Annex

GNU Radio and its tools have been configured on Intel Core i5 at 1.8 GHz 5 running Ubuntu 14.04.5 LTS. Steps of installation have to be introduced giving highest attention for each command. We installed source codes of GNU Radio using PyBOMBS, which allowed us to avoid dependency problems.

Install and configuration of GNU Radio and its useful tools

1. Installing and update PIP and PyBOMBS

$ sudo apt-get install python-pip

$ sudo pip install -U pip

$ sudo pip install PyBOMBS

1. Initiating the default prefix, where the gnuradio shall be mapped

$ sudo pybombs prefix init /opt/usr -a usrlocal

$ sudo pybombs config default\_prefix /opt/usr

1. Adding recipes

$ sudo pybombs recipes add gr-recipes git+https://github.com/gnuradio/gr-recipes.git

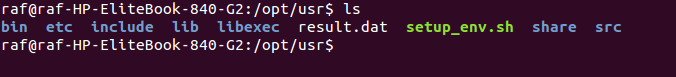
$ sudo pybombs recipes add gr-etcetera git+https://github.com/gnuradio/gr-etcetera.git

1. Installing UHD, GNURadio and all related packages
   1. This step should take more time, please wait!

$ sudo pybombs install uhd gnuradio

$ sudo ldconfig

1. Your GNU Radio is now present with sources in your hard disk. You shall have your files as on the ScreenShot.



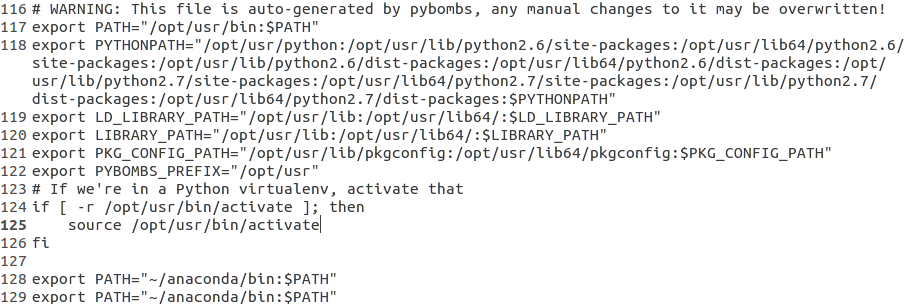
1. Now you have to run setup\_env.sh to change the environment parameters.

$ source ./setup\_env.sh

1. Check your environment parameters in your .bashrc file
   1. Go to your home position and open the file .bashrc

$ gedit ~\.bashrc

* 1. If the environment parameters are not up to date, you have to add the content of the setup\_env.sh to the bashrc file.



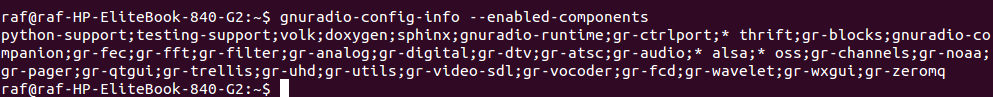
1. When you introduce on the shell comamnd $ gnuradio-companion, you should have gnuradio companion interface.

$ gnuradio-companion

1. Check configurations of gnuradio

$ gnuradio-config-info --enabled-components

You have to obtain the following configuration



1. Configuration of USRP drivers UHD. You have to update the firmware database in order to make sure that the firmwares are up-to-date, you can run the following command.

$ sudo "/usr/local/lib/uhd/utils/uhd\_images\_downloader.py"

1. Copy the uhd-usrp.rules file into /etc/udev/rules.d/ and then reload the udev rules:

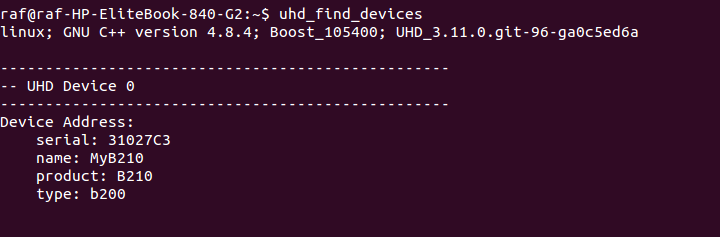
$ sudo cp uhd-usrp.rules /etc/udev/rules.d/

$ sudo udevadm control --reload-rules

Disconnect and Connect the USRP board.

