Lab 3: Introduction to Software Defined Radio and GNU Radio

1 Introduction

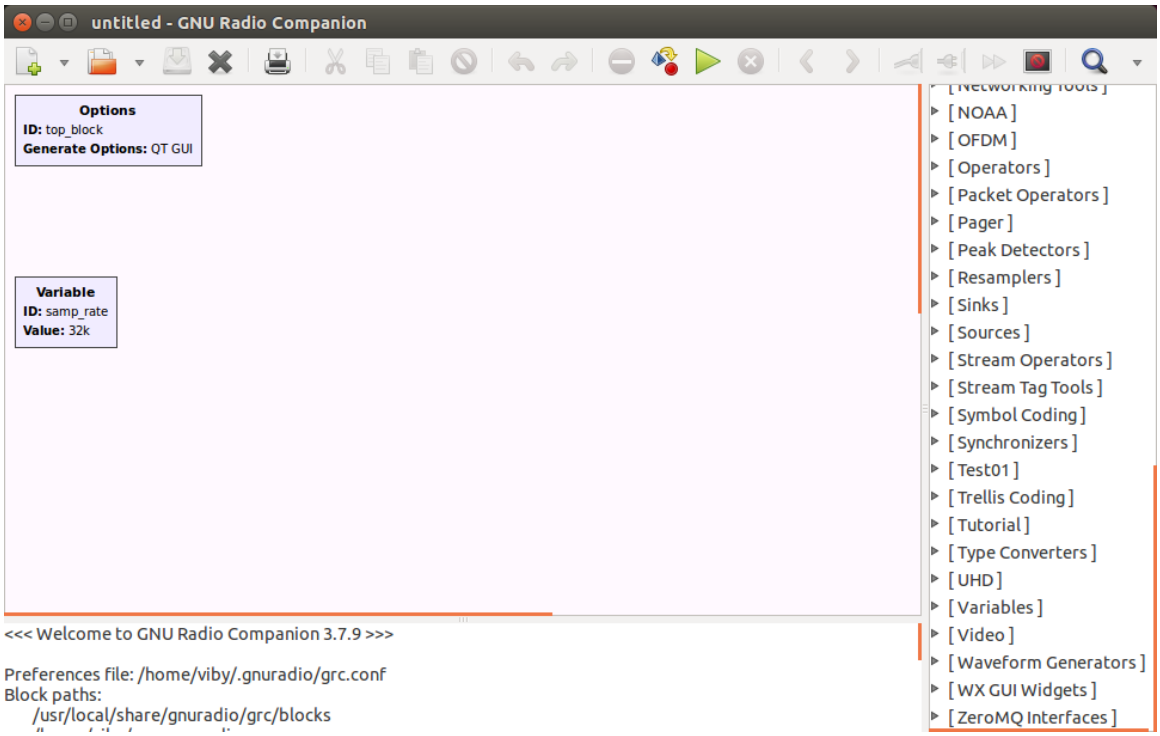
A software defined radio (SDR) is a "Radio in which some or all of the physical layer functions are software defined." A radio is any kind of device that transmits and/or receives signals wirelessly in the radio frequency (RF) spectrum from about 3 kHz to 300 GHz. Traditional radio devices are defined by their hardware and are typically only useable in a specific frequency band and for a particular type of modulation. SDRs, on the other hand, perform most of the complex signal processing needed for modern communications systems at baseband, using digital signal processing (DSP). Analog hardware is then used to translate between a (complex-valued) baseband signal and a corresponding (real-valued) bandpass signal in a frequency band that is suitable for wireless transmission and reception. In its purest form a SDR consists of an antenna and an analog-to-digital converter for the receiving part and a digital-to-analog converter connected to a power amplifier and an antenna for the transmitting part. All the filtering and signal processing then takes place in the digital domain that can be more precisely controlled in an economic fashion than is possible for traditional analog signal processing. In modern practice, DSP is used for frequencies up to several tens or a few hundreds of MHz and analog hardware is used for frequencies of several hundreds of MHz and beyond. Often the device that translates the digial baseband signal to the analog bandpass signal is referred to as the SDR and then the device (computer) that generates the digital baseband signal is referred to as the DSP.

The main advantage of an SDR over a traditional radio is that (most of) the radio's operating functions (often referred to as physical layer processing) are implemented through modifiable and upgradeable software and firmware on programmable devices such as field programmable gate arrays (FPGA), DSPs, general purpose computers (GPP), programmable System on a Chip (SoC), etc. The use of these technologies allows new wireless features and capabilities to be added to an existing radio system without replacing or modifying its hardware.

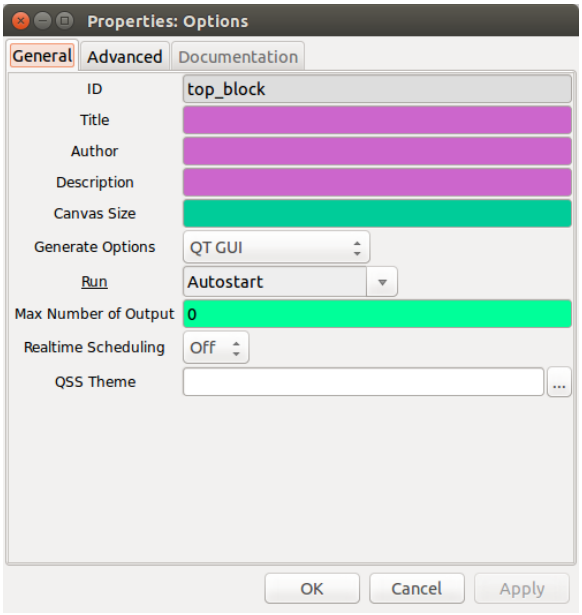
GNU Radio is a free software development toolkit that provides the DSP runtime blocks used to implement SDRs in conjunction with readily available low-cost external RF hardware. It is widely used in hobbyist, academic, commercial, and military environments to support wireless communications reearch, as well as to implement real-world radio systems. The GNU Radio Companion (GRC) is a graphical user interface that makes it possible to build GNU Radio flow graphs in a user-friendly graphical tool environment.

1.1 Getting Started with the GNU Radio Companion

GNU Radio Companion (GRC) is a graphical user interface that allows you to build GNU Radio flowgraphs using predefined DSP blocks. Start the GRC by typing `gnuradio-companion` in a terminal window in Linux and you will see an untitled GRC window similar to the following.

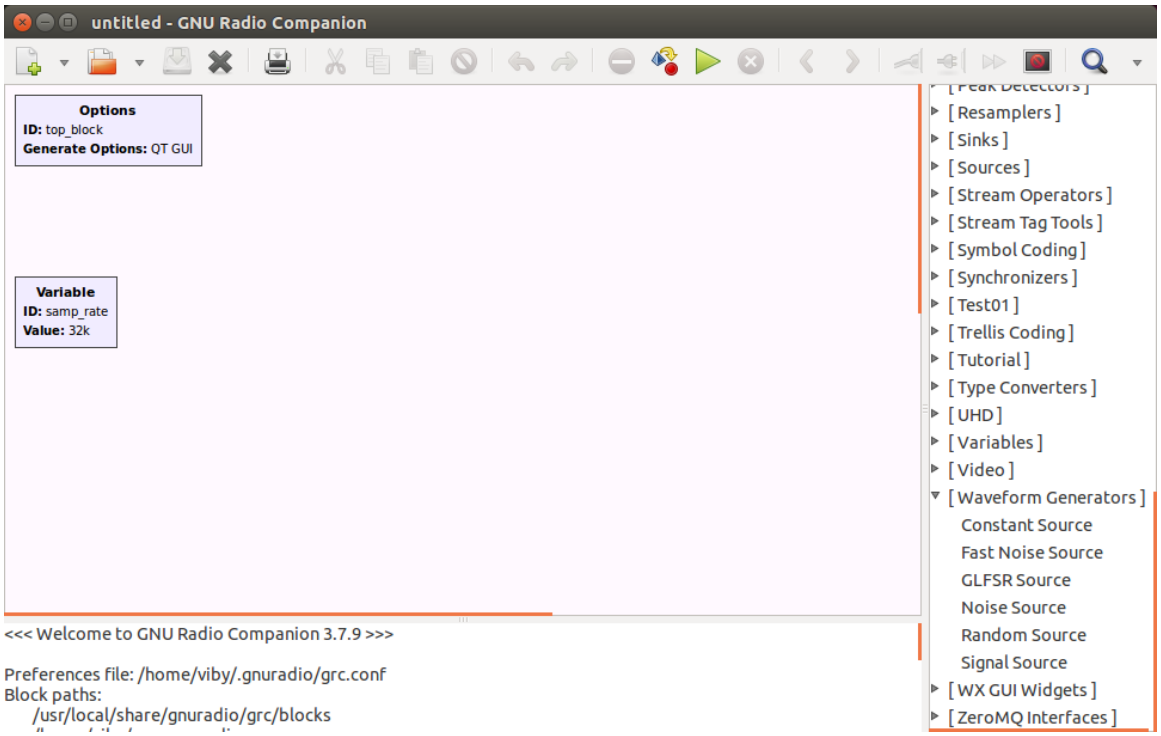


The Options block at the top left is used to set some general parameters of the flowgraph, such as the graphical user interface (GUI) for widgets and result displays, or the size of the canvas on which the DSP blocks are placed. Right-click on the block and click on Properties (or double-click on the block) to see all parameters that can be set.

