

Lab 2 - Getting Started on PlutoSDR



ECE531 – Software Defined Radio

Spring 2024

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

1	Overview	2
2	Required Software	3
2.1	MATLAB	3
2.2	GNU Radio	5
2.3	IIO-Scope	5
2.4	Hardware Drivers	6
2.5	Putty SSH Client	6
2.6	Other SDR Programming Methods	7
3	Getting Started with PlutoSDR	7
3.1	Industrial Input / Output (IIO)	8
4	Radio Setup and Environmental Noise Observations	10
4.1	Signal Loopback	12
4.2	Measurements and the Radio	14
5	Lab Report Preparation & Submission Instructions	16

1 Overview

This lab introduces the ADALM-PLUTO Active Learning Module (PlutoSDR); an easy to use tool available from Analog Devices Inc. (ADI) that can be used to introduce fundamentals of software-defined radio (SDR) [1]. Based on the AD9363, it offers one receive channel and one transmit channel which can be operated in full duplex, capable of generating or measuring RF analog signals from 325 to 3800 MHz, at up to 61.44 Mega Samples per Second (MSPS) with a 20 MHz bandwidth. A software update increases this capability beyond what ADI advertises [2]. The PlutoSDR box includes two small 4cm long whip antennas a short 15 cm SMA cable and USB cable.

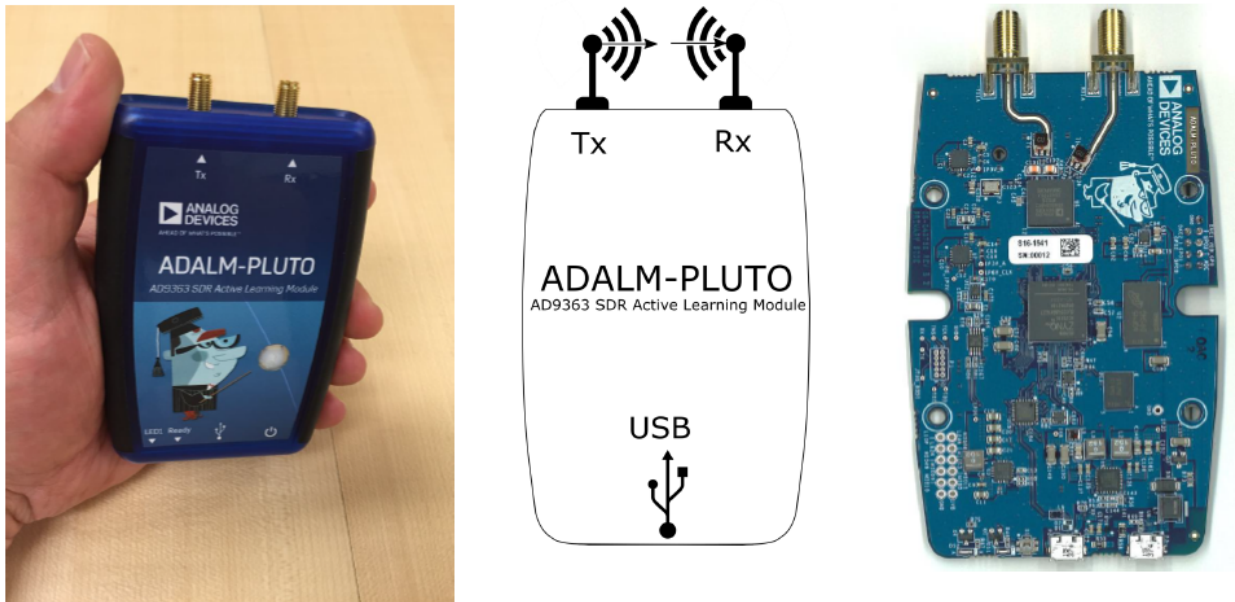


Figure 1: The PlutoSDR from Analog Devices

The radio inside the ADALM-PLUTO is the AD9363, is a high performance, highly integrated RF agile transceiver, based on is a direct conversion receivers.

- The receive subsystem includes a low noise amplifier (LNA), the direct conversion mixer, configurable analog filters, a high speed analog to digital converter (ADC), digital decimation filters, and a 128-tap finite impulse response (FIR) filters to produce a 12-bit output signal at the appropriate sample rate. The receive chain is augmented with configurable automatic gain control (AGC) or manual gain modes, dc offset correction, quadrature correction. The resulting received I and Q signals are passed onto the to the digital baseband processor, in this case the Xilinx Zynq SoC.
- The transmit subsystem also use a direct conversion architecture. Accepting 12-bit I and Q samples from the baseband processor (in this case, the same Xilinx Zynq SoC), running them through the 128-tap finite impulse response (FIR) filters, digital interpolation filters, a high speed digital to analog converter (DAC), an analog filter, the direct conversion mixers, and small power amplifier (PA) out to the antenna.

