# **GNU Radio Python Flow Graph: Underwater Low Frequency Signal Processing**

## **Overview**

This code implements a GNU Radio flow graph for processing underwater low frequency signals. It creates a software-defined radio (SDR) application with a graphical user interface using PyQt5 that simulates, filters, and visualizes signals.

## **Key Components**

### **Signal Generation**

**Cosine Wave Source**: Generates a cosine wave at 50 kHz with amplitude 0.5  
 self.analog\_sig\_source\_x\_0 = analog.sig\_source\_c(samp\_rate, analog.GR\_COS\_WAVE, 50e3, 0.5, 0, 0)

**Noise Source**: Adds Gaussian noise with amplitude 0.1 to simulate real-world conditions  
 self.analog\_noise\_source\_x\_0 = analog.noise\_source\_c(analog.GR\_GAUSSIAN, 0.1, 0)

**Signal Adder**: Combines the cosine wave and noise into a single complex signal  
 self.blocks\_add\_xx\_0 = blocks.add\_vcc(1)

### **Signal Processing**

* **Frequency-Translating FIR Filter**: A critical component that:  
  + Applies low-pass filtering (cutoff at 1.5 kHz)
  + Frequency shifts the signal by -48.5 kHz
  + Uses a Hamming window for better frequency response

self.freq\_xlating\_fir\_filter\_xxx\_0 = filter.freq\_xlating\_fir\_filter\_ccc(1, firdes.low\_pass(

gain=1.0, # Amplitude gain

sampling\_freq=200e3, # Sample rate (200 kHz)

cutoff\_freq=1.5e3, # Cutoff frequency (1.5 kHz)

transition\_width=500, # Transition width (500 Hz)

window=window.WIN\_HAMMING # Hamming window

), (-48500), samp\_rate)

**Complex to Float Converter**: Converts complex signal to float for audio output  
  
 self.blocks\_complex\_to\_float\_0 = blocks.complex\_to\_float(1)

### **Visualization Tools**

* **QT GUI Sink**: Main visualization widget with multiple views:  
  + Frequency spectrum
  + Waterfall display
  + Time domain
  + Constellation diagram
* **QT GUI Frequency Sink**: Dedicated frequency spectrum analyzer with settings:  
  + 1024-point FFT size
  + Blackman-Harris window
  + Y-axis range from -140 to 10 dB
  + Updated every 0.1 seconds
* **QT GUI Constellation Sink**: Shows the I/Q constellation diagram of the processed signal

### **Audio Output**

**Audio Sink**: Routes the processed signal to the system's audio output  
 self.audio\_sink\_0 = audio.sink(48000, '', True)

## **Signal Flow**

1. Generate cosine wave (50 kHz carrier)
2. Generate Gaussian noise
3. Add both signals together
4. Apply frequency-translating FIR filter
   * Shifts frequency by -48.5 kHz (bringing the 50 kHz signal close to baseband)
   * Low-pass filters with 1.5 kHz cutoff
5. Convert complex signal to real/float
6. Send to audio output at 48 kHz sample rate
7. Simultaneously send to visualization tools

## **Sample Rate**

* Main sample rate: 200 kHz for signal processing
* Audio output sample rate: 48 kHz (standard audio rate)

## **Application Structure**

* Inherits from gr.top\_block and Qt.QWidget for functional and UI capabilities
* Uses Qt's signal-slot mechanism for event handling
* Implements proper signal handling for clean shutdown
* Saves and restores window geometry between sessions

## **Running the Application**

When executed, this script:

1. Creates a Qt application
2. Initializes the flow graph
3. Starts signal processing
4. Displays the GUI
5. Sets up signal handlers for clean termination
6. Enters the Qt event loop

This application effectively simulates underwater acoustic signal processing with real-time visualization and audio monitoring capabilities.