

Streaming processed data from the E31x with GNU Radio and ZMQ

Contents

- 1 Application Note Number
- 2 Revision History
- 3 Abstract
- 4 Overview
- 5 Required Hardware
- 6 Application Overview
- 7 E310 flow graph
- 8 Host flow graph
- 9 Generating Python executable from flow graph
- 10 Verify USRP E310 connectivity
- 11 Copy Python file to E310
- 12 SSH into the E310
- 13 Running the flow graph on E310
- 14 Running the flow graph on the host
- 15 Stopping the application
- 16 Changing IP Address
- 17 Downloads

AN-335

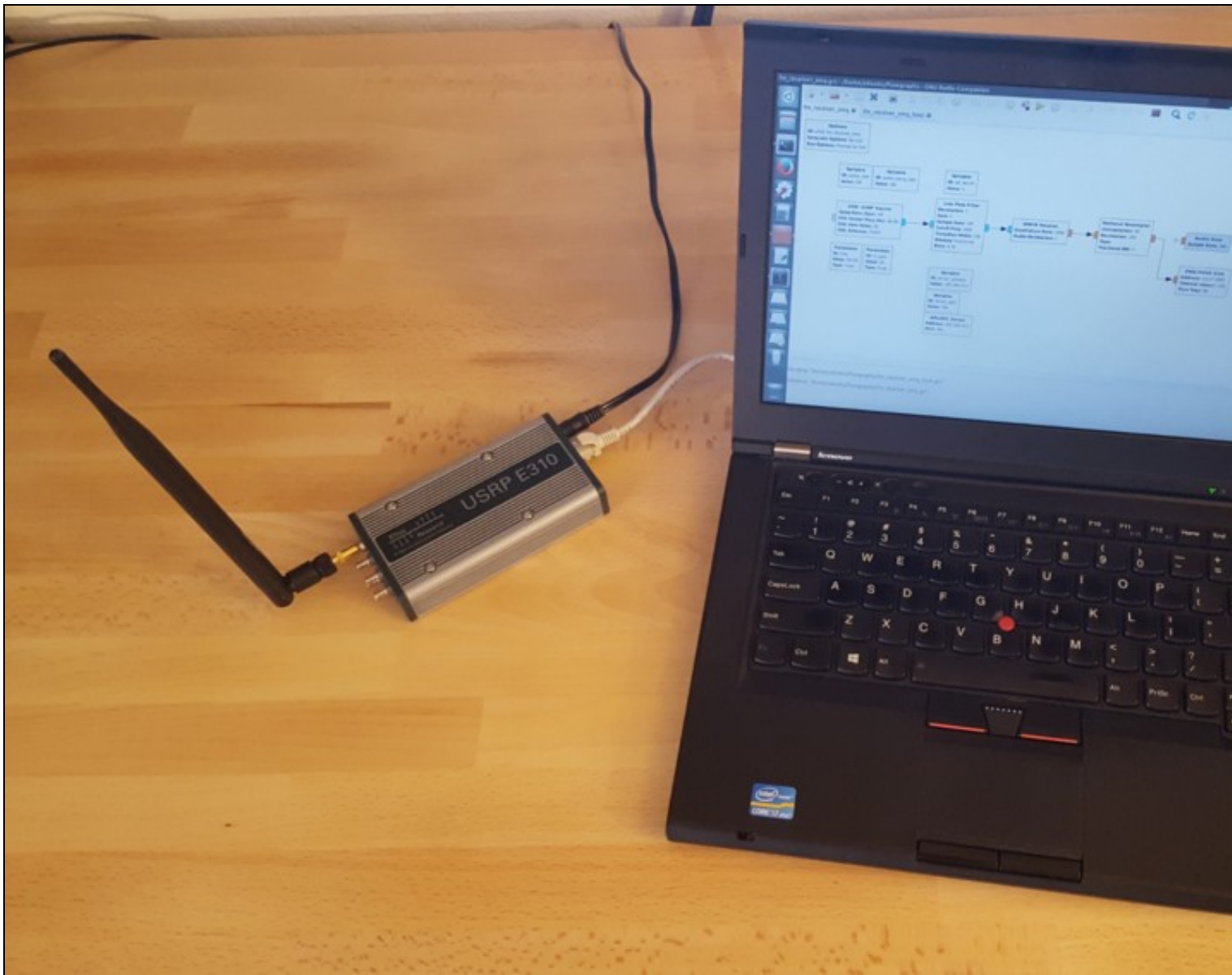
This application note will demonstrate using the USRP E310 to remotely stream processed data to a host machine.

The USRP E310 being an embedded USRP is designed to process data locally, unlike other USRP models which will stream samples to the host machine for processing. The USRP E310 is built around the Analog Devices AD9361 transceiver and Xilinx Zynq 7020 FPGA, which includes an ARM processor. Running an Open Embedded Linux distribution it supports UHD, GNU Radio and RFNoC for processing.

This application note will focus on using GNU Radio to bring in the stream of samples and perform the digital signal processing to demodulate a commercial wide-band frequency modulated radio station (WBFM) locally on the USRP E310. The demodulated audio stream will then be sent via the ZeroMQ (ZMQ) interface to a host machine, where it then will be played on the host's audio system / speakers. This application note will also demonstrate using the XMLRPC blocks of GNU Radio to adjust variables within a running flow graph on the remote E310.

- Ethernet cable
- Antenna
- USRP E310/E312/E313
- Host computer with GNU Radio installation

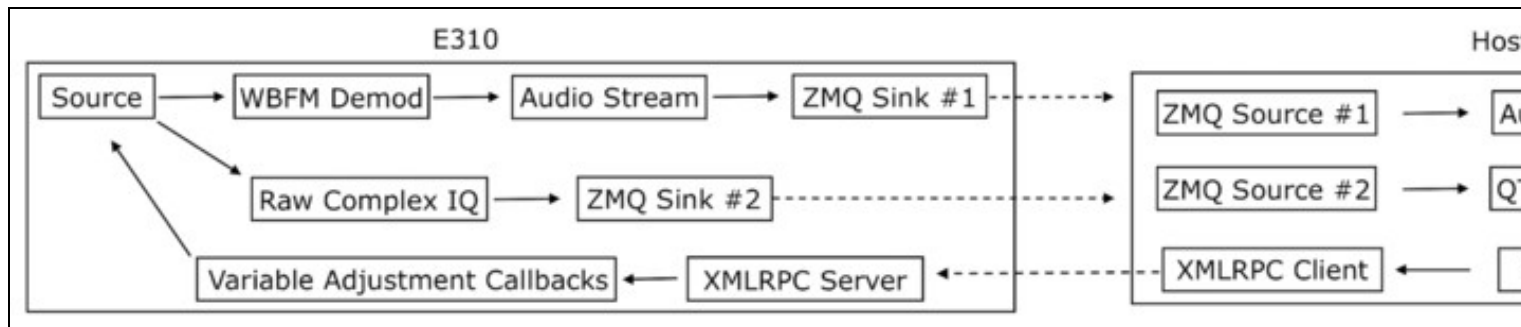
Note: The GNU Radio installation will need to have the `gr-zeromq` component enabled. For details on installing GNU Radio, please see the step-by-step guides at the Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on [Linux](#), [OS X](#) and [Windows](#) Application Notes.



