PlutoSDR + GNU Radio

In this lab we will explore PlutoSDR and general IIO support in GNU Radio. This is important if you want to extend or modify the existing PlutoSDR software interfaces to support more features available in the transceiver itself.

In the first section of this lab we will be constructing a streaming device from the ground up to get data from PlutoSDR. Start by launching GNU Radio Companion on your PC. Start a fresh QT flowgraph as shown in Figure 1.

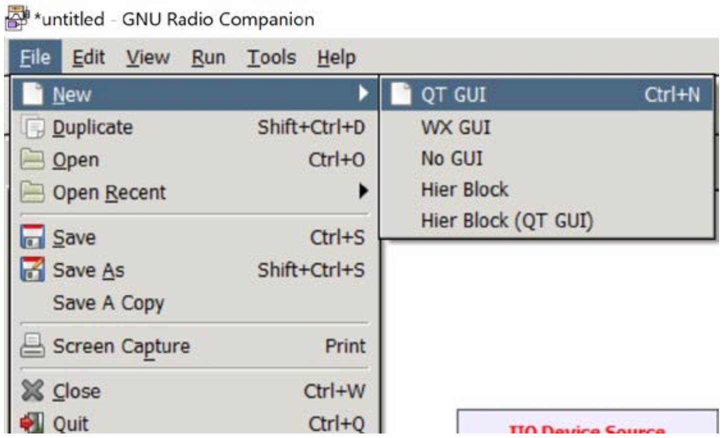
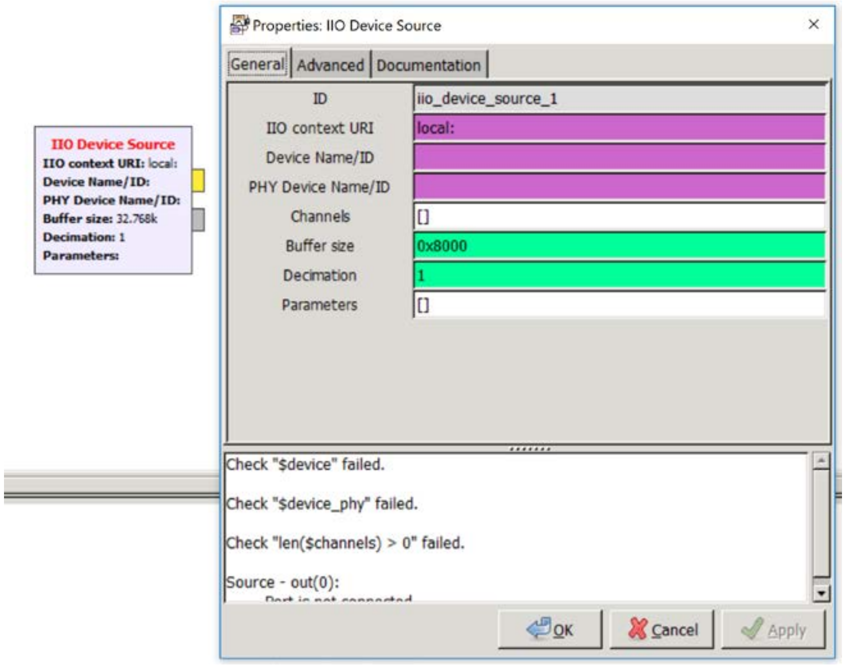


Figure 1

Add an “IIO Device Source” block to the canvas and open its mask as in Figure 2.



IIO Device Source
IIO context URI: local:
Device Name/ID:
PHY Device Name/ID:
Buffer size: 32.768k
Decimation: 1
Parameters:

