REDHAWK Installation and Usage

V1.0

July 18, 2016

1.1 Deployment

The provided REDHAWK modules assume that the system is CENTOS 6.5+ with a functional REDHAWK 2.0.1 base install and working 10 Gbe Ethernet NICs. The basic steps for getting REDHAWK working are to:

1. Verify the 10 Gbe setup
2. Check radio operation
3. Install the CyberRadioDriver code
4. Patch the REDHAWK installation
5. Install the CyberRadio REDHAWK RPMs

1.1.1 10 Gbe Setup

The recommended NIC is X520-DA2 (http://ark.intel.com/products/39776/Intel-Ethernet-Converged-Network-Adapter-X520-DA2). Additionally, the recommended approach is:

- Use the latest available driver from Intel
- Increase the maximum hardware buffer from 512 to 4096 (ethtool -G ...)
- Disable flow control, on the PC side (ethtool -A ...) and/or the radio side
    - The radio's transmit buffering can get messed up if the receiving application falls behind and triggers an Ethernet PAUSE frame

1.1.2 Basic Radio Setup

The basic radio operation can be verified via a telnet session and wireshark. The following command sequence will create traffic flow on the 10 gbe interface assuming the radio is at 192.168.0.3 and the 10 gbe interface is at 192.168.2.10.

```
telnet 192.168.0.3 8617
DIP 1, 0, 192.168.2.10, 90:E2:BA:9A:5B:20, 31000, 41000
WBDDC 1, 4, 0, 1, 1, 41000
```

At this point there should be about 6250 packets per second flowing on the 10 gbe Ethernet interface directed to UDP port 41000.

1.1.3 CyberRadioDriver

The CyberRadioDriver code and installation instructions (README file) are on github at https://github.com/CyberRadio/gr-cyberradio/tree/master/cyberradiodriver.

Once installed the CRD can be validated within Python using the following sample code. Note the IP address should be changed to the actual address of the particular radio being controlled.

```
import CyberRadioDriver as crd
```

```
crd.getSupportedRadios()
radio=crd.getRadioObject("ndr651")
radio.connect("tcp","192.168.0.3","8617")
radio.getVersionInfo()
radio.sendCommand("*idn?\n")
```

A correct reply should be similar to:

```
['NDR651', 'S/N 0007', 'OK']
```

1.1.4 Patch REDHAWK

The base 2.0.1 REDHAWK installation Python code has a bug in it where the DigitalTunerPort does not inherit the setCenterFrequency, setTunerBandwidth, etc. methods correctly. The code at /usr/local/redhawk/core/lib/python/frontend/input_ports.py:204 needs to be modified so that the bold text below is added:

```
class InDigitalTunerPort(FRONTEND__POA.DigitalTuner, InAnalogTunerPort):
  def __init__(self, name, parent=digital_tuner_delegation()):
  self.name = name
  self.port_lock = threading.Lock()
  self.parent = parent
  InAnalogTunerPort.__init__(self, name, parent)
```

Alternatively, the provided .patch file can be used. The instructions are:

```
cd /usr/local/redhawk/core/lib/python/frontend/
patch < rh_cyberradio.patch
```

1.1.5 Install RPMs

An .rpm file can be generated by the build.sh script:

```
./build.sh rpm
```

The resulting rpm is located in ~/rpmbuild/RPMS/x86_64. Its contents can be inspected with:

```
[cpope@REDHAWK x86_64]$ rpm -qlp rh.SourceVITA49-3.0.0-1.el6.x86_64.rpm
/var/REDHAWK/sdr/dom/components/rh/SourceVITA49
/var/REDHAWK/sdr/dom/components/rh/SourceVITA49/SourceVITA49.prf.xml
/var/REDHAWK/sdr/dom/components/rh/SourceVITA49/SourceVITA49.scd.xml
/var/REDHAWK/sdr/dom/components/rh/SourceVITA49/SourceVITA49.spd.xml
/var/REDHAWK/sdr/dom/components/rh/SourceVITA49/cpp
/var/REDHAWK/sdr/dom/components/rh/SourceVITA49/cpp/SourceVITA49
```

So there are three xml files and one executable (for a C++ project) that when installed in a REDHAWK system end up in either /var/REDHAWK/sdr/dev or /var/REDHAWK/sdr/dom depending on whether it's a device or not.

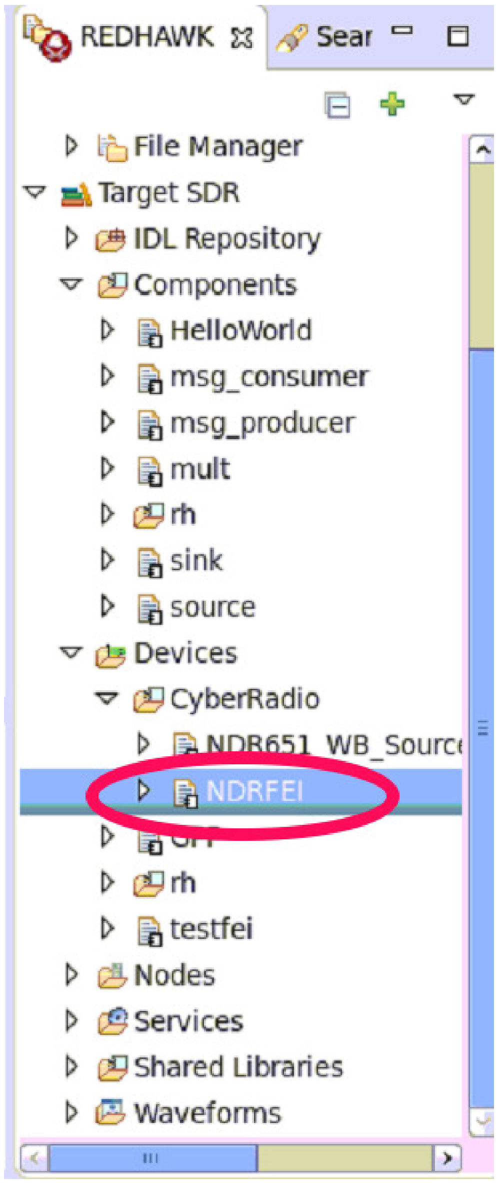
To manually install the rpms run:

```
rpm -Uvh rh.SourceVITA49-3.0.0-1.el6.x86_64.rpm
rpm -Uvh rh.SinkVITA49-3.0.0-1.el6.x86_64.rpm
rpm -Uvh CyberRadio.NDRFEI-1.0.0-1.el6.noarch.rpm
```

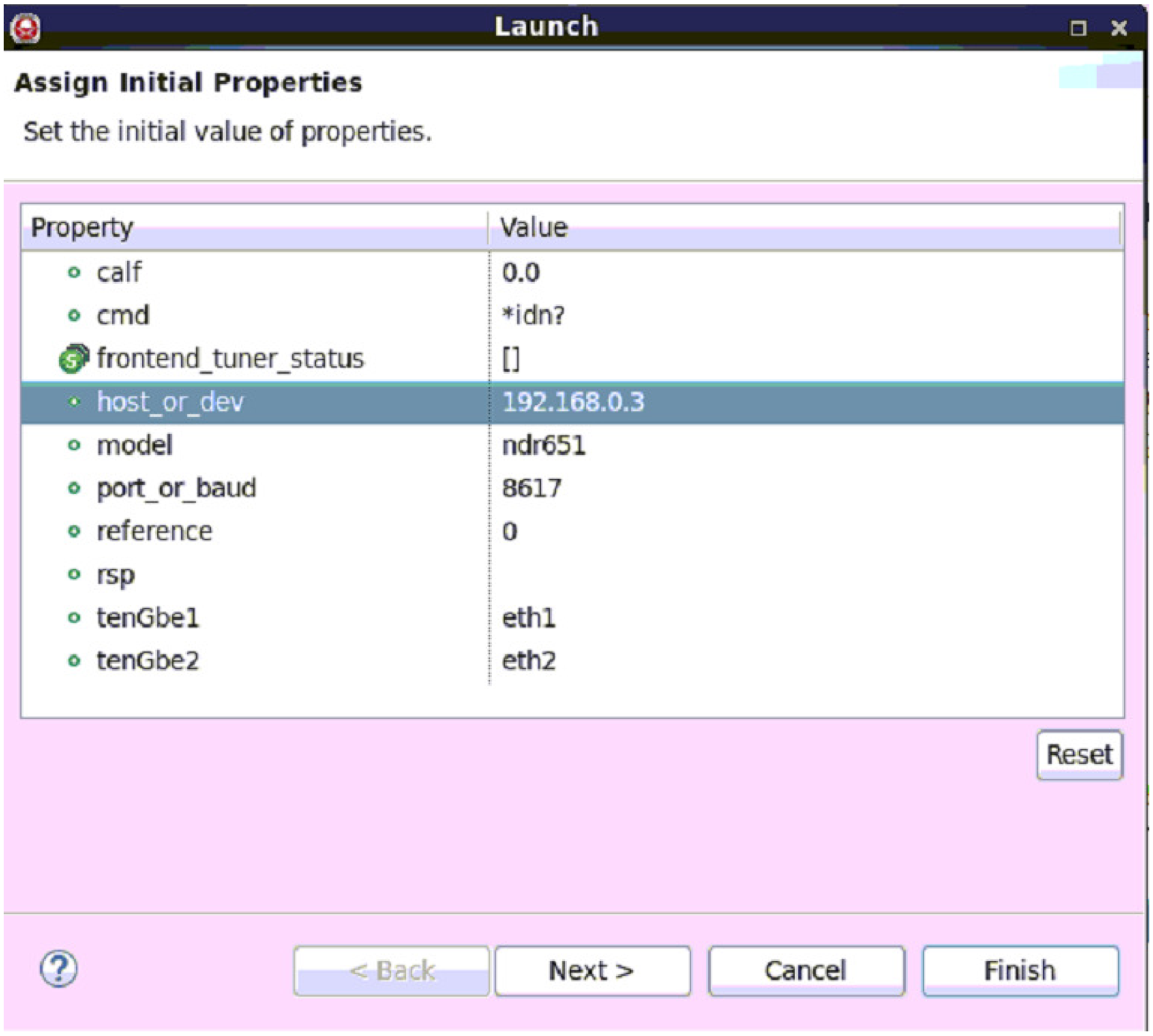
1.2 Usage

There are currently three REDHAWK modules to support Cyber Radio products. The first, CyberRadio.NDRFEI, provides configuration and control capability. This module is where receivers and transmitters can be allocated and configured. The other modules are rh.SinkVITA49 and rh.SourceVITA49 and they are modified versions of the stock modules provided by the REDHAWK team. The modifications are to adapt the headers to match what the Cyber Radio products provide and expect. These modules are a superset of the stock modules so they can replace them in existing applications with little or no impact.