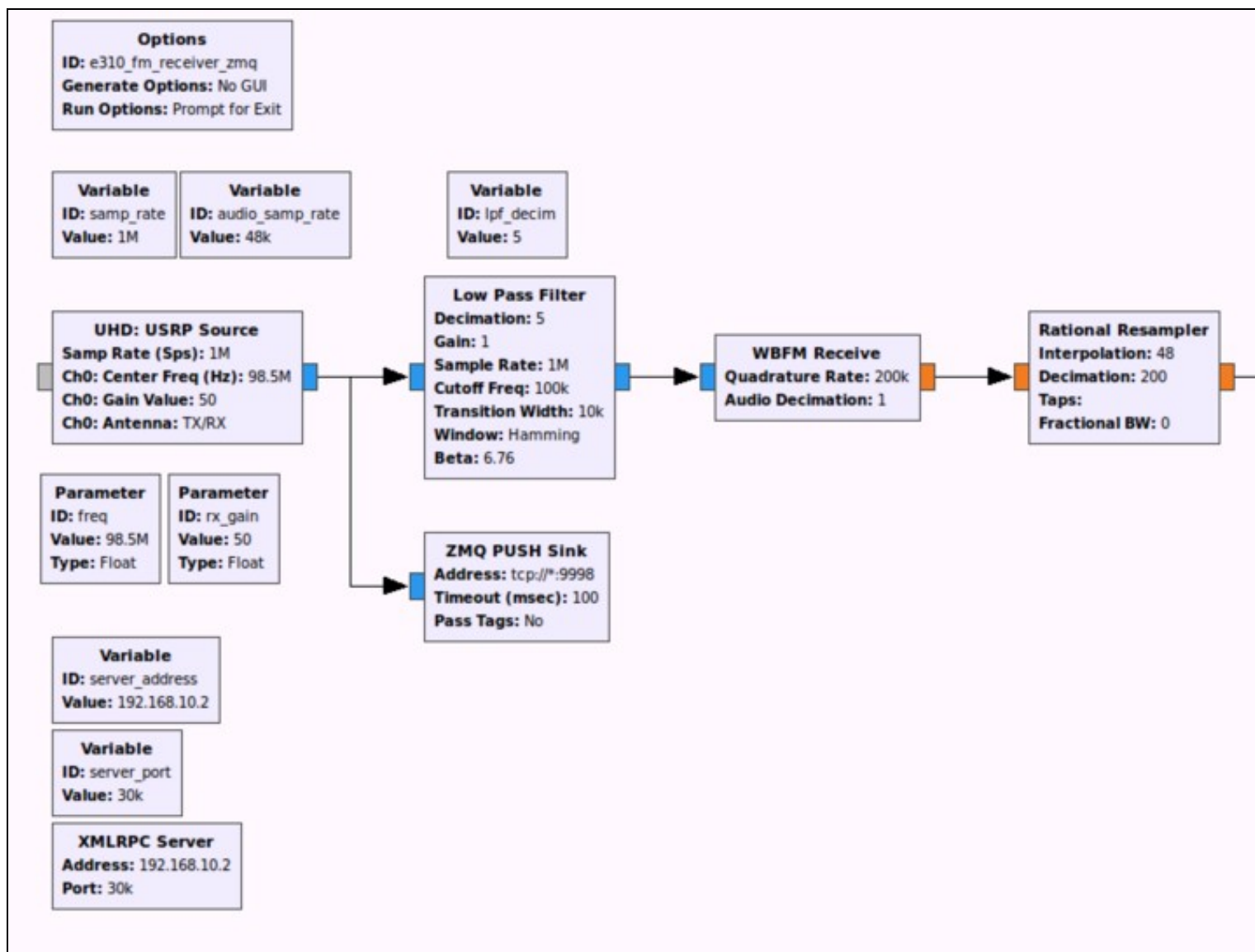
The block diagram below shows the high-level overview of the complete application. The flow graph running on the USRP E310 will demodulate a WBFM radio station, and then stream the audio over a ZMQ Sink to the host computer, where it will then be played over the host's speakers via the Audio Sink.

This application will also stream the complex IQ samples to the host via a second ZMQ Sink to produce a FFT on the host computer for visual reference only.

On the GNU Radio flow graph that is running on the host computer, there are QT GUI widgets to adjust the Frequency and RF Gain, that will be sent via XMLRPC commands to the E310, to adjust the variables within the running flow graph. The Audio Gain GUI widget will locally adjust the audio volume on the host computer.



The figure below shows the flow graph that will run on the E310. There are several important settings to note.



Within the Options block:

- Generate Options: No GUI
- Run Options: Prompt for Exit

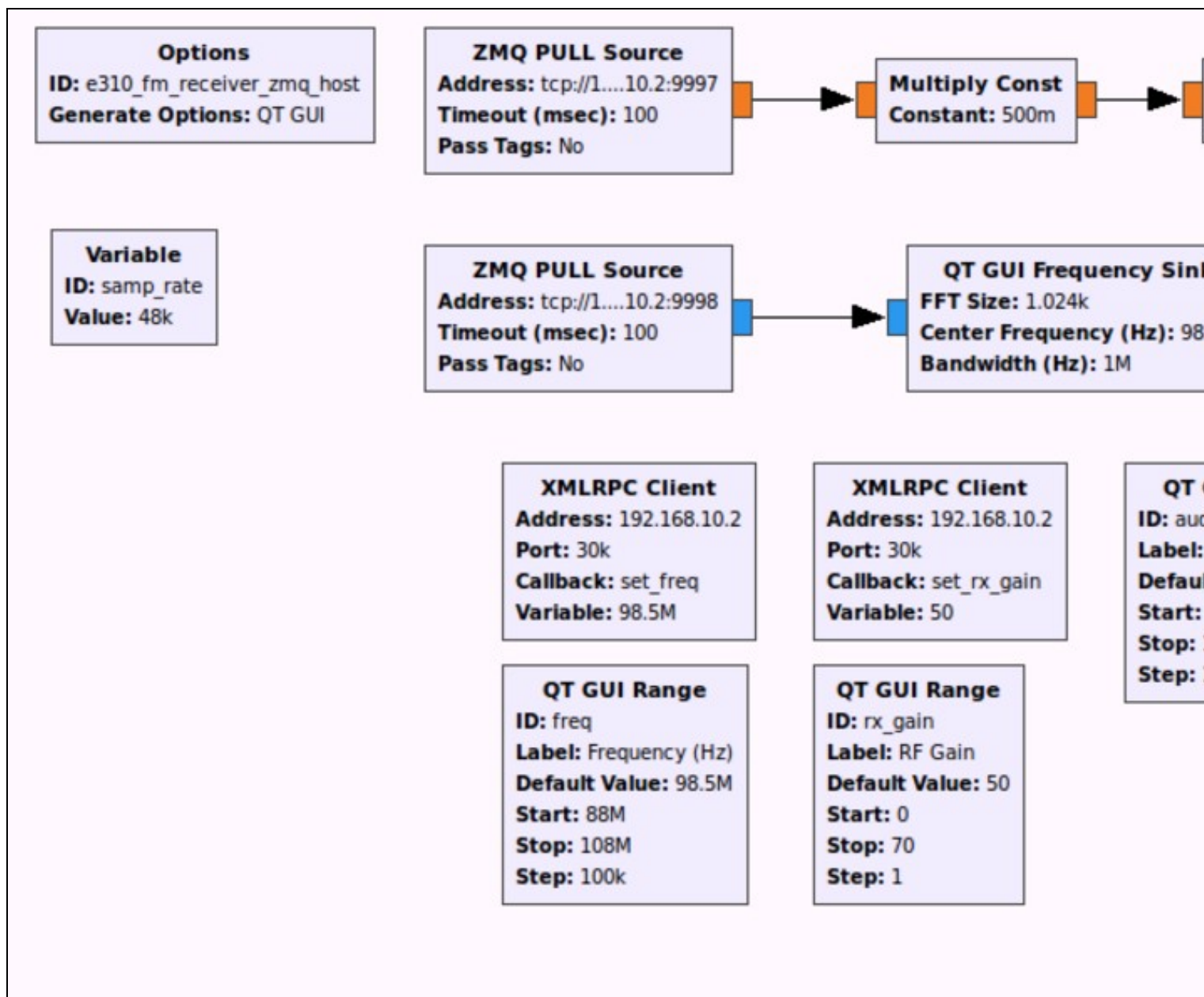
Properties: Options

General Advanced Documentation

ID	e310_fm_receiver_zmq
Title	
Author	
Description	
Canvas Size	1600, 1200
Generate Options	No GUI
Run Options	Prompt for Exit
Max Number of Output	0
Realtime Scheduling	Off

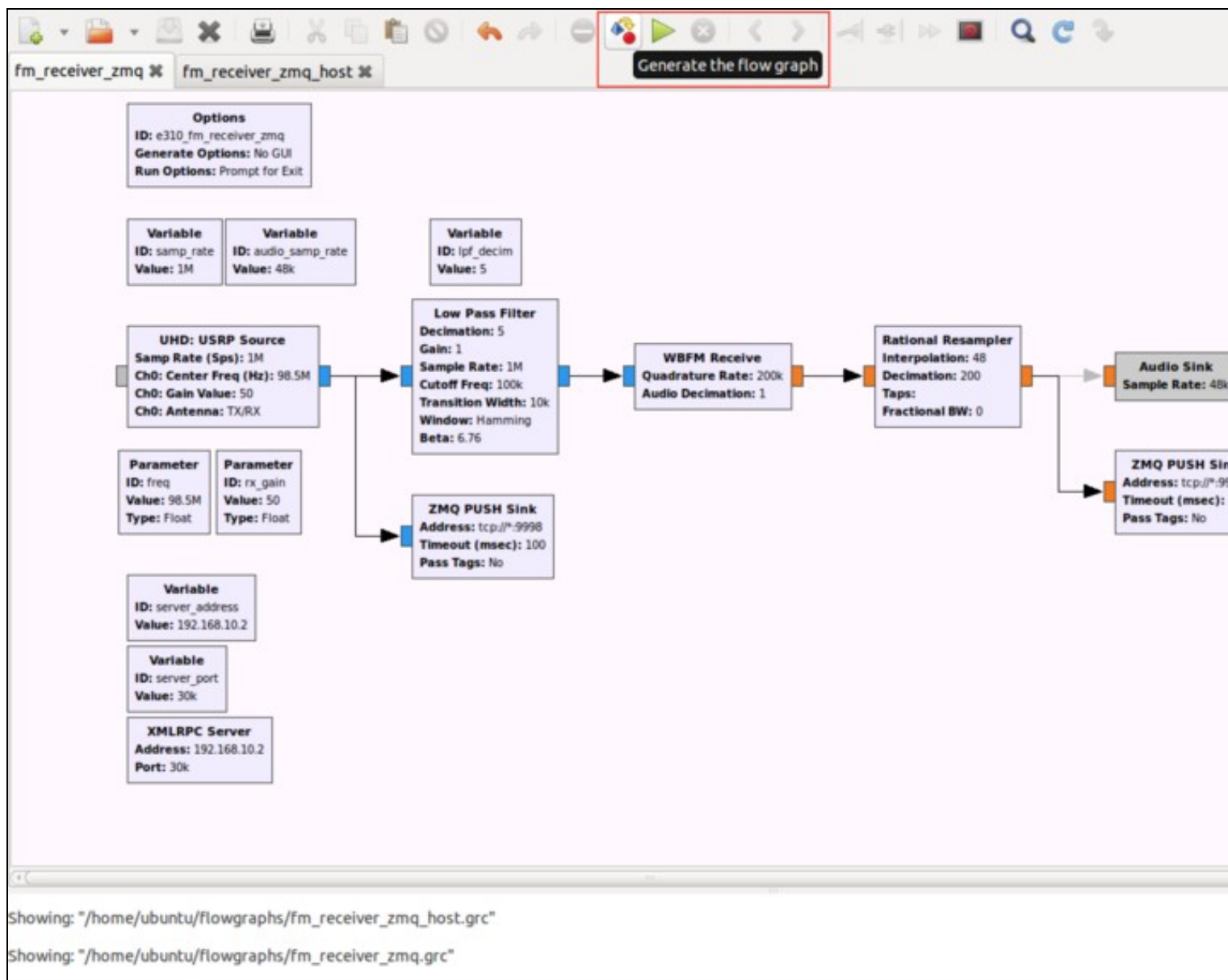
OK Cancel Apply

The figure below shows the flow graph [fm_receiver_zmq_host.grc] that will run on the host computer.

