REDHAWK Installation and Usage

V2.0

July 22, 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 ARP Configuration

For transmit operations the system must know the Ethernet MAC address of the receive side of the 10 gbe interfaces. Unfortunately, the radios do not respond to ARP queries from the system so the ARP table must be modified manually. This only has to be done once at boot time and can be easily scripted.

First to determine the MAC addresses send the #mac? command to the radio:

```
[ndruser@ndr-demo-1032 ~]$ telnet 192.168.0.3 8617
```

```
Trying 192.168.0.3...
Connected to 192.168.0.3.
Escape character is '^]'.
Connected to NDR651.  You are user #3
#mac?
#MAC 1, 00:50:C2:F5:91:1F
#MAC 2, 00:50:C2:F5:91:20
```

Then to edit the arp table send the following commands (with root permissions):

```
[ndruser@ndr-demo-1032 ~]$ cat setarp.sh
arp -s 192.168.2.1 00:50:c2:f5:91:1f
arp -s 192.168.1.1 00:50:c2:f5:91:20
arp -a -n
```

For convenience a script, setarp.sh, is available to be copied/edited as needed.

Note these commands must be modified/resent whenever another radio is used since they MAC addresses are unique to the radio Ethernet ports.

1.1.4 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']
```

1.1.5 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
```

1.1.6 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.

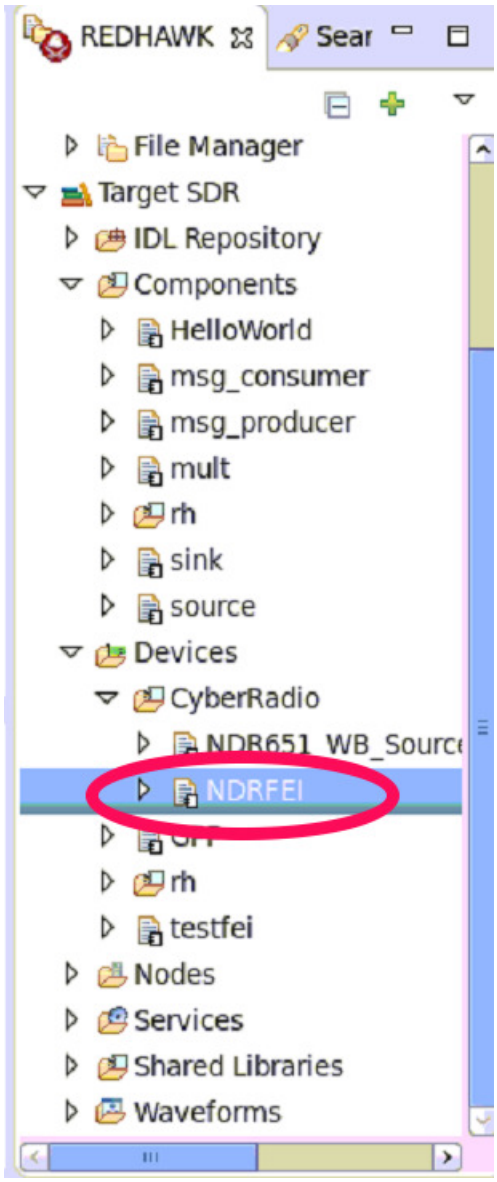
To manually install the rpms run:

```
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
```

