General Introduction

Software-defined radio (SDR) along with its hardware can be used to illustrate examples of the physical layer implementation in the OSI model.

SDR is a radio communication system in which components that have been traditionally implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are instead implemented by means of software on a personal computer or embedded system.

SDRs turn a standard PC into a next-generation wireless prototyping tool. However, RF (Radio Frequency) front-end, ADC (Analog to Digital Converters) and DAC (Digital to Analog Converters) are still needed. As it is indeed the always the case for the physical layer, it is not possible with SDR to totally bypath the hardware.

Figure 1 and Figure 2 show a traditional radio receiver and the equivalent receiver implemented on SDR, respectively. SDR uses RF to IF, ADC and DAC and an FPGA unit. Our aim is to program the FPGA unit to perform what a typical radio does.

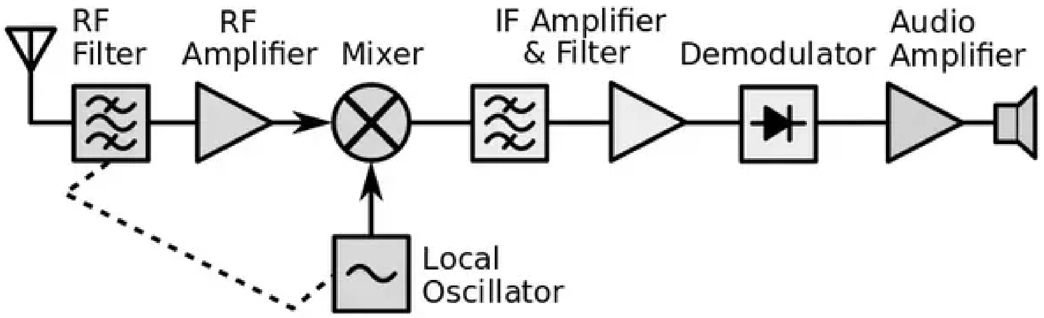


Figure 1. Traditional hardware of Radio receiver

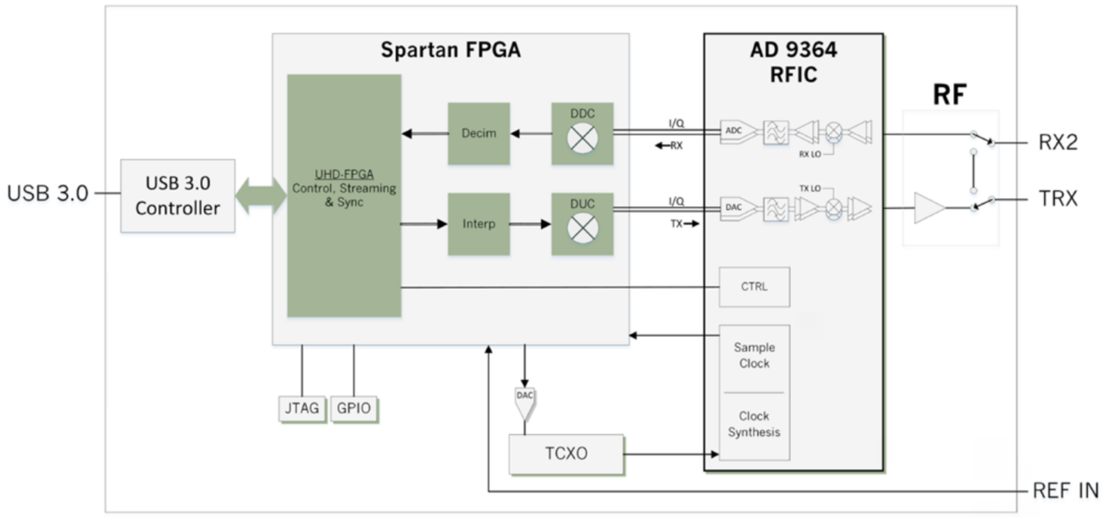


Figure 2. Schematic diagram of the SDR Module kit

Some features of SDR:

A software-defined radio can be flexible enough to avoid the "limited spectrum" assumptions of designers of previous kinds of radios, in one or more ways including:[2]

- Spread spectrum and ultrawideband techniques allow several transmitters to transmit in the same place on the same frequency with very little interference, typically combined with one or more error detection and correction techniques to fix all the errors caused by that interference.
- Software defined antennas adaptively "lock onto" a directional signal, so that receivers can better reject interference from other directions, allowing it to detect fainter transmissions.
- Cognitive radio techniques: each radio measures the spectrum in use and communicates that information to other cooperating radios, so that transmitters can avoid mutual interference by selecting unused frequencies. Alternatively, each radio connects to a geolocation database to obtain information about the spectrum occupancy in its location and, flexibly, adjusts its operating frequency and/or transmit power not to cause interference to other wireless services.
- Dynamic transmitter power adjustment, based on information communicated from the receivers, lowering transmit power to the minimum necessary, reducing the near-far problem and reducing interference to others, and extending battery life in portable equipment.
- Wireless mesh network where every added radio increases total capacity and reduces the power required at any one node.[3] Each node only transmits loudly enough for the message to hop to the nearest node in that direction, reducing near-far problem and reducing interference to others.
- You can present applications with real-world signals such as multiple input, multiple output (MIMO) and LTE/WiFi testbed.

The SDR Kit that We Use:

Link to:

https://www.ettus.com/product/details/USRP-B200mini-i

check the features of this kit. It can simultaneously send and receive data.

Important Hint: This kit only works with WX GUI type modules.

Kit contents:

- USRP B200 Mini

- USB 3.0 cable

- Antenna

Cautions:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or a 50 Ω load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.

- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.
- Never apply more than 0 dBm of power into any RF input.
- Always use at least 30 dB attenuation if operating in loopback configuration.

The software we use is GNU Radio:

https://kb.ettus.com/B200/B210/B200mini/B205mini_Getting_Started_Guides

How to install:

Method1:

https://kb.ettus.com/Live_SDR_Environment

Method2:

https://kb.ettus.com/Building_and_Installing_the_USRP_Open_Source_Toolchain_(UHD_and_GNU_Radio)_on_Windows

Use method 2 to create a bootable USB memory.

Getting Started with It:

Introduction to GNU Radio and Software Radio

GNU Radio is a framework that enables users to design, simulate, and deploy highly capable real-world radio systems. It is a highly modular, "flowgraph"-oriented framework that comes with a comprehensive library of processing blocks that can be readily combined to make complex signal processing applications.

GNU Radio has been used for a huge array of real-world radio applications, including audio processing, mobile communications, tracking satellites, radar systems, GSM networks, Digital Radio Mondiale, and much more - all in computer software.

It is, by itself, not a solution to talk to any specific hardware. Nor does it provide out-of-the-box applications for specific radio communications standards (e.g., 802.11, ZigBee, LTE, etc.,), but it can be (and has been) used to develop implementations of basically any band-limited communication standard.

Why would I want GNU Radio?

Formerly, when developing radio communication devices, the engineer had to develop a specific circuit for detection of a specific signal class, design a specific integrated circuit that would be able to decode or encode that particular transmission and debug these using costly equipment.

Software-Defined Radio (SDR) takes the analog signal processing and moves it, as far as physically and economically feasible, to processing the radio signal on a computer using algorithms in software.

You can, of course, use your computer-connected radio device in a program you write from scratch, concatenating algorithms as you need them and moving data in and out yourself. But this quickly becomes cumbersome: Why are you re-implementing a standard filter? Why do you have to care how data moves between different processing blocks? Wouldn't it be better to use highly optimized and peer-reviewed implementations rather than writing things yourself? And how do you get your program to scale well on a multi-core architectures but also run well on an embedded device consuming but a few watts of power? Do you really want to write all the GUIs yourself?