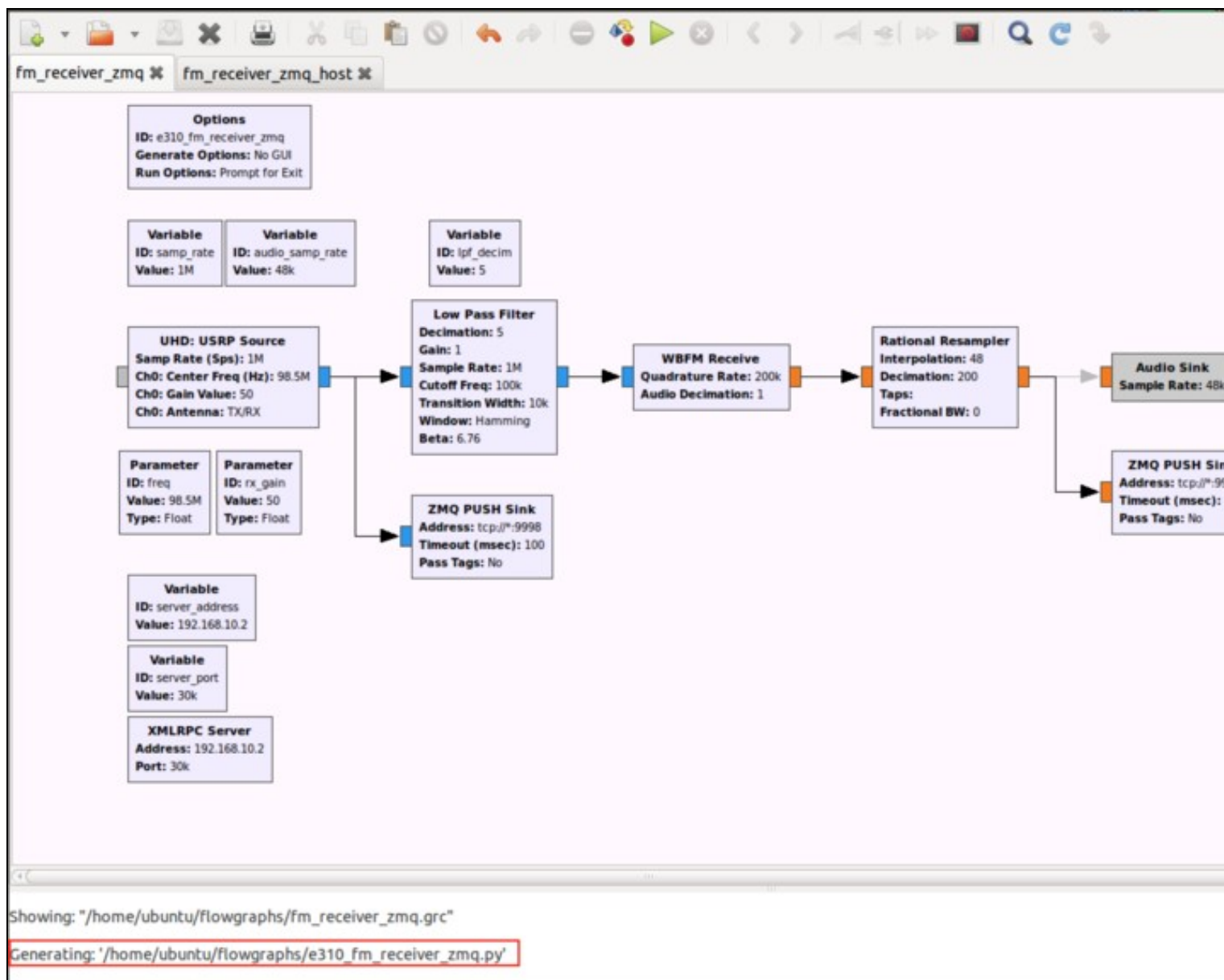


Before you're able to run the flow graph on the USRP E310, you will first need to generate the Python file and transfer it to the USRP E310.

To generate the Python application code from the flow graph is done by using the Generate button with GNU Radio as shown in the figure below.



Note: After running the Generate function, within the console of GNU Radio Companion, it will print the location of the generate Python file. The filename will match the value of the ID setting within the Options block.



Open a new Terminal window, and navigate to the directory where the Python file was generated.

```

ubuntu@ubuntu: ~/flowgraphs
ubuntu@ubuntu:~/flowgraphs$ ls -alh
total 72K
drwxrwxr-x  2 ubuntu ubuntu 4.0K Dec 22 01:06 .
drwxr-xr-x 29 ubuntu ubuntu 4.0K Dec 21 23:55 ..
-rwxrwxr--  1 ubuntu ubuntu 5.6K Dec 22 01:05 e310_fm_receiver_zmq.py
-rw-rw-r--  1 ubuntu ubuntu 34K Dec 22 00:12 fm_receiver_zmq.grc
-rw-rw-r--  1 ubuntu ubuntu 18K Dec 22 01:05 fm_receiver_zmq_host.grc
ubuntu@ubuntu:~/flowgraphs$

```

Before attempting to copy the file the USRP E310, ping it to verify you're able to communicate with it. **Note:** This application note assumes the IP address of the USRP E310 is 192.168.10.2.

```
$ ping 192.168.10.2
```

```

ubuntu@ubuntu: ~/flowgraphs
ubuntu@ubuntu:~/flowgraphs$ ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=0.175 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.144 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=0.170 ms
^C
--- 192.168.10.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.144/0.163/0.175/0.013 ms
ubuntu@ubuntu:~/flowgraphs$

```

Next, we will use the command line utility `scp` (Secure Copy) to copy the generated Python file to the E310.

```
$ scp e310_fm_receiver_zmq.py root@192.168.10.2:~/.
```

```

ubuntu@ubuntu: ~/flowgraphs
ubuntu@ubuntu:~/flowgraphs$ scp e310_fm_receiver_zmq.py root@192.168.10.2:~/
e310_fm_receiver_zmq.py                  100% 5649      5.5KB/s   00:00
ubuntu@ubuntu:~/flowgraphs$

```

Next, we will SSH into the E310 and verify the file was transferred successfully.

```
$ ssh root@192.168.10.2
```

```

ubuntu@ubuntu: ~/flowgraphs
ubuntu@ubuntu:~/flowgraphs$ ssh root@192.168.10.2
Last login: Tue Oct 25 11:11:13 2016 from 192.168.10.1
root@ettus-e3xx-sg3:~#

```

```
root@ettus-e3xx-sg3:~# ls -al e310*
```

```

ubuntu@ubuntu: ~/flowgraphs
ubuntu@ubuntu:~/flowgraphs$ ssh root@192.168.10.2
Last login: Tue Oct 25 11:11:13 2016 from 192.168.10.1
root@ettus-e3xx-sg3:~#
root@ettus-e3xx-sg3:~#
root@ettus-e3xx-sg3:~# ls -alh e310*
-rwxr-xr-- 1 root root 5.6K Oct 25 12:04 e310_fm_receiver_zmq.py
root@ettus-e3xx-sg3:~#

