Filters & Filtering

Tom Rondeau

(tom@trondeau.com)

2014-08-13

Filtering

"The need for filters intrudes on any thought experiment about the wonders of abundant information."

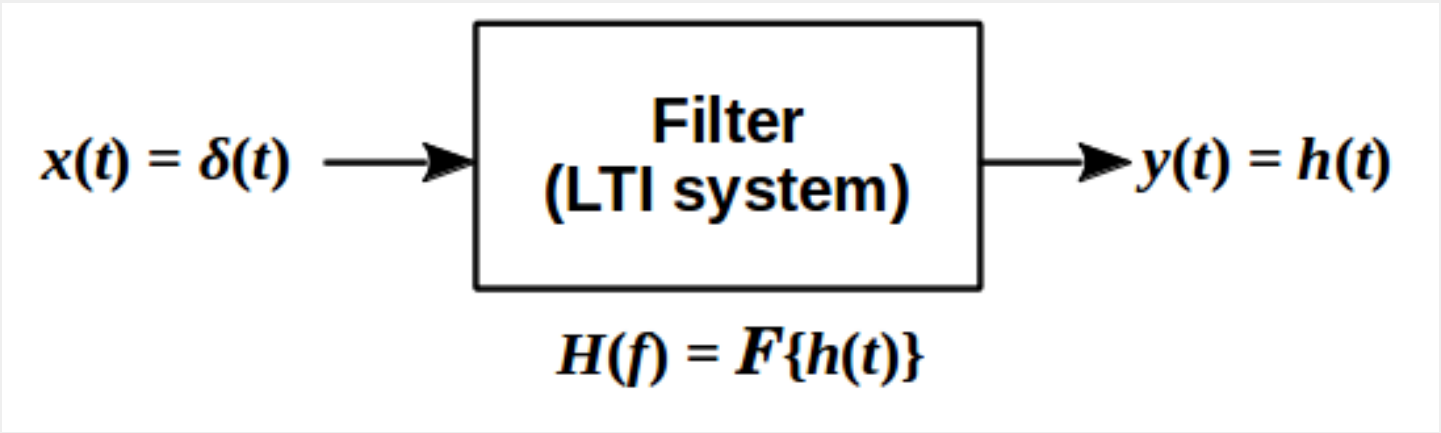
- James Gleik, *The Information*

Quick review of filtering (FIR filters)

Filtering = Convolution

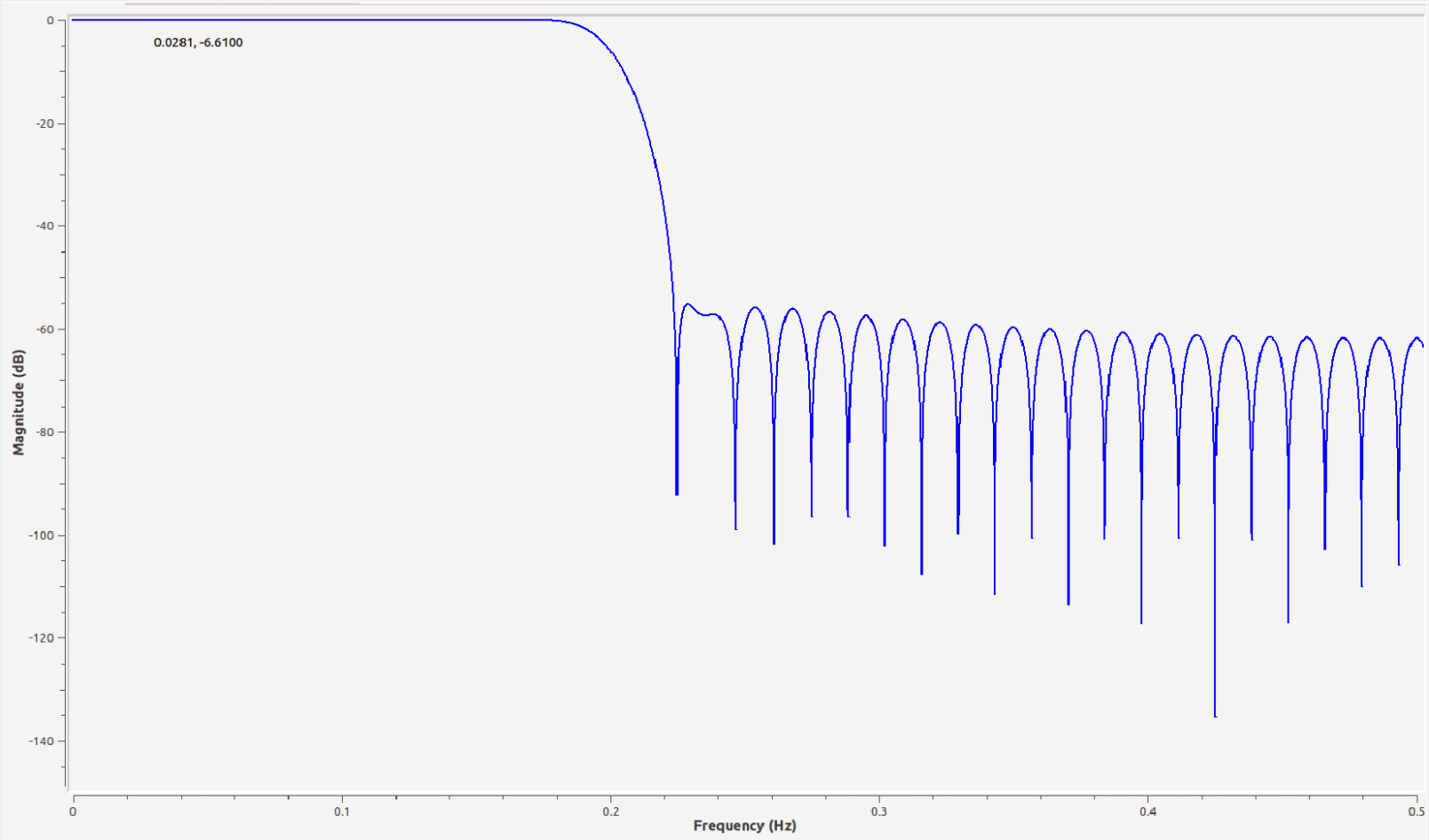
$$y[i] = \sum t_{N-n-1} x[i+n]$$

Filter Shapes



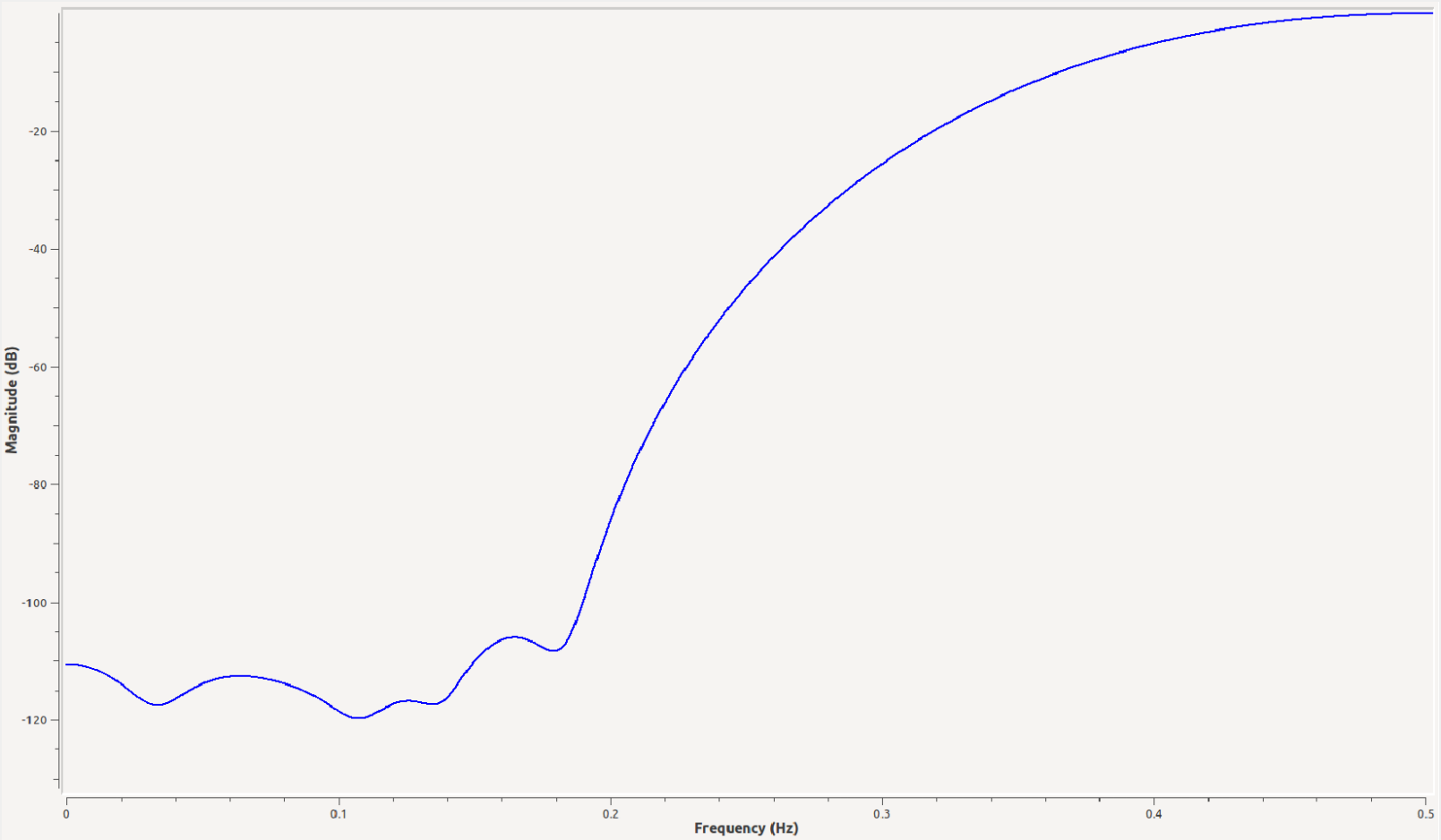
Low Pass Filter

filter.firdes.low_pass(1, 1, 0.2, 0.05, 80)



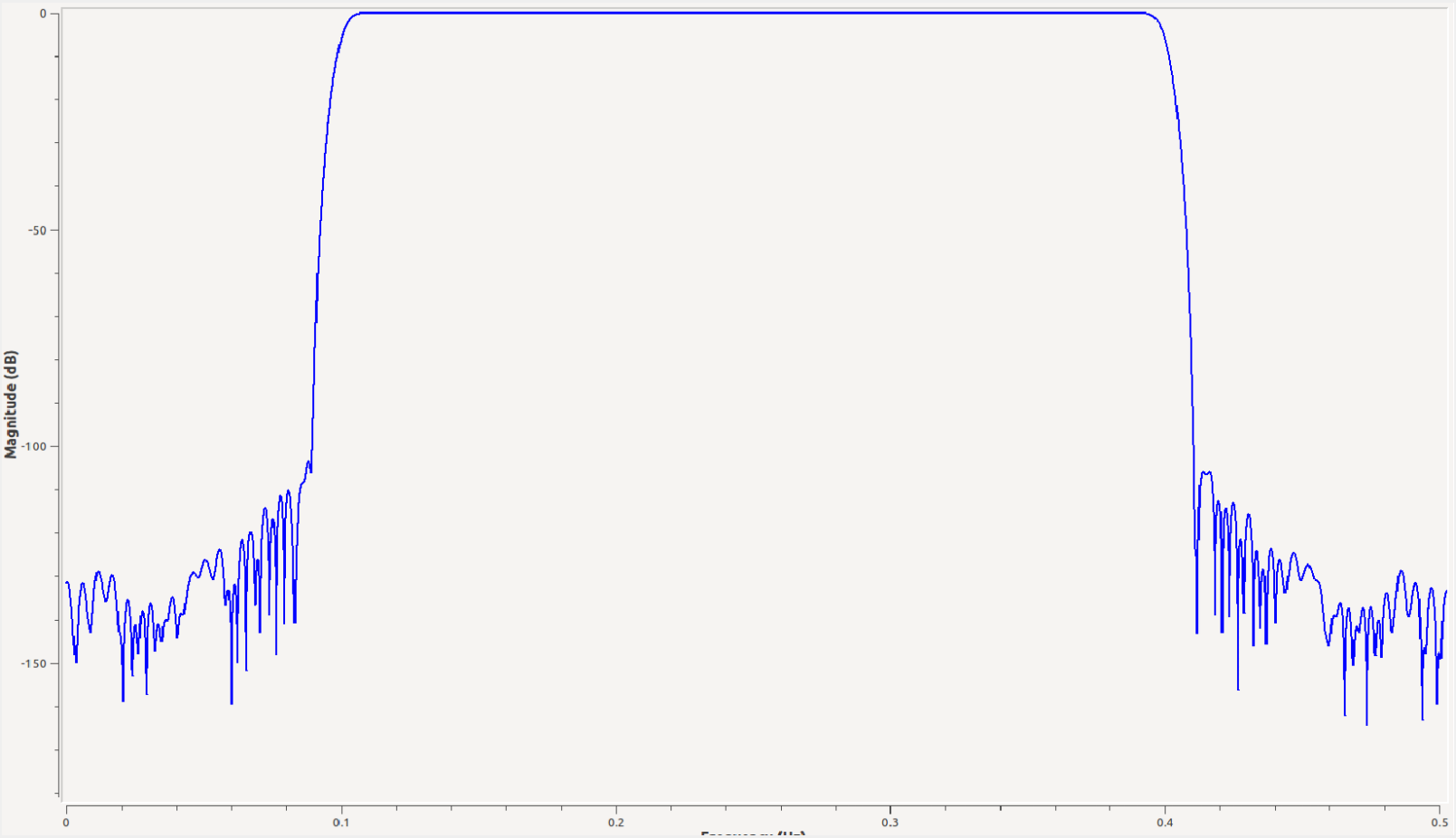
High Pass Filter

filter.firdes.high_pass_2(1, 1, 0.4, 0.2, 80)

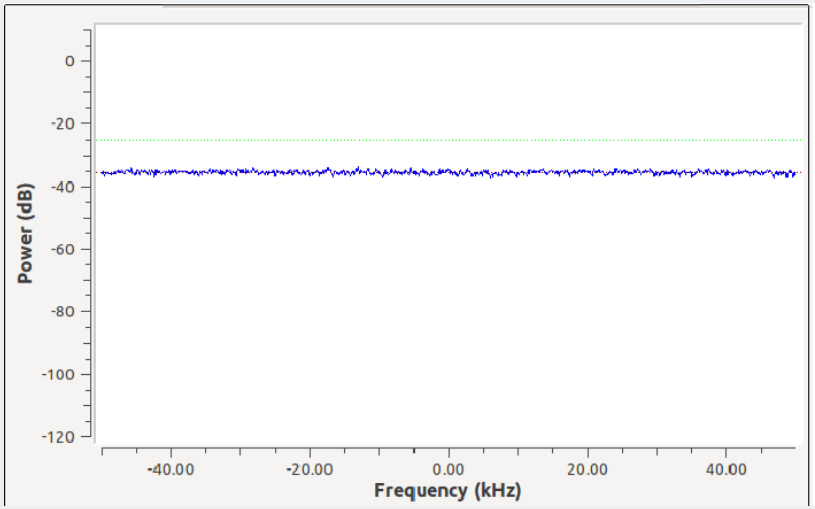


Band Pass Filter

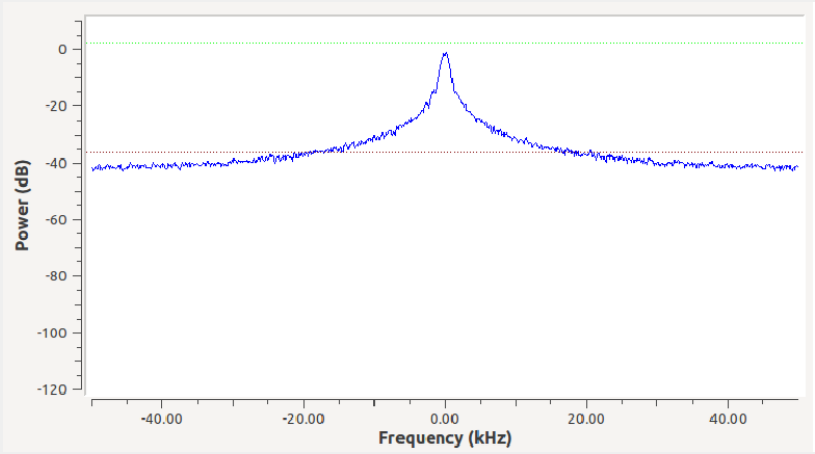
filter.firdes.complex_band_pass_2(1, 1, 0.1, 0.4, 0.01, 80)



Arbitrary Filter (taps = [1, 100])



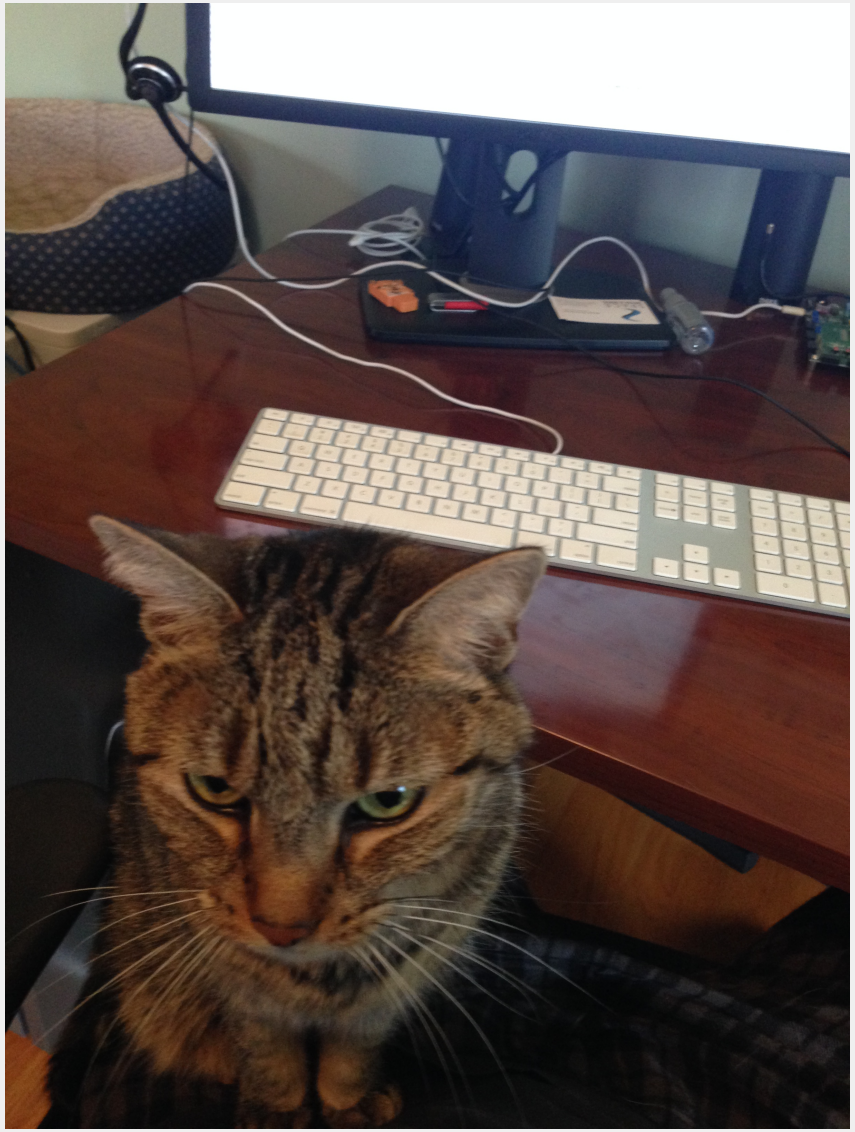
Input signal (white noise)



Shaped output signal

Sample Rates

Not managing your sample rates correctly? Fox is not impressed.

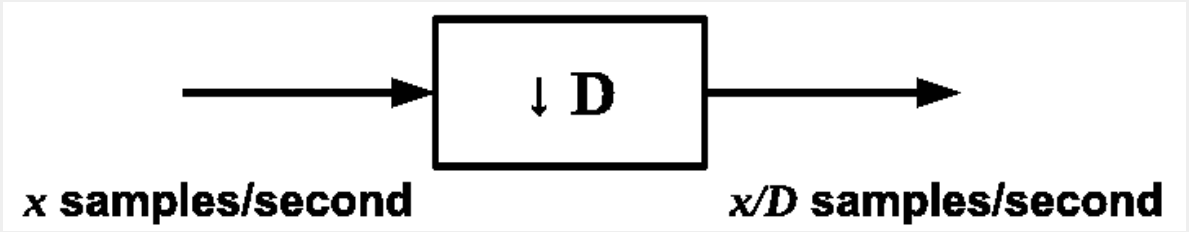


Sample Rate Changes

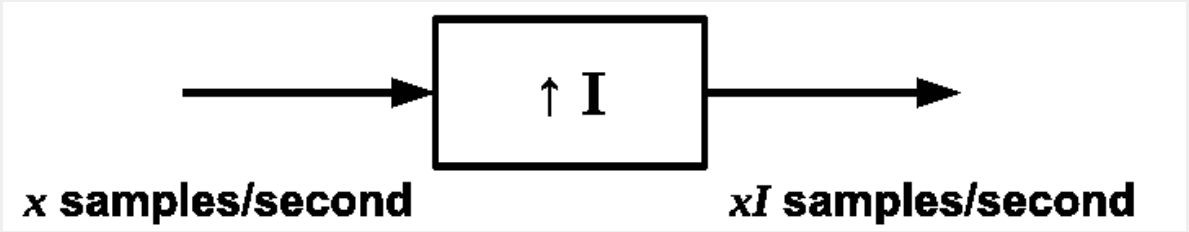
- Computational Demands are proportional to sampling rate.
  - In a filter, the number of multiply and adds per second.
- Nyquist says we only need $>$ 2x the bandwidth, so we want to keep as close to this as possible.
  - More is wasteful.
  - Less loses information.