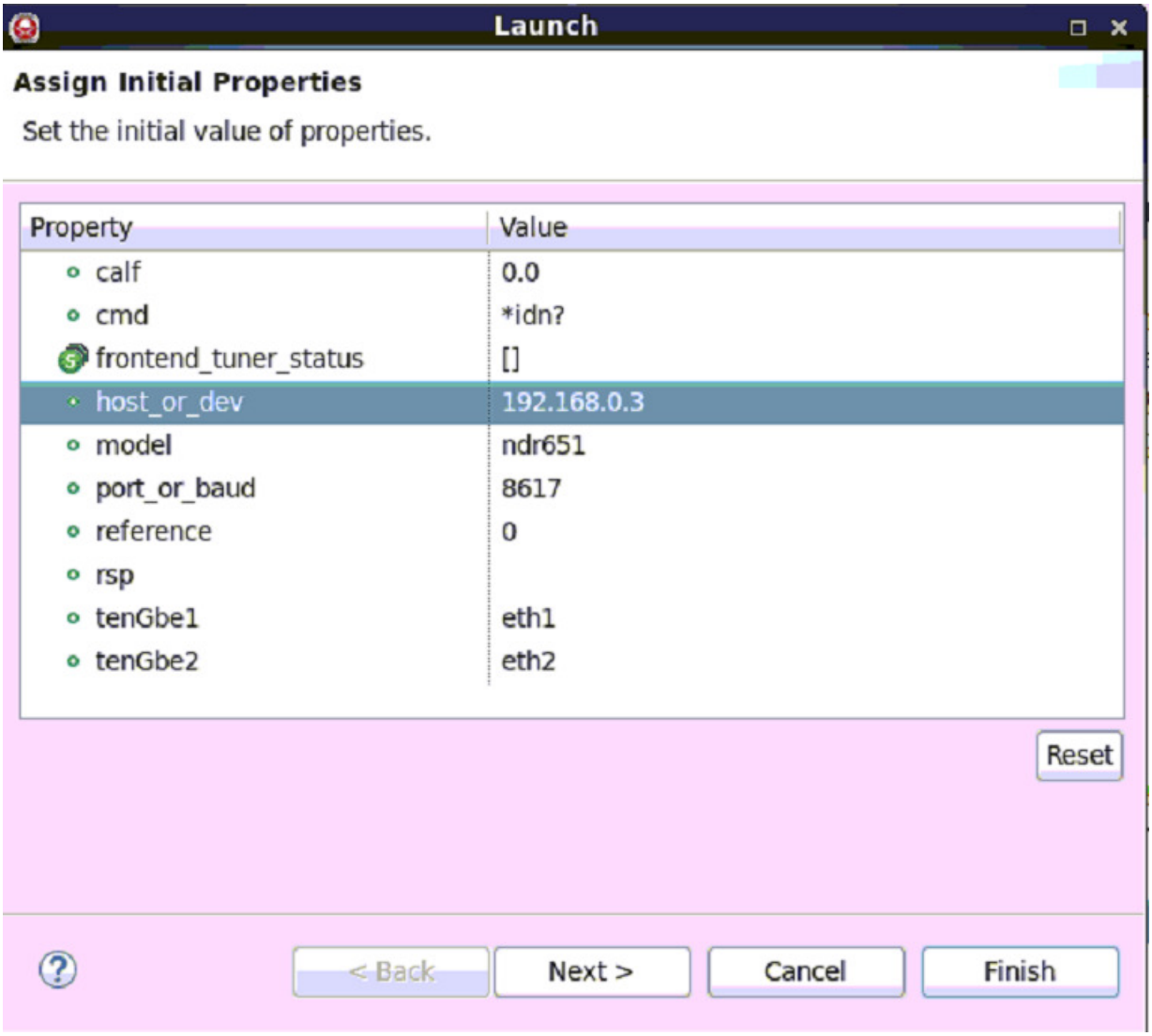
1.2 Usage

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.

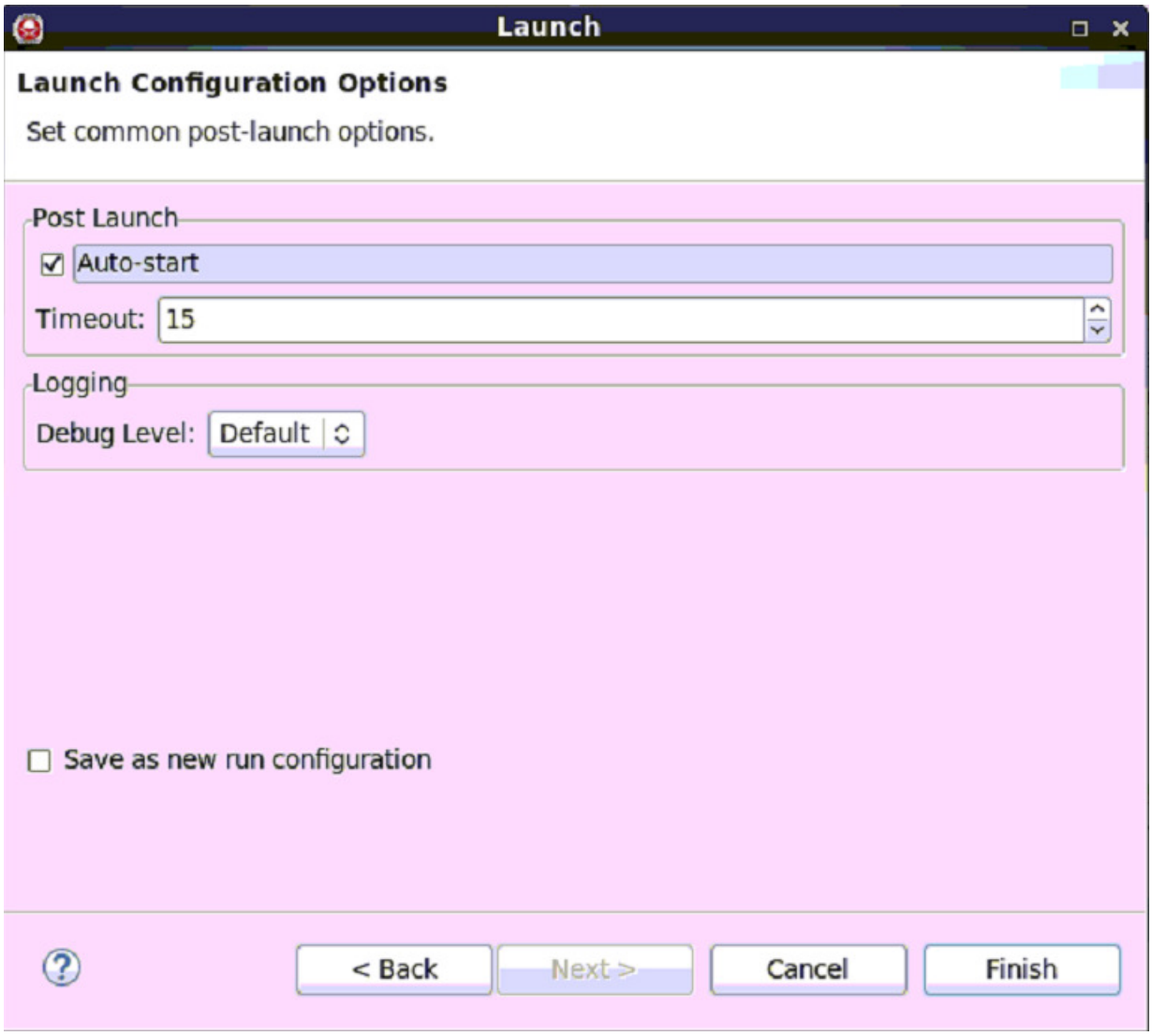
The Network Defined Radio Front End Interface (NDRFEI) is available in the REDHAWK IDE under Target SDR->Devices->CyberRadio.