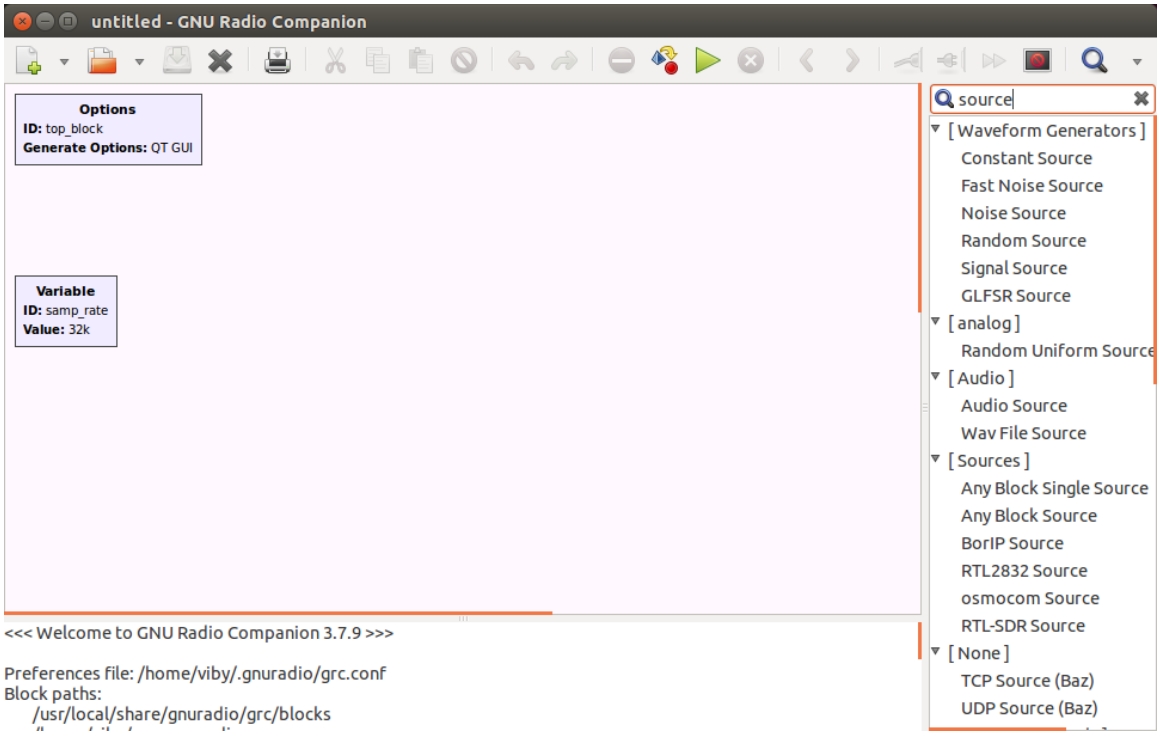


For now we will leave the default settings unchanged. Below the Options block is a Variable block that is used to set the sample rate, to $F_s = 32000$ Hz in the GRC window above.

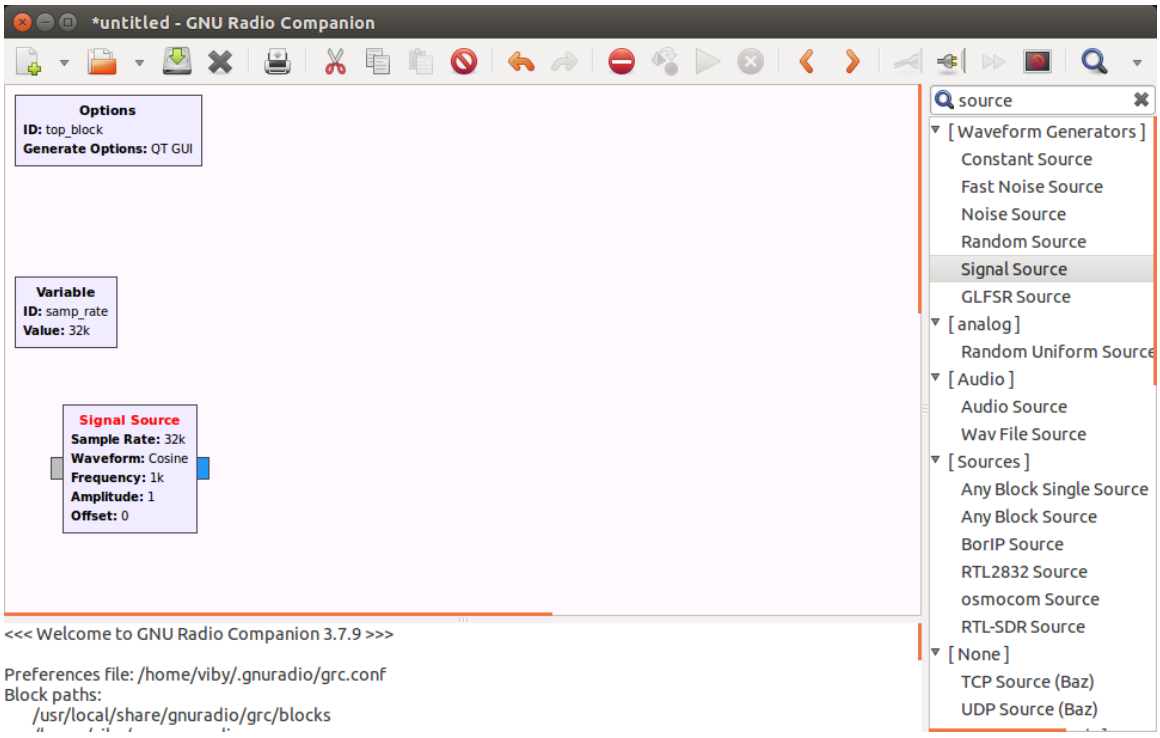
On the right side of the window is a list of the block categories that are available. Click on a triangle next to a category, e.g., Waveform Generators, to see what blocks are available in that category.



You can also use the search function (click on the looking glass on the top right) to enter a specific keyword, e.g., source, to see all blocks with "source" in their name.

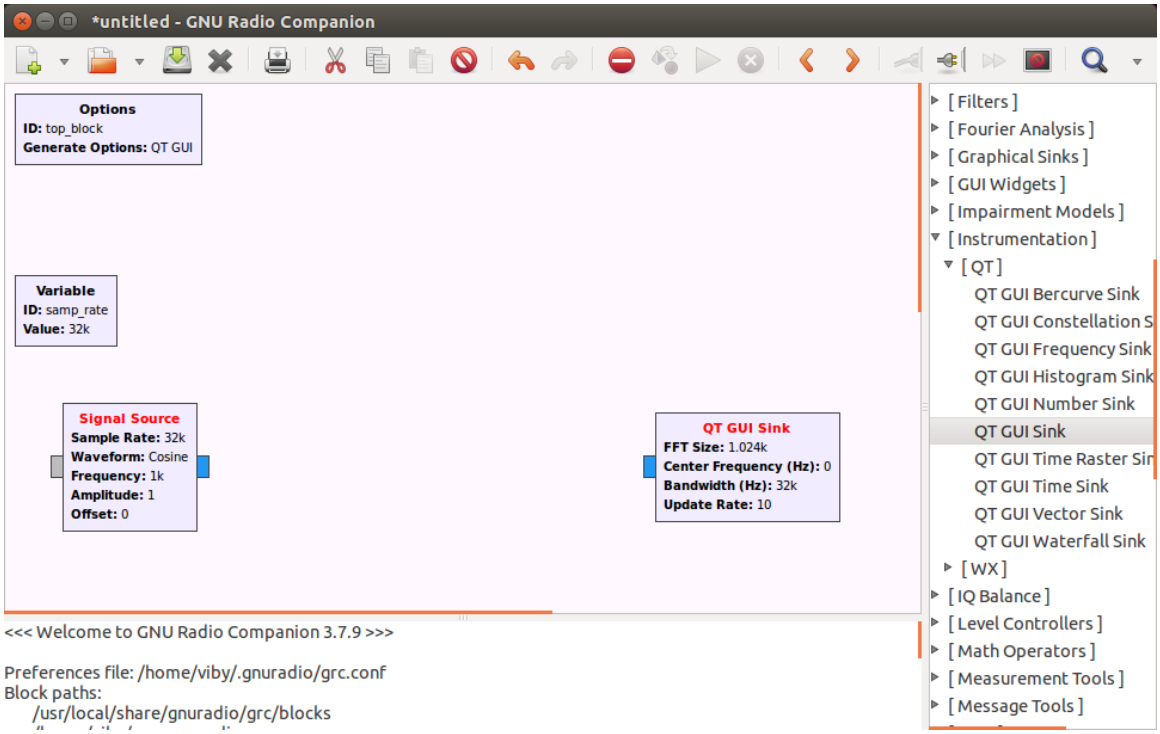


Double-click on Signal Source to place a Signal Source on the GRC canvas as shown below.

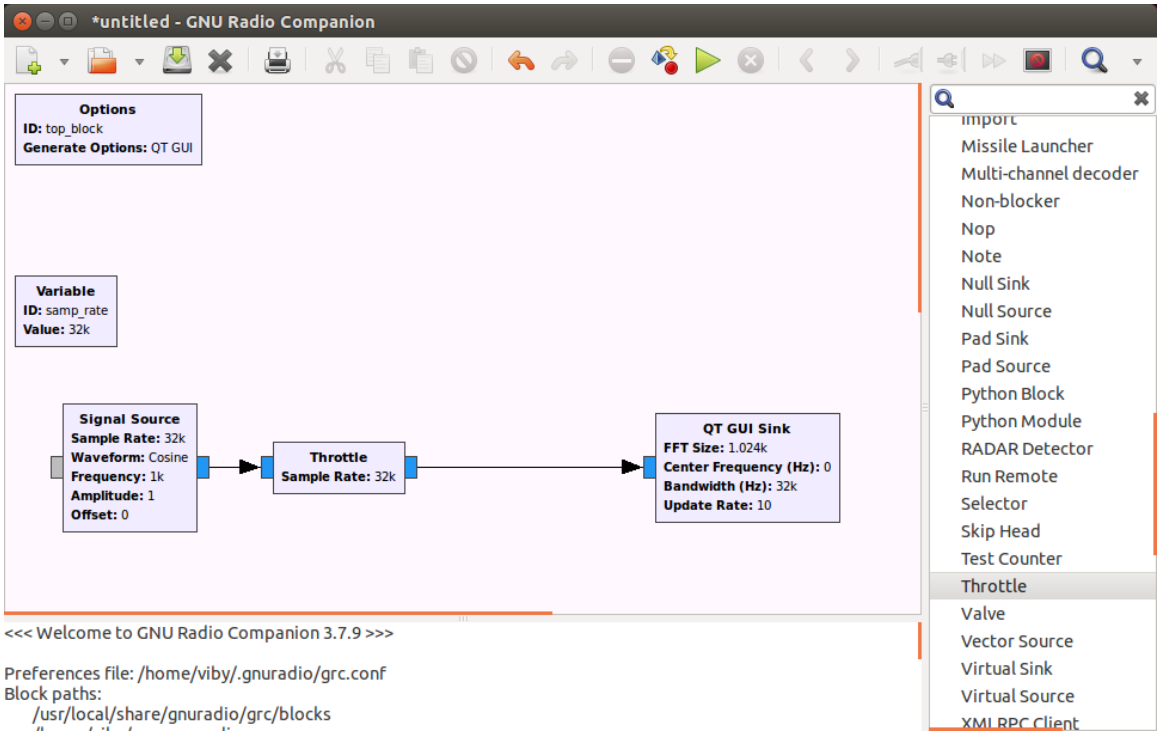


As a first experiment, we want to generate a cosine signal with frequency 1000 Hz (default for the Signal Source) and display it in the time and frequency domains. Under Instrumentation