Rate change is fundamental to SDR

Downsampling (Decimation)

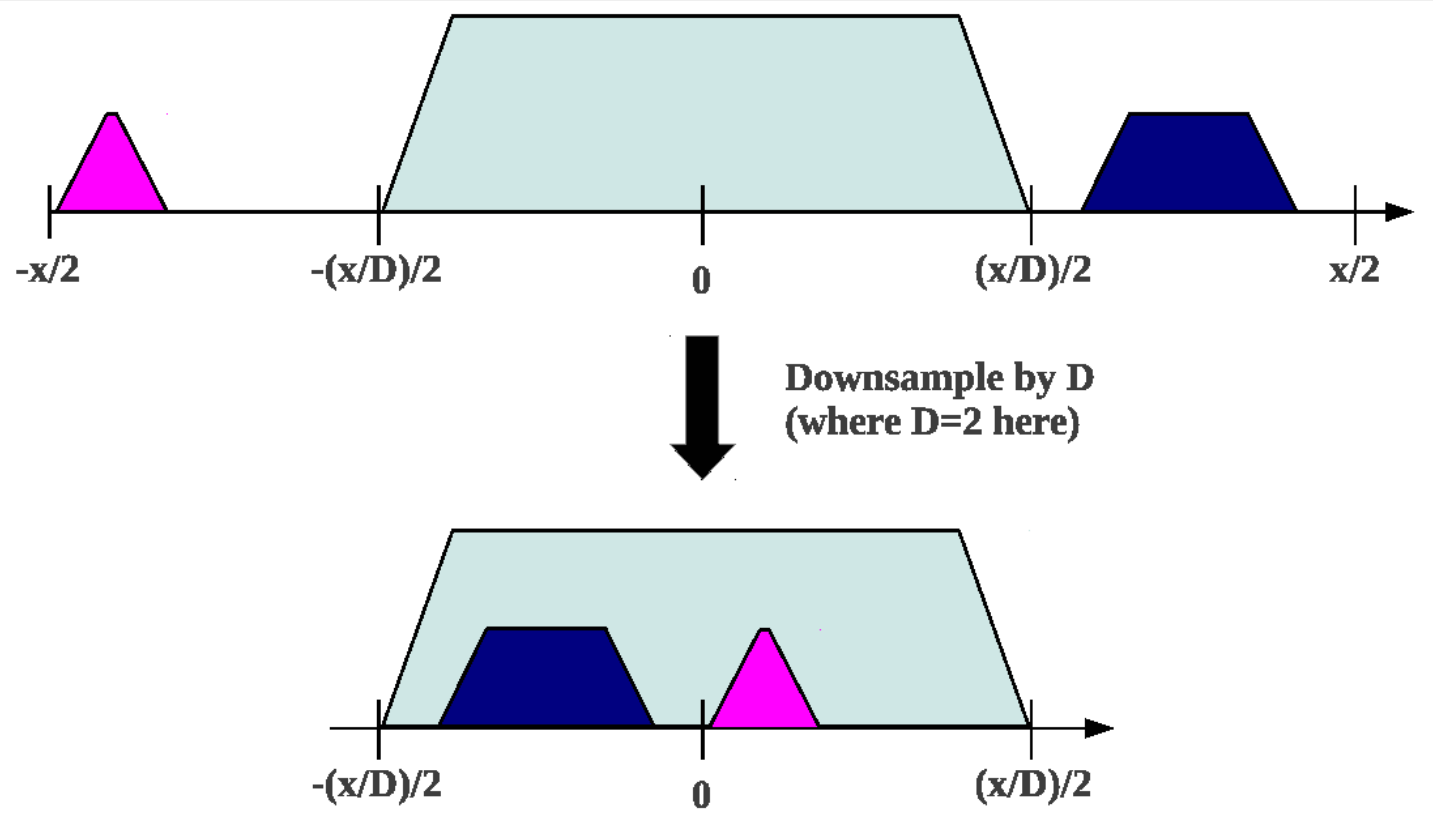


Upsampling (Interpolation)



