GETTING STARTED GUIDE

NI RF Data Recording API v0.4

This document provides basic information about how to get started with the NI RF Data Recording API.

Contents

System Requirements 2
Software 2
Hardware 2
Building and Installing UHD Python API 4
Pre-requirements 4
Install Python 4
Un-install pre-installed UHD 4
Install other packages 5
UHD Python API Installation 5
UHD Installation Test 7
Connect the USRP 7
USRP Probe 8
Load FPGA Images onto the Device 8
FPGA Image Flavors 8
Downloading the UHD FPGA Images 8
Load the Images onto the On-board Flash 9
Increase Performance of Interface 9
To know the network interfaces, run 9
UHD Python API – Test 11
Benchmarking USRP Speed in Python 11
RF Data Recording API Support Packages 12
SigMF Installation 12
npTDMS to Read TDMS Files 12
Change the Colour of your Linux Terminal 12
Setup Networking 13
Setup the host interface 13
Multiple devices per host 13
Change the USRP's IP address 13
Network setup example 14
Configuration for USRP-X Series device 0: 14

Configuration for USRP-X Series device 1:....14
Hardware Setup....15
Configuring the USRPs system: Cabled RF....15
Components of the RF Data Recording API....15
Running the RF Data Recording API Components....15
RF Replay Data Transmitter....15
Description of Controls....16
RF RX Data Recorder....16
Description of Controls....16
RF Data Recording API....17
Description of Variation Parameters....17
Description of General Configuration Parameters....17
Appendix....22
Waveform Metadata fields description....19
Notes on Config file: Describe the API operation modes....19
Troubleshooting....22
Known Issues....22
Related Information....22

System Requirements

Software

Ubuntu 20.04 (used in our test environment)

Note: The user can build UHD on different operating systems. See: [Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux - Ettus Knowledge Base]

Hardware

To use the NI RF Data Recording API, you need at least two NI RF USRP devices (tested on X310 only). The devices should be connected to single or different host computers based on the operation mode. Figure 1 shows the setup of two TX stations and RX Station. Table 1 presents the required hardware for this configuration.

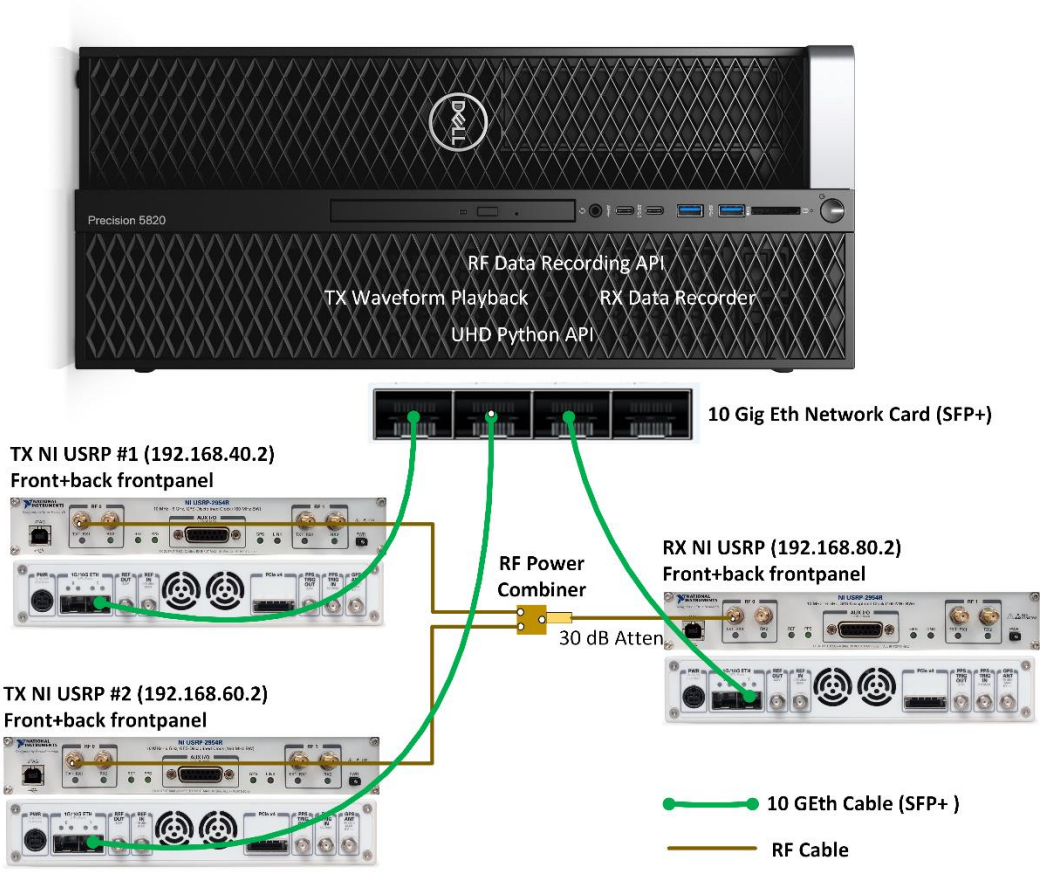

Figure 1 Hardware Configuration

Table 1 Required Hardware Accessories (cabled setup)

Item	Number
Host PC (Linux Server)	1
X310 USRP Device	3
SMA Cable	3
30 dB Attenuator	1
10 Gig Eth Cable (SFP+)	3

- Host PC Linux server: Recommended—with 10Gig Eth card (SFP+).
- SMA cable: Female/female cable that is included with the USRP RIO device.
- USRP RIO device: USRP-X310, USRP-2940/2942/2943/2944/2950/2952/2953/2954 Software Defined Radio Reconfigurable Devices with 40 MHz, 120 MHz, or 160 MHz bandwidth.
- Attenuator with 30 dB attenuation and male/female SMA connectors that are included with the USRP RIO device.

Ensure your host has enough free disk space and RAM.

Caution: Before using your hardware, read all product documentation to ensure compliance with safety, EMC, and environmental regulations.

Caution: To ensure the specified EMC performance, operate the RF devices only with shielded cables and accessories.

Caution: To ensure the specified EMC performance, the length of all I/O cables except for those connected to the GPS antenna input of the USRP device must be no longer than 3 m (10 ft.).

Caution: The USRP RIO RF devices are not approved or licensed for transmission over the air using an antenna. As a result, operating this product with an antenna may violate local laws. Ensure that you are in compliance with all local laws before operating this product with an antenna.