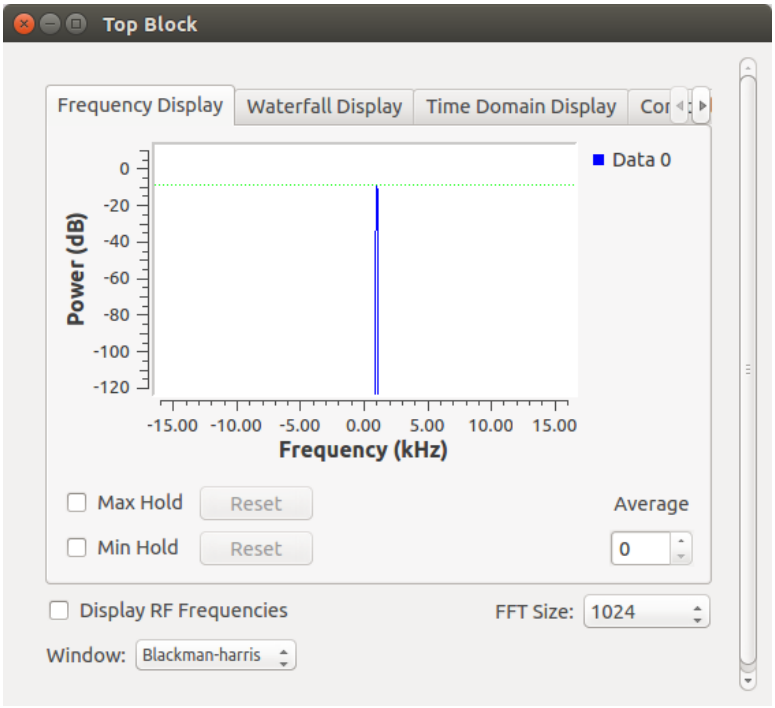
we choose QT and then double click on QT GUI Sink which will allow us to see the Frequency Display and the Time Domain Display of the signal produced by the Signal Source.



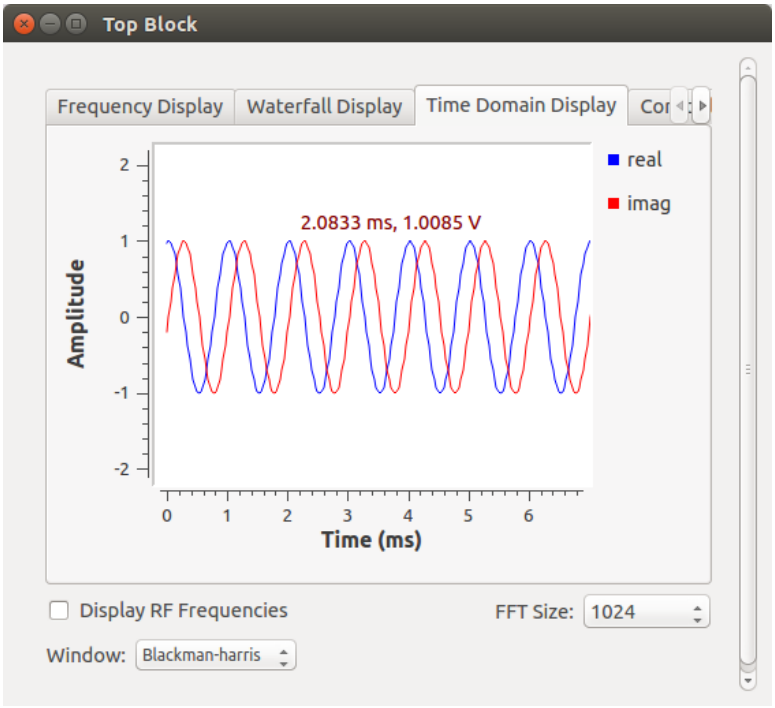
Note that at this point both Signal Source and QT GUI Sink are written in red. This indicates that there is something wrong, e.g., a parameter is missing or set incorrectly or, as is the case here, no wires are connected to the inputs/outputs of the blocks. To connect the output of the Signal Source to the QT GUI Sink we click on one of the blue pads, followed by clicking on the other one, which will result in a line with an arrow pointing from left to right. But before we do so, it is time to explain a little more about sampling rates in the GRC. To produce a DT cosine from $x(t) = \cos(2\pi f_1 t)$ as $x_n = x(nT_s) = \cos(2\pi f_1 n T_s)$, we need to know $T_s = 1/F_s$ where F_s is the sampling rate in Hz. This sampling rate is the "DSP sampling rate" which determines how many samples we produce for each period of $x(t)$. But when the resulting DT sequence is passed from a signal source to a signal sink, the speed of the transmission is determined by the connection from the source to the sink and/or by the speed at which the sink can accept new samples. The resulting sample rate is the "hardware sampling rate". In general, we try to make the DSP and the hardware sampling rates the same. But in a case where the source is a software generated waveform and the sink is a software-based display, the only speed limitation is the processing speed of the computer on which the software is run. To prevent the computer from choking up and devoting 100% of the processing time to the execution of the flowgraph, we use a Throttle block (from the Misc category) as shown next.



Note that now (after the connections between the blocks are made) the red "one-way street sign" in the menu bar has greyed out and the arrow to the right of it has become green, indicating that the flowgraph is ready to be executed. Save the file under a suitable name, e.g., `test_001.grc`, and then click on the green arrow. You should see a screen similar to the following pop up.

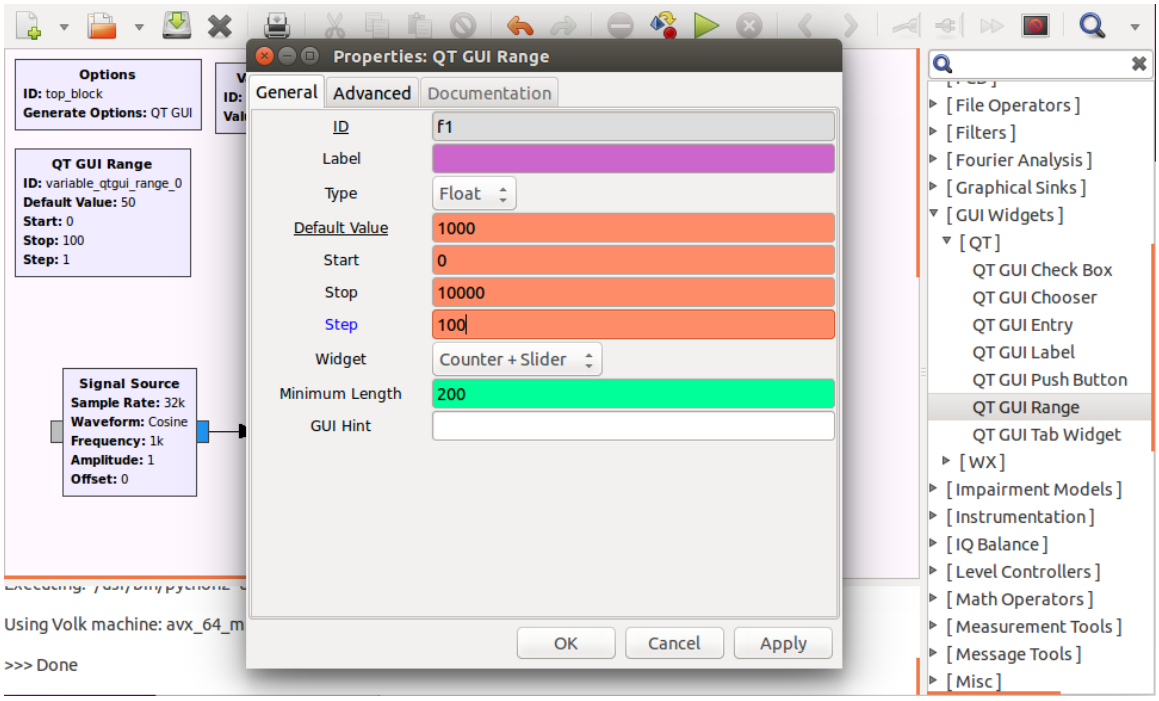


This is the frequency (power spectral density) display of the QT GUI Sink. Click the tab labeled Time Domain Display to see the waveform produced by the Signal Source in the time domain.