- Fully integrated phase-locked loops (PLLs) inside the AD9363 provide clocks and local oscillators for receive and transmit channels, and clocks for the ADC, DAC and output sample rate.

2 Required Software

The subsections below outline how to get the required software tools needed for this course.

2.1 MATLAB

MATLAB and all required toolboxes should have been installed previously as part of Lab 1. Otherwise, install MATLAB and necessary toolboxes prior to beginning Lab 2. To add PlutoSDR capability to this installation, a Mathworks Hardware Support Package (HSP) must be installed.

2.1.1 SDR Hardware Support Packages

MathWorks Hardware Support Packages can be considered 'add-ons' that provide specific MATLAB and Simulink interfacing support for third-party hardware, including SDR hardware such as the PlutoSDR, RTL-SDR, and Ettus USRP. The process for installing the HSP also installs necessary system drivers for Linux or Windows. Note: Hardware Support Package software installation requires a free MathWorks account.

The HSP installation can be found through the MATLAB add-on explorer located on the home toolbar, as shown in Figure 2

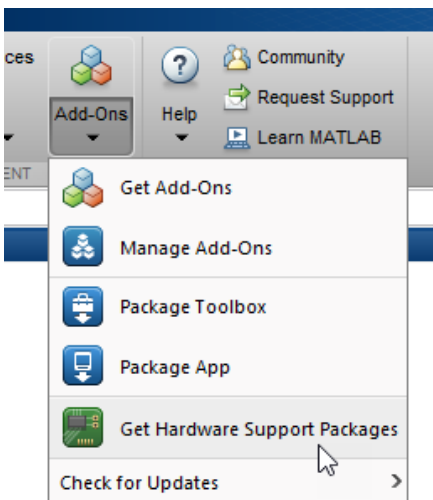


Figure 2: Mathworks Add-ons – Get Hardware Support Packages

In the Add-on explorer, you can find numerous HSPs among other usefull add-ons. A search for “SDR” results in five add on packages, including “Communications Toolbox Support Package for

Analog Devices ADALM-Pluto Radio” [3]. Another useful SDR HSP to consider is support for the affordable, RX only, RTL-SDR dongle [4].

You can initialize the PlutoSDR HSP from this menu by selecting the appropriate package then selecting install. The menu will guide you through the setup of all required drivers and libraries needed for your operating system, in addition to updating the PlutoSDR firmware. The AD9363 RF in the Pluto SDR is specified to operate from 325MHz to 3.8GHz. This support package automatically reconfigures the radio to operate over a wider tuning range.

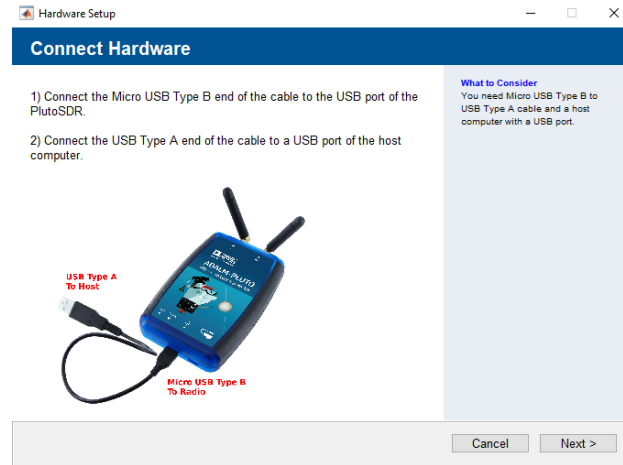


Figure 3: MATLAB PlutoSDR Hardware Setup

If you ever need to re-run the hardware setup wizard, run the following from the MATLAB console:

```
>> targetupdater
```

Once finished installing the PlutoSDR HSP, run the demo software to ensure it functions properly.

```
>> plutoradiodoc
```

Test your new PlutoSDR hardware using the Mathworks demos found by typing:

```
>> plutoradioexamples % Display PlutoSDR Examples
```

2.2 GNU Radio

The PlutoSDR hardware is fully supported in GNU Radio with `gr-iio` [5]. If you are using the provided virtual machine image or have installed a Windows installation as part of Lab 1, your install already supports the hardware. If you are using Linux, you should be able to install from binary packages or source. Note: the class virtual machine may have been updates to `gr-iio` not available with the native GNU Radio version. For example, the `gr-iio` master has the ability to use exponential notation in the block parameters. The GNU Radio 3.10 does not at the time of writing. Other version discrepancies may arise as the GNU Radio 3.10 release matures.

- The ADI wiki has GNU Radio instructions at:
<https://wiki.analog.com/resources/tools-software/linux-software/gnuradio>.
- PyBOMBS has been used in the past to install `gr-iio` for the virtual machine Linux configuration. More info can be found at <https://github.com/gnuradio/pybombs>. Using PyBOMBS is no longer the preferred method for building GNU Radio at the time of writing but is useful to know about.