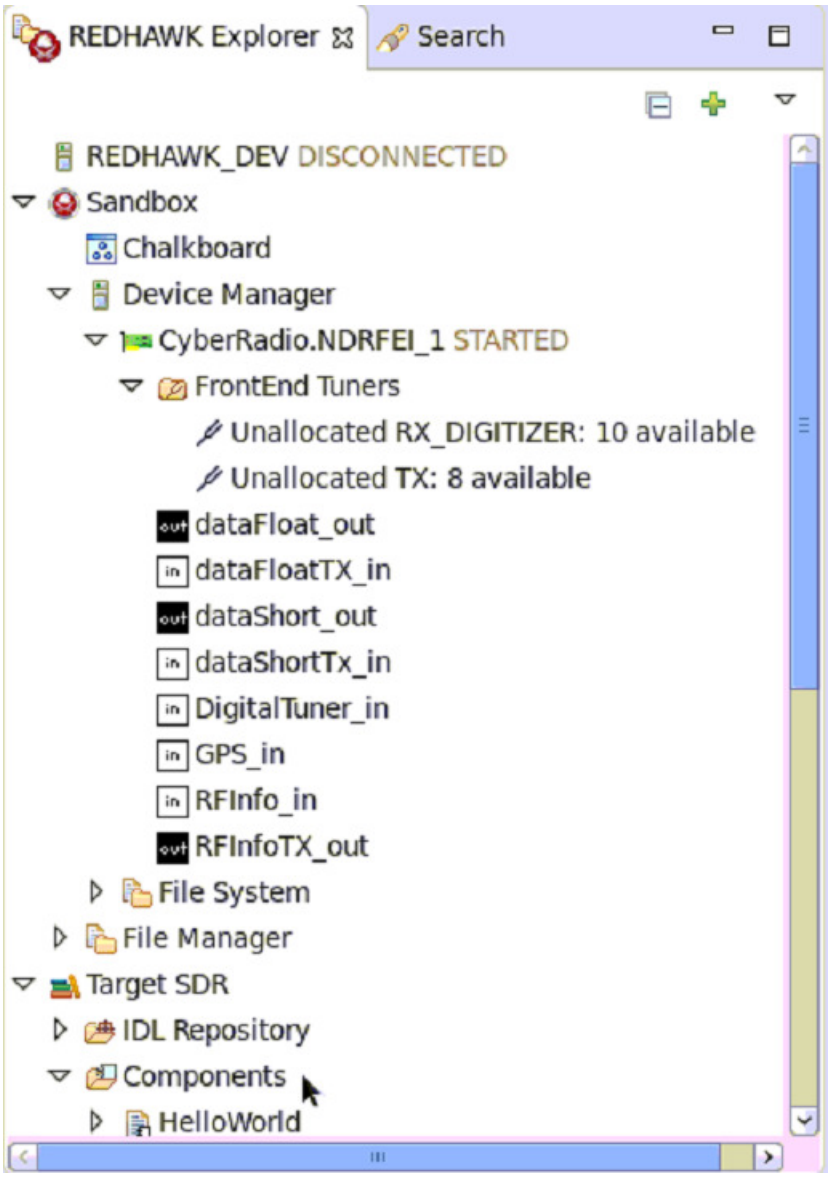
Right clicking the device and selecting Launch in Sandbox->Advanced from the pop-up menu allows the parameters to be configure. The primary settings are the IP address and port number of the control interface to the radio, the radio model, and the local Ethernet interfaces of the two 10 gigabit data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



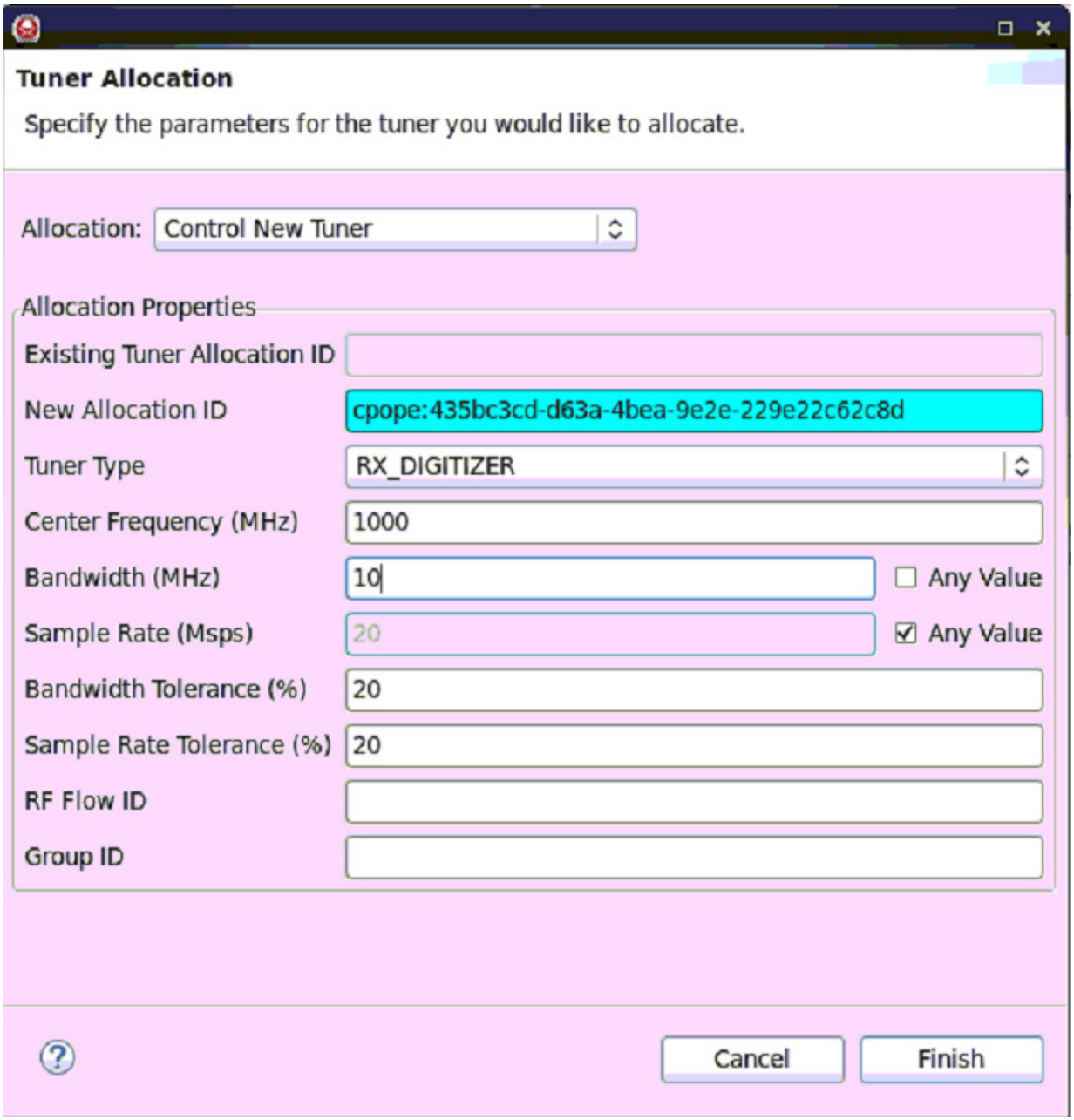
The next Tab allows the modulation to start automatically when launched then click Finish.



