Gnuradio Companion module for openHPSDR Hermes / Metis SDR Radio

Tom McDermott, N5EG
3950 Southview Ter.
Medford, OR 97504
n5eg@tapr.org

Keywords: openHPSDR, Gnuradio, SDR, Hermes, Metis

Abstract

This paper discusses the design and implementation of software that provides an interface for the OpenHPSDR Hermes/Metis Ethernet-based SDR transceiver module to Gnuradio. General design requirements imposed by Gnuradio and the hardware itself are discussed. Some applications of Gnuradio using this software are illustrated. The module has been tested with Hermes and should be compatible with Metis. The module has been tested on Ubuntu 12.04, 12.10, and 13.04. It has been tested at the time of this writing through the current version of Gnuradio 3.6.5

Introduction.

Gnuradio is an open, free software application that can be used to rapidly prototype many different kinds of digital signal processing tasks[1]. It contains a number of modules for the generation, processing, and consumption of signals in digital format. Natively, gnuradio processes signals in a fast path primarily in C++. An associated program, Gnuradio Companion (GRC)[2] provides a GUI-based design surface for instantiation, interconnection, configuration, and setting parameters for the various modules. Python is used in limited cases for additional non-real-time aspects.

Most of the modules that come with gnuradio provide computation-only capabilities, such as the generation of random data streams in various formats, processing (such as FFT, convolution, filtering, etc, and consuming data streams (such as oscilloscope, spectrum analyzer, constellation domain, and other displays, as well as producing audio output to the speakers on the host computer via an audio output sink. In gnuradio, a source of a data steam is called a *source*, the termination of a data stream is called a *sink*.

While purely computational sources and sinks are valuable, it is also desirable to be able to source and sink actual data streams from hardware. A couple of interface modules exist for gnuradio, such as the low-cost SDR DVB-T dongle, and the Ettus Research UPSR modules. This project provides a software module that allows the Ethernet interface on the openHPSDR based Hermes and Metis modules to provide both sources and sinks of data to gnuradio from the hardware, and allows gnuradio computational modules to process them. In general the gnuradio computational modules are sufficiently fast that, depending on the host computer, they can keep up with the real-time nature of the data streams, even as fast as 384k I+Q samples per second for relatively large number of processing modules. Older PC's however may be more limited in processing capability.

The advantage to the gnuradio architecture is that real-time software applications are designed and debugged graphically without needing to write DSP or other code.

The term Hermes will be used in the rest of this

paper to mean Hermes and/or Metis. To help in reading the rest of this article, it is advisable to download the source code tree from the TAPR Subversion (SVN) repository, and refer to the code, files, and folders for the following discussion.

In this article the terms:

- Hermes means the actual physical module,
- hermesNB is the software running on the Linux system under gnuradio. The NB means narrowband – the wideband raw sample set from the ADC is not used, only the narrowband digitally down-converted and decimated samples are used by the module.

Gnuradio Requirements

Most of the design of a gnuradio module is in C++. Additionally a simple XML module needs to be written that provides the GUI interface for the module being written for gnuradio companion. The build of the modules is quite complex, and a tool, called modtool.py[3] is available to generate the directory structure, build directory, and the CMake files if you want to invent a new module.

Directory Structure

Modtool.py creates the directory structure required to organize and build modules within gnuradio and for gnuradio companion. Three of the directories are of concern to those writing a new module, the rest are needed but can be pretty much left intact. It creates the subdirectories, makefiles, cmake files, shell program files and stubs. The process of making or installing a new module is called an 'out-of-tree module' because it's outside the standard build tree of gnuradio (i.e. it is a user written module). Three directories contain what needs to be edited:

- LIB – contains the C++ source code modules for the out-of-tree module to be written. A skeleton is auto-created for the main module by modtool.
- INCLUDE – contains the C++ header files for the C++ code in the LIB directory. A skeleton is auto-created for the main module.
- GRC – contains the XML file that instructs GRC how to display the module that is being written. Again, a skeleton is auto-created.

There are multiple different CmakeLists.txt files, different ones in the different directories. The one located in the lib directory is hand edited to tell the build which C++ files are to be compiled and linked into the single gnuradio module.

For the Hermes interface four modules were written:

metis.cc - Ethernet interface and device discovery (borrowed and then modified from an early version of the metis.c module written by John Melton) and translated into C++.

hpsdr_hermesNB.cc - a gnuradio interface module that instantiates and removes the software interface called by the gnuradio application. It is called directly from gnuradio and needs to supply several well-defined interfaces. The tool auto-creates a module with the minimum set of needed interfaces created and stubbed. Gnuradio uses the C++ boost library to create reference-counted instances.