For those using PyBOMBS to build GNU Radio and OOT modules from source, you will need to update the `gr-iio` recipe to use the master branch. This can be achieved using the command below.

```
$ sed -i 's/gitrev.*/gitbranch: master/' ~/.pybombs/recipes/gr-recipes/gr-iio.lwr
$ pybombs install gr-iio
```

PlutoSDR support can also be achieved using the SoapySDR hardware abstraction library and generalized API at <https://github.com/pothosware/SoapySDR/wiki>.

2.3 IIO-Scope

The ADI IIO Oscilloscope allows the user to interface with a variety of ADI SDR devices including the PlutoSDR. It is highly recommended that users install this software to understand and troubleshoot the many registers on the PlutoSDR transceiver chip. Early version of the virtual machine install IIO Oscilloscope from source, but may not be available on the latest VM image. An up-to-date windows installer is available from the releases section of Github.

Documentation: [IIO Oscilloscope](#)

- Windows installers are available here:
<https://github.com/analogdevicesinc/iio-oscilloscope/releases>
- Linux users will need to build from source:
<https://github.com/analogdevicesinc/iio-oscilloscope>

The method used to install IIO-Scope, libad9361, libiio, and more on Ubuntu Linux for virtual machine image configuration was achieved using the “`adi_update_tools.sh`” shell script; located

at: https://github.com/analogdevicesinc/linux_image_ADI-scripts/blob/master/adi_update_tools.sh. This script automatically builds basic IIO resources on Linux; including the drivers listed below.

2.4 Hardware Drivers

Most of the drivers needed to use the PlutoSDR are installed automatically for your operating system with the MATLAB Add-on wizard. If you need to install the drivers and libraries manually, the links in this section will be helpful.

2.4.1 OS Interface Drivers

Windows: <https://wiki.analog.com/university/tools/pluto/drivers/windows>

Linux: <https://wiki.analog.com/university/tools/pluto/drivers/linux>

Mac: <https://wiki.analog.com/university/tools/pluto/drivers/osx>

2.4.2 libIIO library

Documentation: [What is libiio?](#)

- Packages for Linux and Windows are available here:
<https://github.com/analogdevicesinc/libiio/releases/latest>
- Source available here: <https://github.com/analogdevicesinc/libiio>

After installing libIIO, you should now be able to interact with the PlutoSDR from the command-line. Example commands are listed in Section 3.1.

2.4.3 libad9361-iio library

- For linux users only, you will need the libas9361 library at:
<https://github.com/analogdevicesinc/libad9361-iio>
- Instructions for building it are [here](#).

2.4.4 PlutoSDR Firmware

- Pluto/M2k Firmware Update instructions can be found at:
<https://wiki.analog.com/university/tools/pluto/users/firmware>.

2.5 Putty SSH Client

The PlutoSDR hosts an SSH server. Linux includes the SSH client software necessary. However, Windows requires the installation of an additional tool. Putty is a popular solution for this on Windows. Putty is available at: <https://www.putty.org/>.

2.6 Other SDR Programming Methods

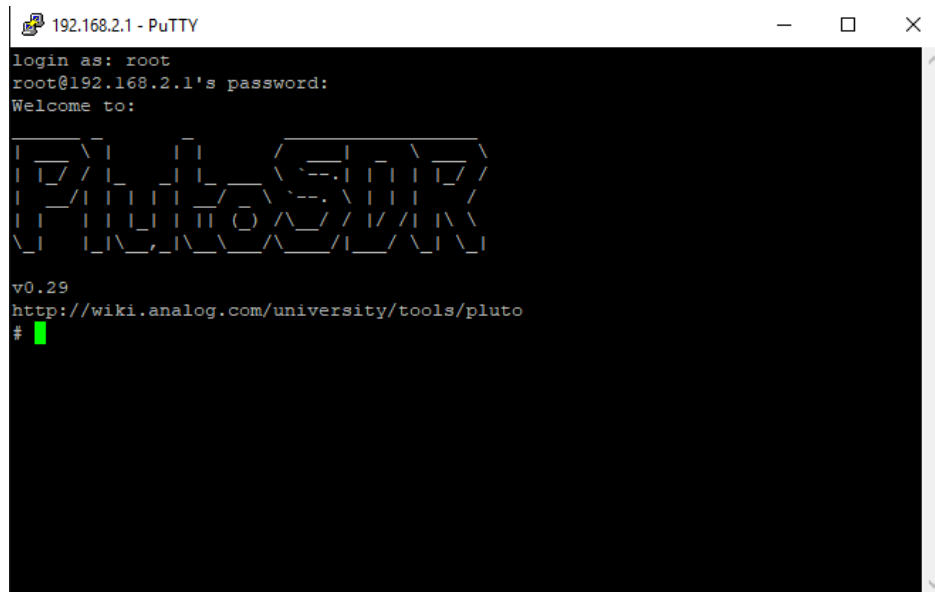
Example scripts for interacting with the PlutoSDR using shell scripts, C/C++, and Python are located at: https://github.com/analogdevicesinc/plutosdr_scripts.