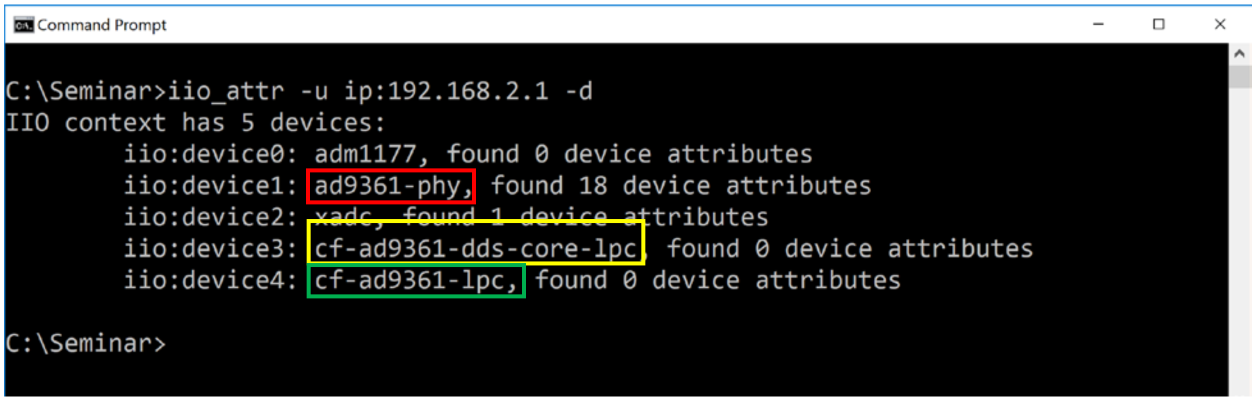
ID	iio_device_source_1
IIO context URI	local:
Device Name/ID	
PHY Device Name/ID	
Channels	[]
Buffer size	0x8000
Decimation	1
Parameters	[]

```
Check "$device" failed.

Check "$device_phy" failed.

Check "len($channels) > 0" failed.

Source - out(0):
```

Figure 2

“IIO Device Source” blocks are designed to provide access to streaming channels in GNU Radio, along with some attribute configuration. However, sometimes this information can be complex to fill out depending on your knowledge of IIO and its drivers. “uri” was already discussed in the previous lab, which simply defines the context we will use to connect to the IIO drivers. Let’s next examine how to fill out the remaining parameters of the block.

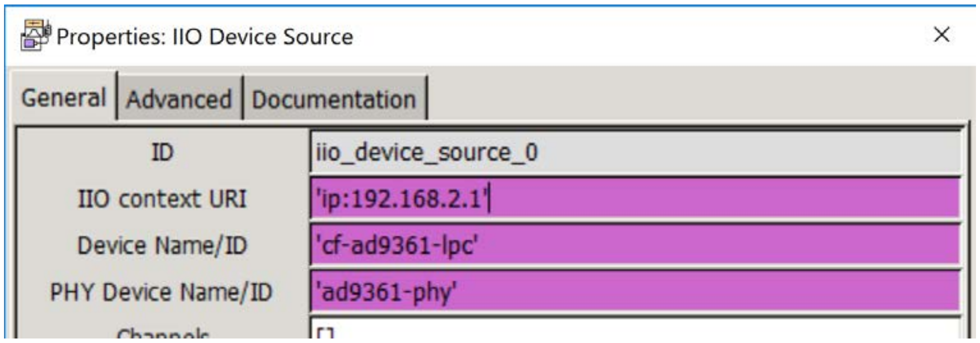
To do so, open a terminal or command prompt. Next list the devices for the context of interest, here we can just use auto since we have PlutoSDR attached. To do this type “iio_attr -u ip:192.168.2.1 -d” as in Figure 3.



```
C:\Seminar>iio_attr -u ip:192.168.2.1 -d
IIO context has 5 devices:
        iio:device0: adm1177, found 0 device attributes
        iio:device1: ad9361-phy, found 18 device attributes
        iio:device2: xadc, found 1 device attributes
        iio:device3: cf-ad9361-dds-core-lpc, found 0 device attributes
        iio:device4: cf-ad9361-lpc, found 0 device attributes

C:\Seminar>
```

Figure 3

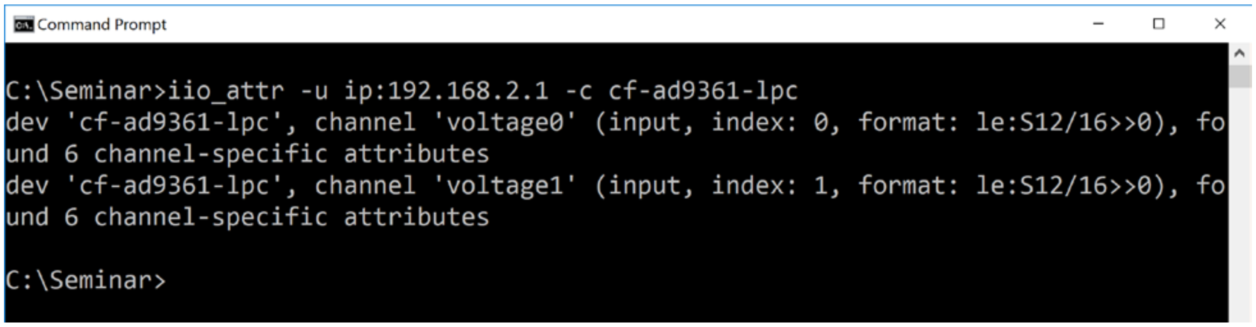
From the devices shown in Figure 3, “ad9361-phy” is the “PHY Device Name” needed for the “IIO Device Source” block. This is responsible for device configuration, and any parameters written will use that device’s underlying attribute tree. “cf-ad9361-lpc” and “cf-ad9361-dds-core-lpc” are the “Device Name” for the receiver (Source) and transmitter (Sink) respectively. Using this information, we can fill out the mask as in Figure 4. Do so on your own block masks to match with your respective “uri”.



