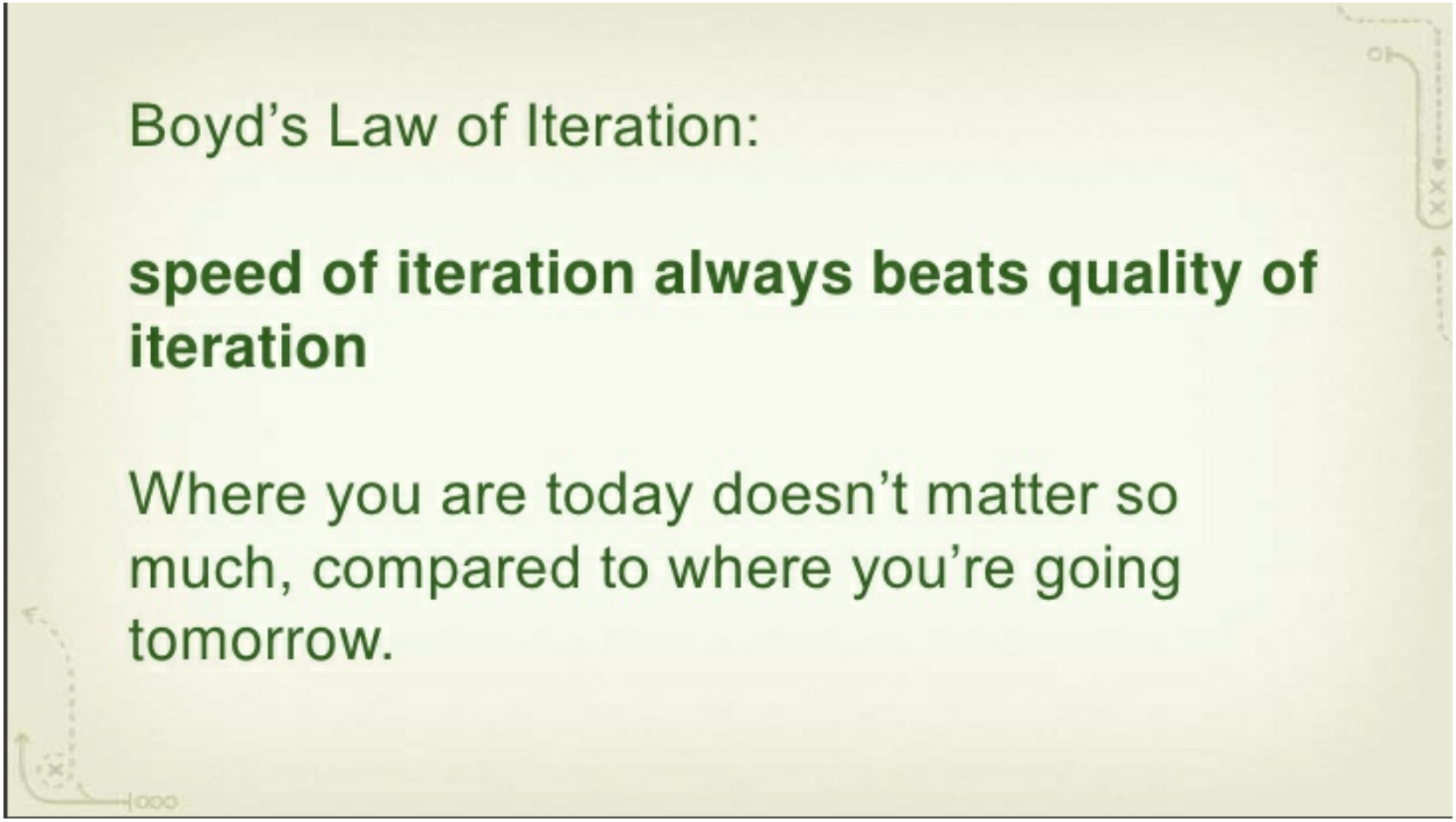
# GNU Radio Channel Simulation

## Trolling sub-par modem algorithms and implementations for fun and profit

Tim O'Shea, Research Faculty
Virginia Polytechnic Institute and University
Arlington, VA
1 Oct 2013

Why simulate channels?




- This is a fundamental strength of **software** radio
- Better simulation allows for rapid iteration

GNU Radio 2012: Channel modeling, where we were...

- Timing offset: Fractional Interpolator at fixed rate (**fixed**)
- Multipath: FIR Filter with fixed taps (**fixed**)
- Freq Offset: Mix with fixed sinusoid generator (**fixed**)
- Noise: Gaussian noise generator and adder (time varying)

This was a good start and allowed testing non-perfectly synchronized signals under a variety of white noise SNR conditions