```

The command below will start the flow graph on the E310, there will be a verbose output.

```
root@ettus-e3xx-sg3:~# python e310_fm_receiver_zmq.py
```



```

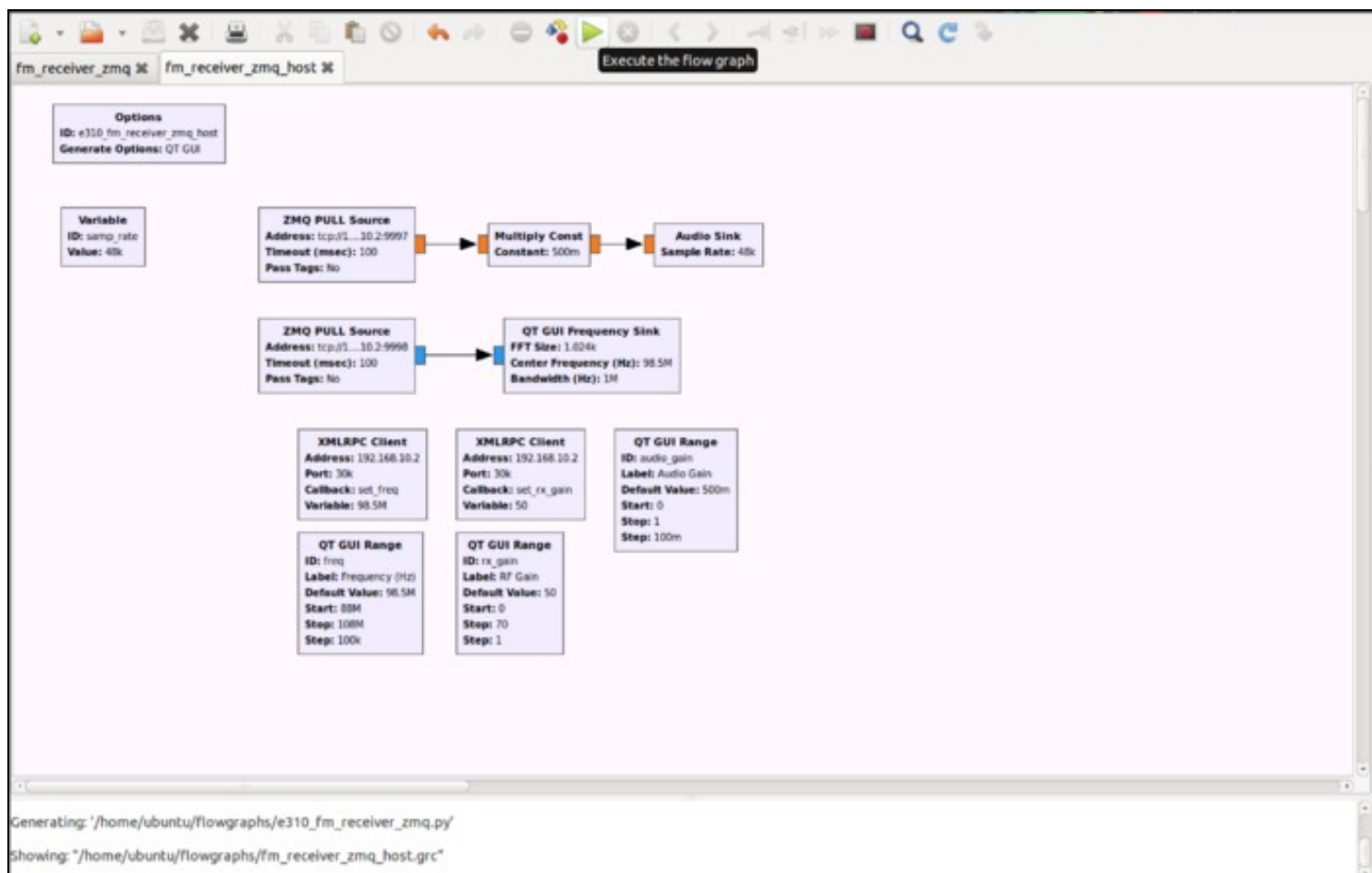
ubuntu@ubuntu: ~/flowgraphs

root@ettus-e3xx-sg3:~# python e310_fm_receiver_zmq.py
linux; GNU C++ version 4.9.2; Boost_105700; UHD_003.010.rfnoc-radio-redo-0-e915cbee

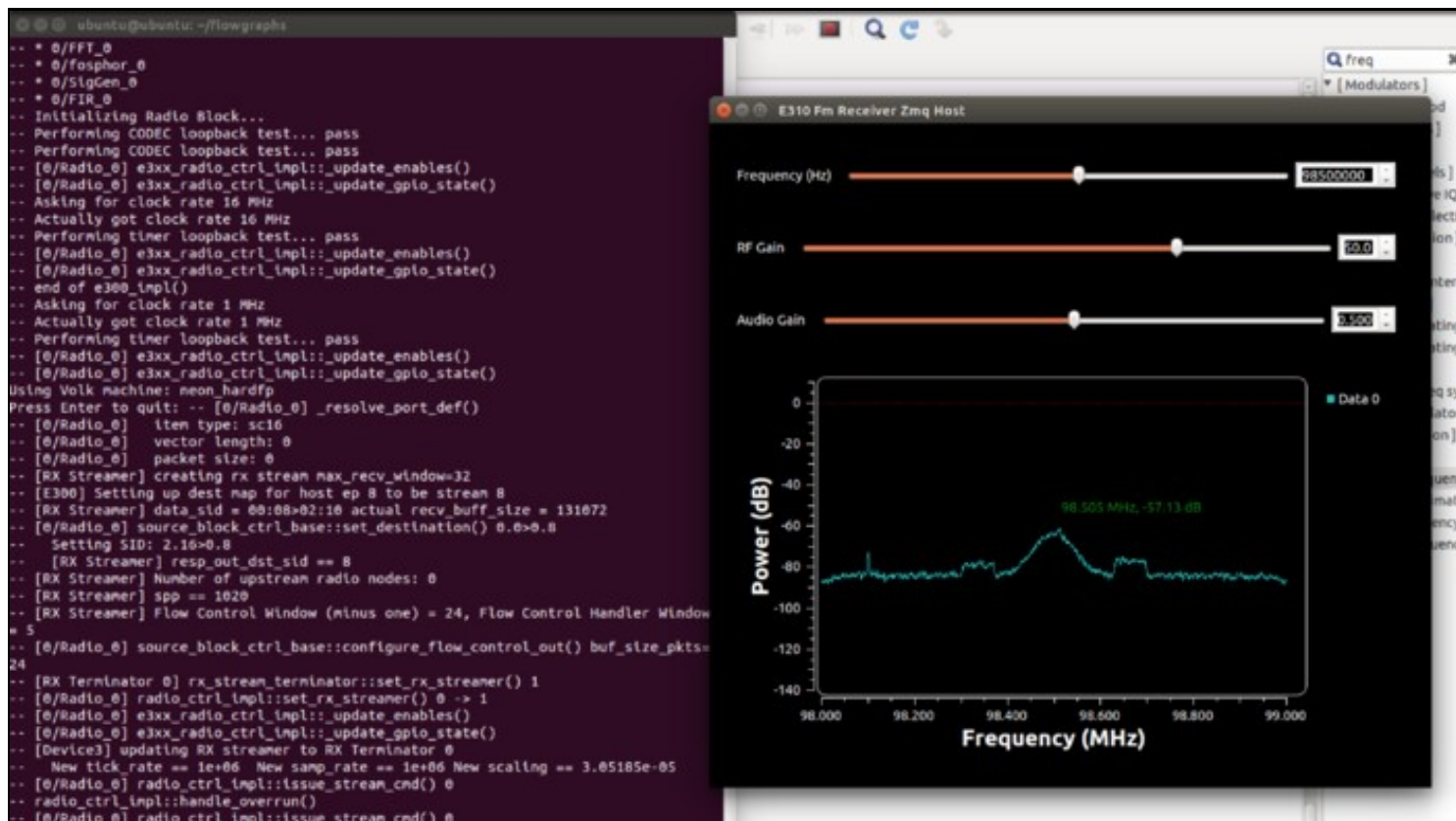
-- Loading FPGA image: /usr/share/uhd/images/usrp_e310_fpga_sg3.bit... done
-- Detecting internal GPSDO .... found
-- Initializing core control (global registers)...
-- Performing register loopback test... pass
-- Initializing AD9361 using hard SPI core...OK
-- [RFNOC] ----- Block Setup -----
-- [E300] Setting up dest map for host ep 0 to be stream 0
-- Setting up NoC-Shell Control for port #0 (SID: 00:00>02:10)...OK
-- Port 16: Found NoC-Block with ID 12AD100000000000.
-- base_path = "/usr/share/uhd/rfnoc/"
-- Reading XML file: /usr/share/uhd/rfnoc/blocks/radio_e3xx.xml
-- [E300] Setting up dest map for host ep 1 to be stream 1
-- Setting up NoC-Shell Control for port #1 (SID: 00:01>02:11)...OK
-- [RFNoC Factory] block_ctrl_base::make()
-- base_path = "/usr/share/uhd/rfnoc/"
-- Reading XML file: /usr/share/uhd/rfnoc/blocks/radio_e3xx.xml
-- [RFNoC Factory] Using controller key 'E3XXRadio' and block name 'Radio'
