Digital Communication Using GNU Radio

Prof Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

Week-01

Lecture-02

This is the first lecture where we will encounter the GNU radio software. It is essential that you have the GNU radio software installed and ready to perform the simulations which we discuss over here. We will be discussing the very basics of signals and blocks in GNU radio and put together several simple flow graphs that explore both the features of GNU radio and how you can perform basic signal processing on GNU radio. In our GNU radio demonstration, we will take a close look at various components of GNU radio including blocks, sources, sinks and the various interactive features that GNU radio offers. We will look at the signal model, sampling and processing features that GNU radio offers which enable you to put together various software and hardware signal processing and communication experiments. We will then compare and contrast hardware and software, sources and sinks and look at the issues that you must be aware of when you use them as part of your experiment.

Finally, we will take a close look at filtering examples wherein signals can be filtered to obtain different outputs using the inbuilt blocks in GNU radio. GNU radio is an open source software defined radio application. It is maintained by a community of developers and engineers. It is an open source model wherein you can download, view the source, make modifications and even contribute your own features and changes back to the GNU radio developers.

It is cross platform, it runs on Windows, Linux, Mac and it can operate both as a simulator as well as a hardware interface wherein it can be used along with a variety of audio and radio hardware to put together several complex experiments based on existing standards as well as new ones. You can get more information about GNU radio from the website gnu-radio.org. You can also download the latest version of GNU radio from this website. After installing GNU radio for your operating system, you must look for the GNU radio companion application.

The GNU radio companion application is a WYSIWYG or what you see is what you get

application that allows you to build simulations and hardware experiments using various GNU radio blocks. We will now take a tour of GNU radio and look at the various features that it has to offer. For the purposes of this particular lecture and the remaining lectures, we will be using GNU radio version 3.10.5.1. We recommend that you always use the latest GNU radio because it would always have the most recent features and bug fixes. There may be minor variations in the blocks that we discuss. However, we will stick to those blocks and features of GNU radio that are largely common among the most recent versions. Upon starting GNU radio, you will be greeted with a screen that looks similar to this.

You have a menu bar and a toolbar. At the bottom, you have some information panels and on the right, you have a list of various blocks that can be used and in the middle, you have a canvas where you draw what is called a flow graph. In order to change the size of this canvas, you can hold the control key or command key on your board and use your mouse scroller to zoom in or zoom out to adjust this font size according to your needs. By default, there are two blocks that are already present. The first is called options and the second is called variable.

In the options box, we can see the attributes title, output language which is Python which is the programming language which GNU radio is implemented in and generate options QT-GUI indicating that when you generate the program from your flow graph, it will result in a graphical user interface application that you can control interactively. You can modify any block by double clicking it. Let us double click this options box. You can now change various attributes of this block as well as find various features that you can infer about the block. We will change the id to my first and we will change the title to my first application.

We will leave the other options as they are and say ok. You can see that the changes that you have made are being reflected. The variable which is samp_rate is the variable that sets the sampling rate for your simulation or hardware experiment. This can be altered for all the blocks that use this variable by double clicking here and changing this value. We can now put together a very simple application that takes a source and allows us to visualize it.

To get a source, we need to look at the source block on the right panel. Unfortunately the right panel has a lot of options and it appears difficult to look for what you want unless you know the sub-menu in which you will find it. To find the block that you need easily, you can press control F or command F if you are using a Mac and type the name of the block. We are going to type signal. This will filter the menu and give you all the blocks that have the word signal in their name.

We need the signal source present within waveform generators. To add a signal source to your flow graph, you can click and drag it and place it on your flow graph. You can see our signal source. It has a sample rate of 32k which is same as what is written here. It has a frequency of 1k, unit amplitude, zero offset and zero initial phase.

We can now visualize this source. However, before doing that, since we are going to perform a simulation experiment, you need to add exactly one throttle block. So we will press control F and type throttle and you can drag and place the throttle over here. A throttle block is necessary because whenever you perform simulations, you need to tell Clue Radio to limit the CPU usage. There should be exactly one throttle block whenever you perform a simulation and there should be no throttle blocks whenever you have interfaced any hardware either a source or a sink.

The next course of action is for us to connect the signal source to the throttle. You can see that the signal source title is red. Once we connect the signal source to the throttle which is by clicking the out button here and then clicking the in here, you will see that the signal source is now black in color indicating that there is no error. In general, red text on the title of a block indicates an error. We will now visualize the signal source.

To do so, we need to add a time sync to our flow graph. You can press control F or command F and type time and here you have a QT GUI time sink. You can drag this and place it on your flow graph and you can connect the out of the throttle to the in of the time sink. Let us now see what this program will look like when we run it. The first time you hit this play button which executes the flow graph, it will ask you to save the file.

I will call it myfirst.grc where grc is short for GNU radio companion files and once you save it, you are greeted with a window. The title of the window is myfirstapplication which is the same title that you had chosen and there is a graph containing some signals. By maximizing the window, we can see that there are two signals, a blue signal and a red signal. Let us use the mouse to zoom in by clicking and dragging within the window.