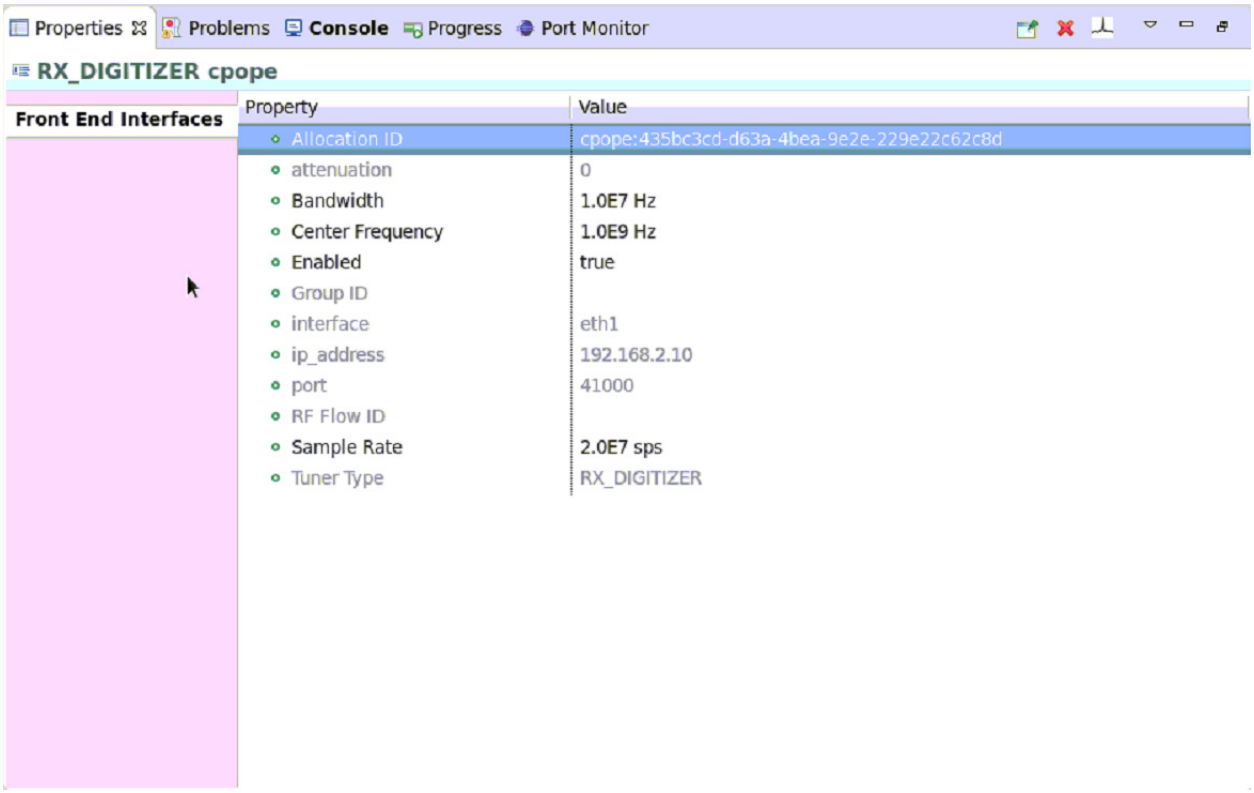
Once running the device will automatically contact the radio and determine the number of available transmitters and receivers to populate the FrontEnd Tuners listing.



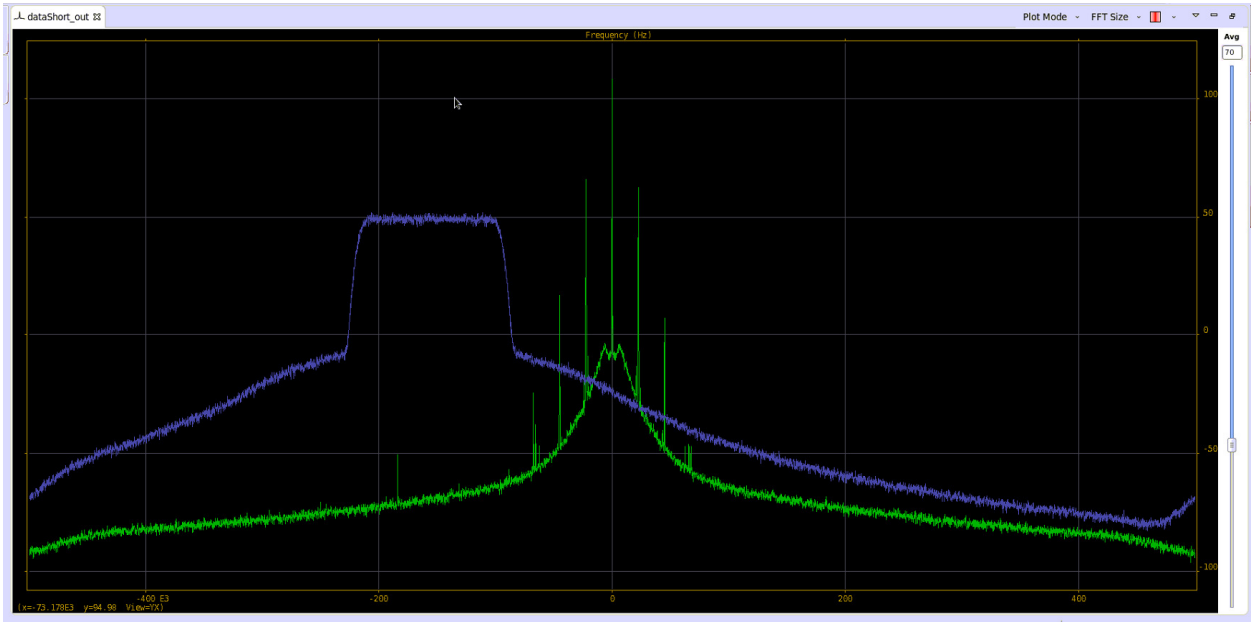
Right click an RX_DIGITIZER and select Allocate to bring up a configuration panel. The available bandwidths/sample rates and tuning rangers will vary from radio to radio. The IDE will report an exception if an unsupported request is made.



