Gnuradio Installation

Supporting Code:

You will need *git* and *cmake*. If these are not already installed on your system, you should install them. They are probably available using *yum* on Fedora or *apt-get* on Ubuntu. Otherwise, try `http://www.cmake.org/cmake/resources/software.html` and Google for *git* will show a lot of options; `http://vogella.com/articles/Git/article.html` is a useful link. You will also need *python* and a *C++* compiler, but probably have them.

There are some other packages that you may not have, and I mention them explicitly below. If you can find sufficiently modern binary packages they should be OK. The only one you should build from source is *gnuradio*.

uhd: This is the software to run the Ettus software radio devices, the USRP2 and the DBSRX2. It is available at in binary form for Fedora and Ubuntu or source (which builds nicely with *cmake* on OS-X 10.7). Go to `http://www.ettus-apps.sourcerepo.com/redmine/ettus/projects/uhd/wiki`; the Ettus web site has a lot of other useful information, and images for the fpga and firmware in the USRP hardware.

For Fedora 15 I set up the `uhd_stable` repo as described and used *yum*. That also installed *boost cmake* and *python* run time support; however there are lots of other things you will probably need. The best way to find out what they are is to build *gnuradio*. **boost:** This is available using *yum* on Fedora or *apt-get* on Ubuntu. It should be version 1.36 or higher. This is a set of C++ libraries that help bind C++ to python. See `http://www.boost.org/` for more information. The macports version seems to work for OS-X.

swig and **swig-python:** are more packages to help C++ and python work together. Available using *yum* on Fedora or *apt-get* on Ubuntu. The macports packages `swig` and `swig-python` seem to work for OS-X.

fftw: This is a package developed at MIT that does all the heavy lifting for FFTs. It's available using *yum* on Fedora or *apt-get* on Ubuntu. The macports packages `fftw-3` and `fftw-3-single` seem to work for OS-X. See `http://www.fftw.org/`. To enable `wx` graphics using `OpenGL`, get **PyOpenGL**.

doxygen, **sphinx** and **python-sphinx:** may be useful as they will make rather skimpy documentation (based on the `*.hpp` files) as you build the gnuradio packages. The analogous macports packages are OK on OS-X.

OS-X Notes:
Installing on OS-X can have some annoying moments, most of which I have experienced. In most cases *macports* is helpful: `http://www.macports.org/`. However (in June 2012) the gnuradio software was too old to be useful, so don't install any macports gnuradio packages.

Another issue is that release 10.7 (Lion) is 64-bit and some of the macports packages are only 32-bit. For example, the Gnu Software Library (GSL) is needed by the gnuradio `gr-wavelet` package, and the following command added 64-bit support to the macports gsl package.

```
prompt> sudo port upgrade --enforce-variants gsl +universal
```

Gnuradio:

I think it is important to build this from source. One reason is that the documentation is sparse at best and you will need to look at the source code sometimes. Another is that the binary downloads are a bit long in the tooth. The first thing to do is to get recent source code. That can be done with this command.

```
prompt> git clone git://gnuradio.org/gnuradio
```

That will create a directory called `gnuradio` in whatever directory you were in when you typed the command; I was in `/usr/local/src`. The new `gnuradio` directory will contain the source for all of the gnuradio packages. You will probably not want to build them all.

The `gnuradio` directory has a file `README` and the first thing to do is read it. Then go to the link `http://gnuradio.org/doc/doxygen/page_build.html` to find the dependencies you will need and install them.

The next step is to make a new directory `build` in `gnuradio` and change to it.