The Network Defined Radio Front End Interface (NDRFEI) is available in the REDHAWK IDE under Target SDR->Devices->CyberRadio.



Right clicking the device and selecting Launch in Sandbox->Advanced from the pop-up menu allows the parameters to be configure. The primary settings are the IP address and port number of the control interface to the radio, the radio model, and the local Ethernet interfaces of the two 10 gigabit data ports.

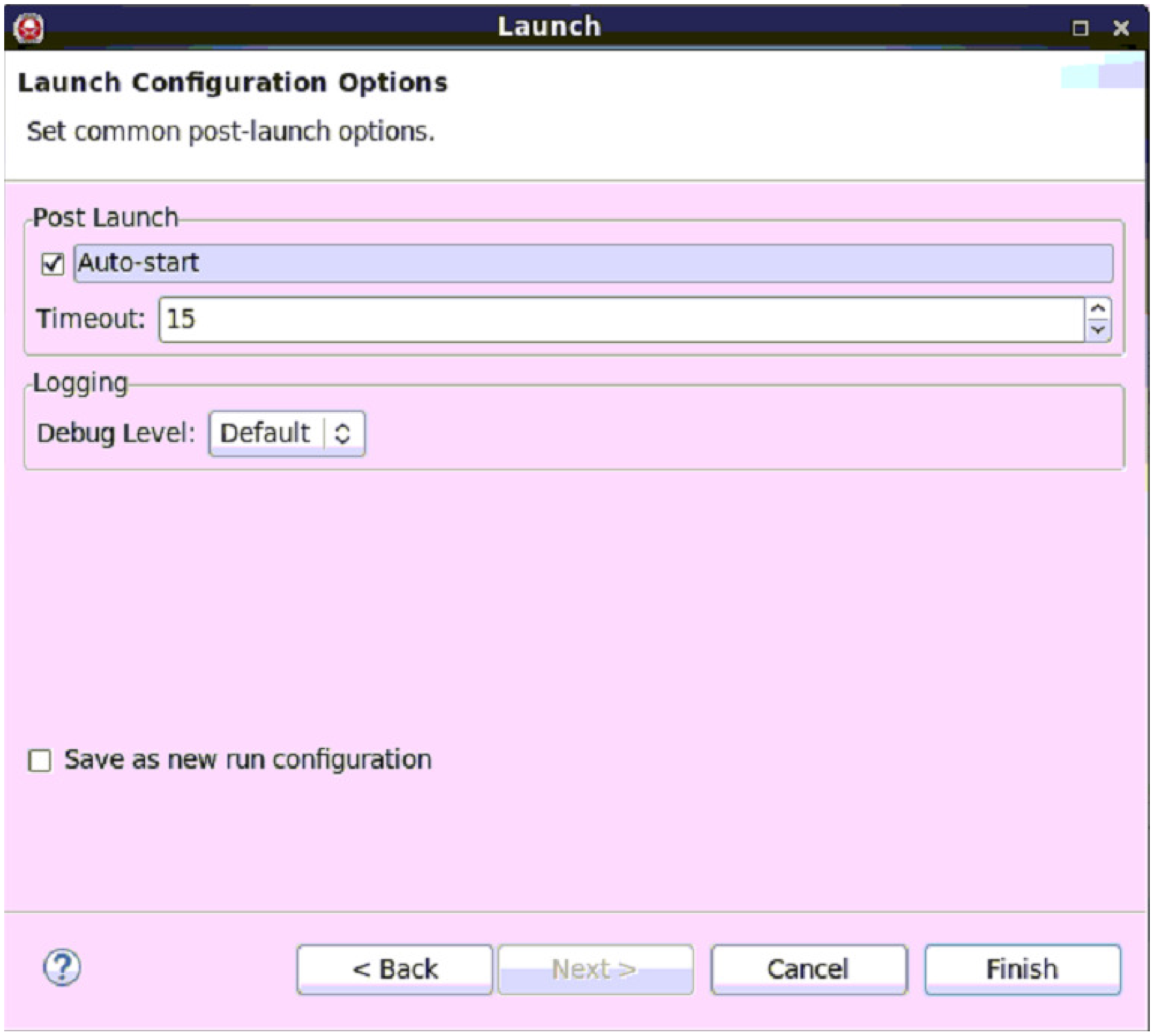


Assign Initial Properties

Set the initial value of properties.

Property	Value
calf	0.0
cmd	*idn?
frontend_tuner_status	[]
host_or_dev	192.168.0.3
model	ndr651
port_or_baud	8617
reference	0
rsp	
tenGbe1	eth1
tenGbe2	eth2

The next Tab allows the modulation to start automatically when launched then click Finish.



