Lab 1 - Introduction to Software-Defined Radio

ECE531 – Software Defined Radio

Spring 2022

Daniel Gallagher, danielgallagher@arizona.edu
Department of Electrical & Computer Engineering
University of Arizona, Tucson AZ 85721

Due Date: *February* 3, 2022

---

1 Overview

This laboratory will introduce basic software-defined radio (SDR) concepts through GNU Radio simulations.

2 Required Software

This course will require a number of different software tools. Many of these tools are compatible with Windows, Linux, and OSX operating systems. While the goal of the labs will be to remain platform agnostic, there are many SDR software tools which are available solely for the Linux operating system.

Students are encouraged to take the time to setup a Ubuntu Linux installation for use in this class and the final project [1]. This can be accomplished in a variety of methods; including a dual-boot, "Live USB" environment [2], or through visualization software like VMware or VirtualBox. An Instant GNU Radio [3] OVA VM image has been posted to D2L and can be used for this lab. The VM is reconfigured with PlutoSDR support.

2.1 GNU Radio

GNU Radio is a free & open-source software development toolkit that provides signal processing blocks to implement software radios [4]. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in research, industry, academia, government, and hobbyist environments to support both wireless communications research and real-world radio systems.

GNU Radio 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. It 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, and much more - all in computer software.

It has filters, channel codes, synchronisation elements, equalizers, demodulators, vocoders, decoders, and many other types of blocks which are typically found in signal processing systems. More importantly, it includes a method of connecting these blocks and then manages how data is passed from one block to another.

Extending GNU Radio is also quite easy; if you find a specific block that is missing, you can quickly create and add it. The GNU Radio community has releases am always-expanding library of 3rd party GNU Radio applications. The Comprehensive GNU Radio Archive Network (CGRAN) is a free open source repository for Out-Of-Tree (OOT) Modules that are not officially supported by the GNU Radio project [5]. "gr_modtool" is also available to assist in custom GNU Radio module and signal processing block creation [6].

2.1.1 Install GNU Radio

Obtain and install a copy of GNU Radio to simulate software-defined radio baseband signal processing. A prebuilt GNU Radio windows binary installer was distributed by Analog Devices Inc. (ADI) and may now be considered outdated. This installer includes native "IIO" support needed to interface with the Pluto hardware. This installer is also archived on the class D2L web site.

VirtualBox VM: Preconfigured Linux Virtual Machine image available here.
radioconda: Conda-based Python Distribution available here.
Windows Installer: Windows installer is available here. (version 3.7)
Linux Installer: Linux install instructions are here.

2.2 MATLAB

MATLAB also provides support needed to prototype and test software-defined radio (SDR) systems using Analog Devices ADALM-PLUTO with MATLAB and Simulink. An academic license for MATLAB and toolboxes is available at no cost to UA students at: `https://softwarelicense.arizona.edu/mathworks-matlab`.

2.2.1 Install MATLAB with Toolboxes

Students may choose to install all toolboxes with their MATLAB installation. This may grant some added capability for final projects. At a minimum, The Mathworks suggests the following toolboxes to support the PlutoSDR hardware:

Required Toolboxes:

- MATLAB
- Communications Toolbox
- DSP System Toolbox
- Signal Processing Toolbox

Recommended Toolboxes:

- Simulink
- LTE Toolbox
- WLAN Toolbox

MATLAB installation can be interfaced with hardware such as the PlutoSDR through Hardware Support Packages [7]. We will install the Hardware Support Package and begin working with the PlutoSDR hardware in Lab 2.

2.3 Git

Git is a distributed version control system (DVCS) used heavily for collaboration of open-source project such as GNU Radio. A vast majority of SDR software and signal processing algorithms are hosted on GitHub. Source code for the software, firmware, drivers, and utilities needed for the PlutoSDR hardware used in this course are publicly available on the Analog Devices github account [8].