-- block_ctrl_base()
-- base_path = "/usr/share/uhd/rfnoc/"
-- Reading XML file: /usr/share/uhd/rfnoc/blocks/radio_e3xx.xml
-- Found valid blockdef
-- NOC ID: 0x12AD100000000000 Block ID: 0/Radio_0
-- [0/Radio_0] block_ctrl_base::clear()
-- node_ctrl_base::clear()
-- [0/Radio_0] block_ctrl_base::_clear()
-- [0/Radio_0] block_ctrl_base::_clear()
-- [0/Radio_0] Adding port definition at xbar/Radio_0/ports/in/0: type = 'sc16' pkt
_size = '0' vlen = '0'
-- [0/Radio_0] Adding port definition at xbar/Radio_0/ports/in/1: type = 'sc16' pkt
_size = '0' vlen = '0'
-- [0/Radio_0] Adding port definition at xbar/Radio_0/ports/out/0: type = 'sc16' pk
t_size = '0' vlen = '0'
-- [0/Radio_0] Adding port definition at xbar/Radio_0/ports/out/1: type = 'sc16' pk
t_size = '0' vlen = '0'
-- [RFNoC Radio] Performing register loopback test... pass
-- [RFNoC Radio] Performing register loopback test... pass
-- [0/Radio_0] radio_ctrl_impl::_update_spp(): Requested spp: 364
-- [0/Radio_0] radio_ctrl_impl::_update_spp(): Setting spp to: 364
-- [0/Radio_0] e3xx_radio_ctrl_impl::ctor()
-- Setting time source to internal
-- [0/Radio_0] e3xx_radio_ctrl_impl::_update_gpio_state()
-- [0/Radio_0] Creating internal GPIOs...
-- [0/Radio_0] Setting tick rate...

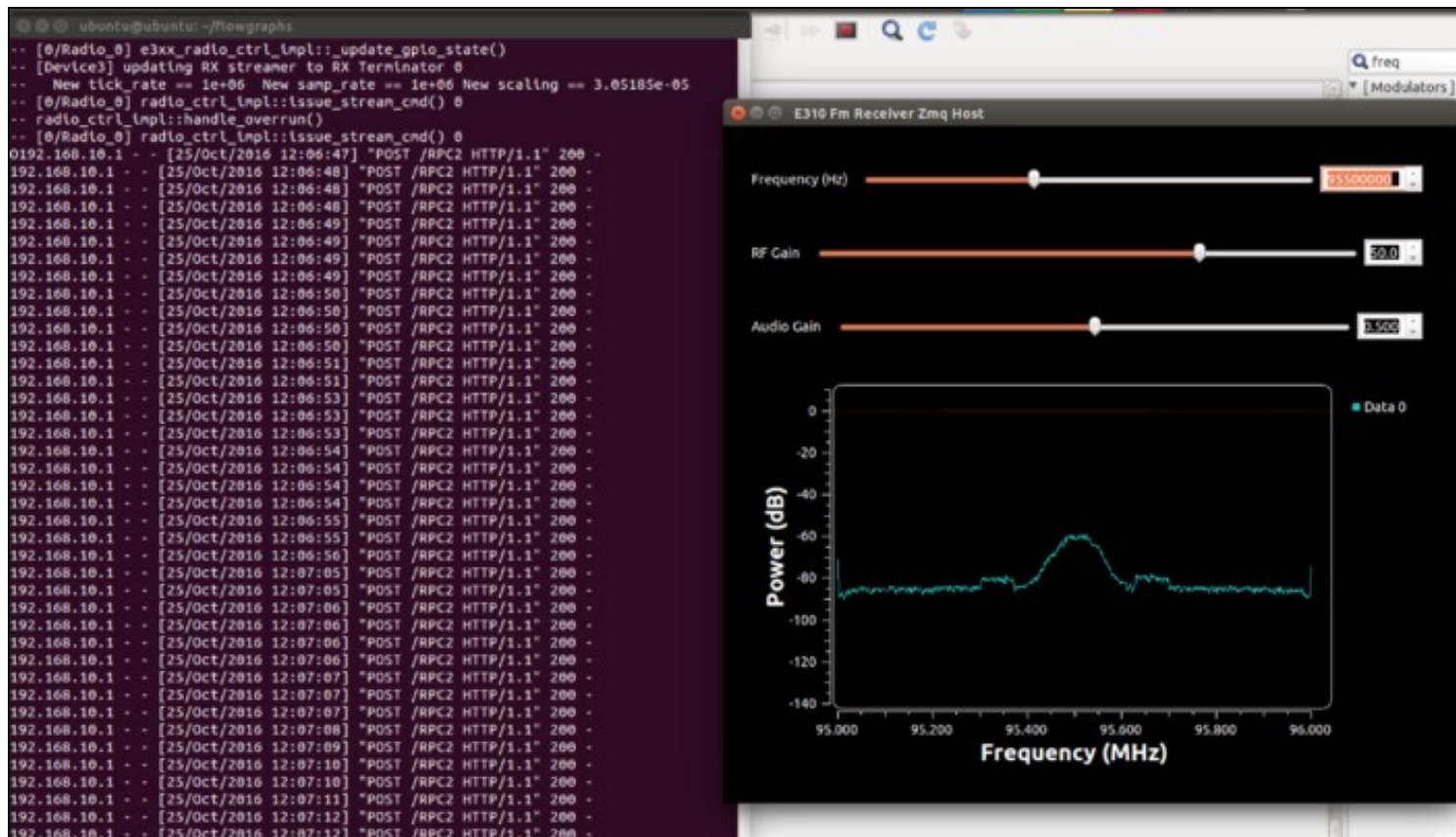
```

