

Example WBFM Walkthrough

Consider example record given in your repo, `rtl_wbfm.dat`. This file has been recorded using RTL-SDR set to **94.1MHz** at a rate of **2.048Mps**, ie 2048000 samples per second.

```
[1]: import os
file_name = 'rtl_wbfm.dat'
number_of_samples = os.path.getsize(file_name) / 2
print("Record contains " + str(int(number_of_samples)) + " complex samples and this correspond
```

Record contains 2048000 complex samples and this corresponds to 10.0 seconds data.

We will use the same Python function to read IQ file recorded by RTL-SDR.

```
[2]: def RTLSDRloadRecord(file_name):
import numpy as np
```

At the bottom of the interface, the status bar shows 'Simple', '0', '3', 'Python 3 (ipykernel) | Idle', 'Mode: Command', 'Ln 1, Col 1', and 'LabWBFM.ipynb'.

Thanks!

murat-sever@live.com