How to program a new block for GNU Radio and GNU Radio Companion

1 Introduction

We present in this document an example of development, compilation and deployment of a new GNU Radio block to be used in the GNU Radio Companion (grc) environment. The designed block corresponds to a gain amplifier called gr_my_amplifier_ff and whose gain can be changed in run-time through the graphical interface.

This work has been elaborated from the information contained in the GNU Radio web site ("http://www.gnu.org/software/gnuradio/doc/howto-write-a-block.html") together with different patches published by users of the tool. More information is also included in the README and README.hacking files. It is highly recommended to read these sources before proceeding with this document.

This document describes the whole process of creating a new module and a new block from scratch, if you intend only to add a new block to a previously existing module the process is much easier. Also we want to make it clear that this example have been tested in Ubuntu 10.04 with the 3.2.2 version of GNU Radio and GNU Radio Companion.

Together with the current example we provide some template files, which users could change (making query replace of names as explained in appendix A) in order to develop a new signal processing block. It is important to highlight that the aim of this tutorial is not to teach signal processing programming with C++ but to give a general description of the use of the GNU Radio environment.

Finally, you can download the example and the templates from the following links: http://gnuradio.org/redmine/attachments/191/gr-my-basic.zip and http://gnuradio.org/redmine/attachments/192/gr-my-blocks-template.zip . If you need some technical support authors may be contacted at: asier.alonso@tecnalia.com

2 Root Folder

Once you have downloaded the example you may open the project which contains the following directories and files:

Directories:

- *apps:* Contains test applications and examples.
- *config:* Contains different configuration files which must **not be modified.**
- *lib:* Contains source files (.cc and .h) for the developed blocks.
- *swig:* As GNU Radio combines C++ programmed blocks with Python flowgraphs, Swig tool is employed to automatically generate python interfaces for C++ blocks. .i files required by this tool are stored in this folder.
- *python:* Contains python scripts.
- *grc:* Contains the .xml files needed by the grc tool.

Each folder contains a Makefile.am file because autotools (autoconf, automake and libtool) are employed in order to automatically compile and link. Also due to this tools the following files are included in the root folder: AUTHORS, bootstrap, ChangeLog, config.guess, config.sub, configure.ac, Makefile.am, Makefile.common, Makefile.swig, Makefile.swig.gen.t, NEWS, README, README.hacking, version.sh.

3 lib Folder

This folder contains all the C++ source files (gr_my_amplifier_ff.cc and gr_my_amplifier_ff.h) and the Makefile.am for compilation. All the signal processing functionality of the block is inserted in the *work* method of the .cc file. In the amplifier this method is as follows:

```
int gr_my_amplifier_ff::work
(int noutput_items, gr_vector_const_void_star &
input_items,gr_vector_void_star &output_items) {
     const float *in = (const float *) input_items[0];
     float *out = (float *) output_items[0];
     for (int i = 0; i < noutput_items; i++){
          out[i] = in[i] * (float)d_k;
     }
     return noutput_items;
}
```

Obviously this a very simple example of a variable gain amplifier with little signal processing, but your code may be as complex as you desire.

4 swig Folder

This folder contains the files employed by the SWIG tool to create python interfaces for C++ classes. These files are: gr_my.i, gr_my_amplifier_ff.i, Makefile.am and Makefile.swig.gen. Astonishingly, the provided Makefile.swig.gen file is incorrect and must be regenerated afterwards (explained in section 6) but surely due to a configuration error the autotools need it in order to work correctly.

5 grc Folder

This folder contains the files employed by the grc tool, i.e. the definition file myBlock.xml and the compilation one Makefile.am.

myBlock.xml

- tag name: the name we want to assign to our block in the graphical interface.
- tag key: the name of the C++ block.
- tag category: The folder in which our block will be clasified in the graphical interface.
- tag import: make reference to the module.
- tag make: The invocation to the public constructor of the block.
- tag calback: Includes the methods for changing the attributes related to the

parameters inside the block.

- tag param: Characteristics of each parameter, name in the interface, internal name and type
- tag chek: Checking rules for all the parameters, e.g. param1>= 1
- tags sink and source: The example is prepared for one input and one output. Otherwise, add as many entries as inputs/outputs.