Building and Installing UHD Python API

The following instructions are based on Ubuntu 20.04 OS. To support different OS, look to the links below.

- Building UHD: Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux - Ettus Knowledge Base
- UHD Python API: USRP Hardware Driver and USRP Manual: Python API (ettus.com)

Pre-requirements

Install Python

Note: The UHD version should be at least 4.2. To be able to build UHD version 4.xx based on the current UHD source code, the CMake requires at least Python 3.6. The Ubuntu 20.04 uses Python version 3.8. Although, the getting started guide will be based on Python 3.9, but it is not required.

- To install Python, assume 3.9.10 (used in our test environment), run the following commands on the terminal:

```
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.9
python3.9 --version
python
```

- Make Python 3.9 your default:

```
sudo update-alternatives --set python /usr/bin/python3.9 alias
python='/usr/bin/python3.9'
```

- Export the path to Python. For example, go to: /home/user/.bashrc, then add the following line:

```
export PYTHONPATH=/usr/local/lib/python3.9/site-packages/
```

Un-install pre-installed UHD

- If UHD is already installed, check the UHD version by issuing UHD commands on terminal, for example

```
Uhd_find_devices
```

- Uninstall UHD if it is already installed on the system and its version is less than 4.0. For example, if the old UHD is UHD_3.15.0:

```
sudo apt-get remove libuhd3.15.0
```

- If you installed UHD manually, apt cannot know about this. To uninstall it, go to the UHD build directory and run:

```
 sudo make uninstall
```

Install other packages

- pandas, sympy, pip. For example, to install pip:

```
sudo apt install python3-pip
```

UHD Python API Installation

- Currently, the Python API is part of Master branch (EttusResearch/uhd: The USRP™ Hardware Driver Repository (github.com)), but not part of the release. You need to build UHD binary to get the UHD Python API.
- To install the Python API when building UHD from source, make sure you have the CMake variable ENABLE_PYTHON_API set to ON (e.g., by running CMake -DENABLE_PYTHON_API=ON). UHD requires Python header files in order to compile the Python API. Installing the Python API is described here: https://files.ettus.com/manual/page_python.html
- For building and installing UHD, follow the instructions given in the following link: Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux - Ettus Knowledge Base. The commands summary is listed below.
- To install any missing package, assume numpy is missing, use the following command where Python version should be specified

```
sudo python3.9 -m pip install numpy
```

Commands Summary:

- On Ubuntu 20.04 systems, run:

```
   sudo apt-get -y install autoconf automake build-essential ccache cmake cpufrequtils
doxygen ethtool fort77 g++ gir1.2-gtk-3.0 git gobject-introspection gpsd gpsd-clients
inetutils-tools libasound2-dev libboost-all-dev libcomedi-dev libcppunit-dev libfftw3-bin
libfftw3-dev libfftw3-doc libfontconfig1-dev libgmp-dev libgps-dev libgsl-dev
liblog4cpp5-dev libncurses5 libncurses5-dev libpulse-dev libqt5opengl5-dev libqwt-qt5-dev
libsdl1.2-dev libtool libudev-dev libusb-1.0-0 libusb-1.0-0-dev libusb-dev libxi-dev
libxrender-dev libzmq3-dev libzmq5 ncurses-bin python3-cheetah python3-click python3-
click-plugins python3-click-threading python3-dev python3-docutils python3-gi python3-gi-
cairo python3-gps python3-lxml python3-mako python3-numpy python3-numpy-dbg python3-
opengl python3-pyqt5 python3-requests python3-scipy python3-setuptools python3-six
python3-sphinx python3-yaml python3-zmq python3-ruamel.yaml swig wget
```

- To build UHD from source code, clone the GitHub repository, check out a branch or tagged release of the repository, and build and install. Please follow the steps below. Make sure that no USRP device is connected to the system at this point.
- First, make a folder to hold the repository.

```
   cd $HOME
   mkdir workarea
   cd workarea
```

- Next, clone the repository and change into the cloned directory.

```
git clone https://github.com/EttusResearch/uhd
cd uhd
```

- Next, checkout the desired UHD version. You can get a full listing of tagged releases by running the command:

```
git tag -l
```

- Next, create a build folder within the repository.

```
cd host
mkdir build
cd build
```

- Next, invoke CMake.

```
cmake ..
```

- If CMake can't find the Python headers or library, specify the `PYTHON_INCLUDE_DIR` and / or `PYTHON_LIBRARY` CMake variables manually.

```
cmake -DPYTHON_INCLUDE_DIR=/usr/include/python3.9 -DPYTHON_LIBRARY=/usr/lib/x86_64-linux-gnu/libpython3.9.so ..
```

Note: If the CMake is not successful, look to: [Buipythonlding and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux - Ettus Knowledge Base]

- The output from CMake should be something like the picture below:

```
- Utilizing the python install directory: /usr/local/lib/python3.8/site-packages
-
- ################################################
- # UHD enabled components
- ################################################
-   * LibUHD
-   * LibUHD - C API
-   * LibUHD - Python API
-   * Examples
-   * Utils
-   * Tests
-   * USB
-   * B100
-   * B200
-   * USRP1
-   * USRP2
-   * X300
-   * MPMD
-   * SIM
-   * N300
-   * N320
-   * E320
-   * E300
-   * X400
-   * OctoClock
-   * Manual
-   * API/Doxygen
-   * Man Pages
-
- ################################################
- # UHD disabled components
- ################################################
-   * DPDK
```

Figure 2 Output of CMake Command: UHD Python API is enabled

- Once the CMake command succeeds without errors, build UHD.

```
make
```

- Next, you can optionally run some basic tests to verify that the build process completed properly.

```
make test
```

- Next, install UHD, using the default install prefix, which will install UHD under the /usr/local/lib folder. You need to run this as root due to the permissions on that folder.

```
sudo make install
```

- Next, update the system's shared library cache.

```
sudo ldconfig
```

- Finally, make sure that the `LD_LIBRARY_PATH` environment variable is defined and includes the folder under which UHD was installed. Most commonly, you can add the line below to the end of your `$HOME/.bashrc` file:

```
export LD_LIBRARY_PATH=/usr/local/lib
```

- If the `LD_LIBRARY_PATH` environment variable is already defined with other folders in your `$HOME/.bashrc` file, then add the line below to the end of your `$HOME/.bashrc` file to preserve the current settings.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

- For this change to take effect, you will need to close the current terminal window, and open a new terminal.