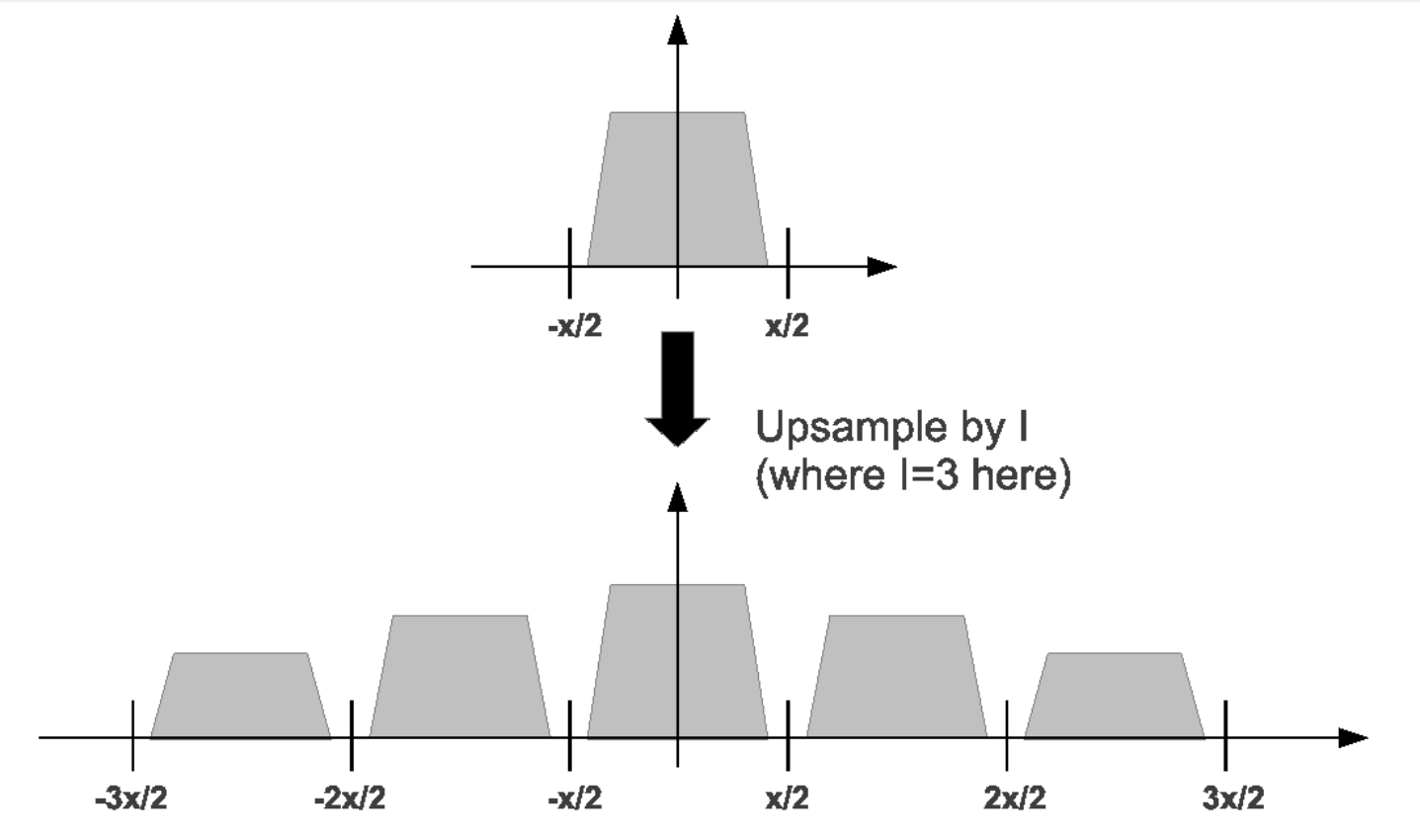
Downsampling aliases outside bands

The 'Nyquist Zones' now fall into the new pass band



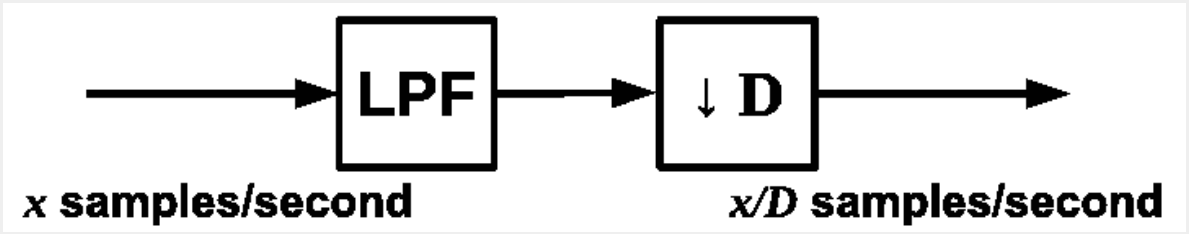
Upsampling creates images

Images fall into outside Nyquist zones

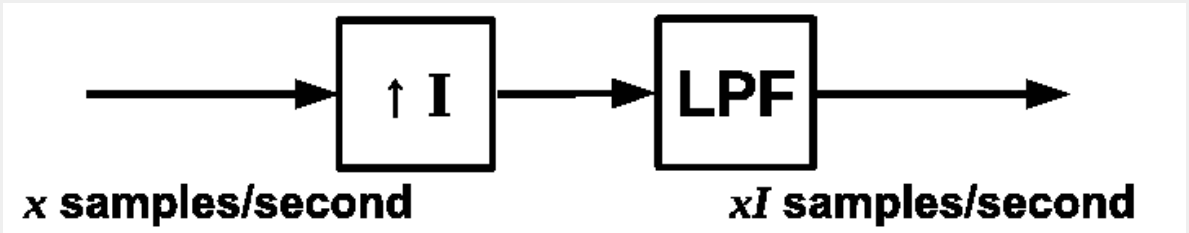


Always filter when changing rates

Filter before downsampling

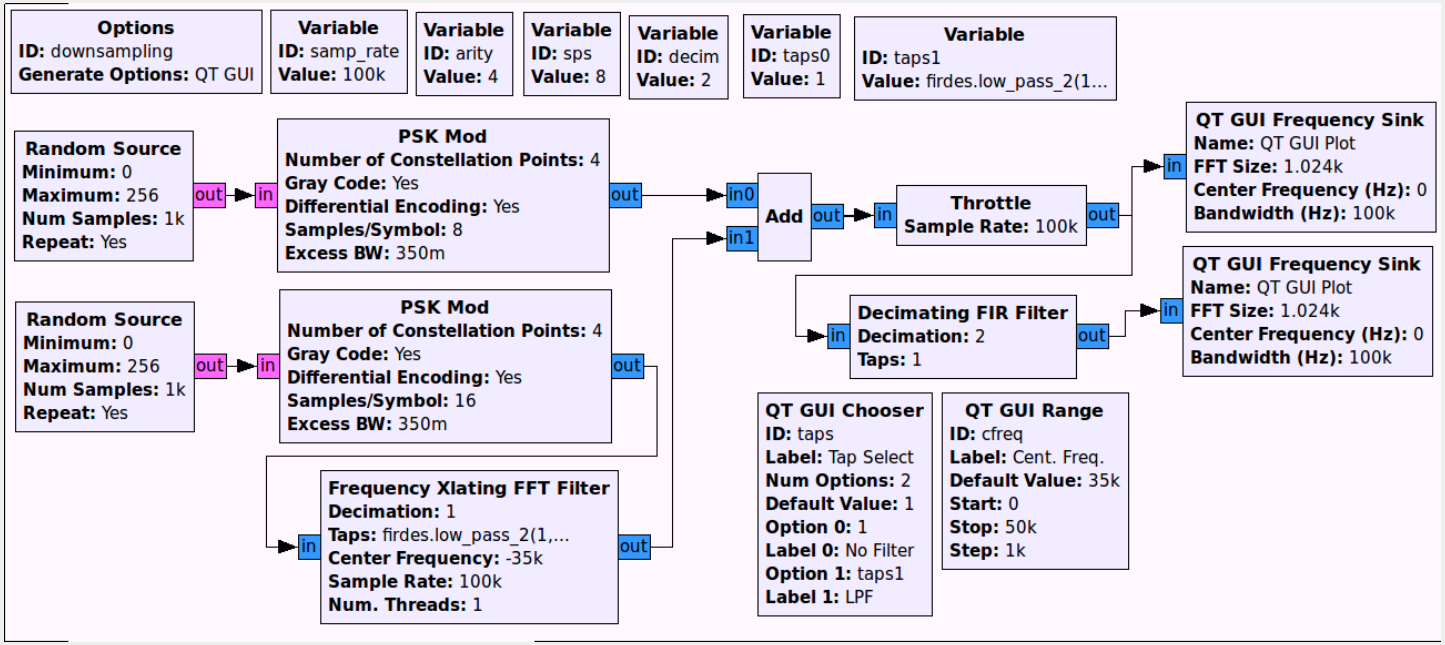


Filter after upsampling



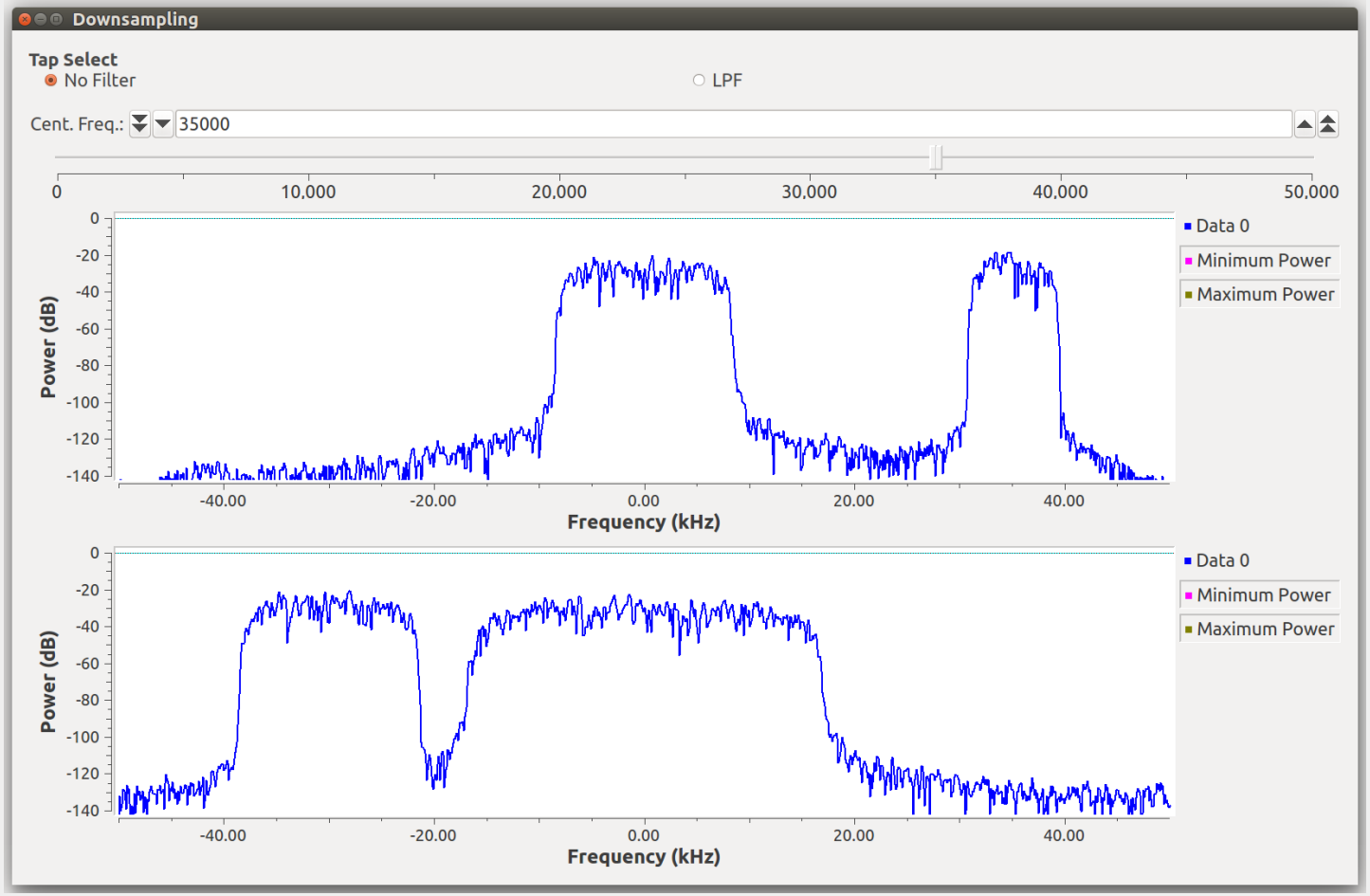
PSK Channels with Aliases

Two PSK signals and a decimating FIR filter



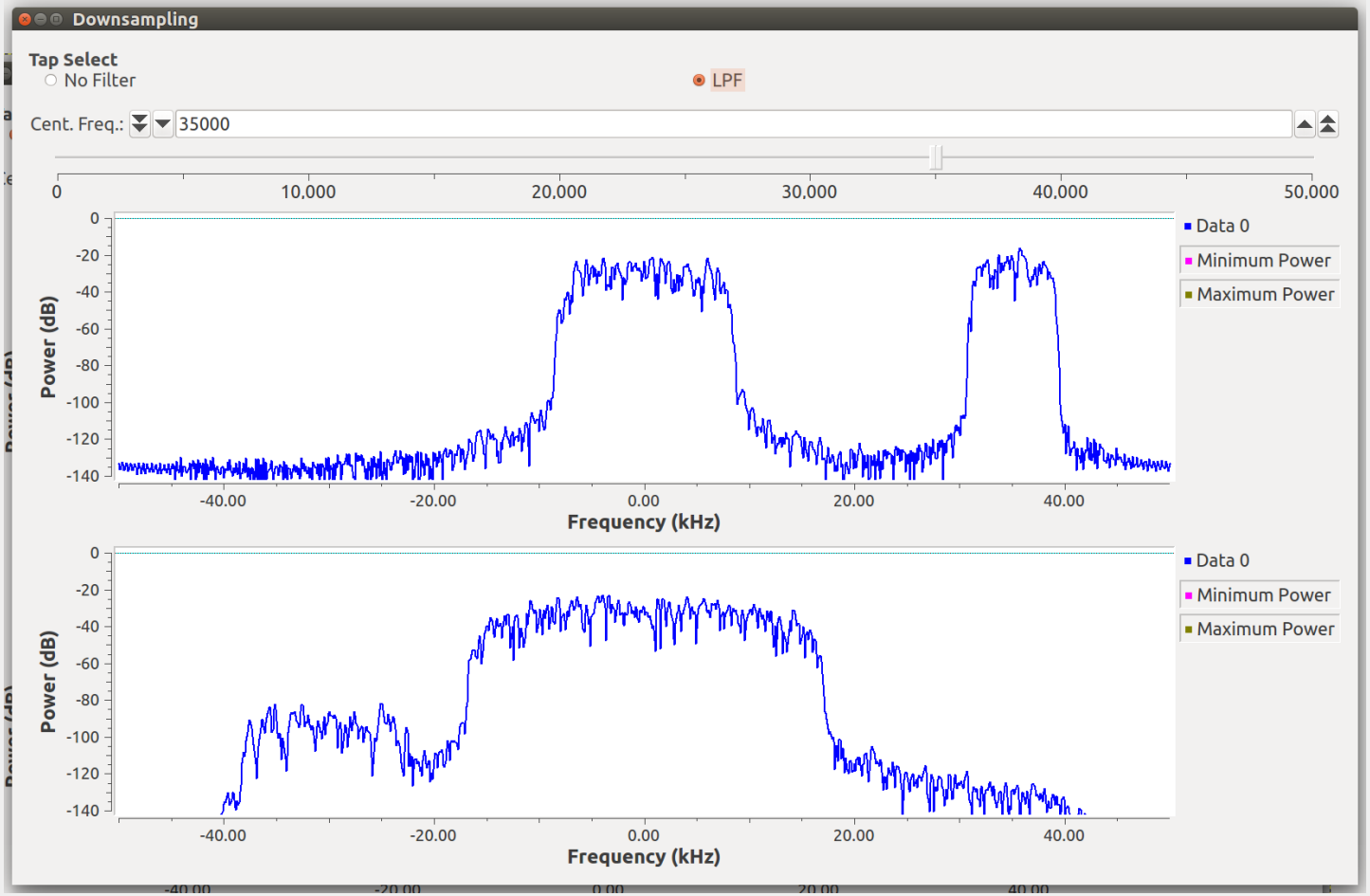
PSK Channels with Aliases

Using no filter taps – second channels aliases into first channel



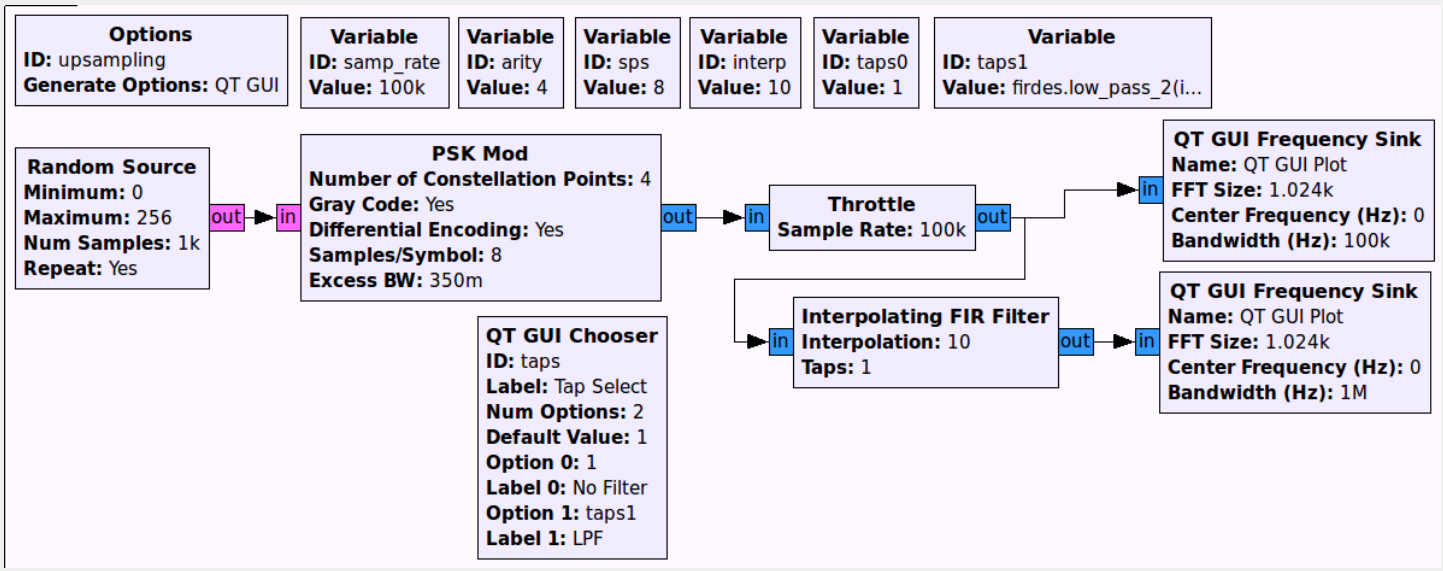
PSK Channels with Aliases

Using FIR filter to reduce adjacent channel interference



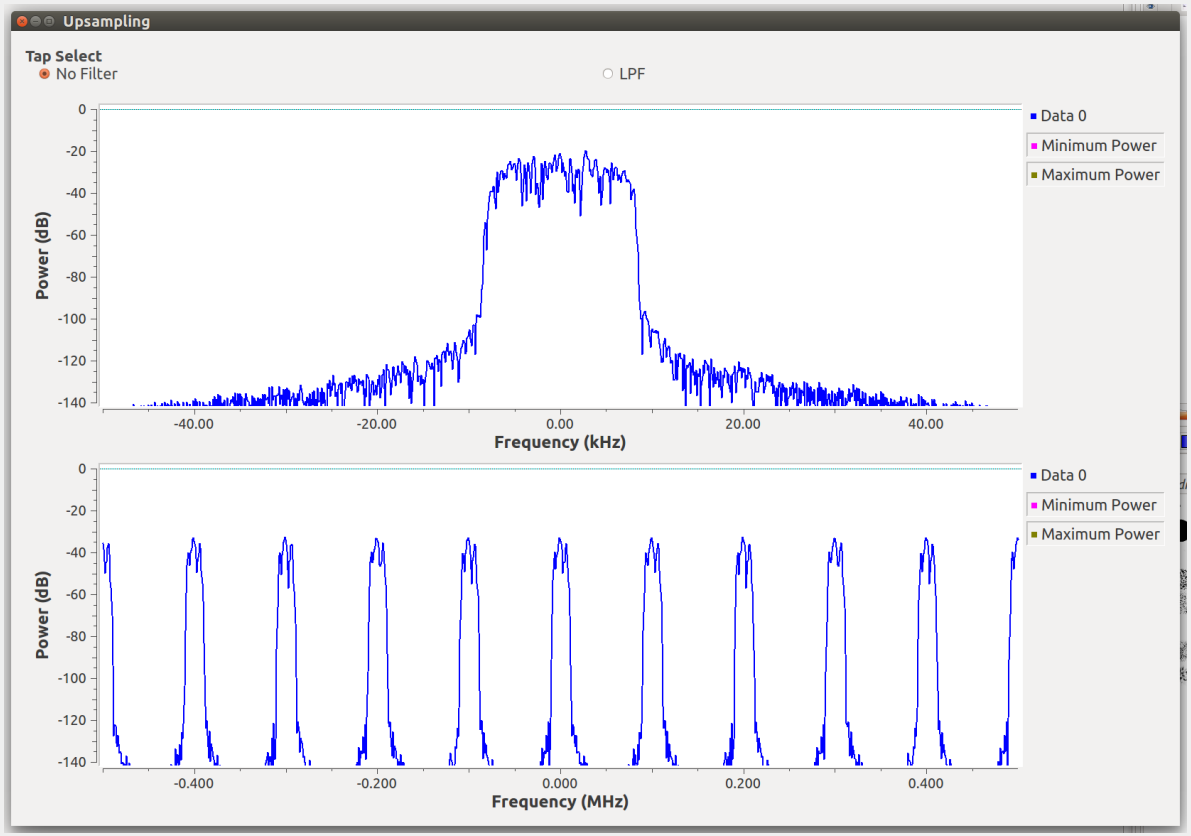
Interpolating a PSK Signal

Upsample a PSK signal by a factor of 10



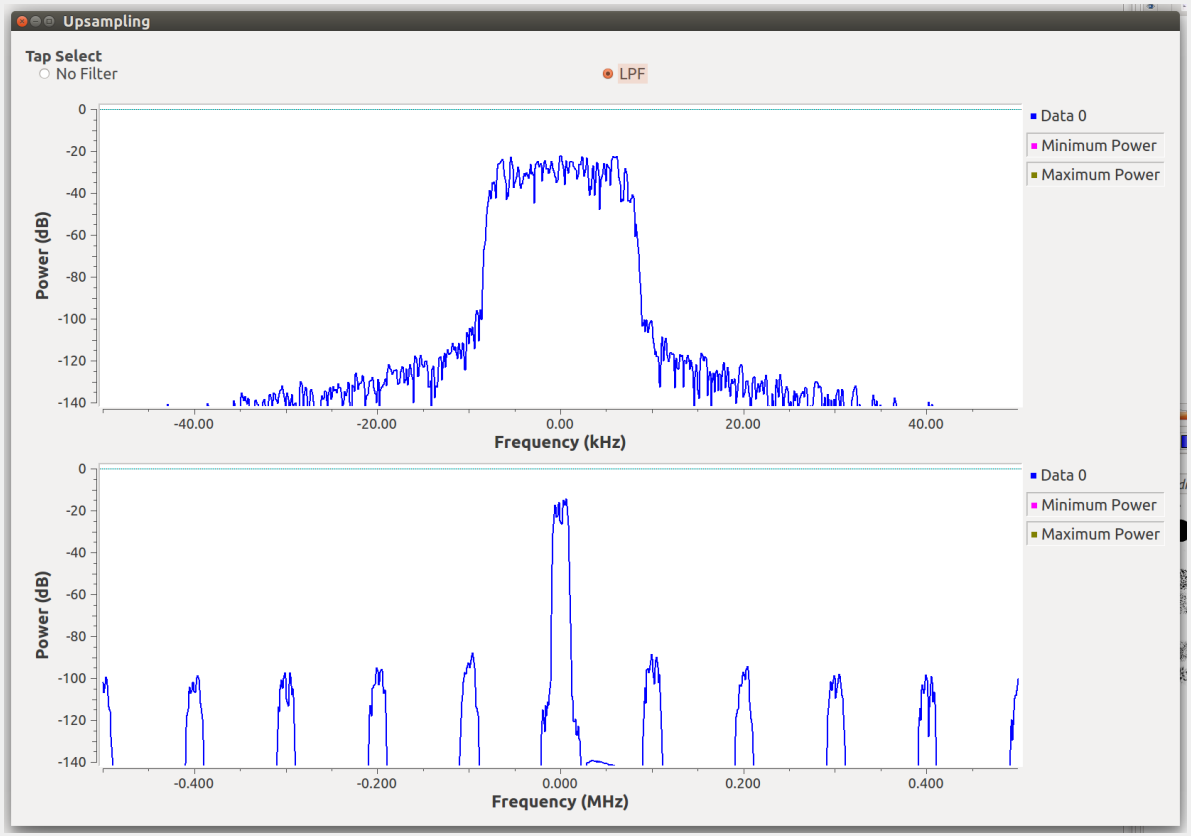
Interpolating a PSK Signal

Using no filter taps – each of the 10 Nyquist zones gets an image

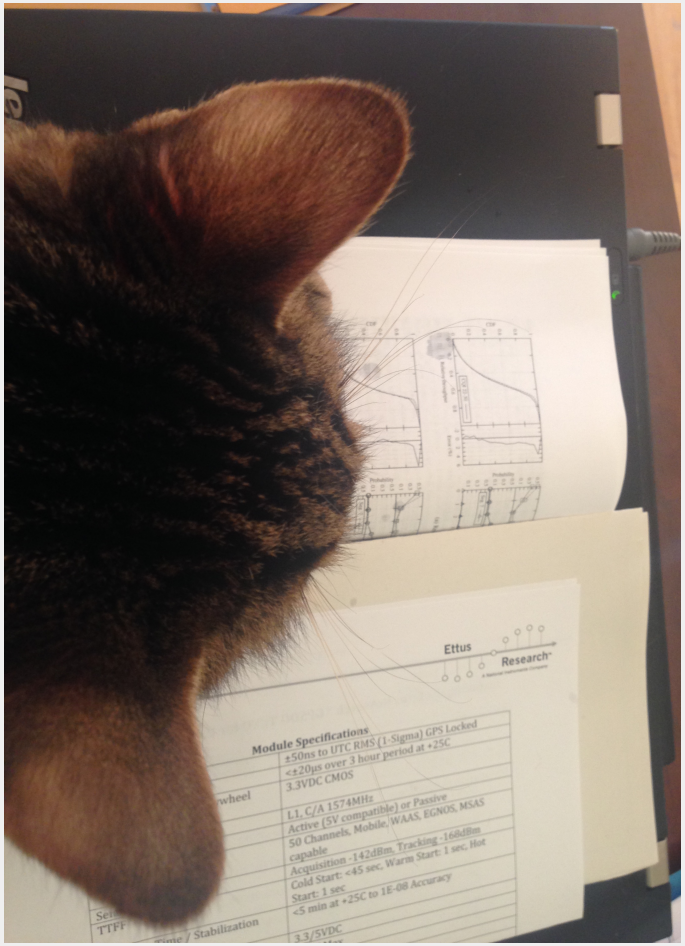


Interpolating a PSK Signal

FIR filter attenuates the images

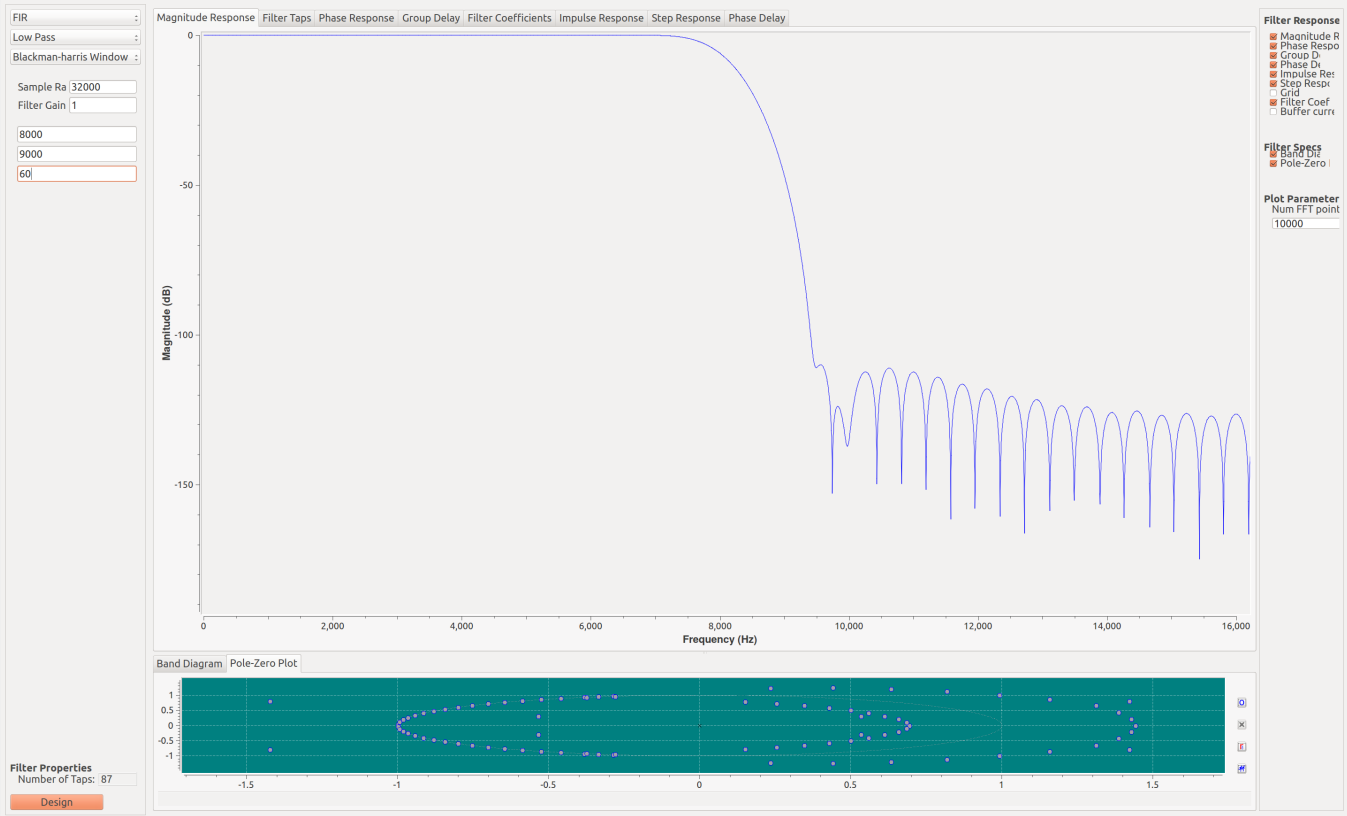


Designing Filters in GNU Radio



The gr_filter_design Tool

GNU Radio comes with a gr_filter_design tool



Filters available in GNU Radio

Finite Impulse Response (FIR) Filters

- Windowed filters using**filter.firdes**.
- Parks-McClellan based using**filter.optfir**.
- Can build: low, high, and band pass, band reject.
- **firdes** also does: Gaussian, Hilbert, root raised cosine.

Infinite Impulse Response (IIR)

- Not fully supported.
- Use Scipy to build IIR.
- Available blocks to compute IIR response (**filter.iir_filter_ffd**,**ccc**,**ccd**,**ccf**,**ccz**).

Filters available in GNU Radio

single-pole

- Implements a single-pole IIR filter moving average.
- **filter.single_pole_iir_filter_{cc,ff}**
- $y[n] = \alpha x[n] + (1 - \alpha) y[n - 1]$

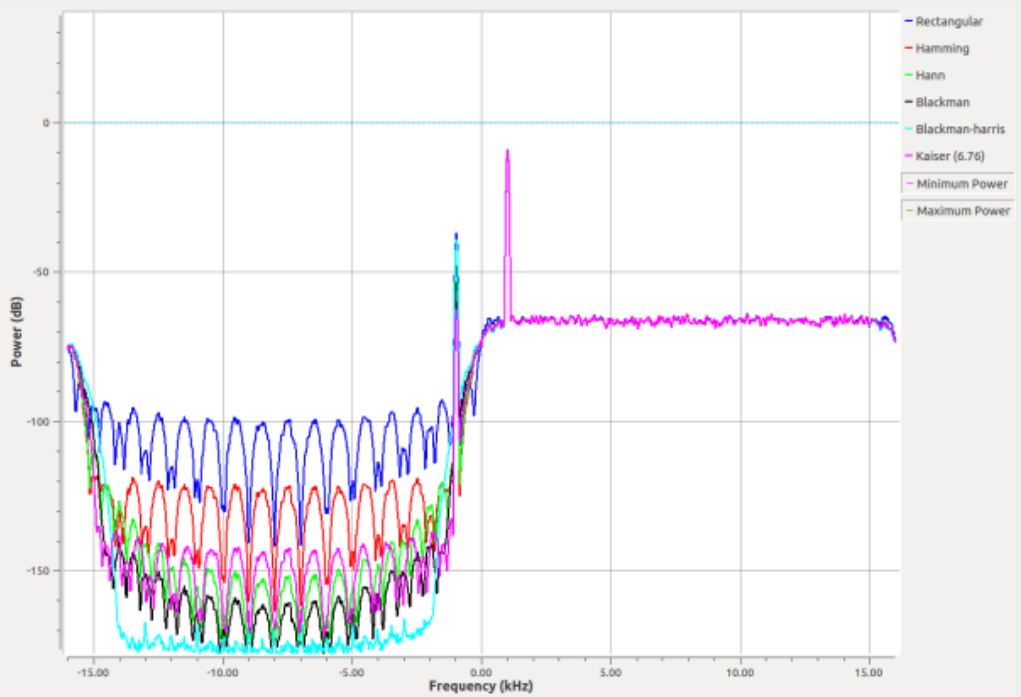
adaptive

- Supports creating adaptive equalizers.
- CMA:**digital.cma_equalizer_cc**
- Decision-directed LMS:**digital.lms_dd_equalizer_cc**
- More possible.

Filters available in GNU Radio

Hilbert

- Use **filter.firdes.hilbert(ntaps, window)** to create filter.
- Convenience block **filter.hilbert_fc(ntaps)** to convert real signal to analytic signal



Filters available in GNU Radio

polyphase filterbanks

- Channelizer (analysis):**filter.pfb_channelizer_ccf**
- Synthesizer:**filter.pfb_synthesizer_ccf**
- Decimator (single channel):**filter.pfb_decimator_ccf**
- Interpolator (single channel):**filter.pfb_interpolator_ccf**
- Arbitrary Resampler:**filter.pfb_arb_resampler_{ccf,fff}**
- **filter.pfb.***: wrapper to make connections easier.

Fast Convolution Filtering

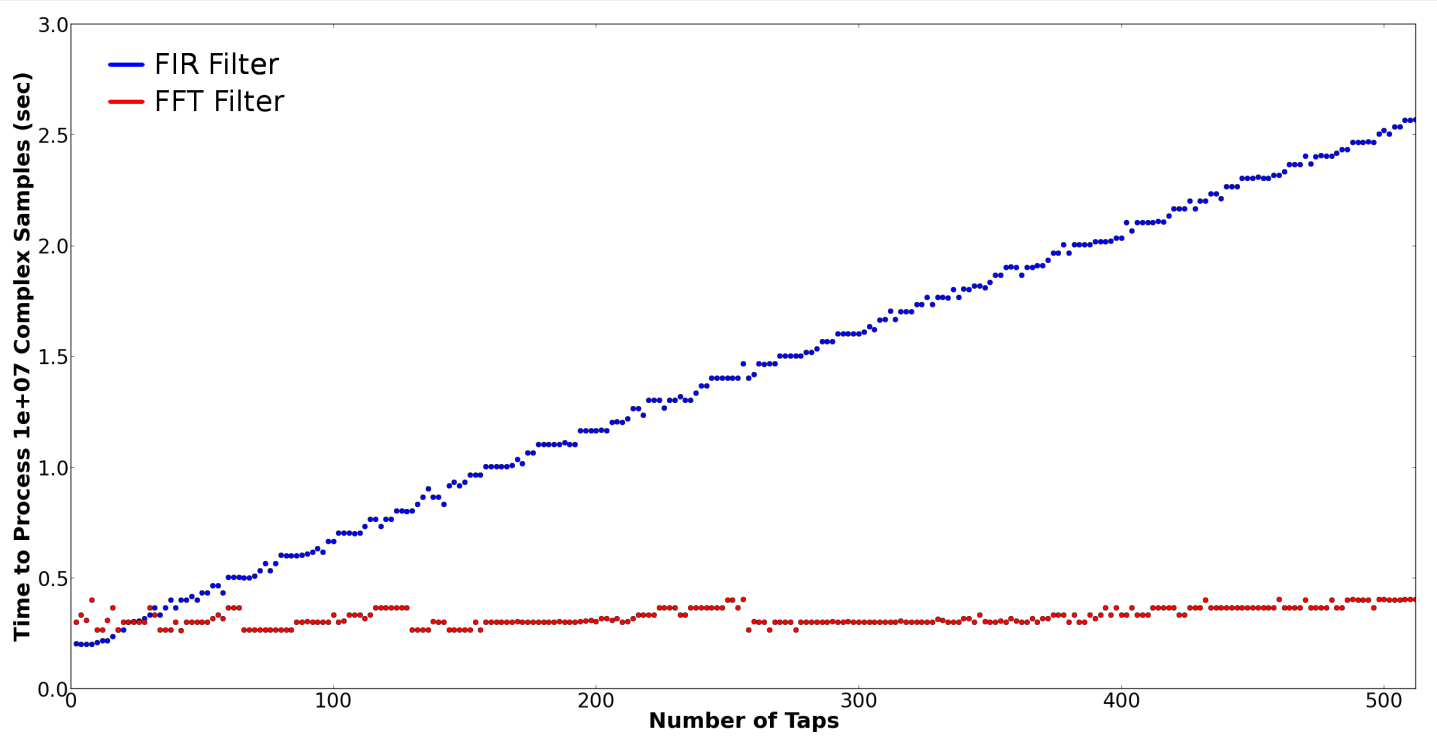
Time vs. Frequency Domain

Convolution in Time $\Longleftrightarrow$ Multiplication in Frequency

$$\mathcal{F}(t * x) = \mathcal{F}(t) \cdot \mathcal{F}(x)$$
$$t * x = \mathcal{F}^{-1}(\mathcal{F}(t) \cdot \mathcal{F}(x))$$

- $\rightarrow \mathcal{F}$ is the Fourier transform operator.
- And we know an FFT can be done with complexity $O(N \log(N))$.