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.

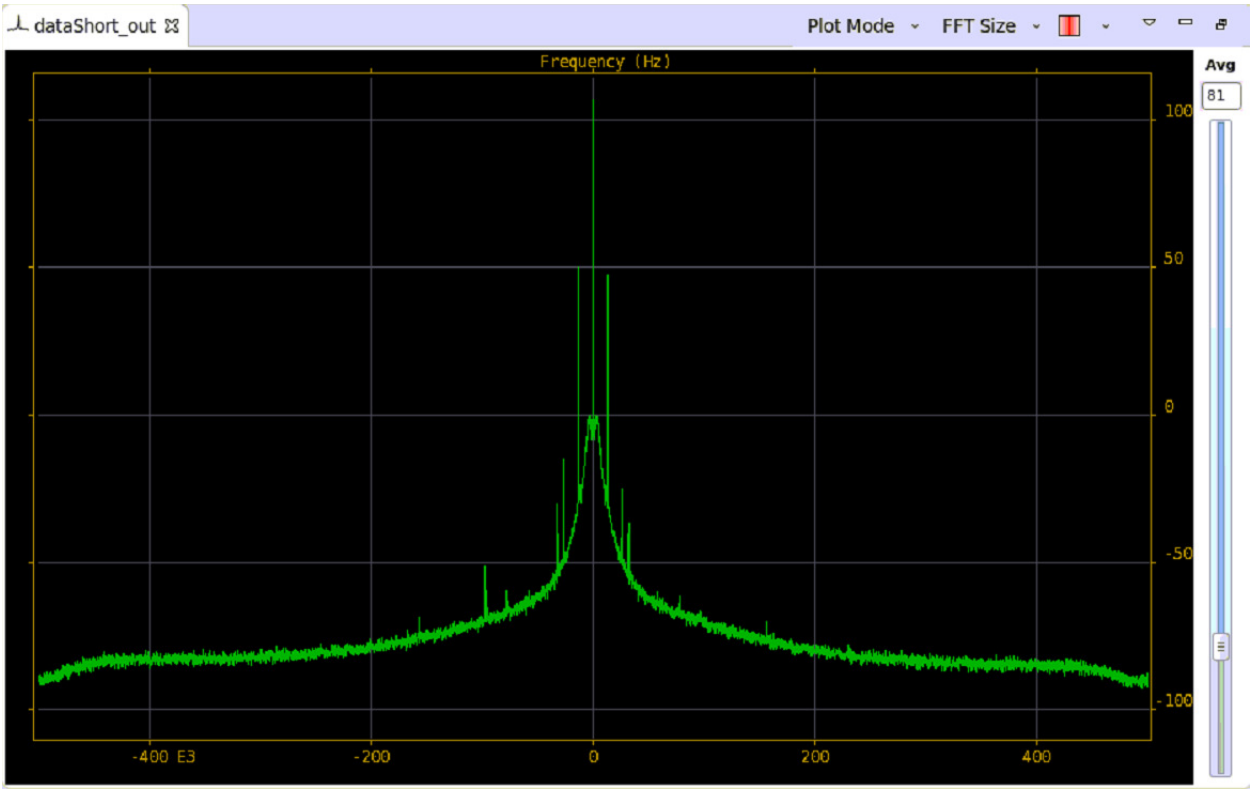


It is important to note (or set) the Allocation ID of the RX_DIGITIZER as this is used to distinguish the dataflows from multiple receivers that use the same data port. Right click on dataShort_out port and select “Plot Port FFT”. A graph showing all the current streams will appear.