ID	iio_device_source_0
IIO context URI	'ip:192.168.2.1'
Device Name/ID	'cf-ad9361-lpc'
PHY Device Name/ID	'ad9361-phy'

Figure 4

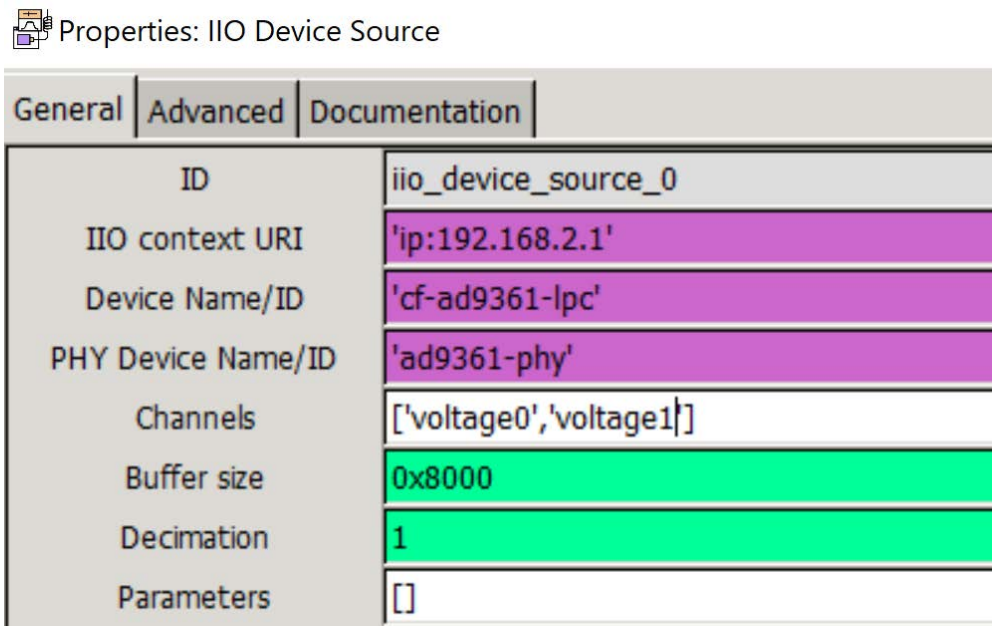
Next, we need to set the channels. The channels are the streams of data we want to pull data from. To list the available channel names for the receiver, run the command “iio_attr -u ip:192.168.2.1 -c cf-ad9361-lpc”, which will list all the available channels for that device as in Figure 5. Two channels are listed, voltage0 and voltage1. These are the I and Q data streams respectively, and if you wish to learn more about these channels read over the driver’s wiki page here.



```
Command Prompt

C:\Seminar>iio_attr -u ip:192.168.2.1 -c cf-ad9361-lpc
dev 'cf-ad9361-lpc', channel 'voltage0' (input, index: 0, format: le:S12/16>>0), fo
und 6 channel-specific attributes
dev 'cf-ad9361-lpc', channel 'voltage1' (input, index: 1, format: le:S12/16>>0), fo
und 6 channel-specific attributes

C:\Seminar>
```

Figure 5

We next will take these channel names and apply them to our "Channels" parameter in the block mask as a list of strings as in Figure 6. Do so on your own block masks to match.



Properties: IIO Device Source

ID	iio_device_source_0
IIO context URI	'ip:192.168.2.1'
Device Name/ID	'cf-ad9361-lpc'
PHY Device Name/ID	'ad9361-phy'
Channels	['voltage0','voltage1']
Buffer size	0x8000
Decimation	1
Parameters	[]

Figure 6

At this point, we have the necessary pieces to stream data from the transceiver itself. The data coming out of the source block will be shorts, which is accurate for the transceiver. However, this is not super friendly for many of the GNU Radio scopes. Add additional blocks and a scope as shown in Figure 7.

