Lab 3: Introduction to Software Defined Radio and GNU Radio

1 Introduction

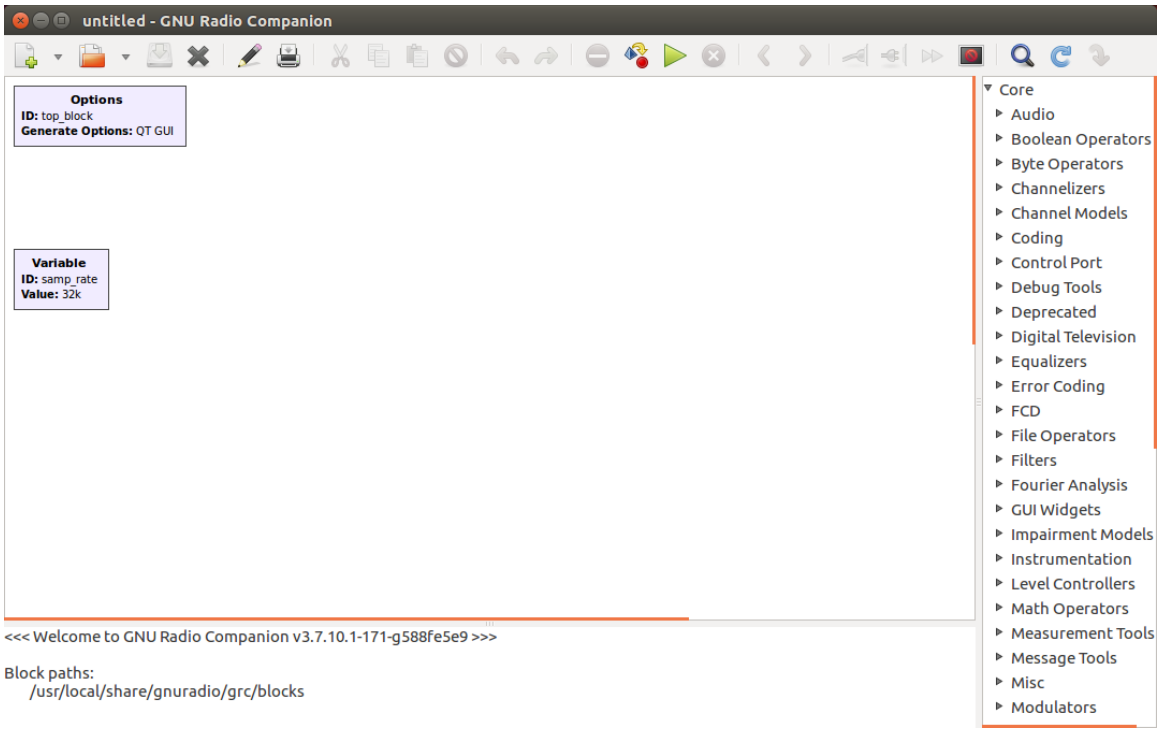
A **software defined radio (SDR)** is a "Radio in which some or all of the physical layer functions are software defined." A radio is any kind of device that transmits and/or receives signals wirelessly in the radio frequency (RF) spectrum from about 3 kHz to 300 GHz. Traditional radio devices are defined by their hardware and are typically only usable 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 its 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 upgradable 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.

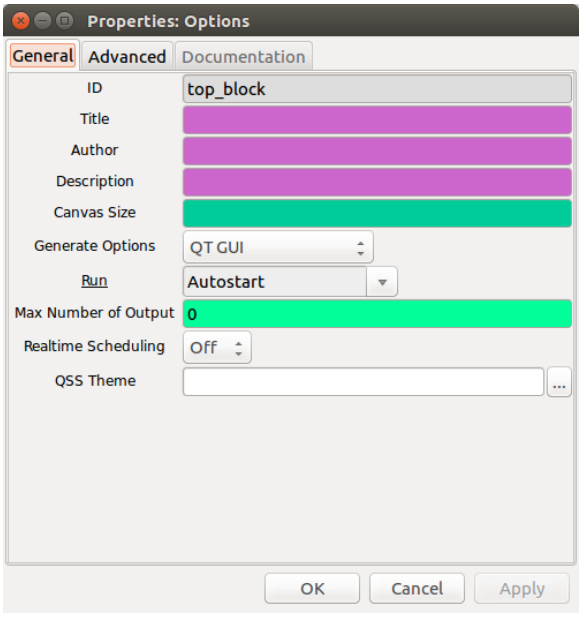
GNU Radio is a free software development toolkit that provides the DSP runtime blocks used to implement SDRs in conjunction with readily available low-cost external RF hardware. It is widely used in hobbyist, academic, commercial, and military environments to support wireless communications reearch, 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

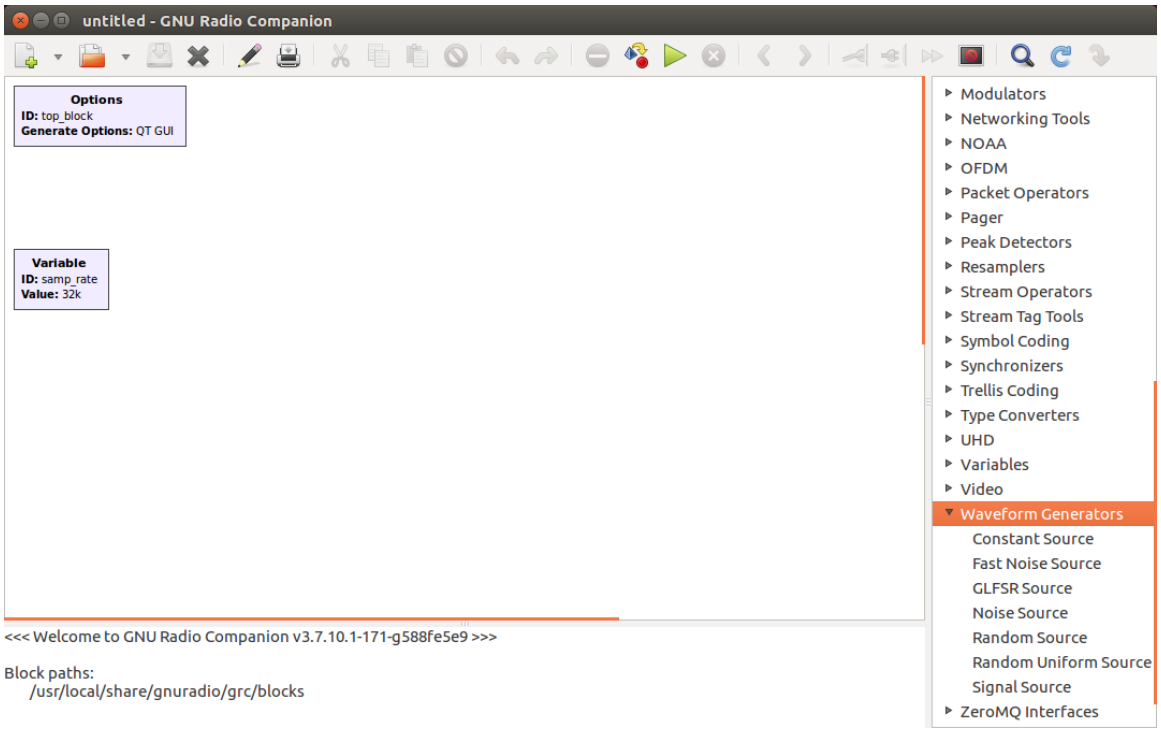
GNU Radio Companion (GRC) is a graphical user interface that allows you to build GNU Radio flowgraphs using predefined DSP blocks. Start the GRC by typing `gnuradio-companion` in a terminal window in Linux and you will see an untitled GRC window similar to the following.



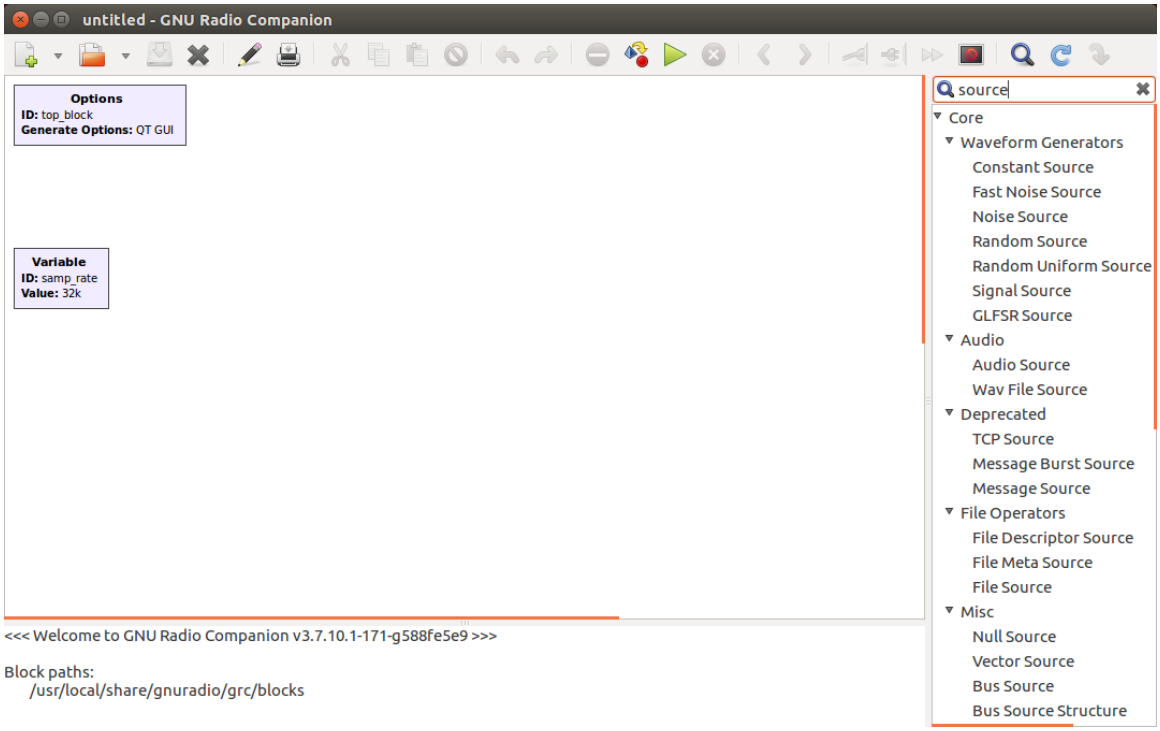
The "Options" block at the top left is used to set some general parameters of the flowgraph, such as the graphical user interface (GUI) for widgets and result displays, or the size of the canvas on which the DSP blocks are placed. Right-click on the block and click on Properties (or double-click on the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



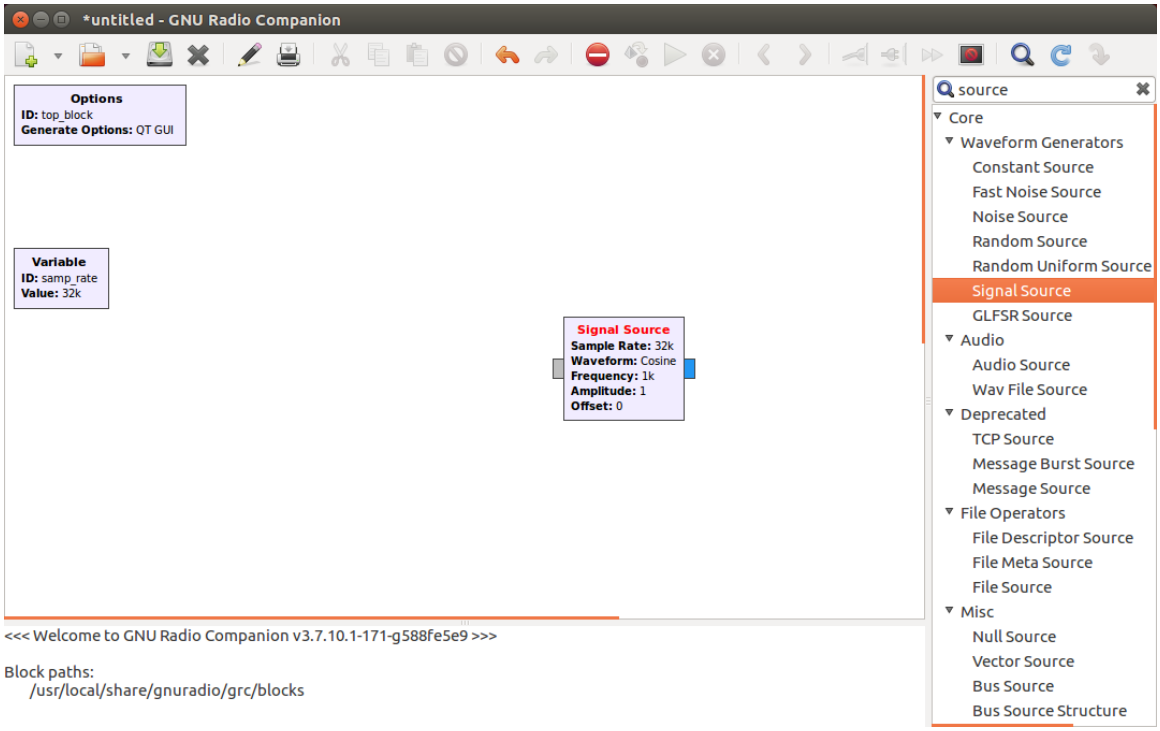
For now we will leave the default settings unchanged. Below the Options block is a "Variable" block that is used to set the sample rate, e.g., to $F_s = 32000$ Hz in the GRC window above. On the right side of the window is a list of the block categories that are available. Click on a triangle next to 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.

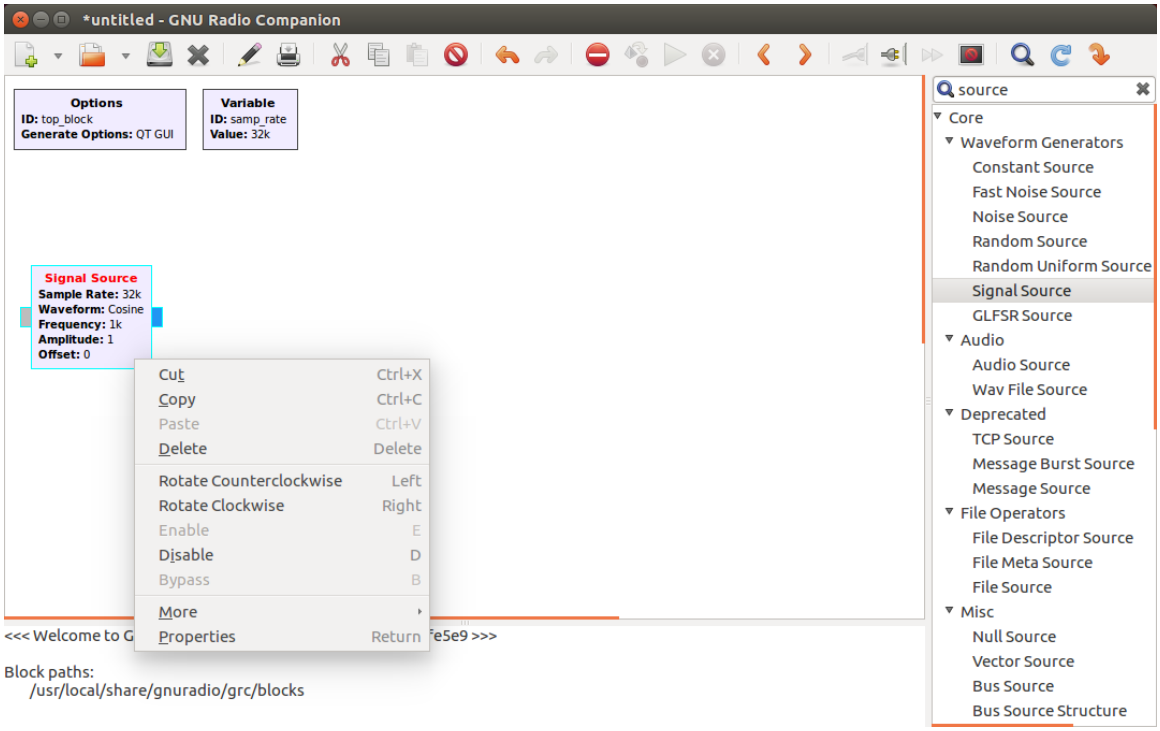


Double-click on Signal Source to place a "Signal Source" on the GRC canvas as shown below.