Enter GNU Radio: A framework dedicated to writing signal processing applications for commodity computers. GNU Radio wraps functionality in easy-to-use reusable blocks, offers excellent scalability, provides an extensive library of standard algorithms, and is heavily optimized for a large variety of common platforms. It also comes with a large set of examples to get you started.

Digital Signal Processing

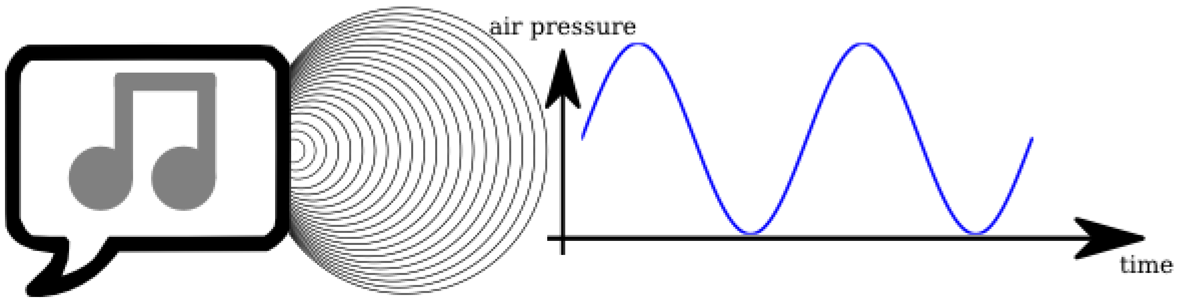
As a software framework, GNU Radio works on digitized signals to generate communication functionality using general-purpose computers.

A little signal theory

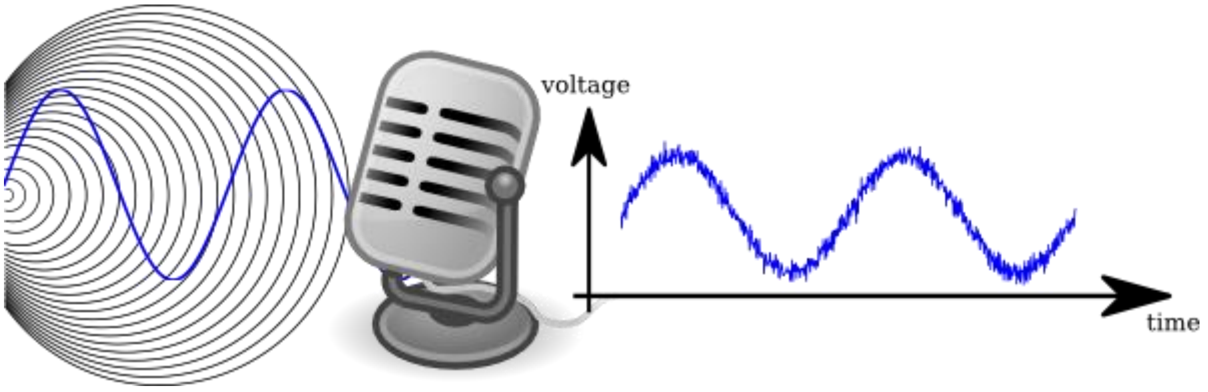
Doing signal processing in software requires the signal to be digital. But what is a digital signal?

To understand better, let's look at a common "signal" scenario: Recording voice for transmission using a cellphone.

A personal physically speaking creates a sound 'signal' - the signal, in this case, is comprised of waves of varying air pressure being generated by the vocal chords of a human. A time-varying physical quantity, like the air pressure, is what is defined as a signal.



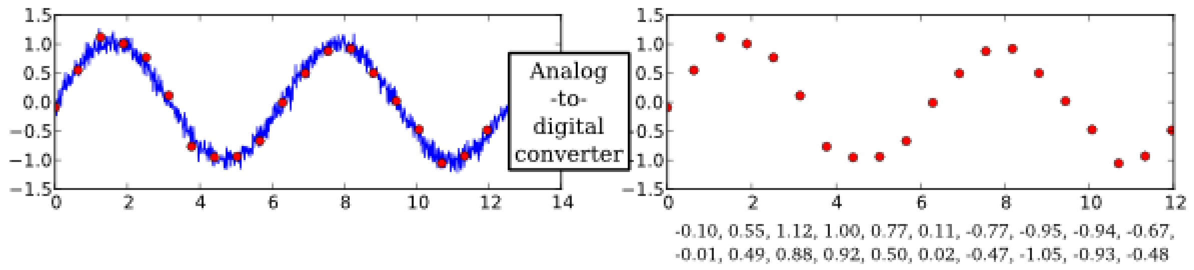
When the waves reach the microphone, it converts the varying pressure into an electrical signal, a variable voltage:



Now that the signal is electrical, we can work with it. The audio signal, at this point, is analog - a computer can't yet deal with it; for computational processing, a signal has to be digital, which means two things:

It can only be one of a limited number of values.

It only exists for a non-infinite amount of time.



This digital signal can thus be represented by a sequence of numbers, called samples. A fixed time interval between samples leads to a signal sampling rate.

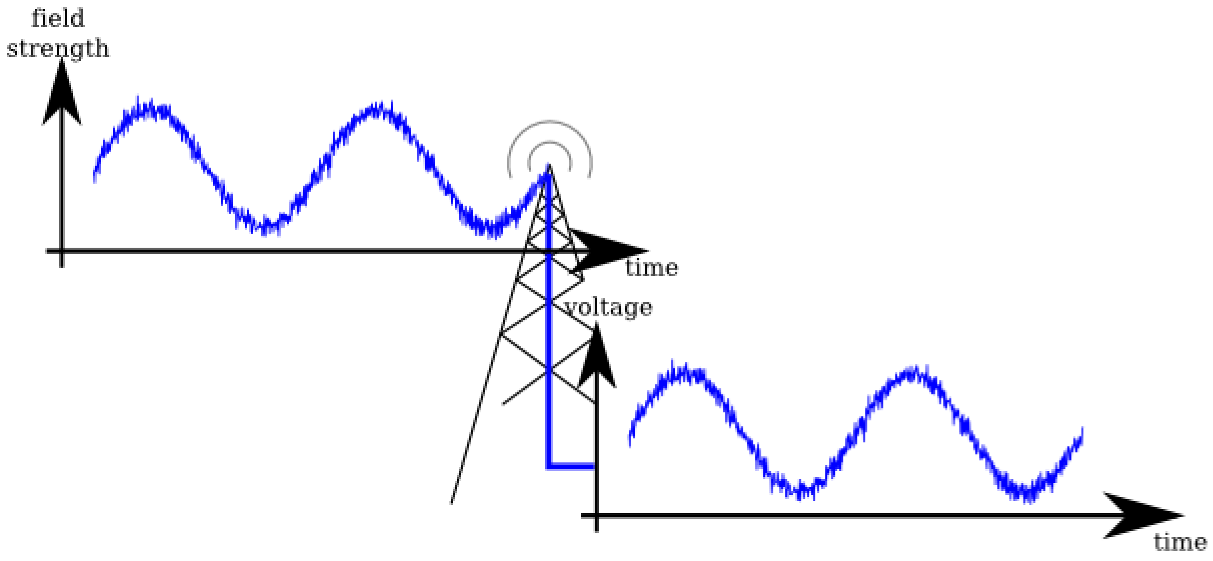
The process of taking a physical quantity (voltage) and converting it to digital samples is done by an Analog-to-Digital Converter (ADC). The complementary device, a Digital-to-Analog Converter (DAC), takes numbers from a digital computer and converts them to an analog signal.

Now that we have a sequence of numbers, our computer can do anything with it. It might, for example, apply digital filters, compress it, recognize speech, or transmit the signal using a digital link.