3 Getting Started with PlutoSDR

Once the USB drivers are installed, a drive is mounted upon connecting the PlutoSDR to a USB port; similar to when you connect a thumb drive. The PlutoSDR is capable of auto-executing a properly named script that is placed in this directory. This can be handy for projects requiring standalone applications.

Browse the newly mounted PlutoSDR drive on your system and open the `info.html` file in your web browser. This file provides links to documentation and details about your PlutoSDR.

Find the IP address for your PlutoSDR, use this address to SSH into the embedded Linux operating system located on the PlutoSDR. The default SSH username is “root” and default password is “analog”. An example session using Putty is shown in Figure 4.



```
192.168.2.1 - PuTTY
login as: root
root@192.168.2.1's password:
Welcome to:

PlutoSDR

v0.29
http://wiki.analog.com/university/tools/pluto
#
```

Figure 4: Connecting to the PlutoSDR SSH server using Putty.

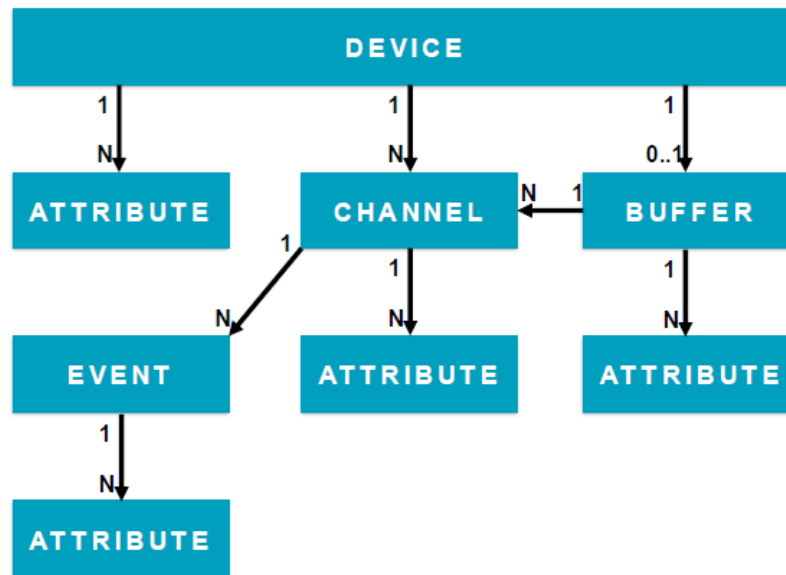


Figure 5: IIO Device Attributes. All hardware parameters represented as directories in sysfs.

3.1 Industrial Input / Output (IIO)

The Linux kernel Industrial Input / Output (IIO) framework is used by Analog Devices as a low-level, low-latency interface that allows access to hardware. IIO provides hardware abstraction layer that allows users to develop with the SDR without needing to develop kernel drivers as well.

Device attributes may be set through IIO. GNU Radio interfaces directly with these attributes and recently supports attribute blocks as part of `gr-iio`. The structure for the IIO device attributes provided by sysfs is outline in Figure 5

The “`iio_info -s`” command is very helpful for finding your current PlutoSDR device Universal Resource Identifier (URI). In the example below, the URI is `usb:3.6.5`. This value can be used to point to GNU Radio or MATLAB to your current hardware address if it can not auto-find it, or if you want to connect multiple PlutoSDR devices to the same PC.

```

$ iio_info -s

Library version: 0.15 (git tag: 2b4503d)
Compiled with backends: local xml ip usb serial
Available contexts:
    0: 0456:b673 (Analog Devices Inc. PlutoSDR (ADALM-PLUTO)),
       serial=c1d3a773be696cff61bc37f517ce03aa [usb:3.6.5]

```

Execute the command above on both the PlutoSDR using (A) an SSH terminal session and (B) from the local PC. What differences did you notice regarding the URI?

Another useful command line tool is “iio_attr”:

```
$ iio_attr -h
Usage:
    iio_attr [OPTION]...
                -d [device] [attr] [value]
                -c [device] [channel] [attr] [value]
                -B [device] [attr] [value]
                -D [device] [attr] [value]
                -C [attr]

Options:
    -h, --help           : Show this help and quit.
    -I, --ignore-case    : Ignore case distinctions.
    -q, --quiet          : Return result only.
    -a, --auto           : Use the first context found.

Optional qualifiers:
    -u, --uri            : Use the context at the provided URI.
    -i, --input-channel  : Filter Input Channels only.
    -o, --output-channel : Filter Output Channels only.

Attribute types:
    -s, --scan-channel   : Filter Scan Channels only.
    -d, --device-attr    : Read/Write device attributes
    -c, --channel-attr   : Read/Write channel attributes.
    -C, --context-attr   : Read IIO context attributes.
    -B, --buffer-attr    : Read/Write buffer attributes.
    -D, --debug-attr     : Read/Write debug attributes.
```