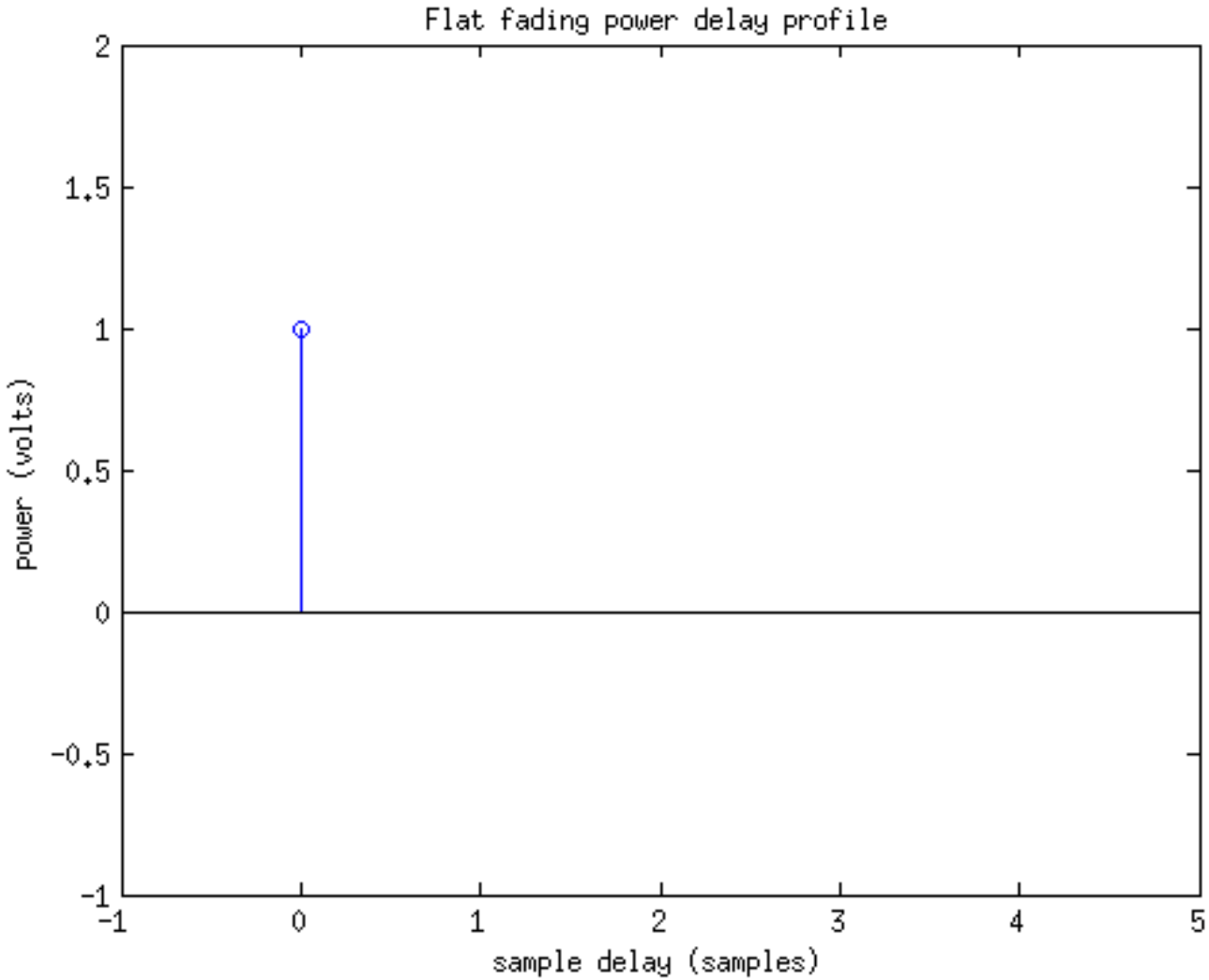
Dynamic behavior matters

- Most modems include some form of error feedback tracking loop for recovery of symbol timing, carrier frequency, and channel response
- Virtually all of these have a fundamental trade off between tracking speed and sensitivity to estimation noise
- Tuning of these algorithms is an important piece of any over the air system
- Static impulsive channel responses rarely exist in any real wireless system, tracking to fixed values is not realistic
- Model the channel coherence time consistently with your target environment
- Do not assume perfect synchronization, do not pass go

Flat fading models

- Provides a single complex tap channel response
- As the name indicates, is flat across the entire simulated band of interest
- Has a non-impulsive autocorrelation function, unlike the AWGN channel
- Allows for testing of synchronization routines through fades/outages

We are still just varying a simple channel amplitude



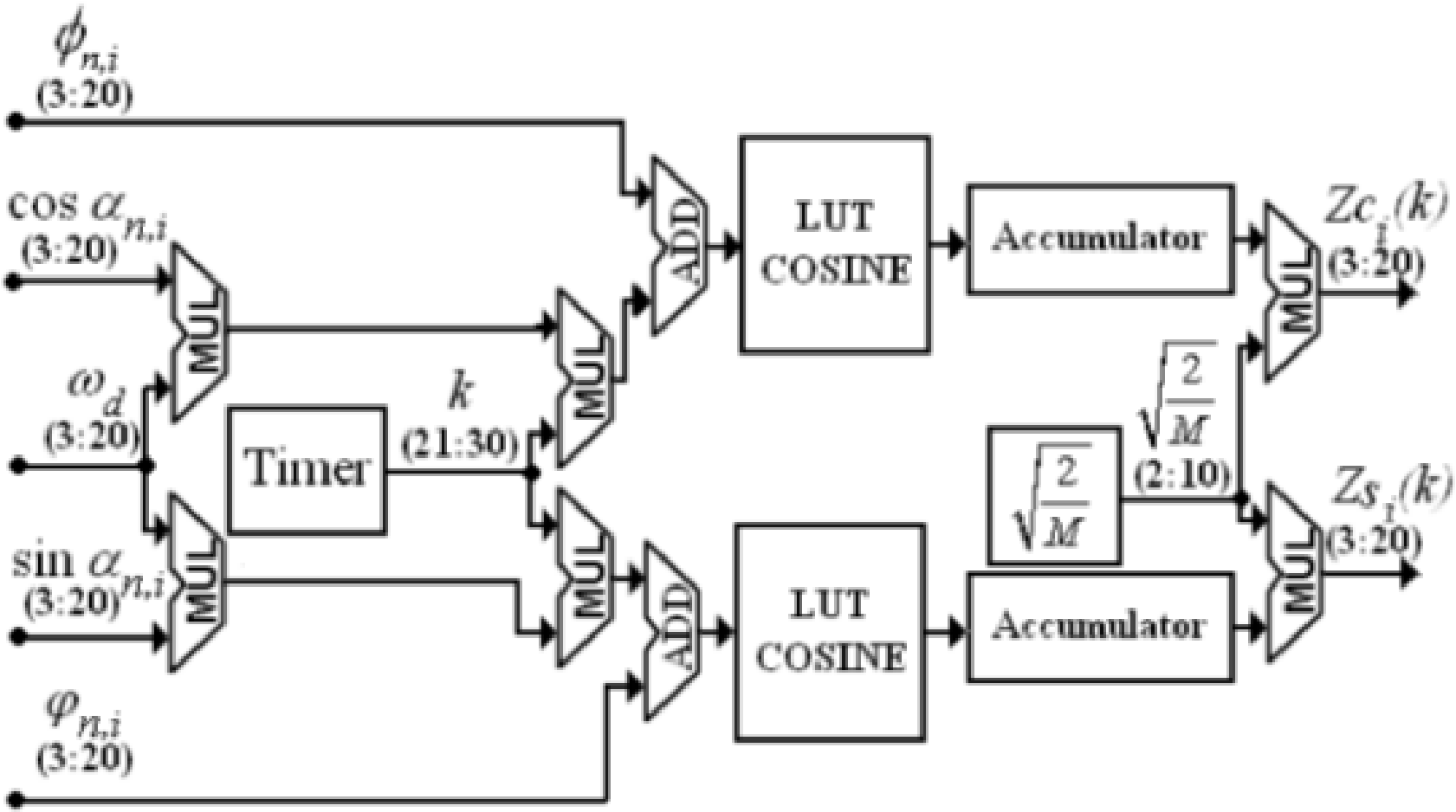
Sum of Sinusoids Implementation of the flat Rayliegh fading model

1. $c_i[m] = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} cos(2\pi f_D T_s m cos(\alpha_n[m]) + \varphi_n)$
2. $c_q[m] = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} cos(2\pi f_D T_s m sin(\alpha_n[m]) + \psi_n)$
3. $\alpha_n[m] = \frac{2\pi n - \pi + \theta[m]}{4N}$
4. random walk parameter $\theta[m] iv \sim Uniform[-\pi, \pi)$ updated each time step with $\delta \sim U(0, \delta_0)$ where $\delta_0 \sim (1e-5, 5e-8)$

use of the random walk parameter allows for reduced number of sinusoids and increased entropy