Normal vs. Fast Convolution



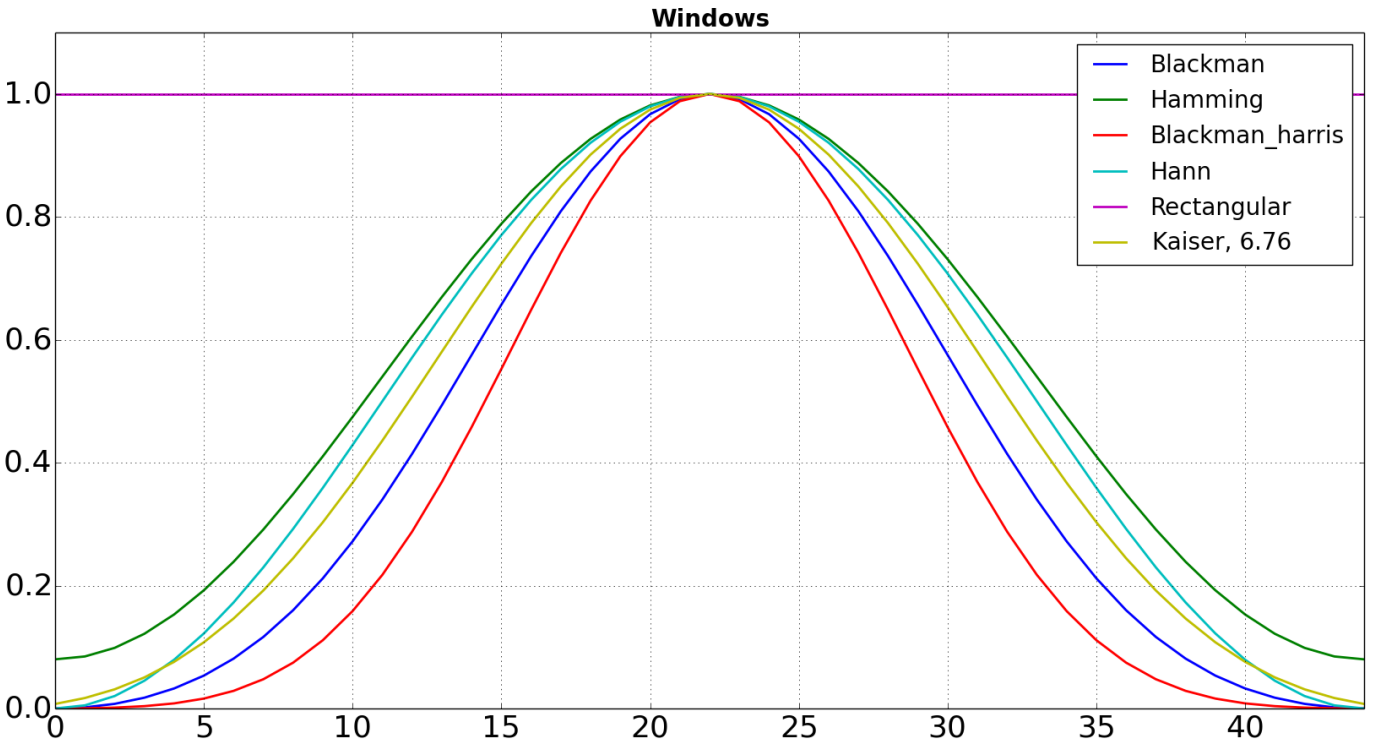
Using firdes and optfir

Using filter.firdes

Types of windows

- fft.window.WIN_BLACKMAN
- fft.window.WIN_HAMMING
- fft.window.WIN_NONE
- fft.window.WIN_BLACKMAN_HARRIS
- fft.window.WIN_HANN
- fft.window.WIN_RECTANGULAR
- fft.window.WIN_BLACKMAN_hARRIS
- fft.window.WIN_KAISER

Windows Behaviors



Using filter.firdes

Standard Filters

- low_pass
- high_pass
- band_pass
- band_reject
- complex_band_pass
- hilbert
- root_raised_cosine
- Gaussian

Using filter.firdes

Filters with Attenuation Parameter

- low_pass_2
- high_pass_2
- band_pass_2
- band_reject_2
- complex_band_pass_2

filter.firdes.low_pass_2

Low Pass

```
filter.firdes.low_pass_2(
  gain, # Filter gain
  sampling_freq, # Sample rate of filter
  cutoff_freq, # End of pass band
  transition_width, # width of pass band to stop band
  attenuation_dB, # stop band attenuation
  window = firdes.WIN_HAMMING, # window type
  beta = 6.76) # only used if window is WIN_KAISER
```

- All frequencies are relative to sample_freq.

filter.firdes.complex_band_pass_2

Band Pass (complex)

```
filter.firdes.complex_band_pass_2(
  gain, # Filter gain
  sampling_freq, # Sample rate of filter
  low_cutoff_freq, # start of pass band
  high_cutoff_freq, # end of pass band
  transition_width, # width of pass band to stop band
  attenuation_dB, # stop band attenuation
  window = WIN_HAMMING, # window type
  beta = 6.76) # only used if window is WIN_KAISER
```

- Set start and stop of pass band; symmetric transition band on either end.

Using filter.optfir

Type of filters

- filter.optfir.low_pass
- filter.optfir.high_pass
- filter.optfir.band_pass
- filter.optfir.band_reject
- filter.optfir.complex_band_pass

filter.optfir.low_pass

Low pass

```
filter.optfir.low_pass(
  gain, # Filter gain
  Fs, # Sample rate of filter
  freq1, # end of pass band
  freq2, # start of stop band
  passband_ripple_db, # pass band ripple (in dB)
  stopband_atten_db, # stop band attenuation (in dB)
  nextra_taps=2) # for Remez convergence
```

- Instead of transition band, this specifies the end of the pass band and the start of the transition band.
- **nextra_taps** is to be able to specify some number of taps more than estimated to help converge.
- Using the Parks-McClellan algorithm,**optfir** filters may not converge.

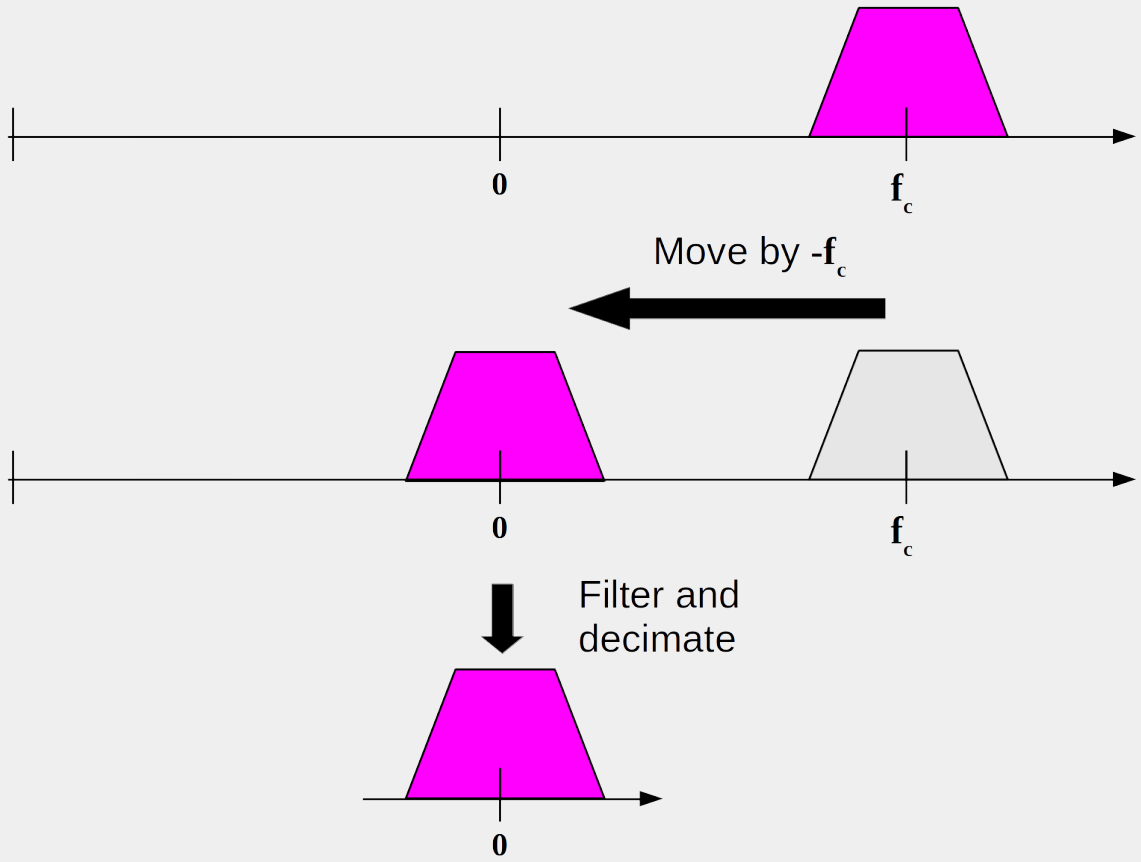
Polyphase Filterbanks

"It's a neat trick."

- fred harris

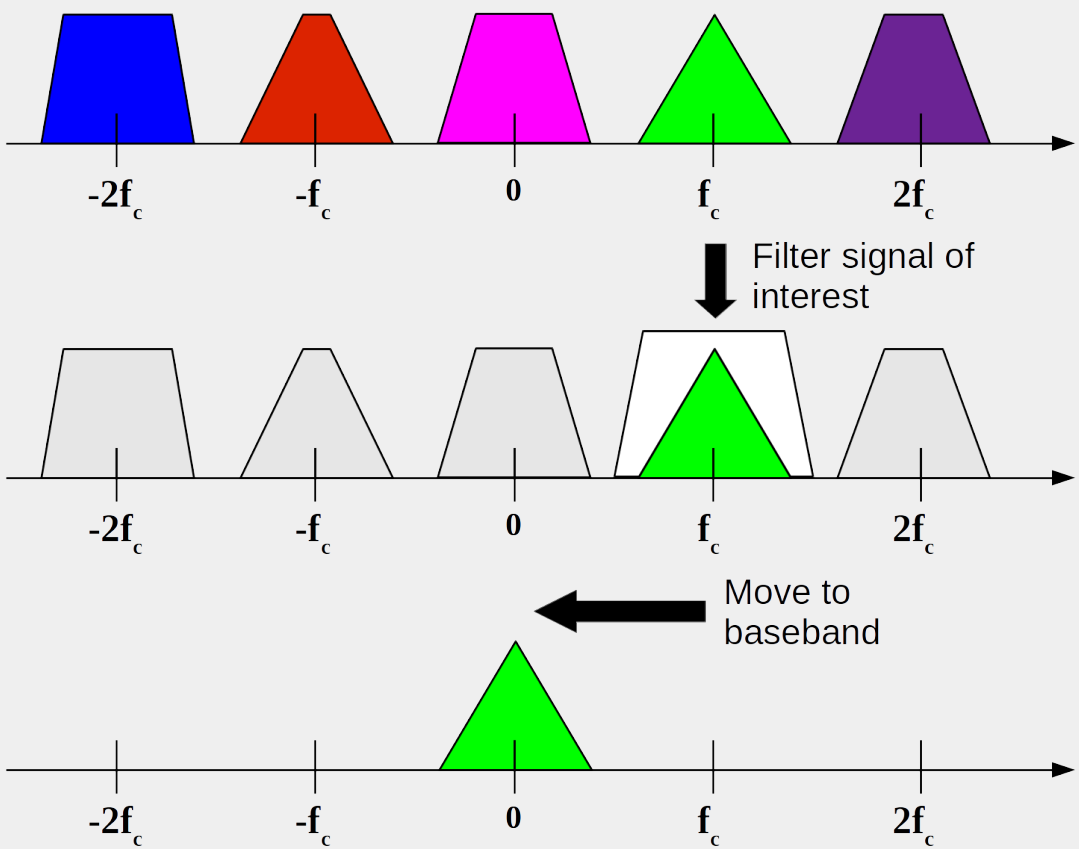
Single Channel Downconversion

Pretty simple concept to grab a single channel and process it



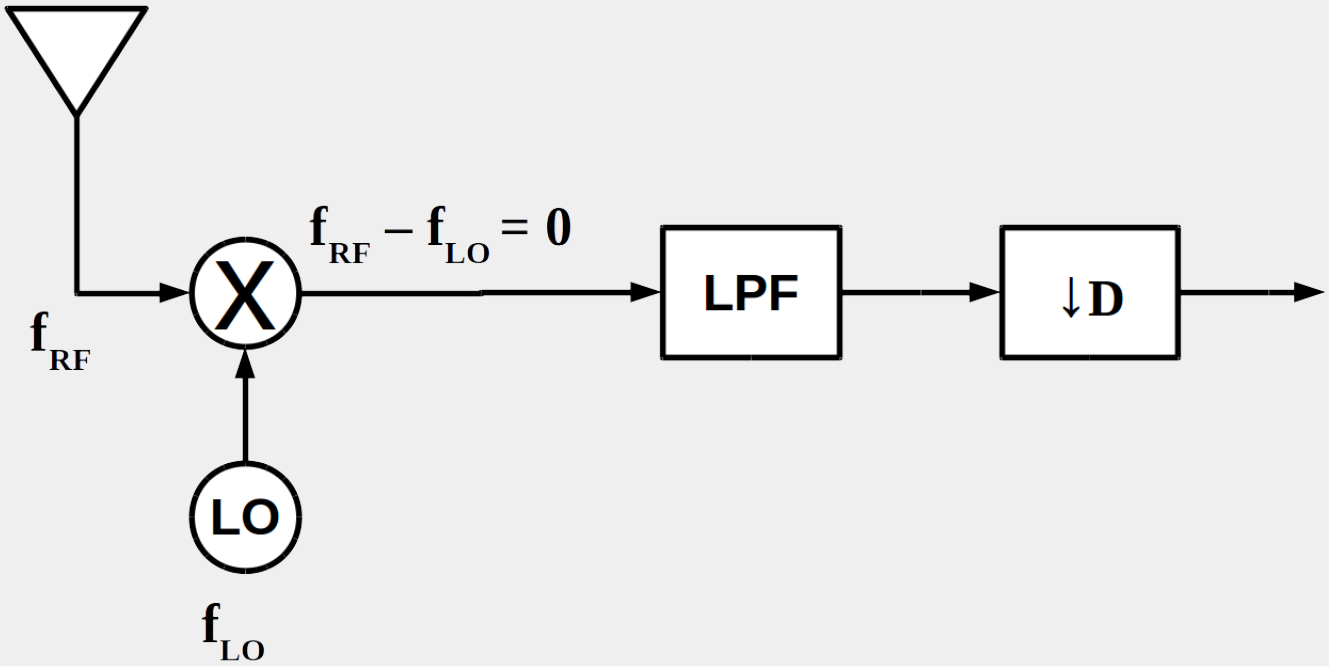
Processing Multiple Channels

What if we want to process all these channels together?



Single Channel Downconversion

We could replicate this structure for every center frequency.

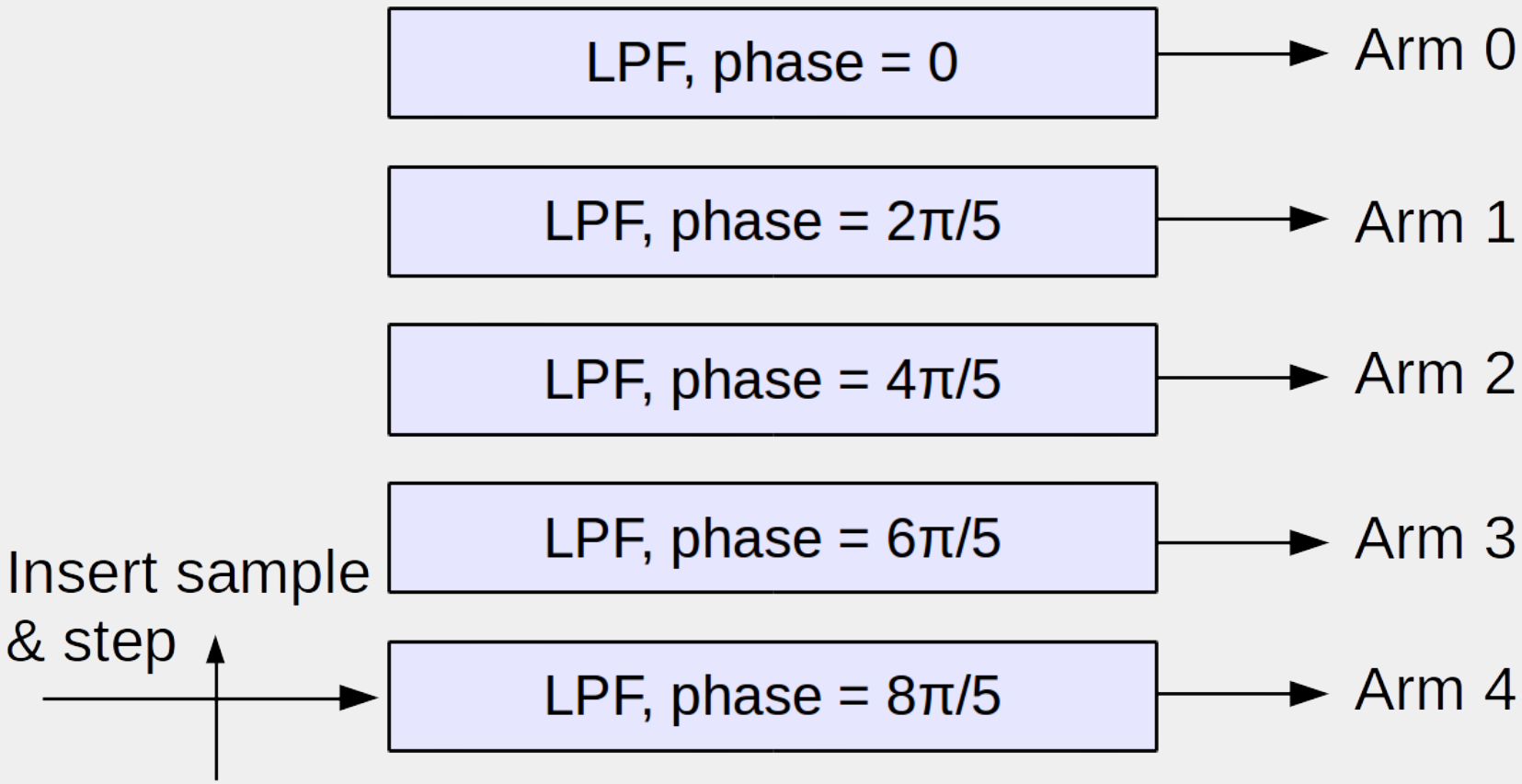


- How well can this scale?