Returning to GNU Radio Companion, open the "fm_receiver_zmq_host.grc" flow graph, and Execute the flow graph.

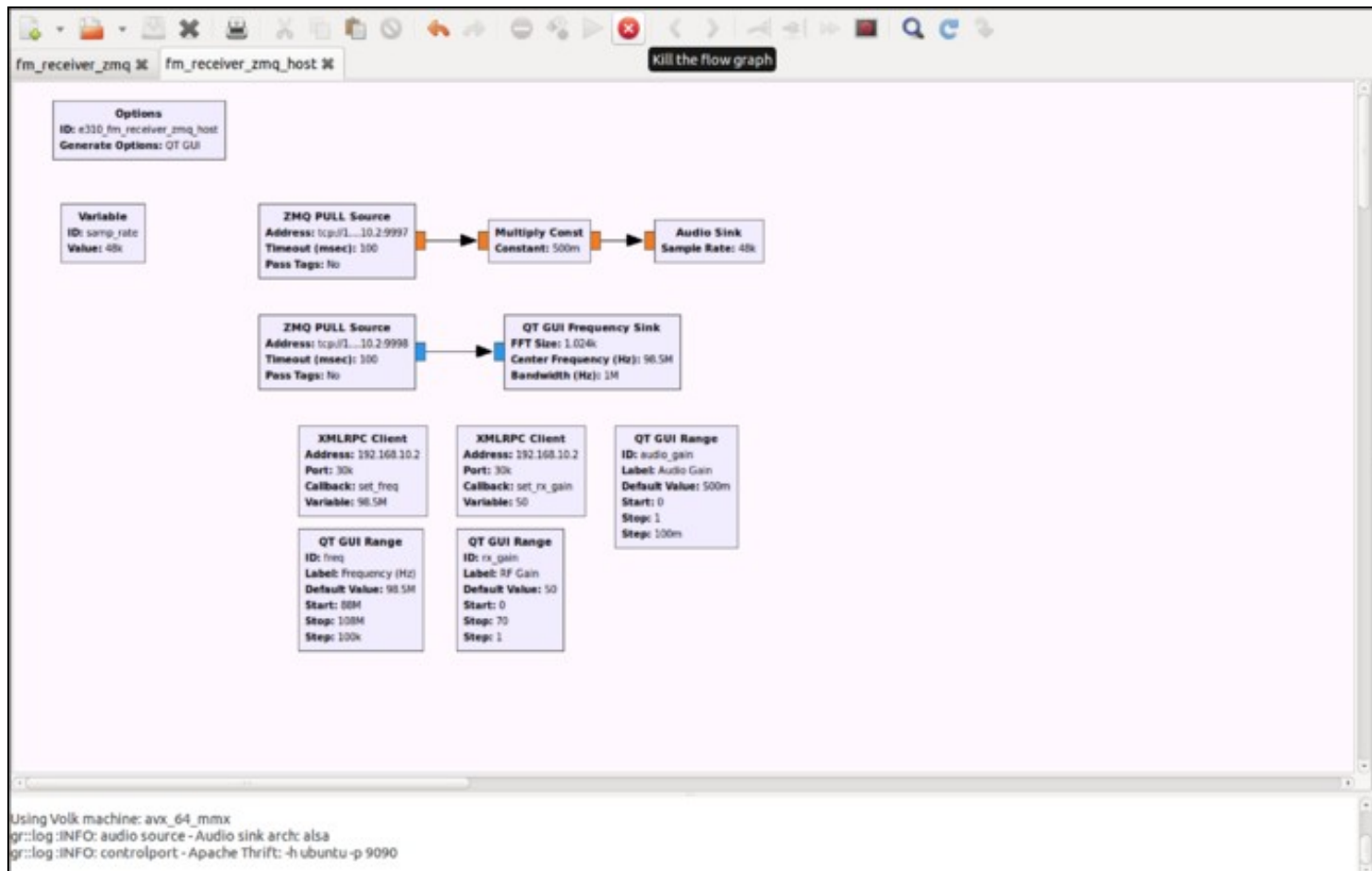


A QT based widget window will open as shown in the figure below. Tune to a strong local radio station by adjusting the QT Range widget.





Stopping the application is a multiple step process. First you will need to Kill the flow graph running on the host computer. This can be done by either hitting the "X" in the upper corner of the QT widget window, or by using the "Kill the flow graph" button within GNU Radio Companion.



Next, within the Terminal window that is running the flow graph on the E310, simply hit **Enter**.