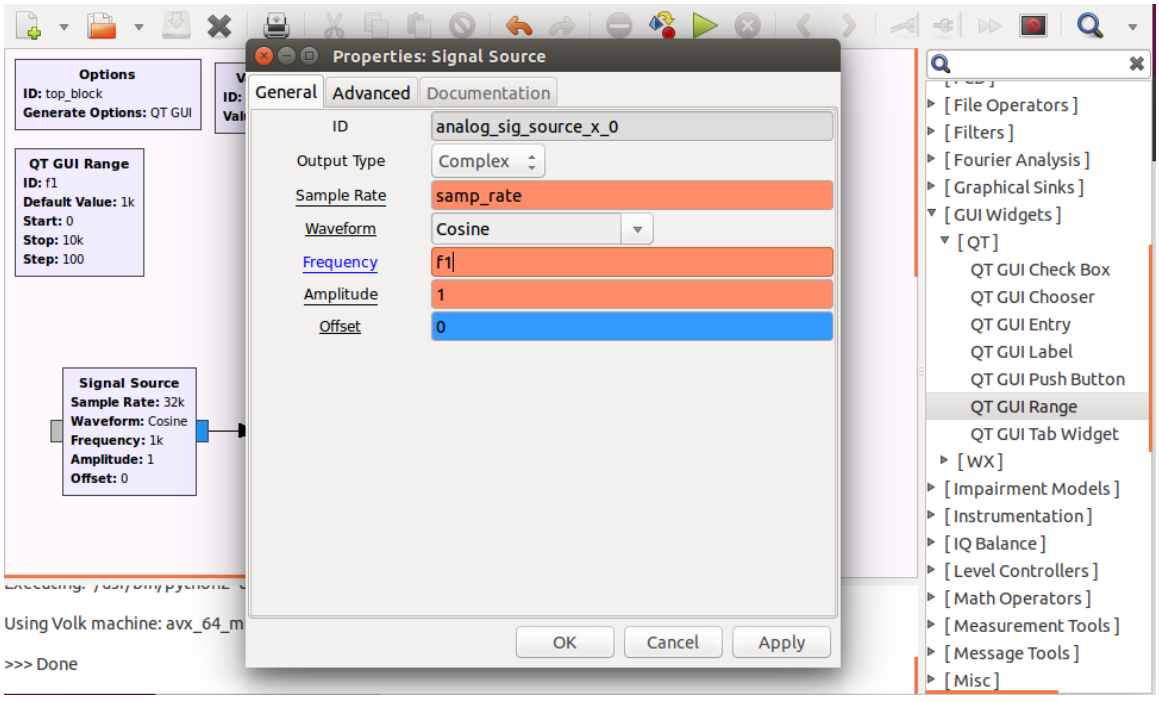


Notice that there are two sinusoids in the time domain and there is only one spectral line at +1.00 kHz in the frequency domain. This comes from the fact that, by default, the Signal Source produces complex-valued output signals, i.e., for a sinusoidal waveform with frequency f_1 the output is $e^{j2\pi f_1 n T_s}$, where $T_s = (\text{Sample Rate})^{-1}$. In the time domain the real part and the imaginary part are displayed as two separate waveforms (blue for the real part and red for the imaginary part).

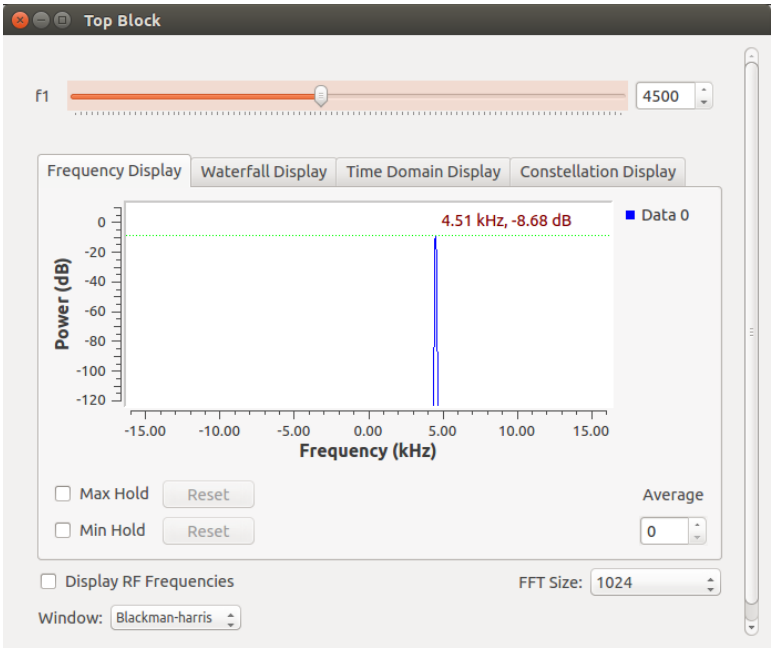
Next, we would like to be able to vary the frequency of the sinusoid generated by the Signal Source. In GRC blocks it is possible to change most parameters on the fly, i.e., during flowgraph execution using GUI Widgets such as sliders, push buttons, tabs, and choosers. In the QT subcategory of the GUI Widgets category double-click on QT GUI Range. Double-click on the resulting QT GUI Range block to open the properties as shown below. Set the ID to `'f1'` which will also serve as a Python variable name that can be used in other blocks to affect their behavior. Change the Default Value, the Start and Stop values, and the Step size as shown below.



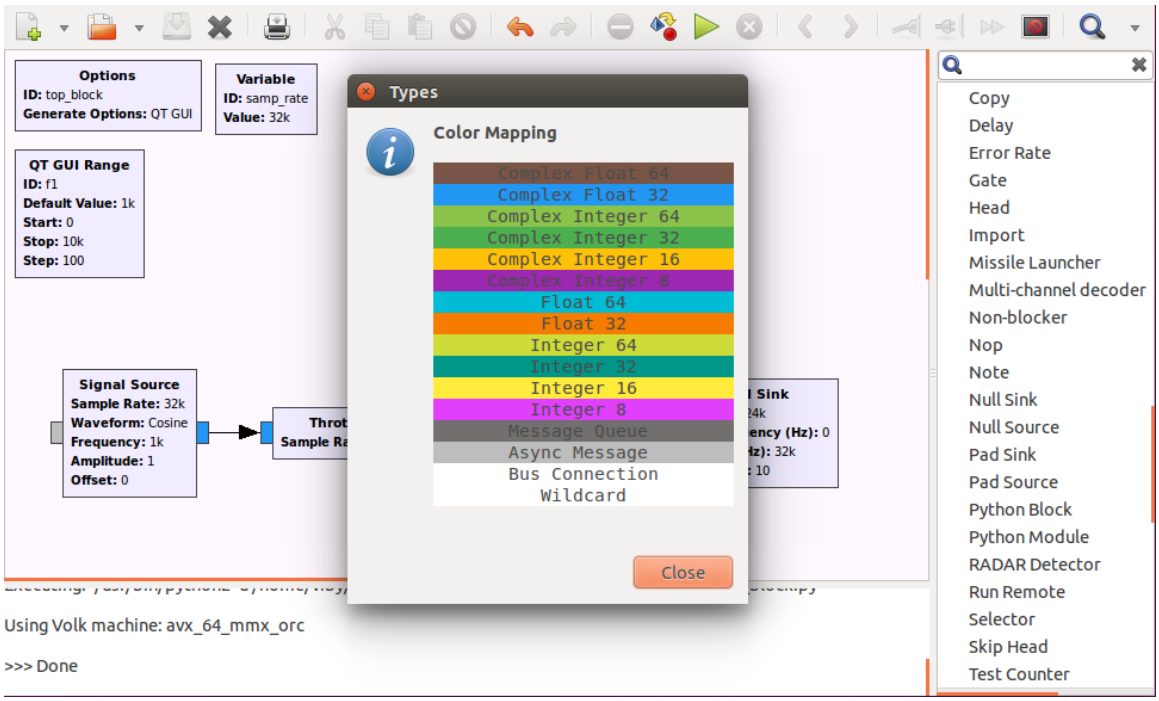
Then double-click on the Signal Source and change the frequency from the fixed value of 1000 to the Python variable `'f1'` that was defined in the QT GUI Range block.



Now save the modified flowgraph and execute it (using the green triangle in the menu bar at the top). As you move the slider from (the default) 1000 Hz to 4500 Hz (or any other frequency) you should see the spectral line in the frequency display move accordingly.

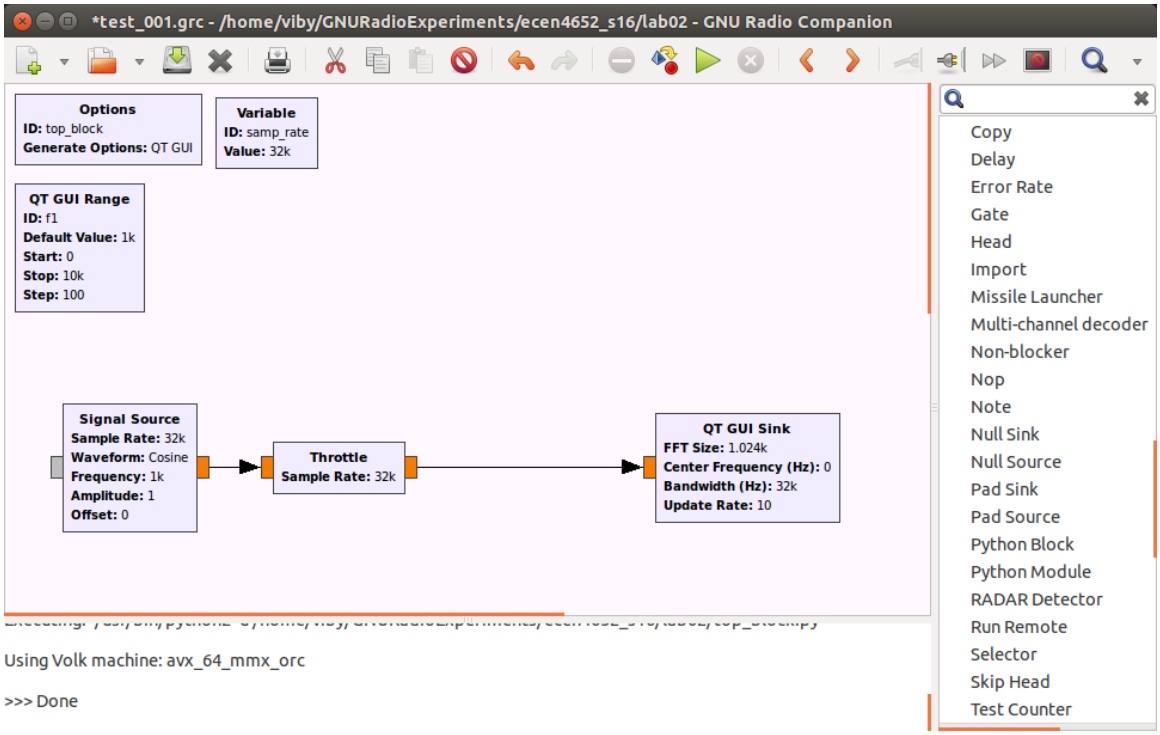


Now let's suppose we want to generate a real-valued sinusoidal waveform instead of the complex-valued one. The Output Type of the Signal Source can be set (under Properties) to Complex, Float, Int, or Short. In general, the color of a port of a block indicates which type of samples flow through the port. If you click on Help on the menu bar of the GRC window and select Types you will see the color mapping of the different Types of samples as shown in the screen snapshot below.