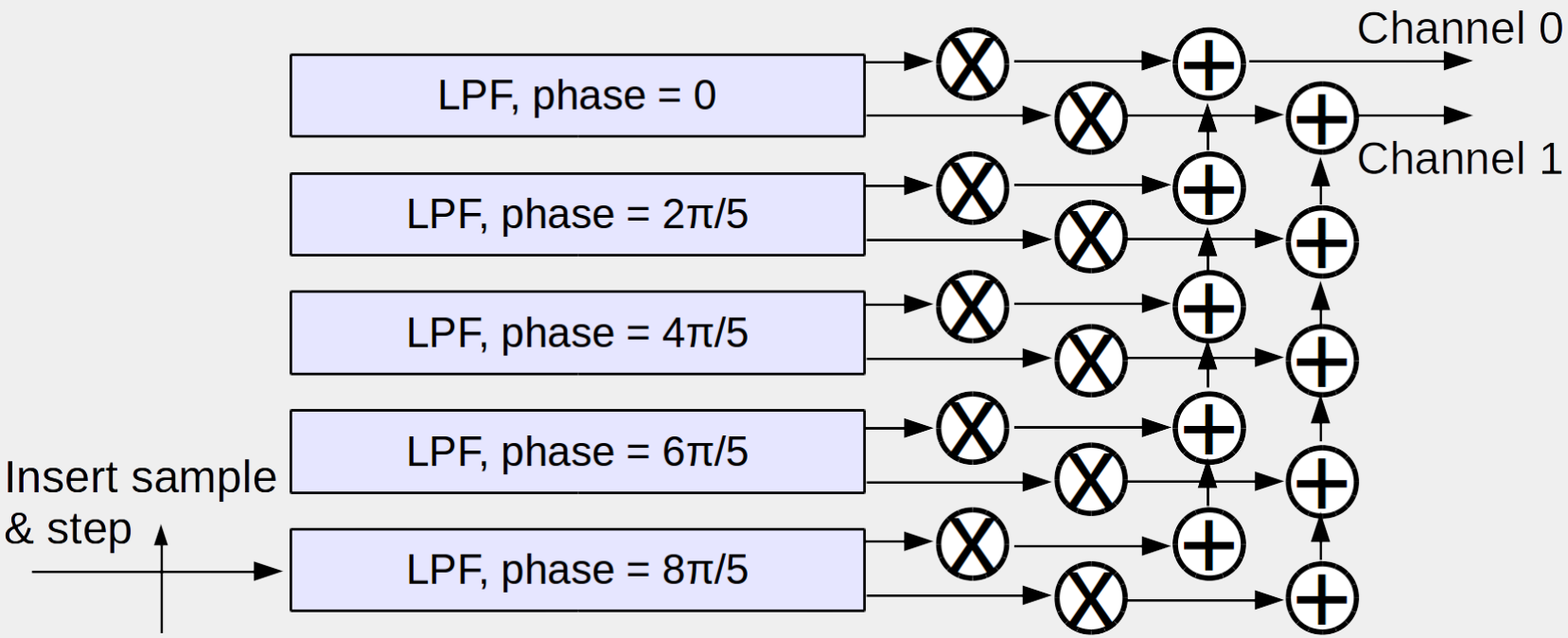
The Polyphase Filterbank Channelizer can do all these channels at the same time.

- Abuse aliasing to shift all channels down onto baseband.
- Each channel shows up at a different phase.
- We can combine each phase independently to separate the channels.
- Algorithm:
  - distribute samples 1 per filter arm (downsamples data stream; aliases signals)
  - filter each arm with a specific filter phase
  - rotate the output of each filter by a complex exponential to sum together each phase.

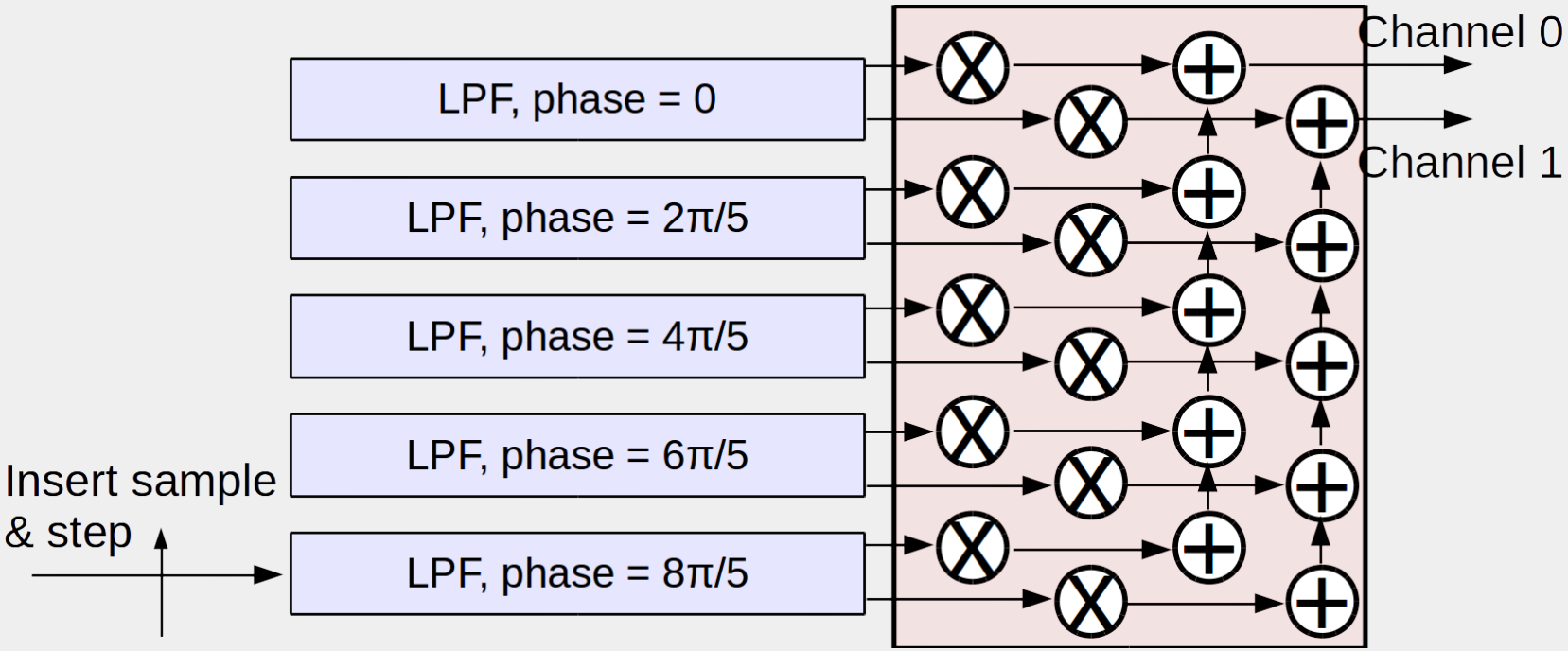
Polyphase Filterbanks



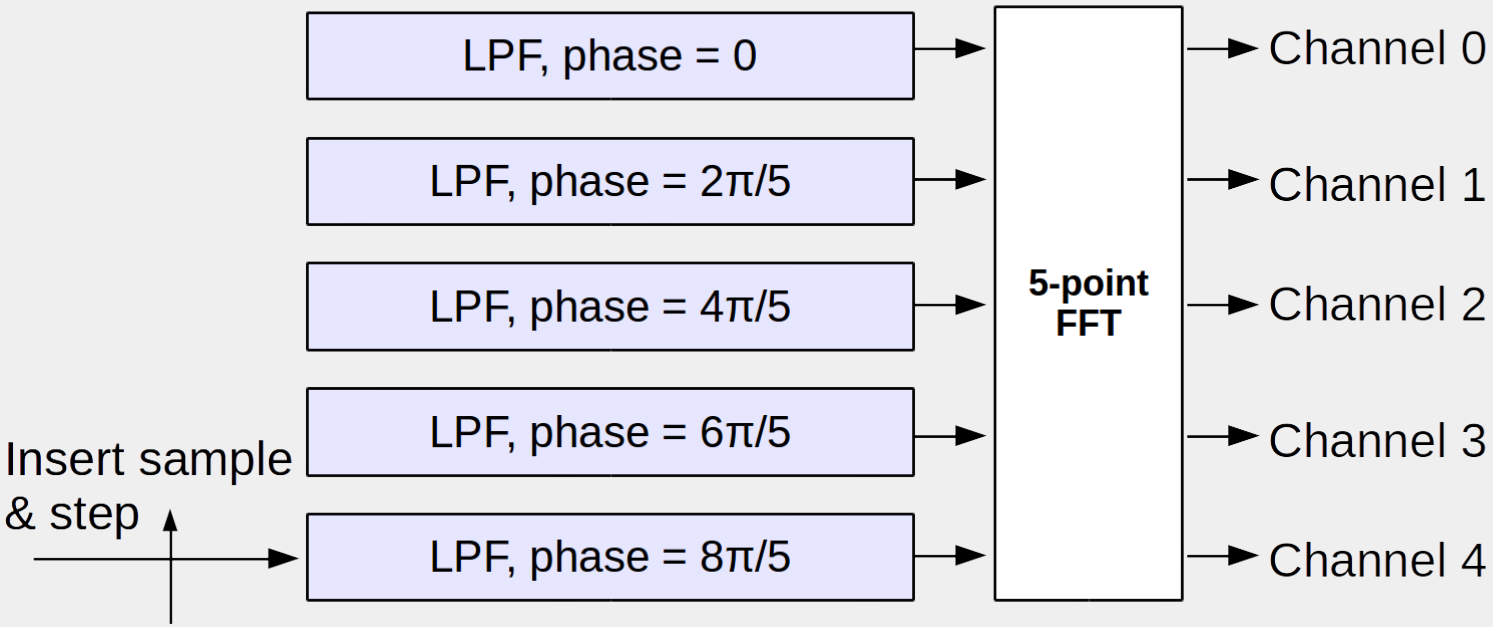
Polyphase Filterbanks



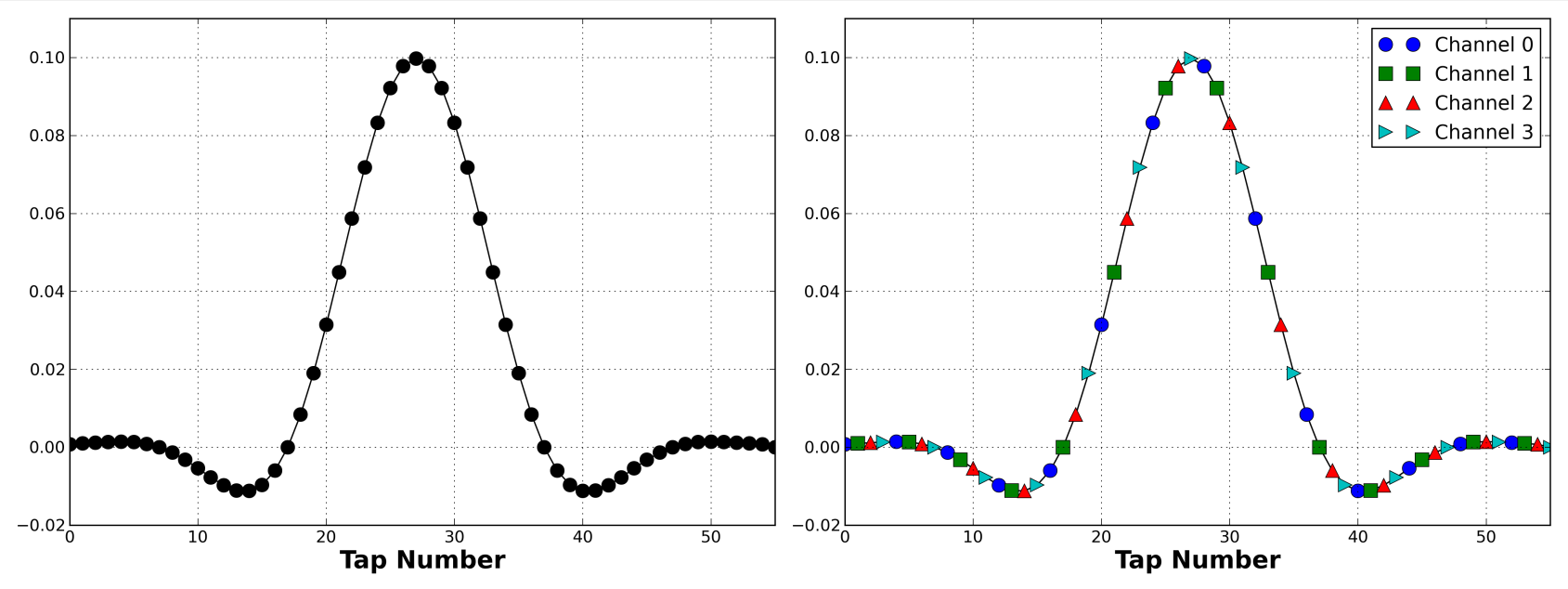
Polyphase Filterbanks



Polyphase Filterbanks



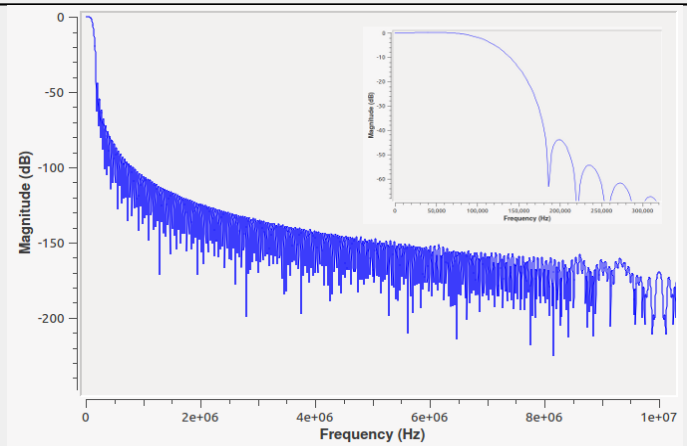
Prototype Filter



Building the FIR Filter to channelize the FM band

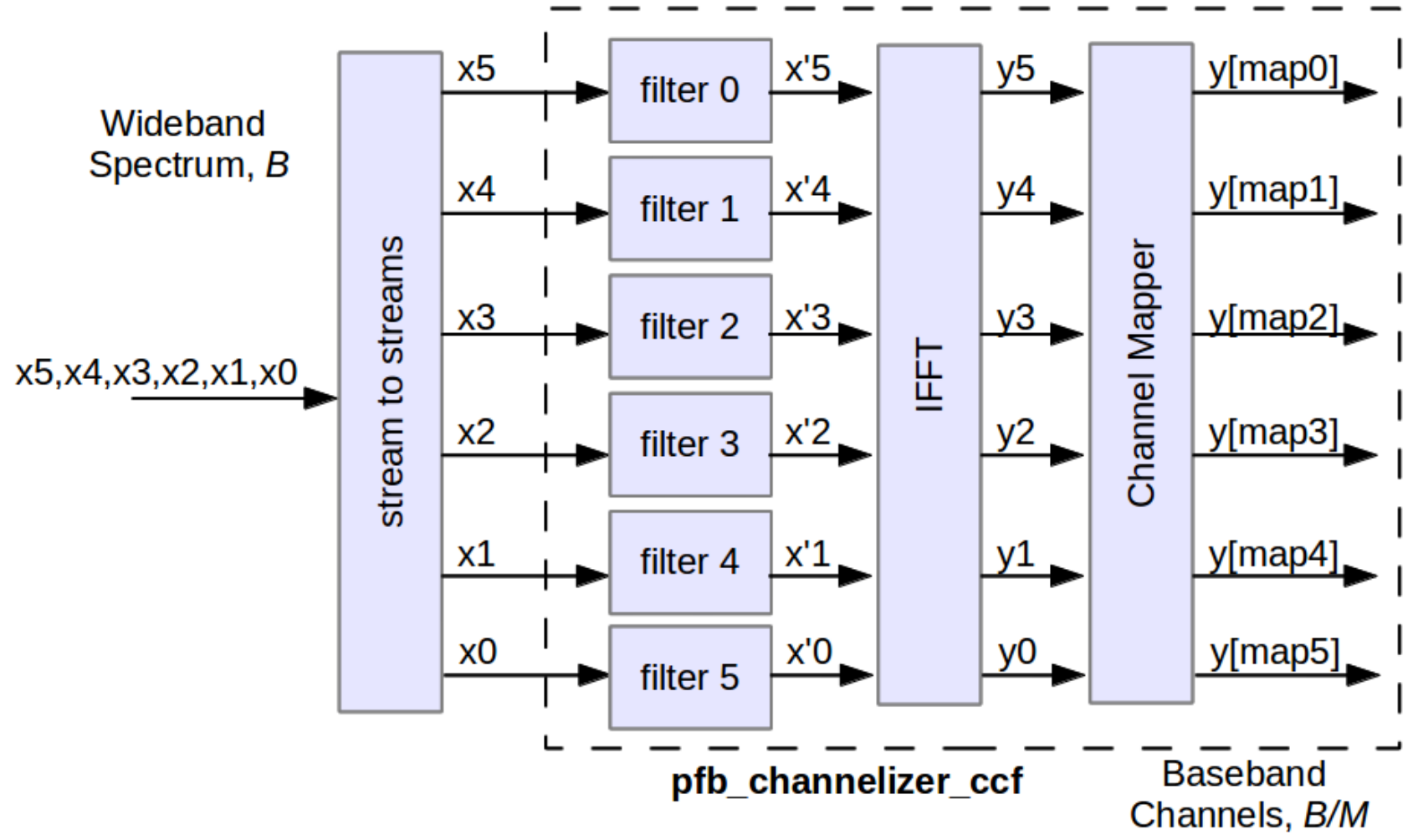
firdes.low_pass_2(1, 20e6, 125e3, 100e3, 60, firdes.WIN_HANN)

Parameter	Value
Type	FIR
Style	Low Pass
Window	Hann
Sample Rate	20 MHz
Filter Gain	1
End of Pass Band	125 kHz
Start of Stop Band	225 kHz
Stop Band Atten	60 dB

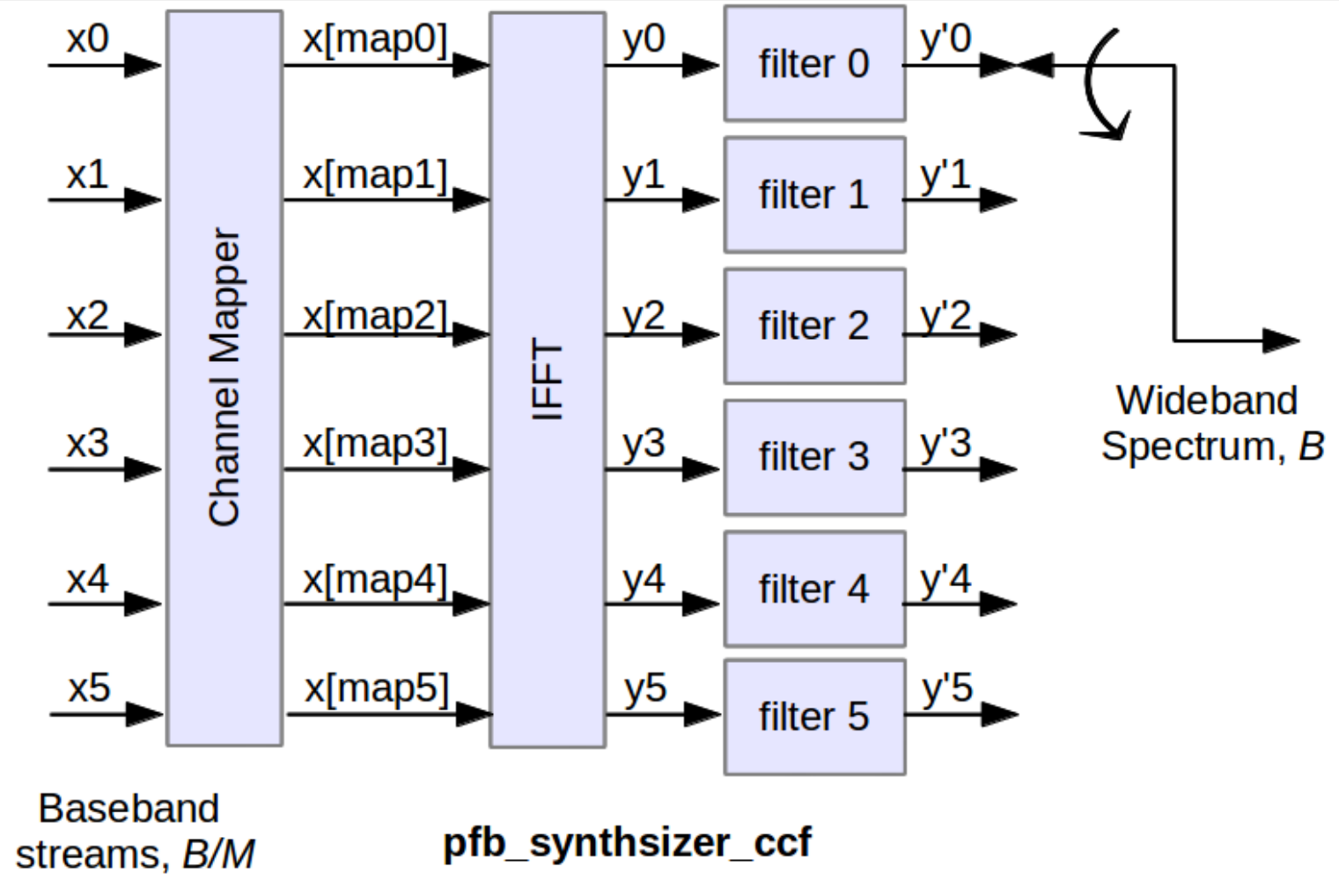


Number of Taps: 545
Channels: 100
Per filter: $\lceil 545/100 \rceil = 6$

Channelizer (channel bw = sample rate)



Synthesizer (channel bw = sample rate)