HermesProxy.cc - a Proxy that does most of the work. This module provides a circular pool of buffers for transmit and receive, queues transmit and receive data, sends control register data to Hermes, and receives status updates. It reads and updates controls from the GUI control panel. It also keeps count of the number of packets transmitted and received, and determines if any have been dropped. It displays status, Ethernet and IP addresses, and packet counts on the GRC status panel.

hpsdr_hermesNB.xml – a small XML formatted module to interact with the GRC display software, and display on-screen help for the module. The xml module also defines the way

that grc hands parameters to the C++ code via callbacks.

Installing Gnuradio

The gnuradio site contains a comprehensive script to install and build gnuradio on an Ubuntu machine from scratch. It will also check for and install dependencies (such as the C++ compiler and tool chain) automatically. A wget command is provided[4].

Installing hermesNB

To actually install the hermesNB module into your instance of GRC, the hermesNB module for gnuradio needs to be downloaded from the subversion SVN repository hosted by TAPR. Instructions for building it are on the TAPR SVN in the /trunk/n5eg document "How to build gnuradio Hermes-Metis.pdf" (it is built and installed as an out-of-tree module). Modtool is not needed since the files, directories, and make files for hermesNB already have been completely configured. The TAPR SVN repository contains instructions on the use of the module itself as well as the source code of all the files (released under the GPL license). You can browse the repository with either your web browser or download using subversion. The URLs are different for the browser and SVN.

Web URL: http://svn.tapr.org
Browse to OpenHPSDRMain, then to trunk/n5eg where you will find the instructions, and current source.

The Subversion URL is: http://svn.tapr.org/repos_sdr_hpsdr/trunk/N5EG

When building the software it is recommended that you check out the files from TAPR using subversion since they will all get properly located and the proper directories created. The build instructions assume Ubuntu version 12 or 13, but it will probably build for other versions of Linux as well.

Design Requirements

Gnuradio imposes some design limitations on any module. One is that all the outputs from the module must run at the same sample rate. In general, the inputs can run at different sample rates. For the Hermes implementation is was decided to have a single input – the transmit baseband sample stream, and one or two outputs, representing the output stream(s) for one or two receivers. The model could be extended to additional receivers but two seems sufficient for the kind of experiments most likely to be done within gnuradio.
The output streams can run at 48k, 96k, 192k, or 384k rates supported by Hermes, and the input runs at a fixed 48k rate. The input and output samples are formatted as floating point I and Q values. While Hermes itself has 16-bit 2's complement resolution on transmit and 24-bit 2's complement resolution on receive, the hermesNB module converts all I/O to and from the 2's complement fixed point format to the system complex floating I/Q type for gnuradio use.

In addition to the transmit and receive data, the hermesNB module also inserts and removes command and control data to the module. In Hermes, these C&C data are interleaved in special positions within the ethernet frames sent and received from the module. The hermesNB module handles both setting and querying the C&C data from the GUI as well as formatting it for transmission and reception over Ethernet.

The I/O to the Hermes module is pretty low level socket based I/O using the socket.h interface. The first thing that the module does on startup is to find the Hermes module by sending out a special HPSDR broadcast packet that Hermes will respond to. Once that has been found, it can then start Ethernet I/O with that module itself. The Ethernet interface on which the hermesNB code operates is specified in the gui as a parameter, defaulting to eth0. The module also provides a little bit of I/O debug data on the GRC console (which helps a lot in tracking down IP addresses, Ethernet addresses, and which interfaces are available to the system).

The receive I/O is started on a separate thread

and blocks awaiting the reception of Ethernet frames from Hermes. In Ubuntu 13.04 the socket can unblock on a system message without any data, so additional code was added to handle restarting the blocking system call when an unimportant message is received. On receipt of an Ethernet frame, the code validates the Ethernet packet (length and header bytes), extracts the actual I/Q samples and then packs the samples into a buffer from a large circular pool of buffers and returns.
Similarly, on transmit, the hermesNB code checks to see if there are any fully-built Ethernet frames ready to be sent to the actual Hermes hardware, and if it is time to sent a packet. If so, then it pulls the head packet for the transmit circular buffer pool and transmits it. The buffers are prebuilt into Ethernet packets and placed into a transmit circular queue whenever a sufficient number of transmit samples are available from gnuradio.

