DSPIRA GnuRadio Lessons

Lesson 1: Building a Waveform Generator

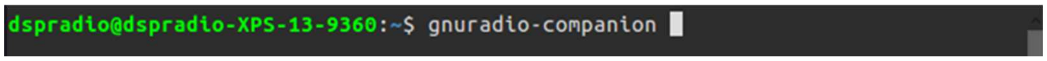
GNURadio is very popular and robust software defined radio package. It is open source and is relatively easy to use. All “coding” is done using flowgraphs comprised of interconnected **Digital Signal Processing (DSP)** blocks. Most commonly used blocks come predefined as part of the software package. One can program one's own blocks as well.

Let’s get Familiar – Building a Waveform Generator

For our first project we will build a program that generates a real-valued cosine signal with a frequency of 400 Hz, and outputs it to a speaker. We will also display the signal in a time domain graph and in a frequency domain graph. Then we will modify it so that we can adjust the frequency in real time.

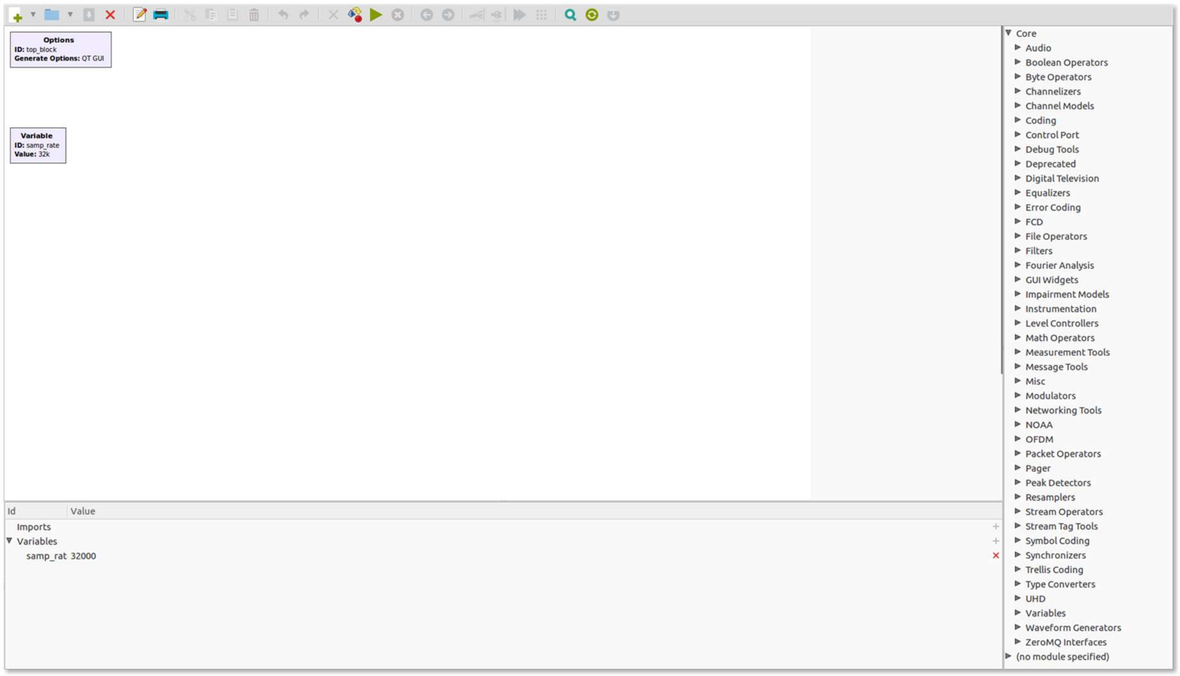
Step 1. Open GNURadio

- To open GNURadio, open a terminal window. Type "gnuradio-companion" and hit Enter.



```
dspradio@dspradio-XPS-13-9360:~$ gnuradio-companion
```

This opens GNU Radio Companion (GRC):