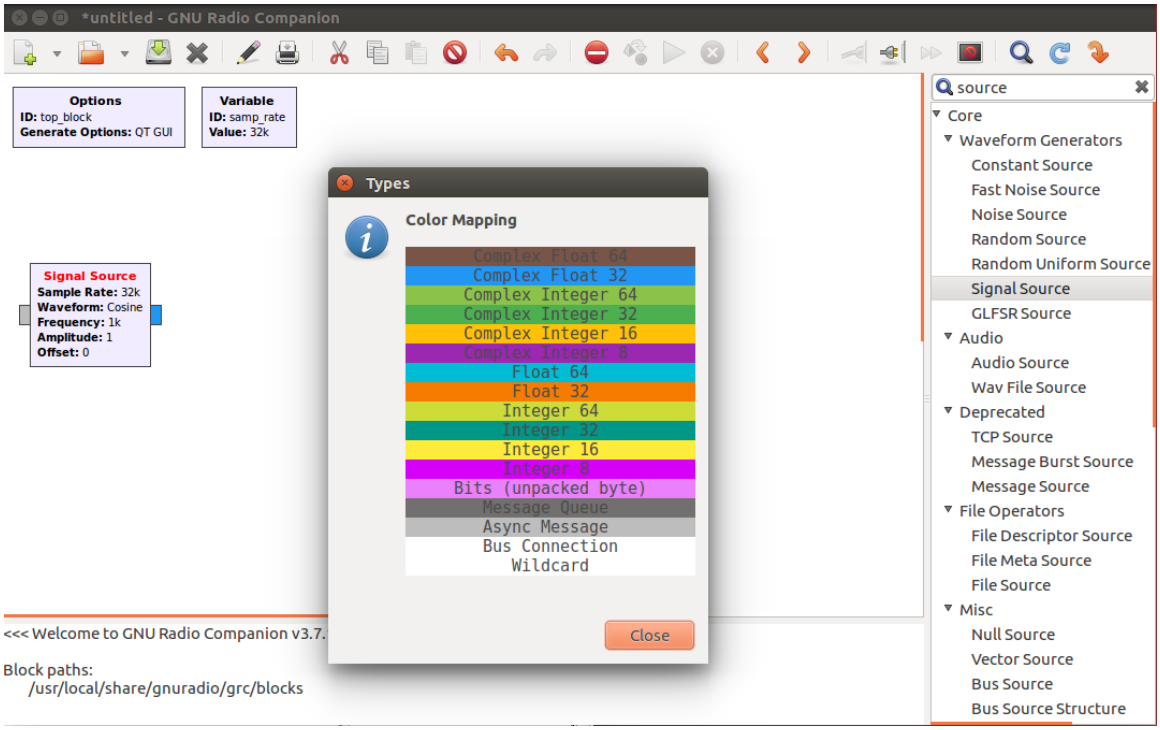


To move a block on the canvas, grab it with the cursor, press the left mouse button, and move the block to the desired location. You can also rotate blocks by right-clicking on them

and then clicking either “Rotate Counterclockwise” or “Rotate Clockwise”. Blocks can also be temporarily disabled by clicking on “Disable”, which is useful for debugging and what-if questions. The rearranged blocks with the options for the “Signal Source” visible are shown next.



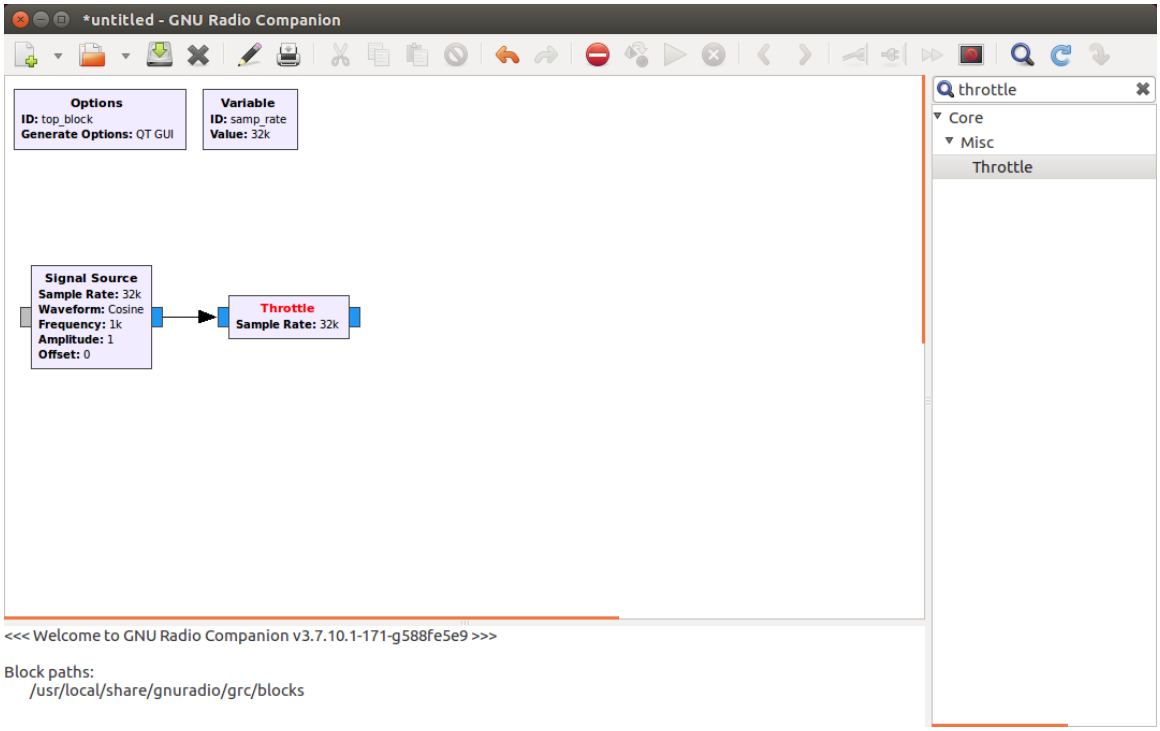
Notice that the “Signal Source” block has two ports, a grey one on the left and a blue one on the right. The color of a port indicates the type of data generated for an output port or the type of data accepted for an input. Click on “Help” on the menu bar of the GRC window and select “Types” to see the Color Mapping for different data types as shown in the screen snapshot below.



The most common data types that we will use are:

- Blue for complex-valued 32-bit floating point data samples (32 bits for each, real and imaginary part).
- Orange for real-valued 32-bit floating point data samples
- Blue-Green for real-valued 32-bit (long) integer data samples
- Yellow for real-valued 16-bit (short) integer data samples
- Magenta for real-valued 8-bit (byte) integer data samples

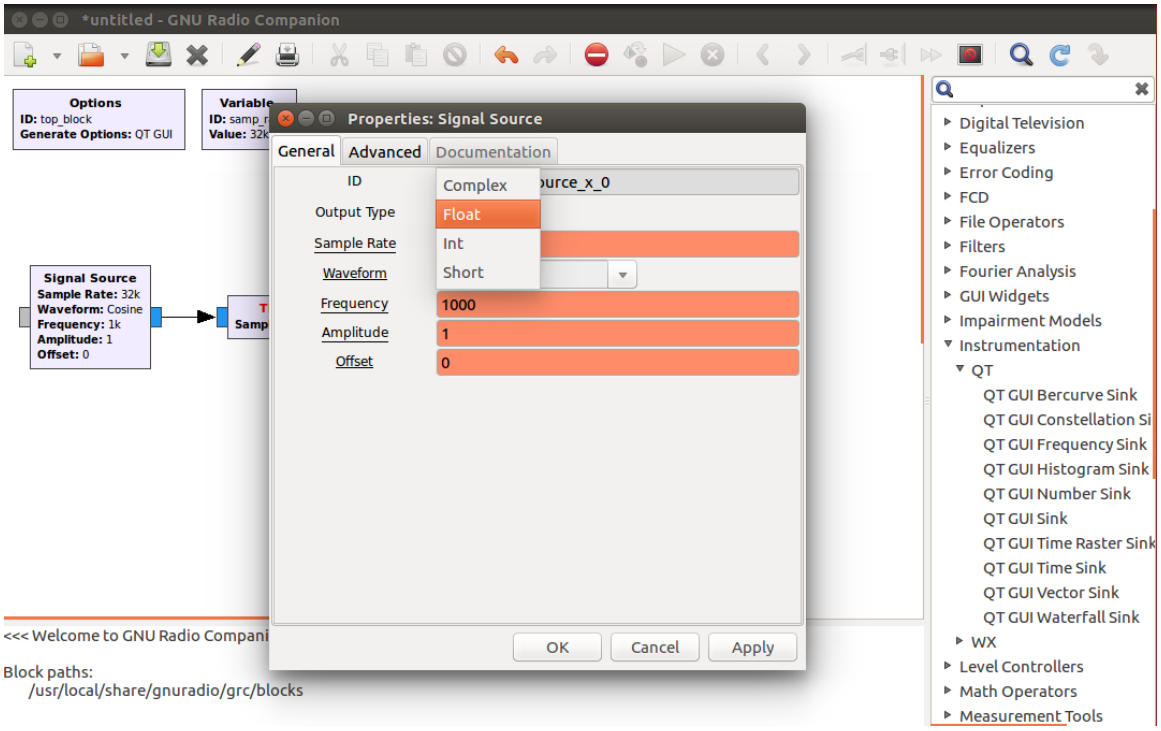
GNU Radio uses a stream processing model to process large amounts of data in real-time as opposed to a traditional array processing environment (like Matlab, for instance). In practice this means that each signal processing block has an independent scheduler running in its own execution thread and each block runs as fast as the CPU, dataflow, and buffer space allows. If there is a hardware source and/or sink that imposes a fixed rate (e.g., 44100 samples/sec for an audio signal, or 10 Msamples/sec for and SDR interface), then that determines the overall processing rate. But if both the source and the sink are implemented purely in software (like a signal generator feeding a time or frequency display), then some form of timing constraint must be imposed in software to limit the processing speed to a specified sampling rate. A special "Throttle" block that we will frequently encounter is used for this purpose. The figure below shows a "Throttle" block connected to the output of the "Signal Source" that we placed earlier.



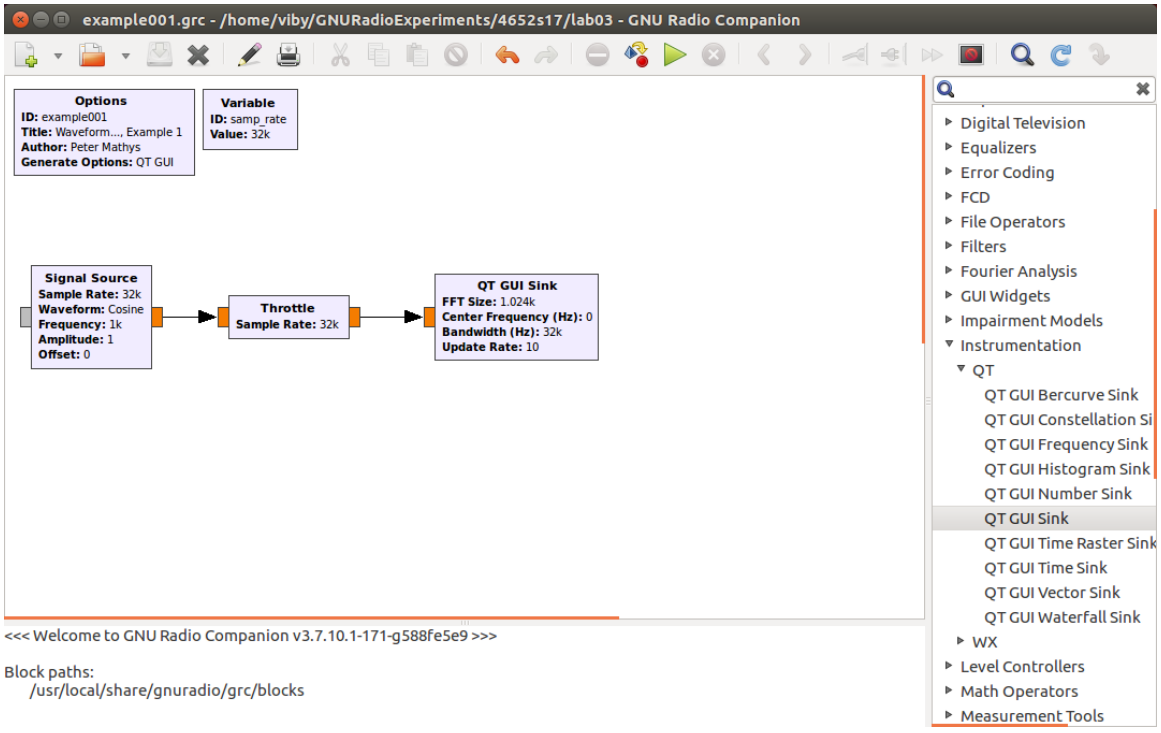
To wire the output port of one block to the input port of another block you click on one port followed by clicking on the other port. For the flowgraph that is constructed in this way to work, both ports must use data of the same type (i.e., both ports must be of the same color). If they are of different types, then the arrow of the connection will be red instead of black. Notice also that the word "Throttle" appears in red on the Throttle block above, indicating that there is something wrong with this block in the flowgraph. Things that can go wrong are unspecified or undefined parameters or, as is the case above, connections to/from some ports are missing. If you see any red arrows or red writing in a flowgraph you will not be able to run the flowgraph until the offending condition has been fixed.

1.2 A Cosine Waveform Generator in the GNU Radio Companion

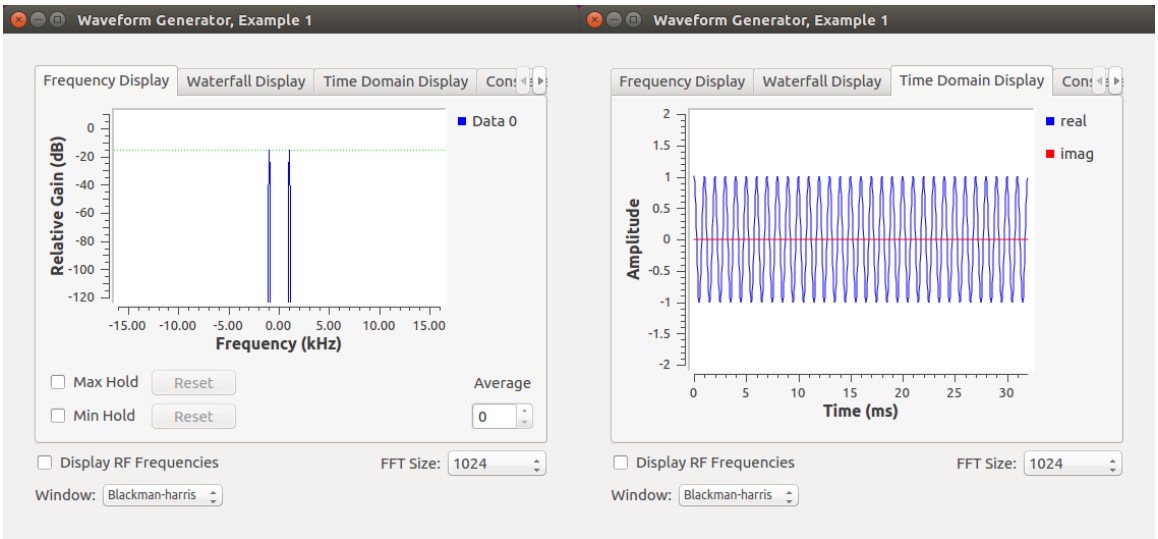
As a first experiment we want to generate a real-valued cosine signal with frequency 1000 Hz (default for the "Signal Source") and display it in the time and frequency domains. We start from a flowgraph which consists of a "Signal Source" connected to a "Throttle". To make the output of the Signal Source real-valued, double-click on the block and in the Properties window that shows up click on "Complex" under "Output Type" and select "Float" as shown below.



Repeat for the "Throttle Block". Then choose "QT" under "Instrumentation" (or just simply search for "QT GUI Sink") and double-click on "QT GUI Sink". This block will allow you to see the waveform at the input in the frequency as well as in the time domain. Change the data "Type" from "Complex" to "Real" and connect the input to the output of the "Throttle" block. Save the flowgraph, e.g., as `example001.grc`.



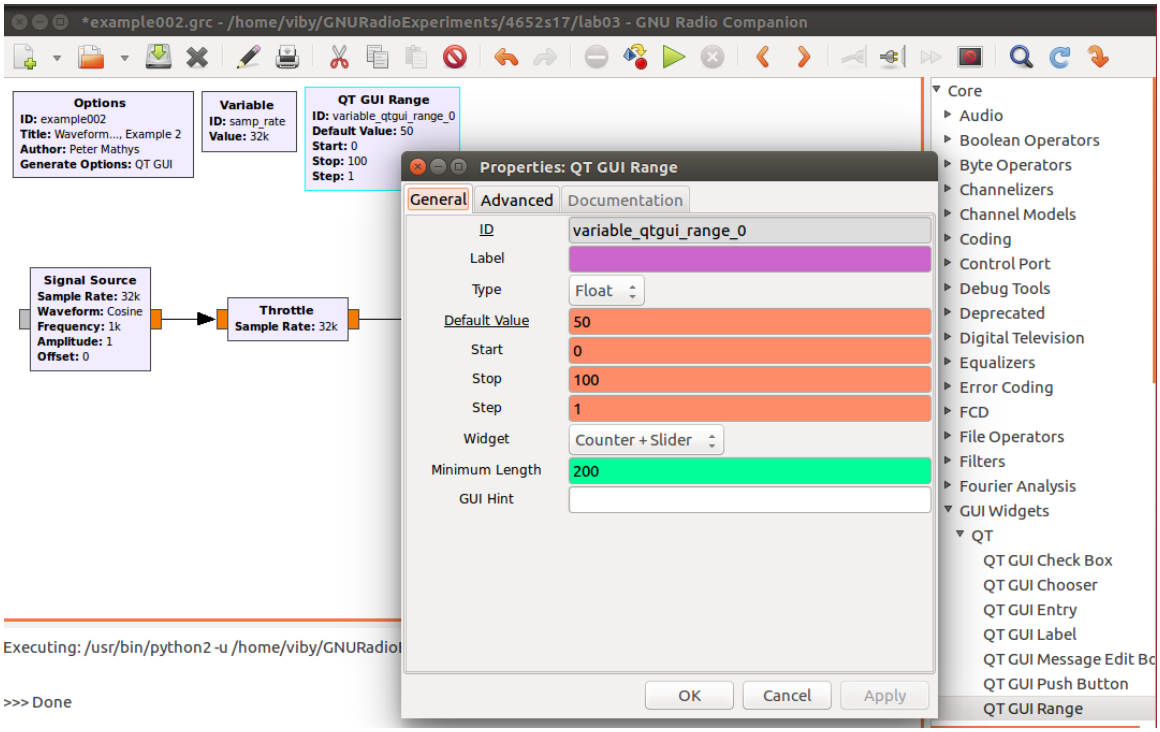
Now you can run the flowgraph by clicking on the green triangle above the canvas or by clicking "Run" on the menu bar. You can choose between the "Frequency Display" and the "Time Domain Display" tabs as shown below. Of course you can also try the other tabs and guess what they are displaying.