6 Configuration, Compilation and Deployment

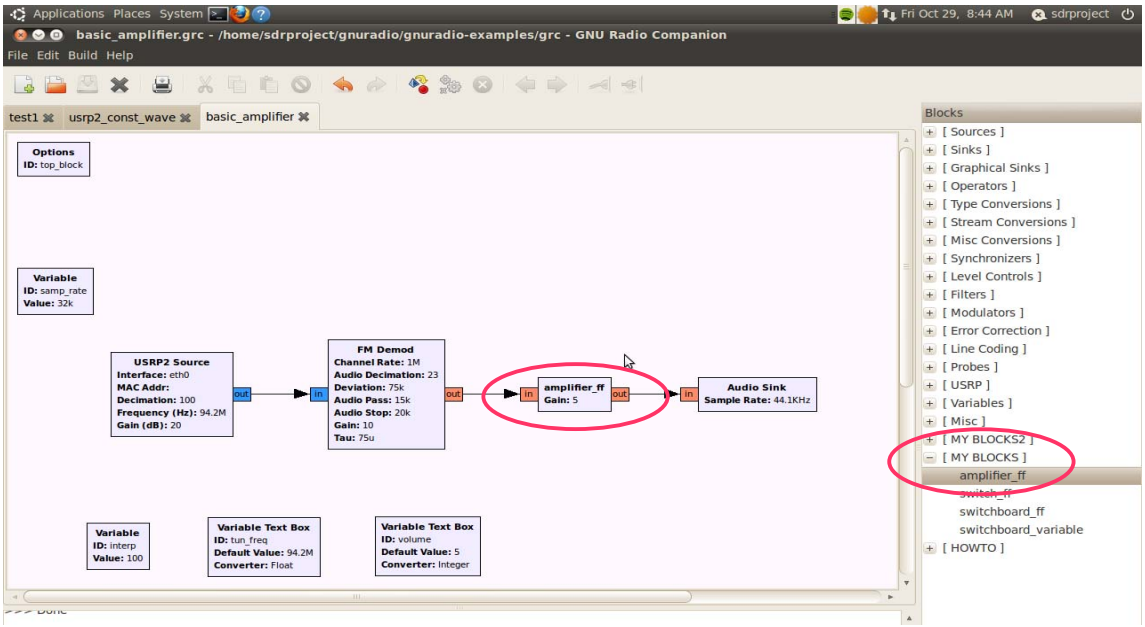
The next step is to compile and deploy the module. Open a terminal go to the project root directory and execute the following commands:

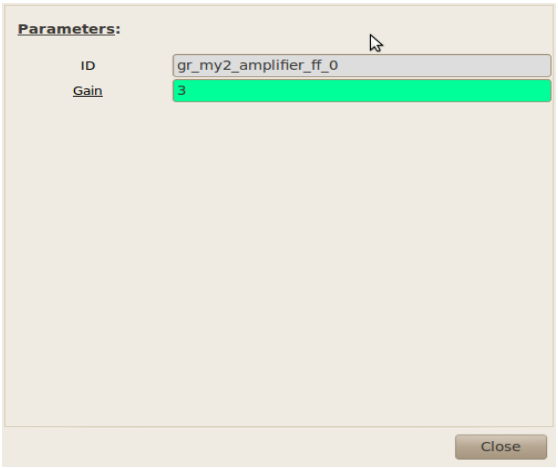
```
$ sudo ./bootstrap
$ sudo ./configure
$ cd swig
$ sudo make generate-makefile-swig
```
(this is done for regenerating the Makefile.swig.gen file in the swig folder)

```
$ cd ..
$ sudo make
$ sudo make install
$ sudo ldconfig
```
(only the first time the module is installed)

Once these commands are succesfully executed you have your block ready to be used with the gnu radio companion interface. So you can run the graphical tool and see that our module and block have been added to the list of blocks on the right. Now it is possible to add our new block to different designs and change its gain with the gnu radio companion graphical interface.

In the figure the amplifier has been added to a mock flow graph consisting of a USRP2 source, a FM demodulator, an audio sink and our block. The window to change the gain value has been opened by double-clicking in the block.





Appendix A: Templates Folder

A template folder is provided so new users can employ it to create their new blocks. The template has the following naming convention:

- *myModule:* Name of the new module to be created. GNU Radio naming conventions indicate that it should include some prefix which indicates the function of the blocks contained in the module. In our case this prefix should be nearly in all cases gr and therefore, our module should have a name like gr_my_module, as GNU radio conventions recommend "_" to separate names.
- *myBlock:* Name of the block to be developed. As in the previous case, there exist some rules for choosing an appropriate one: first of all, the name of the block should make reference to the module which it belongs to and it should have a suffix which indicates the type of its inputs and outputs (f for float, c for complex, s for short and i for integer). According to all this rules an appropriate name could be gr_my_module_my_block_ff, assuming float type inputs and outputs.

- *myFolder:* Name of the folder containing the new module. Although there are no indications for its naming. A uniform name could be gr-my-module.
- *make_myBlock:* Name of the public constructor of our block's C++ class.
- *param1:* Name of our block's input parameter (we are considering only one parameter in the template)
- *d_param1:* internal attribute of our block's class. It is permanently associated to the input parameter. According to GNU Radio naming convention the d prefix is added.
- *grcName:* Name of the developed block in the grc interface.
- *grcParam1:* internal name of the parameter in the grc.
- *grcParam1Name:* name of the parameter in the graphical interface.
- *grcParamType:* data type of the parameter.