UHD Installation Test

At this point, UHD should be installed and ready to use. You can quickly test this, with no USRP device attached, by running `uhd_find_devices`. You should see something similar to the following.

```
[INFO] [UHD] linux; GNU C++ version 9.4.0; Boost_107100; UHD_4.2.0.git-413-g159c21b7#
```

Connect the USRP

The installation of UHD should now be complete. At this point, connect the USRP to the host computer.

- Look to **Setup Networking Section** in this document for more details about how to setup the network.
- Look to X300/X310 - Ettus Knowledge Base for more information about choosing a host interface for X300/X310 USRPs.
- Look to USRP Hardware Driver and USRP Manual: USRP X3x0 Series (ettus.com) for more information about setup networking.

If the interface is Ethernet, then open a terminal window, and try to ping the USRP with "ping 192.168.10.2". The USRP should respond to the ping requests. Also try running "uhd_find_devices", the output should be similar to the following:

```
agaber@dre-elbe-s05:~/workarea$ uhd_find_devices
[INFO] [UHD] linux; GNU C++ version 9.3.0; Boost_107100; UHD_4.2.0.git-290-g2c7ce2db
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    serial: 30F1628
    addr: 192.168.40.2
    fpga: HG
    name:
    product: X310
    type: x300
```

USRP Probe

To check if the UHD image of the connected USRP is *compatible* with the installed UHD, run the following command:

```
uhd_usrp_probe --args addr=192.168.10.2
```

If the output of `uhd_usrp_probe` does not show any warning, you do not need to update Image. However, if there are errors regarding the FPGA version compatibility number, you will have to update the FPGA image before you can start using your USRP.

Load FPGA Images onto the Device

For more information, look to: **USRP Hardware Driver and USRP Manual: USRP X3x0 Series (ettus.com)**

FPGA Image Flavors

The USRP-X Series devices contains two SFP+ ports for the two Ethernet channels. Because the SFP+ ports support both 1 Gigabit (SFP) and 10 Gigabit (SFP+) transceivers, several FPGA images are shipped with UHD to determine the behavior of the above interfaces.

Table 2 UHD Images and related SFP+ interface

FPGA Image Flavor	SFP+ Port 0 Interface	SFP+ Port 1 Interface
HG (Default)	1 Gigabit Ethernet	10 Gigabit Ethernet
XG	10 Gigabit Ethernet	10 Gigabit Ethernet
HA	1 Gigabit Ethernet	Aurora
XA	10 Gigabit Ethernet	Aurora

Note: The Aurora images need to be built manually from the FPGA source code.

FPGA images are shipped in 2 formats:

- **LVBITX**: LabVIEW FPGA configuration bitstream format (for use over PCI Express and Ethernet)
- **BIT**: Xilinx configuration bitstream format (for use over Ethernet and JTAG)

Downloading the UHD FPGA Images

To get the latest images, simply use the uhd_images_downloader script (see Firmware and FPGA Images). On Unix systems, use this command:

```
$ [sudo] uhd_images_downloader
```

Note: It is recommended to download the HG image, you will be able to use both the 1 Gig Eth and the 10 Gig Eth connection.

Load the Images onto the On-board Flash

To change the FPGA image stored in the on-board flash, the USRP-X Series device can be reprogrammed over the network or PCI Express. Once you have programmed an image into the flash, that image will be automatically loaded on the FPGA during the device boot-up sequence.

Note: Different hardware revisions require different FPGA images. Determine the revision number from the sticker on the rear of the device. If you are manually specifying an FPGA path, the utility will not try to detect your device information, and you will need to use this number to select which image to burn.

Note: The image loader utility will default to using the appropriate BIT file if no custom FPGA image path is specified, but it is compatible with BIN, BIT, and LVBITX images.

Use the `uhd_image_loader` utility to load the image from the default path and update the FPGA image. On the command line, run:

```
uhd_image_loader --args="type=x300,addr=<IP address>"
```

- To specify the image type, run the following command:

```
uhd_image_loader --args="type=x300,addr=192.168.40.2,fpga=HG"
```

- To load the FPGA image from a given path, run the following command:

```
uhd_image_loader --args="type=x300,addr=192.168.40.2, --fpga-
path=”your_path\usrp_x310_fpga_HG.bit”
```

- Wait a few minutes until it is successful, then restart the USRP.
- To check if the UHD image of the connected USRP is compatible with the installed UHD, run the following command:

```
uhd_usrp_probe --args addr=192.168.10.2
```

It should show the USRPs specifications without any error.

Increase Performance of Interface

To know the network interfaces, run `ifconfig` command. The output looks like the following:

```
enp3s0f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet 192.168.40.1  netmask 255.255.255.0  broadcast 192.168.40.255
        inet6 fe80::b1e:9f17:2b67:6030  prefixlen 64  scopeid 0x20<link>
        ether 90:e2:ba:d1:c8:28  txqueuelen 1000  (Ethernet)
        RX packets 2228470  bytes 175930400 (175.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2115340  bytes 419244814 (419.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp3s0f1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet 192.168.60.1  netmask 255.255.255.0  broadcast 192.168.60.255
        inet6 fe80::ada4:5fc3:9320:4628  prefixlen 64  scopeid 0x20<link>
        ether 90:e2:ba:d1:c8:29  txqueuelen 1000  (Ethernet)
        RX packets 1627865  bytes 122994502 (122.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1521434  bytes 315288179 (315.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp7s0f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet 192.168.80.1  netmask 255.255.255.0  broadcast 192.168.80.255
        inet6 fe80::5c37:52b0:6069:26de  prefixlen 64  scopeid 0x20<link>
        ether 3c:fd:fe:9f:d5:c0  txqueuelen 1000  (Ethernet)
        RX packets 2240387  bytes 5245791862 (5.2 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1523059  bytes 125786867 (125.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 3 Output of ifconfig command

- Set the MTU size of all Eth ports connected to USRPs to 9000 to get maximum sampling rate. Assume, we need to configure the network interface of "enp3s0f0".:

```
sudo ip link set enp3s0f0 mtu 9000
```

- Resize buffer to be more efficient. To check the default and maximum amount for the receive socket memory, run the following commands:

```
$ cat /proc/sys/net/core/rmem_default
$ cat /proc/sys/net/core/rmem_max
```

- To check the default and maximum amount for the send socket memory, run the following commands:

```
$ cat /proc/sys/net/core/wmem_default
$ cat /proc/sys/net/core/wmem_max
```

- To Set them, use the following Commands:

```
sudo sysctl -w net.core.wmem_max=24912805
sudo sysctl -w net.core.rmem_max=24912805
```

- To check if the value has been changed:

```
$ sudo sysctl net.core.rmem_max
$ sudo sysctl net.core.wmem_max
```

- To make the change permanent, add the following lines to the /etc/sysctl.conf file, which is used during the boot process:

```
net.core.rmem_default=24912805
net.core.wmem_default=24912805
net.core.rmem_max=24912805
```

```
net.core.wmem_max=24912805
```

For more information, look to:

USRP Hardware Driver and USRP Manual: Transport Notes (ettus.com)

UHD Python API – Test

Once it's built and installed, you'll be able to import the UHD Python module. The Python has an object called MultiUSRP which is an equivalent of the C++ multi_usrp API. The methods on both classes are the same and take the same arguments.

- On the terminal, run the following commands:

```
python3.9
import uhd
usrp = uhd.usrp.MultiUSRP("addr=192.168.40.2")
samples = usrp.recv_num_samps(10000, 100e6, 1e6, [0], 50)
print(samples[0:10])
```

- If you able to import UHD and open the MultiUSRP session, then it is working fine.

Note: If you are not able to import UHD and open the MultiUSRP session, then the reason could be the following:

- The UHD Python API is not enabled during UHD build
- The UHD Python API created using another python version
- Or you need to export the path to python. For example, go to: /home/user/.bashrc, then add the following line:

```
export PYTHONPATH=/usr/local/lib/python3.9/site-packages/
```

Benchmarking USRP Speed in Python

The maximum sampling rate of UHD Python API is 100 MS/s. You can run this command on terminal:

```
python3.9 /usr/local/lib/uhd/examples/python/benchmark_rate.py --
args="type=x300,addr=192.168.40.2,num_recv_frames=1000" --rx_rate 100e6 --channels 0 --
rx_channels 0
```

The output on terminal looks like the following. The number of dropped samples is zero.

```
[INFO] [UHD] linux; GNU C++ version 9.3.0; Boost_107100; UHD_4.2.0.git-290-g2c7ce2db
[INFO] [X300] X300 initialization sequence...
[INFO] [X300] Maximum frame size: 8000 bytes.
[INFO] [GPS] Found an internal GPSDO: LC_XO, Firmware Rev 0.929a
[INFO] [X300] Radio 1x clock: 200 MHz
[WARNING] [0/Radio#0] Non-critical minor compat number mismatch for `0/Radio#0':
Expecting 0.1, got 0.0.
[WARNING] [0/Radio#1] Non-critical minor compat number mismatch for `0/Radio#1':
Expecting 0.1, got 0.0.[14:40:54.59]
 [INFO] (MainThread) Using Device: Single USRP:  Device: X-Series Device  Mboard 0: X310
RX Channel: 0    RX DSP: 0    RX Dboard: A    RX Subdev: UBX RX  RX Channel: 1    RX DSP:
```

```
1    RX Dboard: B    RX Subdev: UBX RX  TX Channel: 0    TX DSP: 0    TX Dboard: A    TX
Subdev: UBX TX  TX Channel: 1    TX DSP: 1    TX Dboard: B    TX Subdev: UBX TX

[14:40:54.59] [INFO] (MainThread) Selected [0] RX channels and no TX channels

[14:40:54.59] [INFO] (MainThread) Setting device timestamp to 0...

[14:40:54.60] [INFO] (Thread-1  ) Testing receive rate 100.000 Msps on 1 channels

[14:41:04.61] [DEBUG] (MainThread) Sending signal to stop!

[14:41:04.61] [DEBUG] (MainThread) RX Statistics Dictionary: {'num_rx_samps': 1000886216,
'num_rx_dropped': 0, 'num_rx_overruns': 0, 'num_rx_seqerr': 0, 'num_rx_timeouts': 0,
'num_rx_late': 0}

[14:41:04.61] [DEBUG] (MainThread) TX Statistics Dictionary: {}[

14:41:04.61] [DEBUG] (MainThread) TX Async Statistics Dictionary: {}[14:41:04.61] [INFO]
(MainThread) Benchmark rate summary:    Num received samples:     1000886216    Num
dropped samples:      0    Num overruns detected:    0    Num transmitted samples:  0
Num sequence errors (Tx): 0    Num sequence errors (Rx): 0    Num underruns detected:   0
Num late commands:        0    Num timeouts (Tx):        0    Num timeouts (Rx):        0
```

RF Data Recording API Support Packages

SigMF Installation

The Signal Metadata Format (SigMF) specifies a way to describe sets of recorded digital signal samples with metadata written in JSON. See SigMF/sigmf-spec.md at sigmf-v1.x · gnuradio/SigMF (github.com).

- Install sigmf for a selected python

```
sudo python3.9 -m pip install sigmf
```

- To run the included QA tests:

```
sudo python3.9 -m pip install pytest
```

npTDMS to Read TDMS Files

To create wireless standard complaint waveforms, the NI RFmx Waveform Creator, i.e. the NI RFmx 5G NR waveform creator. To read TDMS files, the npTDMS is used. The npTDMS is a cross-platform Python package for reading and writing TDMS files as produced by LabVIEW, and is built on top of the numpy package. For more information, see Welcome to npTDMS's documentation — npTDMS 1.3.1 documentation. To install npTDSM, run the following command:

```
sudo python3.9 -m pip install npTDMS
```

Note: There are optional features available that require additional dependencies. These are hdf for hdf export, pandas for pandas DataFrame export, and thermocouple_scaling for using thermocouple scalings. You can specify these extra features when installing npTDMS to also install the dependencies they require:

```
sudo python3.9 -m pip install npTDMS[hdf,pandas,thermocouple_scaling]
```

Change the Colour of your Linux Terminal

To make the output coloured on Python terminal, download those packages:

```
sudo python3.9 -m pip install colored
```

```
sudo python3.9 -m pip install termcolor
```

Setup Networking

The USRP-X Series only supports Gigabit and Ten Gigabit Ethernet and will not work with a 10/100 Mbps interface.

Please note that 10 Gigabit Ethernet defines the protocol, not necessary the medium. For example, you may use 10GigE over optical with optical SFP+ transceiver modules.

Note: For more info, see: USRP Hardware Driver and USRP Manual: USRP X3x0 Series (ettus.com)

Note: For more Info about supported host interfaces, see: X300/X310 - Ettus Knowledge Base

Setup the host interface

The USRP-X Series communicates at the IP/UDP layer over the Gigabit and Ten Gigabit Ethernet. The default IP address for the USRP X300/X310 device depends on the Ethernet Port and interface used. You must configure the host Ethernet interface with a static IP address on the same subnet as the connected device to enable communication, as shown in the following table:

Ethernet Interface	USRP Ethernet Port	Default USRP IP Address	Host Static IP Address	Host Static Subnet Mask	Address EEPROM key
Gigabit	Port 0 (HG Image)	192.168.10.2	192.168.10.1	255.255.255.0	`ip-addr0`
Ten Gigabit	Port 0 (XG Image)	192.168.30.2	192.168.30.1	255.255.255.0	`ip-addr2`
Ten Gigabit	Port 1 (HG/XG Image)	192.168.40.2	192.168.40.1	255.255.255.0	`ip-addr3`

As you can see, the X300/X310 actually stores different IP addresses, which all address the device differently: Each combination of Ethernet port and interface type (i.e., Gigabit or Ten Gigabit) has its own IP address. As an example, when addressing the device through 1 Gigabit Ethernet on its first port (Port 0), the relevant IP address is the one stored in the EEPROM with key `ip-addr0`, or 192.168.10.2 by default.

See **Configuring the host's IP address** on details how to change your machine's IP address and MTU size to work well with the X300.

Multiple devices per host

For maximum throughput, one Ethernet interface per USRP is recommended, although multiple devices may be connected via an Ethernet switch. In any case, each Ethernet interface should have its own subnet, and the corresponding USRP device should be assigned an address in that subnet.

Change the USRP's IP address

You may need to change the USRP's IP address for several reasons:

- to satisfy your particular network configuration
- to use multiple USRP-X Series devices on the same host computer
- to set a known IP address into USRP (in case you forgot)

To change the USRP's IP address, you must know the current address of the USRP, and the network must be setup properly as described above. You must also know which IP address of the X300 you want to change, as identified by their address EEPROM key (e.g. `ip-addr3`, see the table above). Run the following commands:

```
cd <install-path>/lib/uhd/utils
./usrp_burn_mb_eeprom --args="type=x300,addr=192.168.40.2" --values="ip-addr3=192.168.60.2"
```

Power cycle the USRP to get the new IP.

Network setup example

Figure 1 shows three X310 USRPs. Assume:

- The UHD image is HG
- Three 10 Gig Eth cables (SFP+) are available

Then, do the following:

- Connect the SFP+ Port 1 of each USRP to the Linux server. Figure 4 shows the rear panel.

Figure 4 X310 Rear Panel

To change the IP address of USRPs, do that one by one. Other USRPs should be off or leave the related Ethernet interfaces not configured or set the related Ethernet interfaces down. Since the default IP of USRPs is the same.

Configuration for USRP-X Series device 0:

The default ip-addr3 is 192.168.40.2. No need to change. So, the configuration will be:

- Ethernet interface IPv4 address: `192.168.40.1`
- Ethernet interface subnet mask: `255.255.255.0`
- USRP-X Series device IPv4 address: `192.168.40.2`

Configuration for USRP-X Series device 1:

First, switch off device 0 or set its Ethernet interface down. Then, use the following default config to be able to communicate with USRP:

- Ethernet interface IPv4 address: `192.168.40.1`
- Ethernet interface subnet mask: `255.255.255.0`
- USRP-X Series device IPv4 address: `192.168.40.2`

To change the IP address, i.e. to 192.168.50.2, run the following command:

```
./usrp_burn_mb_eeprom --args="type=x300,addr=192.168.40.2" --values="ip-addr3=192.168.50.2"
```

Then, Power cycle the USRP to get the new IP.

Change the IP on host to 192.168.50.1. To activate the new IP on the Host, set the interface to be Up and then Down. The configuration will be then:

- Ethernet interface IPv4 address: `192.168.50.1`
- Ethernet interface subnet mask: `255.255.255.0`
- USRP-X Series device IPv4 address: `192.168.50.2`

Hardware Setup

The following descriptions assume that there are three USRPs. They are referred to as TX USRP1, TX USRP2, and RX USRP as it is shown in Figure 1.

Configuring the USRPs system: Cabled RF

1. Ensure the USRP devices are properly connected to the host systems
2. Complete the following steps to create RF connections as shown in Figure 1.
    1. Connect RF0/TX1 ports on TX USRP1 and TX USRP2 to RF cables
    2. Connect the other ends of the two RF cables to RF Power splitter.
    3. Connect the common port of power splitter to 30 dB attenuator
    4. Connect the other end of the attenuator to RF cable
    5. Connect the RF cable to RF0/TX1 port on RX USRP
3. Power on the USRP devices.

The RF cables and power splitter should support the operating frequency.

Components of NI RF Data Recording API

The related components of the project are described in the following:

- **RF Replay Data Transmitter:** Python script
    - rf_replay_data_transmitter_usrp_uhd.py
- **RF RX Data Recorder:** Python script
    - rf_data_recorder_usrp_uhd.py
- **RF Data Recording API**: It has two main components:
    - Configuration file: config_rf_data_recording_api.json and config_rf_data_recording_api.yaml
    - Top-level script: main_rf_data_recording_api.py
- **Wireless Standard Waveforms:** The waveform-files folder has three sub folders:
    - **tdms:** Sever 5GNR and LTE standard compliant waveforms have been generated in advance using NI RFmx Waveform generator (RFmx NR - NI):
    - **matlab_ieee:** The IEEE waveform generator is used to generate a single waveform in MATLAB format.
    - **matlab:** The radar waveforms created by Northeastern University in MATLAB format and related configuration file in CSV format.

NI RF Data Recording API Components

RF Replay Data Transmitter

Assume you would like to replay the 5G NR waveform “NR_FR1_DL_FDD_SISO_BW-20MHz_CC-1_SCS-30kHz_Mod-64QAM_OFDM_TM3.1.tdms”, with the following parameters (frequency = 2 GHz, rate = 30.72 MS/s, gain = 30dB, USRP IP = 192.168.40.2). Run the following command:

```
python3.9 rf_replay_data_transmitter_usrp_uhd.py  --
args="type=x300,addr=192.168.40.2,master_clock_rate=184.32e6" --freq=2e9 --rate=30.72e6 -
-gain=30 --path="waveform-files/tdms/" --file="NR_FR1_DL_FDD_SISO_BW-20MHz_CC-1_SCS-
30kHz_Mod-64QAM_OFDM_TM3.1" --waveform_format="tdms"
```

To stop data transmission, click on the terminal ctrl+c.

Description of Controls

Application settings are applied when the script starts and cannot be changed once the application is up and running.

Table 3 Configuration list of RF Replay Data Transmitter

Parameter	default	Description
args	“type=x300,addr=192.168.40.2,master_clock_rate=184.32e6”	Device args to use when connecting to the USRP
radio_id	0	Radio block to use (e.g., 0 or 1).
radio_chan	0	Radio channel to use, for example 0 or 1 for X310 USRP
replay_id	0	Replay block to use (e.g., 0 or 1)
replay_chan	0	Replay channel to use
duc_chan	0	Duc channel to use
path	"waveform-files/tdms/”	Path to file - "waveform-files/tdms/” - "waveform-files/matlab_ieee/” - "waveform-files/matlab/”
file	"NR_FR1_DL_FDD_SISO_BW-20MHz_CC-1_SCS-30kHz_Mod-64QAM_OFDM_TM3.1.tdms"	Waveform file name or a folder name without extension. It could be: - TDMS file name without extension - Waveform folder name of a waveform in MATLAB format generated by IEEE waveform generator - MATLAB file name for arbitrary waveform
waveform_format	“tdms”	Possible values: “tdms”, “matlab”, “matlab_ieee"
freq	2.0e9	RF target center frequency in Hz
lo_offset	20.0e9	Tx LO offset in Hz
enable_lo_offset	True	Enable or disable LO offset: True or false
rate	30.72e6	Rate of radio block
gain	30	Gain for the RF chain in dB
antenna	"TX/RX"	Antenna port selection
bandwidth	20	Analog front-end filter bandwidth in Hz
reference	"internal"	Clock reference source (internal, external, gpsdo)

RF RX Data Recorder

Assume you would like to record the IQ data and save it in SigMF format to this path “/home/user/workarea/recorded-data”. The configuration parameters are (frequency = 2 GHz, rate = 30.72 MS/s, gain = 30dB, channels =0, duration 10 ms, number of records 1, USRP IP = 192.168.40.2). Run the following command:

```
python3.9 rf_data_recorder_usrp_uhd.py --nrecords 1 --
args="type=x300,addr=192.168.40.2,master_clock_rate=184.32e6" --freq 2e9 --rate 30.72e6 -
-duration 10e-3 --channels 0 --gain 30 --rx_recorded_data_path
/home/user/workarea/recorded-data
```

Since the RX is running independently from TX, the user needs to write some parameters in the meta-data manually. Go to the section of write data to files in the code to update meta-data.

Description of Controls

Application settings are applied when the script starts and cannot be changed once the application is up and running.

Table 4 Configuration list of RF RX Data Recorder

Parameter	default	Description
args	“type=x300,addr=192.168.40.2,master_clock_rate=184.32e6”	Device args to use when connecting to the USRP
rx_recorded_data_path	"../../../../../recorded-data"	Path to store received IQ data
freq	2.0e9	RF center frequency in Hz
rate	30.72e6	Rate of radio block
gain	30	Gain for the RF chain in dB
channel	0	Channel selection, for example 0 or 1 for X310 USRP
antenna	"TX/RX"	Antenna port selection (TX/RX, RX2)
duration	10.0e-3	Time duration of IQ data acquisition
reference	"internal"	Clock reference source (internal, external, gpsdo)
nrecords	2	Number of records

RF Data Recording API

To configure the RF Data recording API, the user need to change the configuration in the JSON file “config_rf_data_recording_api.json” or “config_rf_data_recording_api.yaml” and not from the terminal. The configuration file has three sections:

- **Possible values**: Description for every parameter (ONLY for JSON configuration file)
- **General Config:** List the basic parameters such as the paths of recorded data or the number of records.
- **Transmitters config:**
  - List of transmitters: Every transmitter has a list of parameters that you can sweep over. Three types of variations have been defined:
  - **Range**: The user needs to specify the start, stop, and step such as the following:

```
“range_Param":{"SeqType": „range", "Values": ["start", "stop", "step“]},
```

  - **List**: The user can list a set of parameters to sweep over such as the following´:

```
“List_Param":{"SeqType": „list", "Values": ["x1", "x2", "x3", … , "xn"]}
```

  - **Single**: The user can provide the single value of the parameter