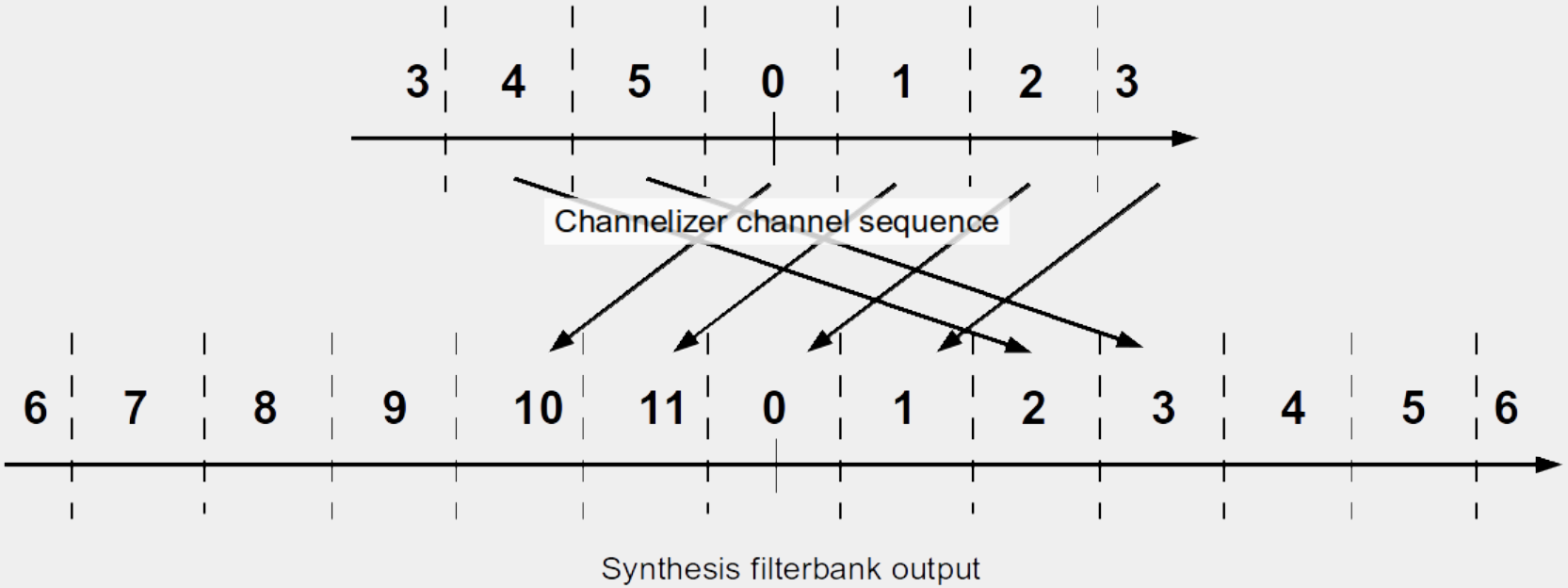


Oversampling Filterbanks

Build Filterbanks using non-critically sampled channels

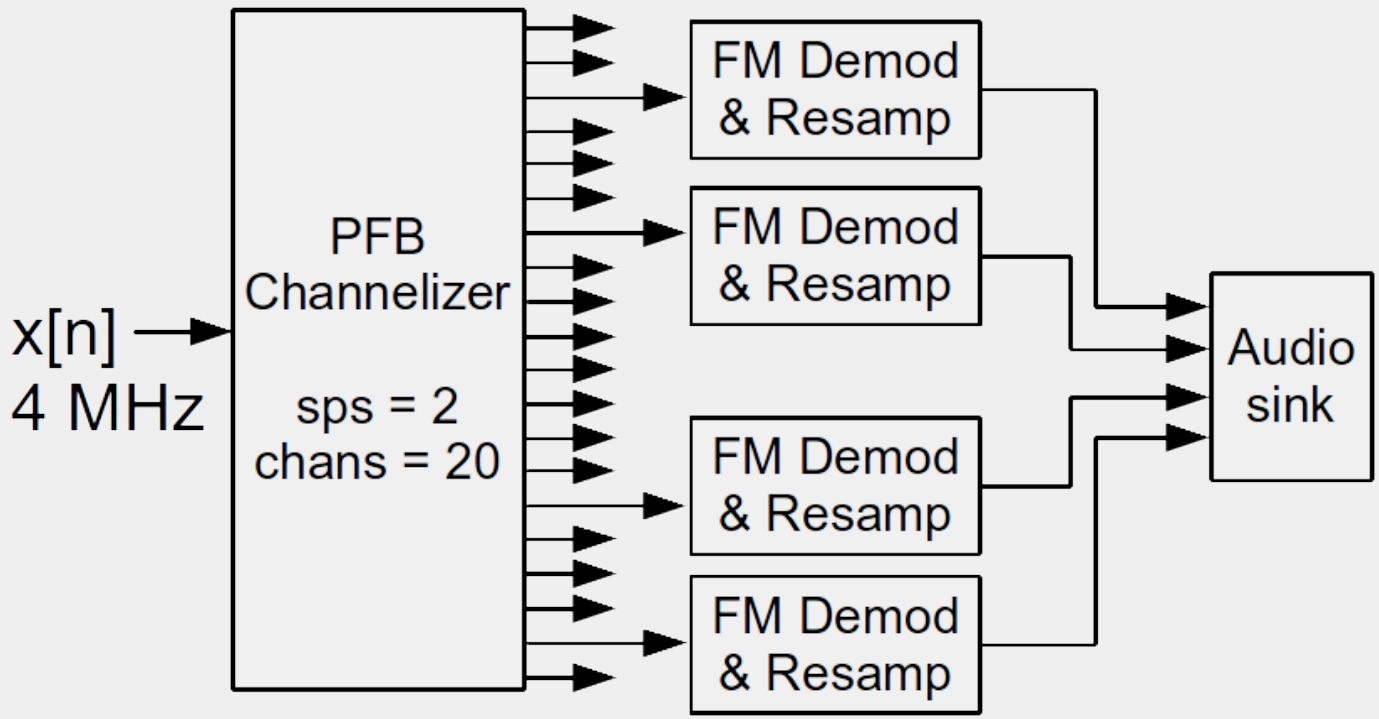
- PFB Channelizer: can set an oversample value as an integer.
  - Restriction is that number of channels*must be*a multiple of the oversampled rate.
  - Set using an optional third argument to constructor (*oversample_rate*).
    - Defaults to 1.
- PFB Synthesizer: can only set oversample rate of 2.
  - Oversampling refers to the*output* rate (i.e., 2x number of channels).
  - Internal structure much more complicated and not generalized.
  - Set using an optional bool argument to constructor (twox).
    - Defaults to false.

Channel Mapping for 2x Oversampled, 6 input channels ([10, 11, 0, 1, 2, 3])



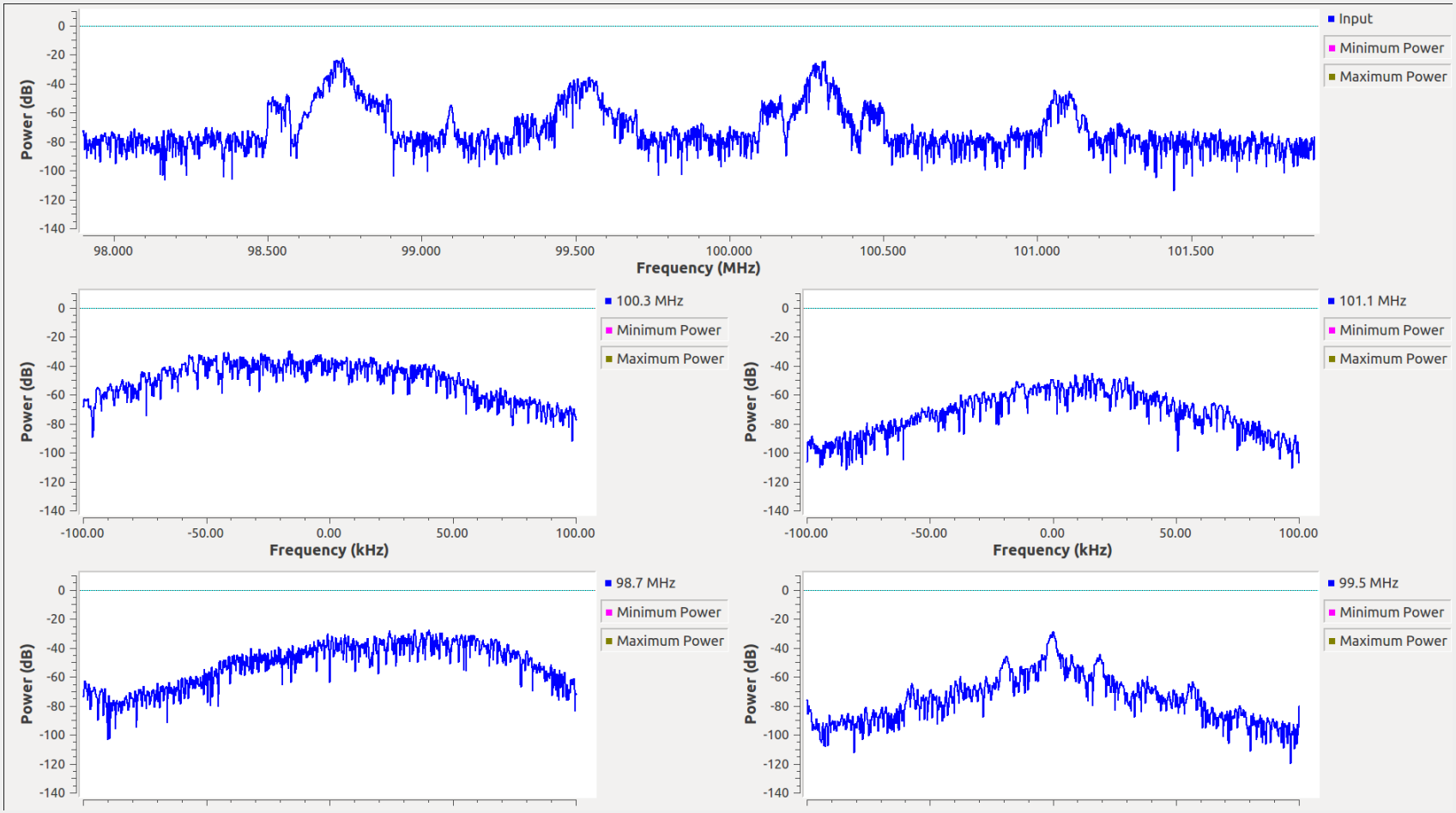
Channelize FM Spectrum

firdes.low_pass_2(1, 4M, 75k, 100k, 60, firdes.WIN_HANN)

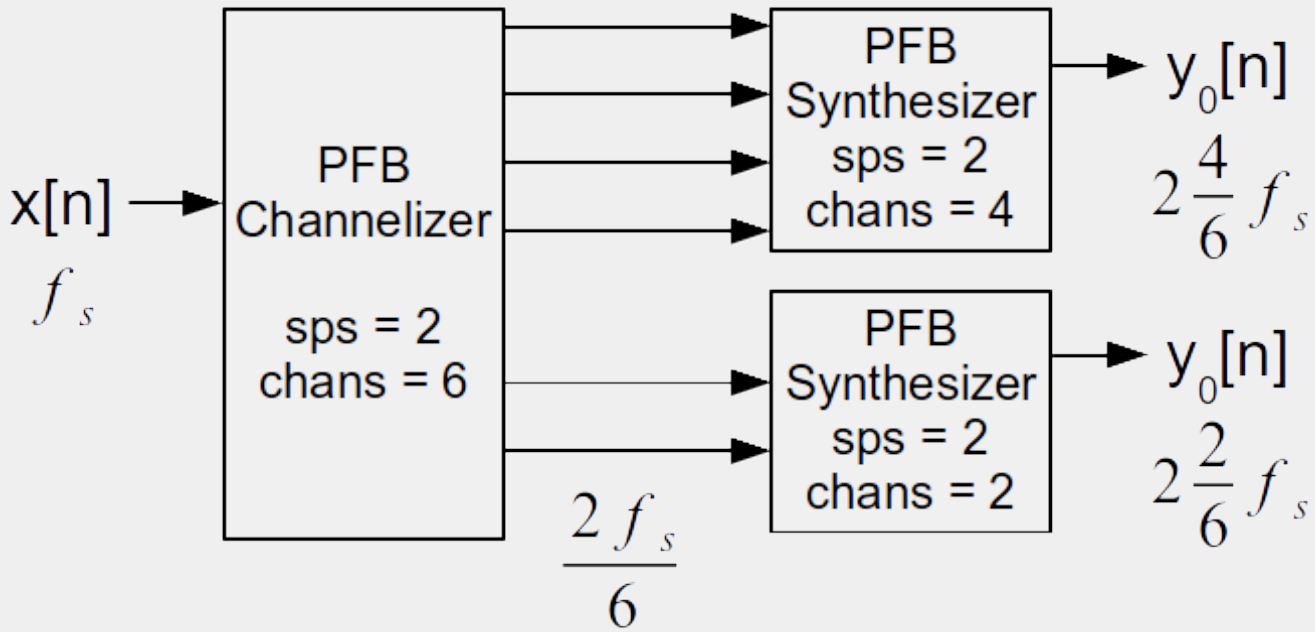


FM Spectrum (in Burlington, VT)

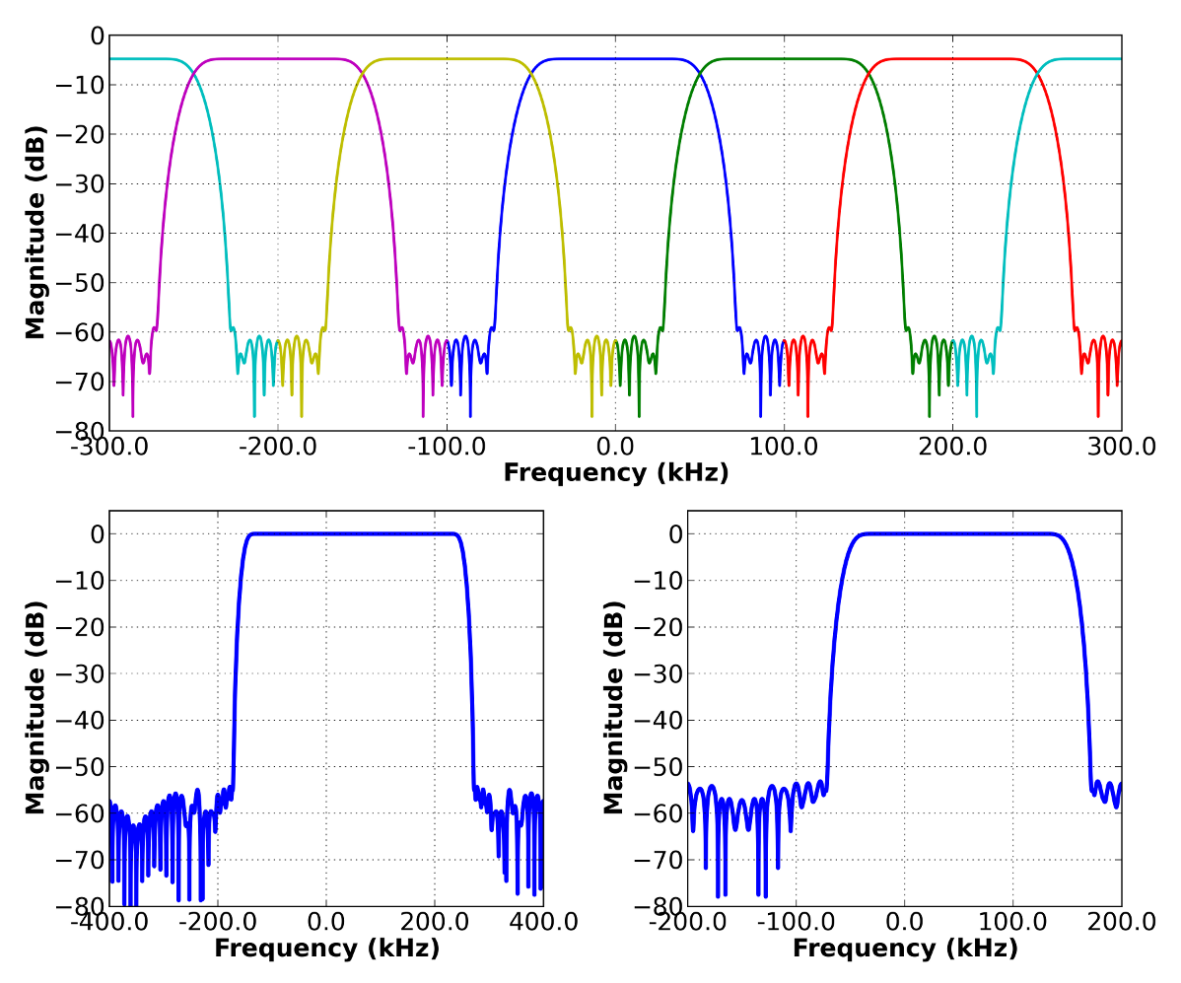
Full 20 MHz spectrum and 4 channels



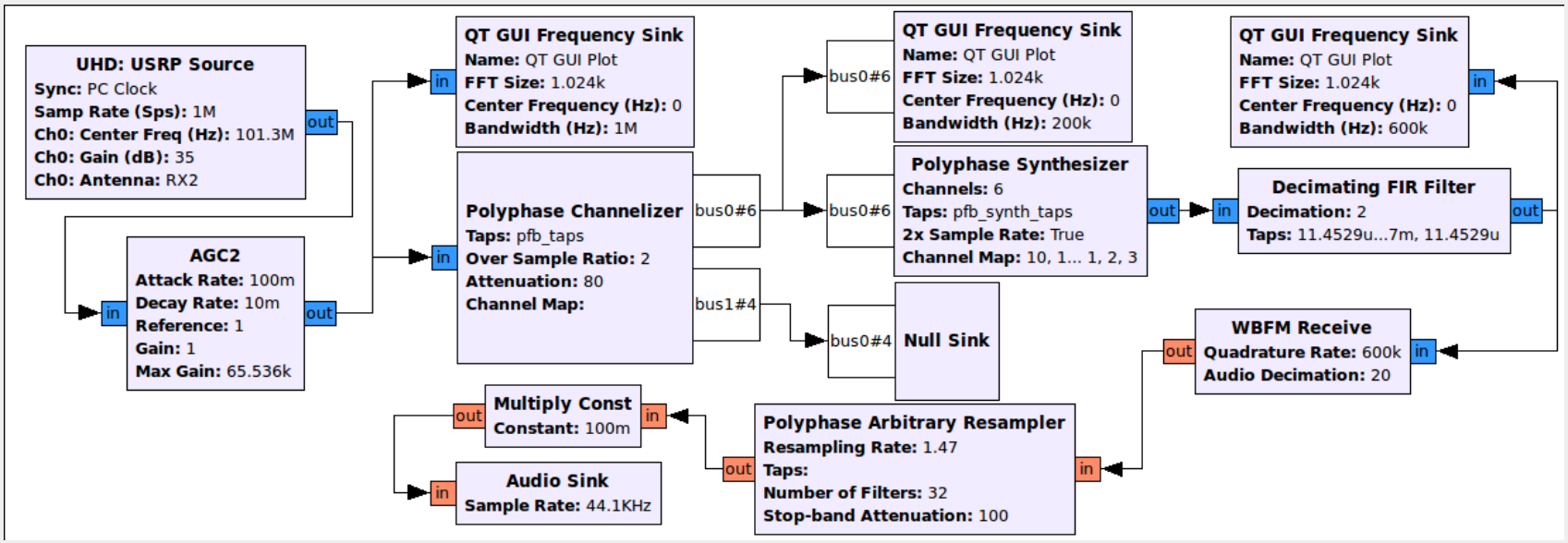
(Re)Synthesizing Channels



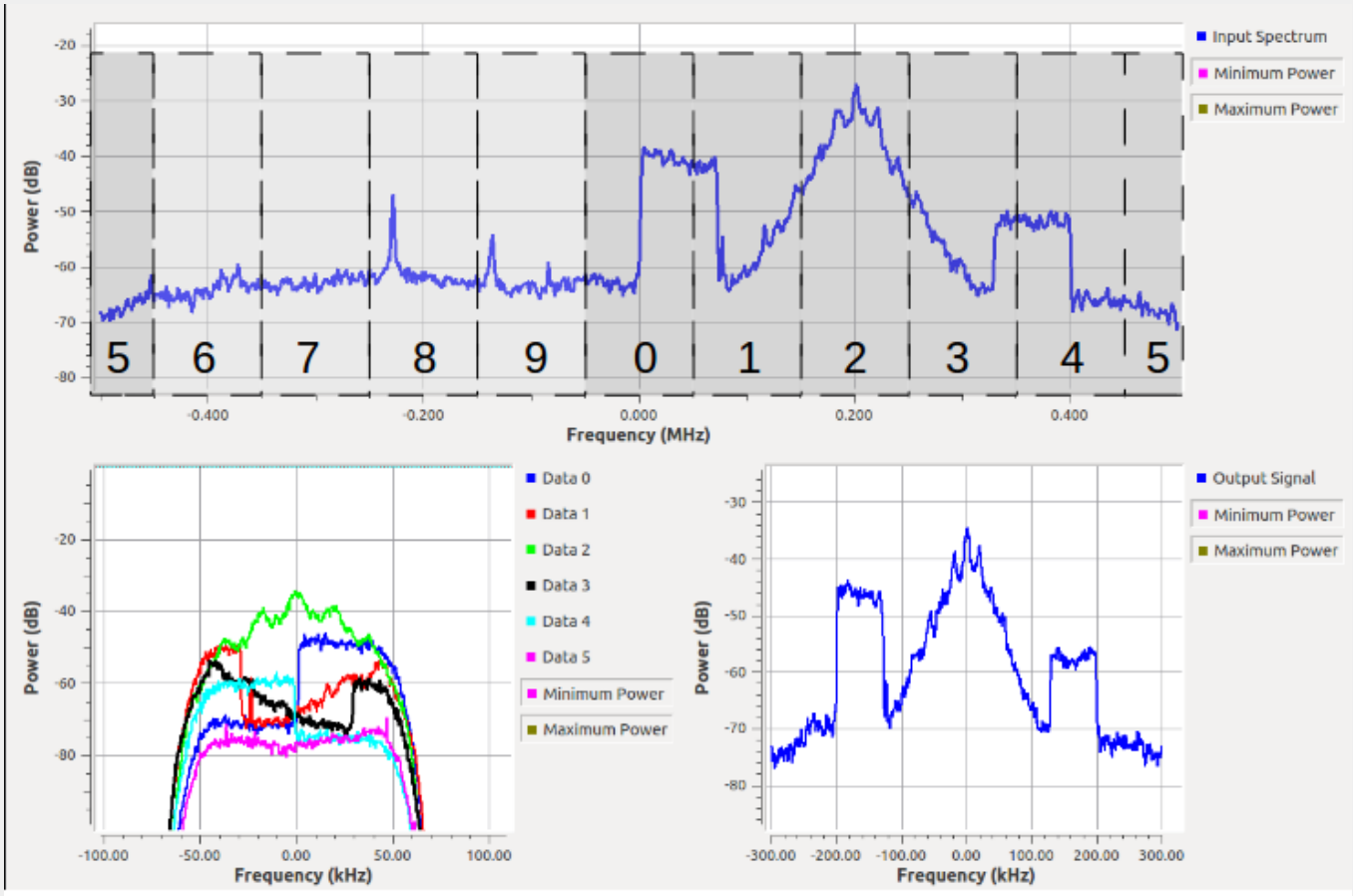
Combining (chans 0-3 and 4,5)