Post Launch

☑ Auto-start

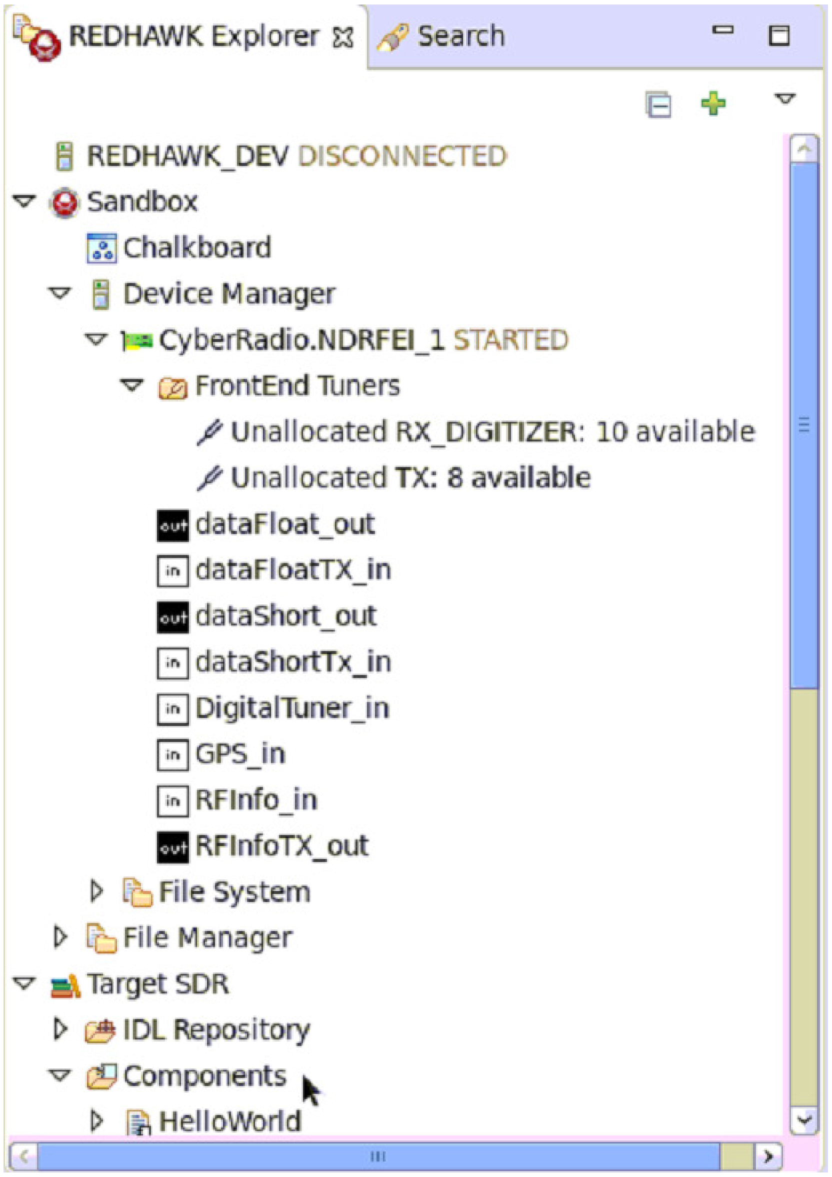
Timeout: 15

Logging

Debug Level: Default

☐ Save as new run configuration

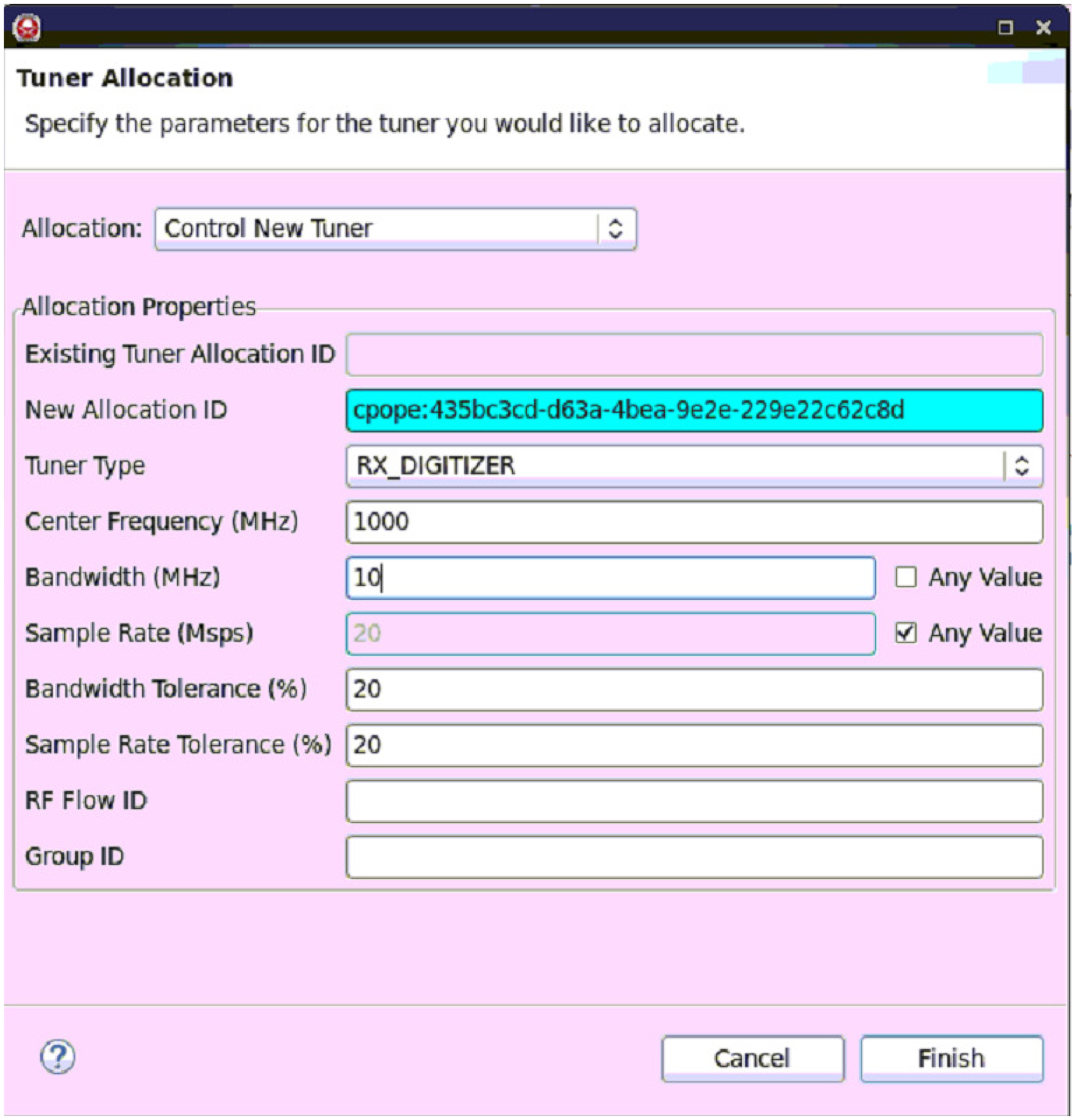
Once running the device will automatically contact the radio and determine the number of available transmitters and receivers to populate the FrontEnd Tuners listing.



REDHAWK Explorer
Search

REDHAWK_DEV DISCONNECTED
Sandbox
Chalkboard
Device Manager
CyberRadio.NDRFEI_1 STARTED
FrontEnd Tuners
Unallocated RX_DIGITIZER: 10 available
Unallocated TX: 8 available
dataFloat_out
dataFloatTX_in
dataShort_out
dataShortTx_in
DigitalTuner_in
GPS_in
RFInfo_in
RFInfoTX_out
File System
File Manager
Target SDR
IDL Repository
Components
HelloWorld

Right click an RX_DIGITIZER and select Allocate to bring up a configuration panel. The available bandwidths/sample rates and tuning rangers will vary from radio to radio. The IDE will report an exception if an unsupported request is made.



Tuner Allocation

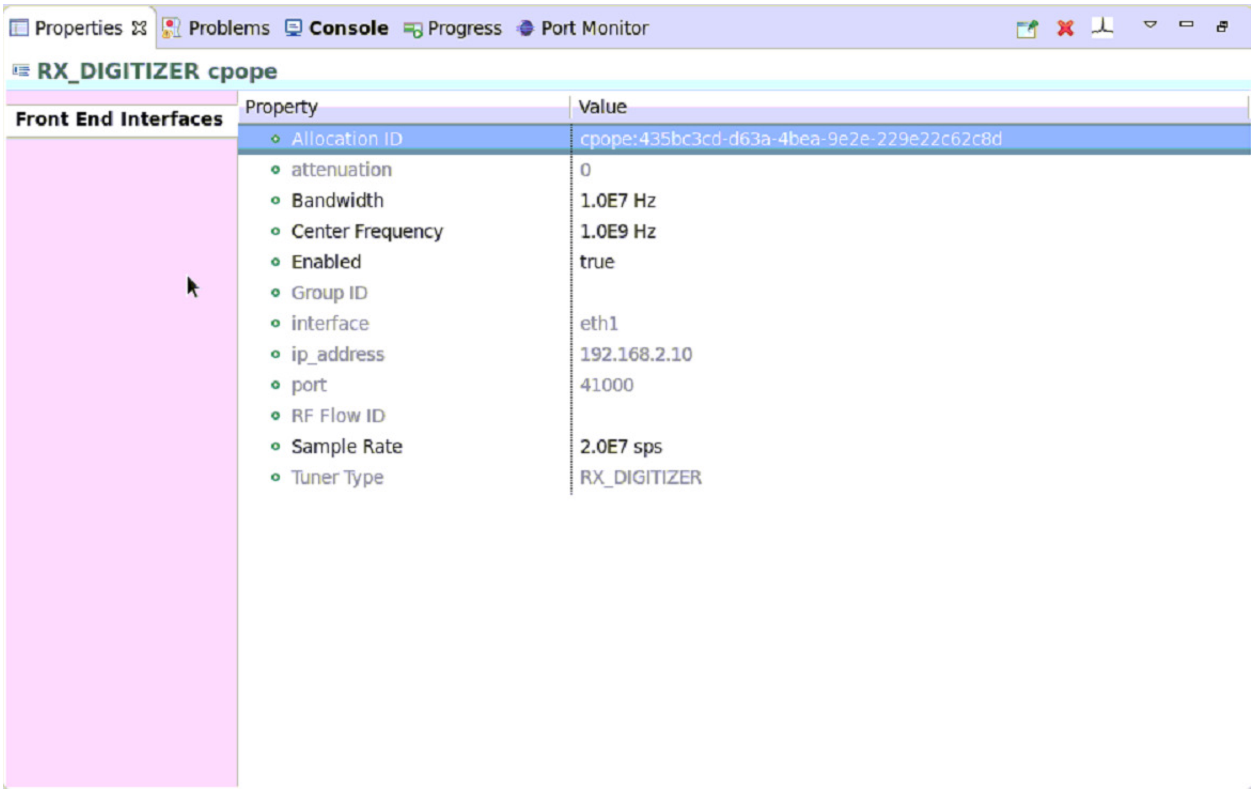
Specify the parameters for the tuner you would like to allocate.

Allocation: Control New Tuner

Allocation Properties

Existing Tuner Allocation ID		
New Allocation ID	cpope:435bc3cd-d63a-4bea-9e2e-229e22c62c8d	
Tuner Type	RX_DIGITIZER	
Center Frequency (MHz)	1000	
Bandwidth (MHz)	10	☐ Any Value
Sample Rate (Msps)	20	☑ Any Value
Bandwidth Tolerance (%)	20	
Sample Rate Tolerance (%)	20	
RF Flow ID		
Group ID		

Once allocated the configuration of the receiver will appear in the properties Tab. The highlighted entries can be modified and the changes will be propagated to the radio.