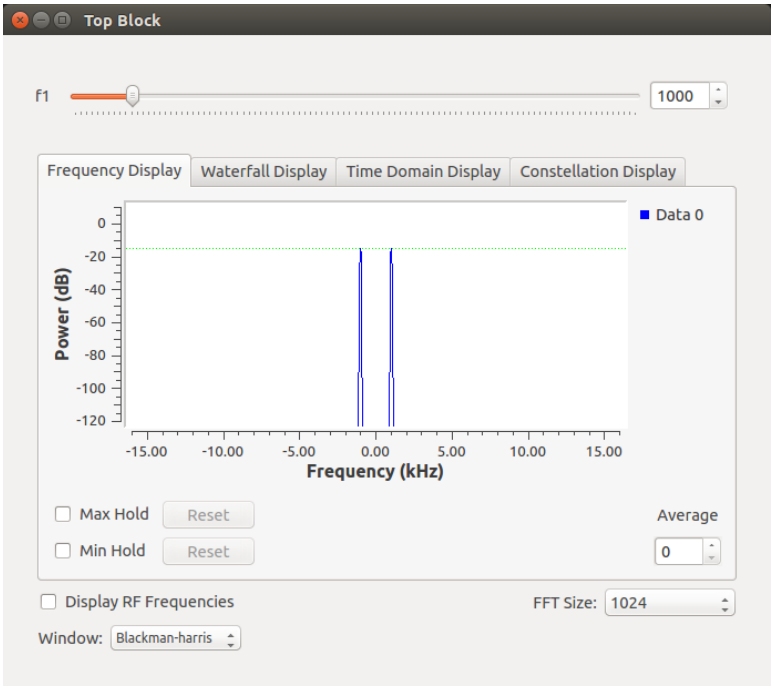


The blue color indicates complex-valued 32-bit floating point samples (Complex, 64 bits total for real and imaginary parts), the (darker) orange color indicates 32-bit (real-valued) floating point (Float) samples, the (very) dark green color indicates 32-bit integers (Int), and the yellow color indicates 16-bit integers (Short). As you can see in the figure above, there are more types than the four we listed explicitly here.

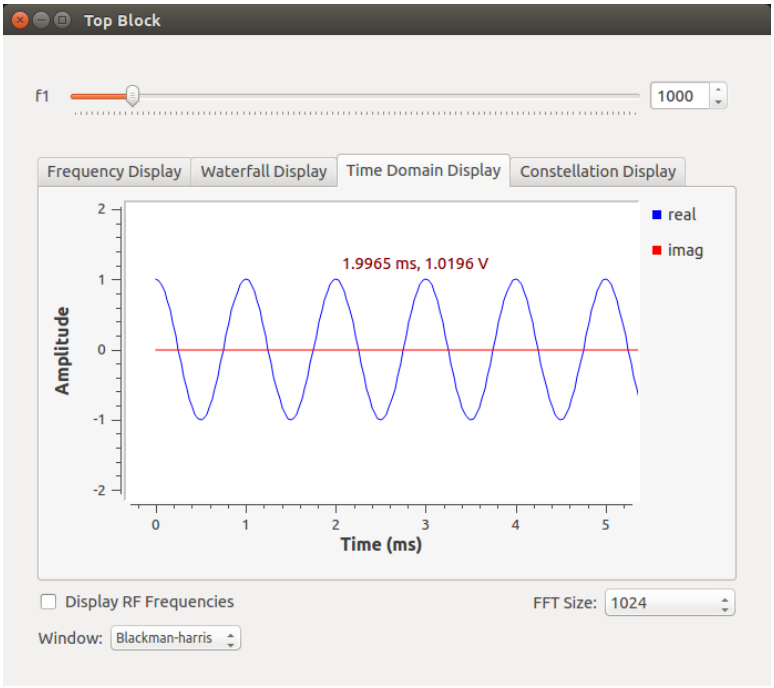
Let's go ahead and change all data types in our flowgraph to Float. While you are doing this, note that if one end of a connection is of Type Complex and the other is of Type Float, the arrow on the connection line turns red, indicating a type mismatch (and disabling the execution of the flowgraph).



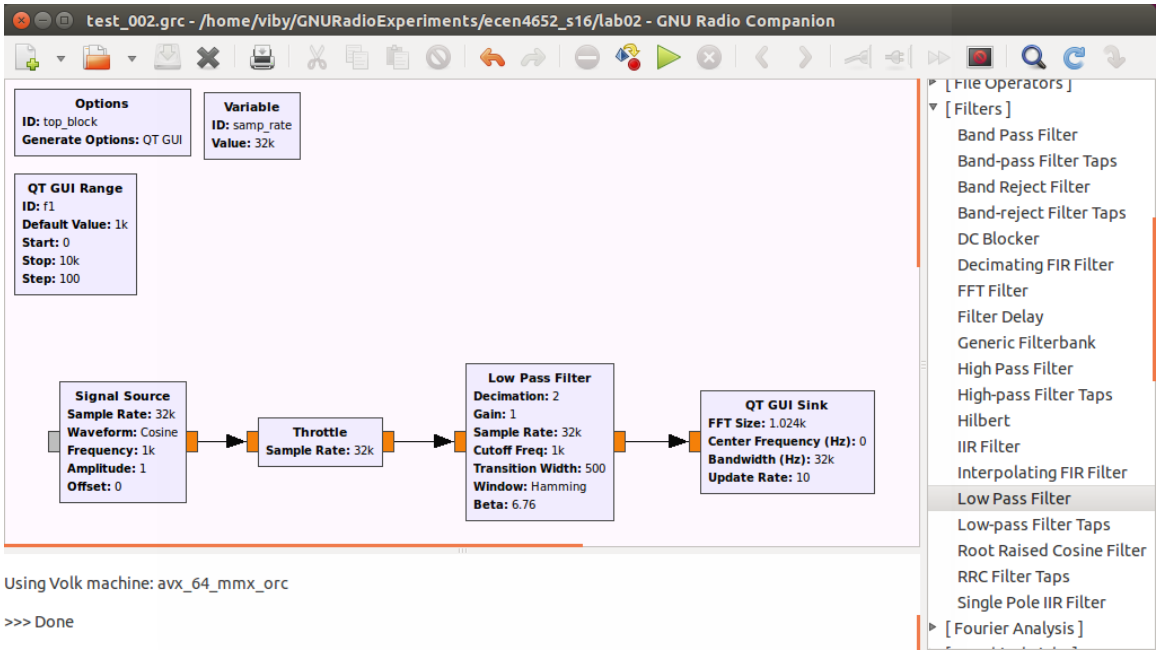
With this change the Frequency Display looks as follows when the flowgraph is executed. Because the sinusoidal signal is now real-valued its power spectral density is an even function of frequency.



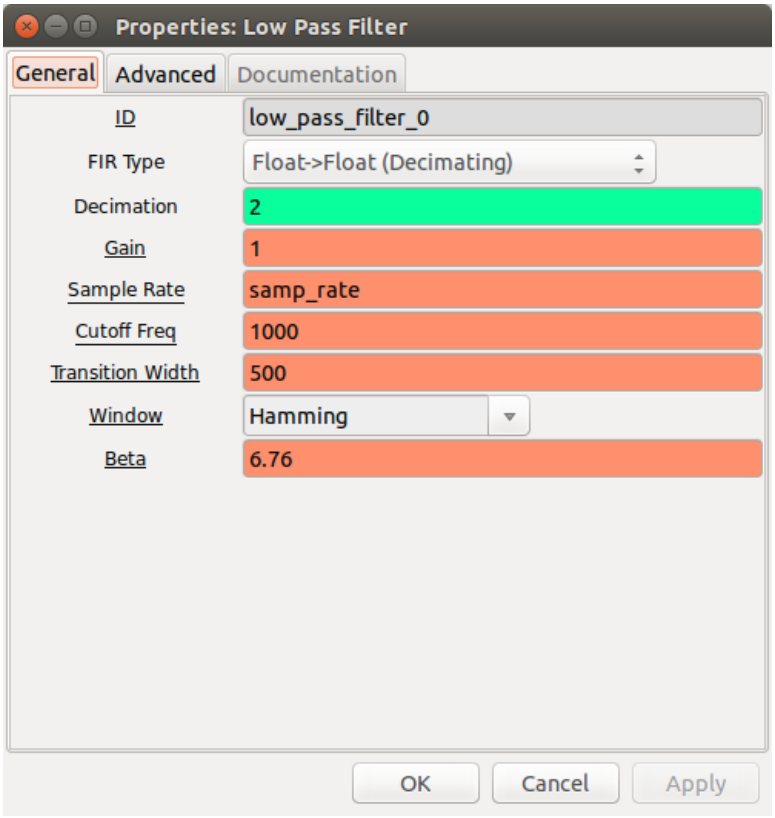
The next graph shows the real-valued sinusoid in the time domain.



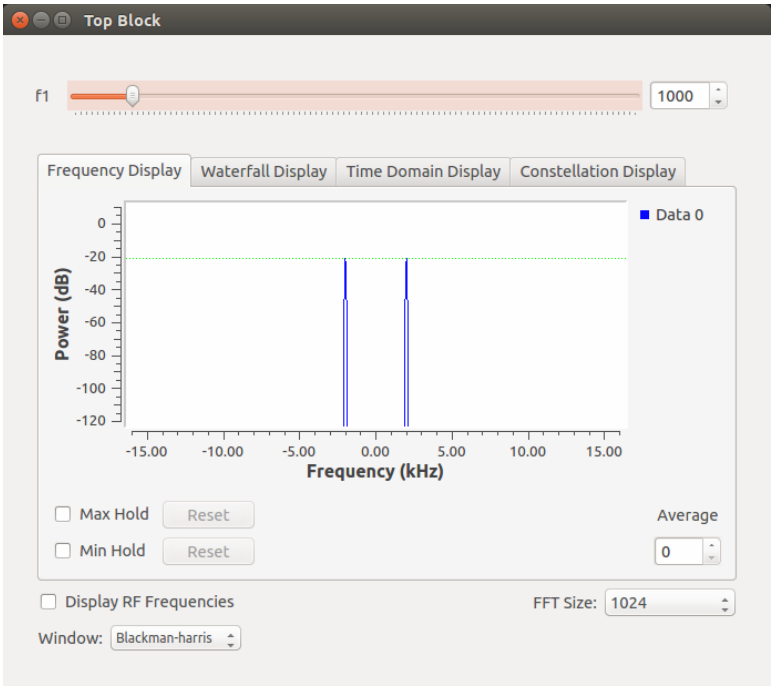
Let's do some simple signal processing next. From the Filters category place a Low Pass Filter block between the Throttle and the QT GUI Sink blocks as shown below.