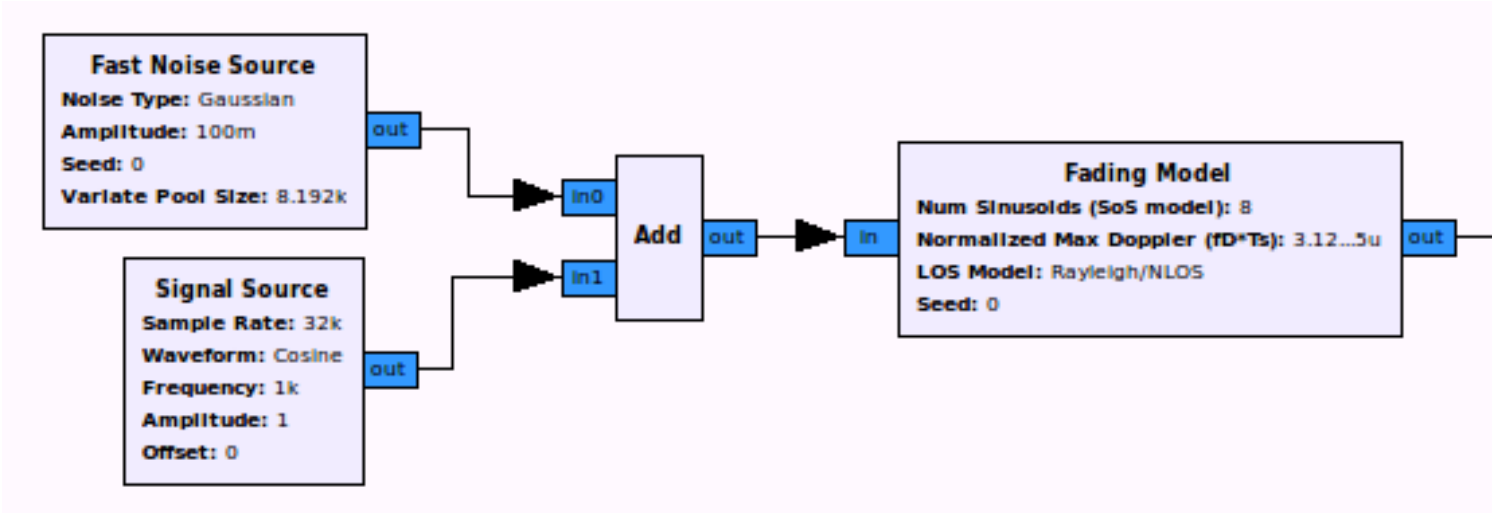
A Rician-K fading model can be produced the same way by simply adding a LOS component

Diagram of flat fader portion

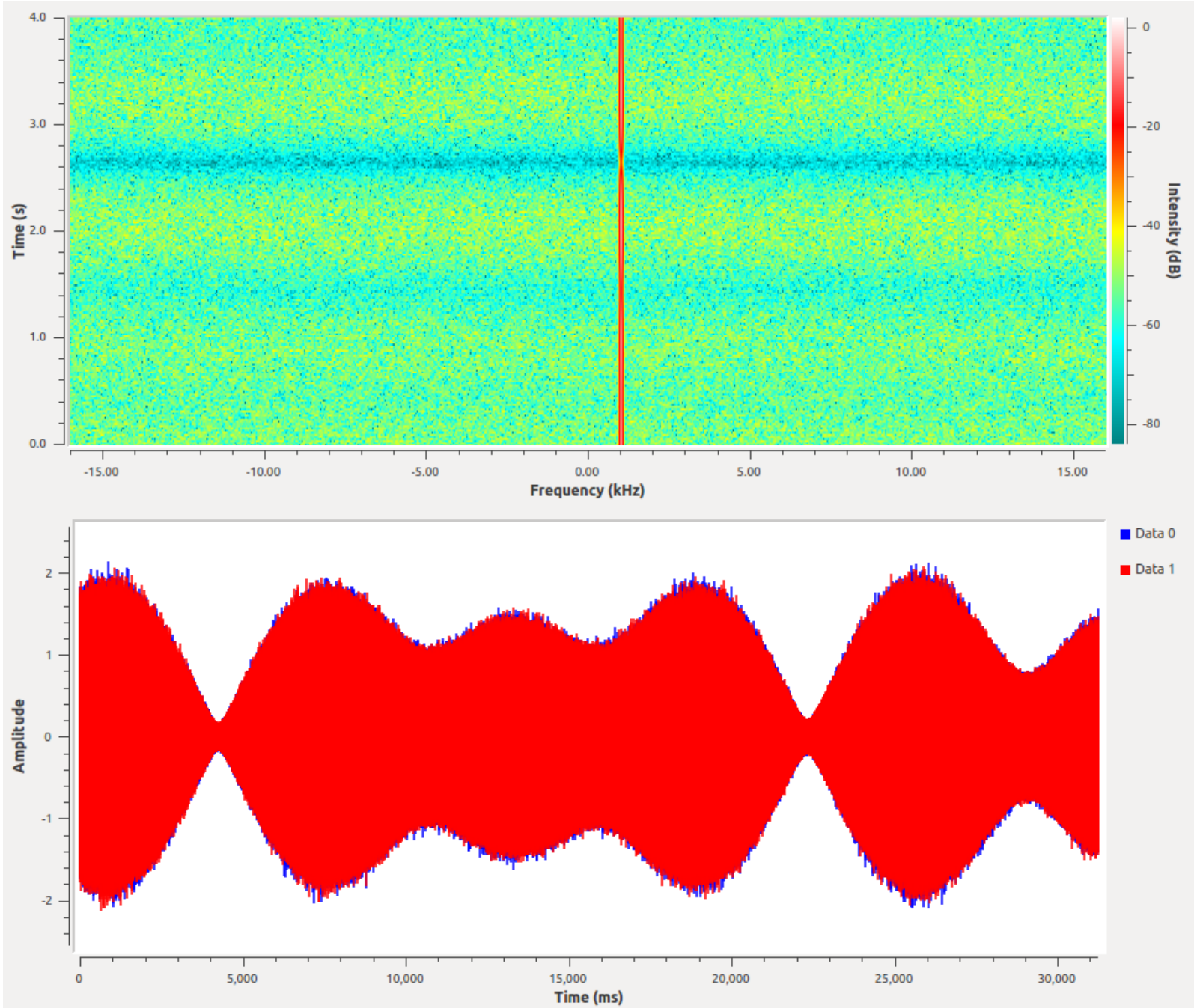



Digram from [1]