Github is an effective way to collaborate and manage source code for the project. Students can also receive a free upgraded account and access to a numerous software tools at https://education.github.com/students. You will require a free GitHub account. A git software installer is available at https://git-scm.com/.

3 Introduction

A software defined radio system (SDR) is a radio communication system where components that have been typically implemented in hardware (e.g., mixers, filters, amplifiers, modulators/demodulators, detectors) are instead implemented by means of software on a personal computer or embedded system. Most conventional radios utilize the classic superheterodyne receiver architecture, similar to what is shown in Figure 1. In a superheterodyne architecture, the RF signal from the antenna is mixed with a local oscillator to produce an intermediate frequency, or IF. Conversely, SDR uses direct-conversion architecture, shown in Figure 2. A direct-conversion transceiver, also known as homodyne, synchrodyne, or zero-IF transceiver, is a radio transceiver design that (de)modulates the radio signal using a local oscillator (LO) whose frequency is identical to, or very close to, the carrier frequency of the intended signal. Direct-conversion eliminates the IF by translating the band of interest directly to baseband (DC).

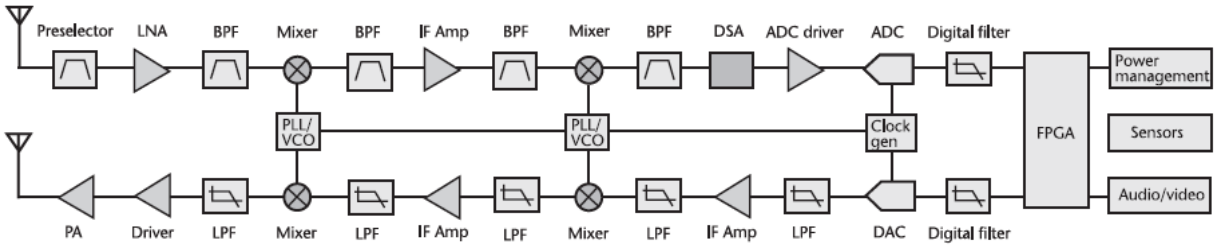

Figure 1: Multistage superheterodyne receive and transmit signal chains

Comparing Figure 1 and Figure 2, direct conversion is attractive due to simplicity of the signal path and is an enabling architecture for software-defined radio.

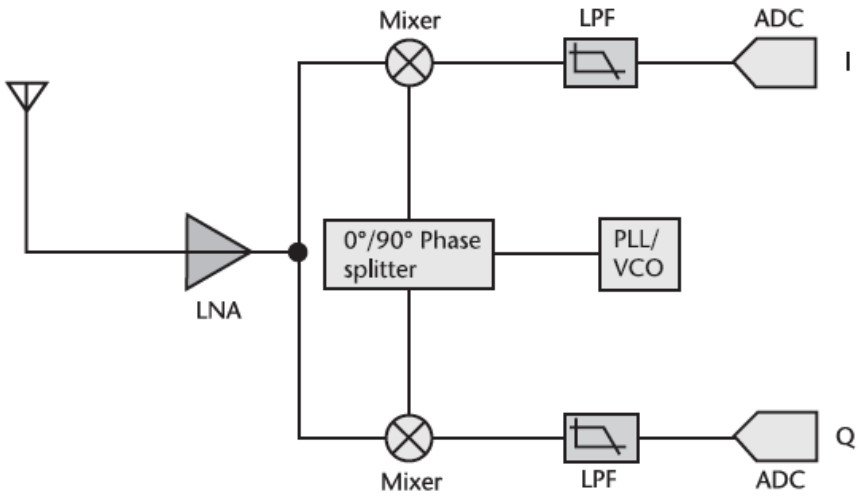

Figure 2: Direct conversion, or Zero-IF transceiver architecture used for software-defined radio.

3.1 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. 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). Start the GRC by typing `gnuradio-companion` in a terminal window in Linux and you will see an untitled GRC window similar to Figure 3.