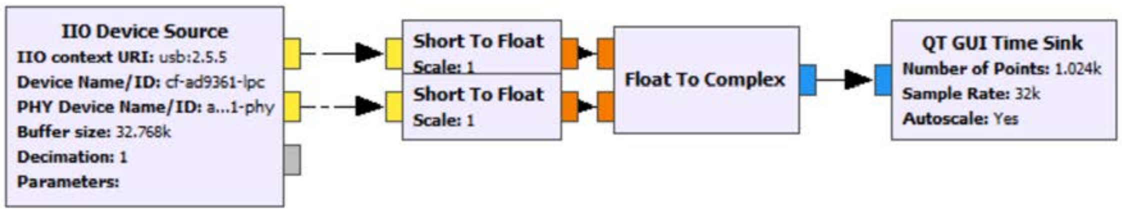
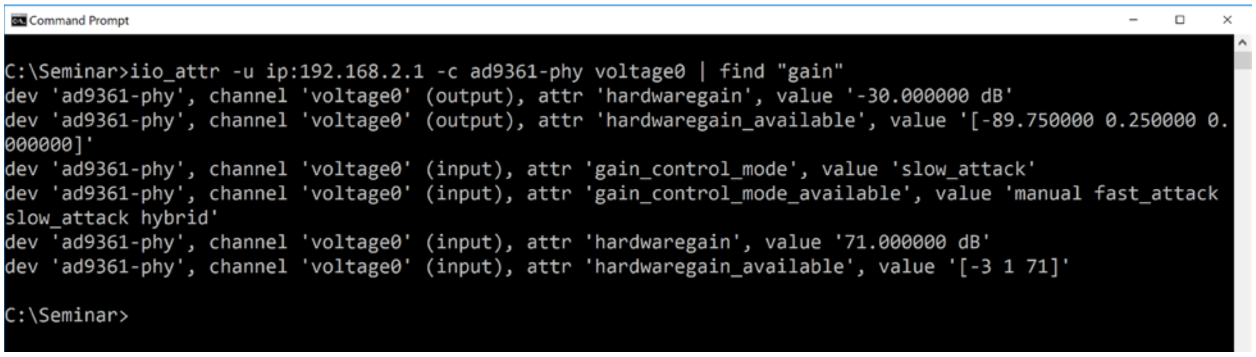


Figure 7

Next hit run on the flowgraph and view some of the data coming out of the transceiver.

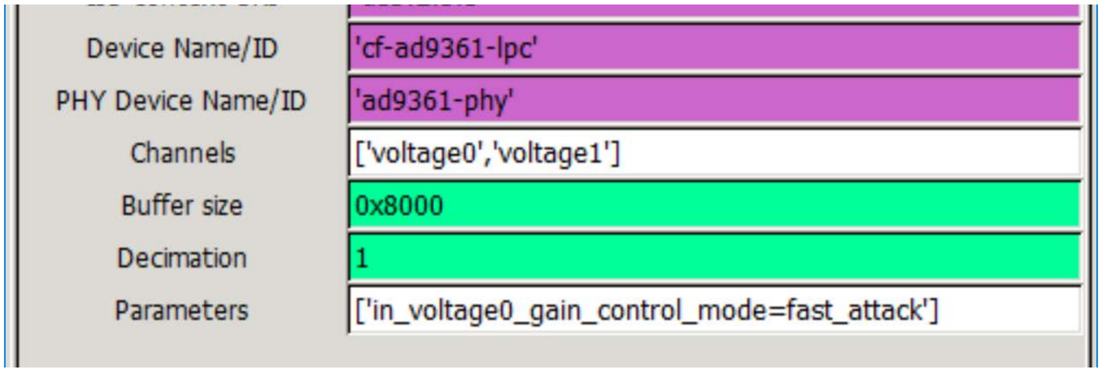
This is nice and all, but we also want to configure some settings on the device. For example, the gain control mode of the receiver. To get the attribute name we add the channel name into our “iio_attr” command with respect to the control data driver name “ad9361-phy” with the following command “iio_attr -u ip:192.168.2.1 -c ad9361-phy voltage0”. In Figure 8 we show the filtered output of this command since there are many attributes associated with this driver, but we get the information we need. The attribute name “gain_control_mode” and possible options with “gain_control_mode_available”.



```
C:\Seminar>iio_attr -u ip:192.168.2.1 -c ad9361-phy voltage0 | find "gain"
dev 'ad9361-phy', channel 'voltage0' (output), attr 'hardwaregain', value '-30.000000 dB'
dev 'ad9361-phy', channel 'voltage0' (output), attr 'hardwaregain_available', value '[-89.750000 0.250000 0.
000000]'
dev 'ad9361-phy', channel 'voltage0' (input), attr 'gain_control_mode', value 'slow_attack'
dev 'ad9361-phy', channel 'voltage0' (input), attr 'gain_control_mode_available', value 'manual fast_attack
slow_attack hybrid'
dev 'ad9361-phy', channel 'voltage0' (input), attr 'hardwaregain', value '71.000000 dB'
dev 'ad9361-phy', channel 'voltage0' (input), attr 'hardwaregain_available', value '[-3 1 71]'

C:\Seminar>
```