# Flat fading model in GRC



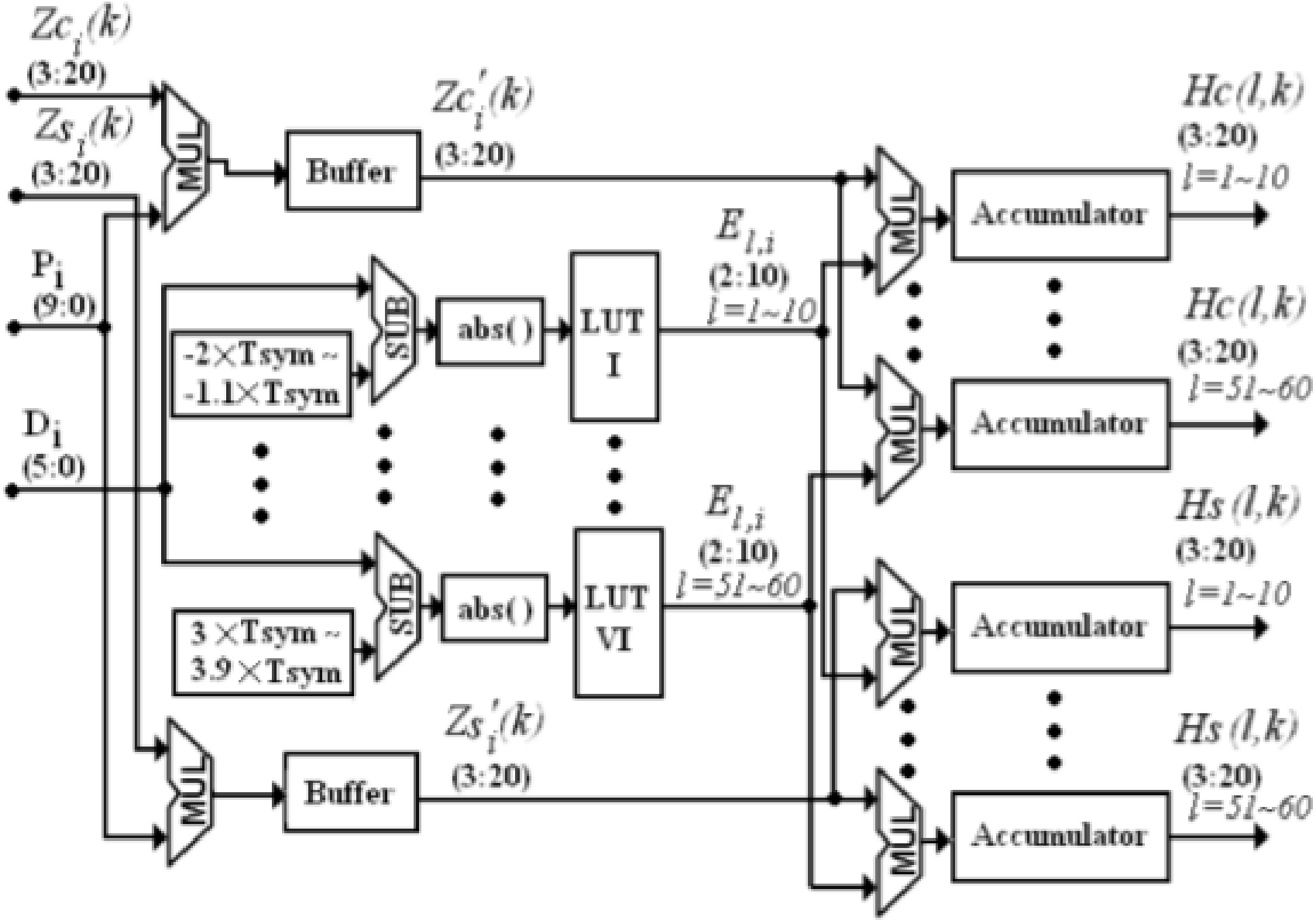
# Flat fading model in GRC



Extension to a frequency selective fading model

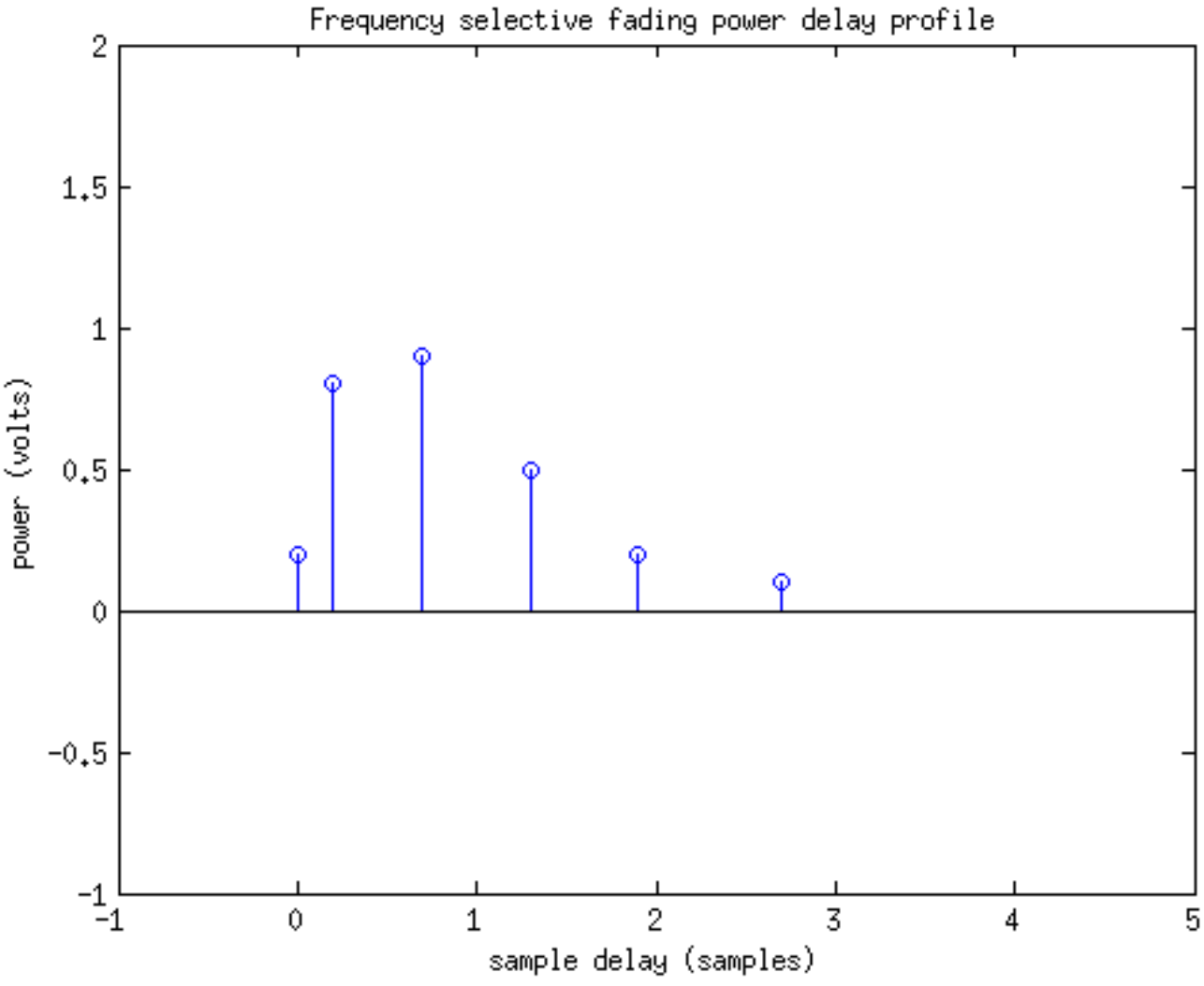
1. define a fixed Power Delay Profile (PDP) in terms of fractional sample times and powers
2. generate a number of flat fading channel sequences
3. interpolate and sum the flat fading responses onto each tap of the PDP to form a set of FIR taps
4. apply the resulting FIR filter to the incoming signal