Using the `iio_attr` command, find the *ad9361-phy* device. Describe which portion of the PlutoSDR this device controls? On the SSH session, browse to `/sys/bus/iio/devices/` to find the same device. Change to this sub directory and cat name. Is this the *ad9361-phy* device? What parameters are available for configuration?

```
$ cat /sys/bus/iio/devices/iio:device0/name
```

This is an example of reading an attribute. You may also set an attribute from the command line using `echo` and redirecting the output. These devices and attributes are also readable and writeable using IIO Oscilloscope; making it an excellent debugging utility.

4 Radio Setup and Environmental Noise Observations

In these laboratory experiments an Analog Devices ADALM-PLUTO SDR with AD9363 transceiver will be used. This SDR in the most basic sense is a System-on-Chip (SoC) with an attached RF module providing complex baseband data. The MATLAB interface utilizes the libiio kernel driver to talk with the SDR through a MATLAB class called `iio_sys_obj_matlab`. The two system objects provided in the hardware support package for PlutoSDR are:

- `comm.SDRRxPluto`: PlutoSDR Receiver System object
- `comm.SDRTxPluto`: PlutoSDR Transmitter System object

These objects are typically constructed through the `sdr_rx` or `sdr_tx` function calls as in:

```
1 rx = sdr_rx('Pluto')
2 tx = sdr_tx('Pluto')
```

However, these objects can also be directly instantiated directly. The resulting object of `sdr_rx` either way will have the following basic properties, which will be directly printed to the terminal when not using the semicolon as:

```
1 rx = sdr_rx('Pluto')
2 rx =
3     comm.SDRRxPluto with properties:
4     DeviceName: 'Pluto'
5     RadioID: 'usb:0'
6     CenterFrequency: 2.4000e+09
7     GainSource: 'AGC Slow Attack'
8     ChannelMapping: 1
9     BasebandSampleRate: 1000000
10    OutputDataType: 'int16'
11    SamplesPerFrame: 3660
12    ShowAdvancedProperties: false
```

The transmitter System object `comm.SDRTxPluto` has near identical properties except for *GainSource*, *SamplesPerFrame*, and *OutputDataType* which do not make sense in the transmitter context. If you want to examine the available parameters simply type **`doc comm.SDRRxPluto`** or **`doc comm.SDRTxPluto`** into the MATLAB command prompt.

Before moving further with the radio it is important to outline how the radio works from the perspective of MATLAB using the `sdr_rx` and `sdr_tx` objects. After an object associated to the SDR has been created, the necessary methods can be run.

When configured in receive mode and the received method is called, with help from the driver, temporary buffers are created of size *SamplesPerFrame*, as set by the class parameter. Then the device will proceed to fill these buffer with contiguous data from the ADCs.

For each receive chain there are technically two ADCs, one for the in-phase portion of the signal and one for the quadrature. This is the reasoning behind the multiple buffers. However, these ADCs are time aligned and sampling of the dual chains happens at the same instances in time. Once the buffers are filled that data is provided to MATLAB and the link between the device and MATLAB is halted. As a result, when the receive method is called again from MATLAB the resulting buffer will be disjoint in time from the preceding buffer. For the overall structure of this communication Analog Devices provides a block diagram presented in Figure 6. The `PlutoSDR` class for convenience provides the output in a single complex vector, and accepts complex vectors as well.

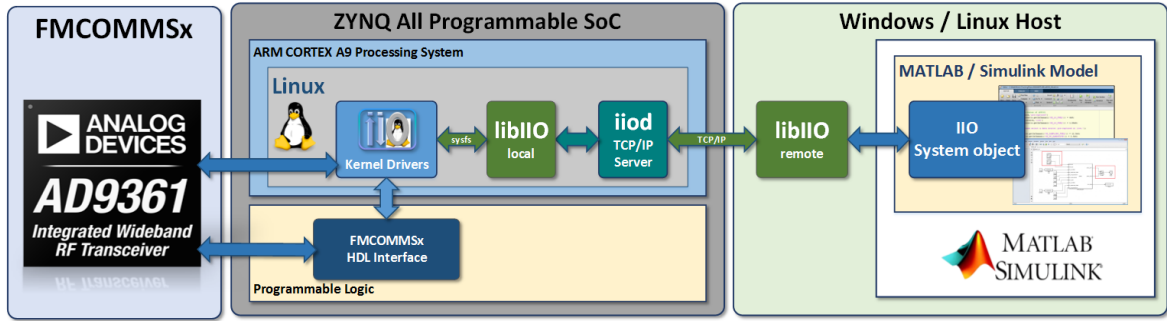


Figure 6: Structure of libiio and SDR hardware interconnection with MATLAB software [6].