1. Connect your USRP B210 and test the command

$ uhd-find-devices or $ uhd\_usrp\_probe



1. Setting up real-time scheduling Since the UHD needs a high priority of execution, it is recommended to add the following line to /etc/security/limits.conf.

@usrp - rtprio 99

1. Hence, it is necessary to create the group usrp and add to this group any user willing to use the USRP hardware.

$ sudo groupadd usrp

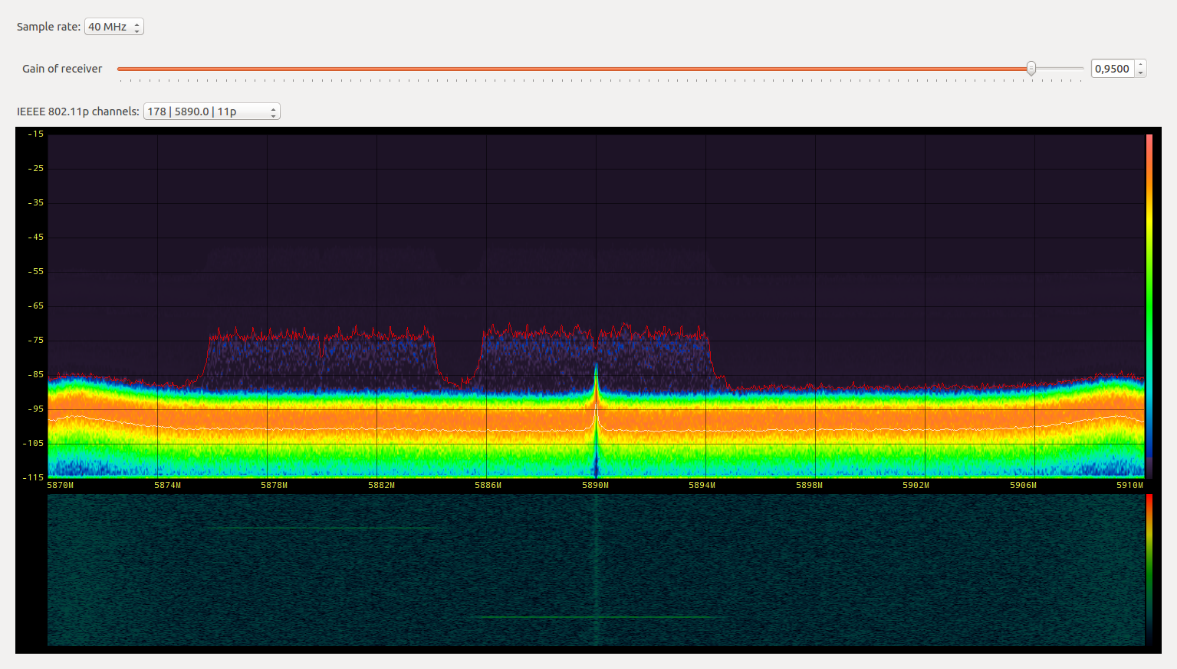
$ sudo adduser <my-login> usrp

At this time, the user shall close and reopen his session. Inside a shell, he can also issue the following instruction

su - <my-login>

The system is now ready for adding an USRP (see documentation for the appropriate model...).

1. Spectrum analyser with waterfall view via fosphore view.
   1. INSTALL fosphor View
   2. You should follow exactly the steps here: <http://sdr.osmocom.org/trac/wiki/fosphor>
   3. If the link is not available, find bellow the corresponding steps.