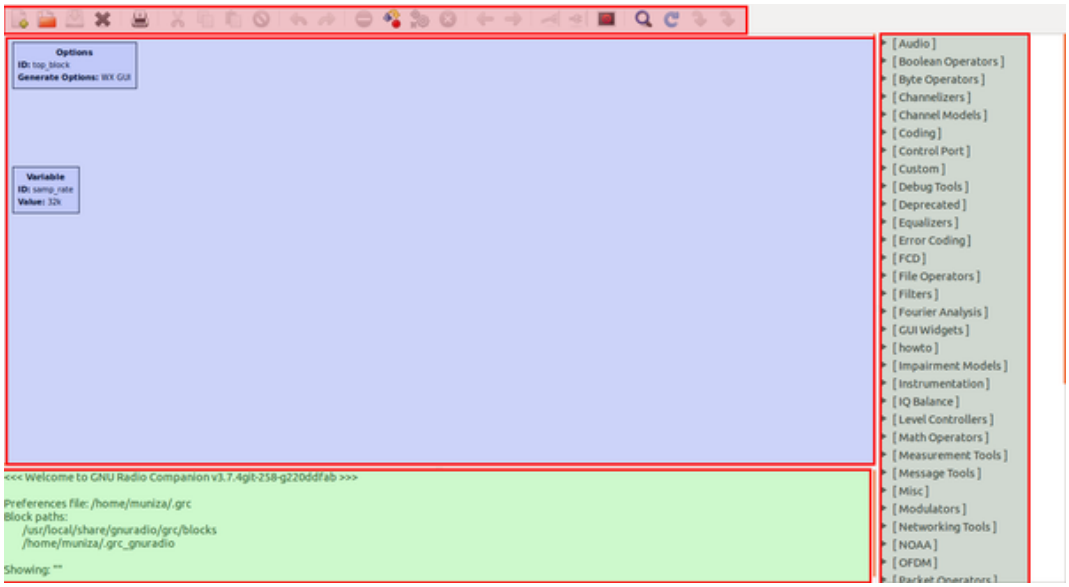

Figure 3: The GNU Radio Companion Interface. There are four main parts: Library, Toolbar, Terminal, and Workspace.

3.2 Sampling Rates

The definition of proper sampling is quite simple. Suppose you sample a continuous signal in some manner. If you can exactly reconstruct the analog signal from the samples, you must have done the sampling properly. Even if the sampled data appears confusing or incomplete, the key information has been captured if you can reverse the process

Given the flowgraph shown in Figure 4, we have a single sinusoidal frequency source with a frequency of 2 kHz. The sampling rate is able to be changed at runtime using the "QT GUI Range" block. Each flowgraph block is colored orange to denote a real, Float 32, input/output type. You can lookup the color for each data type from the GRC help menu.