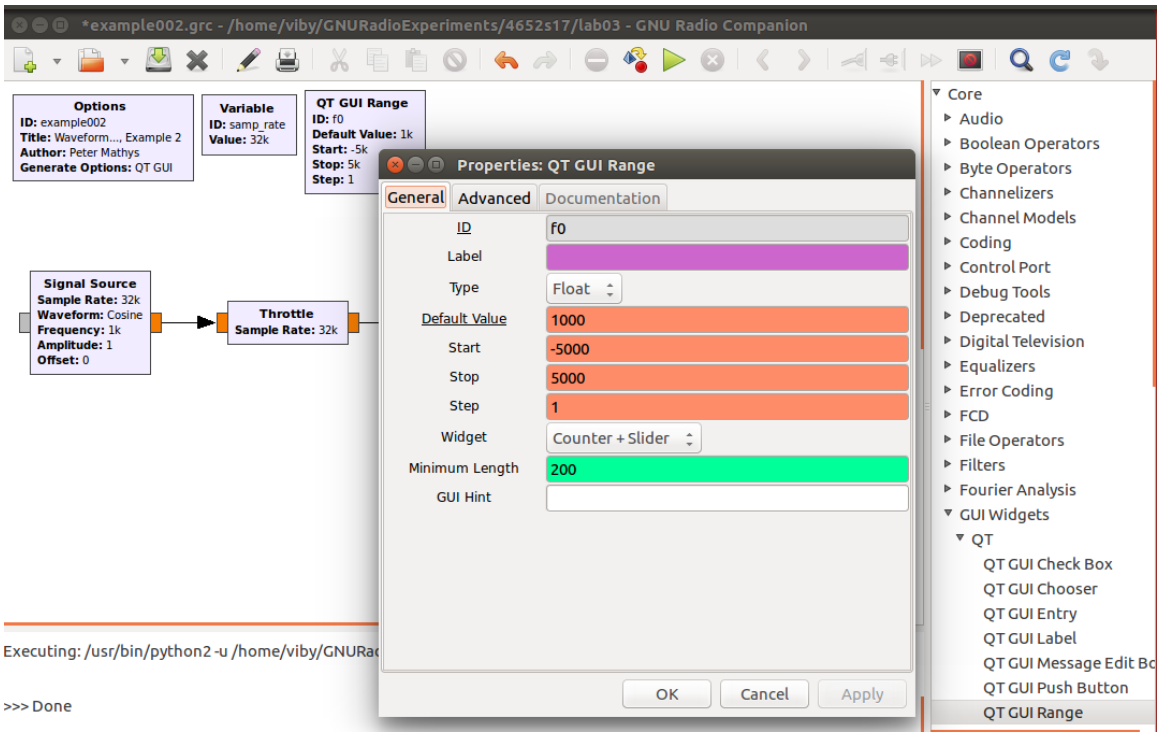
Use the cursor to zoom in on a rectangluar region, increase the FFT size to 4096 or 8192, choose different types of windows, e.g. "rectangular" or "Kaiser" and observe the effects, especially on the Frequency Display. Note that the Frequency Display shows power spectral density (PSD) which is essentially proportional to the magnitude squared of the Fourier transform.

1.3 A Cosine Waveform Generator with Variable Frequency and Sound

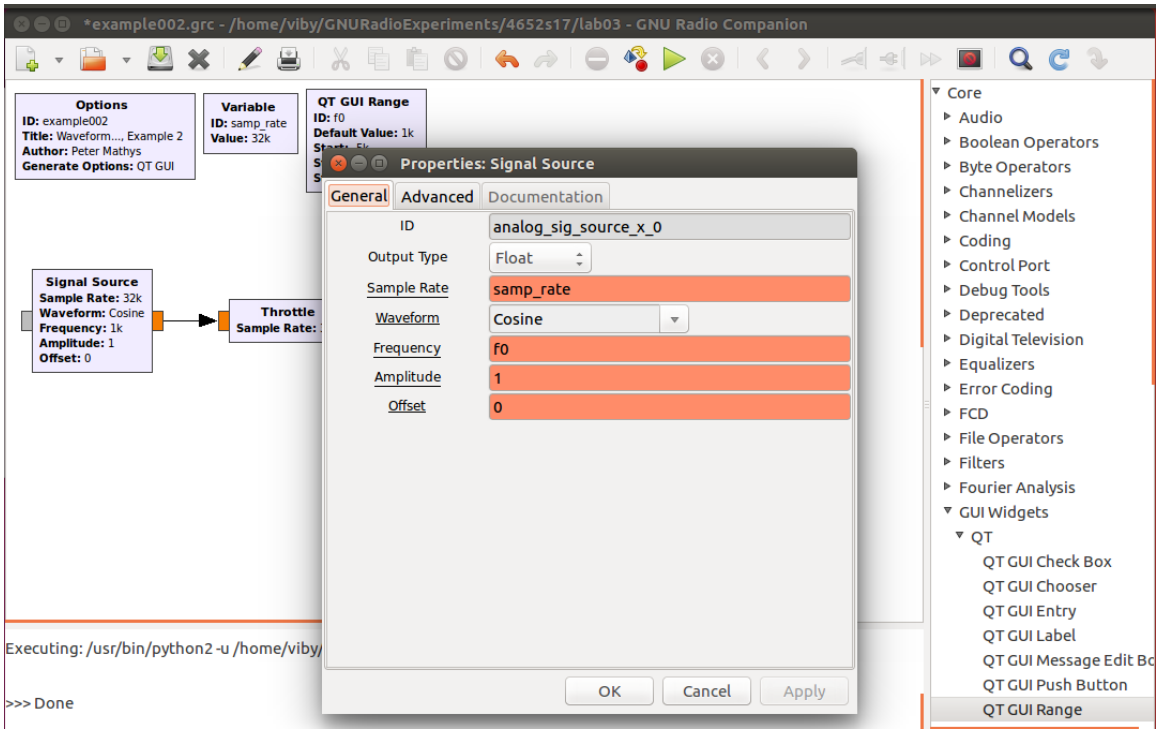
Start from the `example001.grc` flowgraph that consists of a "Signal Source", a "Throttle", and a "QT GUI Sink". Under "GUI Widgets" and "QT" select "QT GUI Range". Double-click on the block so that you get to see its Properties.



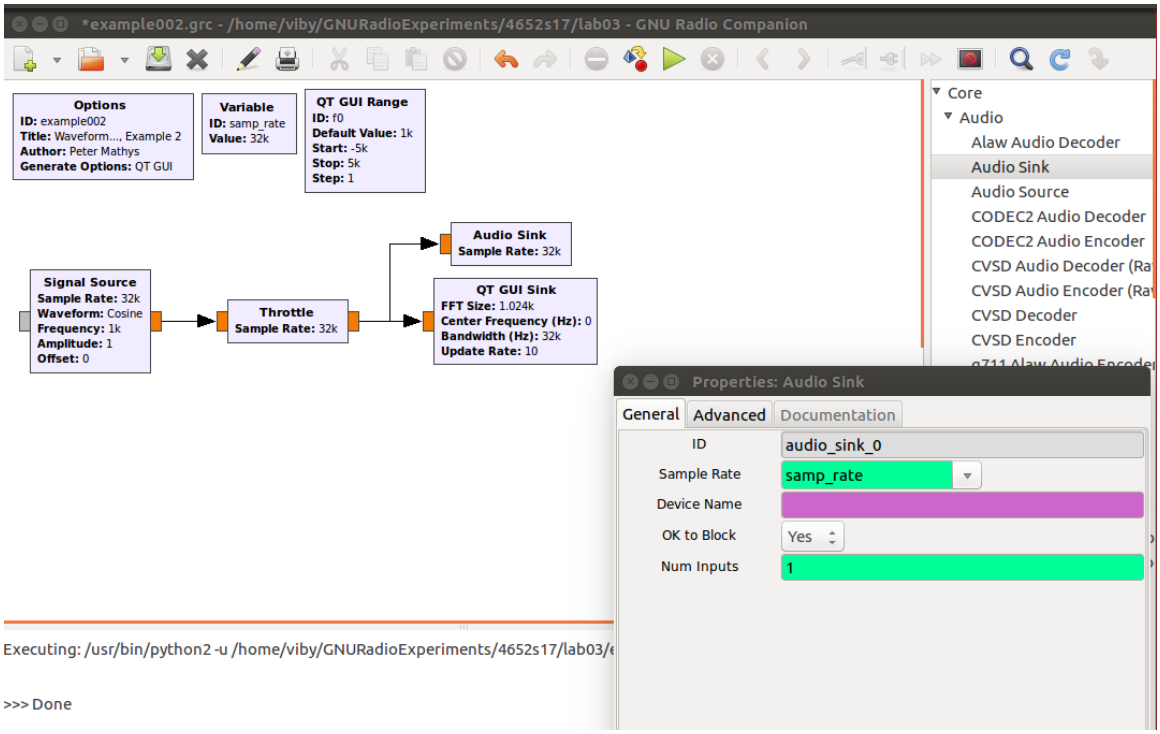
Change the "ID" from `variable_qtgui_range_0` to `f0`. For the "Default Value" enter `1000`. For the "Start" and the "Stop" values enter `-5000` and `5000`, respectively. The new Property values are shown below.



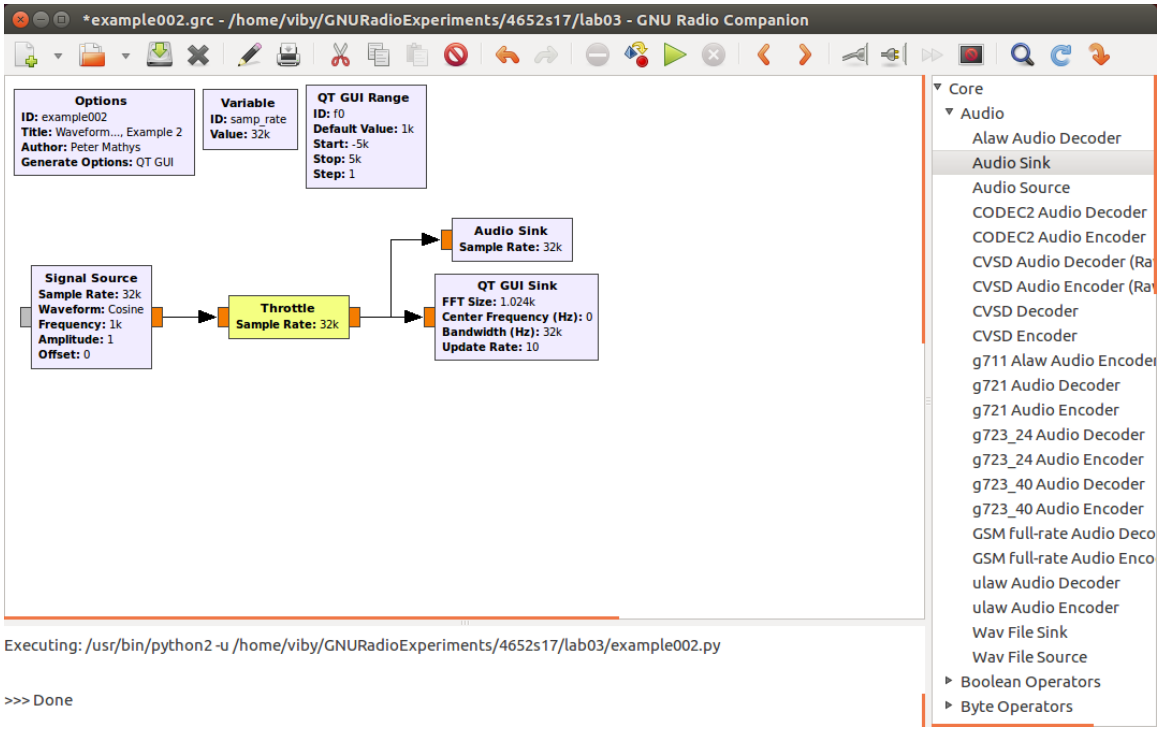
Next double-click on the "Signal Source" block and change the "Frequency" entry from `1000` to `f0`.



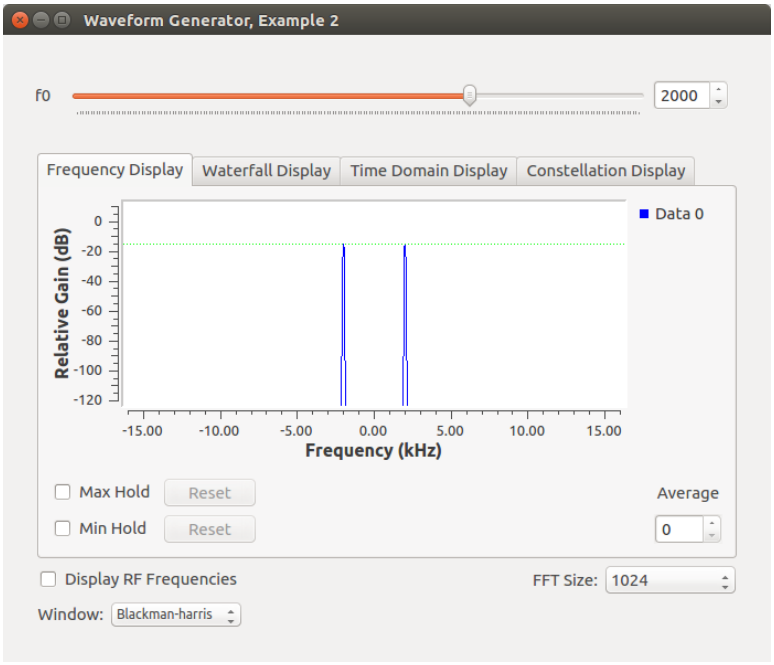
To add sound, select "Audio" and then double-click on "Audio Sink". Connect the "Audio Sink" input to the "Throttle" output and leave the "Sample Rate" at the default value of `samp_rate` (32000 samples/sec).



Since the “Audio Sink” is a rate-setting hardware device we now have a potential timing conflict between the “Throttle” and the “Audio Sink” blocks. For this reason we “Bypass” the “Throttle” block by selecting it and typing B on the keyboard (or by double-clicking and selecting “Bypass”). After that the finished flowgraph looks like this.



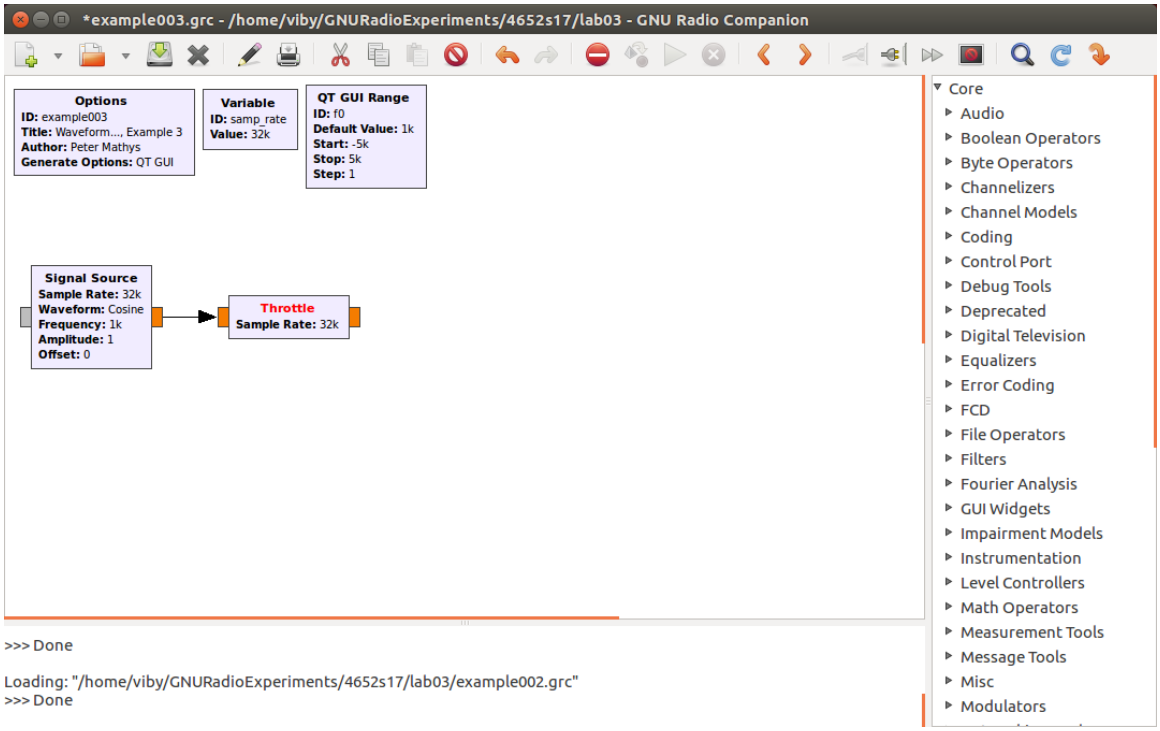
If you run the flowgraph now you will get a slider for changing the “Signal Source” frequency from -5000 to +5000 Hz, you will hear the coresponding sound, and you can choose to display the “Frequency” or the “Time Domain” graph.