Figure 8

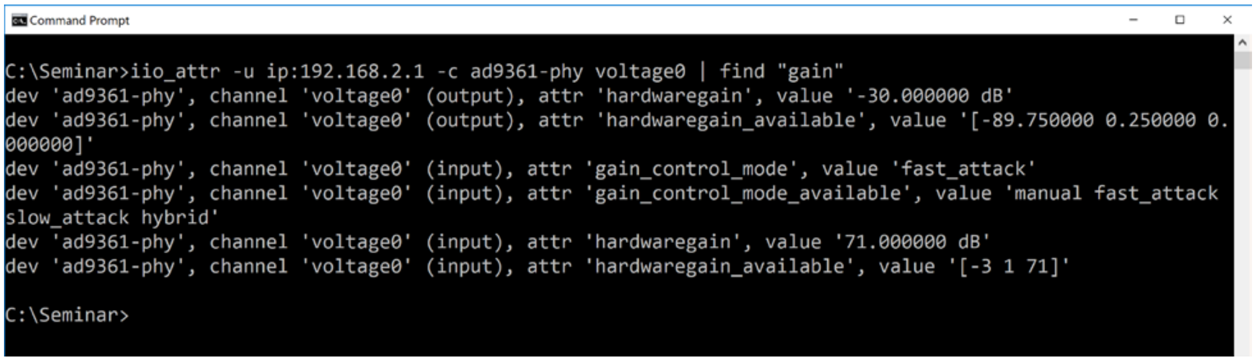
We can use this information in the GNU Radio block masks now under the parameters array. Since the PHY driver and write any attribute on the device, we must explicitly construct the attribute name in the following way: <in/out>_<channel name>_<attribute name>. So for “gain_control_mode” this would be “in_voltage0_gain_control_mode”. To set the value simply append “=<value>” to this string. To set the “gain_control_mode” of the device apply the following to the block mask as shown in Figure 9. Apply this to your block and press play again to apply this change.



```
Device Name/ID       'cf-ad9361-lpc'
PHY Device Name/ID   'ad9361-phy'
Channels             ['voltage0','voltage1']
Buffer size          0x8000
Decimation           1
Parameters           ['in_voltage0_gain_control_mode=fast_attack']
```

Figure 9

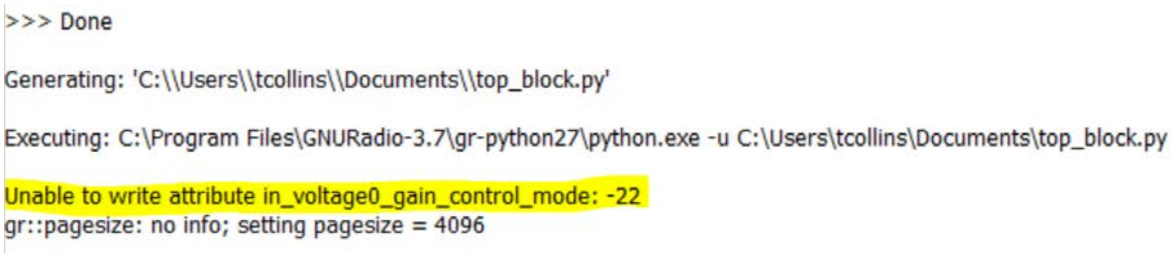
Running the “iio_attr -n 192.168.2.1 -c ad9361-phy voltage0” command we can see the updated attribute as in Figure 10.



```
C:\Seminar>iio_attr -u ip:192.168.2.1 -c ad9361-phy voltage0 | find "gain"
dev 'ad9361-phy', channel 'voltage0' (output), attr 'hardwaregain', value '-30.000000 dB'
dev 'ad9361-phy', channel 'voltage0' (output), attr 'hardwaregain_available', value '[-89.750000 0.250000 0.
000000]'
dev 'ad9361-phy', channel 'voltage0' (input), attr 'gain_control_mode', value 'fast_attack'
dev 'ad9361-phy', channel 'voltage0' (input), attr 'gain_control_mode_available', value 'manual fast_attack
slow_attack hybrid'
dev 'ad9361-phy', channel 'voltage0' (input), attr 'hardwaregain', value '71.000000 dB'
dev 'ad9361-phy', channel 'voltage0' (input), attr 'hardwaregain_available', value '[-3 1 71]'

C:\Seminar>
```