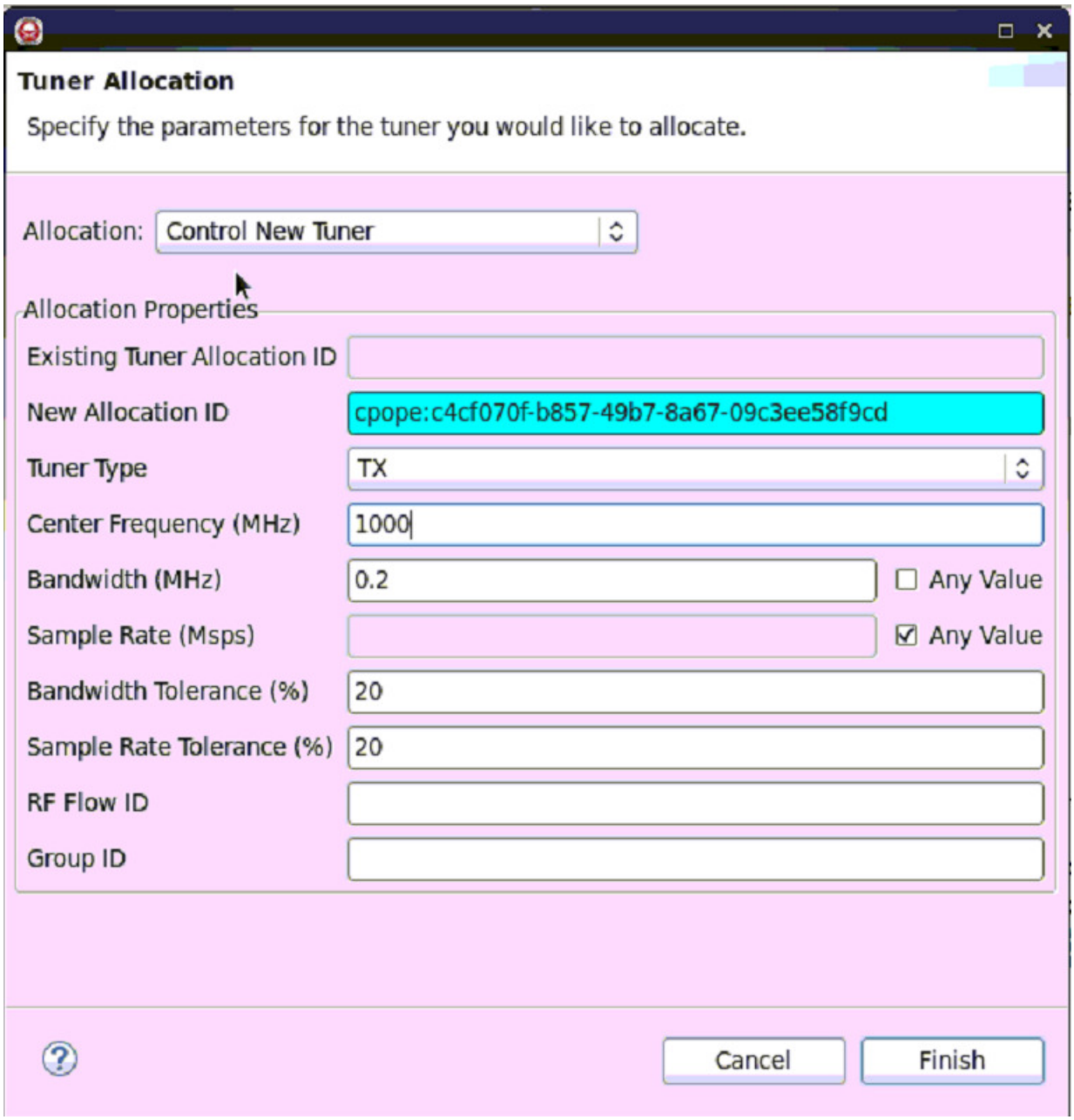


In the above the first receiver (in green) is showing the calibration signal which is looped back so it's possible to tune the calibration signal around by editing the calf property in the NDRFEI properties panel. The blue trace is the second receiver which is showing the second transmitter looped back.

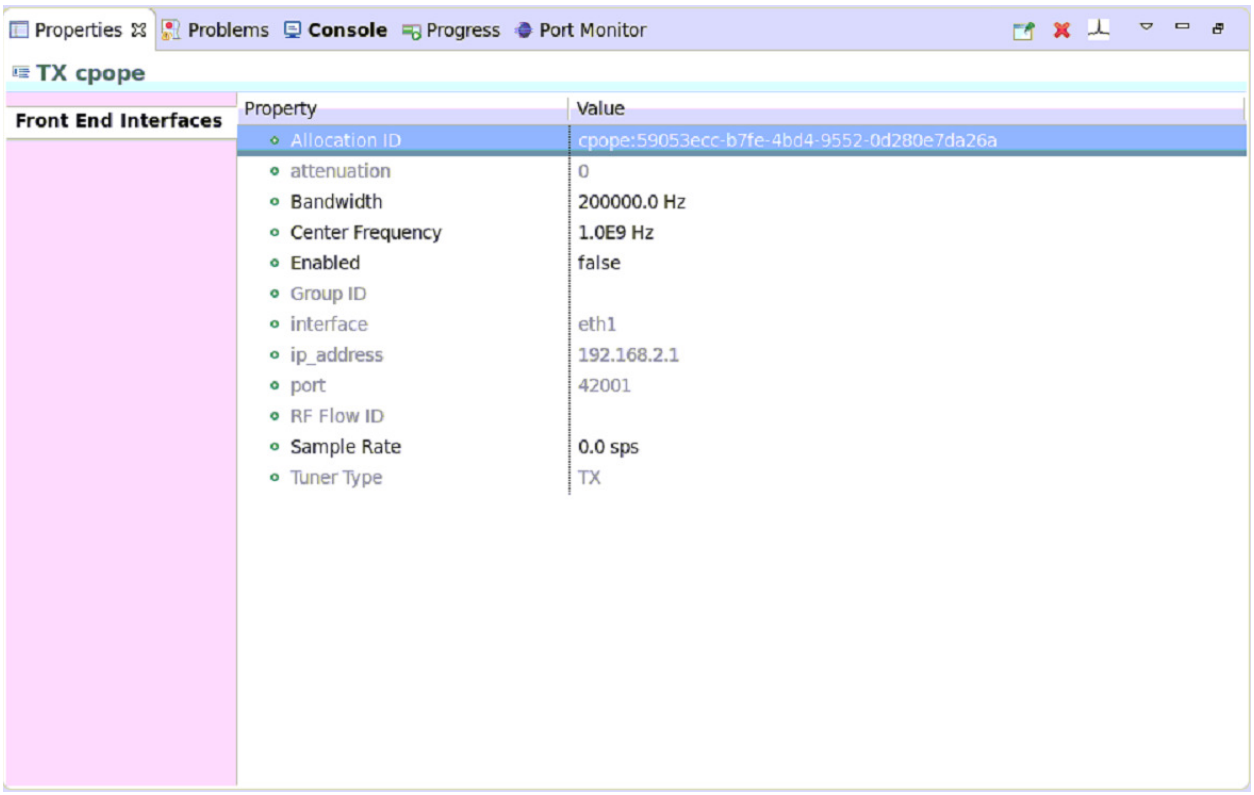
The plot has several configurations such as averaging that can be selected under the Advanced… settings.