The command you issue to *cmake* determines what the system tries to build. The command was long, so I made the following command file

```
#!/bin/bash
#cmake -DCMAKE_INSTALL_PREFIX=/usr/local
cmake -DFFTW3F_INCLUDE_DIRS:PATH=/usr/include \
 -DFFTW3F_LIBRARIES:FILEPATH=/usr/lib/libfftw3f.so \
 -DFFTW3F_THREADS_LIBRARIES:FILEPATH=/usr/lib/libfftw3f_threads.so \
 -DENABLE_GR_ATSC=OFF \
 -DENABLE_GR_COMEDI=OFF -DENABLE_GR_FCD=OFF -DENABLE_GR_PAGER=OFF \
 -DENABLE_GR_VIDEO_SDL=OFF -DENABLE_GR_VOCODER=OFF -DENABLE_GR_WAVELET=OFF \
 ../ > ,out
```

The various "OFF" things determined the modules that I did not want to build; the others (usually each a separate subdirectory in `gnuradio`) will be built by default.

On Fedora 17, *cmake* had could not find several of the dependencies that I knew were installed. I did not have this problem on Fedora 16 or OS-X where the line commented out was used and all of the stuff beginning with `-DFFT...` was not included. As `/usr/local` was the default installation directory, I dropped the definition for it on Fedora 17 and *cmake* then searched more broadly and found everything but the `fftw` package. I found the solution to that by including the three `-DFFT...` definitions explicitly. Someone familiar with the operation of *cmake* could probably find a more elegant solution.

There will be copious output from *cmake*, sometimes complaining about things that are wrong—such as a shared library not being available to link to. It's a good idea to capture this output by directing it to a text file, as in my script, to look at it with an editor. At the end of this output *cmake* will say which modules it will build (enabled) and which it will not build (disabled). If one you expected to be built is on the disabled list, it's probably because some dependency is missing and you will have to fix that. Looking at the `CMakeList.txt` file in the top level source directory for disabled modules will show you what dependencies are needed for that package. In the `build` directory will be a file called `CMakeCache.txt` and you can learn a lot from examining it.

There is a somewhat more permanent fix possible than editing the `CMakeCache.txt` file. In the `gnuradio` source directory is a directory `cmake/Modules` with many files that have the extension `.cmake`. These have the information that *cmake* uses to search for the packages that are needed. The names of many of them start with "Find..." and you can edit them with a text editor to add hints and places to look. Here is what I did to the file `FindUHD.cmake` that enabled *cmake* find all the stuff needed from the Ettus driver package that I had installed in `/opt/uhd` on my Mac.

```
########################################################################
# Find the library for the USRP Hardware Driver
########################################################################

INCLUDE(FindPkgConfig)
PKG_CHECK_MODULES(PC_UHD uhd)

FIND_PATH(
    UHD_INCLUDE_DIRS
    NAMES uhd/config.hpp
    HINTS $ENVUHD_DIR/include
$PC_UHD_INCLUDEDIR
    PATHS /usr/local/include
  /usr/include
  /opt/uhd/include
)

FIND_LIBRARY(
    UHD_LIBRARIES
    NAMES uhd
    HINTS $ENVUHD_DIR/lib
$PC_UHD_LIBDIR
    PATHS /usr/local/lib
  /usr/lib
  /opt/uhd/lib
)

INCLUDE(FindPackageHandleStandardArgs)
FIND_PACKAGE_HANDLE_STANDARD_ARGS(UHD DEFAULT_MSG UHD_LIBRARIES UHD_INCLUDE_DIRS)
MARK_AS_ADVANCED(UHD_LIBRARIES UHD_INCLUDE_DIRS)
```

Once it appears that *cmake* will try to build the things you want, you are ready for the next step. The command `make help` might be useful; it can also take a following argument (e.g., `gr-uhd`) to build just one module.

The next step is

```
prompt> make
```

When that succeeds, you can finish building the shared libraries and install them (probably requires root privileges).

```
prompt> make install
```

Then try a few of the programs in `/usr/local/bin` to see if they work. If you do not have the USRP2 connected, try something in `/usr/local/share/gnuradio/examples`, such as `pyqt_example_c.py`.