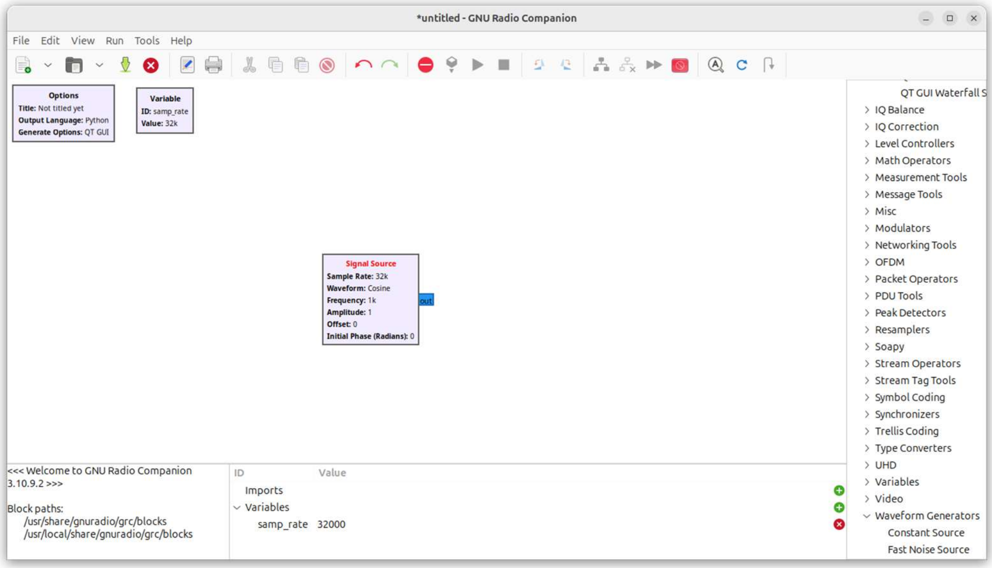


Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

```
Id          Value
 Imports
▼ Variables
   samp_rat 32000
```

- The canvas window is mostly empty. On the right side of the canvas is a list of the block categories that are available. Click on a triangle next to a category to see what blocks are available in that category. Look under the ***waveform generator*** category and look for the ***Signal Source*** block. Alternatively, we can click on the magnifying/looking glass to the top right and search for the block we need.
- Add the ***Signal Source*** block to the canvas by dragging it into the canvas window (or double-click):



Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

```
<<< Welcome to GNU Radio Companion
3.10.9.2 >>>

Block paths:
    /usr/share/gnuradio/grc/blocks
    /usr/local/share/gnuradio/grc/blocks
```

ID	Value
Imports	
Variables	
samp_rate	32000

Helpful Information About GNURadio

Block Port Colors

Notice that the ***Signal Source*** block has a blue port 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. The most common data types that we will use are:

- Orange is for "Float" data types, which are real-valued 32-bit floating point data samples.
- Blue is for "Complex" data types, which are complex-valued 32-bit floating point data samples. [Electrical engineering and digital signal processing often use complex functions and data.]

Connecting Blocks

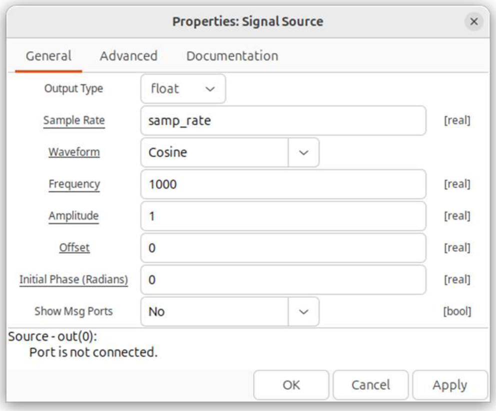
To connect two blocks, click on the output port of the first block and then click on the input port of the second block. An arrow will appear connecting the two ports. For the flowgraph to work both ports must use the same **data type** (i.e., both ports must be of the same color). If they are of different types, then the arrow of the connection will be red instead of black, indicating a problem.

NOTE: 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.

Editing a Block

To edit a block, double-click on it. A window will open for editing the block.

Back to building a waveform generator:



Output Type	float	
Sample Rate	samp_rate	[real]
Waveform	Cosine	
Frequency	1000	[real]
Amplitude	1	[real]
Offset	0	[real]
Initial Phase (Radians)	0	[real]
Show Msg Ports	No	[bool]

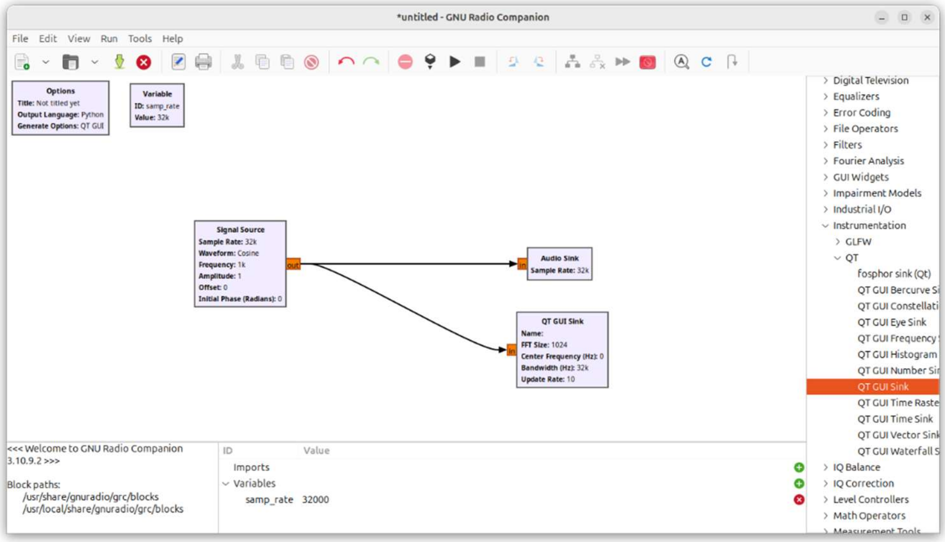
Source - out(0):
Port is not connected.