The actual Hermes hardware has no way to provide information as to when it needs another transmit packet (queue status), nor a buffer ready/done indication. However the receive Ethernet frames from Hermes are completely periodic, but they depend on the receive sample rate and the number of receivers. Faster sample rates and more receivers means that Ethernet packets are sent more frequently by the Hermes hardware to the hermesNB module.

The pacing of Ethernet transmit frames to Hermes thus is determined by counting the number of received frames from Hermes, and then compensating for the receiver sample rate and number of receivers to determine a ratio of how many receive packets should be received for each transmit packet to be sent. This is then tested by the hermesNB module, and it queues transmit frames at the correct periodicity to the Hermes hardware.

The receive and transmit circular buffer pools allow the hermesNB module to decouple the transmission and reception of individual Ethernet frames from the streaming of floating point complex numbers to and from the gnuradio computational engine itself. Gnuradio cannot tolerate blocking of stream I/O, but it can be told there is nothing available for the output stream(s), or that the hermesNB module does not wish to pull anything from the input stream.

The hermesNB module keeps careful count of the transmitted and received Ethernet packets, and also keeps count of the sequence numbers that Hermes hardware inserts into the Ethernet packets. It uses this to detect if any transmit or receive frames were lost or corrupted at several different points in the chain. These counts are displayed (sometimes) when the hermesNB module exits. Unfortunately sometimes GRC has text queued on the console output and it prevents the hermesNB module from achieving a programmed shutdown. Nothing bad happens except that the text sent to the console by hermesNB is lost. If GRC has not queued text, then hermesNB successfully sends the diagnostic packet count information to the console.

Mutexes were placed in the transmit and receive directions to prevent thread based problems, but so far have not been needed as no packet errors have been able to be traced to them. They are still in the code in case they are needed is some future version of gnuradio or linux, but are commented out.

Application Figures
Figure 1 shows the properties panel of the module itself, where the module properties can be set. Note that parameters where the name is not underlined cannot be changed at runtime. Underlined parameters are set to default values at module initialization. Changing them during runtime requires creating a GUI control and passing it's value to the module via a variable name (all done from the GUI).

Figure 2 shows a single-signal SSB receiver implementation using the phasing method (provided by the Hilbert transform which is a standard element built into gnuradio). It should be noted that the Hermes hardware itself is FULL

DUPLEX. Thus controls to this gnuradio module add the ability to mute RX when transmitting, and to mute TX when receiving. Additionally, because Hermes has separate transmit and receive antenna connectors, the various muting controls can be disabled and the unit can actually run full duplex. The various controls provided have been instantiated as GUI controls using the GRC modules. The PTT control merely asserts a digital signal usually used to key a T/R relay. But it does not turn the transmitter on or off (the Hermes transmitter hardware in fact cannot be turned off).

Figure 3 shows a gnuradio model that takes the zero-hertz I/Q carrier output and further decimates it down to a sample rate of 24 samples per second with a bandwidth of +/- 10 hertz. This allows tracking the carrier offset frequency of WWV whilst stripping the modulation off the signal. This is applied to a large FFT and then averaged to give a sort of power spectral density of the received carrier frequency offset for observing ionospheric Doppler shift with sub-hertz resolution.

Figure 4 displays the runtime QT GUI widgets from the SSB Receiver application of figure 2.