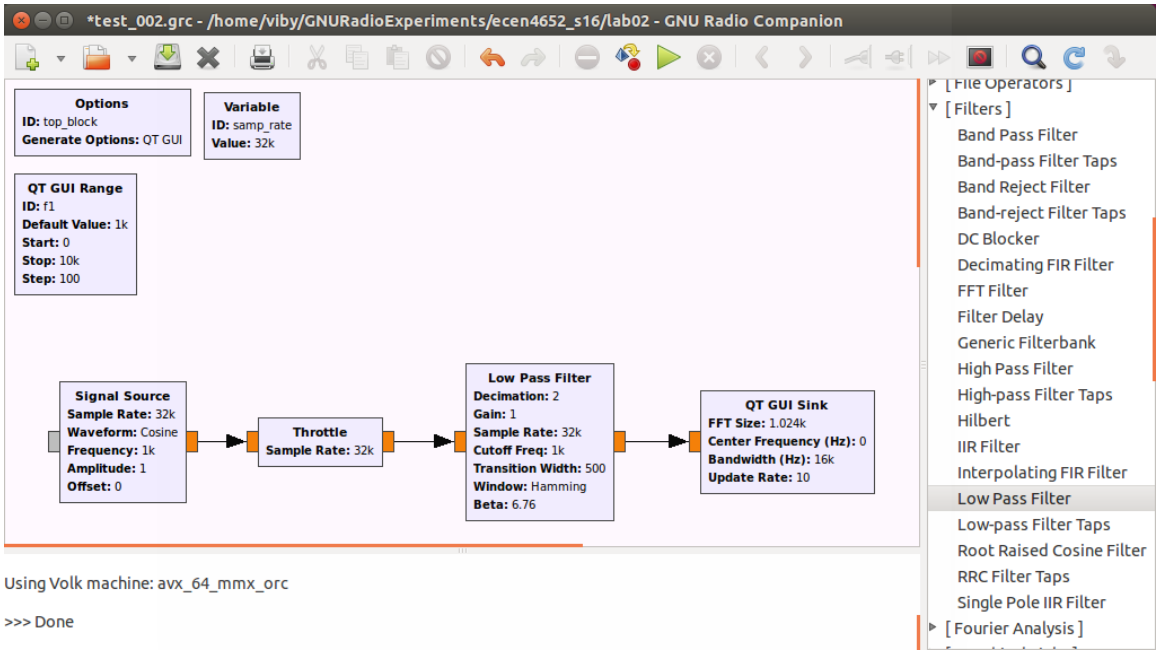
Fill the parameters of the Low Pass Filter in as shown next. Note the decimation factor of 2.



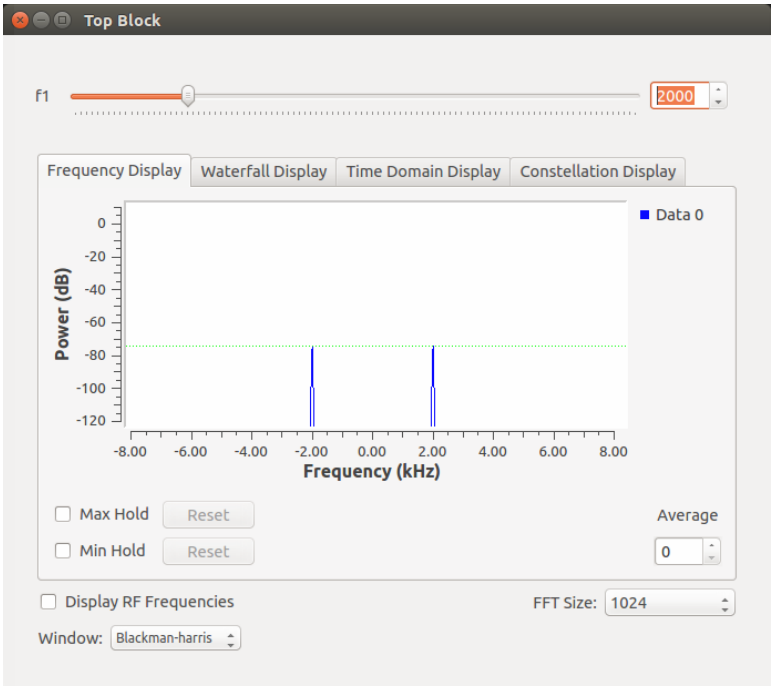
If we execute the flowgraph we get the following frequency display.



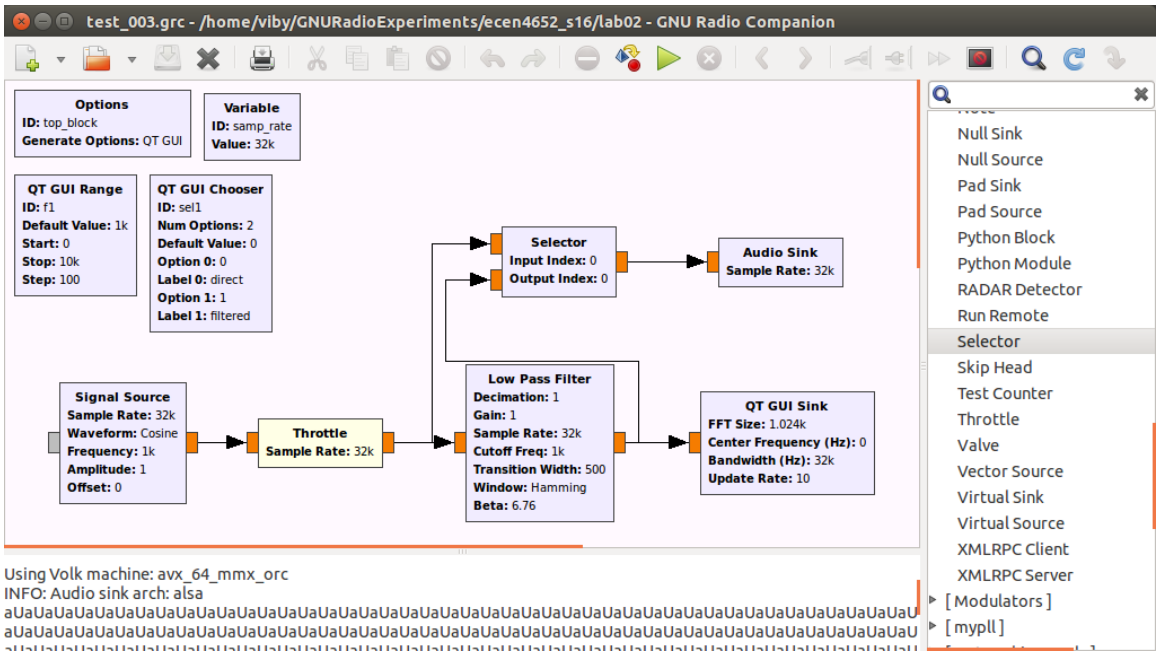
Interestingly enough, the spectral lines are now at ±2.00 kHz even though `f1` is set to 1000. Why does this happen? Well, we specified a decimation factor of 2 for the Low Pass Filter, but we didn't change the Bandwidth of the QT GUI Sink. Go ahead, open the Properties of the QT GUI Sink and change the Bandwidth setting from `samp_rate` to `samp_rate/2` so that flowgraph looks like the one below.



Now `f1` of 1000 should correctly produce spectral lines at ±1.00 kHz. If you move the slider from 1000 Hz to 2000 Hz you should see a significant drop in the amplitude of the spectral lines (from about -20 dB to about -75 dB) as a result of the lowpass filtering.

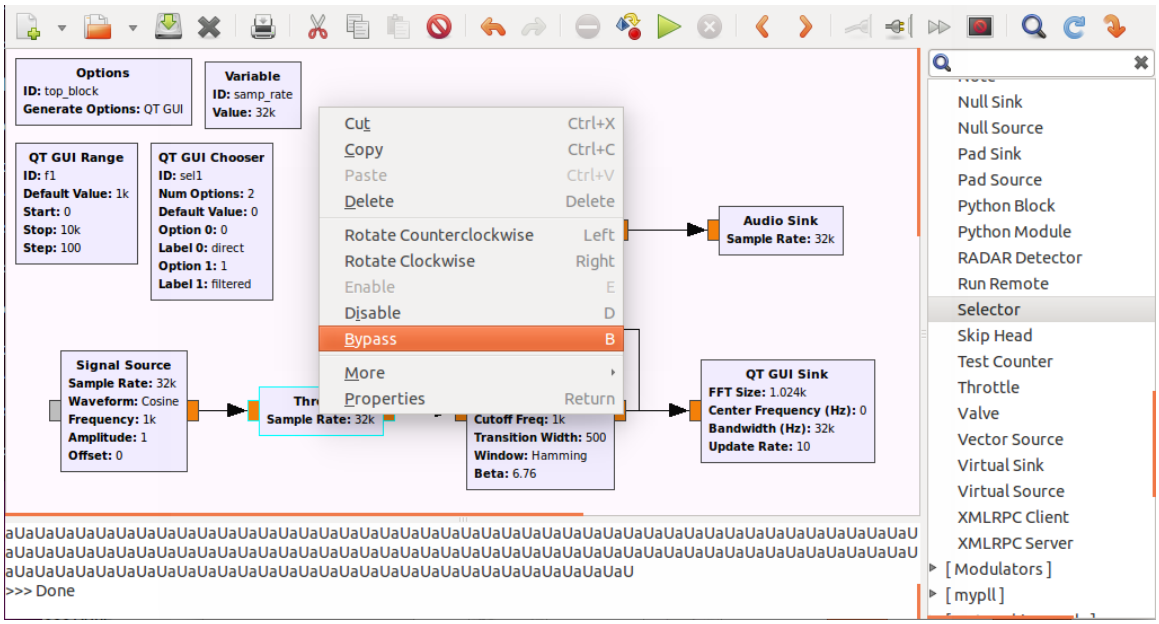


Next, let's listen to the signal before and after the filter. We will use a Selector (from the Misc category) together with a QT GUI Chooser (from the QT subcategory of the GUI Widgets category) to implement the switching. For the audio output we use a Audio Sink from the Audio category. To have the same sampling rate at the input and the output of the filter we set the decimation factor of the Low Pass Filter to 1 and the Bandwidth of the QT GUI Sink back to `samp_rate`.