Within the FMCOMMS/AD9361 direct downconversion is used to translate signals from RF to baseband frequencies. This is different than traditional heterodyne designs which utilize intermediate frequencies. Therefore, all observed signals passed to MATLAB are at baseband which were originally centered around *CenterFrequency* with a bandwidth of *BasebandSampleRate* parameterized by the `sdrrx` object.

4.1 Signal Loopback

Now that we have a basic understanding of how the radio operates and have it setup we can perform some simple experiments. For the loopback experiment, you should use the coaxial SMA cable provided with the PlutoSDR; not the provided antennas.

4.1.1 MATLAB Loopback

First we will run a simple loopback test which transmits a sinusoidal tone out the transmitter and is simultaneously received at the receiver. To perform this we will utilize the provided `loopback.m` script. This script collects multiple buffers of data, which can be necessary since there is an unknown startup lag for the transmitter and the desired signal can be missed. This script should generate a plot similar to the one shown in Figure 7.

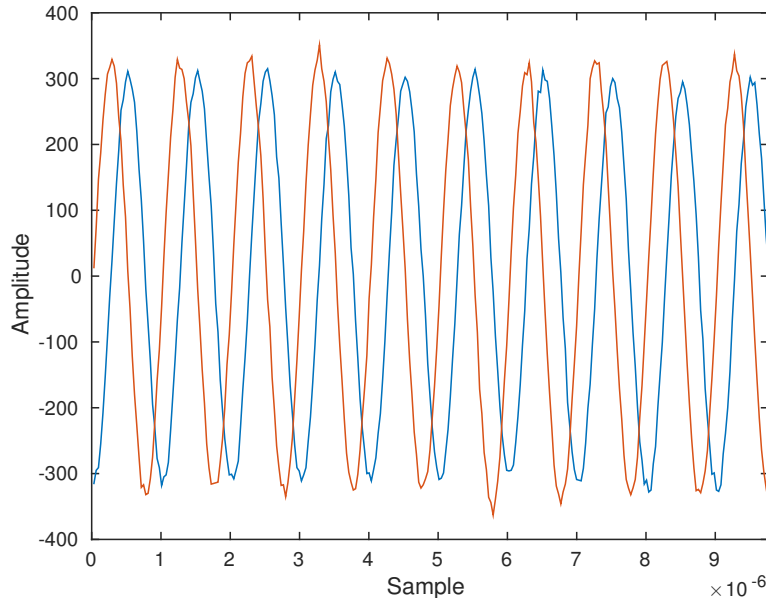


Figure 7: Looped back sinusoid observed from the Pluto receiver.

1. Modify the amplitude of the sinusoid, along with the PLUTO's *GainSource* and observe their effects.
 - (a) The three *GainSource* (AGC) settings are: *Manual*, *AGC Slow Attack*, and *AGC Fast Attack*.
 - (b) When *Manual* is selected another option called *Gain* becomes available and this parameter can be modified.
 - (c) It may be useful to use signals with varying amplitudes or zeros.
 - (d) If enough gain is applied the sinusoid should appear as a square wave at the receiver. What gain value does this occur?

4.1.2 GNU Radio Loopback

Now that MATLAB has been used for a loopback test, let's attempt the same test with GNU Radio to verify both tools are functioning properly.

To perform this test, we can utilize the `PlutoTestSine.grc` flowgraph in GNU Radio Companion, as shown in Figure 8. This flowgraph is configured to simultaneously transmit and receive in the 2.4 GHz band with a sample rate of 2.084 MHz. The “QT GUI Range” variable source frequency and RX gain at run-time.