Diagram of frequency selective fading extension

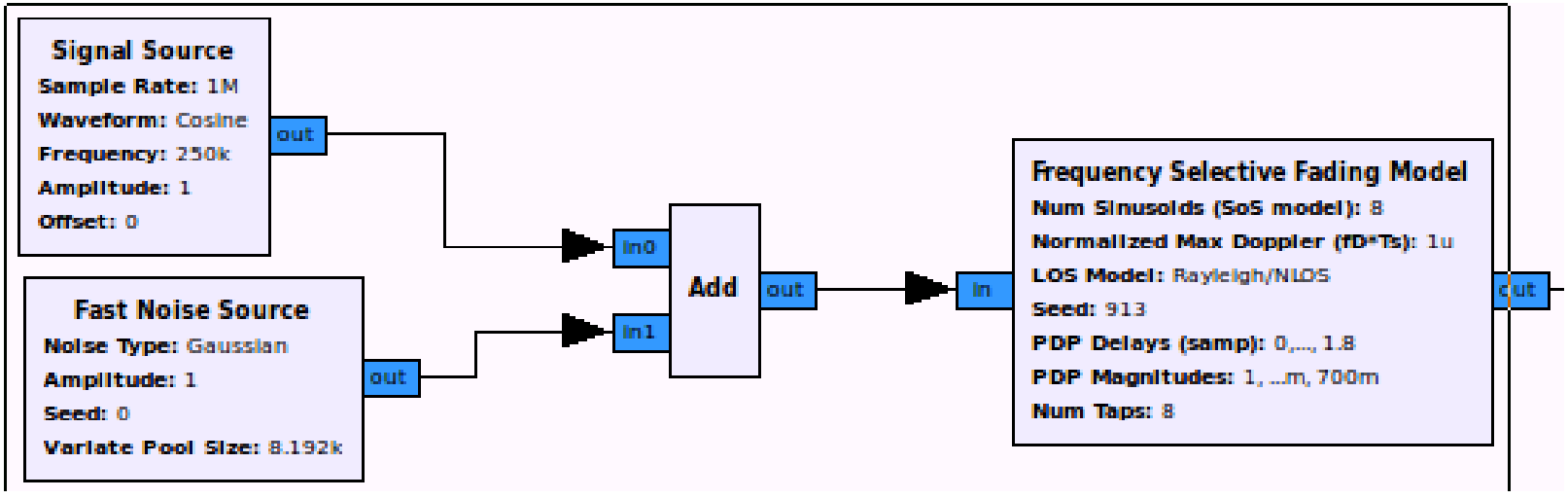



Digram from [1]

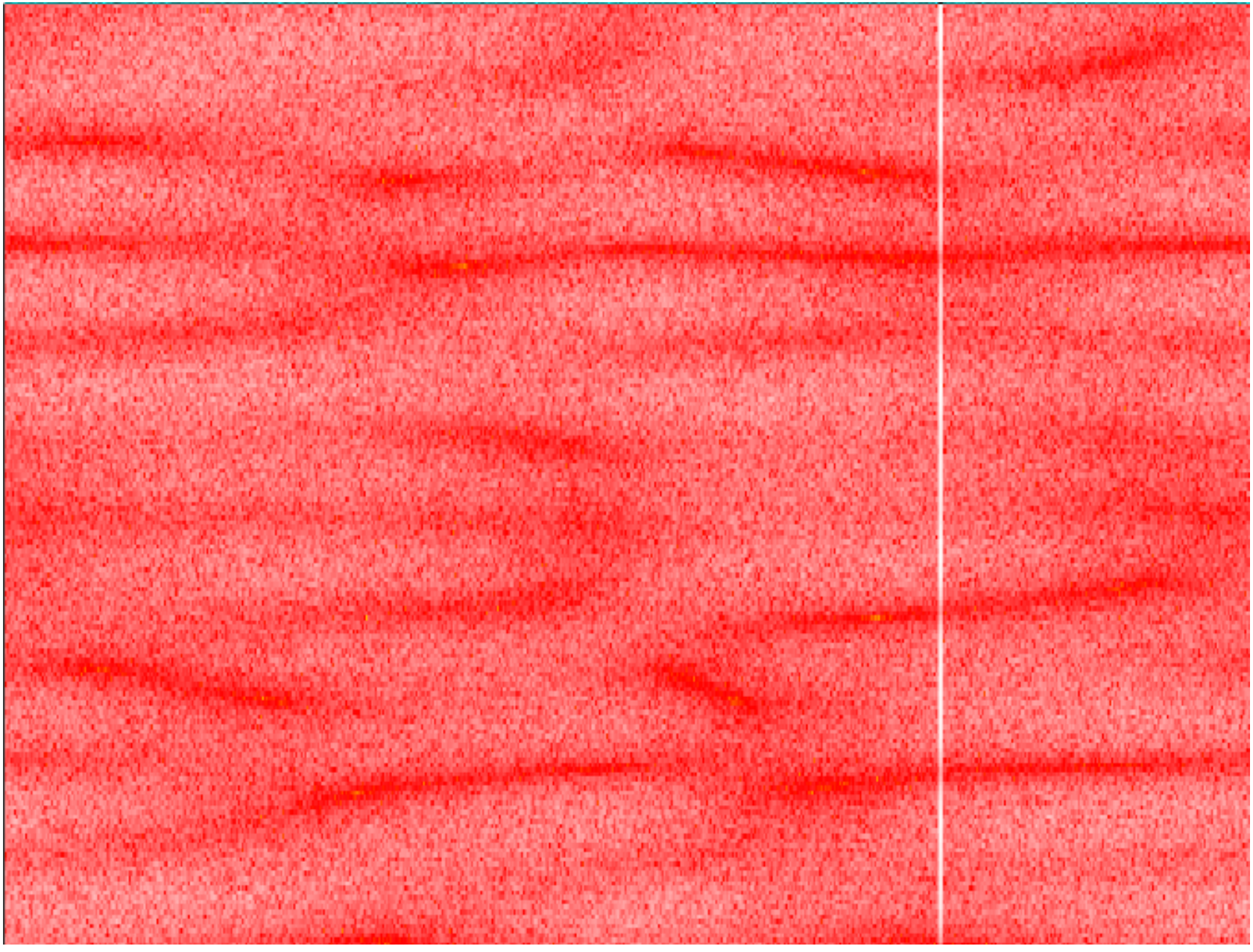
Our PDP can now reflect a more realistic set of discrete powers and delays