Recombining an FM station split into 10 channels



Recombine channels 0-5



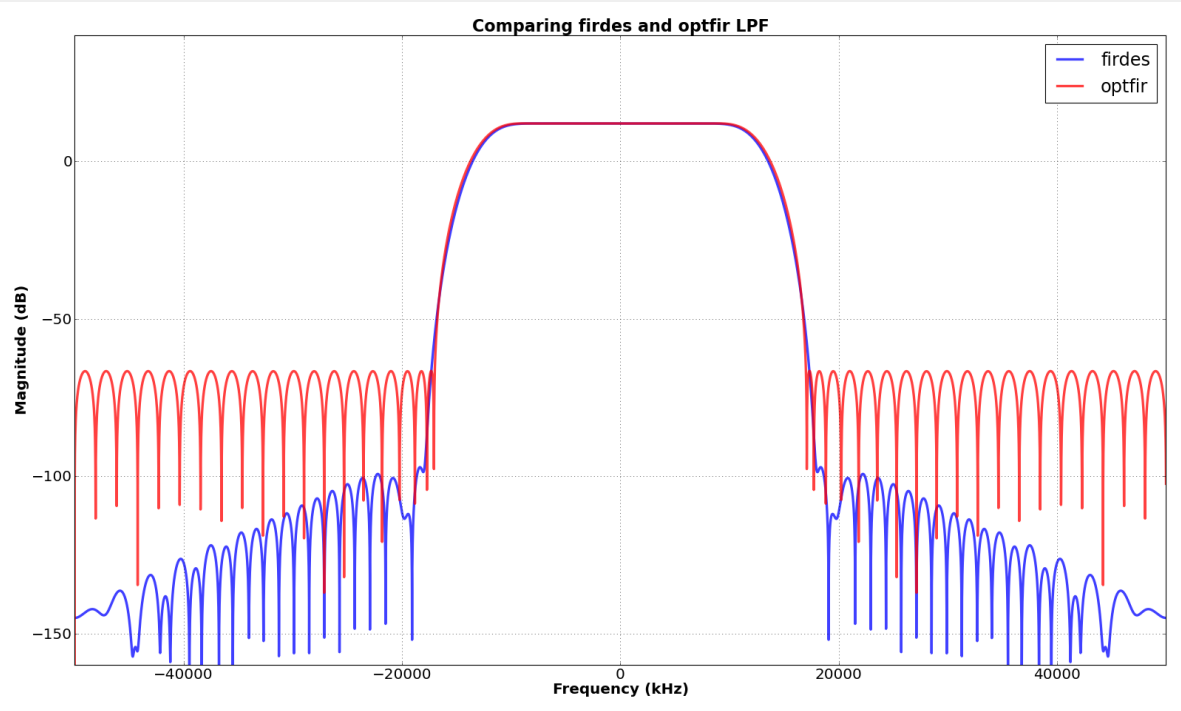
Building the Reconstruction Prototype

- Use firdes filter design tool
  - Other techniques can work; this is a guarantee
- Pass band is 1/2 the channel bandwidth: -6 dB point at 0.5/fs
- Transition band is 1/5 of the channel bandwidth
- For M channels:
  - firdes.low_pass_2(M, M, 0.5, 0.2, 80, window)
  - window can be any window
  - I recommend Blackman-harris

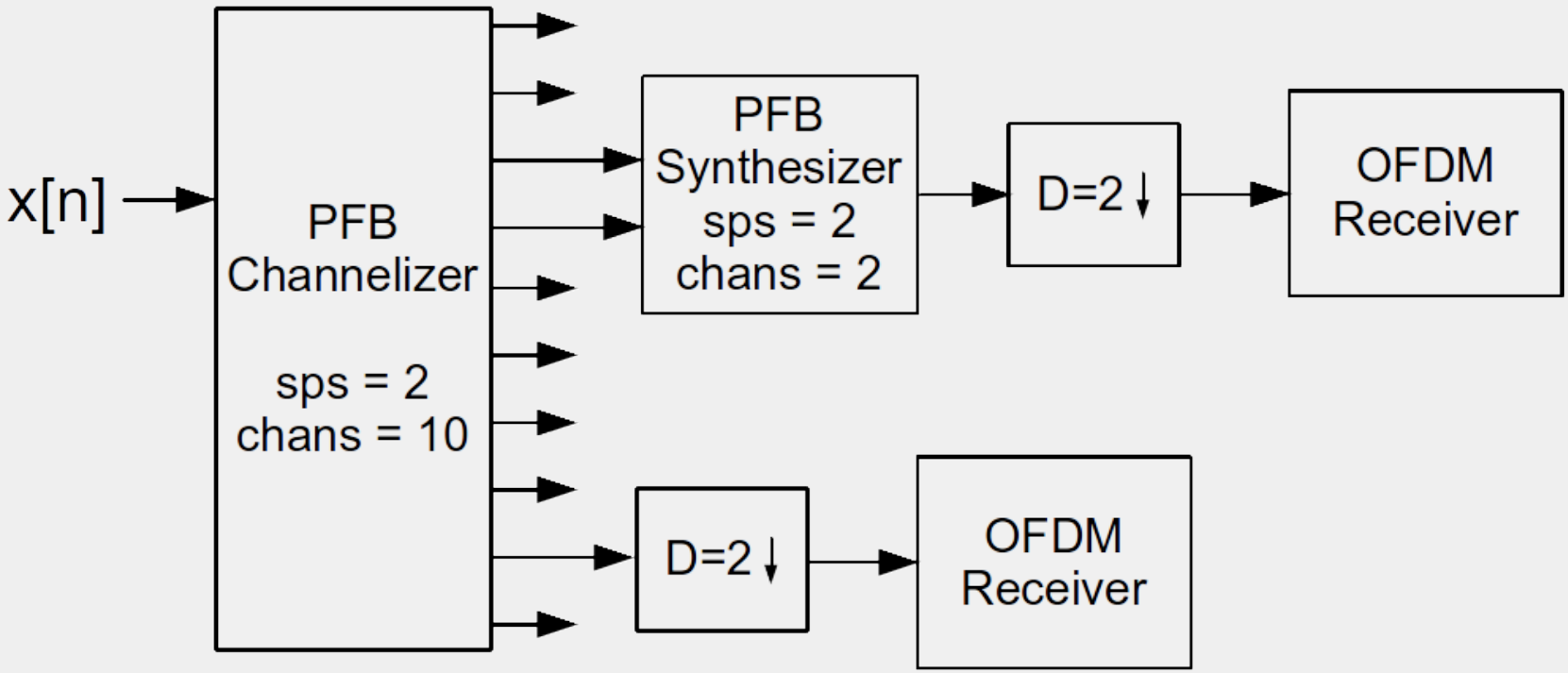
Why firdes?

M = 4

- filter.firdes.low_pass_2(M, M, 0.5, 0.2, 80, filter.firdes.WIN_BLACKMAN_hARRIS) (73 taps)
- filter.optfir.low_pass(M, M, 0.375, 0.305, 0.1, 80) (52 taps)

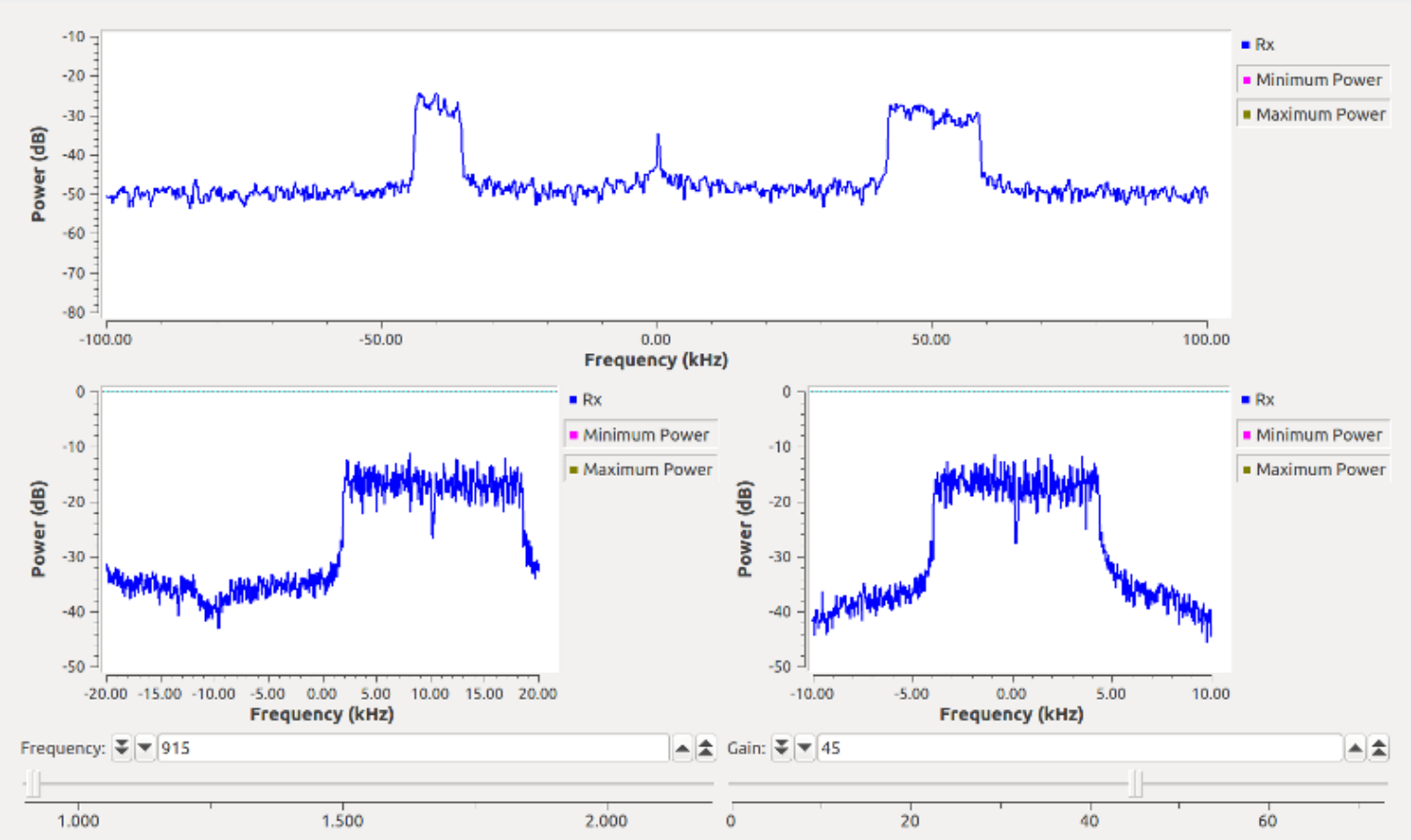


Using a Reconstruction Filterbank

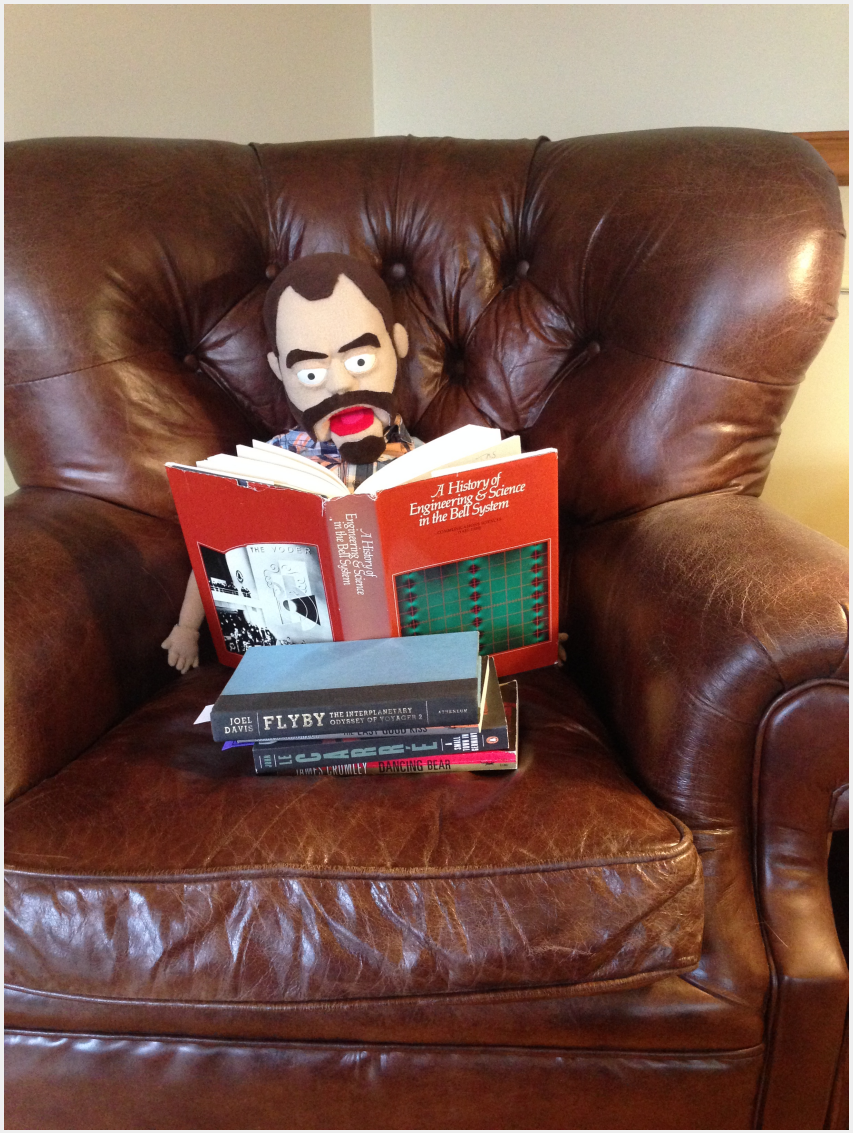


Demoding Both Signals

Received 100% of both channels



Case Studies



Case Study: Signal Detection and Tipping

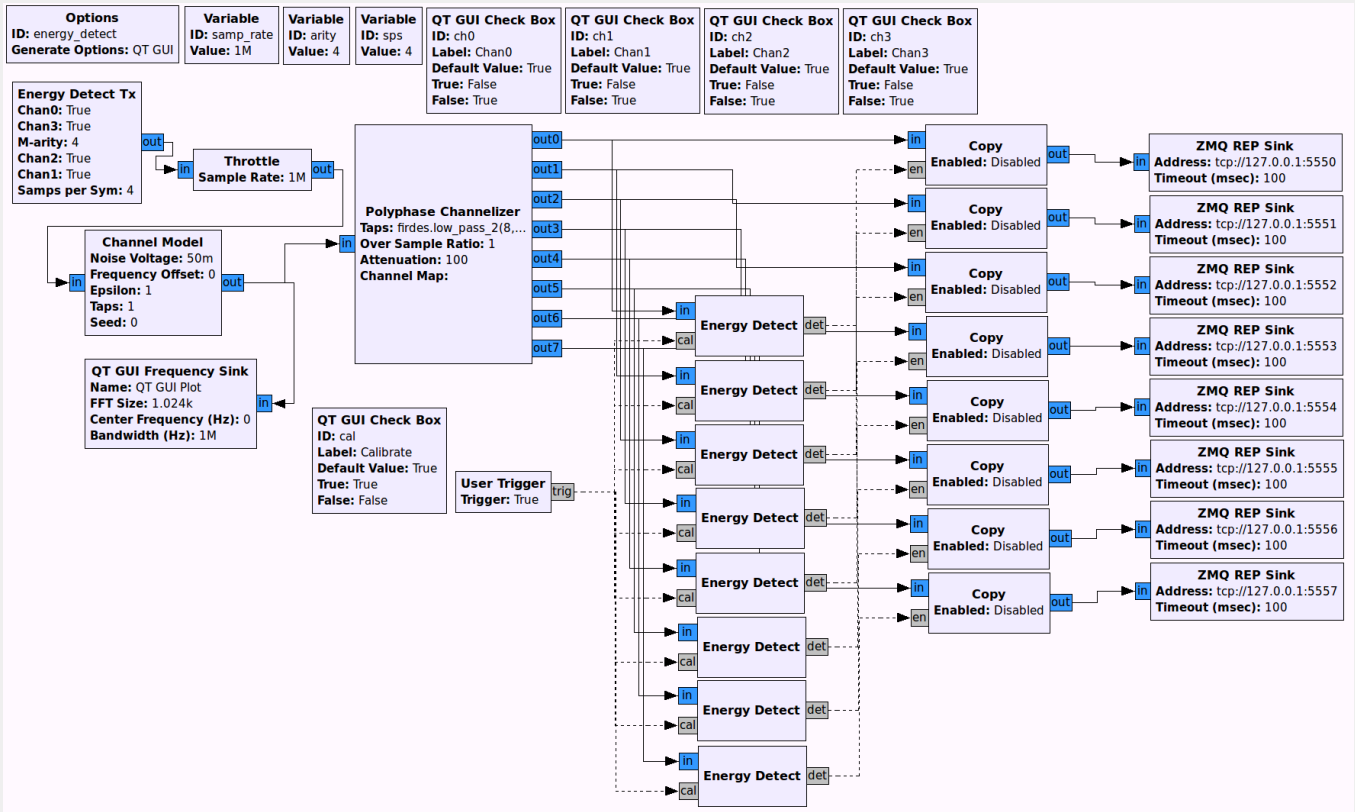
- Common concept for many Cognitive Radio concepts:
  - Detect signals and demodulate.
  - Many signal detection algorithms proposed.
  - Often implemented as energy detection.
- Channelize the spectrum and detect which channels are active.
  - Then tip to the follow-on processing to handle the specific channels.
  - Here, tipping is done using messages to enable a demod path.
  - Demod path enabled over distributed computational model.
- Extend to reconstructing multi-channel behavior.

Energy Detector Setup

- Patched copy block to accept a message.
- Apply copy block's patch to your GNU Radio source:
  - git apply <path-to>/scripts2/blocks_copy_msg.patch
- Rebuild and reinstall GNU Radio.
- Open and build energy_detect_tx.grc; reload GRC blocks.
- Open energy_detect.grc and energy_detect_rx.grc.
- Run energy_detect.grc and energy_detect_rx.grc (in either order).
- Turn off calibration; turn on any of the 4 channels.