Properties | Problems | Console | Progress | Port Monitor

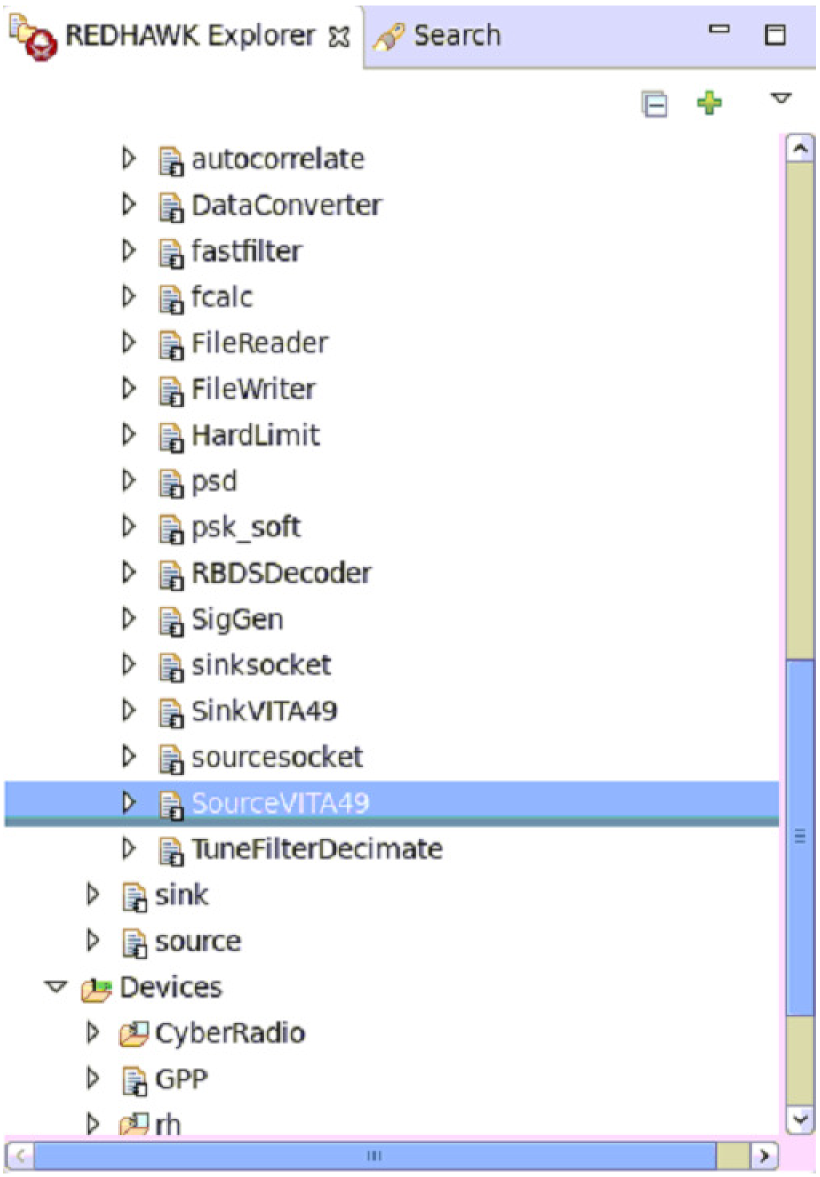
RX_DIGITIZER cpope

Front End Interfaces

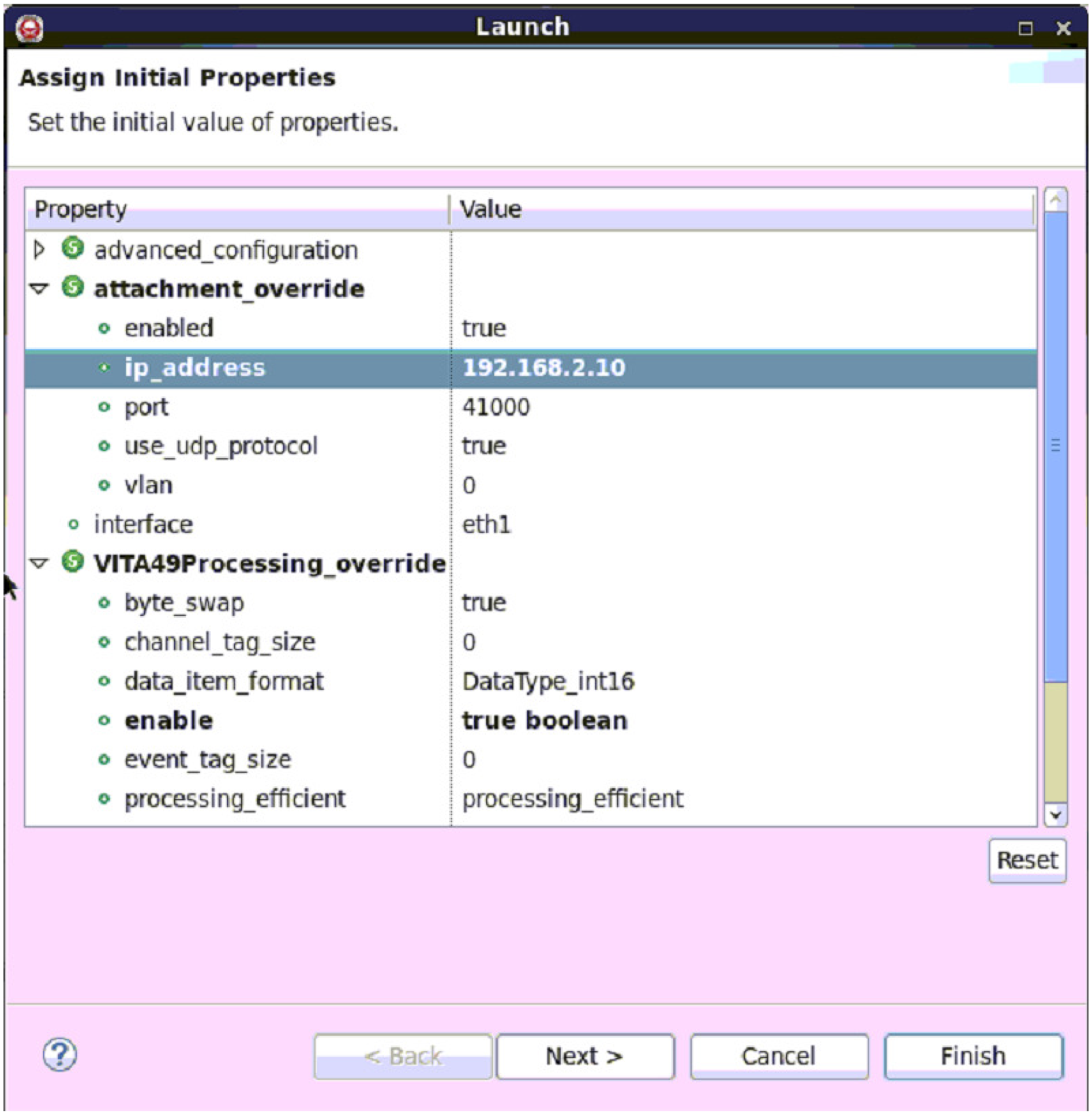
Property	Value
Allocation ID	cpope:435bc3cd-d63a-4bea-9e2e-229e22c62c8d
attenuation	0
Bandwidth	1.0E7 Hz
Center Frequency	1.0E9 Hz
Enabled	true
Group ID	
interface	eth1
ip_address	192.168.2.10
port	41000
RF Flow ID	
Sample Rate	2.0E7 sps
Tuner Type	RX_DIGITIZER

It is important to note the interface, ip_address, and port properties as they will be needed to configure the datapath components.

Next right click the rh.SourceVITA49 component and select Launch in Sandbox->Advanced.



Most of the initial properties are already set correctly for the Cyber Radio products but the ip_address, port, and interface under attachment_override must be set to match the RX_DIGITIZER In the previous steps. Additionally, the VITA49Processing_override.enable property must be set to true.