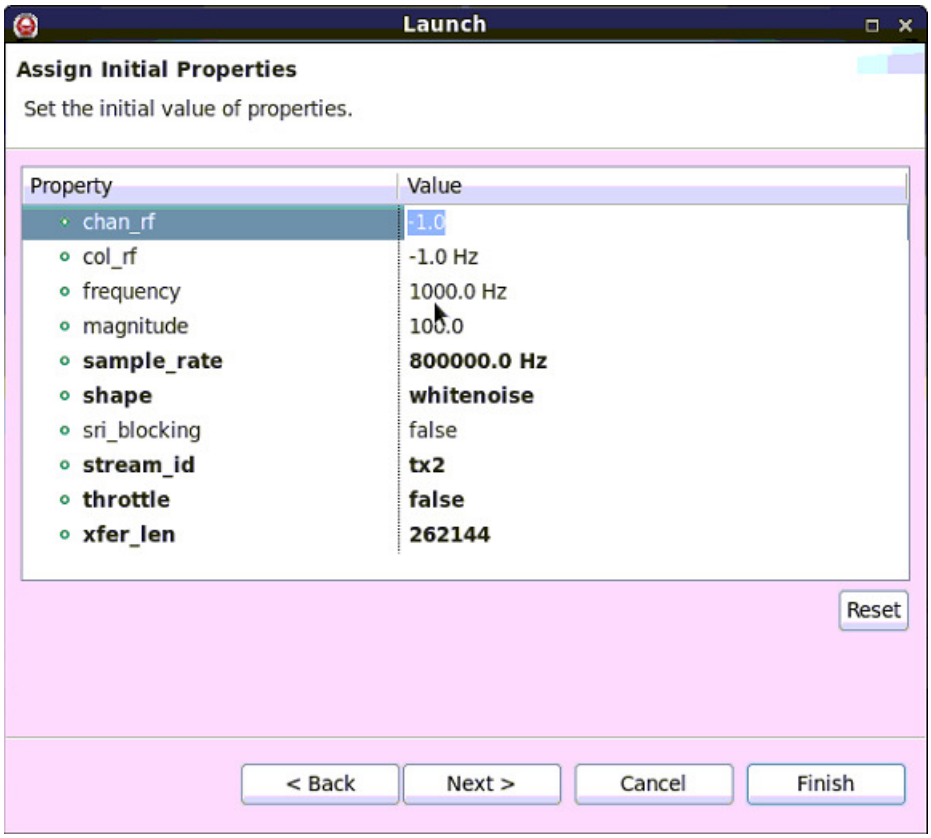
For simplicity the transmitter can be verified by looping the transmit output back to the receiver that has already been configured. Under the Device Manager right click the Unallocated TX and select Allocate. The configuration is the same as for the receiver but choose a narrow bandwidth.



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

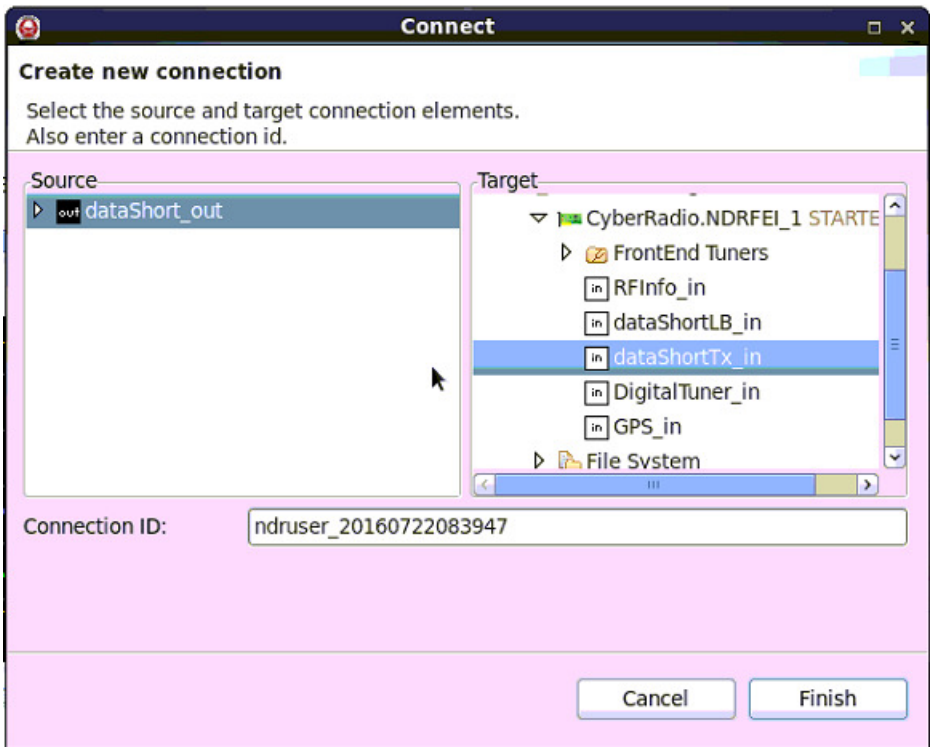


Any REDHAWK datasource could be used but the rh.SigGen is convenient. Right click the rh.SigGen component and select Launch in Sandbox->Advanced. Here the stream_id must be set to the Allocation_ID of the target transmitter. It is easier to assign simpler Allocation_IDs when the transmitter is launched than to work with the auto generated UUIDs set by REDHAWK. In this case the second transmitter Allocation_ID was set to tx2. 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. Also, to maximize throughput and minimize the load on the system a large transfer size (2^18 bytes) should be used.