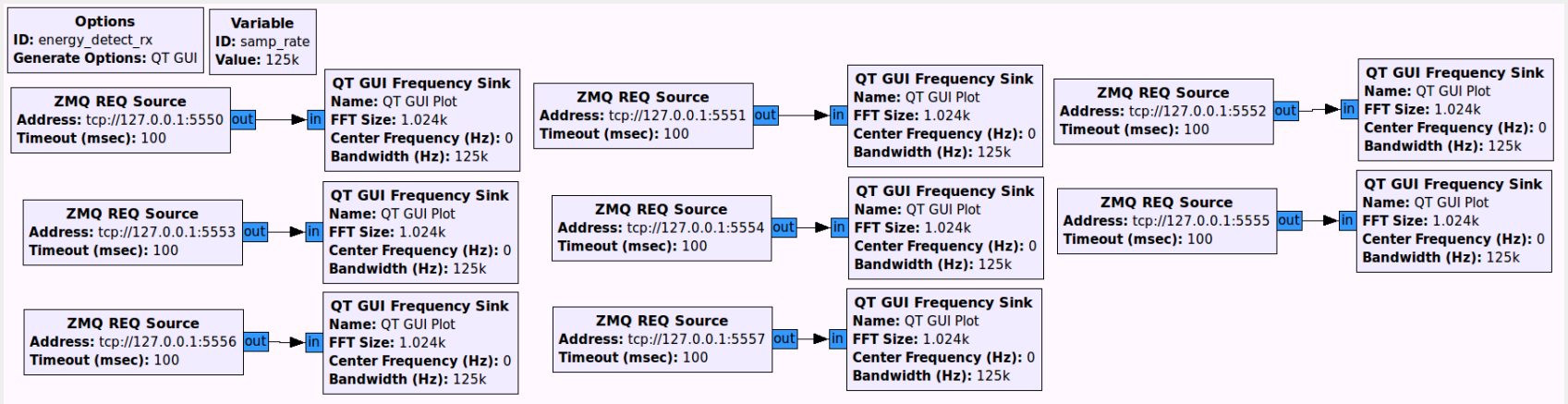
Signal Detection (flowgraph)

Pull in wideband channel ("Energy Detect Tx")



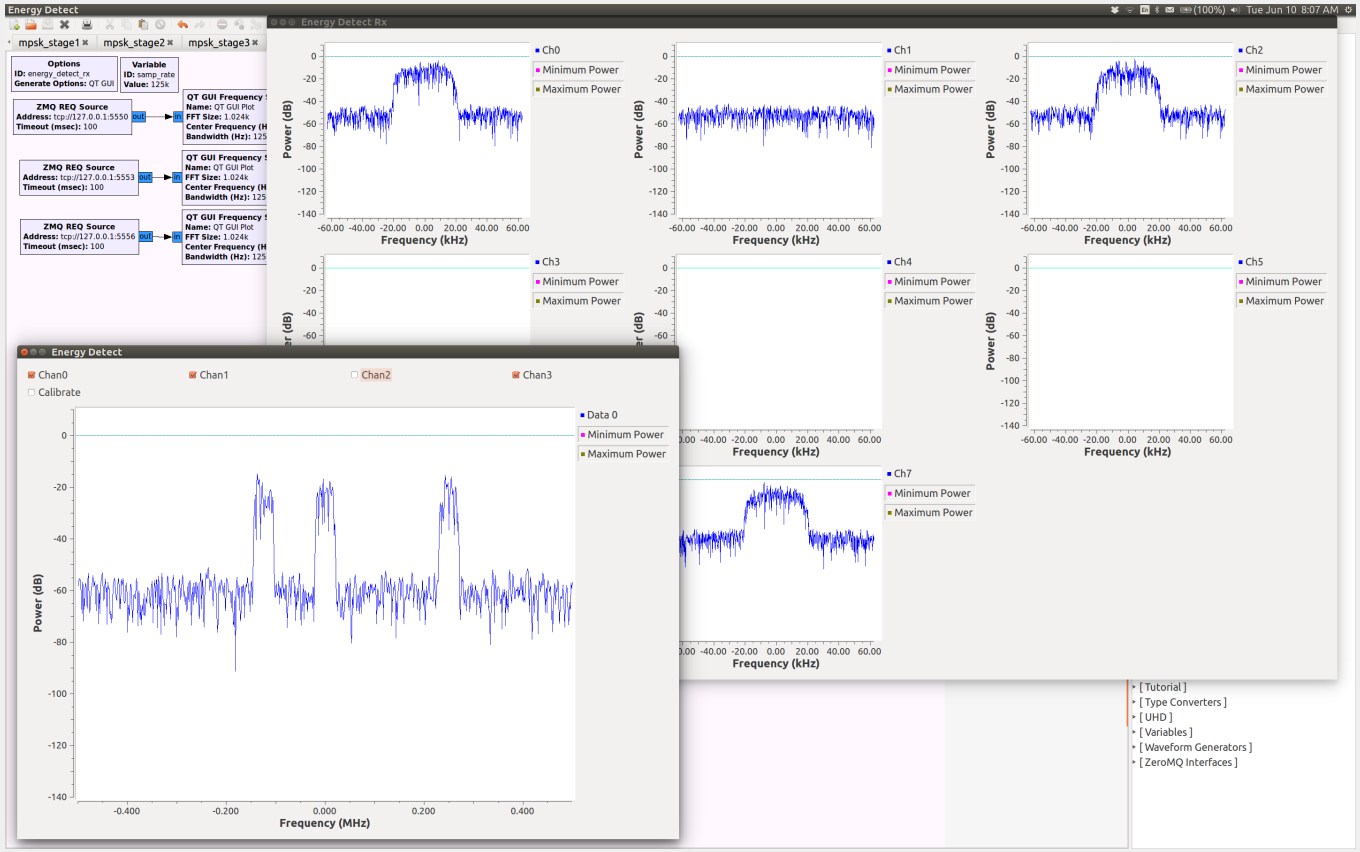
Process Signals

Detected channels can be processed in a distributed system



Signal receiver can process resulting signals

Simple display of the tipped channel



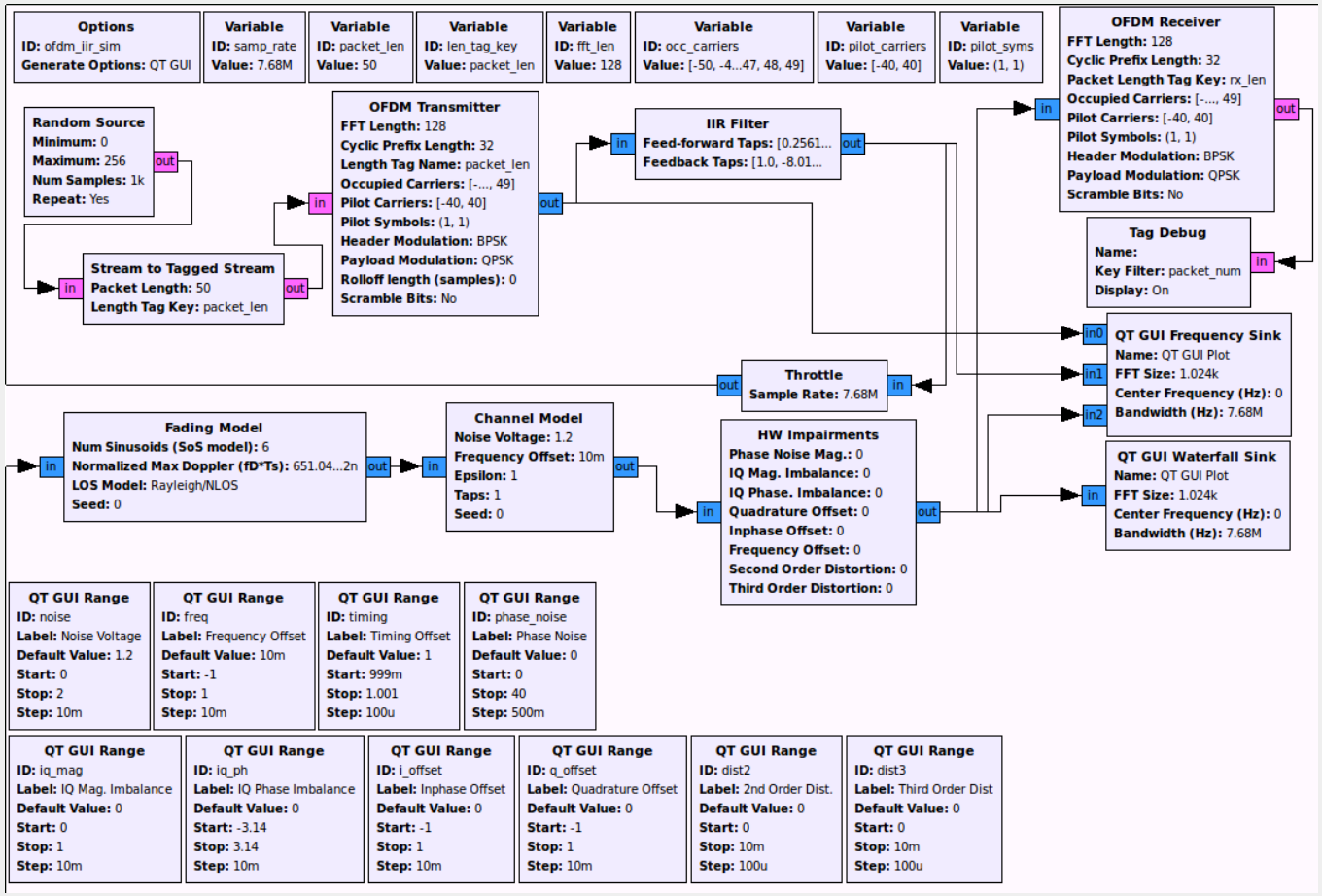
Null OFDM Subcarriers around Other Signals

- Control over OFDM comes in many dimensions.
- Nulling subcarriers to avoid interfering has been popular in the lit.
- Problem is OFDM subcarriers produce high side lobes (sinc function).
- Can we filter the signal to reduce the side lobes?
  - Filtering is done digitally to flexibly set which subcarriers.
  - Does this hold up when passed through real hardware?

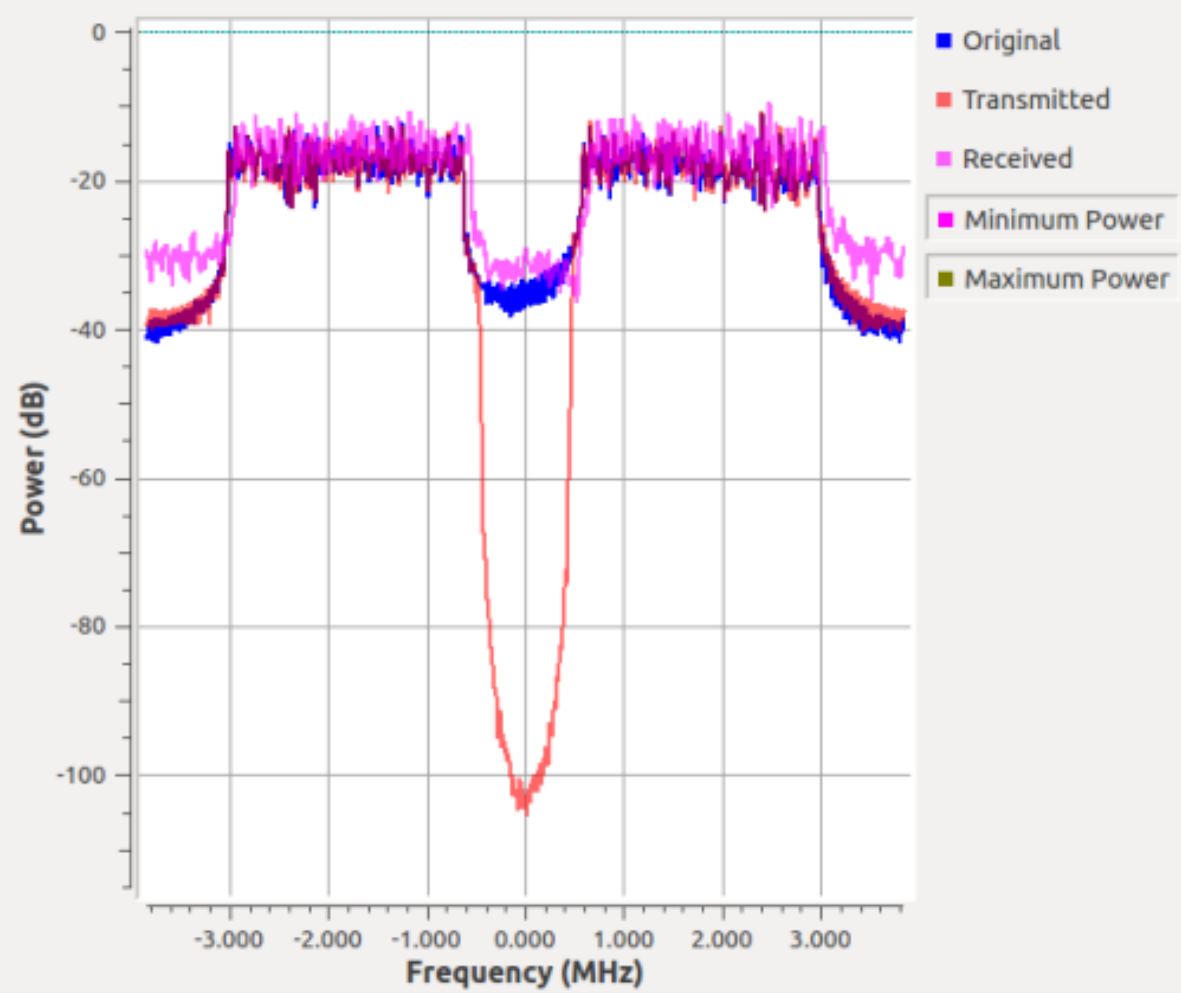
OFDM Nulling with IIR Filters

- Based on a paper published in DySPAN 2014
  - REFERENCE
- Used pre-calculated IIR bandwidths:
  - Combine for different nulling selections.
  - Used IIR to reduce tap count for delay and processing requirements (FPGA).
- Questions:
  - How does OFDM handle the phase distortion from the IIR?
    - Should be corrected in the equalizer.
  - How does it work over real hardware?

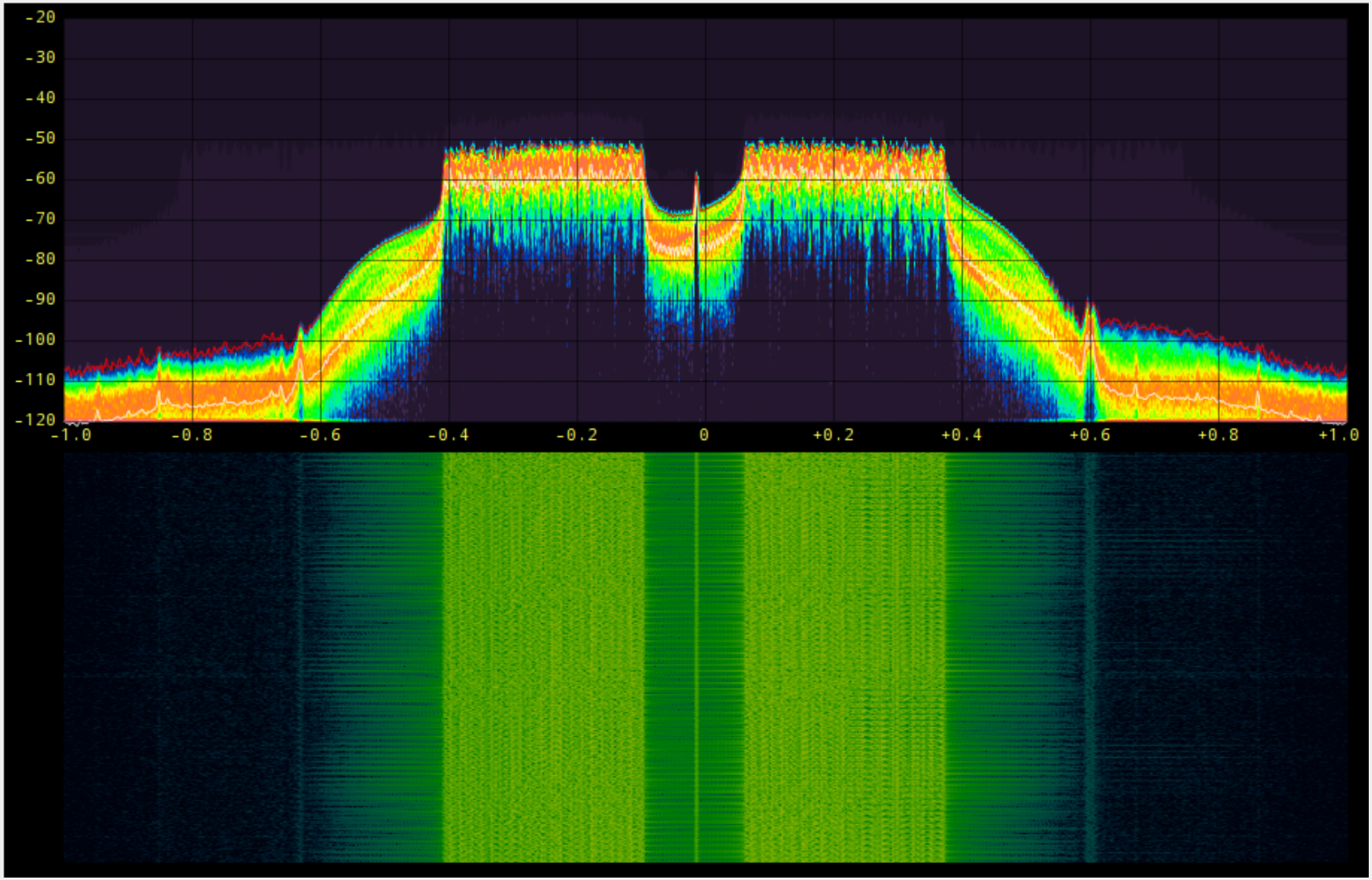
Creating OFDM Nulls



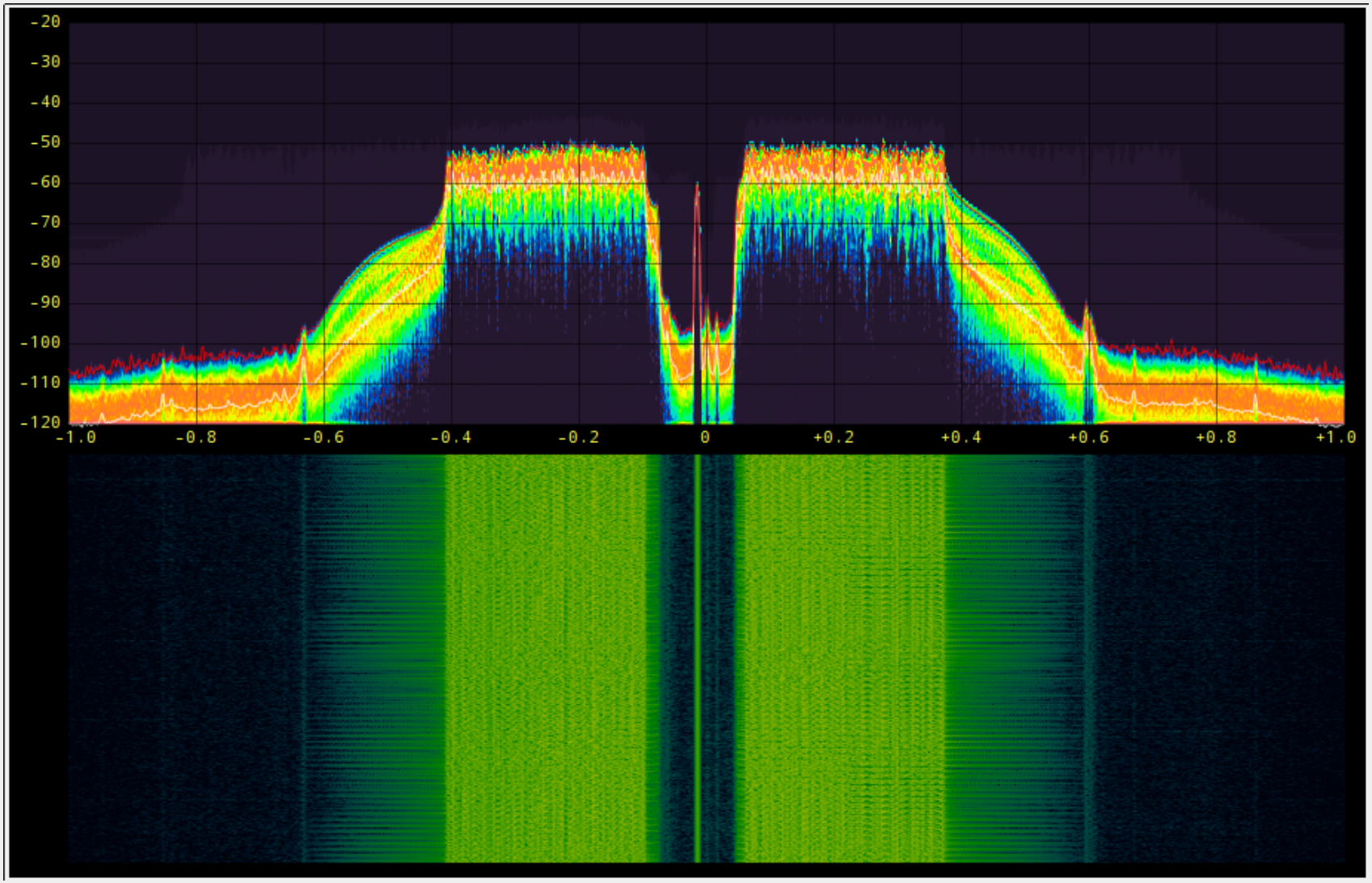
Creating OFDM Nulls: Simulated



Creating OFDM Nulls: OTA w/o IIR



Creating OFDM Nulls: OTA w/o IIR



Creating OFDM Nulls: OTA w/o Two IIR Filters