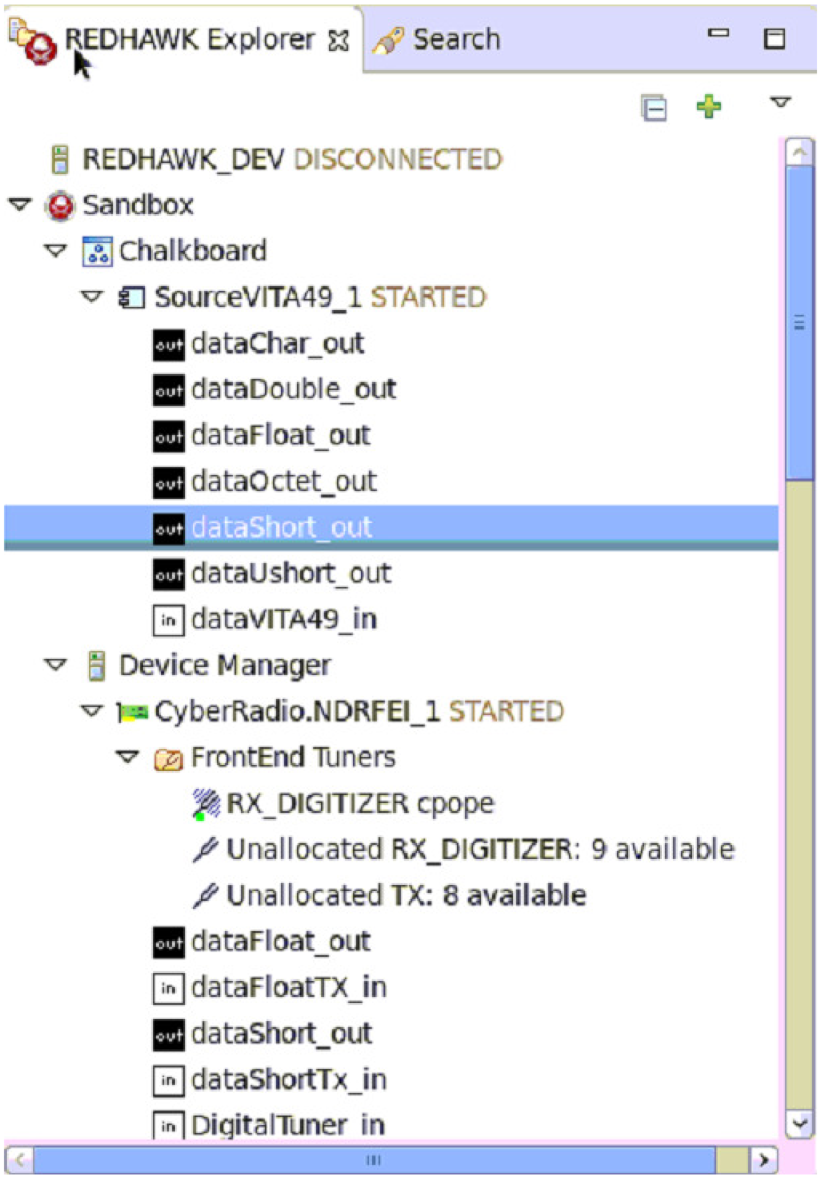


Assign Initial Properties

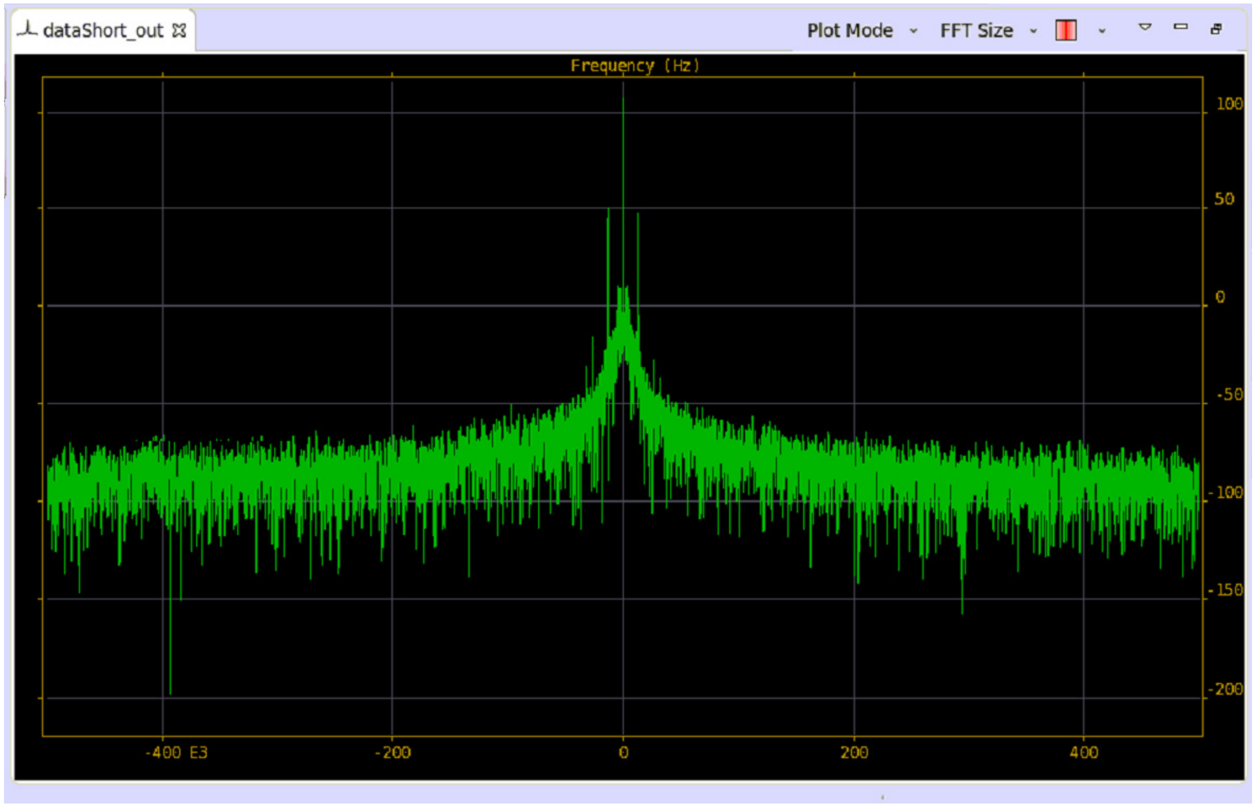
Set the initial value of properties.

Property	Value
advanced_configuration	
attachment_override	
enabled	true
ip_address	**192.168.2.10**
port	41000
use_udp_protocol	true
vlan	0
interface	eth1
VITA49Processing_override	
byte_swap	true
channel_tag_size	0
data_item_format	DataType_int16
enable	**true boolean**
event_tag_size	0
processing_efficient	processing_efficient

Then click the Next tab, enable Auto-Start at click Finish. The SourceVITA49_1 block will show up under the Chalkboard.

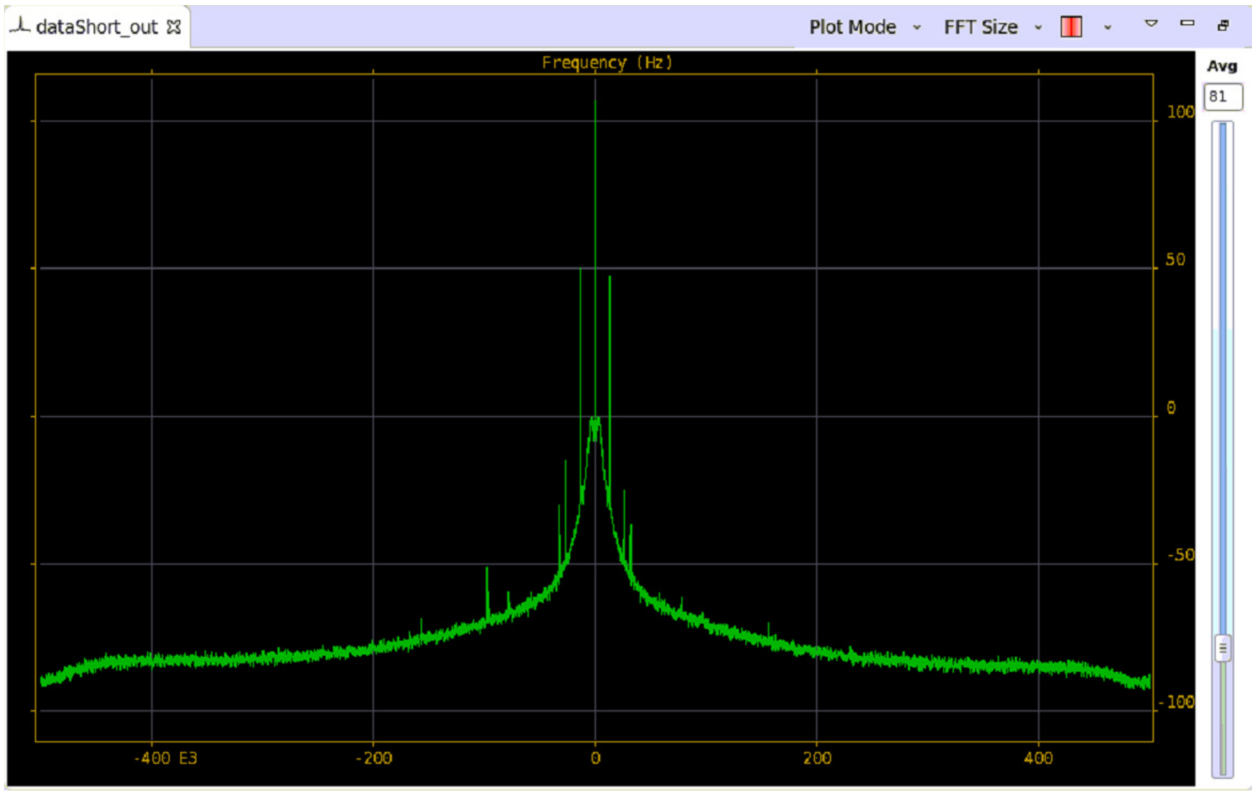


To see the data right click the dataShort_out and select Plot Port FFT.