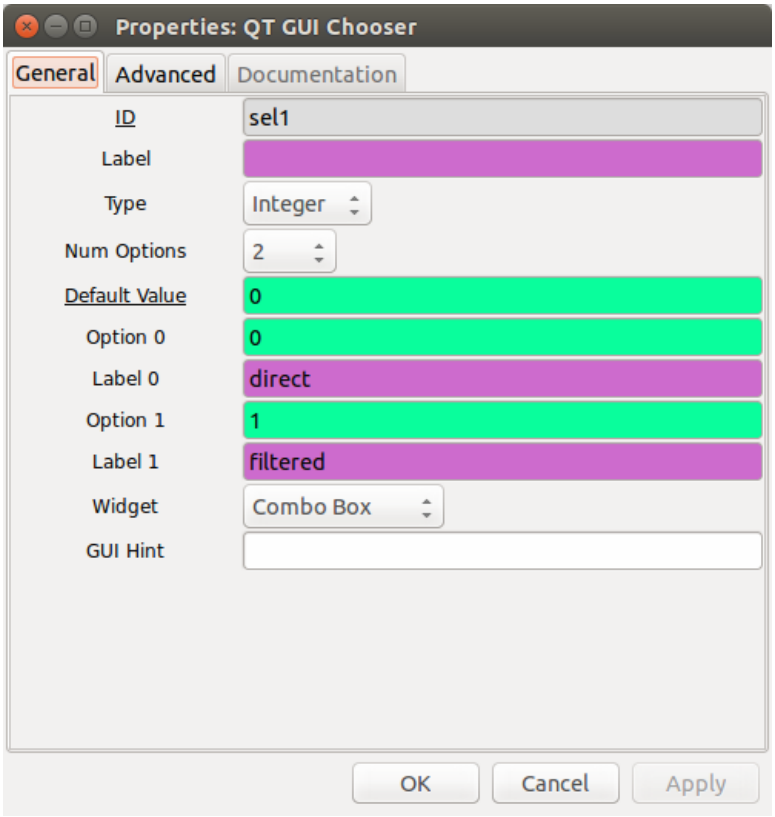
Note that the Throttle block has changed its color. This happened because we selected to bypass the block by right-clicking on it and then selecting Bypass. Below we can also see

other options for flowgraph blocks, like copying, rotating, deleting, disabling, and enabling blocks. The bypassing, disabling, and enabling functions come in handy when we want to experiment with different configurations without the need to actually delete blocks and then recreate them again.

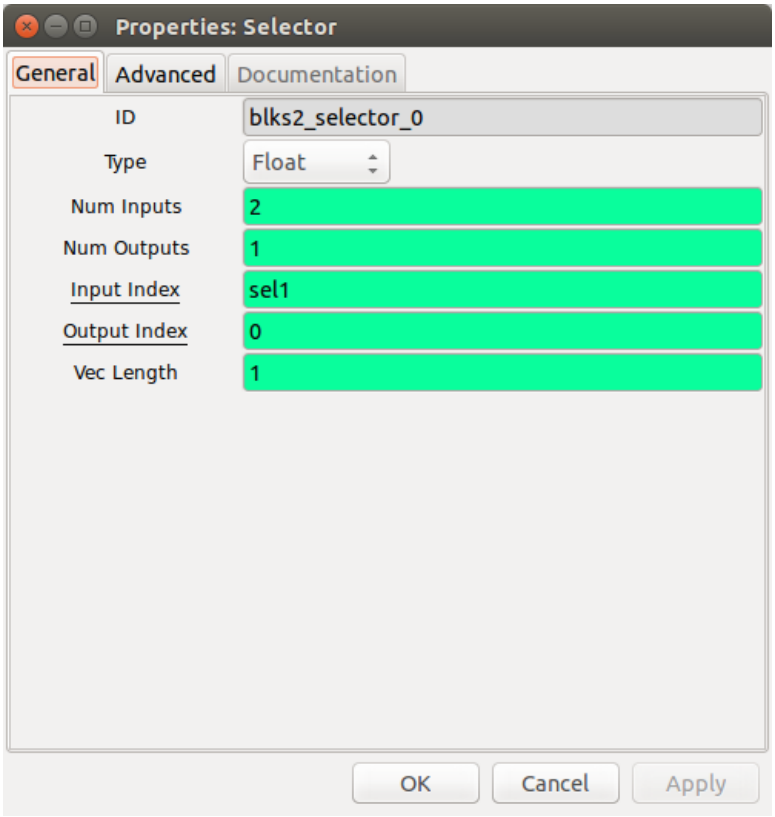


So the question is "Why did we choose to bypass the throttle block?" The answer to this is that we need to avoid the "two-clock" problem that arises because the audio sink is a hardware device that actually dictates a specific speed (32 kHz in this example). Thus, by keeping the Throttle block in the connection path, we would create the situation where two devices, with slightly different ideas of what a sampling rate of 32 kHz means, try to dictate the interconnection speed.

To complete the setup of the flowgraph, set the Properties of the QT GUI Chooser to the values shown below.



Then use the `'sel1'` Python variable from the QT GUI Chooser for the selection of the Input Index in the Selector block and fill out the remaining Properties as shown next.



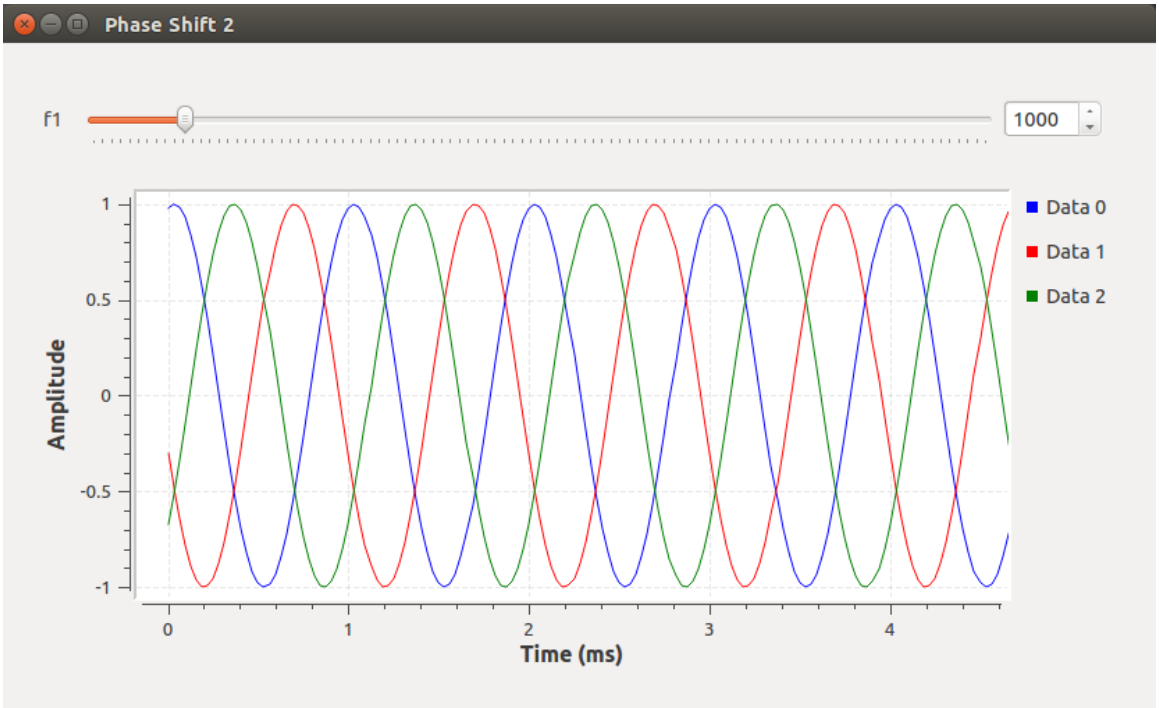
Now you can execute the flowgraph and try out the different features like the Selector switch and the `f1` slider. Depending on the speed of your computer (and the speed of VirtualBox if you are running the GRC inside of it), you may see the string aUaUaUaU... in the message window of the GRC and the output may sound choppy. The aU means "audio underrun" and indicates the samples for the Audio Sink are not produced fast enough.

2 Lab Experiments

E1. Install VirtualBox, Ubuntu, and GNU Radio.

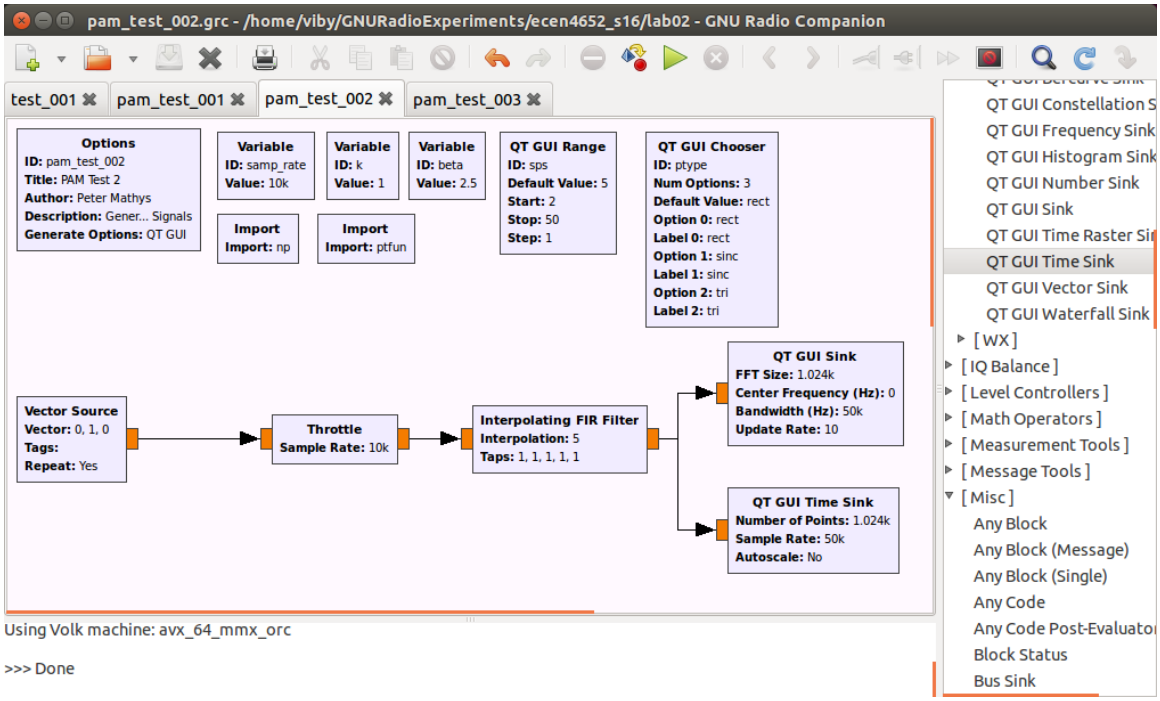
E2. Generation of Test Signals in GNU Radio. **(a)** Work through and recreate the examples given in the "Getting Started with GNU Radio Companion". Familiarize yourself with the basic blocks that are available in the GRC and feel free to experiment, e.g., generate two sinusoids of different frequencies and then adding them together or multiplying them together. Look at the Frequency and Time Domain Displays and check whether the results correspond with what you exprect them to be.