Applying Digital Signal Processing to Radio Transmissions

The same principles as for sounds can be applied to radio waves:

A signal, here electromagnetic waves, can be converted into a varying voltage using an antenna.

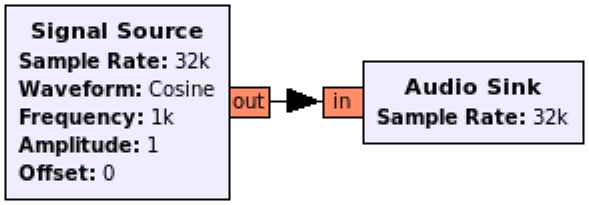


This electrical signal is then on a 'carrier frequency', which is usually several Mega- or even Gigahertz.

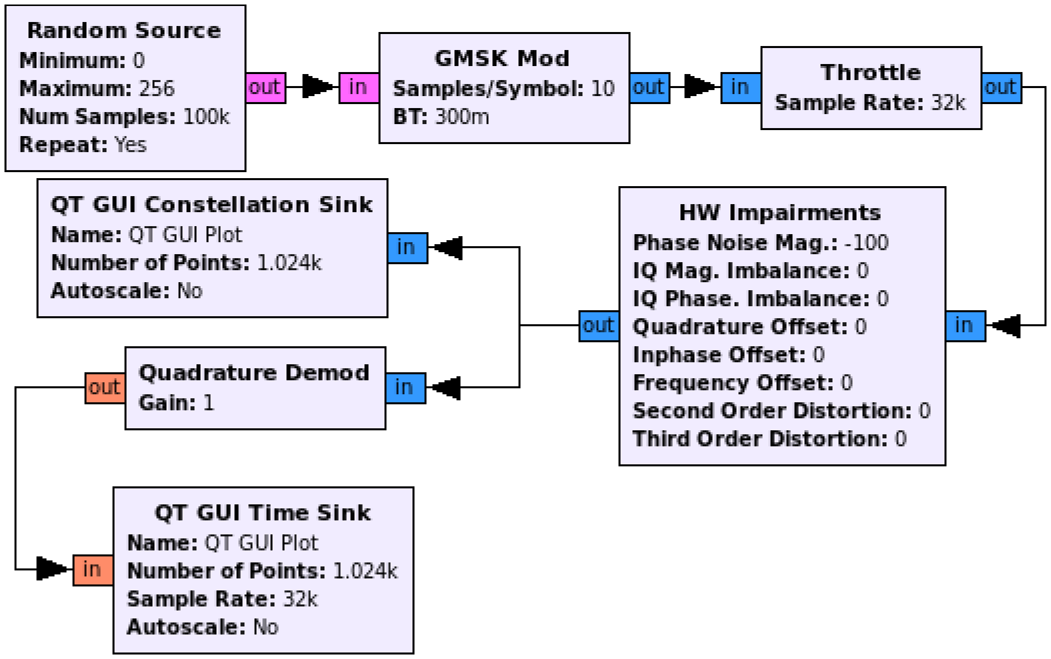
Using different types of receivers (e.g. Superheterodyne Receiver, Direct Conversion, Low Intermediate Frequency Receivers), which can be acquired commercially as dedicated software radio peripherals, are already available to users (e.g. amateur radio receivers connected to sound cards) or can be obtained when re-purposing cheaply available consumer digital TV receivers (the notorious RTL-SDR project).

A modular, flowgraph based Approach to Digital Signal Processing

To process digital signals, it is straight-forward to think of the individual processing stages (filtering, correction, analysis, detection...) as processing blocks, which can be connected using simple flow-indicating arrows:



When building a signal processing application, one will build up a complete graph of blocks. Such a graph is called flowgraph in GNU Radio.



GNU Radio is a framework to develop these processing blocks and create flowgraphs, which comprise radio processing applications.

As a GNU Radio user, you can combine existing blocks into a high-level flowgraph that does something as complex as receiving digitally modulated signals and GNU Radio will automatically move the signal data between these and cause processing of the data when it is ready for processing.

GNU Radio comes with a large set of existing blocks. Just to give you but a small excerpt of what's available in a standard installation, here's some of the most popular block categories and a few of their members:

Waveform Generators

- Constant Source
- Noise Source

- Signal Source (e.g. Sine, Square, Saw Tooth)

Modulators

- AM Demod
- Continuous Phase Modulation
- PSK Mod / Demod
- DPSK Mod / Demod
- GMSK Mod / Demod
- QAM Mod / Demod
- WBFM Receive
- NBFM Receive

Instrumentation (i.e., GUIs)

- Constellation Sink
- Frequency Sink
- Histogram Sink
- Number Sink
- Time Raster Sink
- Time Sink
- Waterfall Sink

Math Operators

- Abs
- Add
- Complex Conjugate
- Divide
- Integrate
- Log10
- Multiply
- RMS
- Subtract

Channel Models

- Channel Model
- Fading Model
- Dynamic Channel Model
- Frequency Selective Fading Model

Filters

- Band Pass / Reject Filter
- Low / High Pass Filter
- IIR Filter
- Generic Filterbank
- Hilbert

- Decimating FIR Filter
- Root Raised Cosine Filter
- FFT Filter

Fourier Analysis

- FFT
- Log Power FFT
- Goertzel (Resamplers)
- Fractional Resampler
- Polyphase Arbitrary Resampler
- Rational Resampler (Synchronizers)
- Clock Recovery MM
- Correlate and Sync
- Costas Loop
- FLL Band-Edge
- PLL Freq Det
- PN Correlator
- Polyphase Clock Sync

Using these blocks, many standard tasks, like normalizing signals, synchronization, measurements, and visualization can be done by just connecting the appropriate block to your signal processing flow graph.

Also, you can write your own blocks, that either combine existing blocks with some intelligence to provide new functionality together with some logic, or you can develop your own block that operates on the input data and outputs data.

Thus, GNU Radio is mainly a framework for the development of signal processing blocks and their interaction. It comes with an extensive standard library of blocks, and there are a lot of systems available that a developer might build upon. However, GNU Radio itself is not a software that is ready to do something specific -- it's the user's job to build something useful out of it, though it already comes with a lot of useful working examples. Think of it as a set of building blocks.

Tutorial: GNU Radio Companion

Objectives

- Create flowgraphs using the standard block libraries
- Learn how to debug flowgraphs with the instrumentation blocks
- Understand how sampling and throttle works in GNU Radio
- Learn how to use the documentation to figure out block's functionality