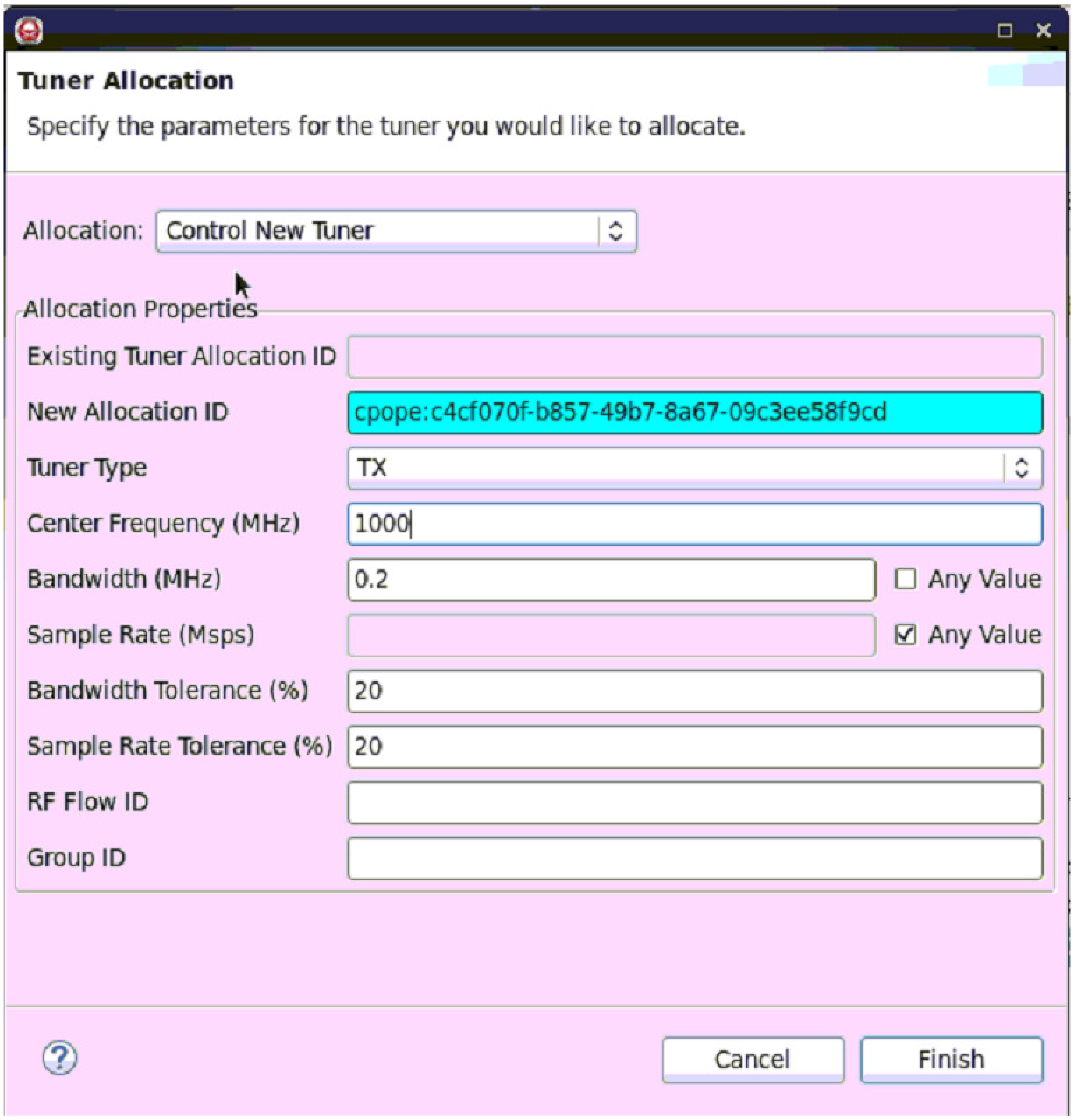


In this case the calibration signal from the radio is fed back to the tuner so it's possible to tune the calibration signal around by editing the calf property in the NDRFEI properties panel. Also, the

The plot has several configurations such as averaging that can be selected under the Advanced… settings.



For simplicity the transmitter can be verified by looping the transmit output back to the receiver that's already been configured. Under the Device Manager right click the Unallocated TX and select Allocate. The configuration is the same as for the receiver but choose a narrow bandwidth.



Tuner Allocation

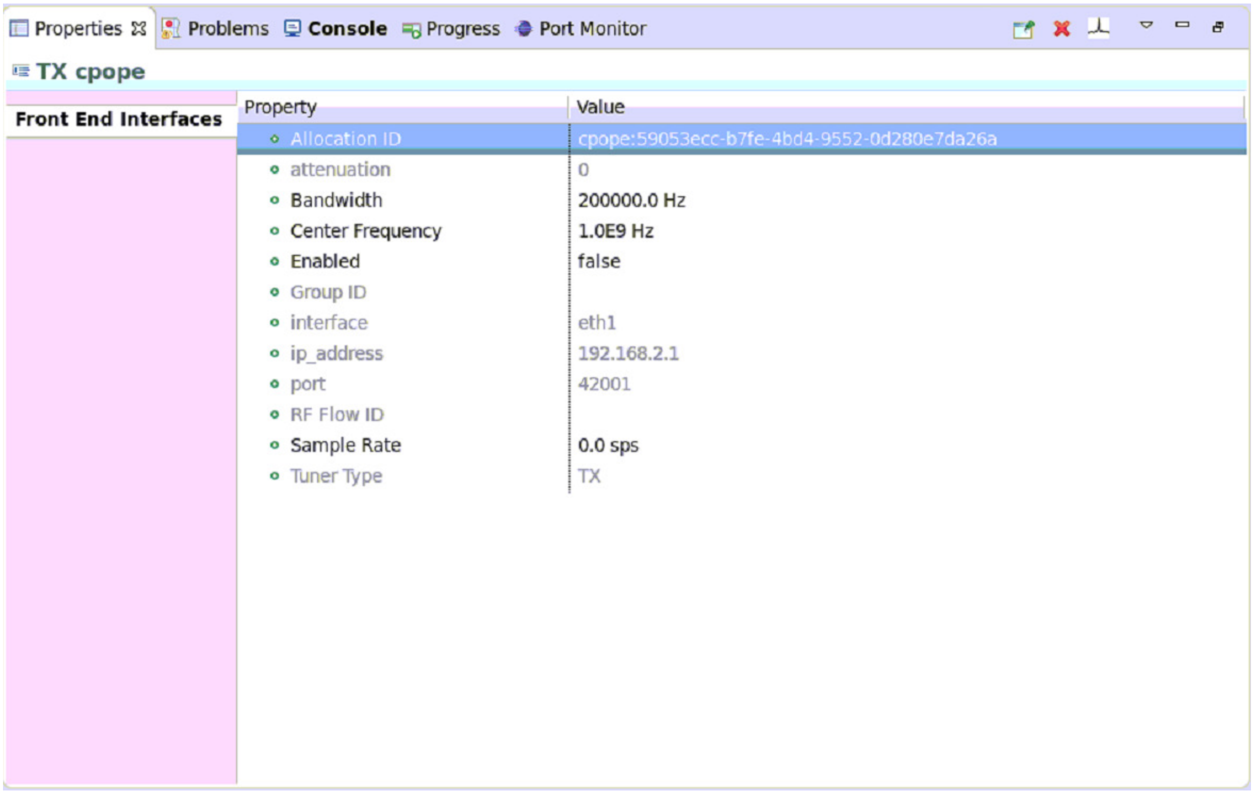
Specify the parameters for the tuner you would like to allocate.

Allocation: Control New Tuner

Allocation Properties

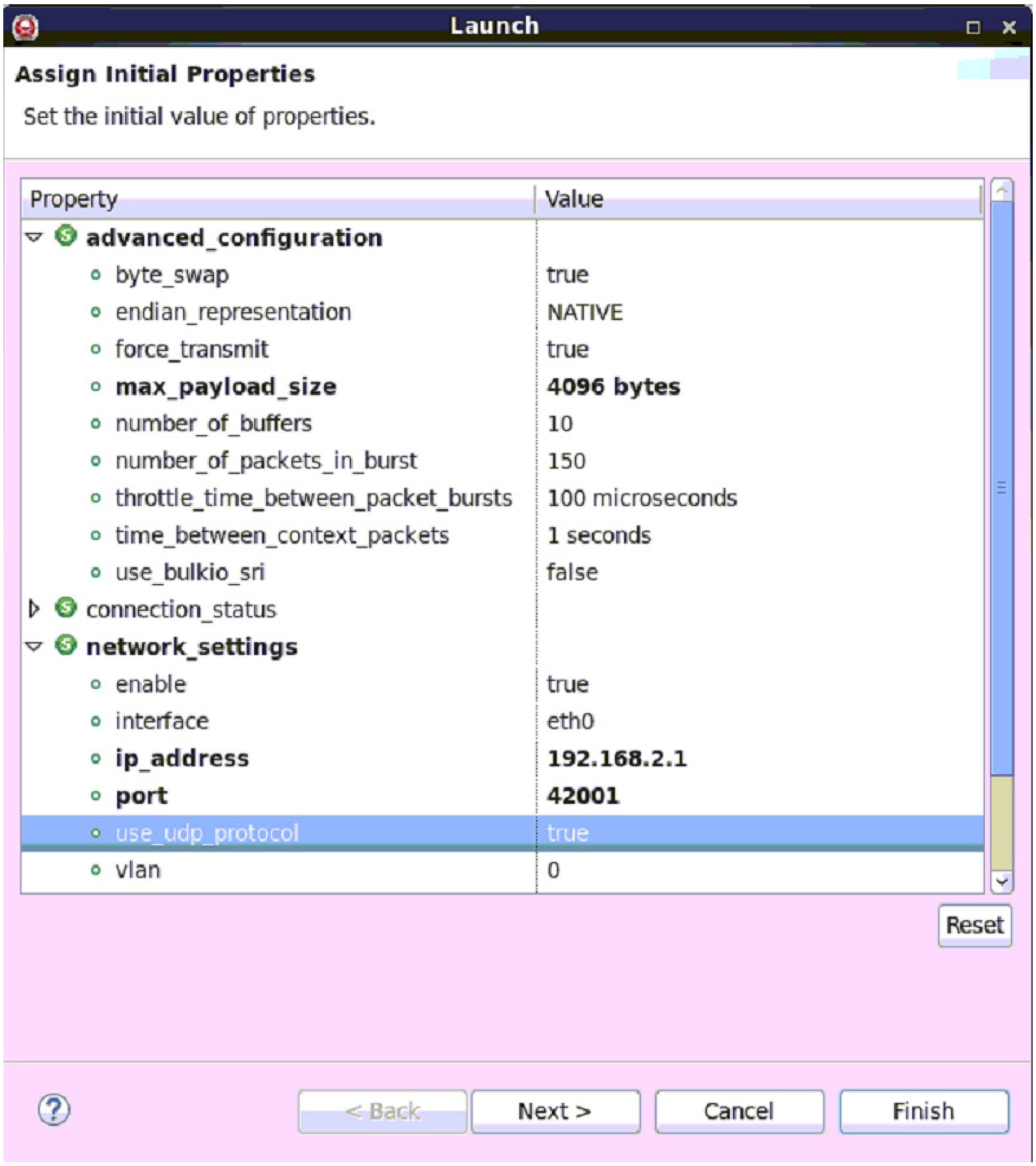
Existing Tuner Allocation ID		
New Allocation ID	cpope:c4cf070f-b857-49b7-8a67-09c3ee58f9cd	
Tuner Type	TX	
Center Frequency (MHz)	1000	
Bandwidth (MHz)	0.2	☐ Any Value
Sample Rate (Msps)		☑ Any Value
Bandwidth Tolerance (%)	20	
Sample Rate Tolerance (%)	20	
RF Flow ID		
Group ID		

The IP address and port number will be reported in the properties tab.