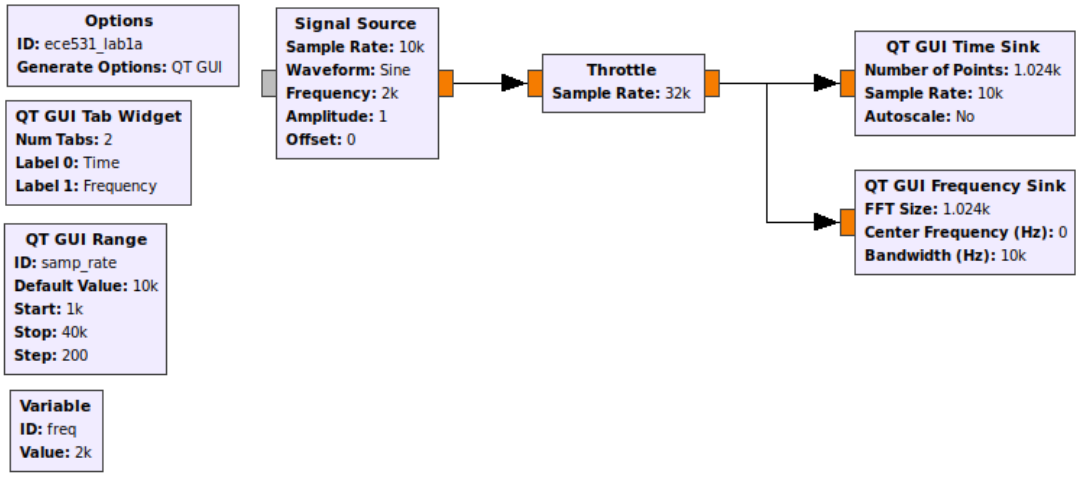

Figure 4: GRC flowgraph for Lab 1, Section 3.2

Given the flow graph in Figure 4, complete the following:

1. Construct and execute the GRC flowgraph shown in Figure 4.
2. Visualize the sinusoidal signal in time and frequency, provided by the "Time" and "Frequency" sink tabs.
3. Using the default 10 kHz sample rate: Zoom in on the time domain signal. Does this look as expected? Determine the location of the frequency peak. Hover over the peaks and measure the frequency location.
4. While continuing to observe frequency tab, slowly adjust the sampling rate from 10 kHz to 40 kHz. Determine the measured frequency? (Note: you may need to give the frequency sink a moment to adjust to the new sample rate.)
5. What does the time domain signal look like at this new sample rate? Why?
6. Adjust the sampling rate to 3.5 kHz. What is the measured frequency? Did the measured frequency remain the same or did it change? Why? If so, how much did the frequency change and why?
7. Save the grc file with a unique filename for use in later sections.

3.3 Complex Sampling

Direct conversion transceivers, like shown in Figure 2, utilize in-phase (I) and quadrature (Q) signal paths to sample a complex signal. In-phase refers to the signal that is in the same phase as the local oscillator (LO), and quadrature refers to the part of the signal whose phase is shifted by 90°.

A bandpass signal can be represented by the sum of its in-phase (I) and quadrature (Q) components:

$$S(t) = I(t)\cos(2\pi f_c t) - Q(t)\sin(2\pi f_c t). \tag{1}$$

where $I(t)$ is in-phase amplitude, $Q(t)$ is quadrature amplitude and f_c is carrier frequency.

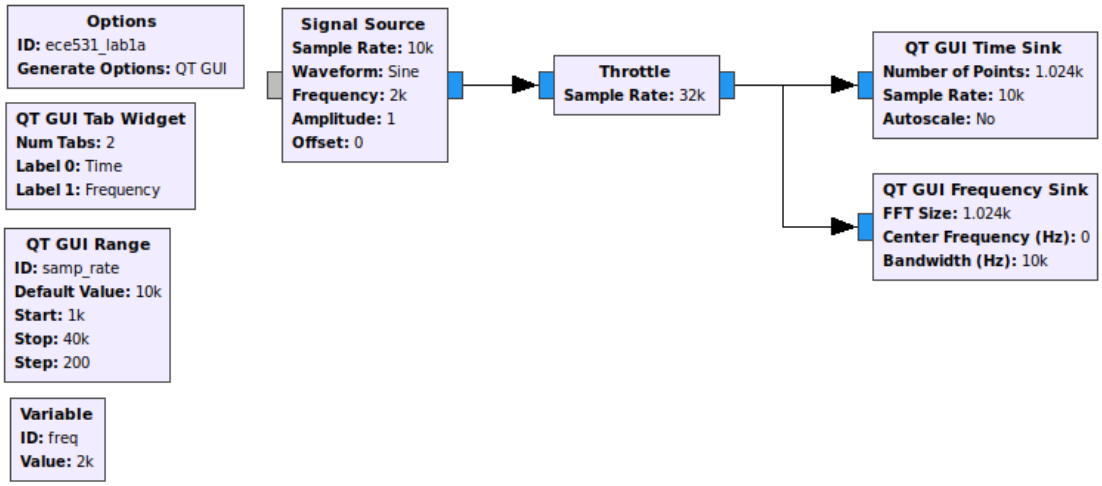

Figure 5: GRC flowgraph for Lab 1, Section 3.3

Given the flow graph used previously in Figure 4:

1. Change all block input/output types from "Float 32" to "Complex Float 32". The port color on each block should change from orange to blue as shown in Figure 5. (Note: You can quickly change the input/output type by highlighting the block(s) and pressing the up/down arrows).
2. Visualize the sinusoidal signal in the frequency domain. What has changed? Why?
3. What are the two time domain waveforms shown on the "Time" tab? What is the phase relationship between them?
4. While observing the time domain tab, slowly adjust the sampling rate from 10 kHz to 40 kHz. Does the signal amplitude appear as expected for each channel? (Note: You may wish to zoom in on the time signal).
5. What is the measured frequency peak at 40 kHz sampling rate?
6. While observing the frequency domain tab, slowly decrease the sampling rate to 4 kHz. What is observed?
7. Continue to slowly decrease to slower sampling rates. What happens? Why?
8. Save the grc file with a unique filename for use in later sections.

3.4 Frequency Observations

3.4.1 Complex-sampled Flowgraph

Given the complex-sampled flowgraph used previously in Figure 5:

1. Add an adjustable frequency slider using a QT GUI Range block with the following parameters:

   Frequency Default value: 2e3, Start: 0, Stop: 15e3, Step: 100
2. Set the signal source frequency to the variable ID used for the QT GUI Range block above.
3. Using a fixed 10 kHz (default) sample rate: Slowly increase the frequency to the maximum frequency value while observing the frequency sink output.
4. Describe what anomaly is observed when increasing the frequency slider. What is this anomaly? Why does it occur?

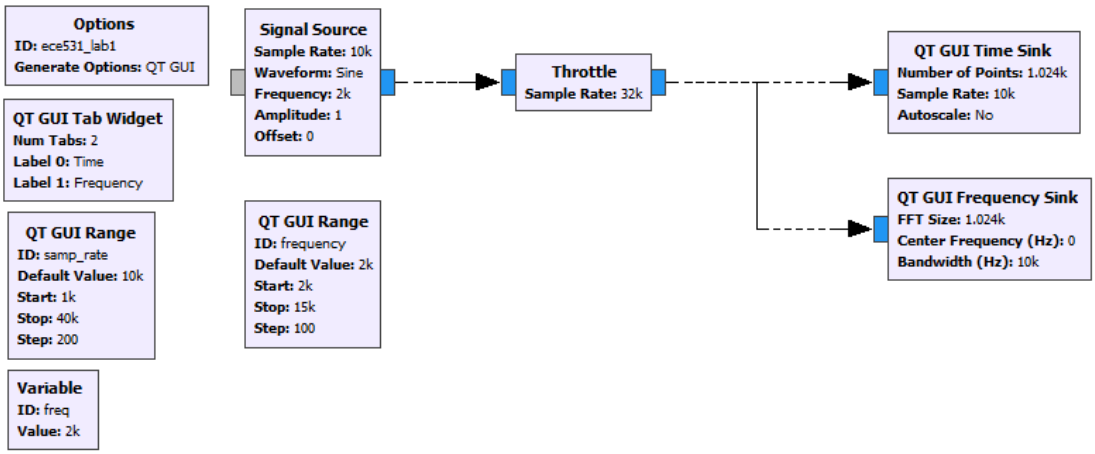

Figure 6: GRC flowgraph for Lab 1, Section 3.4

3.4.2 Real-sampled Flowgraph

Given the flowgraph used previously in Figure 4, or by changing the types to real; Repeat the steps above, specifically:

1. Add an adjustable frequency slider using a QT GUI Range block with the following parameters:

   Frequency Default value: 2e3, Start: 0, Stop: 15e3, Step: 100
2. Set the signal source frequency to the variable ID used for the QT GUI Range block above.
3. Using a fixed 10 kHz (default) sample rate: Slowly increase the frequency to the maximum frequency value while observing the frequency sink output.
4. What is observed when increasing the frequency? How and why is this different than what was observed in Section 3.4.1?