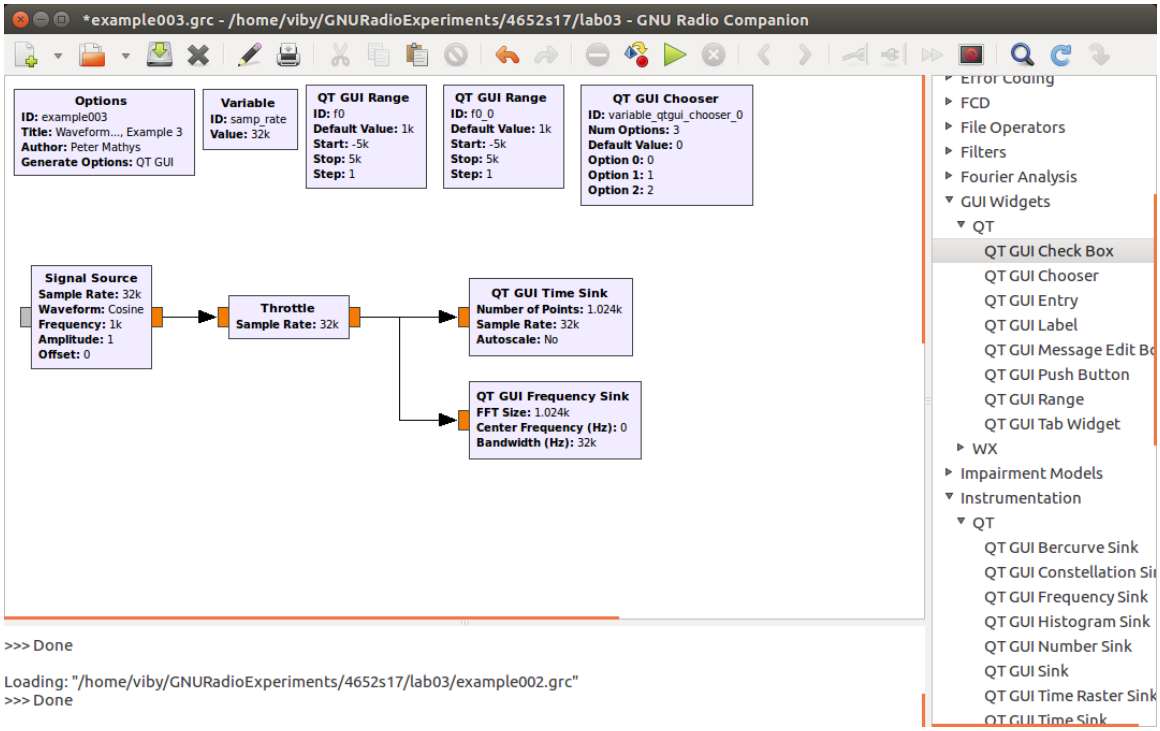
Do you hear or see any difference if you choose negative rather than positive frequencies? Why or why not?

1.4 More General Waveform Generator with Variable Frequency and Offset

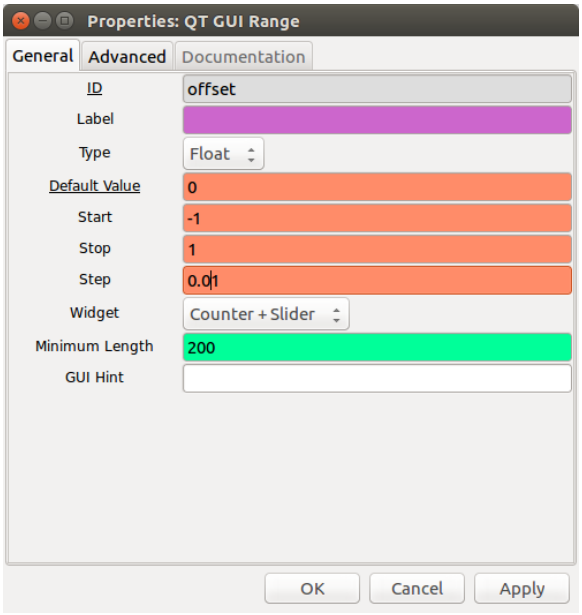
Start from the `example002.grc` flowgraph, remove the "Audio Sink" and the "QT GUI Sink" and re-enable the "Throttle" block (click on the block and then press the "e" key).



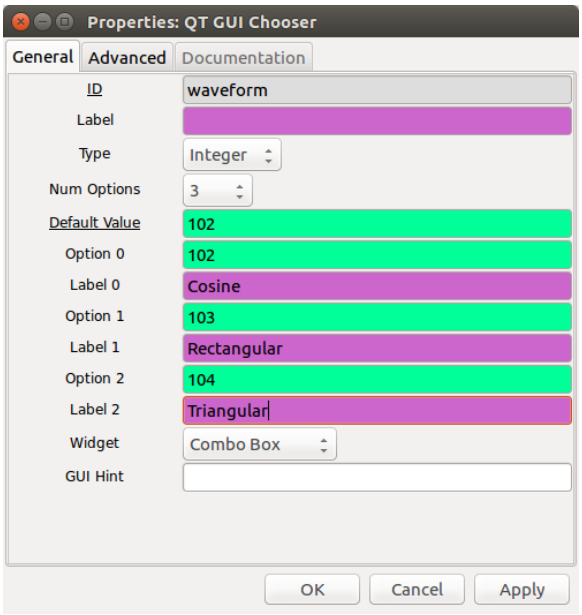
Now connect a "QT GUI Time Sink" and a "QT GUI Frequency Sink" (from "Instrumentation" and "QT") to the output of the "Throttle" Block. Change the input type of the two blocks from "Complex" to "Float". We would like to build a waveform generator that can produce real-valued "Cosine", "Rectangular", and "Triangular" waveforms with variable frequency and variable dc offset. To this end we need another "QT GUI Range" (right-click on the existing block, select "Copy" and then "Paste") and a "QT GUI Chooser (from "GUI Widgets" and "QT"). Your flowgraph should now be looking similar to the following.



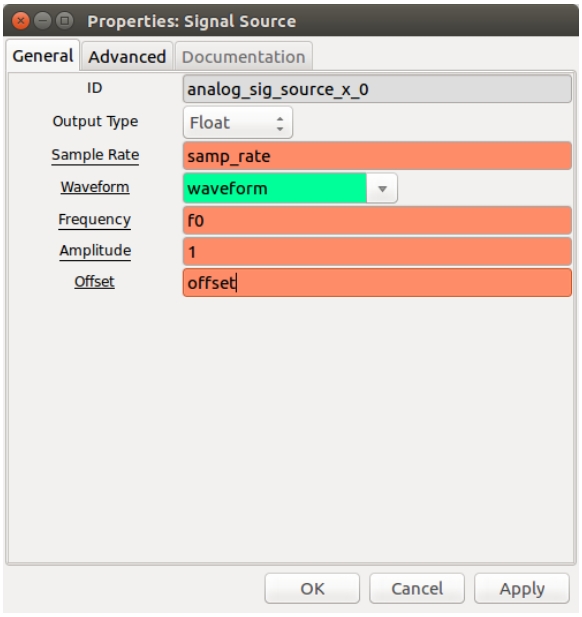
Now, double-click on the second "QT GUI Range" which will be used to adjust the offset of the waveform and modify the "Properties" as shown below.



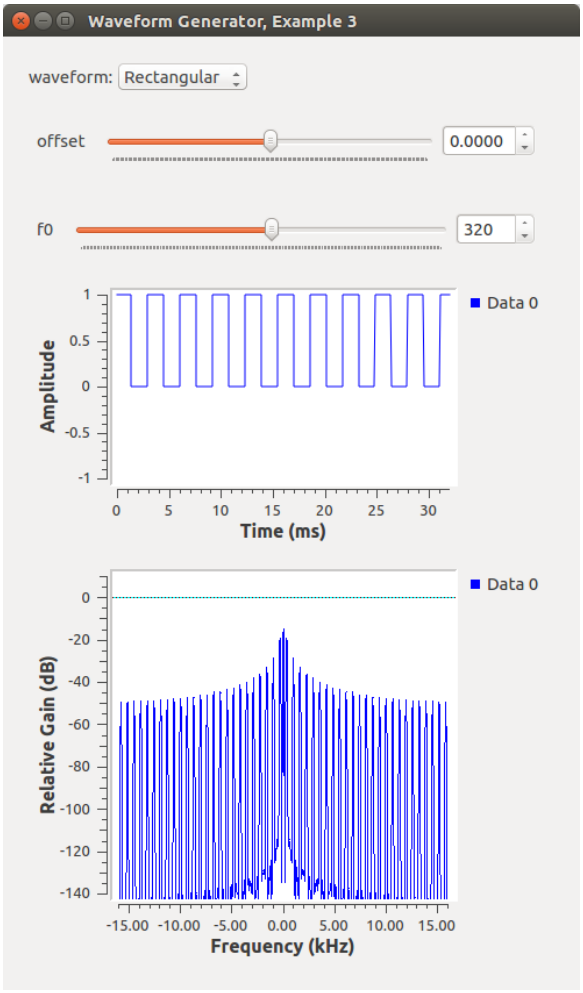
Next, double-click on the "QT GUI Chooser" that will be used to select different waveforms. The (integer) code for "Cosine" is 102, for "Rectangular" it is 103, and for "Triangle" it is 104. The completed "Properties" window looks like this.



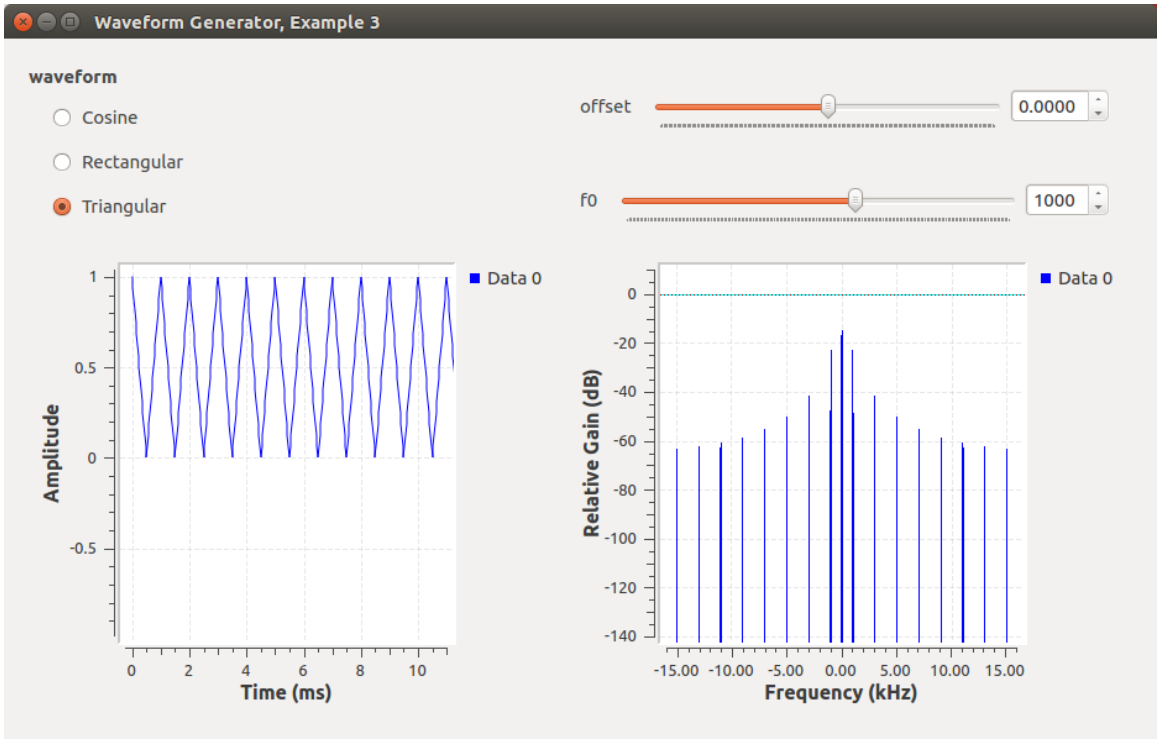
Finally, double-click on the "Signal Source" block and modify the "Properties" to look as follows.



Now we are ready for a first test run. Click the green triangle above the flowgraph or click on “Run” and “Execute” in the GRC menu bar. You should get a display similar to the one shown below (after playing with some of the parameters).



Most computer screens have landscape orientation and it would be nice to see the results displayed as shown in this screenshot.



It is possible to choose the arrangement of graphical elements, such as sliders, choosers, time and frequency sinks, etc., used in a GRC flowgraph by specifying grid positioning arguments in the "GUI Hint" fields of individual blocks. A grid positioning argument is a list of four integers of the form

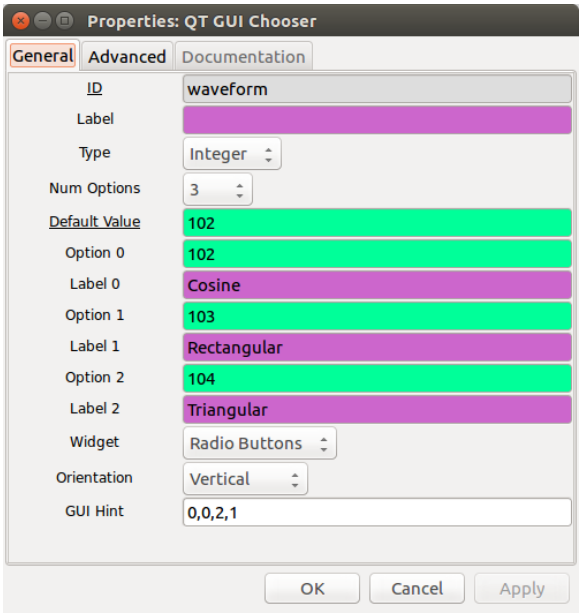
`(row, column, row span, column span)` .

If the "GUI Hint" entry is left blank, then the graphical elements are stacked vertically on top of each other. Otherwise, they are placed in the specified row `row` and the specified column `column`, spanning `rowspan` rows and `colspan` columns. Note that `rowspan` > 1 and `colspan` > 1 are required. The two tables below show the general `row` and `column` numbering on the left and the placement of the graphical elements with the corresponding "GUI Hint" values for this example on the right.

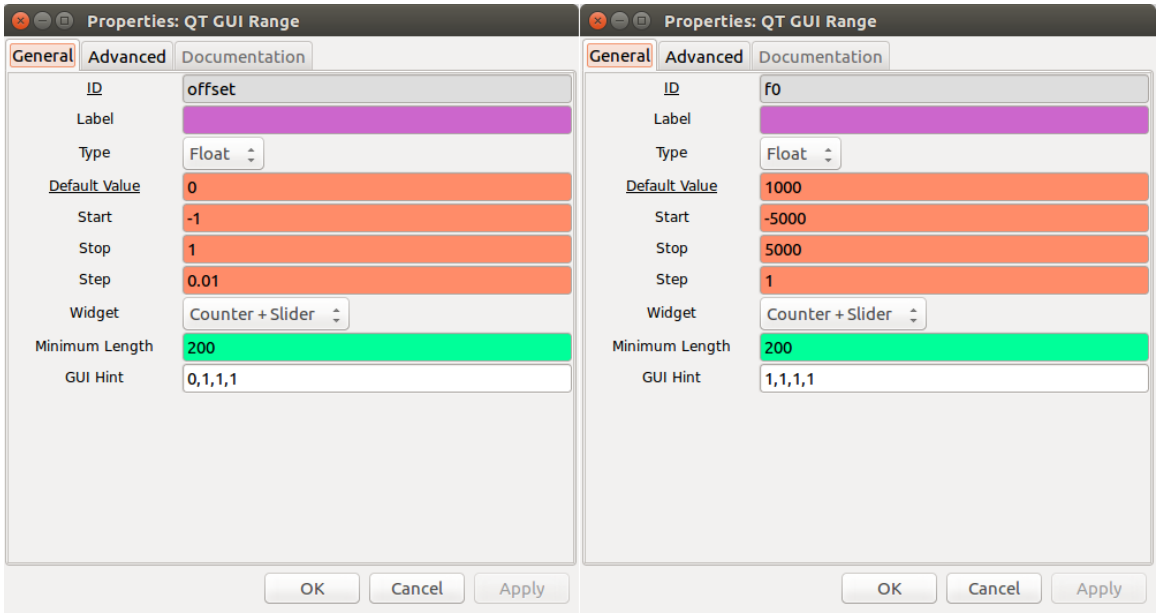
(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)

Waveform Selector (0,0,2,1)	Offset Slider (0,1,1,1)
	Frequency Slider (1,1,1,1)
Time Display (2,0,1,1)	Frequency Display (2,1,1,1)

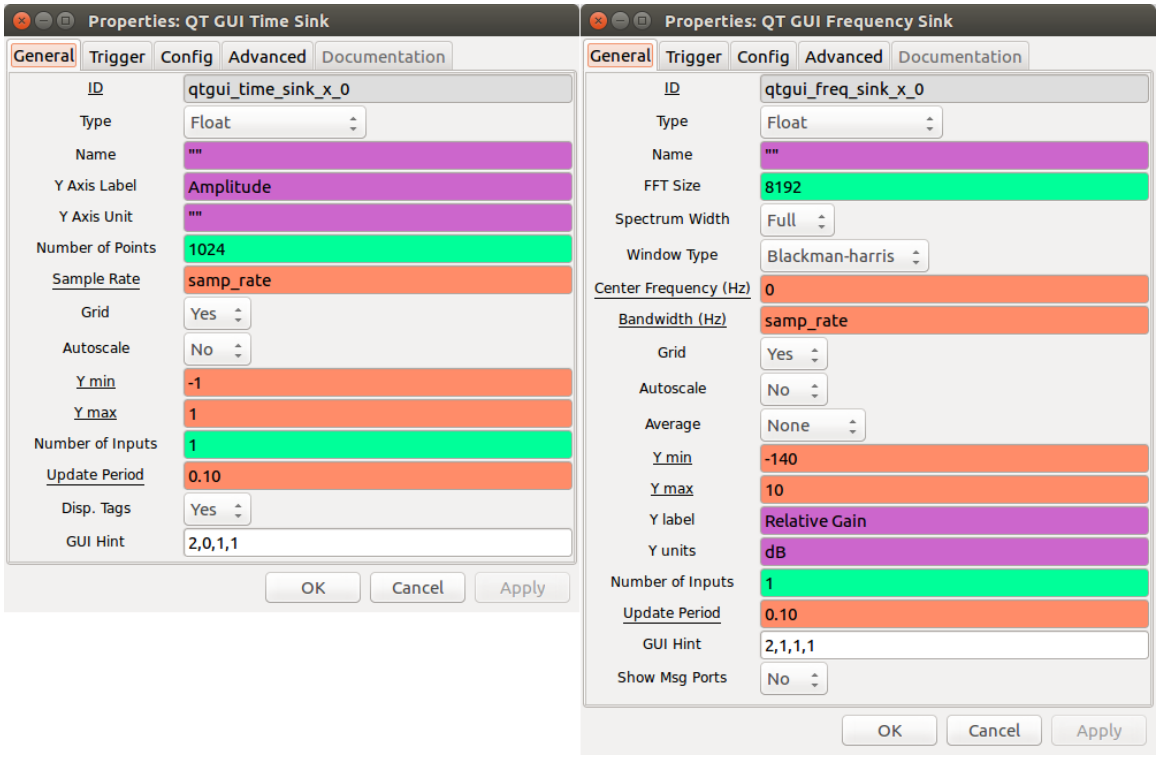
To obtain "Radio Buttons" (instead of a "Combo Box") for the waveform selector and place them in the desired position with respect to the other graphical elements for this example, the Properties window of the "QT GUI Chooser" is filled in as shown below.