Property	Value
Allocation ID	cpope:59053ecc-b7fe-4bd4-9552-0d280e7da26a
attenuation	0
Bandwidth	200000.0 Hz
Center Frequency	1.0E9 Hz
Enabled	false
Group ID	
interface	eth1
ip_address	192.168.2.1
port	42001
RF Flow ID	
Sample Rate	0.0 sps
Tuner Type	TX

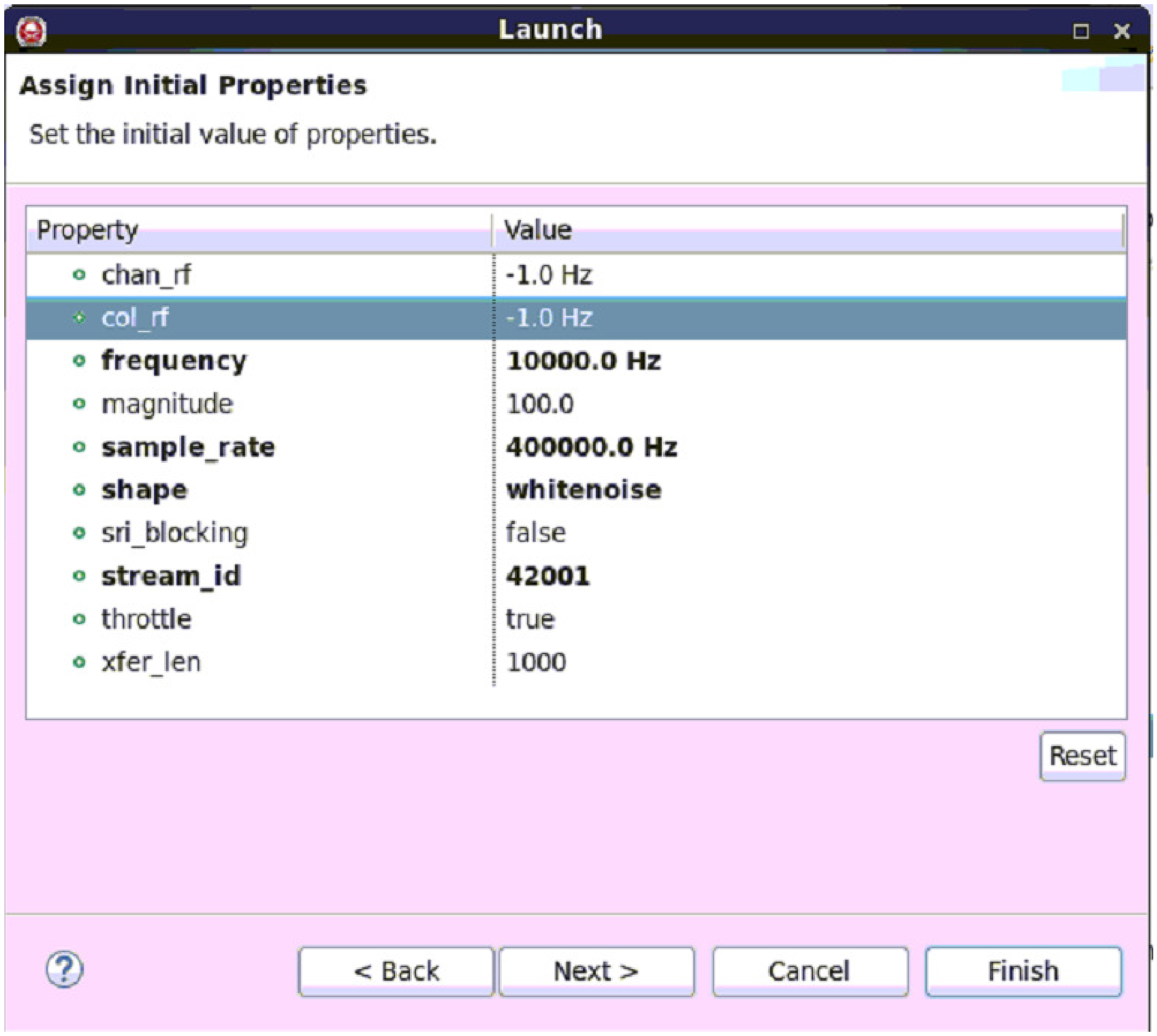
Next right click the rh.SinkVITA49 component and select Launch in Sandbox->Advanced. The max_payload_size, ip_address, and port need to be set correctly.


Launch
Assign Initial Properties
Set the initial value of properties.
Reset
< Back
Next >
Cancel
Finish

Property	Value
advanced_configuration	
byte_swap	true
endian_representation	NATIVE
force_transmit	true
max_payload_size	**4096 bytes**
number_of_buffers	10
number_of_packets_in_burst	150
throttle_time_between_packet_bursts	100 microseconds
time_between_context_packets	1 seconds
use_bulkio_sri	false
connection_status	
network_settings	
enable	true
interface	eth0
ip_address	**192.168.2.1**
port	**42001**
use_udp_protocol	true
vlan	0

Click Next, enable auto-start, and click Finish as before.

Any REDHAWK datasource could be used but the rh.SigGen is convenient. Right click the rh.SinkVITA49 component and select Launch in Sandbox->Advanced. Here the stream_id must be set to the UDP port number that was set in the rh.SinkVITA49 component (the stream ID and UDP port are the same). Also, since the rh.SigGen component outputs real data it should be set to a sample rate that is twice the radio's complex sample rate.



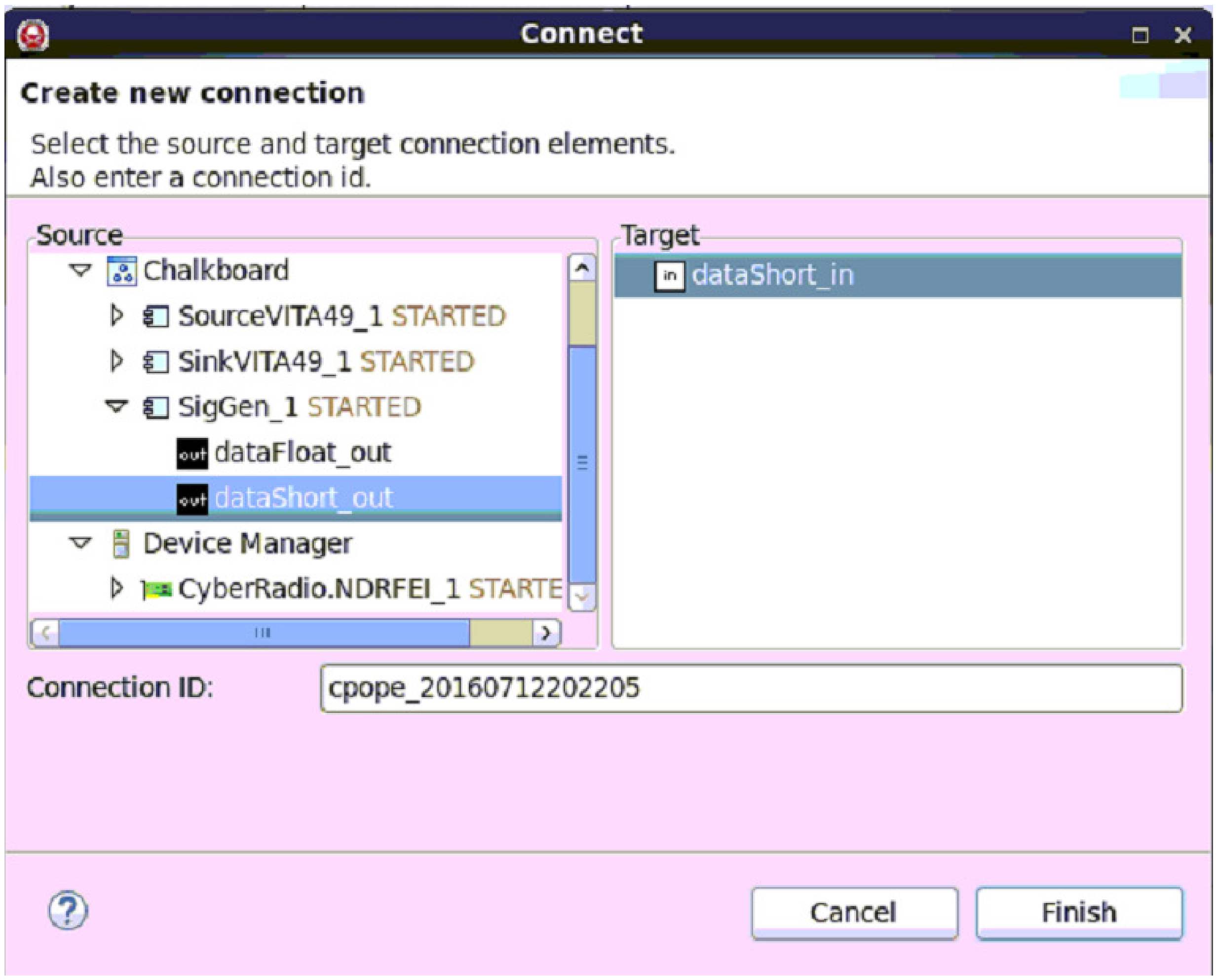
Assign Initial Properties

Set the initial value of properties.