```
"Single_Param”:{"SeqType": „single", "Values":”waveform_path”}
```

- **Common transmitters config:** A list of parameters that are related to transmitters for clock reference configuration and the replay data configuration.
- **Receivers config:**
  - List of receivers: Every Receiver has a list of parameters that you can sweep over. Three types of variations have been defined as it is mentioned above: “Range”, “List”, and “Single”.

Description of Variation Parameters

Description of General Configuration Parameters

Application settings are applied when the script starts and cannot be changed once the application is up and running.

Parameter	default	Description
rx_recorded_data_path	"/home/user/workarea/recorded-data"	Path to store captured rx data

rx_recorded_data_saving_format	SigMF	Only SigMF data format is supported
nrecords	5	Number of snapshots from RX IQ data acquisition
Txs_execution	“parallel”	Tx USRPs execution mode: "parallel" or "sequential" • parallel: TX USRPs will transmit their related waveforms simultaneously (in parallel). • sequential: TX USRPs will transmit their related waveforms one by one in sequential manner: Transmit wavefore1, record IQ data, Transmit waveform2, record IQ data and so on. Note: If there are more than one RX USRP, they will run in parallel.
waveform_config_file	"default_waveform_config.yaml"	Default Tx Waveform Config dictionary. This will be replaced in future releases by a dictionary that includes a list of parameters for each standard individually.
enable_console_logging	"True"	Enable or disable console logging: "True" or "False"
author	“NI”	Author name, handle, email, and/or other ID like Amateur Call Sign.
description	"CBRS use case"	A text description of the SigMF Recording.
comment	"Using NI RF Data Collection API"	User comment

Transmitters Config:

It has a list of sections. Every Tx station has a list of parameters in its section. To add more transmitters, add a section of parameters for every Tx station.

Parameter		**default**	**Description**
RFmode		“Tx”	“Tx”
type		“x300”	USRP type
IPaddress		“192.168.40.2"	The IP address. Note: The USRP type and IP are used to create the arg of TX USRP devices that we need to connect to
Parameters	freq	[2.0e9,5.0e9,1.0e9]	RF target center frequency in Hz
	Lo_offset	20.e9	LO offset in Hz
	enable_lo_offset	“False”	Enable LO offset “True” or “False”.
	rate	[30.72e6]	Rate of radio block It is used if the general config “rate_source” is set to “user_defined” otherwise the rate will be read from waveform configuration file.
	rate_source	“waveform_config”	“waveform_config”: The rate will be read from the waveform config file. "user_defined”: The given rate by the user in this config file will be used.
	bandwidth	[20.0e6]	TX Analog front-end filter bandwidth in Hz. • If rate_source = waveform_config: It will be read from the waveform config file. • If rate_source = user_defined: The given bandwidth by the user in this config file will be used.
	waveform_file_name	["NR_FR1_DL_FDD_SISO_BW-20MHz_CC-1_SCS-30kHz_Mod-64QAM_OFDM_TM3.1"]	TDMS file name without extension or MATLAB folder name Supported Waveform: • TDMS waveform formats • MATLAB waveform formats generated by IEEE waveform generator • Arbitrary MATLAB Waveform
	waveform_format	“tdms”	“tdms” or “matlab_ieee” or “matlab”
	waveform_path	"waveform-files/tdms/”	Path to tx waveform file or folder • "waveform-files/tdms/” • "waveform-files/matlab_ieee/” • "waveform-files/matlab/”
	gain	[30]	Gain for the RF chain in dB
	antenna	["TX/RX"]	Antenna selection (TX/RX)

Common Tx USRPs configuration:

Parameter	**default**	**Description**
tx_clock_reference	"internal"	Tx sync reference source (internal, external, gpsdo)
tx_radio_id	[0]	Radio block to use (e.g., 0 or 1).

tx_radio_chan	[0]	Radio channel to use
tx_replay_id	[0]	Replay block to use (e.g., 0 or 1)
tx_replay_chan	[0]	Replay channel to use
tx_duc_chan	[0]	Duc channel to use

RX USRP configuration:

Parameter		**default**	**Description**
RFmode		"Rx"	"Rx"
type		"x300"	USRP type
IPaddress		"192.168.40.2"	The IP address. Note: The USRP type and IP are used to create the arg of RX USRP devices that we need to connect to
Parameters	freq	[2.0e9,5.0e9,1.0e9]	RF target center frequency in Hz
	rate	[30.72e6]	Rate of radio block It is used if the general config "rate_source" is set to "user_defined" otherwise the rate will be read from waveform configuration file.
	rate_source	"waveform_config"	"waveform_config": The rate will be read from the waveform config file. "user_defined": The given rate by the user in this config file will be used.
	bandwidth	[20.0e6]	For future use
	channels	[0]	Channel selection, for example 0 or 1 for X310 USRP.
	antenna	"TX/RX"	RX antenna selection (TX/RX, RX2)
	gain	[20]	Gain for the RF chain in dB
	clock_reference	"internal"	Rx sync reference source (internal, external, gpsdo)
	duration	10.0e-3	Rx Time duration of IQ data acquisition
	channel_attenuation	33	Expected channel attenuation in dB

Waveform Metadata fields description

The default_waveform_config.yaml file is the default Tx Waveform Config dictionary. This will be replaced in future releases by a dictionary that includes a list of parameters for each standard individually.

Parameter	**Type**	**Description**
standard	string	The protocol used to generate the waveform: i.e. NR, LTE, IEEE, Radar, Unknown
frequency_range	string	FR1, FR2
link_direction	string	UL, DL
duplexing	string	TDD, FDD
multiplexing	string	tdm, fdm, cdm, ofdm, sdm, pdm
spreading	string	fhss, thss, dsss, css …etc
bandwidth	numeric	Unit in Hz
rate	numeric	The sample rate of the signal in samples per second.
n_frames	numeric	Number of frames in the given waveform file
modulation	string	Am, fm, QAM, … etc
MCS	numeric	Modulation and coding scheme
code_rate	numeric	
subcarrier_spacing	numeric	Unit in Hz
test_model	string	

Reference Architecture

The configuration interface reads the configuration file and creates the variation map by doing a cross product over all possible values. Each TX and RX has own list of parameters. Some TX parameters are common for all Tx USRPs, and they are listed under common transmitters config section. The resulting variation map has four sections:

- General configuration
- TX USRP configuration
- Common Tx USRPs configuration
- RX USRP configuration

The application settings are applied when the script starts and cannot be changed once the application is up and running. Figure 5 shows the reference architecture of NI RF Data Recording API.

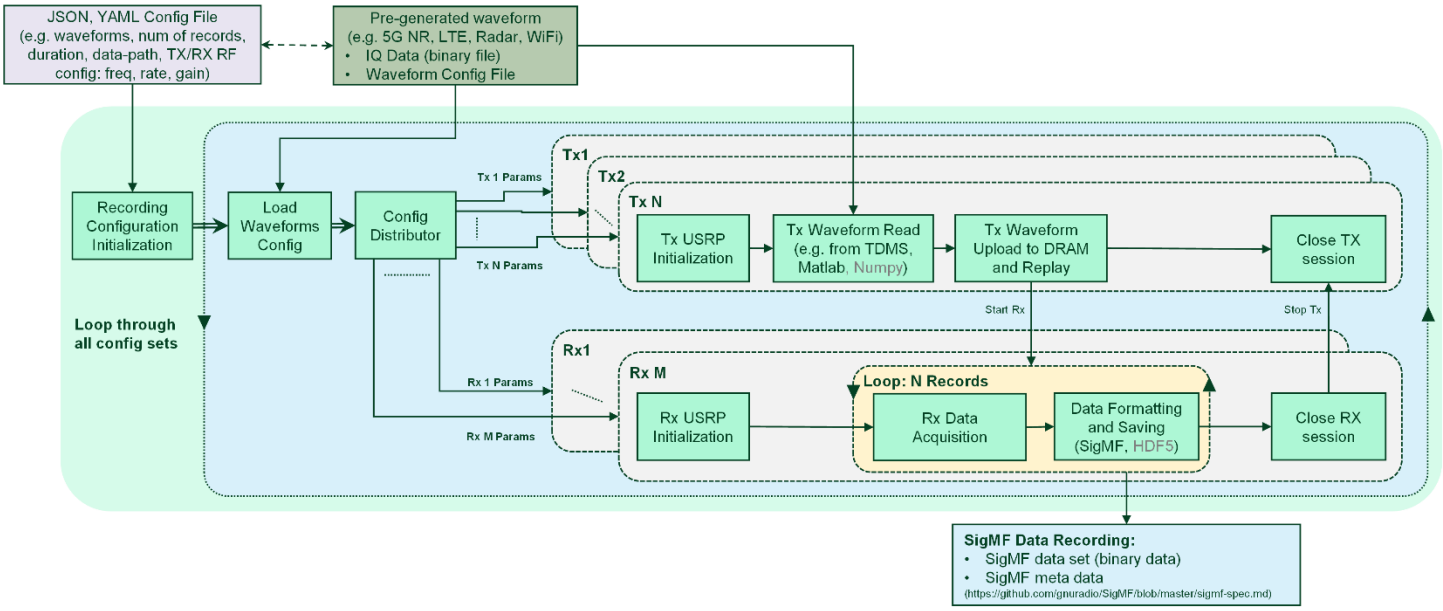

Figure 5 Data Set Recording Reference Architecture

Run NI RF Data Recording API

To run the main script using the default config_rf_data_recording_api.yaml, use the following command:

```
python3.9 main_rf_data_recording_api.py
```

The user can provide the configuration file to the API from the terminal:

```
python3.9 main_rf_data_recording_api.py –main_config config_rf_data_recording_api.yaml
```

Several config files have been created as a template for all operation modes: TX and RX mode, TX only mode or RX only mode

- config_rf_data_recording_api.json
- config_rf_data_recording_api.yaml
- config_rf_data_recording_api_4TX_2RX_test.json
- config_rf_data_recording_api_rx_only.yaml
- config_rf_data_recording_api_tx_only.yaml

Note: For Tx only or Rx only, you can use the config file for this operation mode as it is mentioned in the configuration templates above or you can use the RF Replay Data Transmitter and RF RX Data Recorder scripts by passing the configs directly from the terminal.

LO Configuration

To mitigate the LO leakage/DC offset, the LO can be used for that. Figure 6 shows the relationship between different frequencies:

- Configuration File:
  - TX Target center frequency
  - TX LO offset frequency
- TX RF:
  - TX LO frequency
  - Location of band of interest shifted by LO offset from TX LO frequency
- RX RF
  - RX Target center frequency

Note: $f_{LO-offset,TX}$ can be given with negative sign. So, the $f_{LO,TX} = f_{c,target}$ - $|f_{LO-offset,TX}|$.

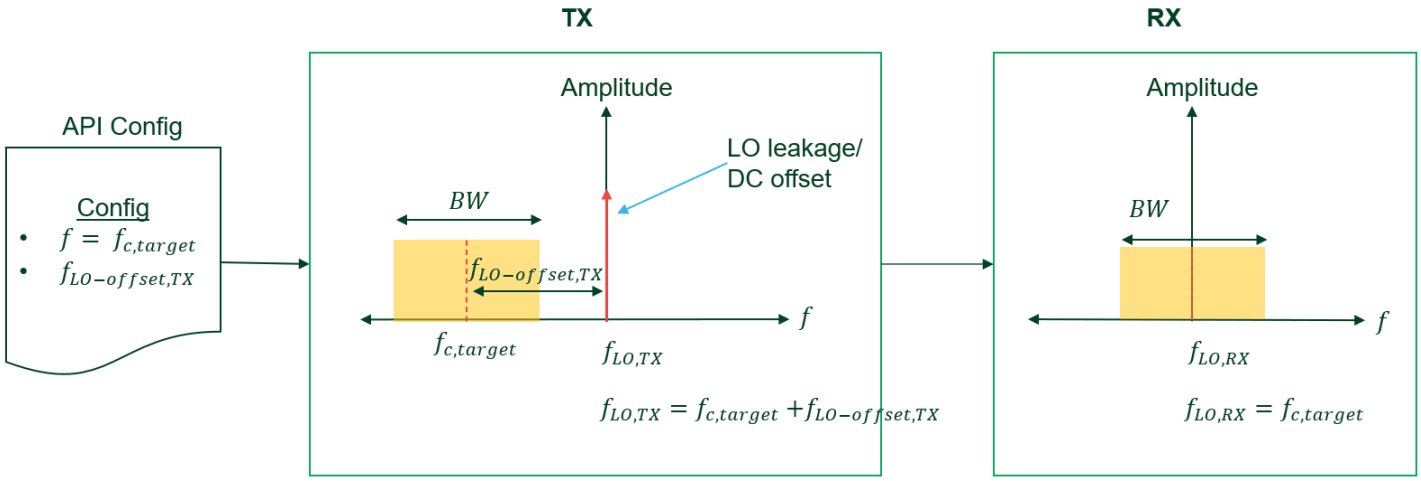

Figure 6 LO Configuration description.

The frequency of LO Offset should be within the following two boundaries:

- Larger than half of the signal bandwidth: $f_{LO-offset,TX} > BW/2$
- Less than half of the deviation between the RF analog bandwidth (USRP daughterboard BW) and signal bandwidth: $f_{LO-offset,TX} < (f_{RF-BW} - BW)/2$

Figure 7 shows the boundaries of LO offset.

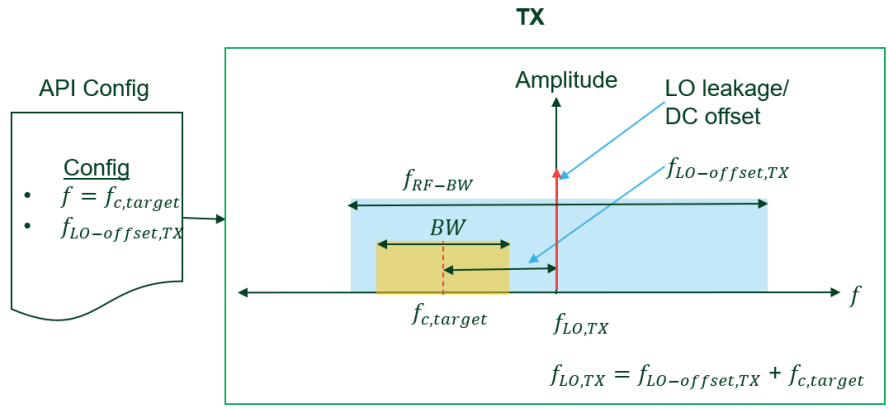

Figure 7 LO Offset boundaries.

Appendix

Troubleshooting

Known Issues

Related Information

[1] If you are transmitting over the air, make sure to consider the instructions given in the "RF Multi Station Mode: Over-the-Air Transmission" section. The USRP devices and NI-5791 are not approved or licensed for transmission over the air using an antenna. As a result, operating those products with an antenna may violate local laws.