# What it looks like in GNU Radio



# What it looks like in GNU Radio



## Performance Optimizations

- Selective model runs 500KSamp/sec on my laptop (i5-2520M) (3 taps, 8 SoS model)
- Highly dependent on number of taps and number of sinusoids
- Fast computation of Sin, Cos and Sinc (done)
- Parallelization of independent flat fading generators (not done)

| Function / Call Stack | CPU Time by Utilization<br>Idle Poor Ok Ideal Over | Ov. an. Spi. | Module | |
|---|---|---|---|---|
| gr::channels::flat_fader_impl::next_sample | 3.978s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | gr::channels::flat_fader_impl::next_sample(void) |
| gr::channels::selective_fading_model_impl::work | 2.993s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | gr::channels::selective_fading_model_impl::work(int, s |
| sincostable::sin | 2.421s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | sincostable::sin(float) |
| sincostable::cos | 2.287s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | sincostable::cos(float) |
| sincostable::sin | 1.737s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | sincostable::sin(float) |
| sincostable::cos | 1.126s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | sincostable::cos(float) |
| sincostable::sinc | 0.693s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | sincostable::sinc(float) |
| operator+=<float> | 0.473s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | operator+=<float> |
| [python2.7] | 0.315s | 0s | python2.7 | [python2.7] |
| _PyImport_GetDynLoadFunc | 0.237s | 0s | python2.7 | _PyImport_GetDynLoadFunc |
| operator+=<float> | 0.227s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | operator+=<float> |
| gr::analog::fastnoise_source_c_impl::sample | 0.200s | 0s | libgnuradio-analog-3.7.2git.so.0.0.0 | gr::analog::fastnoise_source_c_impl::sample(void) |
| sincostable::cos | 0.114s | 0s | libgnuradio-channels-3.7.2git.so.0.0.0 | sincostable::cos(float) |
| memcpy | 0.046s | 0s | libc-2.3.4.so | memcpy |
| gr::fxpt_nco::sincos | 0.040s | 0s | libgnuradio-analog-3.7.2git.so.0.0.0 | gr::fxpt_nco::sincos(std::complex<float>*, int, double) |

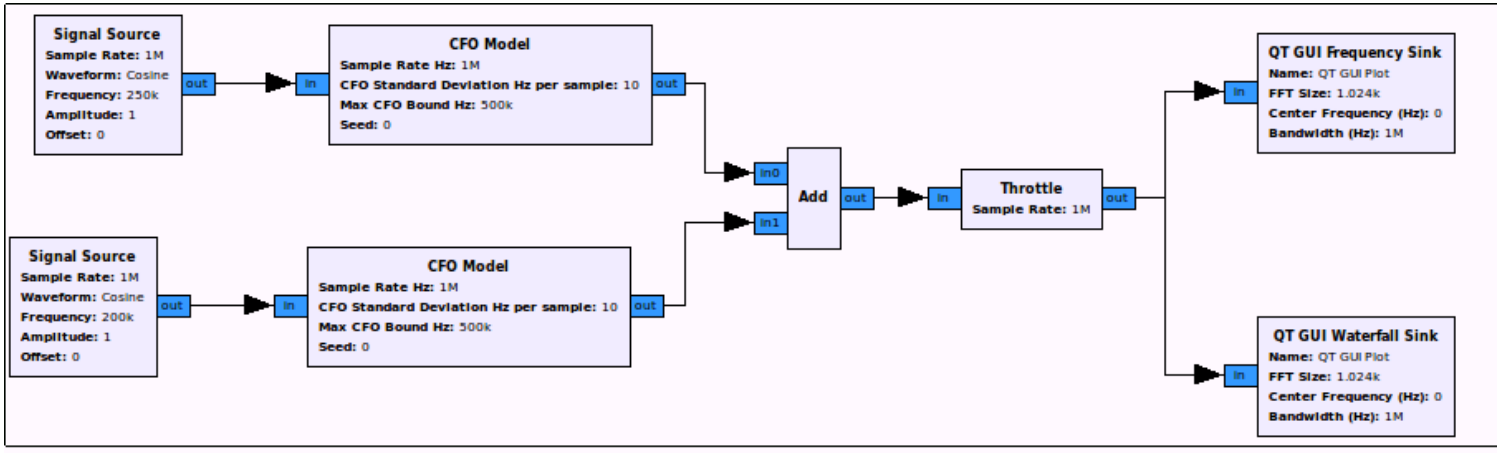
## A Standard set of Power Delay Profiles for GNU Radio

- Most wireless standards specify a set of channel models for compliance testing
- These typically represent expected channel responses in a variety of different environments: rural, urban, dense urban, etc
- Shouldn't we have a similar thing in GNU Radio?
- Future work building a small library of "common" channel conditions for rapid representative system testing
- Would include expected PDP model, LO and sample rate stability for device class
- Allow for consistent rapid and reproducible comparison of performance