To connect to the USRP2 you will need a gigabit ethernet interface on your computer set with a static IP address `192.168.10.1`. (Recent Apple computers have a gigabit wired ethernet connection.) Then you can try some of the scripts in `/usr/bin`.

```
prompt> uhd_find_devices --args "addr=192.168.10.2"
```

should just work. If it does, try

```
prompt> uhd_usrp_probe --args "addr=192.168.10.2"
```

This may come back with a complaint that the versions of the firmware and fpga code in the USRP2 do not play well with the software installed on the computer. In that case, make sure that the uhd software is the latest from Ettus; at this writing it should be `003.004.002_27_stable-1`. It would probably be a good idea to run *yum update*. If everything is up to date and you still get the complaint when you run `uhd_usrp_probe` then you need to update the USRP2 firmware and fpga to match the `uhd 003.004.002_27_stable-1` package.

The Ettus website `http://files.ettus.com/uhd_docs/manual/html/usrp2.html` has documentation telling you how to do it, but here is what to do. (On *ubuntu* the stuff needed is all in `/usr/share/uhd`.) In the subdirectory `utils` is a script `usrp_n2xx_net_burner.py` which will install the images. In the subdirectory `images` are the two images to install. They are `usrp_n210_fw.bin` and `usrp_n210_r4_fpga.bin`.

Assuming you are in directory `/usr/share/uhd/utils` the commands are:

```
prompt> ./usrp_n2xx_net_burner.py --addr=192.168.10.2 \
  --fw=../images/usrp_n210_fw.bin
prompt> ./usrp_n2xx_net_burner.py --addr=192.168.10.2 \
  --fpga=../images/usrp_n210_r4_fpga.bin
```

The first command will take only a few seconds, but burning the fpga takes a minute or two. The USRP2 should be restarted after both the fw and fpga images have been installed.

Network Buffers:
These are likely to be too small as the system was installed. They may be increased temporarily before running a real application by:

```
prompt> sudo sysctl -w net.core.rmem_max=50000000
prompt> sudo sysctl -w net.core.wmem_max=1048756
```

The increase can be made permanent on *ubuntu* by editing `/etc/sysctl.conf`.

2013 Update:
The previous note are from building `gnuradio` on OS-X in July, 2012. When I built it on Ubuntu 12.04 in early 2013, the command to build it was.

```
#!/bin/bash
cmake   -DENABLE_GR_ATSC=OFF -DENABLE_BAD_BOOST=ON \
 -DENABLE_GR_COMEDI=OFF -DENABLE_GR_FCD=OFF -DENABLE_GR_PAGER=OFF \
 -DENABLE_GR_VIDEO_SDL=OFF -DENABLE_GR_VOCODER=OFF -DENABLE_GR_WAVELET=OFF\
 ../ >> ,out
```

There was also a newer version of `gnuradio`3.6.4.1 available at `git clone http://gnuradio.org/git/gnuradio.git`. The Ettus `uhd` driver has long had C++ methods to set the four gains of the DBSRX2 receiver independently as well as the dc_balance and iq_balance. The latest version of `gnuradio` has methods in `gr.uhd_usrp_source` to adjust them, too.

OS Mountain Lion (10.8.2) Update:
wxPython has gotten much better on OS X, so you should use it for GUI programs. But, 10.8.2 insists on 64-bit code and in December 2012 *macports* did not yet have it. The main problem is that Apple dropped carbon support from XCode.

A cocoa version that plays well with XCode 4.4 and later can be found at `http://wxpython.org/download.php`. Click the link `wxPython2.9-osx-cocoa-py2.7` and download `wxPython2.9-osx-2.9.4.0-cocoa-py2.7.dmg`. For "security" reasons, Mountain Lion will not install the package using the GUI, but the command line installer will do the job after some complaining.

```
prompt> sudo install -pkg wxPython2.9-osx-cocoa-py2.7.pkg -tgt /
```

This puts wxPython2.9 (and wxWidgets2.9) in `/usr/local/lib/wxPython-2.9.4.0` with a symlink to `/usr/local/lib/wxPython`.

It may be necessary to add `/usr/local/lib/wxPython/lib/python2.7/site-packages` to `$PYTHONPATH`.