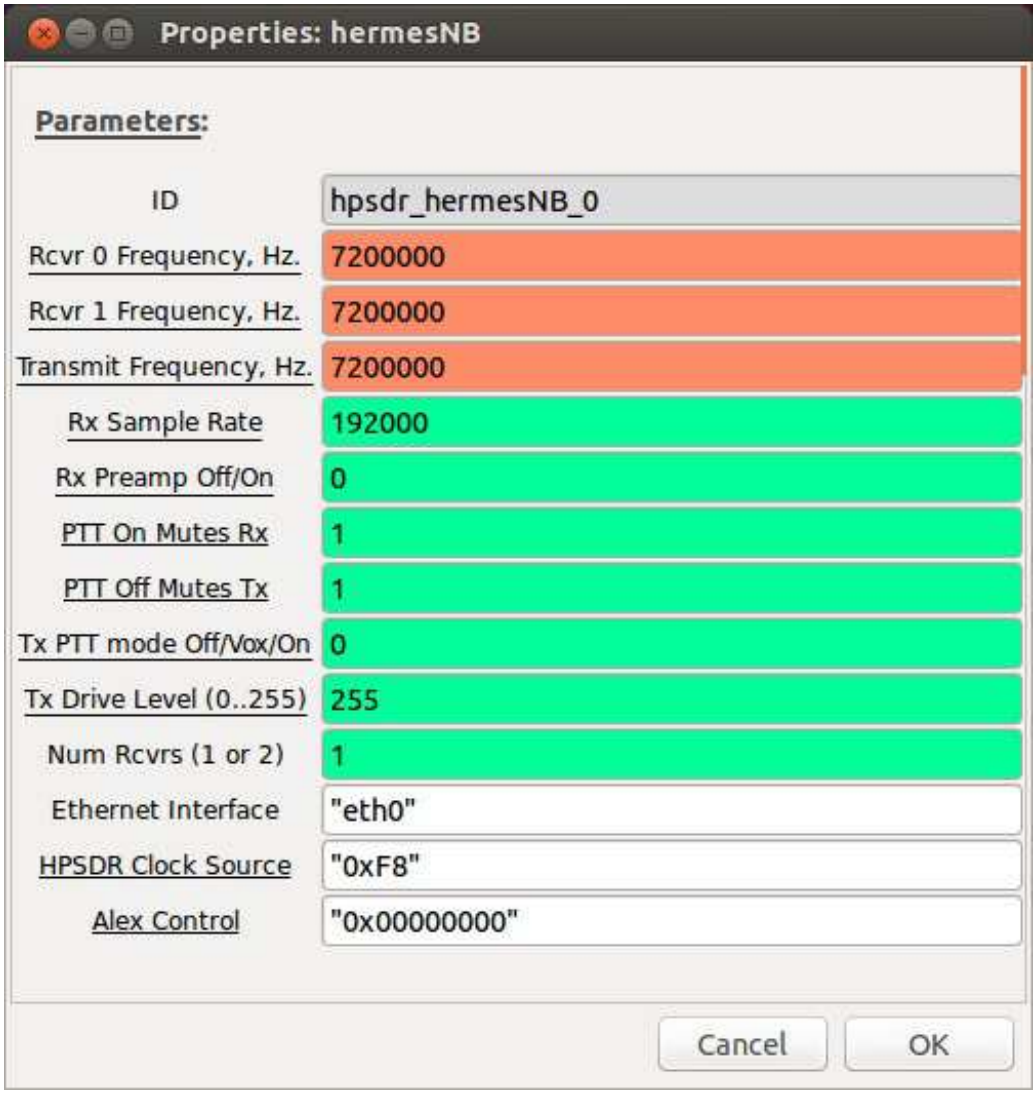

Figure 1 – The Hermes/Metis properties panel for gnuradio.