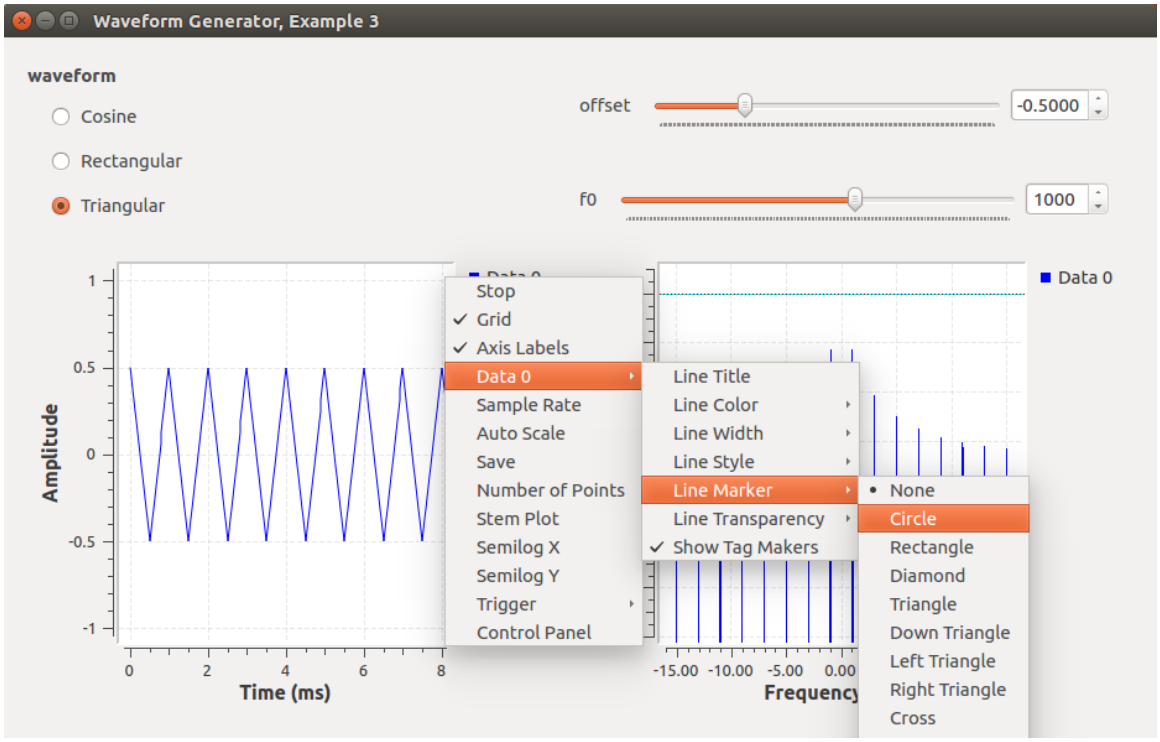
For the offset and frequency sliders the corresponding "QT GUI Range" block Property entries for the "GUI Hint" fields are set as shown next.



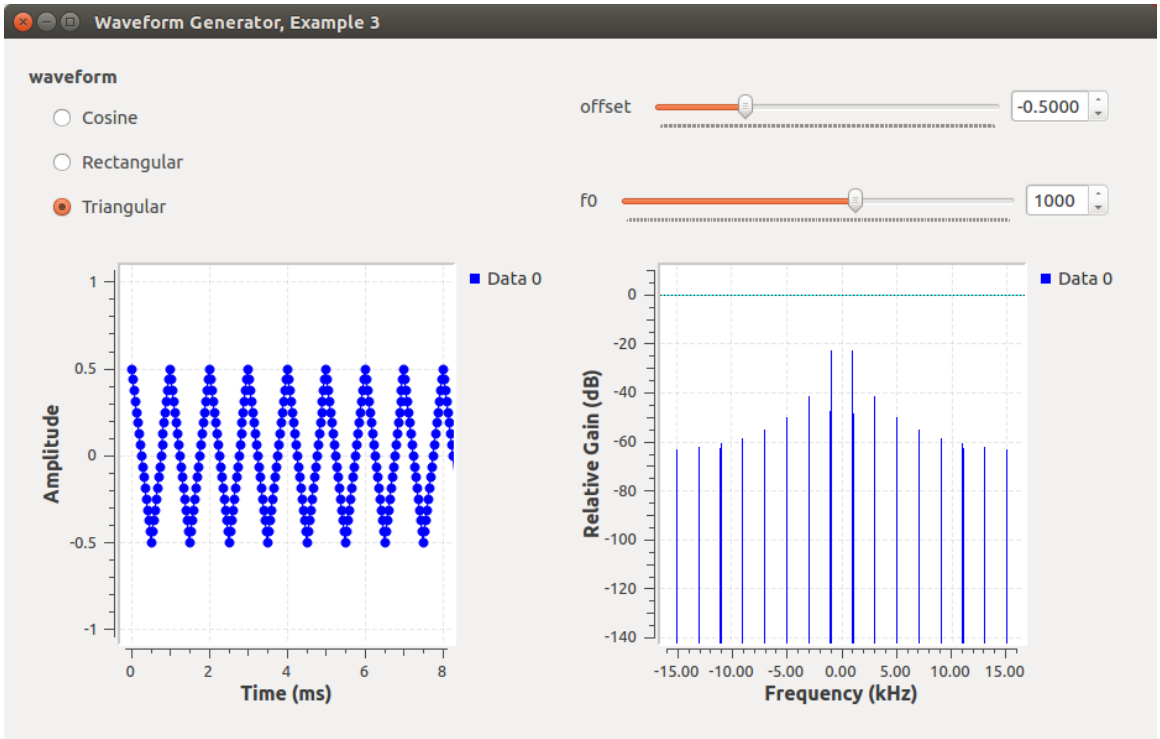
Finally, the Property windows of the "QT GUI Time Sink" and the "QT GUI Frequency Sink" are completed as follows.



Note that the "FFT Size" of the Frequency Sink has been increased to 8192 to improve the frequency resolution (from $32000/1024 = 31.25$ Hz to $32000/8192 = 3.9$ Hz). Also, "Grid" has been set to "Yes" for both, the Time and the Frequency Sinks, to draw grid lines and the "GUI Hint" fields have been filled in for both graphical elements as well. Further adjustments can be made by hovering over the Time or Frequency Sink display and then pressing the **middle** button of the mouse.

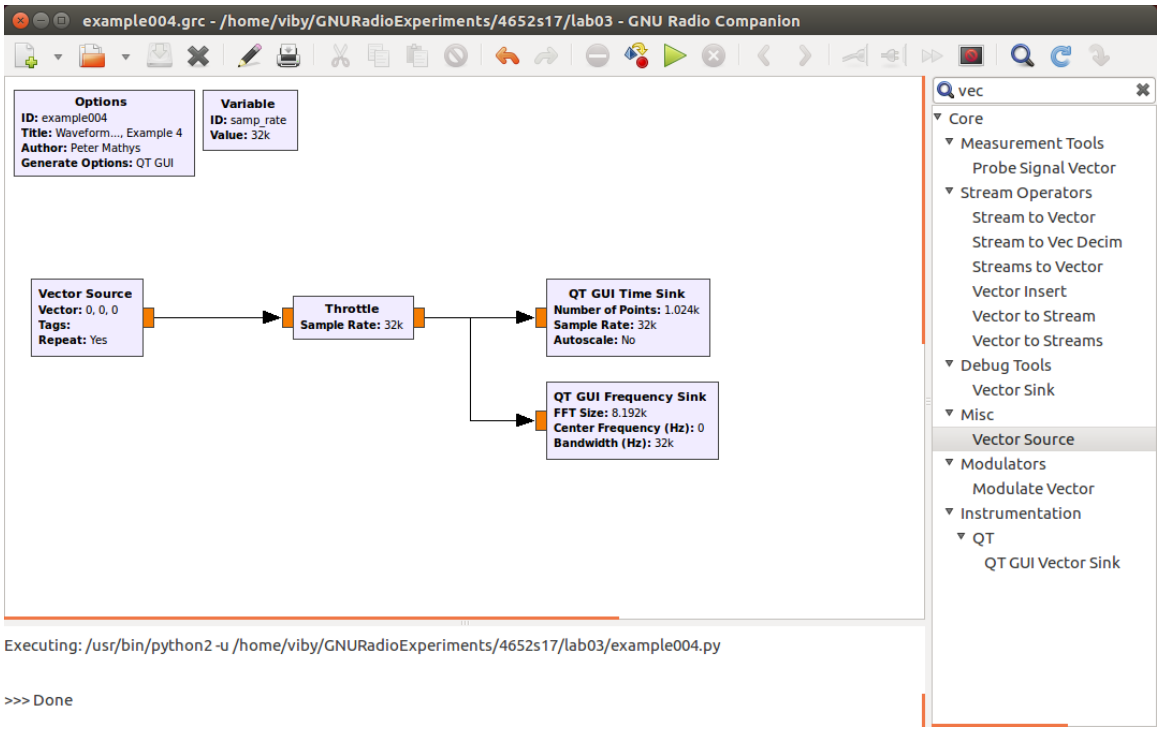


For example, to show the actual sampling points of a waveform plotted in the "QT GUI Time Sink" display, press the middle mouse button over the display, then select "Data 0", "Line Marker", and then "Circle" as shown above. This results in the display captured below.



1.5 Generating a Waveform Using a Vector Source

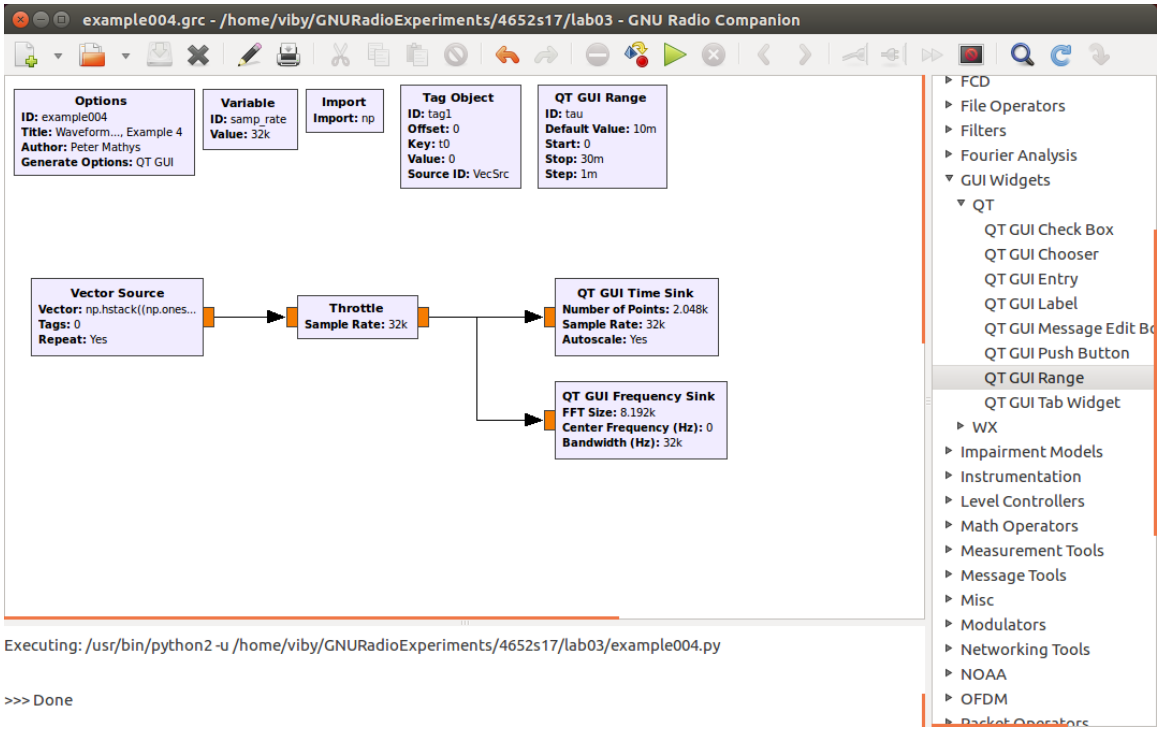
You can generate your own waveforms in the GRC whic h is useful for general testing purposes and for finding out what a specific block does by applying a specifically input and observing the resulting output. In this example we will use a "Vector Source" and Python to produce a rectangular pulse of variable width `tau`, and to look at how the PSD of the signal changes as `tau` is varied. Start the flowgraph from placing the following blocks on the canvas.



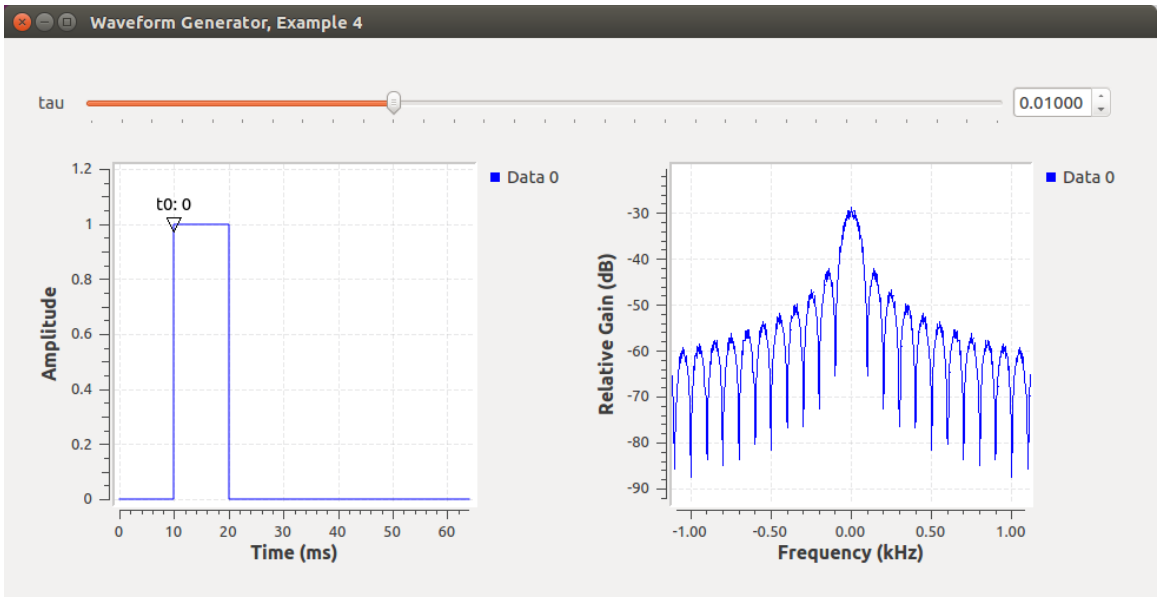
In Python code, the time function that we would like to generate is

```
np.hstack((np.ones(int(tau*samp_rate)), np.zeros(int((0.1-tau)*samp_rate))))
```

Here `tau` is the pulse width in the range of 0 to about 30 ms, the total width of the time function is 100 ms, and `np` stands for the `numpy` module that needs to be imported. The completed flowgraph looks like this.