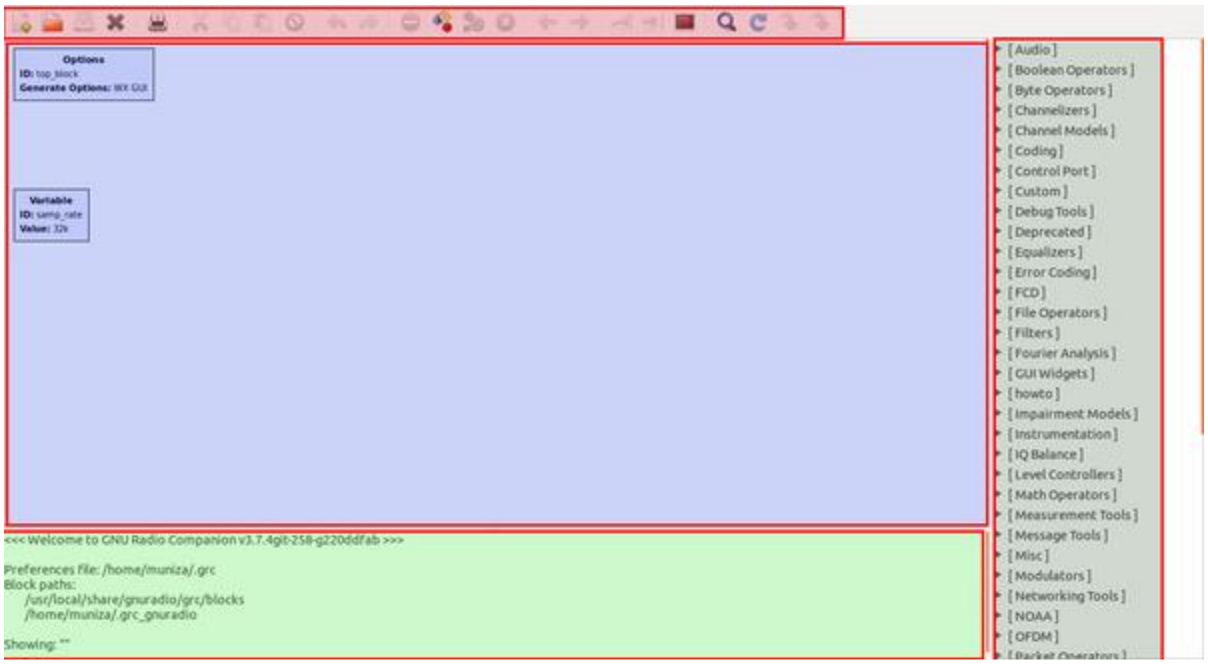
Setting up the Tutorials

Before we can begin, we need to mention that the solutions (including the images, grc files, and module files) are available on gr-tutorial github if you wish to have them. We will be referencing the files but will for the most part be making our own files so that we can get practice and build intuition. Thus, we recommend not downloading and installing them until you finish these tutorials or are stuck on a step. For instructions on installing these solutions, see Installing the Tutorials Modules.

Getting to Know the GRC

We have seen in Tutorial 1 that GNU Radio is a collection of tools that can be used to develop radio systems in software as opposed to completely in hardware. In this tutorial, we start off simple and explore how to use the GNU Radio Companion (GRC), GNU Radio's graphical tool, to create different tones. We should keep in the back of our mind that GRC was created to simplify the use of GNU Radio by allowing us to create python files graphically as opposed to creating them in code alone (we will discuss this more later).

The first thing to cover is the interface. There are four parts: Library, Toolbar, Terminal, and Workspace.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 32k

```
<<< Welcome to GNU Radio Companion v3.7.4git-258-g220ddfab >>>

Preferences file: /home/muniza/.grc
Block paths:
    /usr/local/share/gnuradio/grc/blocks
    /home/muniza/.grc_gnuradio

Showing: ""
```

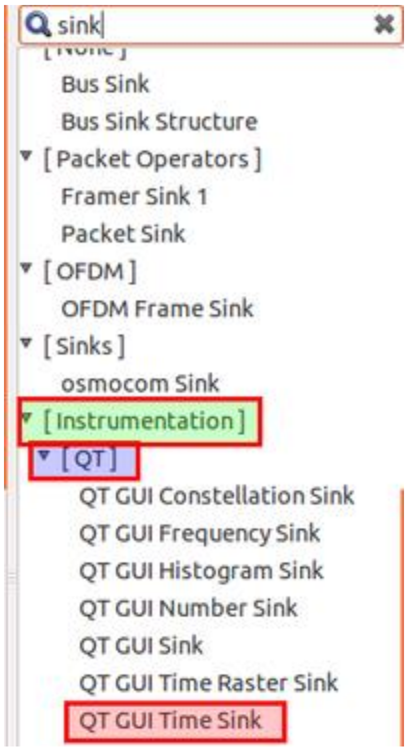
The tutorial is meant to be hands on, so please take a few breaks from reading here and there to experiment. We will reiterate that these tutorials as simply meant as guides and that the best way to learn something is to try it out: come up with a question, think of a possible solution, and try it out. Let us begin by starting up the GRC! This is usually done by opening up a terminal window (ctrl+alt+t in Ubuntu) and typing:

```
$ gnuradio-companion
```

If you are unable to open GRC then you need to go back to Installing GR and troubleshoot the installation.

Searching for Blocks

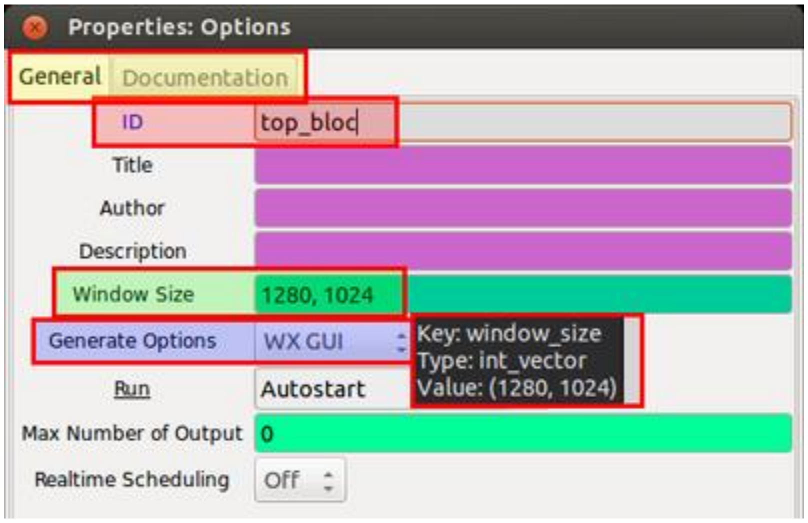
The Library contains the different blocks installed in the GRC block paths. Here we find blocks that are preinstalled in GNU Radio and blocks that are installed on the system. At first it seems daunting to look for blocks. For instance, if we want to generate a waveform, what category should we look in? We see there is a Waveform Generators category, okay not bad. But what if we wanted to find some way to display our data but aren't sure of the best way to display it yet? We know from tutorial 1 that this is known as a sink; however, looking through the list there is no Sinks category. The solution is to use the Search feature by either clicking the magnifying glass icon or typing Ctrl+f and then start typing a keyword to identify the block. We see a white box appear at the top of the Library with a cursor. If we type "sink", we can find all the blocks that contain the words "sink" and the categories we will find each block in.



For now, let's add the block called QT GUI Time Sink by clicking on its name and dragging it to the Workspace or double-clicking on its name for it to be placed automatically in the Workspace.

Modifying Block Properties

The workspace (main area of the screen) contains all of our blocks that make up our flowgraph, and inside each block we can see all the different block parameters. There is, however, one special block that each new flowgraph starts with and is required to have, called the Options Block. Let us double-click on the Options Block to examine its properties. We see a window as below:

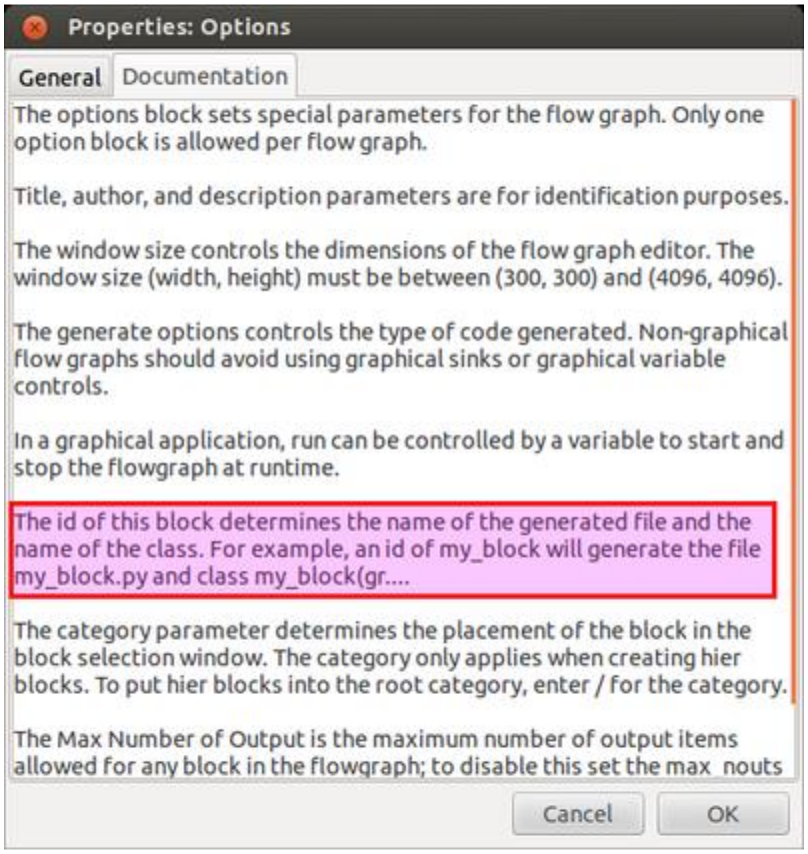

Properties: Options

General | Documentation

ID	top_bloc
Title	
Author	
Description	
Window Size	1280, 1024
Generate Options	WX GUI
Run	Autostart
Max Number of Output	0
Realtime Scheduling	Off

Key: window_size
Type: int_vector
Value: (1280, 1024)

These block properties can be changed from the defaults to accomplish different tasks. Let's remove part of the current name and notice the ID turns blue. This color means that the information has been edited, but has not been saved. Let us change the parameter Window Size to "300,300" and click OK. Yikes! Almost all the workspace got cutoff! Let's do a ctrl+z to go back to our default size. If we go back to the Options Block properties, we can see that there are different tabs and one is titled documentation.


Properties: Options

General | Documentation

The options block sets special parameters for the flow graph. Only one option block is allowed per flow graph.

Title, author, and description parameters are for identification purposes.

The window size controls the dimensions of the flow graph editor. The window size (width, height) must be between (300, 300) and (4096, 4096).

The generate options controls the type of code generated. Non-graphical flow graphs should avoid using graphical sinks or graphical variable controls.

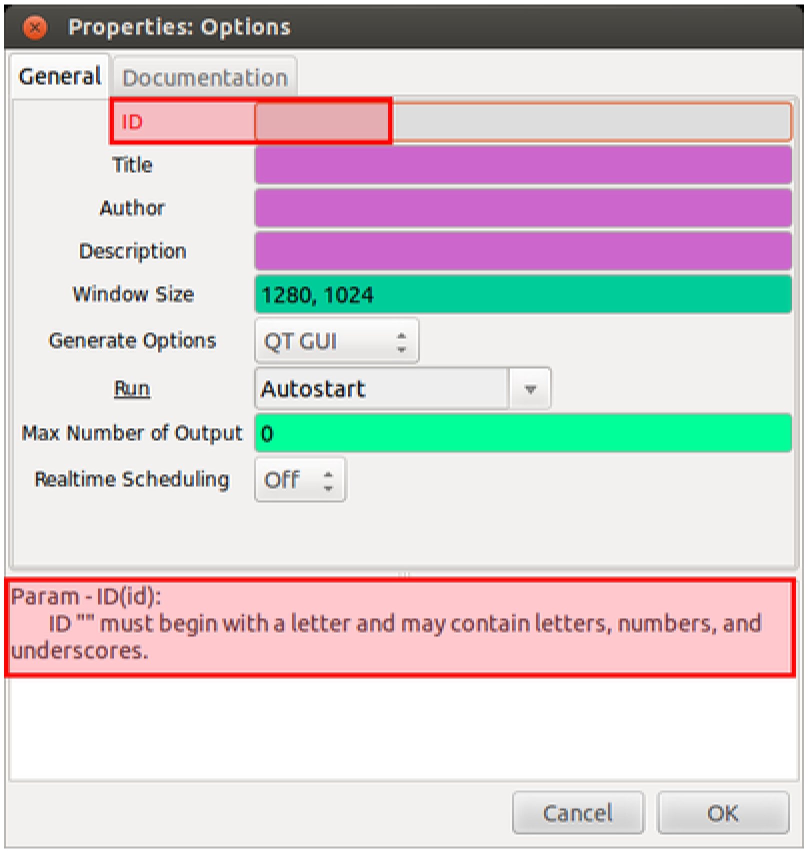
In a graphical application, run can be controlled by a variable to start and stop the flowgraph at runtime.

The id of this block determines the name of the generated file and the name of the class. For example, an id of my_block will generate the file my_block.py and class my_block(gr....

The category parameter determines the placement of the block in the block selection window. The category only applies when creating hier blocks. To put hier blocks into the root category, enter / for the category.

The Max Number of Output is the maximum number of output items allowed for any block in the flowgraph; to disable this set the max_nouts

If we read a couple lines, we can see that the property ID is actually used for the name of the python file the flowgraph generates.



ID

Title

Author

Description

Window Size 1280, 1024

Generate Options QT GUI

Run Autostart

Max Number of Output 0

Realtime Scheduling Off

Param - ID(id):
ID "" must begin with a letter and may contain letters, numbers, and underscores.

So now let's remove the entire ID string. Notice now that at the bottom appears an error message. Also notice that the parameter ID is now red to show us exactly where the error occured.

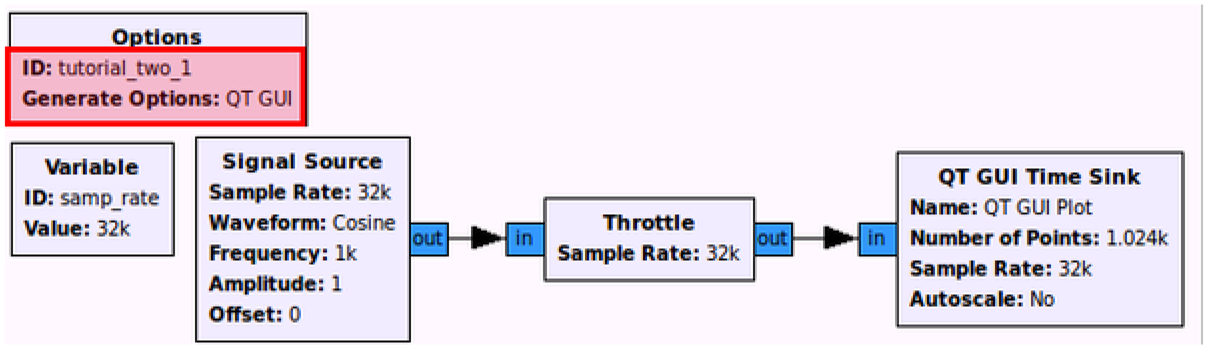
To keep things organized, let us change the ID to "tutorial_two_1". Let us also make sure that the property Generate Options is set to "QT GUI" since we are using a QT GUI sink and not a WX GUI sink. Newer versions of GNU Radio default to using QT GUI. The ID field allows us to more easily manage our file space. While we save the GRC flowgraph as a <filename>.grc, generating and executing this flowgraph produces another output. GRC is a graphical interface that sits on top of the normal GNU Radio programming environment that is in Python. GRC translates the flowgraph we create in the GUI canvas here into a Python script, so when we execute a flowgraph, we are really running a Python program. The ID is used to name that Python file, saved into the same directory as the .grc file. By default, the ID is top_block and so it creates a file called top_block.py. Changing the ID allows us to change the saved file name for better file management.