Figure 10

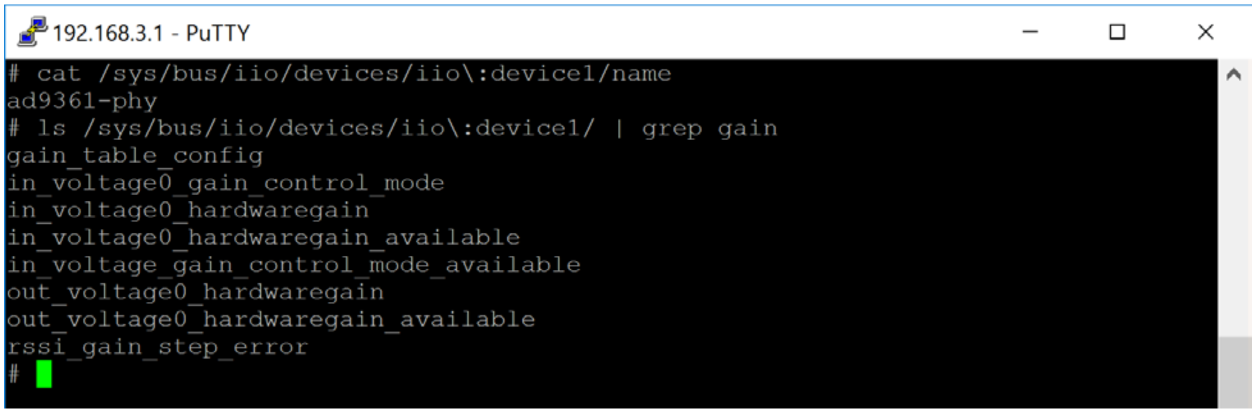
If an attribute is incorrect you will observe some output in the GNU Radio console. For example, if you tried to set the gain_mode to ‘hyper_attack’, you would observe something similar to Figure 11.



```
>>> Done

Generating: 'C:\\Users\\tcollins\\Documents\\top_block.py'

Executing: C:\Program Files\GNURadio-3.7\gr-python27\python.exe -u C:\Users\tcollins\Documents\top_block.py

Unable to write attribute in_voltage0_gain_control_mode: -22
gr::pagesize: no info; setting pagesize = 4096
```