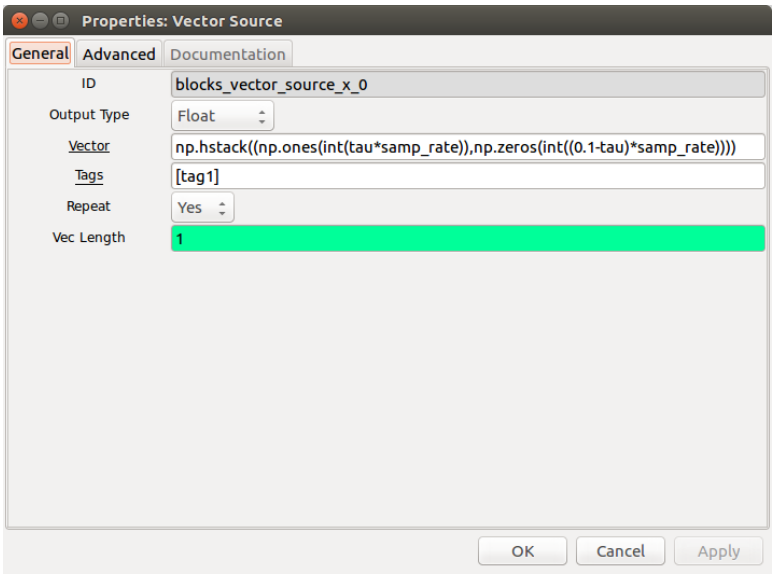


The next screenshot shows the results of running the flowgraph for `tau`=10 ms, after adjusting the Time and Frequency Sink displays using the zooming function and some of the functions that are accessible through the middle mouse button.

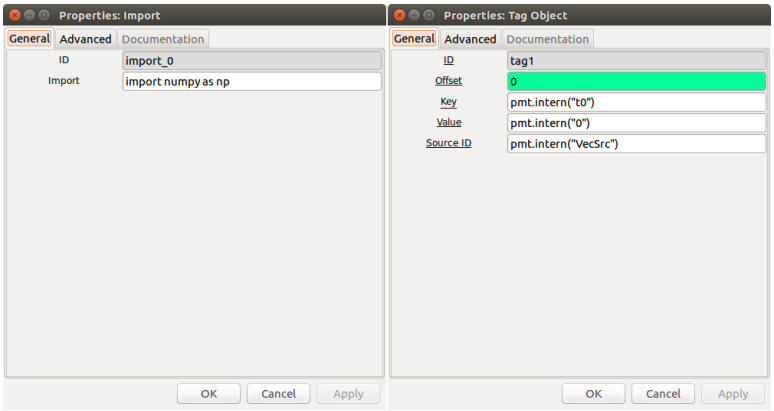


In particular, a stream tag (`tag1` with Key `t0` and Value `0`) is used to synchronize the Time and Frequency Sink displays (visible as `t0:0` in the Time Sink) with the data produced by the "Vector Source". A **stream tag** is produced by a block that generates stream data or samples. It flows downstream with a particular data sample (in this example with the first

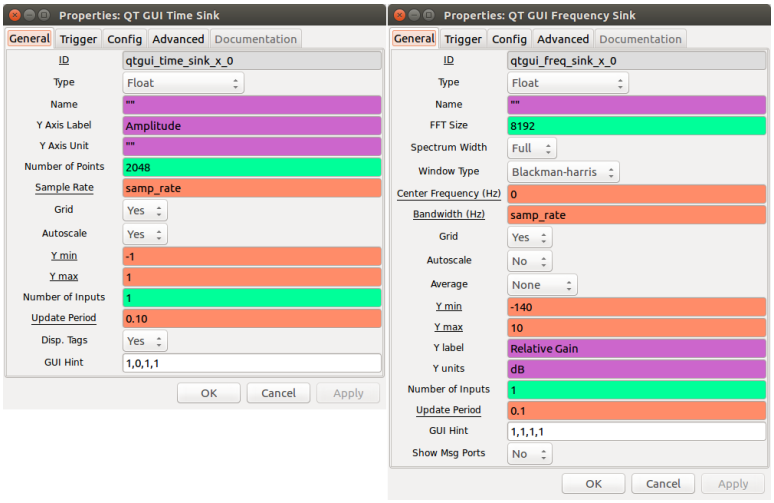
non-zero sample of the pulse generated by the Vector Source) until it reaches a sink (in this example the Time and Frequency Sinks) or is stopped on its way by some downstream block. Among the Properties of the Vector Source is the "Repeat" field which is set to "Yes" in this example so that the pulse of width `tau` is repeated periodically. The figure below shows the Properties of the "Vector Source".



The next figure shows the Properties of "Import" block which imports `numpy` as `np` and the "Tag Object" block which specifies `tag1`.



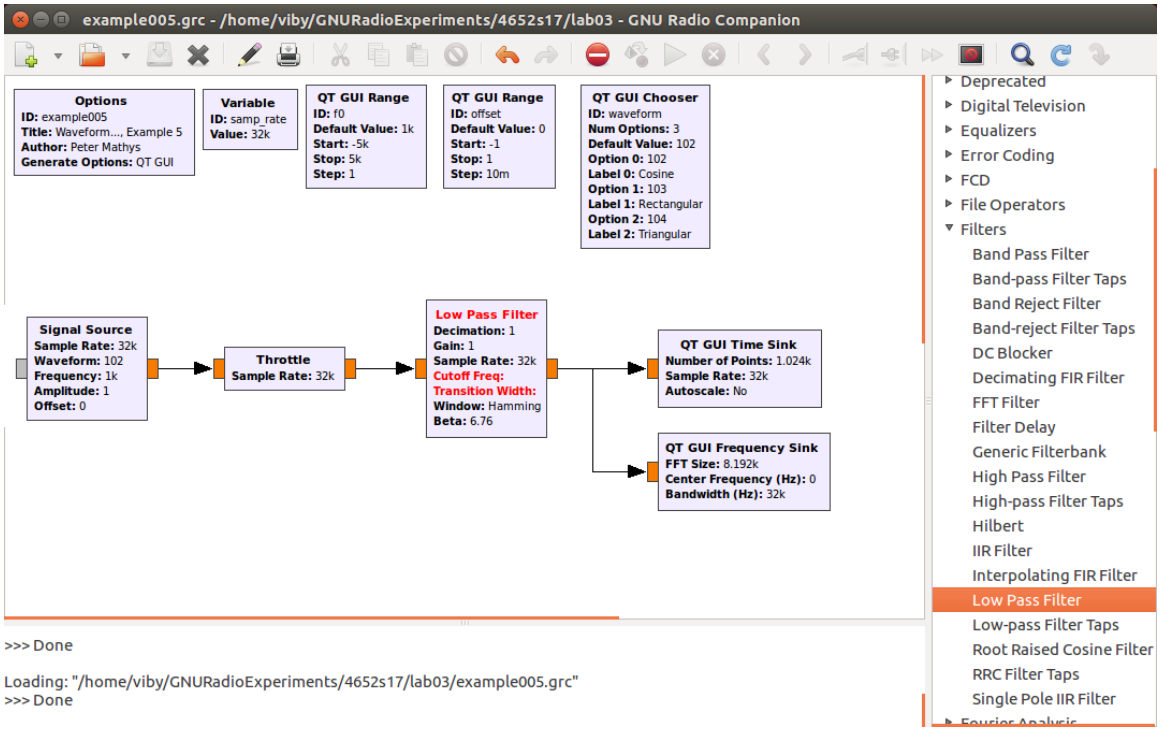
Finally, the Properties of QT GUI Sinks were filled in as follows.



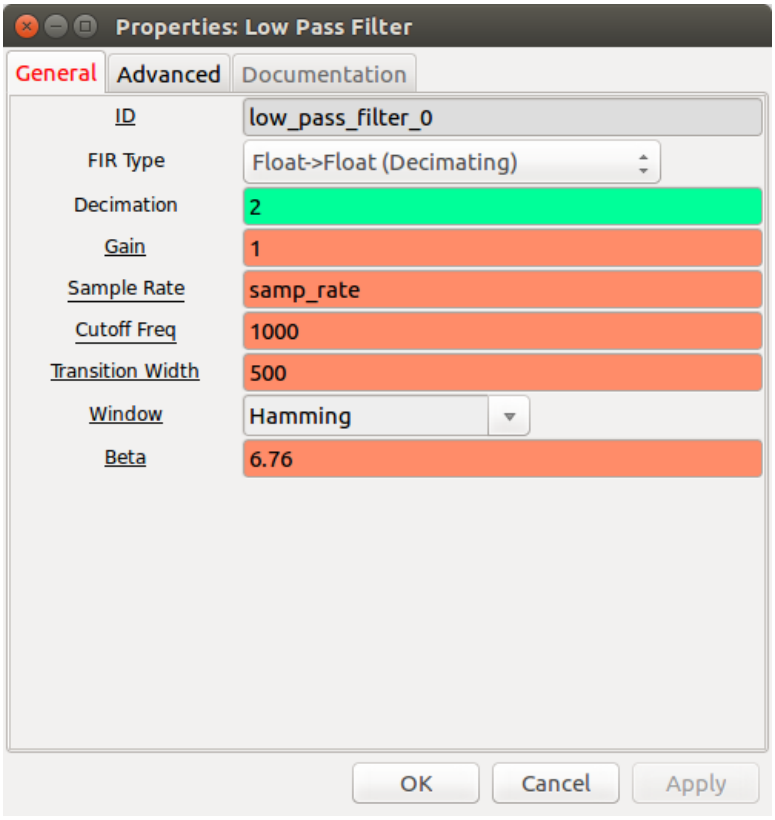
To synchronize the QT GUI Sink displays with the tag generated by the Vector Source, run the flowgraph and then click on the middle mouse button over the time or frequency sink. In the menu that pops up, go to "Trigger", select "Mode" and then left-click on "Tag". Enter the "Tag Key" in the window that pops up and then click "OK". If you want to delay the pulse in the time display, use again the middle mouse button over the sink, and select "Trigger" and "Delay". For this example, enter 0.01 in the "Delay" window that pops up and then click "OK".

1.6 Using a Lowpass Filter

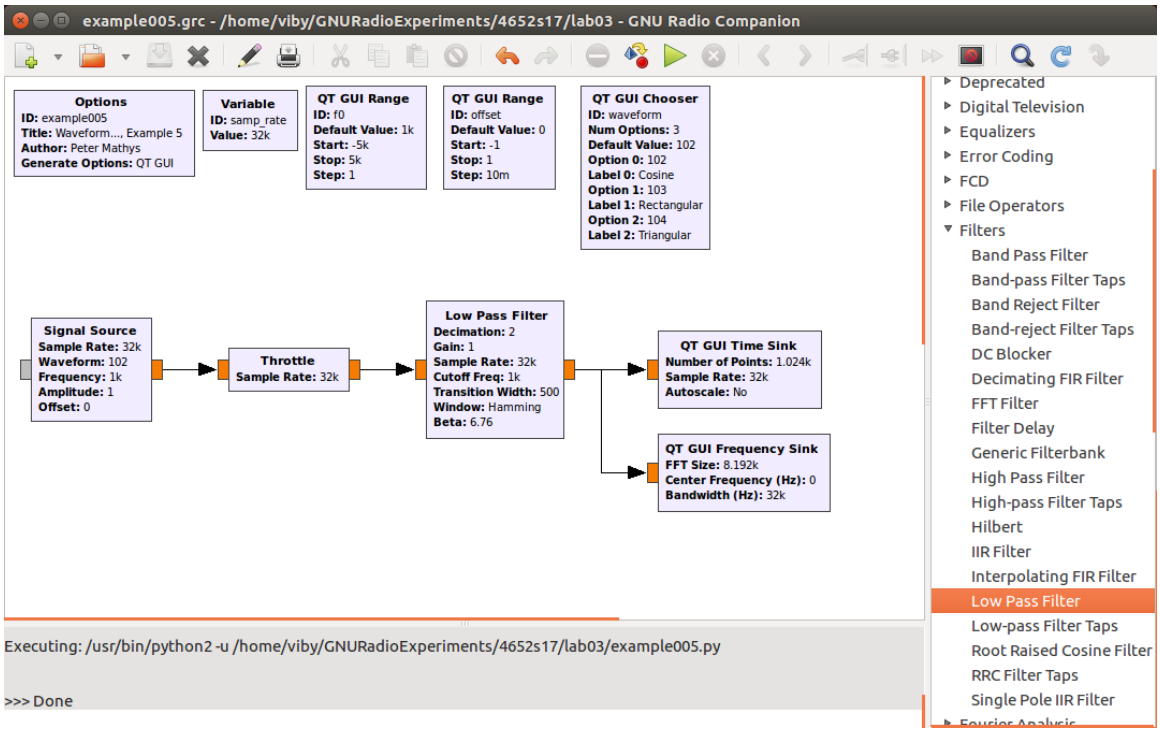
Start from the completed flowgraph of the "More General Waveform Generator with Variable Frequency and Offset and insert a "Low Pass Filter" from the "Filters" category as shown below. In the Properties window select the "FIR Type" as "Float− >Float (Decimating)".



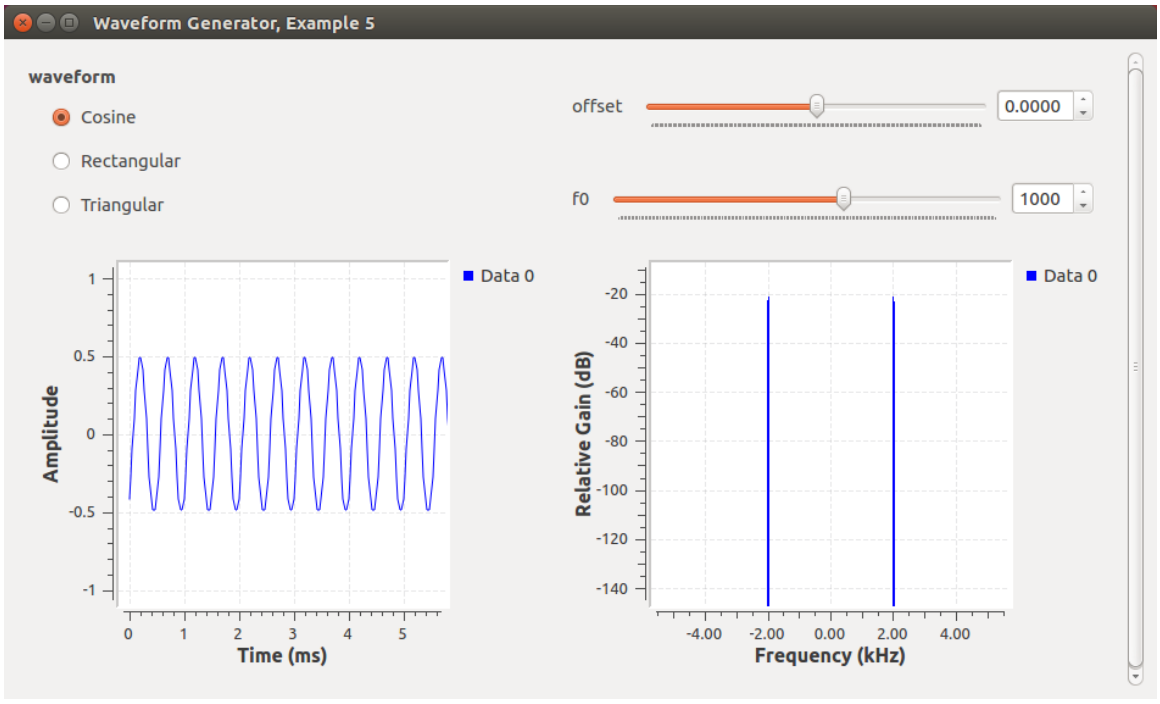
Fill in the Properties of the "Low Pass Filter" as shown below to obtain a lowpass filter (LPF) with cutoff frequency (-6 dB) of 1000 Hz and a transition band of width 500 Hz. Note that the decimation factor of the LPF is set to 2.



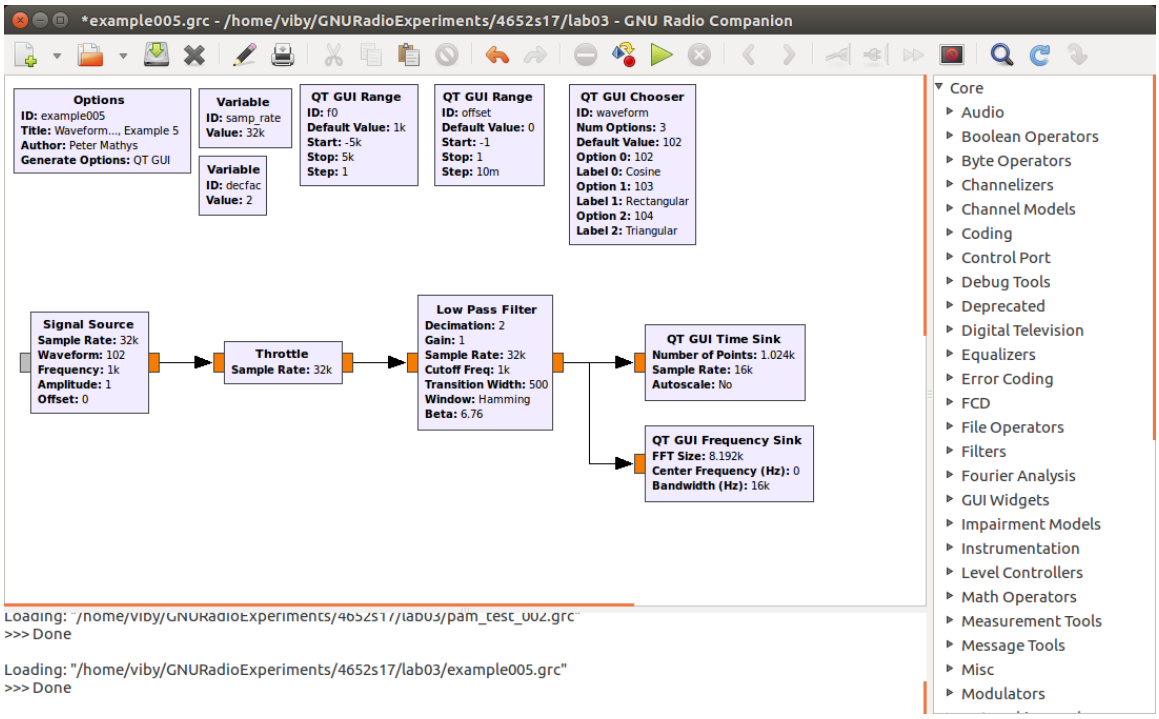
With the specification of the “Low Pass Filter” completed, the flowgraph now looks like that.