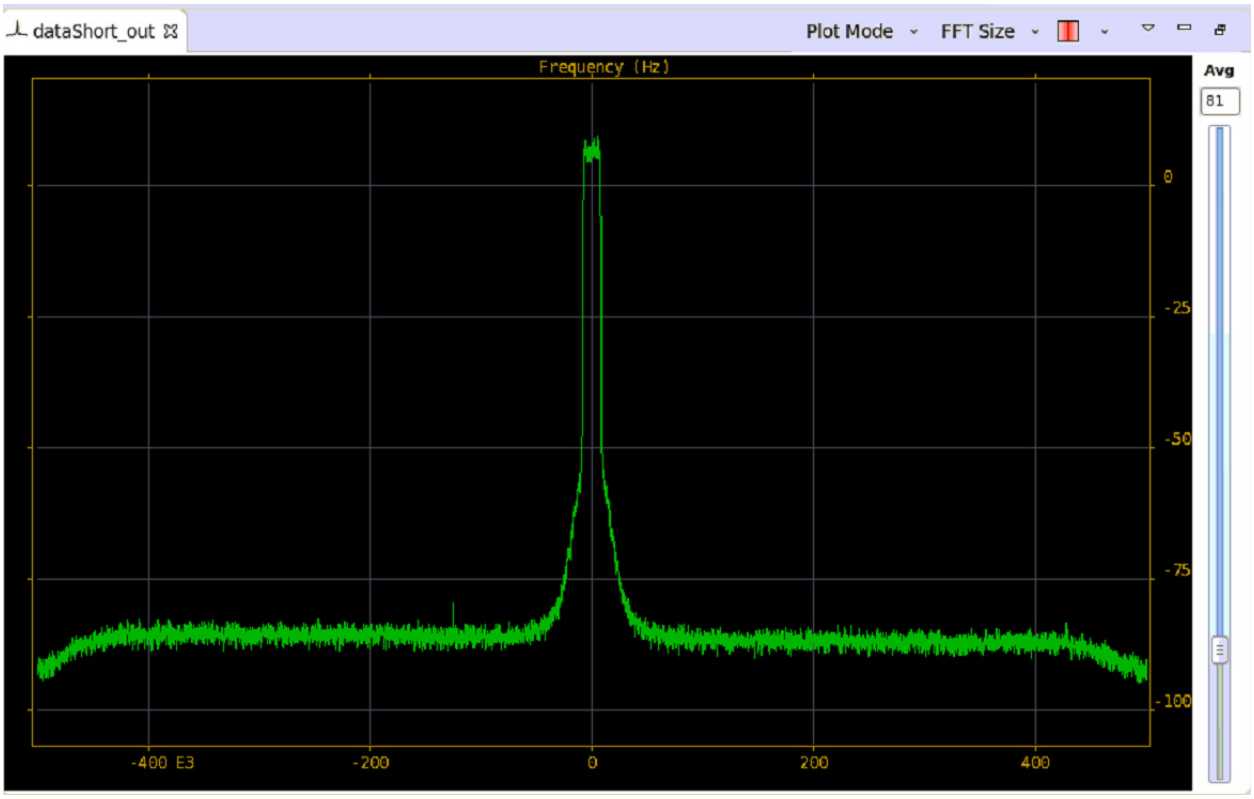


Click Next, Auto-start, and Finish as before.

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

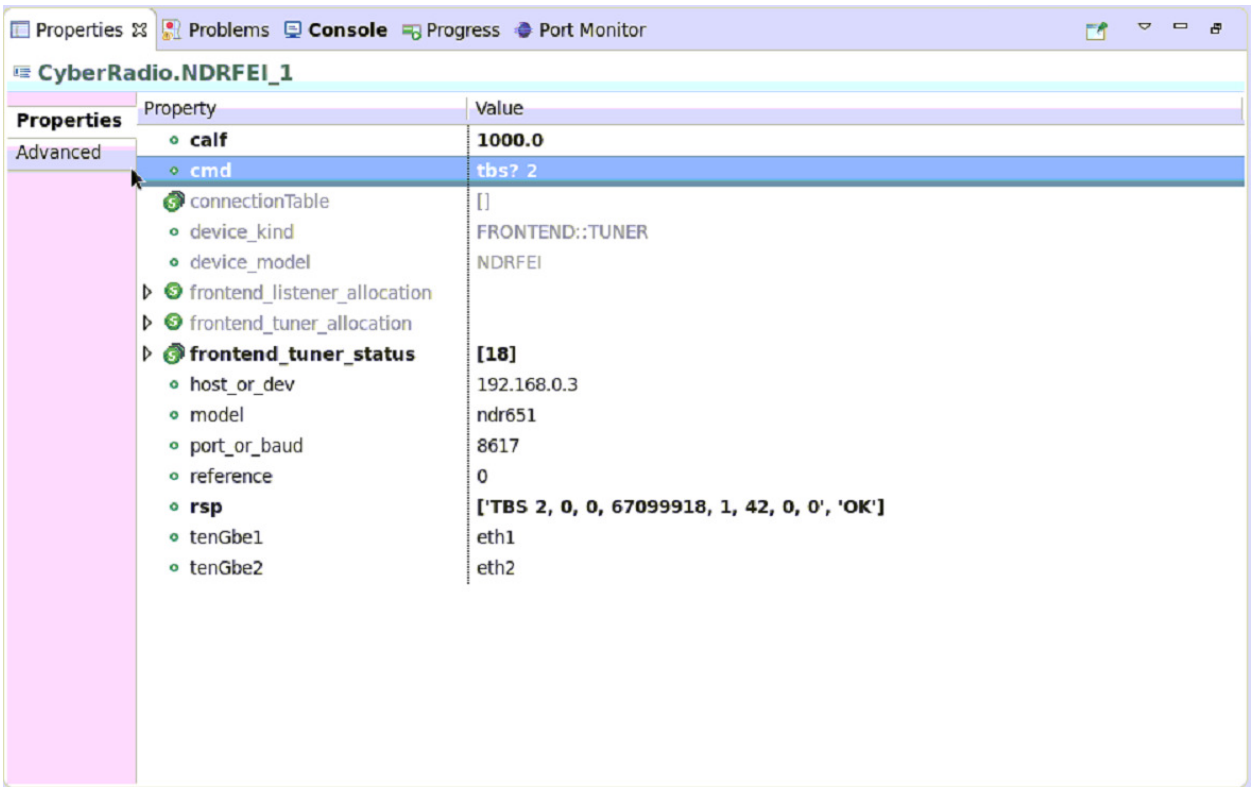


Now, the transmit output should be visible on the receiver FFT plot (if they've been looped back correctly at the radio). In this case the signal is a 200 kHz whitenoise on a ~10 MHz wide plot display so the filtering skirts are clearly visible.



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.

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.


Properties
Problems
Console
Progress
Port Monitor
CyberRadio.NDRFEI_1
Properties
Advanced
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