(b) The graph below shows a three-phase sinusoidal signal with a phase shift of 120° between any two sinusoids.

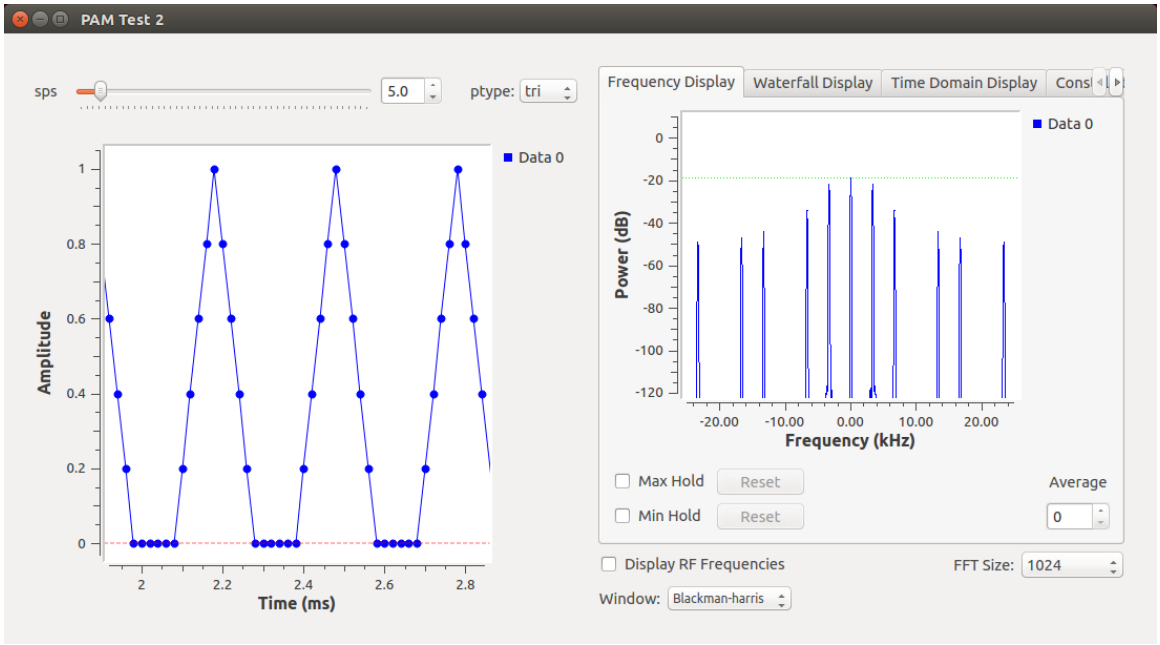


Use a sampling rate of 32 kHz and show how to generate such a signal with variable frequency f_1 in the range 0 to 10000 Hz in the GRC. Hints: Start out with a complex-valued sinusoid and ask yourself what the best way is to obtain a phase change analytically that works for all frequencies. To convert a complex-valued signal to a real-valued signal in the GRC use the corresponding block from the Type Converters category. For the display use a QT GUI Time Sink and change the Number of Inputs from 1 to 3. Include a screen snapshot of your flowgraph and of the Time Sink display in your report.

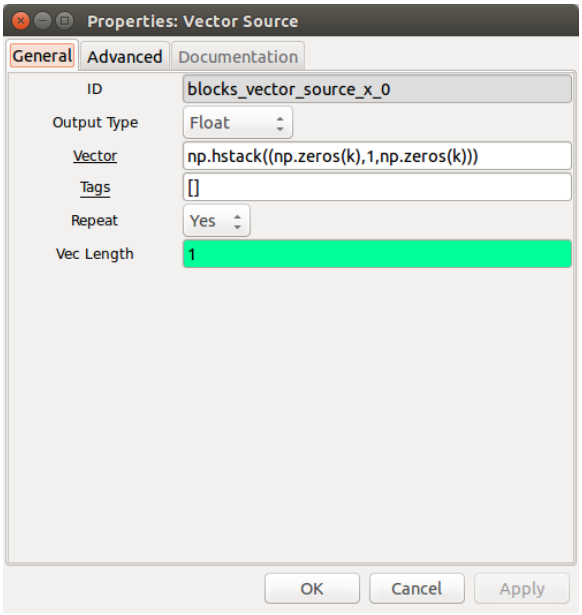
(c) The figure below shows a flowgraph that can be used to generate PAM pulses of type `'rect'`, `'sinc'`, and `'tri'` with a given number of samples per symbol (`sps`).



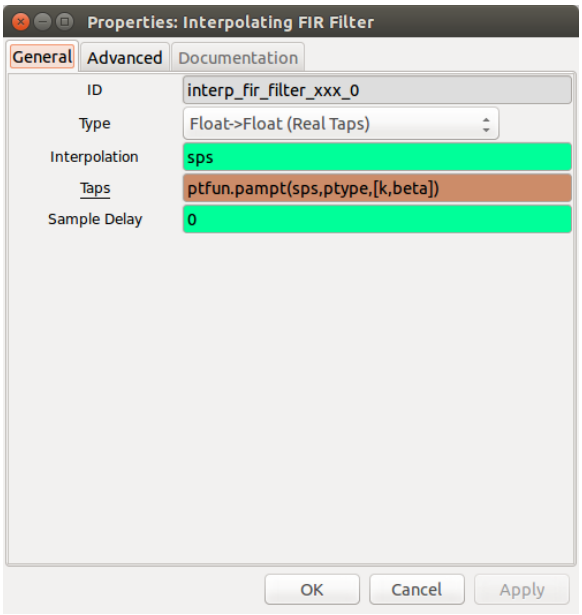
When the flowgaph is executed for pulse type `'tri'` and sps=5, the following graphs are obtained.



The Vector Source produces k zeros followed by a single 1 and followed by another k zeros in order to produce just a single PAM pulse $p(t)$ at the output of the filter. Note that the NumPy Python module must be imported explicitly into the flowgraph so that it can be used in the Vector definition of the Vector Source.



To produce a PAM signal $s(t)$ with the specified number of sps, an Interpolating FIR filter is used whose filter Taps are derived from a Python module `ptfun.py` that you have to write as part of this experiment.



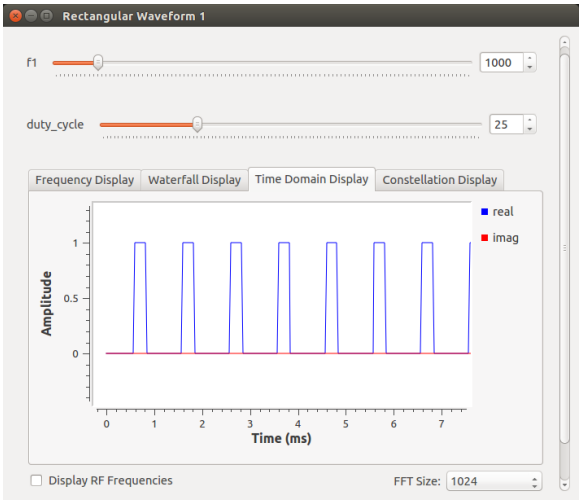
The header of the Python function `ptfun` looks as follows.

```
# File: ptfun
# Functions for gnuradio-companion PAM p(t) generation
import numpy as np

def pampt(sps, ptype, pparms=[]):
    """
    PAM pulse p(t) = p(n*TB/sps) generation
    >>>>> pt = pampt(sps, ptype, pparms) <<<<<
    where  sps:
           ptype: pulse type ('rect', 'sinc', 'tri')
           pparms not used for 'rect', 'tri'
           pparms = [k, beta] for sinc
           k:     "tail" truncation parameter for 'sinc'
                  (truncates p(t) to -k*sps <= n < k*sps)
           beta:  Kaiser window parameter for 'sinc'
           pt:    pulse p(t) at t=n*TB/sps
    Note: In terms of sampling rate Fs and baud rate FB,
          sps = Fs/FB
    """
```

E3. Feature Analysis and Practice with GRC Blocks. (Experiment for ECEN 5002, optional for ECEN 4652) **(a)** Determine the frequency response and the order of the Low Pass Filter (before decimation) used in the "Getting Started with GNU Radio Companion" section. Hint: The lowpass filter is a FIR filter.

(b) Derive a GRC flowgraph that can generate the rectangular signal shown below with a variable frequency f_1 in the range from 0 to 10000 Hz, and a variable duty cycle in the range 0 to 100%, using a sampling rate of 32 kHz. Include a screen snapshot of your flowgraph and the resulting time and frequency domain plots in your lab report.



(c) Derive a GRC flowgraph that can generate the signal shown below with a variable frequency f_1 in the range from 0 to 10000 Hz, using a sampling rate of 32 kHz. Include a screen snapshot of your flowgraph and the resulting time and frequency domain plots in your lab report.