We can clearly see that the blue signal starts at amplitude 1 at 0 milliseconds, looks like a cosine and at 1 millisecond reaches the amplitude 1 once again indicating that this is a 1 kilohertz cosine waveform. The red signal on the other hand starts at 0, reaches the peak value of 1 at 0.25 milliseconds and once again reaches the peak value of 1 at 1.25 milliseconds clearly indicating that this is a sine waveform at the same 1 kilohertz frequency. Why do we get two different waveforms? One indication can be obtained when we close and return to our flow graph.

When we double click our signal source, we get the output type as complex. In GNU radio, the signal source when chosen with the complex data type results in the signal

$e^{j\omega_0 t}$

which in this case is

$e^{j2\pi 1000t}$

which gives you a

$\cos(1000\times 2\pi t)$

plus a

$\sin(1000\times 2\pi t)$

multiplied by j. In other words, GNU radio always interprets complex signals

$e^{j\omega_0 t}$

as a pair of real and imaginary part signals in this case it is

$\cos(\omega_0 t)+j\sin(\omega_0 t)$.

Thus, the two signals that you saw are cos which is the blue part and the sine the red part the blue part is the real part, the red part is the imaginary part. To understand a little more about signals and blocks in GNU radio, let us first look at what the data types in GNU radio are.

The most basic data types for numeric data are complex indicating a pair of real numbers, float indicating a real number that is stored in the floating point notation, int referring to integers, short referring to integers that may have a smaller size than int and byte that corresponds to byte data. Sources can produce a finite or infinite sequence of values. A signal source such as the one you saw can produce a sine wave or a triangle wave or several other such signals. A vector source takes a set of samples and converts them to a waveform. A file source reads data from a file and converts that to a signal.

Sinks on the other hand consume samples and potentially output or display them. For example, time or frequency display sinks consume samples and yield visual outputs using which you can infer some properties of the signal. File sinks can be used to write samples to a file for later analysis. Other sinks such as speakers or USB radio systems consume samples and output them as audio signals or radio signals that are transmitted outside through actual hardware. We will soon see an example of an audio sink that uses the PC's speakers.

Finally, there is a null sink that consumes samples but does not do anything with them.

This comes in quite handy when you have to perform some debugging operations and you do not want any other sinks. So with this new knowledge, let us first understand how we can fix this two signal issue. By double clicking the signal source, we see that we want only a real signal therefore we change this to float. The complex signal which was e power j omega 0 t which resulted in cos omega 0 t plus sine omega 0 t has now been converted to a float signal that gives you only cos omega 0 t.

You now see that this connector has turned red because the orange indicating float cannot be connected to the throttle that has a complex data type. To address this, you can double click the throttle, change the type to float and again when you look at the output of the throttle to the QT-GUI time sink, we double click the time sink, change the type to float, say ok and now if we hit execute, we now see that we have a single waveform which is the cosine. By zooming in, we see that the waveform starts at 0 and with amplitude 1 and the next amplitude 1 happens at 1 millisecond indicating that this is a 1 kilohertz sinusoid. The plot that is shown is also quite flexible. By pressing the middle button of your mouse, you are greeted with a menu.

This menu can be used to change several aspects of your plot. For example, you can choose to have a grid, you can middle click and also change several aspects of your plot. You can also choose auto scale which will automatically scale the amplitude of your signal and make several such changes. Let us now close this window. One important aspect of signals in GNU radio is that they are held in the discrete time representation.

As we are aware, real world signals are generally continuous time. For example, an electromagnetic wave is a continuous time signal that has a value for every instant of time. However, computers can only hold discrete time signals since they have finite precision, finite space. Therefore, we need to operate completely in the sampled mode. To observe this practically by executing our flow graph, if we middle click our button and change this to stem plot.

Upon zooming in, you will see that you do not have complete cycles, but only certain sticks. In fact, there is a significance between the values of these sticks and the original signal. Moreover, the time difference between any two of these stems is also significant. If you carefully look at this, this is 12.78 and this is around 12.81. If you take the accurate values and take the reciprocal of the time difference between these two stems that will be 32000 that corresponds exactly to the sampling rate. Moreover, the sinusoid that is present behind these sticks actually looks pretty accurate and reliable. Why is this the case? So, if you look at a sinusoid or any continuous time signal, because of the infinite number of values, they cannot be represented in computers. Therefore, what we do is to sample the signal sufficiently well to capture the signal properties. In other words, you have enough measurements of the signal that allows you to recover the signal

even from just these samples.

The discrete time representation thus obtained is suitable for representation as well as processing on a computer or any other computing device. The key realization here is that the original signal can be recovered by interpolation or using a digital to analog converter, because if you satisfy the condition for recovery of the signal, which you may recall from DST is the Nyquist sampling criterion, then the original signal can be obtained once again without any loss. In this lecture, we have seen the very basics of how you can use the blocks in GNU radio to put together very simple simulations and how you can extend these blocks and visualize the signals in a very easy way. In the next lecture, we will continue our exploration of GNU radio and see how you have more advanced blocks with better user interface components to allow for richer applications as well as interfacing with hardware such as the sound card of your PC. Thank you.