Another result of this GRC-Python connection is that GRC is actually all Python. In fact, all entry boxes in block properties or variables that we use are interpreted as Python. That means that we can set properties using Python calls, such as calling a numpy or other GNU Radio functions. A common use of this is to call into the filter.firdes filter design tool from GNU Radio to build our filter taps.

Another key to notice is the different colors present in the fields we can enter information. These actually correspond to different data types which we will cover later in this tutorial.

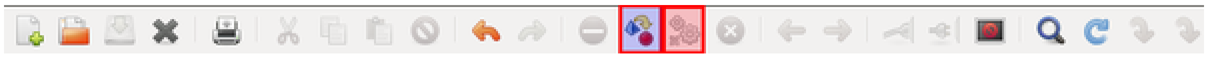
My First Flowgraph

Now that we have a better understanding of how to find blocks, how to add them to the workspace, and how to edit the block properties, let's proceed to construct the following flowgraph of a Signal Sourcebeing sent to a Throttle Block and then to our Time Sink by clicking on the data type colored ports/tabs one after the other to make connections:



With a usable flowgraph we can now proceed to discuss the Toolbar.

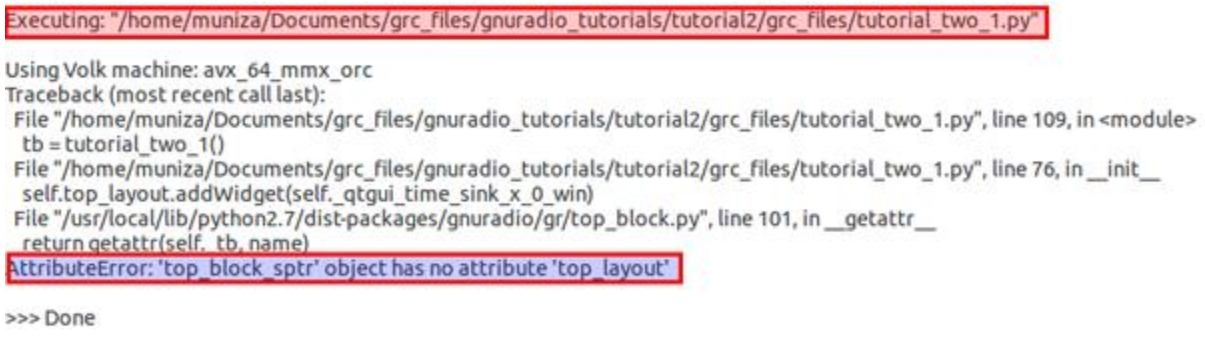
A note on the throttle block: What exactly this does is explained in greater detail later on in this tutorial. For now, it will suffice to know that this block throttles the flow graph to make sure it doesn't consume 100% of your CPU cycles and make your computer unresponsive.



This section of the interface contains commands present in most software such as new, open, save, copy, paste. Let's begin by saving our work so far and titling our flow graph tutorial_two. Important tools here are Generate flowgraph, Execute flowgraph, and Kill flowgraph all accessible through F5, F6, and F7 respectively. A good reference is available in Help->Types that shows the color mapping of types which we will look into later.

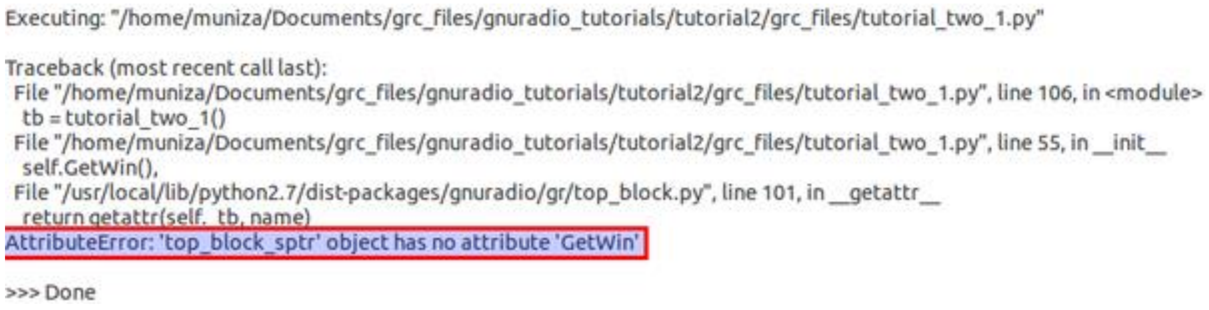
A Note on Generate Options

Let us click the Generate button and turn our eyes to the Terminal at the bottom of the window. We should see it generated a Python file with the same name as the ID from our Options Block. The terminal displays important messages such as errors and warnings. Two common errors are when we mismatch the generate options with the graphical tools we are using. For instance, if we were to use the WX GUI as our generate options but have a QT GUI graphic then we would get in the terminal:



```
Executing: "/home/muniza/Documents/grc_files/gnuradio_tutorials/tutorial2/grc_files/tutorial_two_1.py"

Using Volk machine: avx_64_mmx_orc
Traceback (most recent call last):
 File "/home/muniza/Documents/grc_files/gnuradio_tutorials/tutorial2/grc_files/tutorial_two_1.py", line 109, in <module>
  tb = tutorial_two_1()
 File "/home/muniza/Documents/grc_files/gnuradio_tutorials/tutorial2/grc_files/tutorial_two_1.py", line 76, in __init__
  self.top_layout.addWidget(self._qtgui_time_sink_x_0_win)
 File "/usr/local/lib/python2.7/dist-packages/gnuradio/gr/top_block.py", line 101, in __getattr__
  return getattr(self._tb, name)
AttributeError: 'top_block_sptr' object has no attribute 'top_layout'

>>> Done
```

And if we were to use the QT GUI generate options with a WX GUI graphic we would get in the terminal:

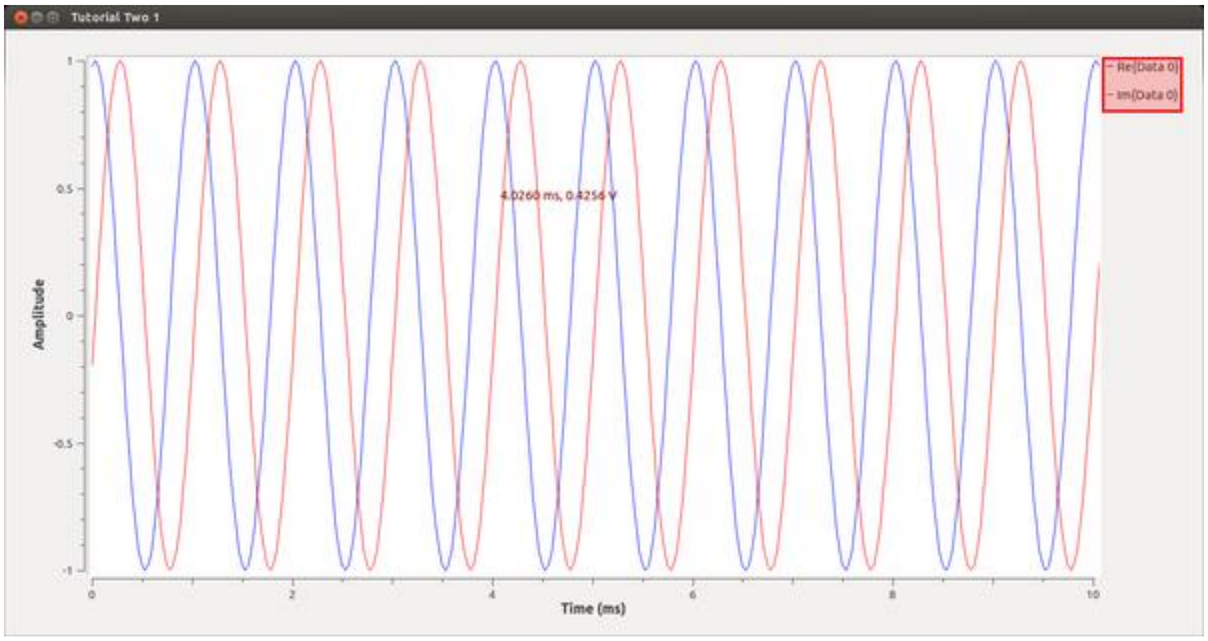


```
Executing: "/home/muniza/Documents/grc_files/gnuradio_tutorials/tutorial2/grc_files/tutorial_two_1.py"

Traceback (most recent call last):
 File "/home/muniza/Documents/grc_files/gnuradio_tutorials/tutorial2/grc_files/tutorial_two_1.py", line 106, in <module>
  tb = tutorial_two_1()
 File "/home/muniza/Documents/grc_files/gnuradio_tutorials/tutorial2/grc_files/tutorial_two_1.py", line 55, in __init__
  self.GetWin(),
 File "/usr/local/lib/python2.7/dist-packages/gnuradio/gr/top_block.py", line 101, in __getattr__
  return getattr(self._tb, name)
AttributeError: 'top_block_sptr' object has no attribute 'GetWin'

>>> Done
```