Step 2. Change the "Output Type" in the ***Signal Source*** block to "float" and hit Ok.

The output terminal on this block should now be orange.

Step 3. Let's add a speaker and graphical display to the signal output:

- **Change the Signal Source frequency:** Double-click the ***Signal Source*** block. Then type in "400" for the frequency value.
- **Add a speaker to the output:** Choose the ***Audio Sink*** block under "Audio". Connect this to the output of the ***Signal Source*** block.
- **Add a graphic output:** Choose "QT" under "Instrumentation" (or just simply search for "QT GUI Sink") and add a ***QT GUI Sink*** to the canvas. This block will allow you to see the waveform at the input in the frequency as well as in the time domain. Connect this to the output of the ***Signal Source*** block. Be sure that all of the data types are properly set.
- The program now should look like the following:



*untitled - GNU Radio Companion

Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Audio Sink
Sample Rate: 32k

QT GUI Sink
Name:
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 32k
Update Rate: 10

```
<<< Welcome to GNU Radio Companion
3.10.9.2 >>>

Block paths:
    /usr/share/gnuradio/grc/blocks
    /usr/local/share/gnuradio/grc/blocks
```

ID	Value
Imports	
Variables	
samp_rate	32000

- **Run the program:** Now you can run the flow graph by clicking on the ▶ triangle above the canvas or by clicking "Run" and executing on the menu bar.

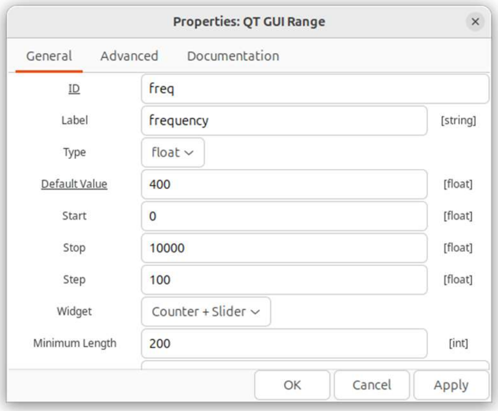
  When prompted to save the file, create a new folder for your work.

  NOTE: The WARNING message that appears can be ignored. It will appear only the first time you run GNURadio.
- To view the graphs, you can choose between the "Frequency Display" and the "Time Domain Display" tabs. Study these graphs, and examine the other graphical displays as well.

Let's Make Some Modifications

Let's add a few features to the program. We want to be able to change the frequency and amplitude of the source signal while the program is running. We will us the ***QT GUI Range*** block to define a variable that can be changed while the program is running.

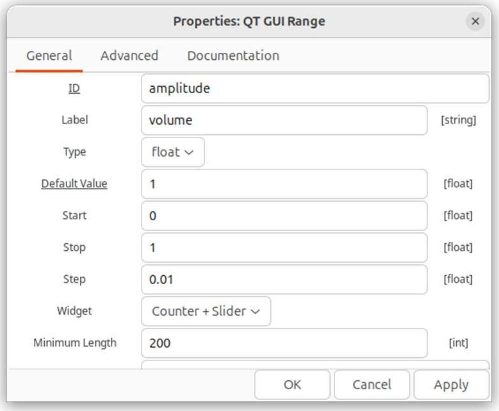
- **Add a frequency variable:** Search for and double-click on ***QT GUI Range*** to add it to the canvas. This block type allows you to define and control a variable. Change the *ID* to "freq" and the *Label* to "frequency". Set the default to "400" and the range in the Start and Stop boxes as shown below.
- **Add an amplitude variable:** Add a ***QT GUI Range*** block. Change the *ID* to "amplitude" and the *Label* to "volume". Set the default to "1", and the range in the Start and Stop boxes as shown below.