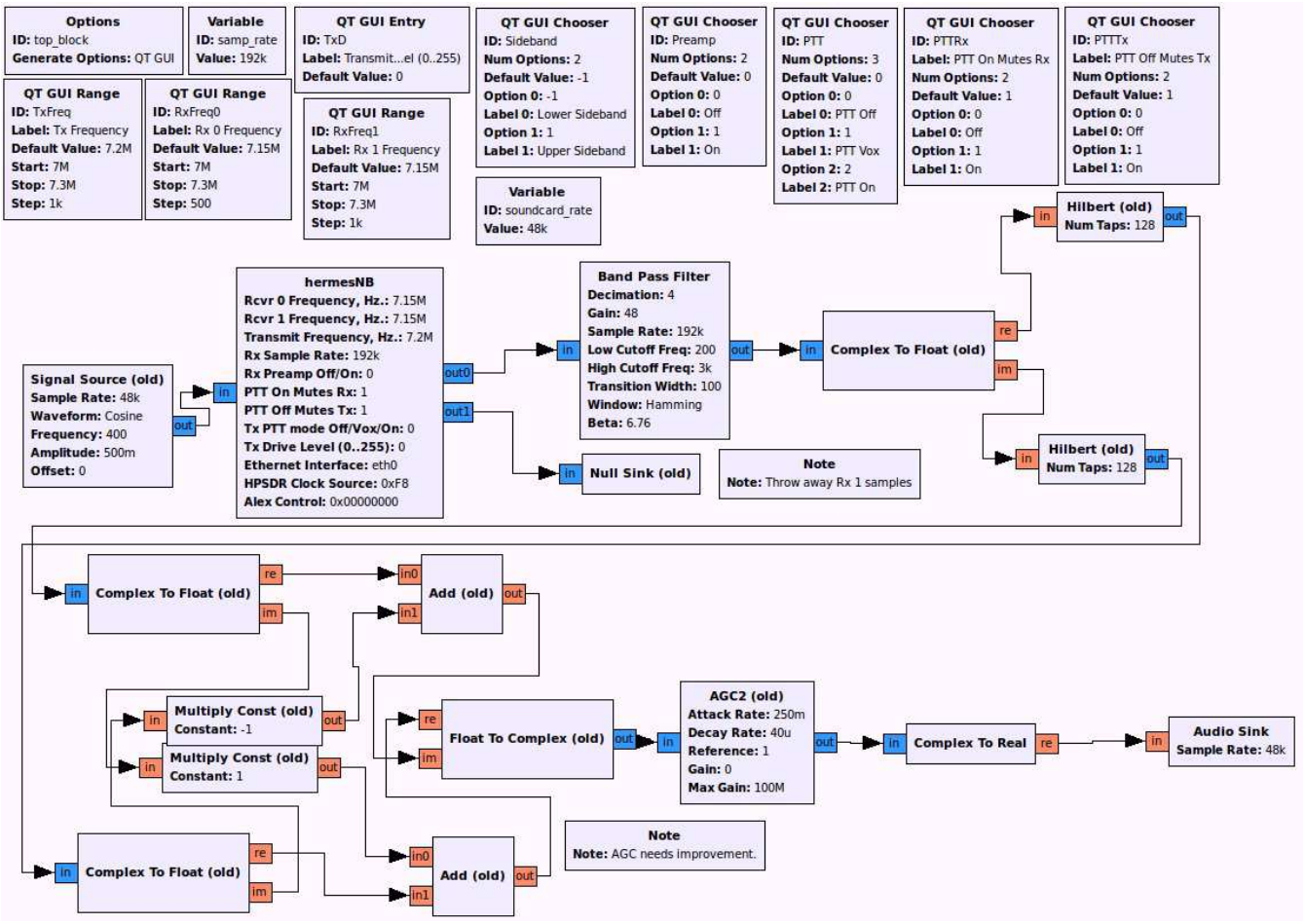

Figure 2 – Single signal SSB receive implementation in gnuradio using Hermes. The second receiver channel is unused but enabled in this example.

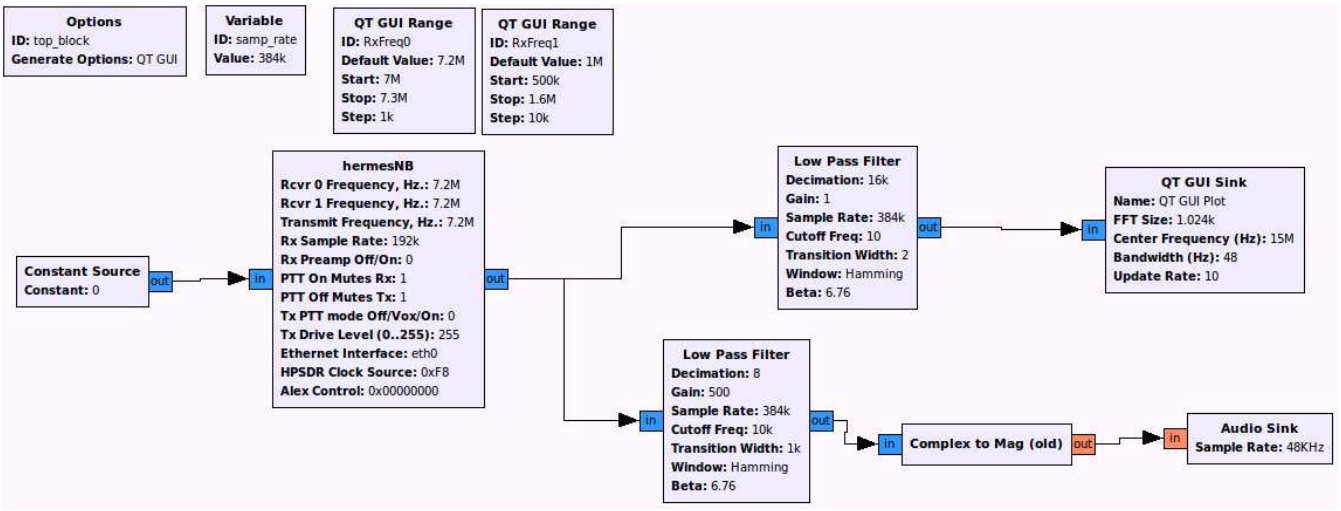

Figure 3 – Double sideband with highly-decimated narrowband carrier output to GUI sink. This is used to track the doppler shifted carrier offset frequency of WWV. The AM demodulated 10 kHz wide output is also sent to the computer soundcard.