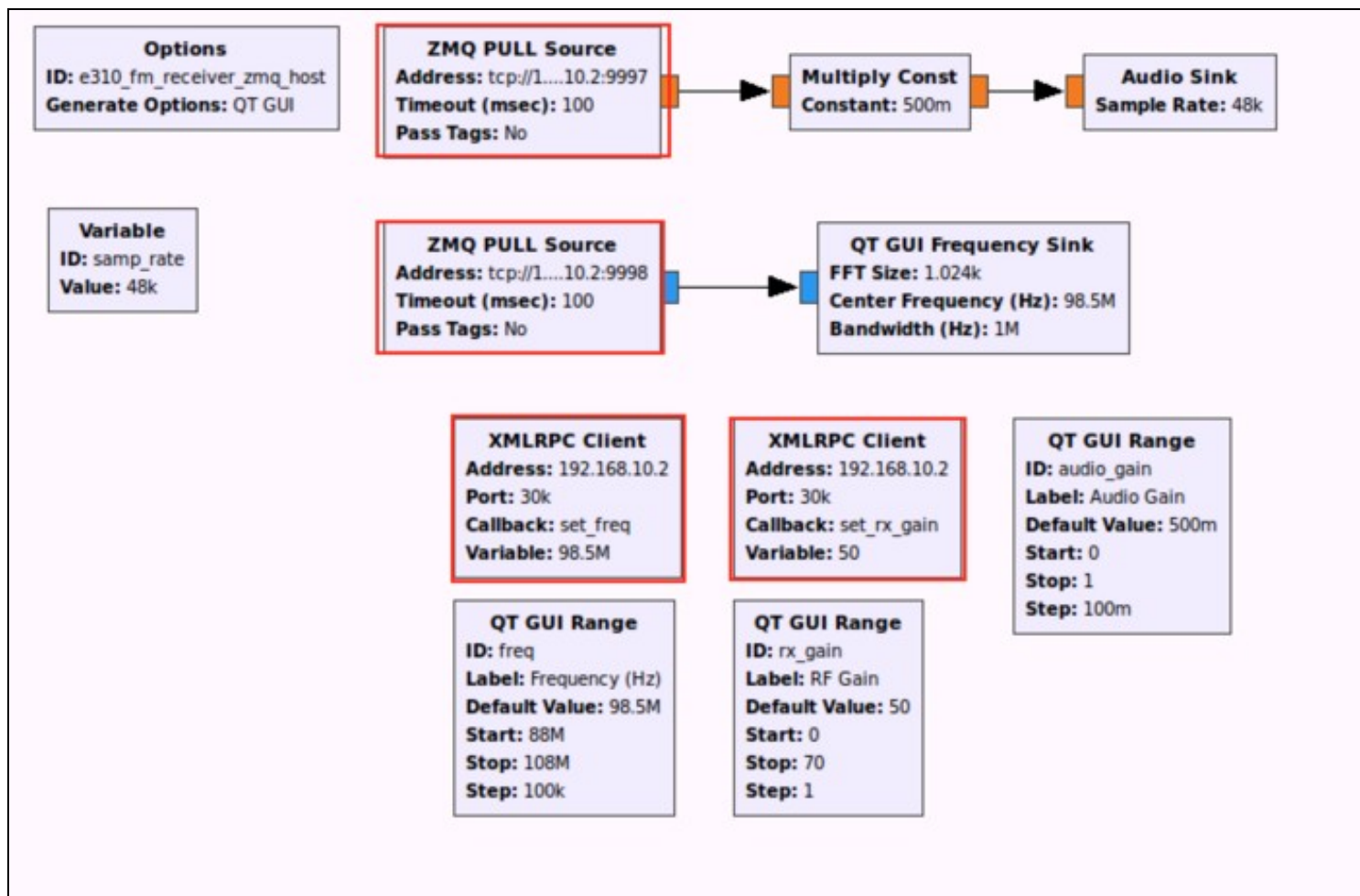
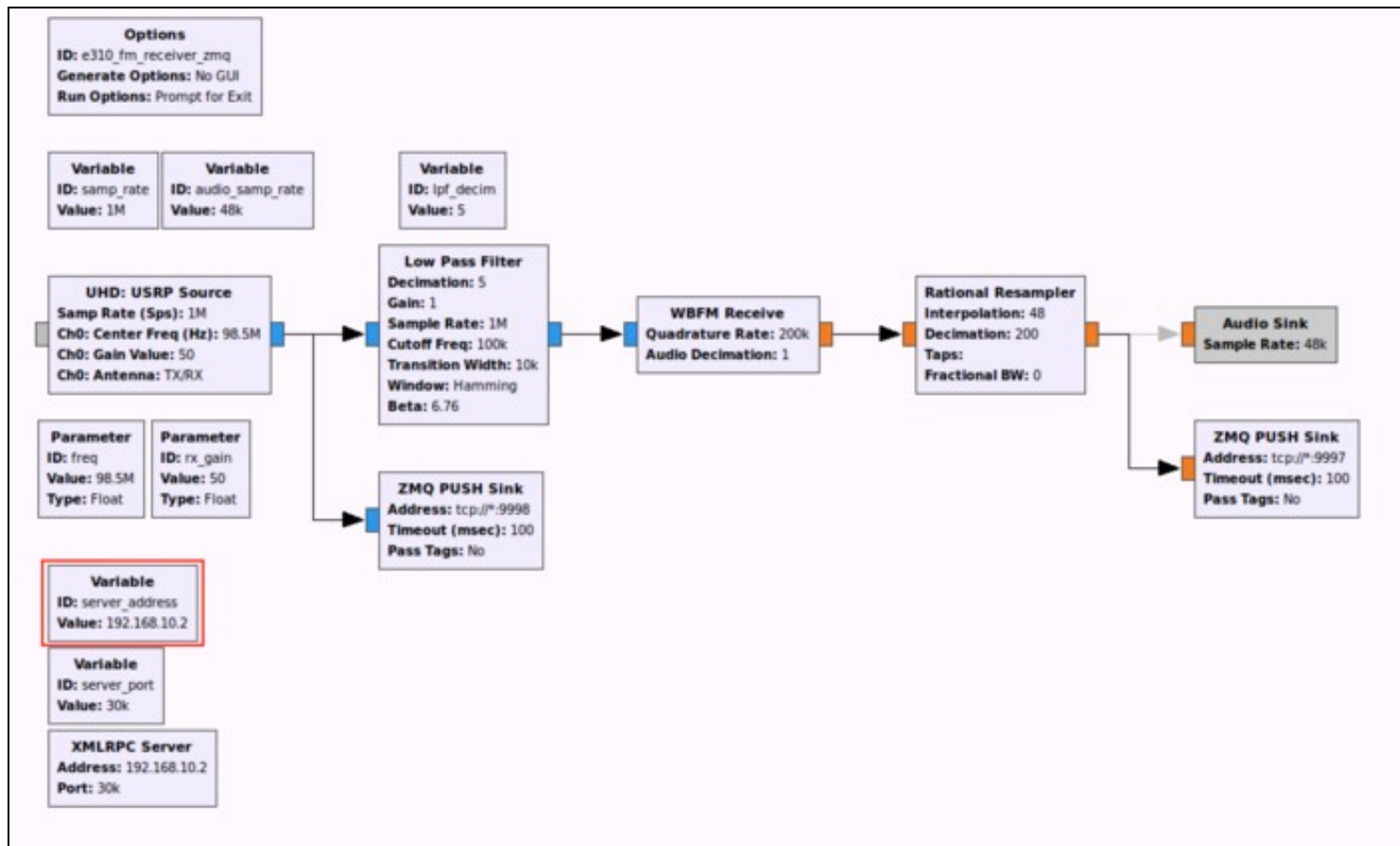

```

ubuntu@ubuntu: ~/flowgraphs
192.168.10.1 - - [25/Oct/2016 12:06:48] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:48] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:48] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:49] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:49] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:49] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:50] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:50] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:50] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:50] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:51] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:51] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:53] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:53] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:53] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:54] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:54] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:54] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:54] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:55] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:55] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:06:56] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:05] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:05] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:06] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:06] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:06] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:07] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:07] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:07] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:08] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:09] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:10] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:10] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:11] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:12] "POST /RPC2 HTTP/1.1" 200 -
192.168.10.1 - - [25/Oct/2016 12:07:12] "POST /RPC2 HTTP/1.1" 200 -

-- [0/Radio_0] radio_ctrl_impl::issue_stream_cmd() 0
-- radio_ctrl_impl::handle_overrun()
-- [0/Radio_0] radio_ctrl_impl::issue_stream_cmd() 0
Oroot@ettus-e3xx-sg3:~#
root@ettus-e3xx-sg3:~#
root@ettus-e3xx-sg3:~#
root@ettus-e3xx-sg3:~#

```

If your E310 is running with an IP address other than 192.168.10.2, the flow graphs will need to be updated in several locations. The figures below highlight the locations that will need to be changed.



- E310 Flowgraph Media: fm_receiver_zmq.grc
- Host Flowgraph Media: fm_receiver_zmq_host.grc