Properties: QT GUI Range

General | Advanced | Documentation

ID	freq	
Label	frequency	[string]
Type	float	
Default Value	400	[float]
Start	0	[float]
Stop	10000	[float]
Step	100	[float]
Widget	Counter + Slider	
Minimum Length	200	[int]

OK | Cancel | Apply

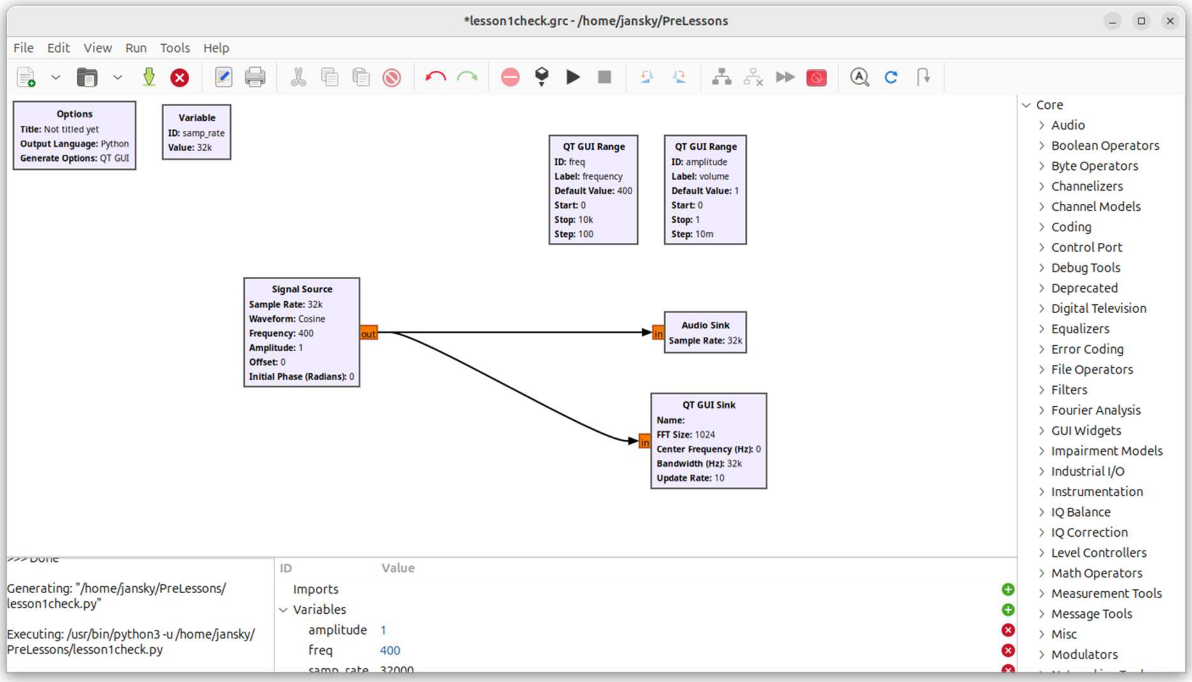

Properties: QT GUI Range

General | Advanced | Documentation

ID	amplitude	
Label	volume	[string]
Type	float	
Default Value	1	[float]
Start	0	[float]
Stop	1	[float]
Step	0.01	[float]
Widget	Counter + Slider	
Minimum Length	200	[int]

OK | Cancel | Apply

- Make changes to the ***Signal Source*** block to match these variables: change the *Frequency* value to "freq" and *Amplitude* to "amplitude."
- The program now should look like the following:


*lesson1check.grc - /home/jansky/PreLessons

Options — Title: Not titled yet; Output Language: Python; Generate Options: QT GUI

Variable — ID: samp_rate; Value: 32k

QT GUI Range — ID: freq; Label: frequency; Default Value: 400; Start: 0; Stop: 10k; Step: 100

QT GUI Range — ID: amplitude; Label: volume; Default Value: 1; Start: 0; Stop: 1; Step: 10m

Signal Source — Sample Rate: 32k; Waveform: Cosine; Frequency: 400; Amplitude: 1; Offset: 0; Initial Phase (Radians): 0

Audio Sink — Sample Rate: 32k

QT GUI Sink — Name: ; FFT Size: 1024; Center Frequency (Hz): 0; Bandwidth (Hz): 32k; Update Rate: 10

```
>>> Done

Generating: "/home/jansky/PreLessons/
lesson1check.py"

Executing: /usr/bin/python3 -u /home/jansky/
PreLessons/lesson1check.py
```

ID	Value
Imports	
Variables	
amplitude	1
freq	400
samp_rate	32000

- **Re-run the program:** Adjust the frequency and amplitude sliders, and observe changes in the Time domain graph and the Frequency domain graph.