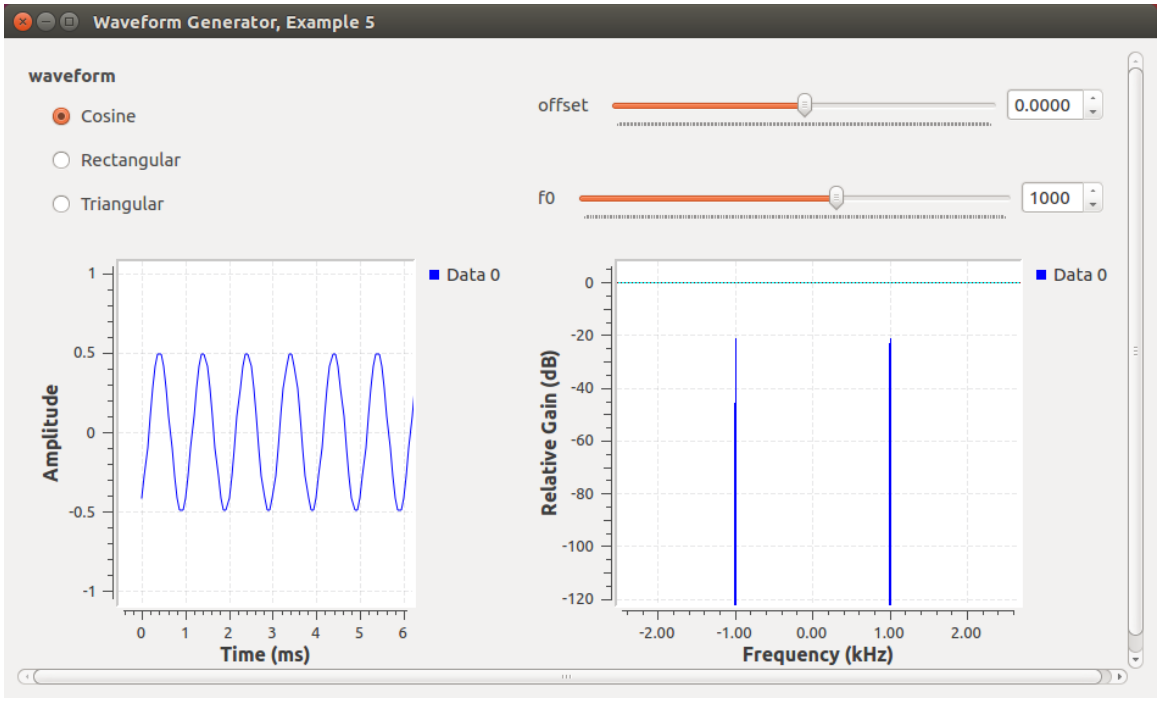
Running the flowgraph yields the following time and frequency displays.



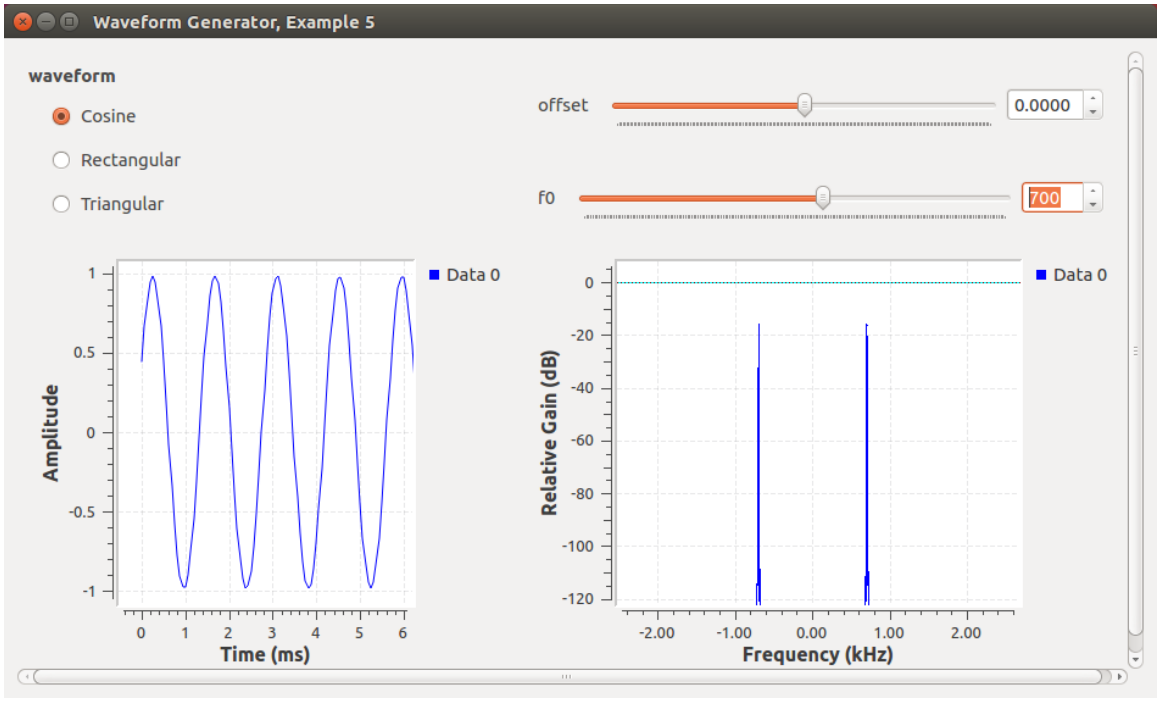
But note that, even though the frequency `f0` is set to 1000 Hz, the time display shows two periods per ms and the frequency display shows spectral lines at plus/minus 2 kHz. Why does this happen? Well, we specified a decimation factor of 2 for the "Low Pass Filter", but we didn't change the sample rate/bandwidth of the QT GUI Sinks accordingly. In the flowgraph below we defined a new variable `decfac` for this purpose and we changed the sample rate/bandwidth settings of the QT GUI Sinks to `sample_rate/decfac`



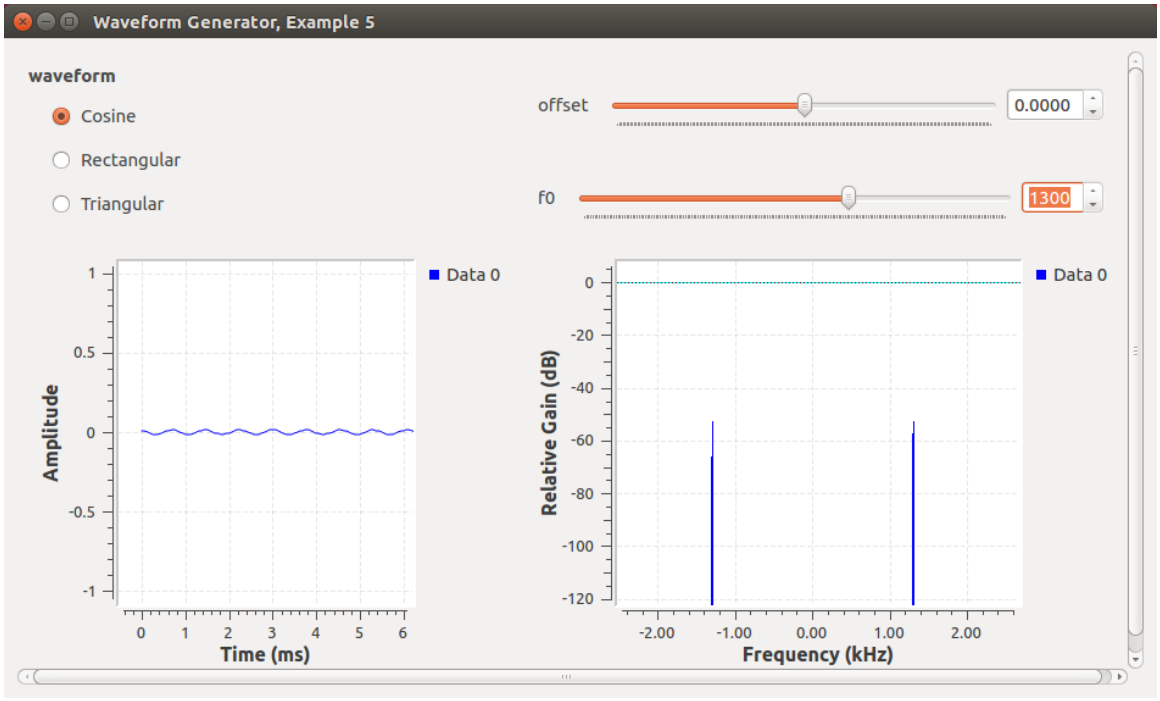
Now `f0` displays correctly as a 1 kHz frequency. Note that the waveform now only has an amplitude of 0.5 as a result of the cutoff frequency (-6 dB) being set as 1000 Hz in the "Low Pass Filter".



Reducing the frequency to 700 Hz restores the full amplitude of 1.0 at the output of the filter.

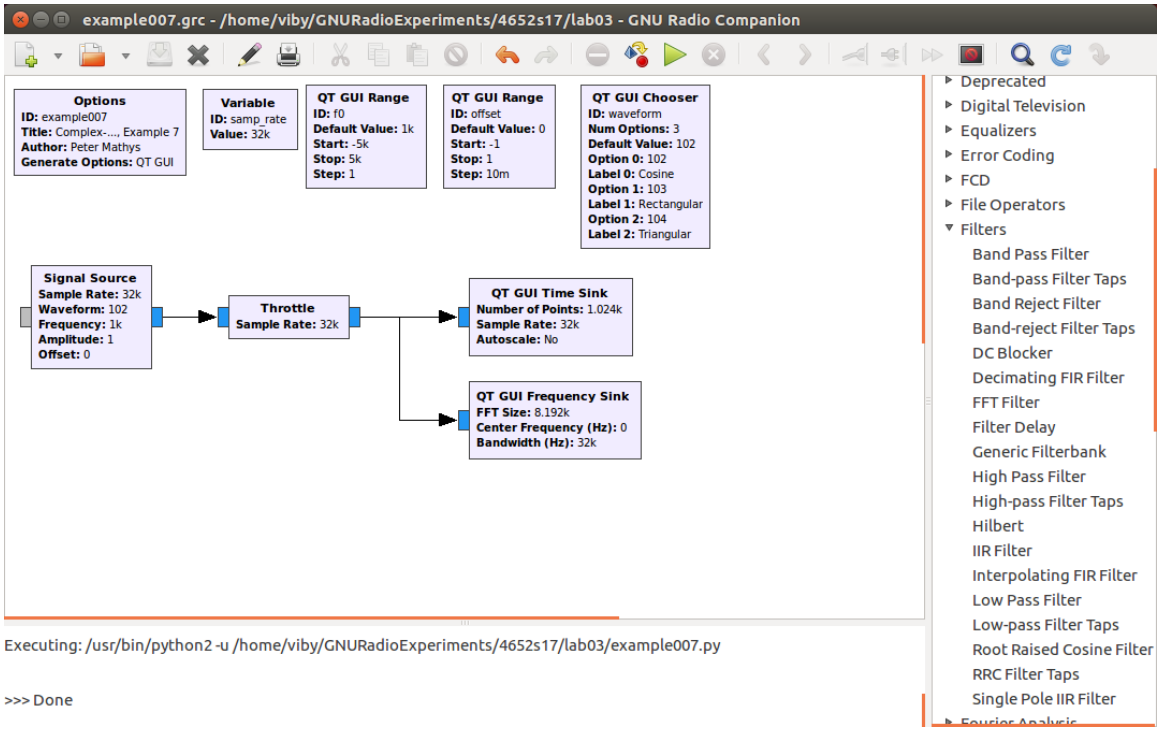


Increasing the frequency to 1300 Hz, on the other hand, reduces the amplitude to almost 0, as would be expected for a LPF with cutoff frequency 1000 Hz and a transition band of 500 Hz.

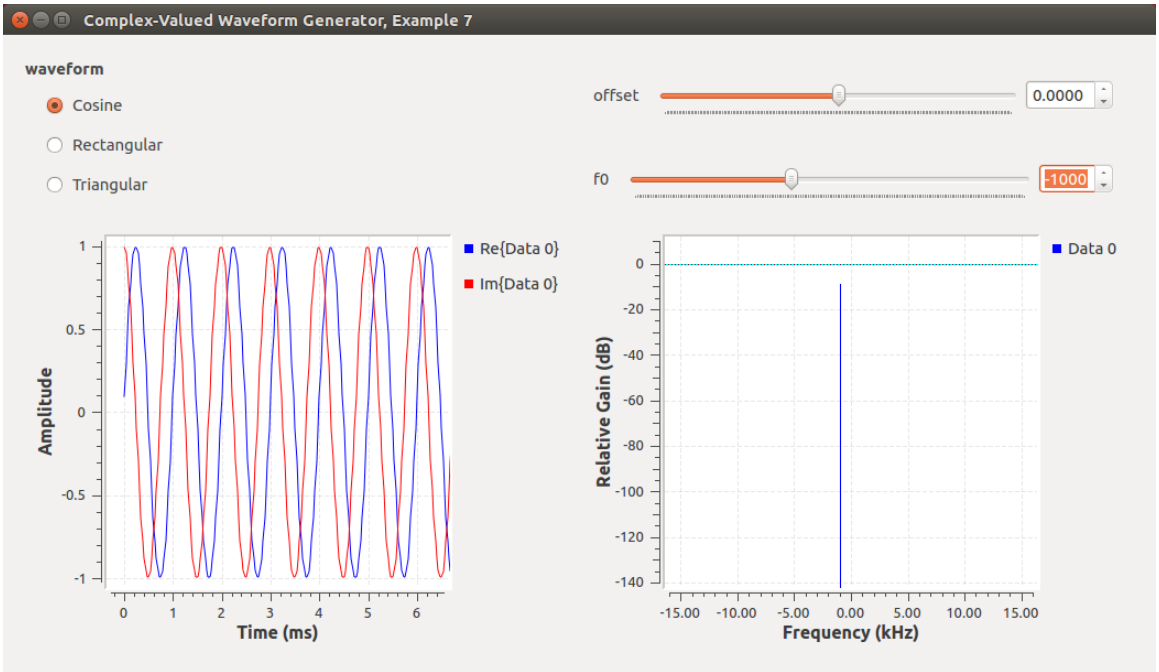


1.7 Complex-Valued Waveform Generator

Start from the completed flowgraph of the "More General Waveform Generator with Variable Frequency and Offset and change all input and output signal types from real (orange) to complex (blue) as shown below.

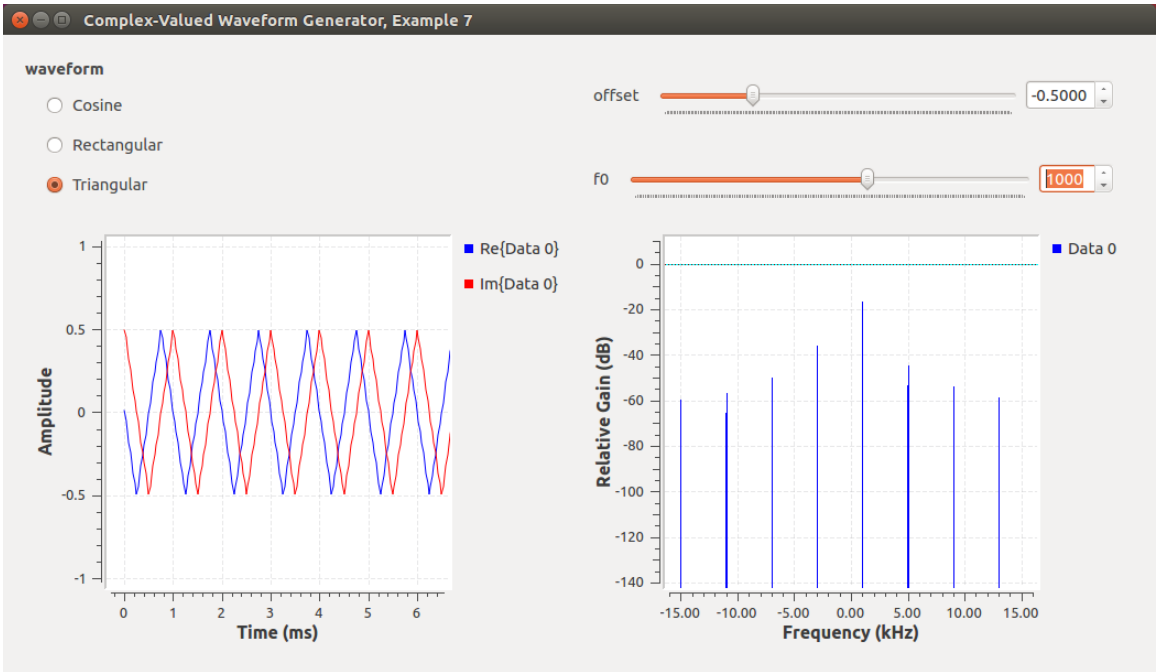


Running the flowgraph for a "Cosine" waveform produces the following graphs in the time and frequency domains.



Note that there is now only a single spectral line in the frequency display and specifying positive and negative frequencies gives different results. For the cosine waveform with zero offset the signal generated is $e^{j2\pi f_0 t}$.