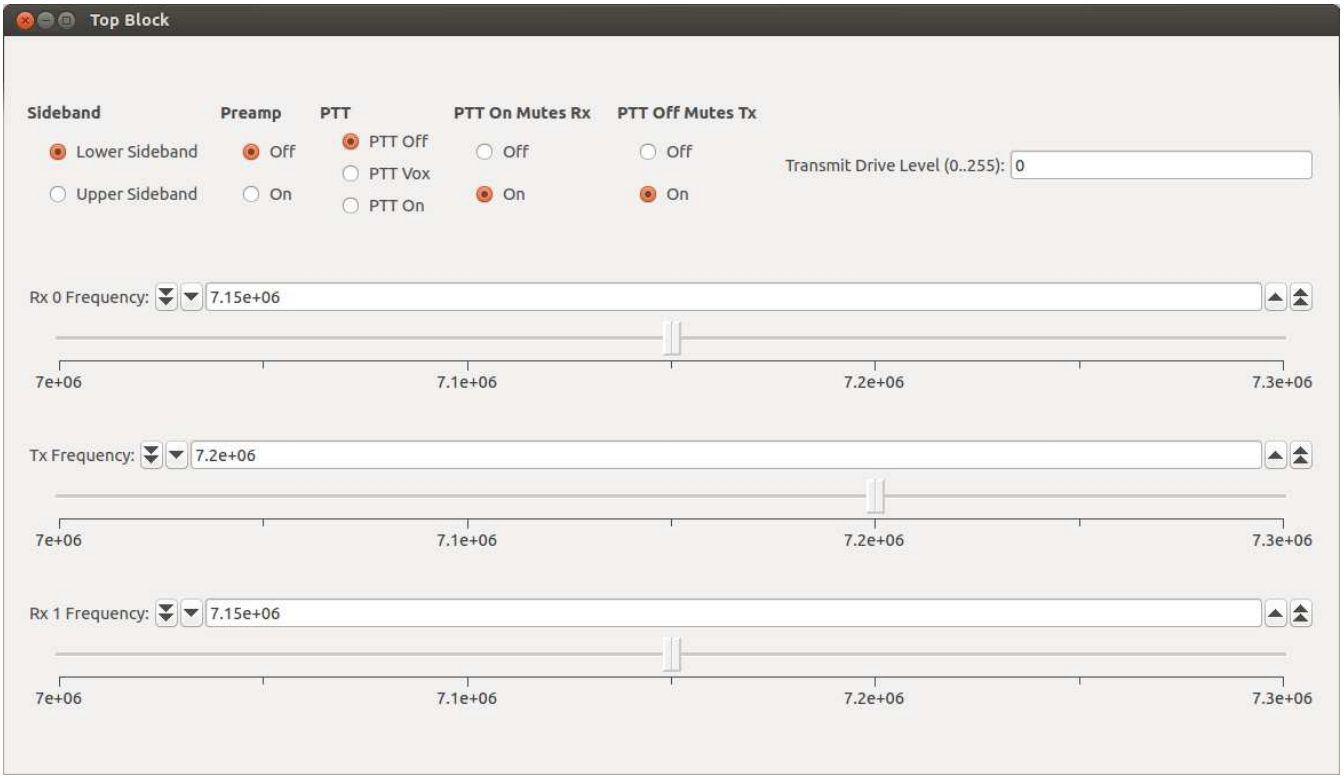

Figure 4 – SSB Receive Control Panel – displays of GUI widgets from SSB Receiver Application of figure 2.

1. The gnuradio wiki can be found at: http://gnuradio.org/redmine/projects/gnuradio/wiki

2. Using GRC avoids the need to generate python code to connect gnuradio blocks together. It's done graphically instead. GRC is installed by the gnuradio build/install script.

3. Modtool.py is now installed by default (in very recent gnuradio builds), the out-of-tree build information can be found at: http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules

4. The general gnuradio install notes page is at:
   http://gnuradio.org/redmine/projects/gnuradio/wiki/InstallingGR#Using-the-build-gnuradio-script
   The install script to build from scratch (the recommended way) is:

```
wget http://www.sbrac.org/files/build-gnuradio && chmod a+x ./build-gnuradio && ./build-gnuradio
```