3.5 I/Q Imbalance

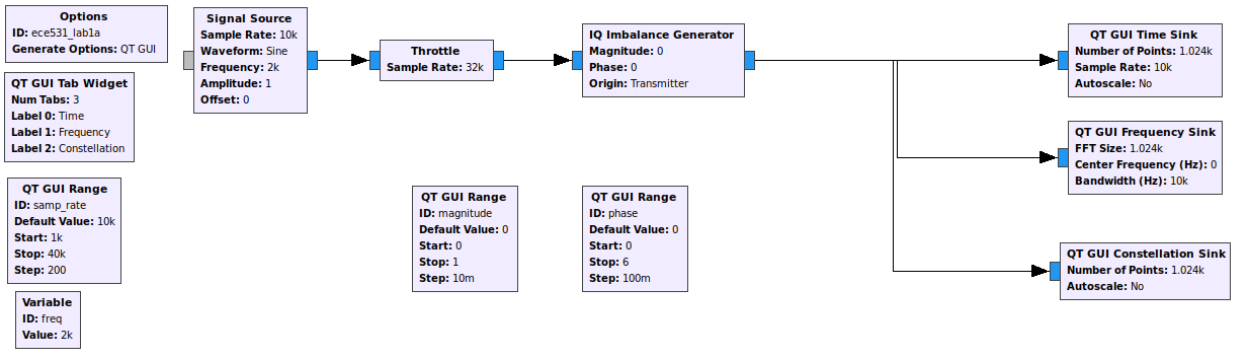

Figure 7: GRC flowgraph for Lab 1, Section 3.5

Given the flow graph used previously in Figure 5:

1. Add an IQ Imbalance Generator block to the flowgraph somewhere in the signal path between the signal source and sink blocks. An example is shown in Figure 7
2. Create two QT GUI Range blocks, one with variable ID "magnitude" and another with ID "phase". Set the following parameters respectively:

   Magnitude Default value: 0, Start: 0, Stop: 1, Step: 0.1
   Phase Default value: 0, Start: 0, Stop: 6.2, Step: 0.1
3. Set the IQ Imbalance Generator values for magnitude and phase to the variable IDs provided to the GUI Range blocks. (Note: these happen to be the Python variable names. You can use Python syntax in most block fields).
4. Add a "QT GUI Constellation Sink" block to a third tab and connect to the same location as the other sinks.
5. Your flowgraph should now resemble the one shown in Figure 7.
6. Execute the new flowgraph. The time and frequency domain should match what was observed previously.
7. While observing the frequency domain tab, slowly increase the phase imbalance from zero. What happens? Why?
8. Return the phase offset to zero and then slowly increase the amplitude imbalance from zero. What happens? Why?
9. While observing the constellation sink output, adjust each of the three variables independently. What happens when each slider is adjusted?

3.6 Adding Noise

Given the flow graph used previously in Section 3.5:

1. Add gaussian noise to the signal using the "Noise Source" and "Add" blocks.
2. Describe what happened to the signals displayed on each tab; time, frequency, and constellation.
3. Use the "QT GUI Histogram Sink" blocks to verify the pdf of the various noise source types by connecting directly to the noise source.

3.7 Interpolation and Decimation

The following flowgraph uses the three signal sources with identical parameters; including frequency. Each source is resampled using a rational resampler block and displayed on three-port sink blocks.