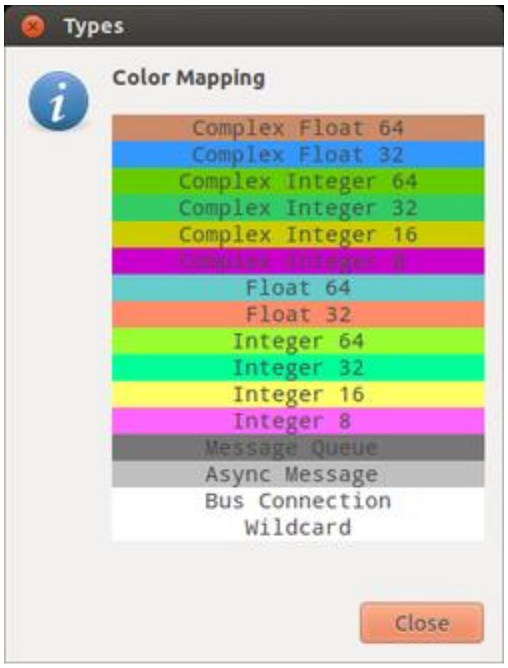
It should be noted that we are doing away with WX GUI in future releases so only use QT GUI.

Examining the Output

Let's go ahead and Execute the flowgraph to see our sine wave.



We should get the above which is a complex sinusoid of the form e jwt. Let us keep things simple by avoiding complex signals for now. First we want to kill the flowgraph with the Kill flowgraph button or by simply closing the Time Sink GUI. Now is a good time to go over data types in GNU Radio by opening up the Help->Types window.

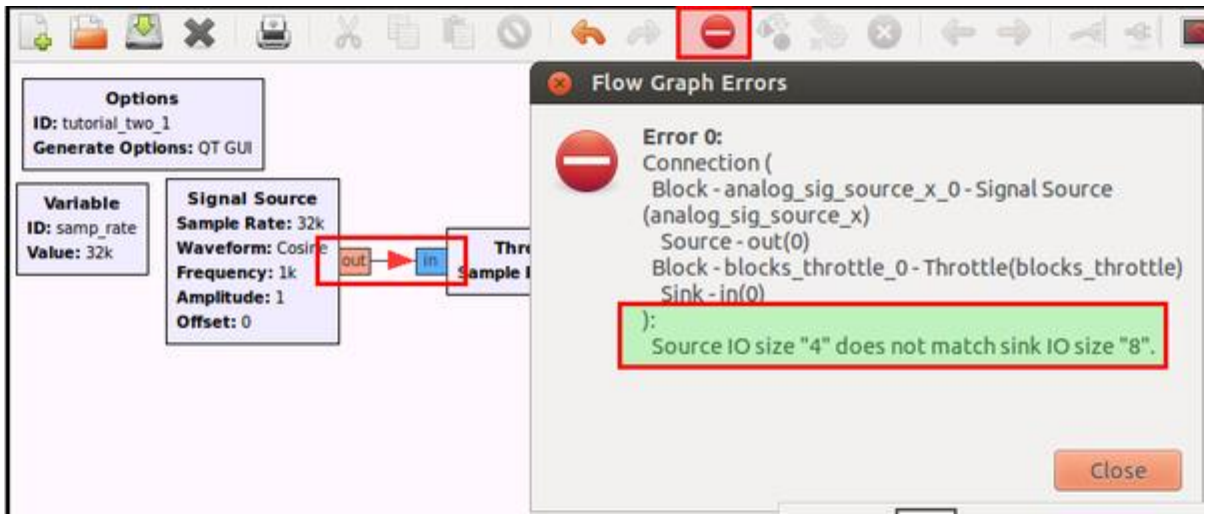


Types

Color Mapping

Complex Float 64
Complex Float 32
Complex Integer 64
Complex Integer 32
Complex Integer 16
Complex Integer 8
Float 64
Float 32
Integer 64
Integer 32
Integer 16
Integer 8
Message Queue
Async Message
Bus Connection
Wildcard

We can see common data types seen in many programming languages. We can see our blocks (blue ports) are currently Complex Float 32 type which means they contain both a real and imaginary part each being a Float 32 type. We can reason that when the Time Sink takes in a complex data type, it outputs both the real and imaginary part on separate channels. So let's now change the Signal Source to a float by going into its block properties and changing the Output Type parameter. We see that its port turns orange, there is now a red arrow pointing to the Throttle Block, and in the Toolbar there is a red "-" button (red) that reads "View flow graph errors". Let's go ahead and click that.