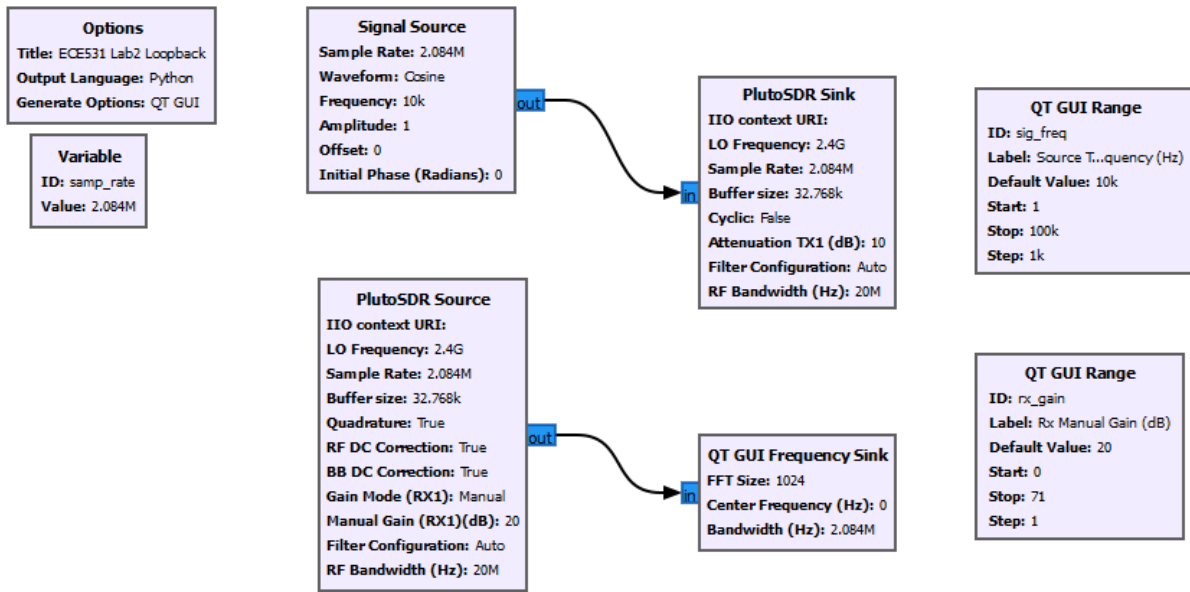


Figure 8: GRC flowgraph for Lab 2, Section 4.1.2

1. Double-click on each PlutoSDR block (source and sink) to look at the block properties.
 - (a) What is the RF Bandwidth? What does this property control in the PlutoSDR?
 - (b) What is the “Cyclic” boolean selection for in the PlutoSDR Sink block?
 - (c) What does the Manual Gain control in the PlutoSDR source? What other strategies are available?
2. Run the flowgraph. Note: you may have to enter change the Device URI. On some operating systems libIIO finds the first available device when this field is left blank.
3. Adjust the RX gain. At what value does the received signal begin to distort or clip?
4. Replace the “QT Time Sink” block with a “QT GUI Sink”. Explore the options provided by the new sink block. What happens if you increase the number of averages? Why does the frequency response change when you change the window function?
5. What is the transmitted RF frequency of the sinusoid? Why?

4.1.3 GNU Radio as a libIIO client

The PlutoSDR source and sink blocks contained in the Industrial IO module provide a convenient way to interface with PlutoSDR hardware. Generic blocks are also available, and can be useful when interfacing with generic SDR hardware compatible with IIO. The specific values necessary for the IIO Device Source/Sink and IIO Attribute blocks were found previously using IIO command line tools, such as `iio_attr`. The default values for these blocks are shown in Figure 9. A quick reference on using these tools will be posted to D2L.

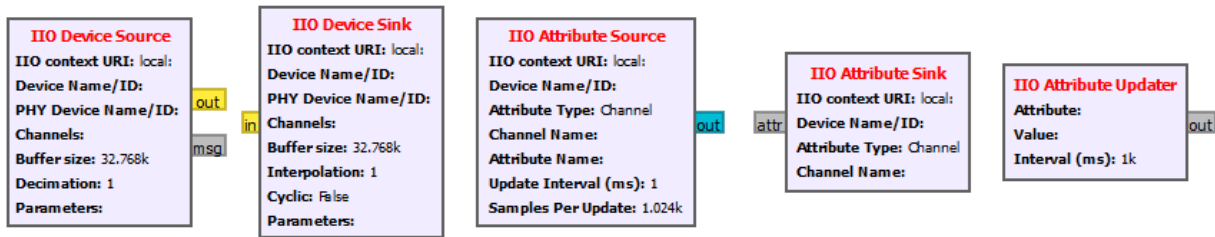


Figure 9: Default values for IIO Device blocks to be used for Section 4.1.3.

1. Repeat the loopback experiment from Section 4.1.2 without using the PlutoSDR source blocks; use only the “IIO Device Source” and “IIO Attribute Source / Sink / Updater”. Optionally, also replace the PlutoSDR sink blocks with the “IIO Device Sink”.
2. Change the TX and RX center frequency from 2.4 GHz to 915 MHz.
3. Increase the sample rate from 2.084 MSPS to a greater value of your choosing.
4. Verify each of the above have been set in the hardware using IIO Scope plugins or `iio_attr`.

4.2 Measurements and the Radio

What a SDR device like the PLUTO does is give you digital samples. What these are is nothing more or less than what the ADC makes out of the voltages it observes. Then, those numbers are subject to the receiver processing chain which includes frequency translation (Mixing), decimation and filtering. Analog Devices provides a great overview here [7] for the *ad936x*. Altogether, the complex signal’s envelope coming from the PLUTO should be proportional to the voltages observed by the ADC. Therefore, the magnitude square of these samples should be proportional to the signal power as seen by the ADC. However, it must be stress that these are in relation to the range of the ADC. Therefore, these values are of an arbitrary measure relative to the full scale of the ADC and the remaining receive chain, commonly denoted as dBFS [8]. Scopes provided by MATLAB may denote the signal amplitude in dB but this is just an arbitrary engineering unit.