* GLFW3

Install dependencies

$ sudo apt-get install cmake xorg-dev libglu1-mesa-dev alien

Build GLFW

$ git clone https://github.com/glfw/glfw

$ cd glfw

$ mkdir build

$ cd build

$ cmake ../ -DBUILD\_SHARED\_LIBS=true -DCMAKE\_INSTALL\_PREFIX=/opt/usr

$ make

$ sudo make install

$ sudo ldconfig

* Intel CPU OpenCL

$ mkdir $HOME/tmp

$ cd $HOME/tmp

$ wget http://registrationcenter.intel.com/irc\_nas/4181/opencl\_runtime\_14.2\_x64\_4.5.0.8.tgz

$ tar xf opencl\_runtime\_14.2\_x64\_4.5.0.8.tgz

$ cd pset\_opencl\_runtime\_14.1\_x64\_4.5.0.8/rpm

$ cd rpm

$ alien --to-tgz opencl-1.2-base-pset-4.5.0.8-1.noarch.rpm

$ tar xf opencl-1.2-base-4.5.0.8.tgz

$ sudo mv -v opt/intel /opt

$ rm -rf opt

$ alien --to-tgz opencl-1.2-intel-cpu-4.5.0.8-1.x86\_64.rpm

$ tar xf opencl-1.2-intel-cpu-4.5.0.8.tgz

$ sudo mkdir -p /etc/OpenCL/vendors

$ sudo mv opt/intel/opencl-1.2-4.5.0.8/etc/intel64.icd /etc/OpenCL/vendors/

$ sudo mv opt/intel/opencl-1.2-4.5.0.8/lib64/\* /opt/intel/opencl-1.2-4.5.0.8/lib64/

$ rm -rf opt

* gr-fosphor

Build gr-fosphor

Build dependencies: gnuradio-dev, opencl-headers, libboost-system-dev, libboost-thread-dev.

$ git clone git://git.osmocom.org/gr-fosphor

$ cd gr-fosphor

$ mkdir build

$ cd build

sudo apt-get install -y opencl-headers

$ cmake ../ -DCMAKE\_INSTALL\_PREFIX=/opt/usr

$ make

$ sudo make install

$ sudo ldconfig

$ Build benchmark tool

Build dependencies: libglfw3-dev.

$ cd gr-fosphor/lib/fosphor

$ make LDFLAGS=-L/opt/intel/opencl-1.2-4.5.0.8/lib64

Fosphor is known to build successfully on Linux, OSX and Windows.

Install and configuration of GR-IEEE802.11p For ITS-G5

Mainly the steps of configuration are those presented in <https://github.com/bastibl/gr-ieee802-11>, but they could disappear from the Bastian Bloessl github, because the project is an open source github with GPL license. Of course this part has been already developed by research community on IEEE 802.11.

* Swig

Swig is required to create the python bindings.

$ sudo apt-get install swig

* log4cpp

I use the new [logging feature](http://gnuradio.org/doc/doxygen/page_logger.html) of GNU Radio which relies on log4cpp. This should be an optional dependency some day, but currently it is required. You can install it with

$ sudo apt-get install liblog4cpp5-dev

* GNU Radio v3.7

You need at least version 3.7.3 There are several ways to install GNU Radio. You can use [pybombs](http://gnuradio.org/redmine/projects/pybombs/wiki) as explained in our previous section of this Annex.

* gr-foo

I have some non project specific GNU Radio blocks in my gr-foo repo that are needed. For example the Wireshark connector. You can find these blocks at <https://github.com/bastibl/gr foo>. They are installed with the typical command sequence:

$ git clone https://github.com/bastibl/gr-foo.git

$ cd gr-foo

$ mkdir build

$ cd build

$ cmake ../ -DCMAKE\_INSTALL\_PREFIX=/opt/usr

<https://github.com/bastibl/gr-foo/issues/6> : branche master

$ make

$ sudo make install

$ sudo ldconfig

* Installation of gr-ieee802-11

To actually install the blocks do

$ git clone git://github.com/bastibl/gr-ieee802-11.git

$ cd gr-ieee802-11

$ mkdir build

$ cd build

$ cmake ../ -DCMAKE\_INSTALL\_PREFIX=/opt/usr

$ make

$ sudo make install

$ sudo ldconfig

* Adjust Maximum Shared Memory

Since the transmitter is using the Tagged Stream blocks it has to store a complete frame in the buffer before processing it. The default maximum shared memory might not be enough on most Linux systems. It can be increased with