Options
ID: tutorial_two_1
Generate Options: QT GUI

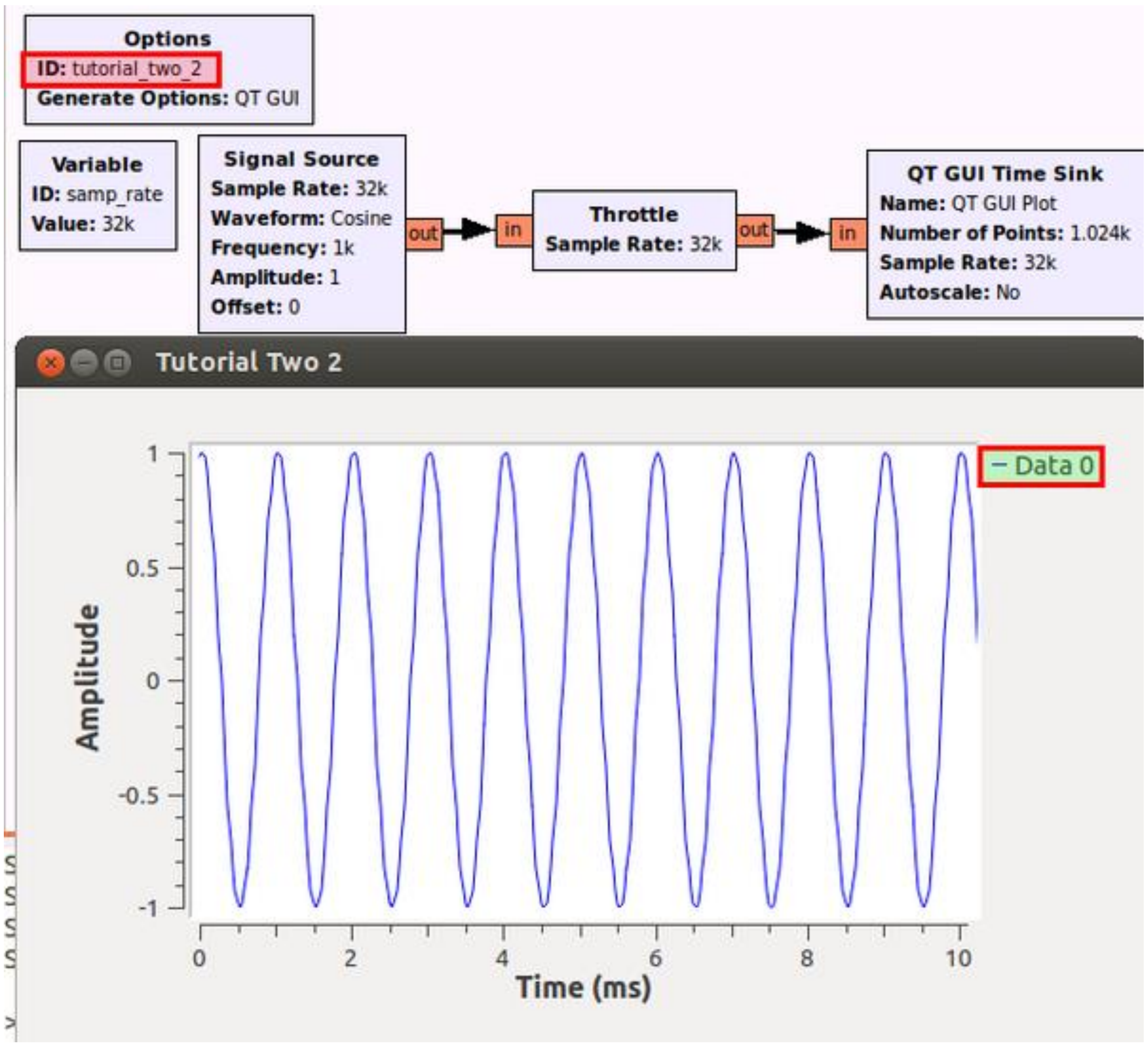
Variable
ID: samp_rate
Value: 32k

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0

Flow Graph Errors

```
Error 0:
Connection (
 Block - analog_sig_source_x_0 - Signal Source
(analog_sig_source_x)
  Source - out(0)
 Block - blocks_throttle_0 - Throttle(blocks_throttle)
  Sink - in(0)
):
 Source IO size "4" does not match sink IO size "8".
```

We see that in the specified connection, there is size mismatch. This is due to our data type size mismatch. GNU Radio will not allow us to chain together blocks of different data sizes, so let's change the data type of all of our subsequent blocks. We can now generate and execute as before.


Options
ID: tutorial_two_2
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0

Throttle
Sample Rate: 32k

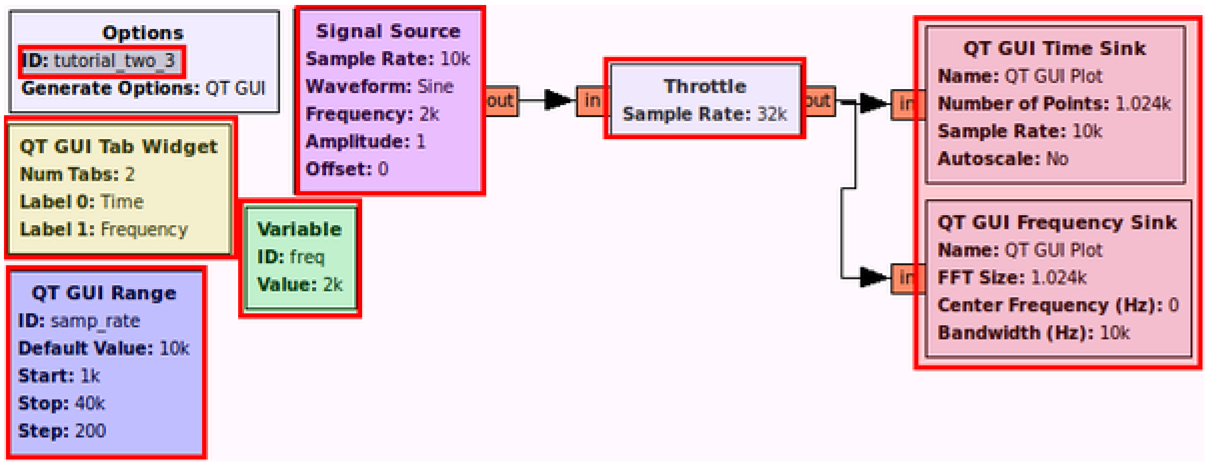
QT GUI Time Sink
Name: QT GUI Plot
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: No

We now see our sine wave on one channel. We can click on the screen and move the mouse to zoom and rescale.

A More Complex Flowgraph

Now that we are able to create flowgraphs on our own, lets try creating a more complicated flowgraph with many specific parameters. This example flowgraph demonstrates many new concepts in GNU Radio like using tabbed windows and QT GUI Ranges. Note that not all block parameters are displayed in the main window, so use the text below (not just the screenshot) to set the parameters of each block.

Time & Frequency Flowgraph


Options
ID: tutorial_two_3
Generate Options: QT GUI

QT GUI Tab Widget
Num Tabs: 2
Label 0: Time
Label 1: Frequency

QT GUI Range
ID: samp_rate
Default Value: 10k
Start: 1k
Stop: 40k
Step: 200

Signal Source
Sample Rate: 10k
Waveform: Sine
Frequency: 2k
Amplitude: 1
Offset: 0

Variable
ID: freq
Value: 2k

Throttle
Sample Rate: 32k

QT GUI Time Sink
Name: QT GUI Plot
Number of Points: 1.024k
Sample Rate: 10k
Autoscale: No

QT GUI Frequency Sink
Name: QT GUI Plot
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 10k

Detailed Changes:
- We are starting a new flowgraph with ID "tutorial_two_3"
- In QT GUI Tab Widget, ID to "tab", Num Tabs to 2, Label 0 to "Time", Label 1 to "Frequency"
- In QT GUI Range, ID to "samp_rate", Default Value to "5*freq", Start to "0.5*freq", Stop to "20*freq", Step to "200"
- In Variable, ID to "freq", Value to "2e3"
- In Signal Source, Frequency to "freq", Waveform to "Sine"
- In QT GUI Time Sink, GUI Hint to "tab@0". In QT GUI Frequency Sink, GUI Hint to "tab@1"
- In Throttle, Sample Rate to 32e3 (more on why later)

Once we have verified our changes, let's Generate, and Execute. It should produce a window that has two tabs, one showing the time domain and one showing the frequency domain. There should also be a slider at the bottom to control the sample rate (of the signal source) in realtime. Changing this slider should change the observed frequency in the time and frequency sinks.

Sampling rate is an interesting subject in GNU Radio -- and, indeed, any software radio platform. Please see the Extras on Sampling Rate page that explores how changing the sample rates in the above flowgraph affects the signals.

Conclusion

And that is it for now with GRC. Let us know your thoughts before going on to the python tutorial.

Hint: you have both Tutorials and Documentation of GNU Radio on the desktop of the bootable system. Please get also familiar with the Documentation.

Exercise: Follow the link below to get familiar with the usage of the hardware and using the hardware to listen to the RTS radio channel. Plot the constellation diagram.

https://wiki.gnuradio.org/index.php/Guided_Tutorial_Hardware_Considerations

Check point: Please show the outcome of the exercise to the lab instructor.

Optional task: Try to receive the Wifi signal in the environment and plot the spectrum.

Some general resources:

http://sdr.ninja/   **https://www.rtl-sdr.com/**

https://wiki.gnuradio.org/index.php/Guided_Tutorial_Introduction

https://wiki.gnuradio.org/index.php/Guided_Tutorial_GRC