With this knowledge we cannot directly determine power of an input signal to the SDR, unless we have performed some calibration and can relate individual samples to received energy. However, SNR can still be determined since signal and noise we be subjected to the same ADC effects and receive chain. This can be done with techniques already presented previously. For Figure 10 we

again used the transceiver setup, but transmitted signal vectors which contain half zeros and half signal.

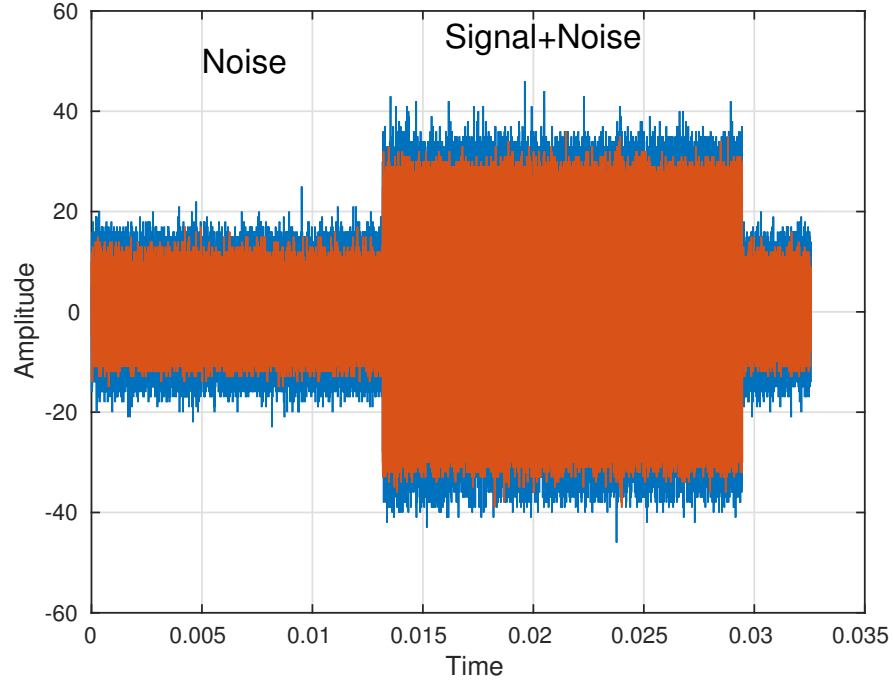


Figure 10: Receive loopback signal with signal half nulled.

After extracting the noise and signal plus noise sections the SNR can be manually calculated from the respective power signals by:

$$SNR_{dB} = 10 \log_{10} \left(\frac{P_{SN} - P_N}{P_N} \right), \quad (1)$$

where P_{SN} is the discrete-time power of signal + noise and P_N is the power of the noise. Note: the noise power may be comparatively low-level in a loopback configuration.

This is a rough method for estimating SNR, but does provides a reasonable metric without additional equipment. To provide better results we would need to remove some of the adaption performed in the receiver conditioning the samples.

1. Estimate the SNR using the method described above.
2. Repeat the process for different GainSource settings and different source amplitudes by modifying the supplied `loopback.m` script. Comment on the changes in SNR.

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.
- Conclusions to the overall lab that discuss meaningful lessons-learned and other take-aways from the assignment.
- Upload source files with report submission. You may also list select code source in your report appendix. Note: Python files autogenerated from GNURadio do not need to be uploaded in addition to .grc files.

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] Analog Devices. (2018) Adalm-pluto overview. [Online]. Available: <https://wiki.analog.com/university/tools/pluto>
- [2] ———. (2018) Customizing the Pluto configuration : Updating to the AD9364. [Online]. Available: https://wiki.analog.com/university/tools/pluto/users/customizing#updating_to_the_ad9364
- [3] The Mathworks. ADALM-PLUTO Radio Support from Communications Toolbox. [Online]. Available: <https://www.mathworks.com/hardware-support/adalm-pluto-radio.html>
- [4] ———. RTL-SDR Support from Communications Toolbox. [Online]. Available: <https://www.mathworks.com/hardware-support/rtl-sdr.html>
- [5] Analog Devices, “GitHub: IIO blocks for GNU Radio.” [Online]. Available: <https://github.com/analogdevicesinc/gr-iio>
- [6] ———, “IIO System Object,” [Online]: https://wiki.analog.com/_detail/resources/tools-software/linux-software/libiio/clients/sys_obj.png?id=resources
- [7] ———, “AD9361, AD9364 and AD9363,” [Online]: <https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/ad9361>.
- [8] Marcus Müller, “How to Calculate power spectral density using USRP data?” [Online]: <http://stackoverflow.com/questions/40523362/how-to-calculate-power-spectral-density-using-usrp-data>.