Figure 11

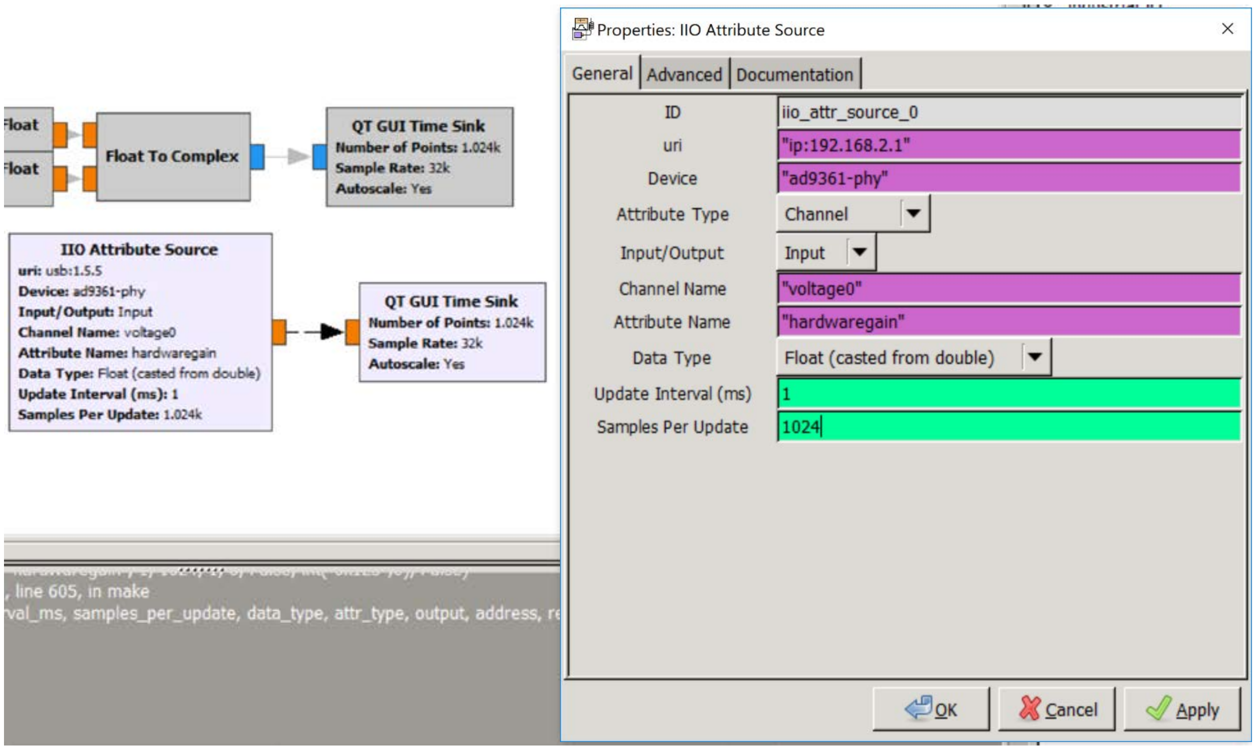
Alternatively, with regards to determining attribute names you can login to PlutoSDR itself over SSH and list device parameters as shown in Figure 12. **This is purely for reference, you are not required to access the radio over SSH in this lab.**



```
# cat /sys/bus/iio/devices/iio\:device1/name
ad9361-phy
# ls /sys/bus/iio/devices/iio\:device1/ | grep gain
gain_table_config
in_voltage0_gain_control_mode
in_voltage0_hardwaregain
in_voltage0_hardwaregain_available
in_voltage_gain_control_mode_available
out_voltage0_hardwaregain
out_voltage0_hardwaregain_available
rssi_gain_step_error
#
```

Figure 12

Now if we are just interested in interacting with attributes there are specific blocks for doing so. They are parameterized in a similar way, but one would argue match nicely to the API of "iio_attr". For example, if we want to read the current gain setting of the transceiver we could use the "IIO Attribute Source" block, will the following configuration as shown in Figure 13.



IIO Attribute Source
uri: usb:1.5.5
Device: ad9361-phy
Input/Output: Input
Channel Name: voltage0
Attribute Name: hardwaregain
Data Type: Float (casted from double)
Update Interval (ms): 1
Samples Per Update: 1.024k

```
, line 605, in make
val_ms, samples_per_update, data_type, attr_type, output, address, re
```

ID	iio_attr_source_0
uri	"ip:192.168.2.1"
Device	"ad9361-phy"
Attribute Type	Channel
Input/Output	Input
Channel Name	"voltage0"
Attribute Name	"hardwaregain"
Data Type	Float (casted from double)
Update Interval (ms)	1
Samples Per Update	1024

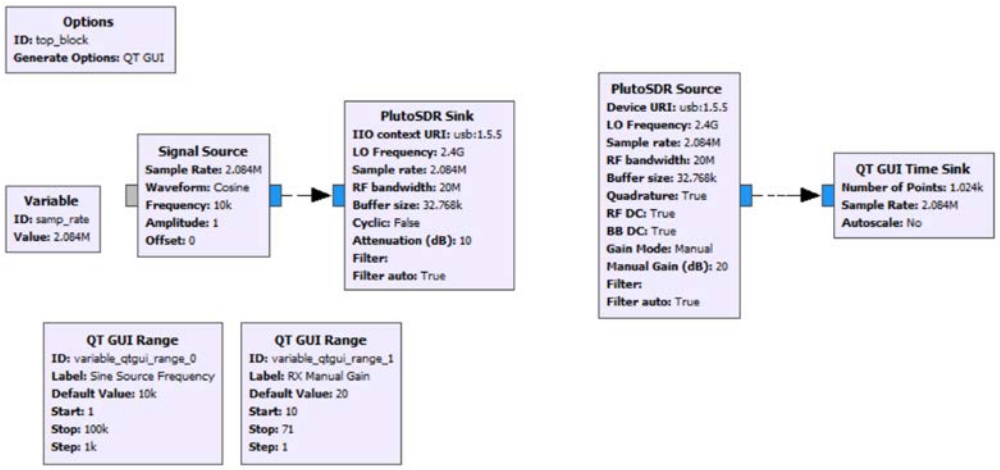
Figure 13

This block will continuously read from an attribute and produce buffers of size "Samples Per Update" where each sample is read at an interval of "Update Interval" milliseconds.