Property	Value
chan_rf	-1.0 Hz
col_rf	-1.0 Hz
frequency	**10000.0 Hz**
magnitude	100.0
sample_rate	**400000.0 Hz**
shape	**whitenoise**
sri_blocking	false
stream_id	**42001**
throttle	true
xfer_len	1000

Click Next, Auto-start, and Finish as before.

Finally, the rh.SigGen output needs to be connected to the rh.SinkVITA49 input.


Connect

Create new connection

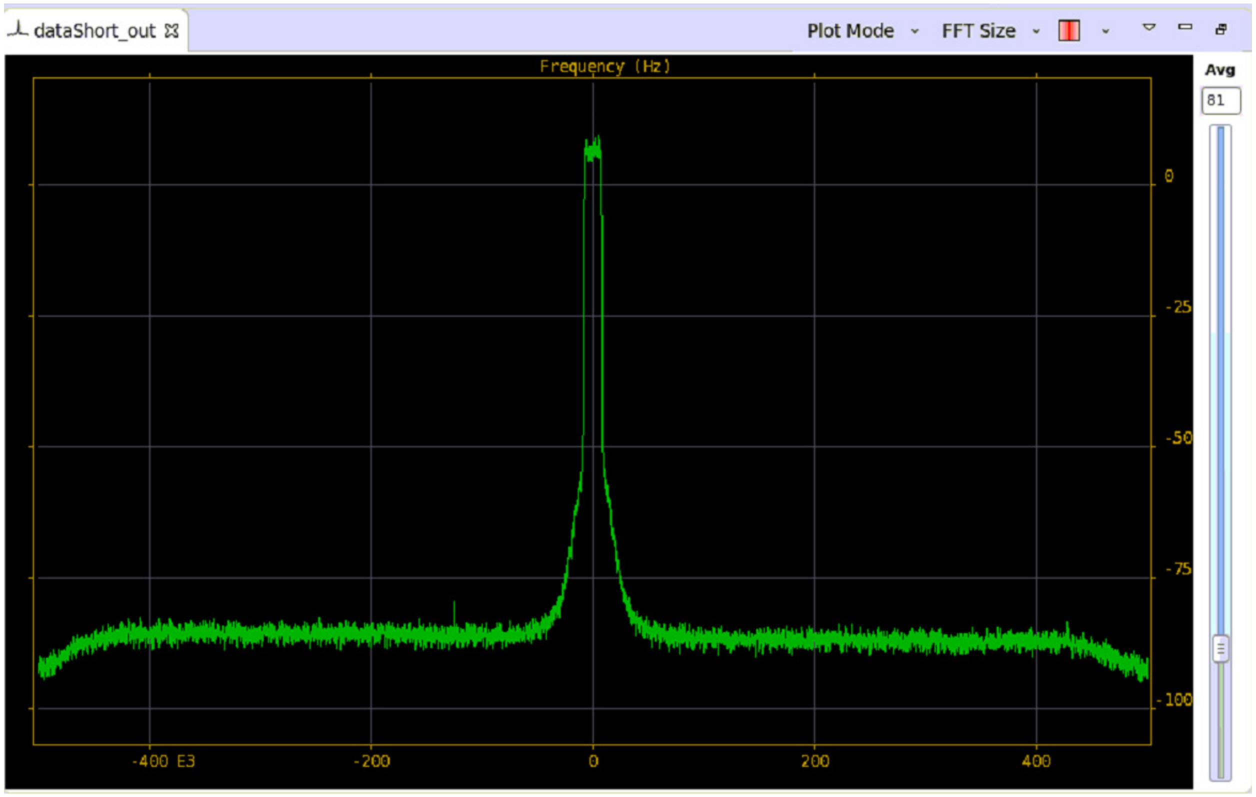
Select the source and target connection elements.
Also enter a connection id.

Source
- Chalkboard
  - SourceVITA49_1 STARTED
  - SinkVITA49_1 STARTED
  - SigGen_1 STARTED
    - dataFloat_out
    - dataShort_out
- Device Manager
  - CyberRadio.NDRFEI_1 STARTE

Target
- dataShort_in

Connection ID: cpope_20160712202205

Now, the transmit output should be visible on the receiver FFT plot (if they've been looped back correctly at the radio). In this case the signal is a 200 kHz whitenoise on a ~10 MHz wide plot display so the filtering skirts are clearly visible.



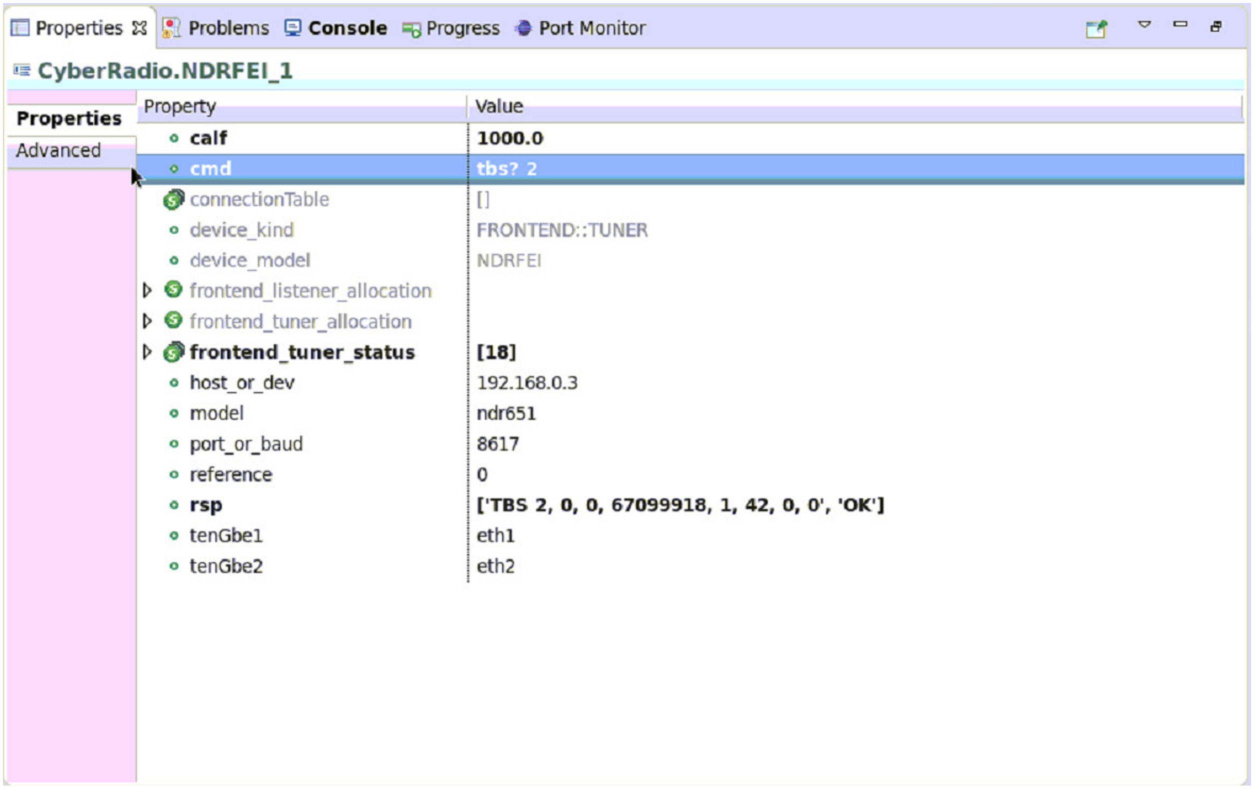
Note that each of the configurations above can be saved as a Run Configuration for easier launching. Also, they can be scripted easily in python programs. For example, the following python program will launch and connect the rh.SinkVITA49 component and rh.SigGen component. Since no display is needed for transmit it is easier to launch programmatically via the Python shell.

```
from ossie.utils import sb
snk= sb.launch("rh.SinkVITA49")
snk.api()
snk.network_settings.interface="eth1"
snk.network_settings.port=42001
snk.network_settings.ip_address="192.168.2.1"
snk.network_settings.enable=True
snk.network_settings.use_udp_protocol=True

snk.VITA49Encapsulation.enable_vrl_frames=True
snk.VITA49IFContextPacket.enable2=False
snk.advanced_configuration.force_transmit=True
snk.advanced_configuration.byte_swap=True
snk.advanced_configuration.max_payload_size=4096
snk.advanced_configuration.throttle_time_between_packet_bursts=0
snk.VITA49IFDataPacket.embed_time_stamp=True
snk.VITA49IFDataPacket.enable_class_identifier=True
snk.VITA49IFDataPacket.enable_stream_identifier=True
snk.VITA49IFDataPacket.enable_trailer=True
```

```
siggen = sb.launch("rh.SigGen")
siggen.stream_id="42001"
siggen.sample_rate=3.2e6
siggen.shape="whitenoise"
siggen.xfer_len=4096
siggen.frequency=10e3
siggen.api()
siggen.connect(snk,usesPortName="dataShort_out")
sb.start()
```

Finally, it's possible to send low level commands to the radio via the cmd property of the CyberRadio.NDRFEI device. One useful command queries the transmit buffer status so the amount of data reaching the radio can be monitored and the transmit data rate can be adjusted accordingly.



Property	Value
calf	1000.0
cmd	tbs? 2
connectionTable	[]
device_kind	FRONTEND::TUNER
device_model	NDRFEI
frontend_listener_allocation	
frontend_tuner_allocation	
frontend_tuner_status	[18]
host_or_dev	192.168.0.3
model	ndr651
port_or_baud	8617
reference	0
rsp	['TBS 2, 0, 0, 67099918, 1, 42, 0, 0', 'OK']
tenGbe1	eth1
tenGbe2	eth2