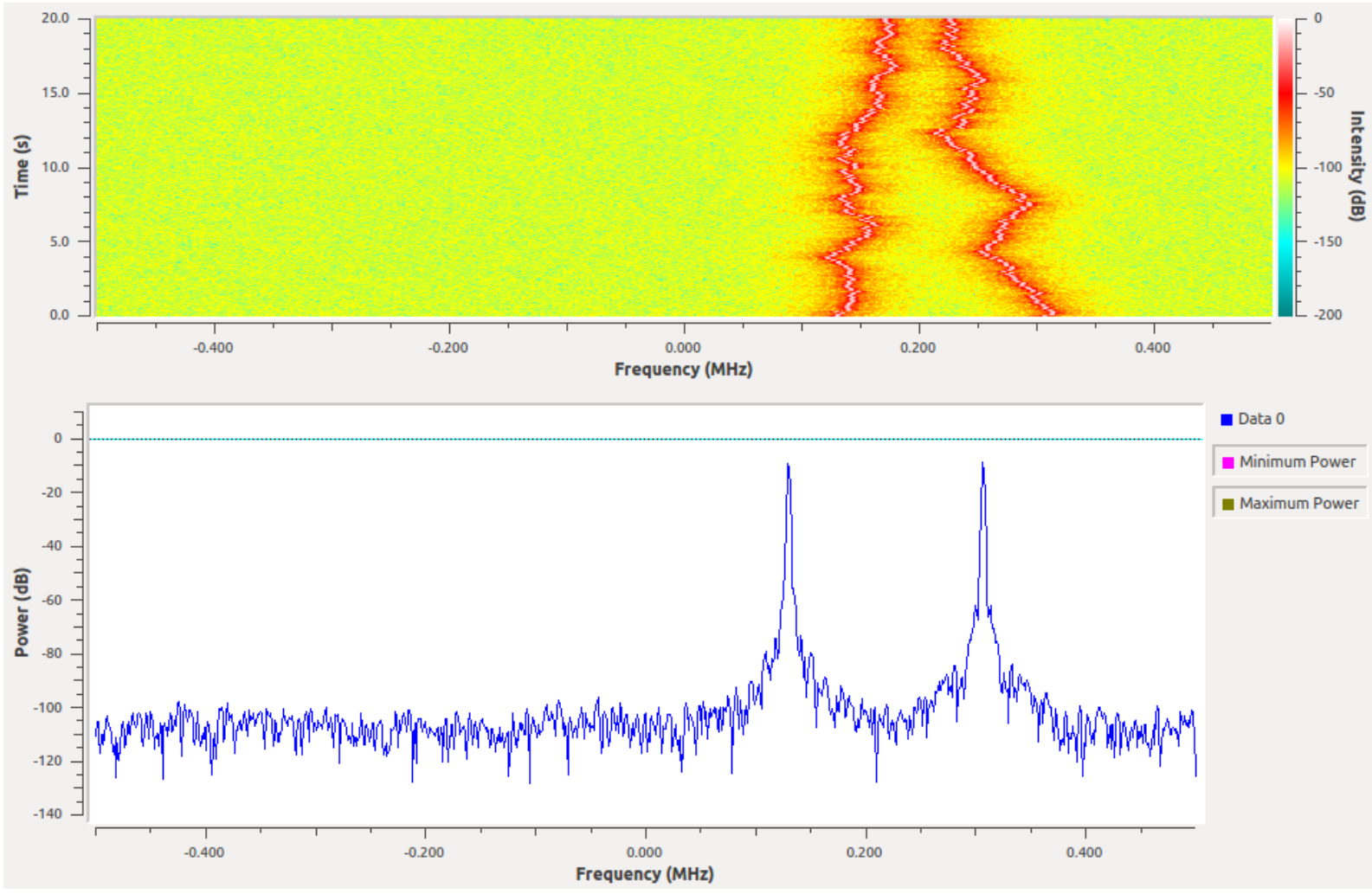
Simulating dynamic oscillator drift (carrier frequency offset (cfo) and sample rate offset (sro))

- Simply introduce a random walk parameter
- Apply updates to oscillator frequency for carrier frequency offset
- Apply updates to interpolation rate in fractional interpolator for sample rate offset

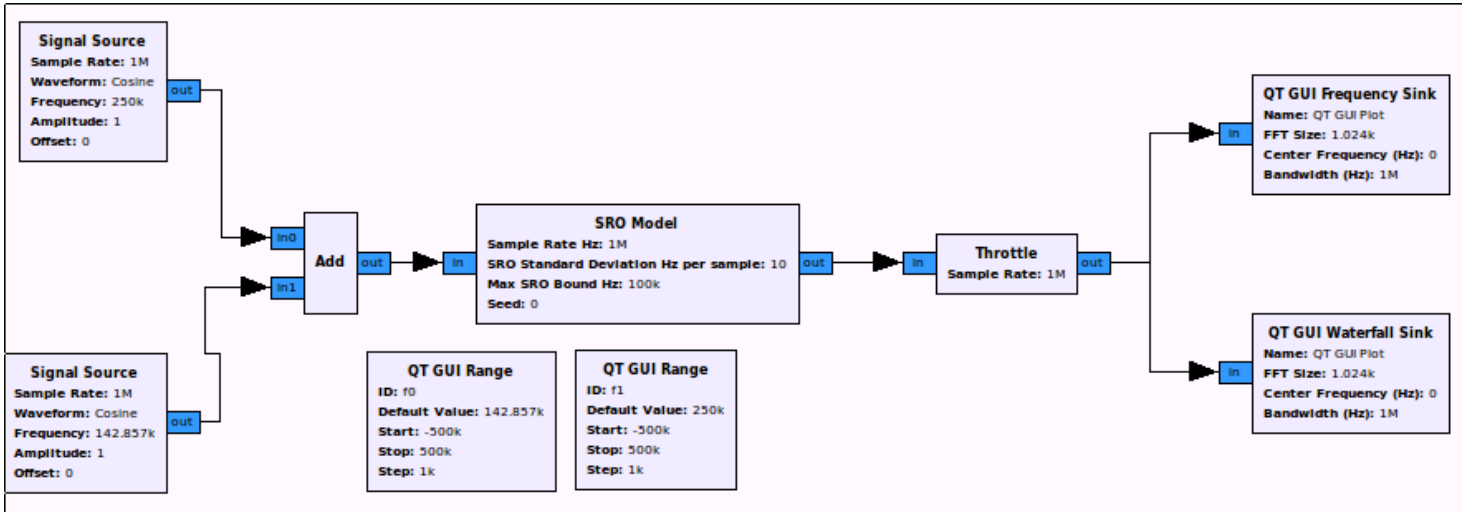
# Simulating Center Frequency Offset in GRC