The next two graphs show the time and frequency plots for a complex-valued triangular waveform of 1000 Hz with an offset of `-0.5*(1+1j)` (note that the offset has to be complex-valued if we want to be able to affect both, the real and the imaginary parts).



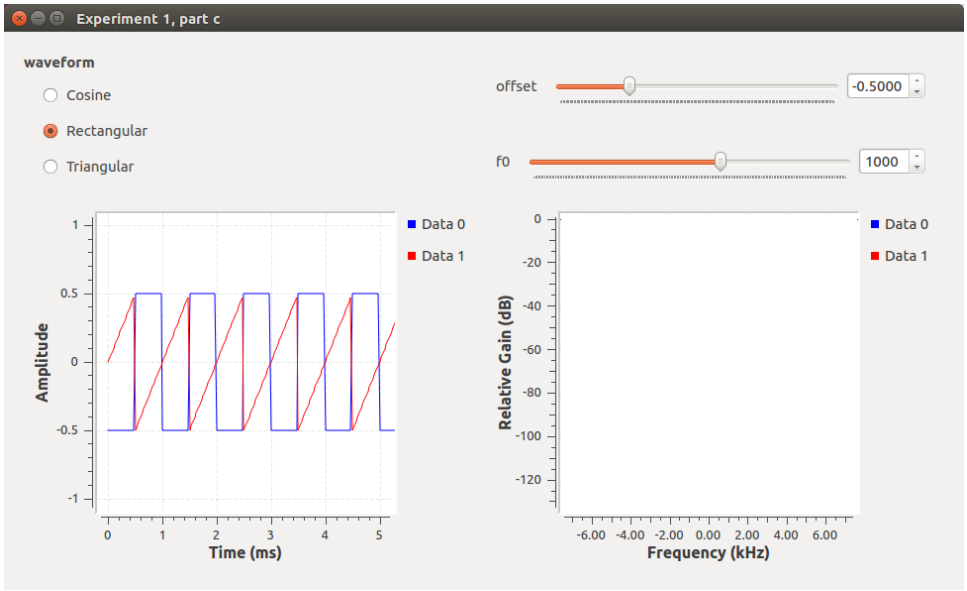
Notice that this triangular waveform has no spectral components at $\ldots, -5, -1, 3, 7, \ldots$ kHz. Can you explain why?

2 Lab Experiments

E1. Install GNU Radio and Try it Out. **(a)** Install VirtualBox, Ubuntu, and GNU Radio or bring an empty USB 3.0 flashdrive with 32 GB or more capacity to which an image of Ubuntu and GNU Radio can be transferred.

(b) Work through and recreate the examples given in the "Getting Started with GNU Radio Companion" section. Familiarize yourself with the basic blocks that are available in the GRC and feel free to experiment, e.g., by generating 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 to what you would expect them to be.

(c) Comparison of cosine, rectangular, and triangular PSDs to sawtooth PSD. Use the "More General Waveform Generator with Variable Frequency and Offset" given in the introduction to generate cosine, rectangular, and triangular waveforms. Use a second "Signal Source" to generate a "Saw Tooth" waveform with the same frequency and the same offset as the general waveform generator. To compare the sawtooth waveform to the other waveforms in the time and frequency domains, add a second input to the "QT GUI Time Sink" and the "QT GUI Frequency Sink". This is achieved by changing the "Number of Inputs" in the property windows of the QT GUI Sinks from 1 to 2. An example displaying a rectangular and a sawtooth waveform in the Time Sink is shown below.

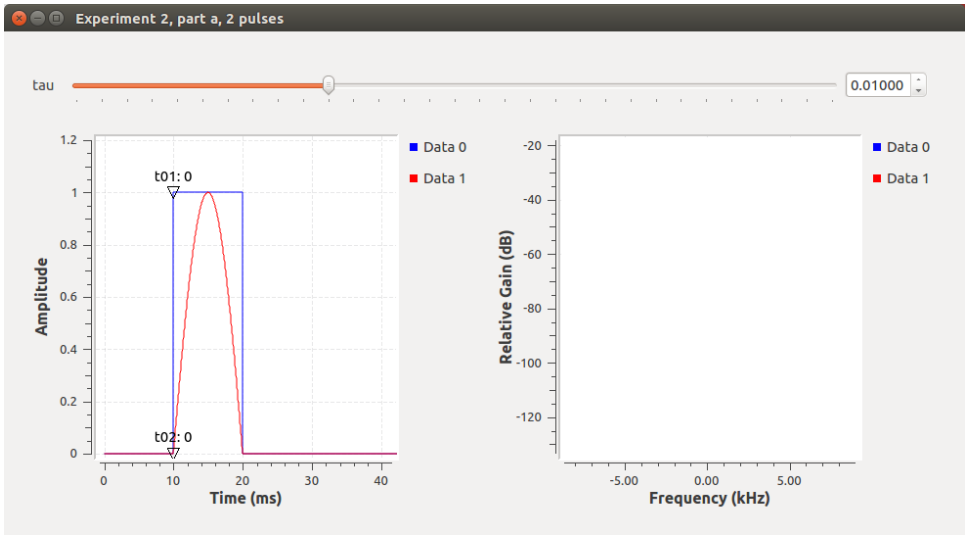


Note that Frequency Sink display is deliberately left blank in the figure above. In your implementation you should see the PSD of the two waveforms that you are comparing. Characterize the differences in terms of which spectral lines are present and in terms of how fast (in dB per decade) the the spectra decay with increasing frequency.

E2. Generation of Test Signals in GNU Radio. (a) Use a "Vector Source" to generate a sinusoidal pulse of the form

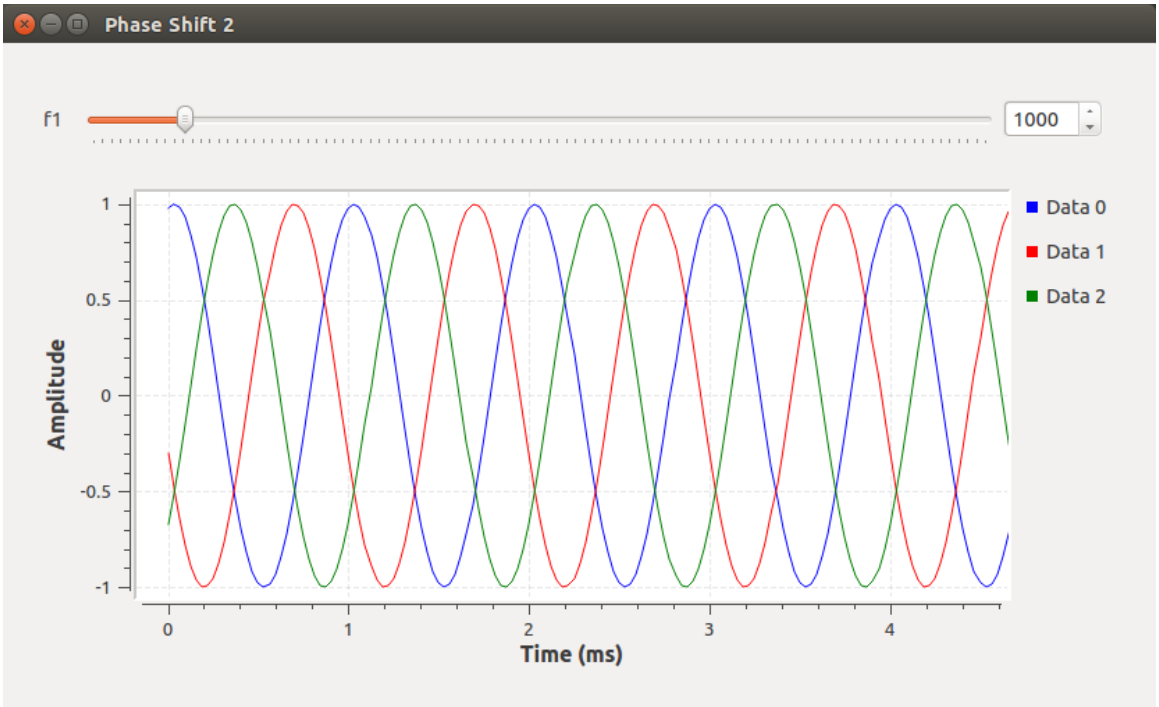
$$p(t) = \begin{cases} \sin(\pi t/\tau)\,, & 0 \le t < \tau\,, \\ 0\,, & \text{otherwise}\,, \end{cases}$$

of adjustable width `tau` with the same settings as the rectangular pulse example in the introduction. Display both, the rectangular and the sinusoidal pulse in the time and frequency domains as shown below (note that the frequency display is deliberately left blank and left for you to fill in).



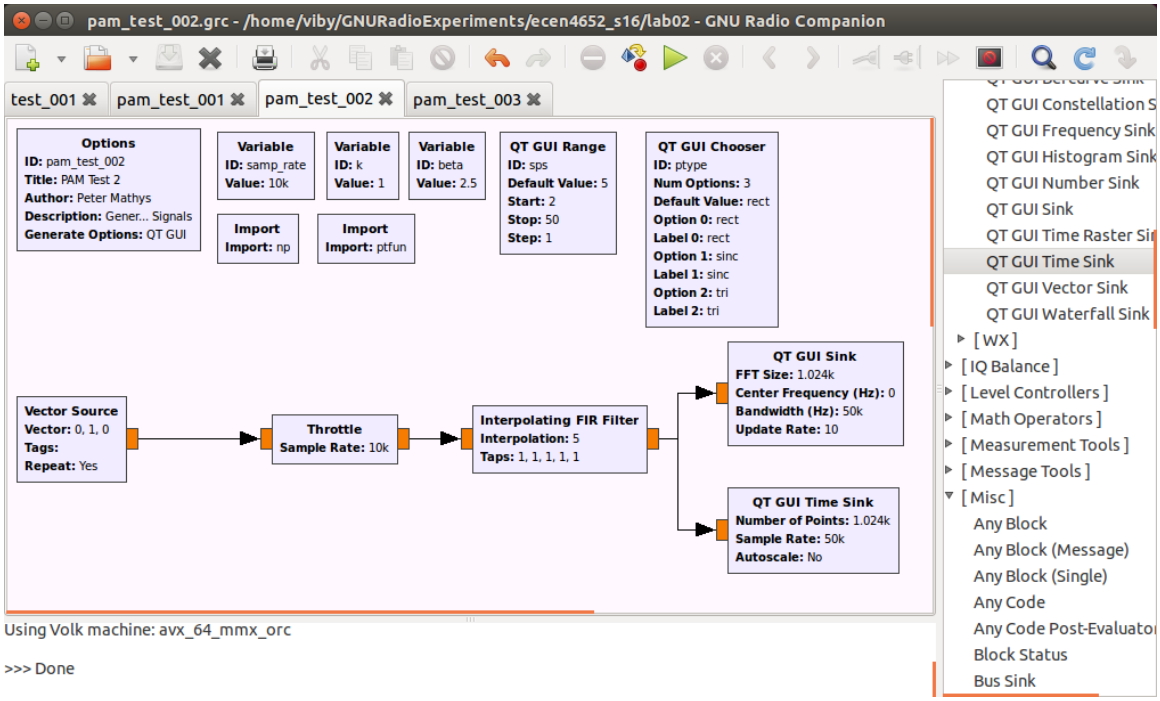
One of the goals of this experiment is to see how the spectrum of a pulse is affected if a more gradual transition between amplitudes of 0 and 1 is used than for a rectangular pulse. Characterize the differences between the rectangular and the sinusoidal pulse of width `tau` for different values of `tau` in the frequency domain.

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

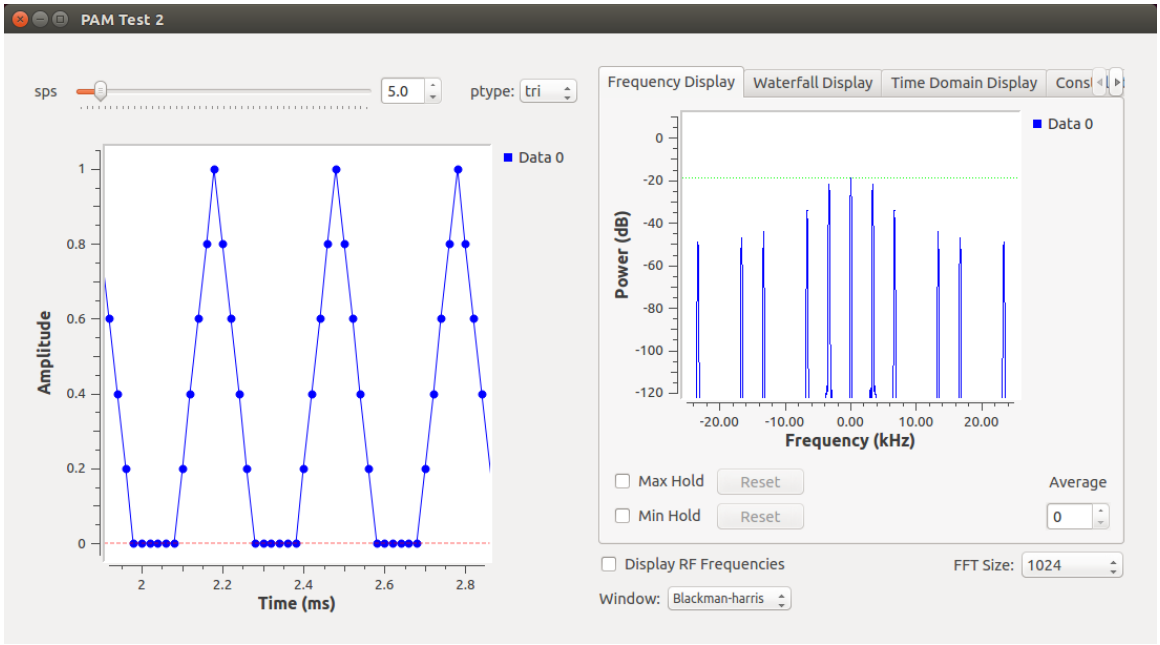


Use a sampling rate of 32 kHz and show how to generate such a signal with variable frequency f_1 in the range 0 to 10000 Hz in the GRC. Hints: Start out with a complex-valued sinusoid and ask yourself what the best way is to obtain a phase change analytically that works for all frequencies. To convert 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.

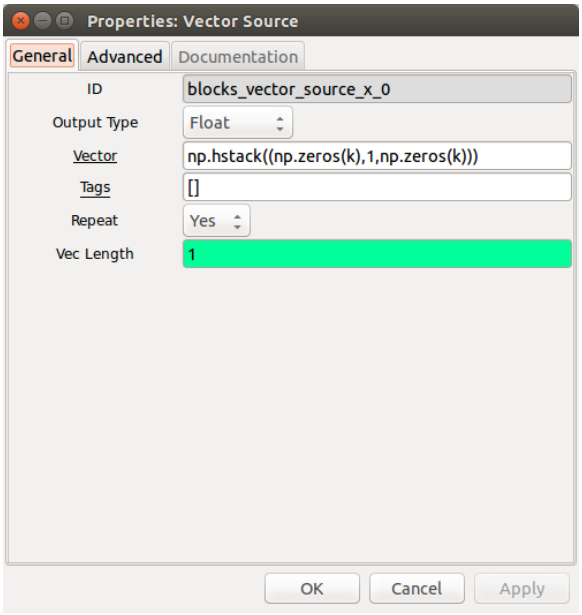
(c) The figure below shows a flowgraph that can be used to generate PAM pulses of type `'rect'`, `'sinc'`, and `'tri'` with a given number of samples per symbol (`sps`).



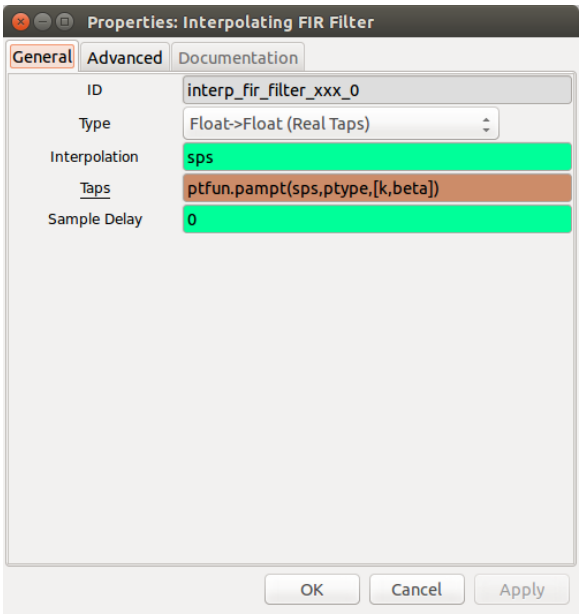
When the flowgaph is executed for pulse type `'tri'` and sps=5, the following graphs are obtained.



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.



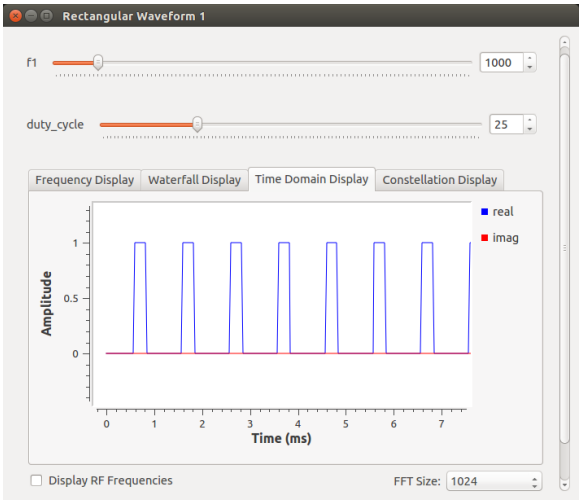
The header of the Python function `ptfun` looks as follows.

```
# 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 5000 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 5000 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.