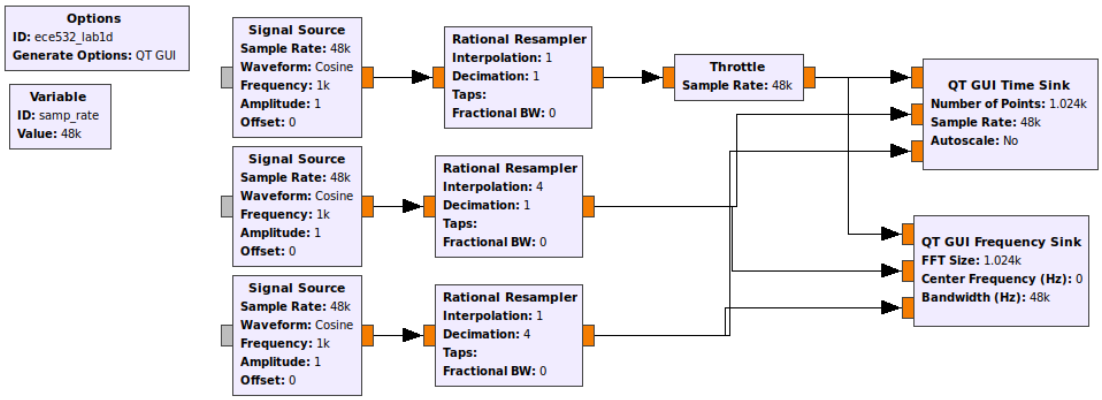

Figure 8: GRC flowgraph for Lab 1, Section 3.7

Given the flow shown in Figure 8:

1. Construct and run the given flowgraph.
2. Observe the time and frequency domain outputs.
3. What do you observe with the three waveforms?
4. The three signal sources have identical properties including frequency. Why do they differ when the frequency spectrum is visualized?
5. What are the relationships between the waveforms? Which one is interpolated and decimated?

4 Questions

1. What are the benefits of using in-phase and quadrature (I & Q) samples for SDR? Fully describe at least three benefits of IQ sampling.
2. In GNU Radio What is the Throttle block for? When should it be used? What happens if you use more than one throttle block? When is a throttle block unnecessary?
3. What are Nyquist zones? How are they useful with software-defined radio?
4. Why is dither noise used on Digital-to-Analog converter (DAC) circuits?

5 Lab Report Preparation & Submission Instructions

Include all your answers, results, and source code in a laboratory report formatted as follows:

- Cover page: includes course number, laboratory title, name, submission date.
- Suggested: Table of contents, list of tables, list of figures.
- Commentary on designed implementations, responses to laboratory questions, captured outputs, and explanation of observations.
- Meaningful conclusions to the lab.
- Source code (as an appendix). You may also upload source files with report submission.

Remember to write your laboratory report in a descriptive approach, explaining your experience and observations in such a way that it provides the reader with some insight as to what you have accomplished. Furthermore, please include images and outputs wherever possible in your laboratory report document.

References

[1] Canonical Ltd. (2018) Download ubuntu desktop. [Online]. Available: https://www.ubuntu.com/download/desktop

[2] Corgan Labs. (2017) GNU Radio Live SDR Environment. [Online]. Available: https://wiki.gnuradio.org/index.php/GNU_Radio_Live_SDR_Environment

[3] B. Bloessl, "Github: Instant gnu radio." [Online]. Available: https://github.com/bastibl/instant-gnuradio

[4] GNU Radio Foundation, Inc. GNU Radio. [Online]. Available: https://www.gnuradio.org/

[5] The Comprehensive GNU Radio Archive Network. [Online]. Available: http://cgran.org

[6] E. Blossom, "gr_modtool: Swiss Army Knife for editing GNU Radio modules and -components." November. [Online]. Available: https://wiki.gnuradio.org/index.php/OutOfTreeModules#gr_modtool_-_The_swiss_army_knife_of_module_editing

[7] The Mathworks. ADALM-PLUTO Radio Support from Communications Toolbox. [Online]. Available: https://www.mathworks.com/hardware-support/adalm-pluto-radio.html

[8] Analog Devices, "GitHub: Analog Devices Inc." [Online]. Available: https://github.com/analogdevicesinc