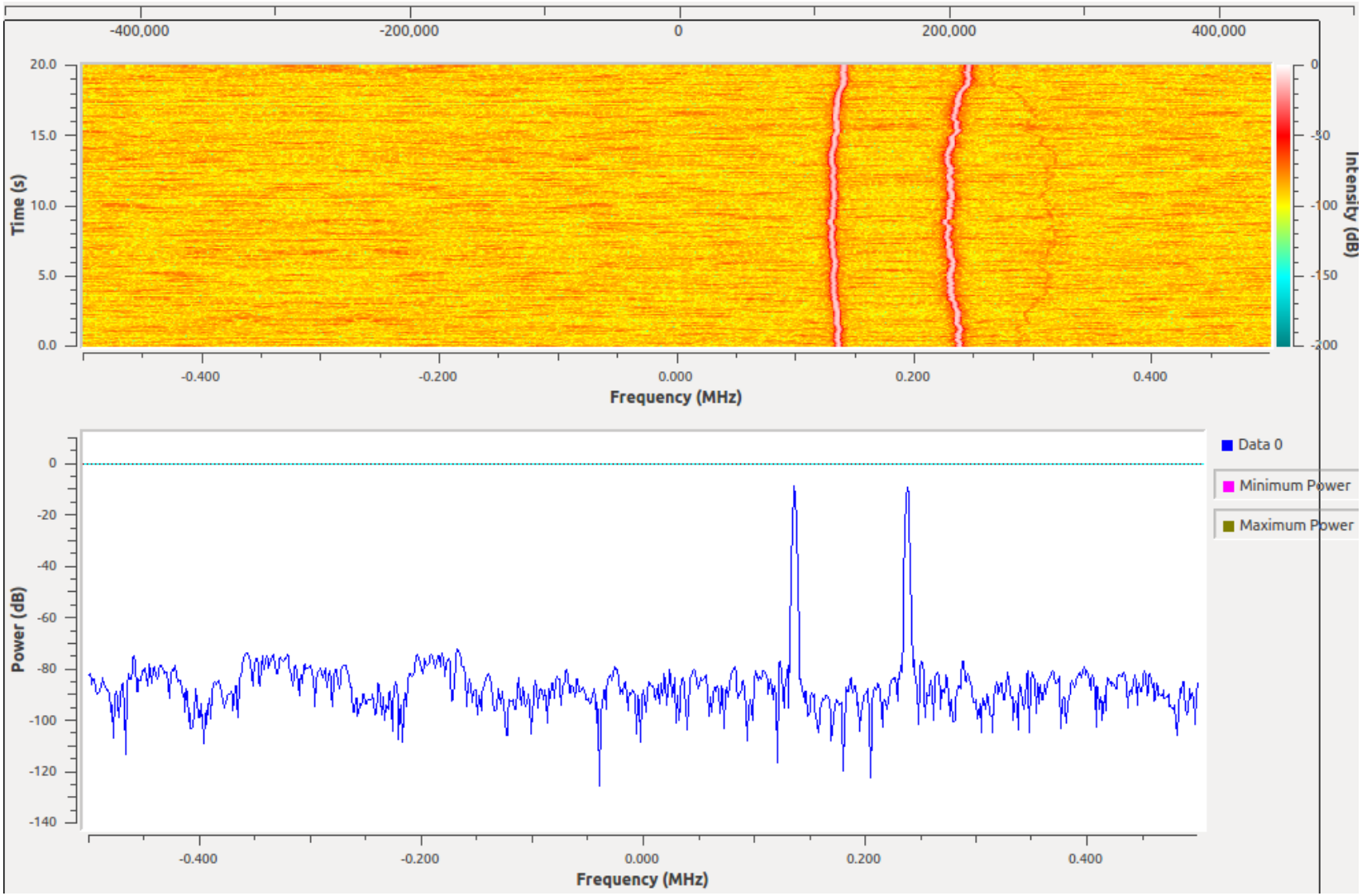
# Simulating Center Frequency Offset in GRC



# Simulating Sample Rate Offset in GRC



# Simulating Sample Rate Offset in GRC

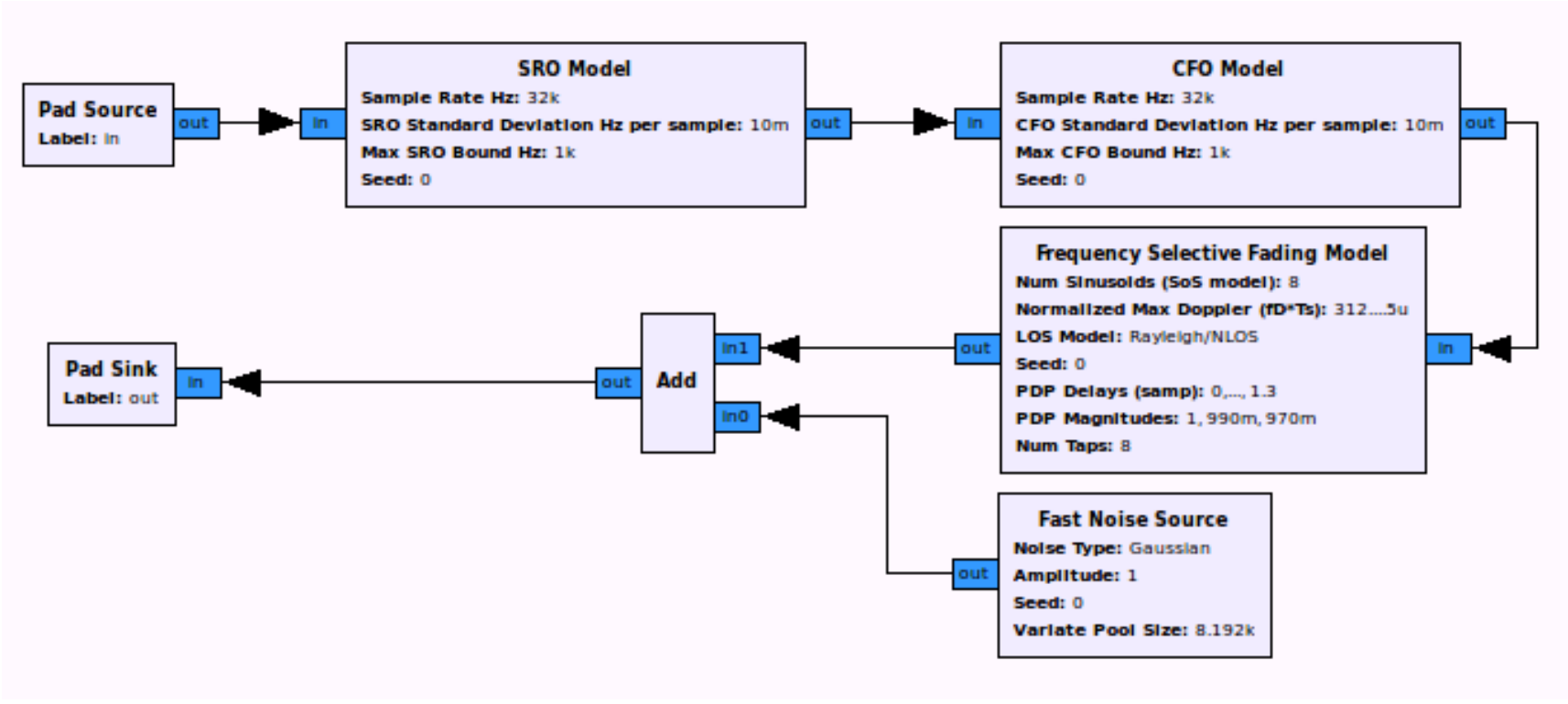


Both currently use the "fastnoise" gaussian noise generator.

- Generates a pool of gaussian random variates with zero mean and variance proportional to the clock stability
- Randomly samples from the pools using a fast uniform random variate
- Applies a random sign perturbation to remove any bias present in the pool

Putting it all together, the **Dynamic Channel Model** block

- Hier block that includes all of these dynamic behaviors
- Implemented as a C++ block but logically shown below



Most of the channel components have been merged into GNU Radio

Thanks for using them!

References:


1. A LOW-COMPLEXITY HARDWARE IMPLEMENTATION OF DISCRETE-TIME FREQUENCY-SELECTIVE RAYLEIGH FADING CHANNELS, Fei Ren and Yahong R. Zheng
2. Compact Rayleigh and Rician fading simulator based on random walk processes, A. Alimohammad S.F. Fard B.F. Cockburn C. Schlegel