$ sudo sysctl -w kernel.shmmax=2147483648

* OFDM PHY

The physical layer is encapsulated in a hierarchical block to allow for a clearer transceiver structure in GNU Radio Companion. This hierarchical block is not included in the installation process. You have to open /examples/wifi\_phy\_hier.grc with GNU Radio Companion and build it. This will install the block in ~/.grc\_gnuradio/.

* Check message port connections

Sometime the connections between the message ports (the gray ones in GNU Radio Companion) break. Therefore, please open the flow graphs and assert that everything is connected. It should be pretty obvious how the blocks are supposed to be wired. Actually this should not happen anymore, so if your ports are still unconnected please drop me a mail.

NOMA and SIC prototyping

This part is complementary to the details given in the deliverable Section III and IV. The attached folder contains the related flow graphs and simulation code written in Python. The content of **GRdeliverable.zip** folder is.

FolderContent

* IEEE802.11PNOMA\_SIC: It contains the chains of NOMA IEEE 802.11p. The SIC needs to be improved to separate perfectly the OFDM symbols.
  + NOMA80211p.grc andNOMA80211p.py: Tx with NOMA IEEE 802.11p working and tested.
  + SIC80211p.grc and SIC80211p.py: It doesn’t work yet. It needs high power processing.
* MPSK: contains the source codes of the real world MPSK chains using USRP B210 source and sink. This part remains important even if it’s not a heart of the NOMA/SIC scheme. The wireless designer could spend much time to setup flow graphs for synchronization.
  + mpsk\_rx\_all.grc: xml GUI of Rx flow graph gathering BPSK, QPSK and 8PSK.
  + mpsk\_rx\_all.py: python fil e of the Rx mpsk\_rx\_all.grc. It’s automatically generated by GNU Radio toolkit.
  + mpsk\_tx\_all.grc: xml GUI of Tx flow graph gathering BPSK, QPSK and 8PSK.
  + mpsk\_tx\_all.py: python file of the Rx mpsk\_rx\_all.grc. It’s automatically generated by GNU Radio toolkit.
  + python file of the Rx mpsk\_rx\_all.grc. It’s automatically generated by GNU Radio toolkit.
* MPSKNOMA\_SIC: Implementation of NOMA and SIC for MPSK modulations trying to include USRPs. The first proof of concept was a prototyping of BPSK/QPSK transmitter and receivers for communicating bit stream. You can find attached the BPSK, QPSK and 8PSK transmitter/receiver’s chains. Note that, beyond 16 QAM the SDR chains have to be defined with frame generator and decoder. The raison behind this limitation has been explained in my last deliverable (Section 5.1, NOMA/SIC flow graphs). The angles , , and are twice as likely to occur than any of the other 8 phases (simply because there are two constellation points on each of these phases, versus one for the rest). Therefore, we have to adjust our decision regions for a self-built Phase Error Detector (PED) accordingly. It is not inherently impossible to correct phase of a QAM reception using actual block but we need a synchronization preamble.
  + noma\_mpsk.grc, noma\_mpsk.py
  + sic\_mpsk.grc, sic\_mpsk.py
* SIMULATIONS: This part has been implemented assuming that for each OFDM subcarrier we have a BPSK coding. The simulation allows us to show the spectrum efficiency of such superposition.
  + sic\_mpsk\_sim.grc, sic\_mpsk\_sim.py: Loopback flow graph of QPSK NOMA and SIC.
  + sic\_mpsk\_sim\_mod.py: The sic\_mpsk\_sim.py has been modified including the simulation of changing the parameters of the superposition such as *alpha*. You can see below the options which can be add as arguments of sic\_mpsk\_sim\_mod.py program. Other files .plt are GNU Plot scripts to generate curves.