Device Specific Blocks

Now having to create a block from scratch each time is rather painful for each device. Therefore, device specific blocks exist for devices like PlutoSDR and the FMComms boards. We will be using these for the remainder of the lab.

Start by open the PlutoTestSine.grc flowgraph file which will look like Figure 14 when loaded.



Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2.084M

Signal Source
Sample Rate: 2.084M
Waveform: Cosine
Frequency: 10k
Amplitude: 1
Offset: 0

PlutoSDR Sink
IIO context URI: usb:1.5.5
LO Frequency: 2.4G
Sample rate: 2.084M
RF bandwidth: 20M
Buffer size: 32.768k
Cyclic: False
Attenuation (dB): 10
Filter:
Filter auto: True

PlutoSDR Source
Device URI: usb:1.5.5
LO Frequency: 2.4G
Sample rate: 2.084M
RF bandwidth: 20M
Buffer size: 32.768k
Quadrature: True
RF DC: True
BB DC: True
Gain Mode: Manual
Manual Gain (dB): 20
Filter:
Filter auto: True

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.084M
Autoscale: No

QT GUI Range
ID: variable_qtgui_range_0
Label: Sine Source Frequency
Default Value: 10k
Start: 1
Stop: 100k
Step: 1k

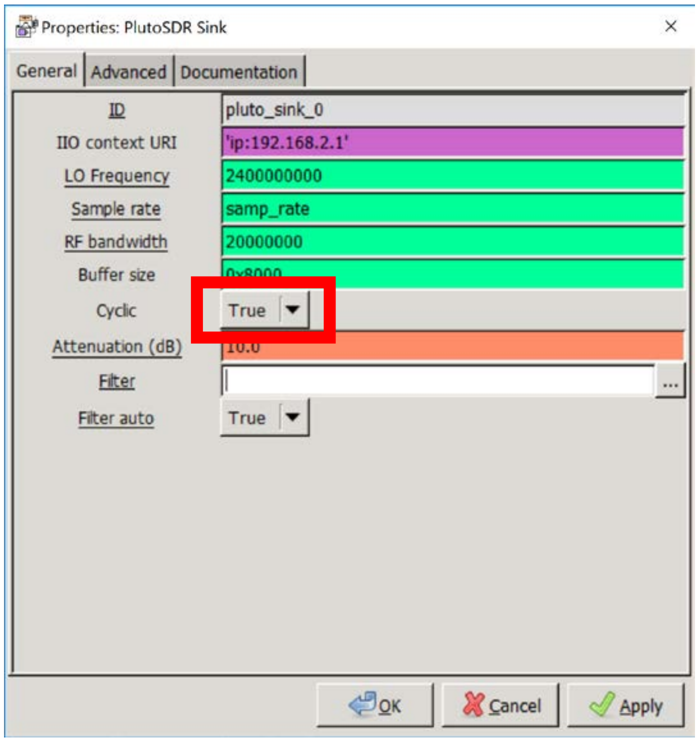
QT GUI Range
ID: variable_qtgui_range_1
Label: RX Manual Gain
Default Value: 20
Start: 10
Stop: 71
Step: 1

Figure 14

There are some nuances to the PlutoSDR blocks that we will discuss here. First start by pressing play and changing the sliders and viewing the resulting effects on the received waveforms. Almost every parameter of the device is configurable during runtime, except buffer settings.

Next, open the "PlutoSDR Sink" and set the "Cyclic" parameter to true as shown in Figure 15.

Then hit play again and move both the sliders to adjust settings.



Properties: PlutoSDR Sink

General | Advanced | Documentation

ID	pluto_sink_0
IIO context URI	'ip:192.168.2.1'
LO Frequency	2400000000
Sample rate	samp_rate
RF bandwidth	20000000
Buffer size	0x8000
Cyclic	True
Attenuation (dB)	10.0
Filter	
Filter auto	True

Figure 15

You will notice that the frequency of the signal will not change. This happens because in Cyclic mode the transmitter takes only the first buffer supplied and continually repeats it. This is useful for testing a design when a waveform is constant or there are large bandwidth requirements of the link over USB/Ethernet/… to the device. This will prevent gaps in the data stream since everything is being produced on the device itself.