The Gnuradio Companion (GRC)

Josh Blum

October 5, 2009

Contents

1 Introduction

GRC is a graphical tool for building gnuradio flowgraphs. Users can drag and drop gnuradio blocks into an editable flowgraph, and connect the blocks, and edit various block parameters.

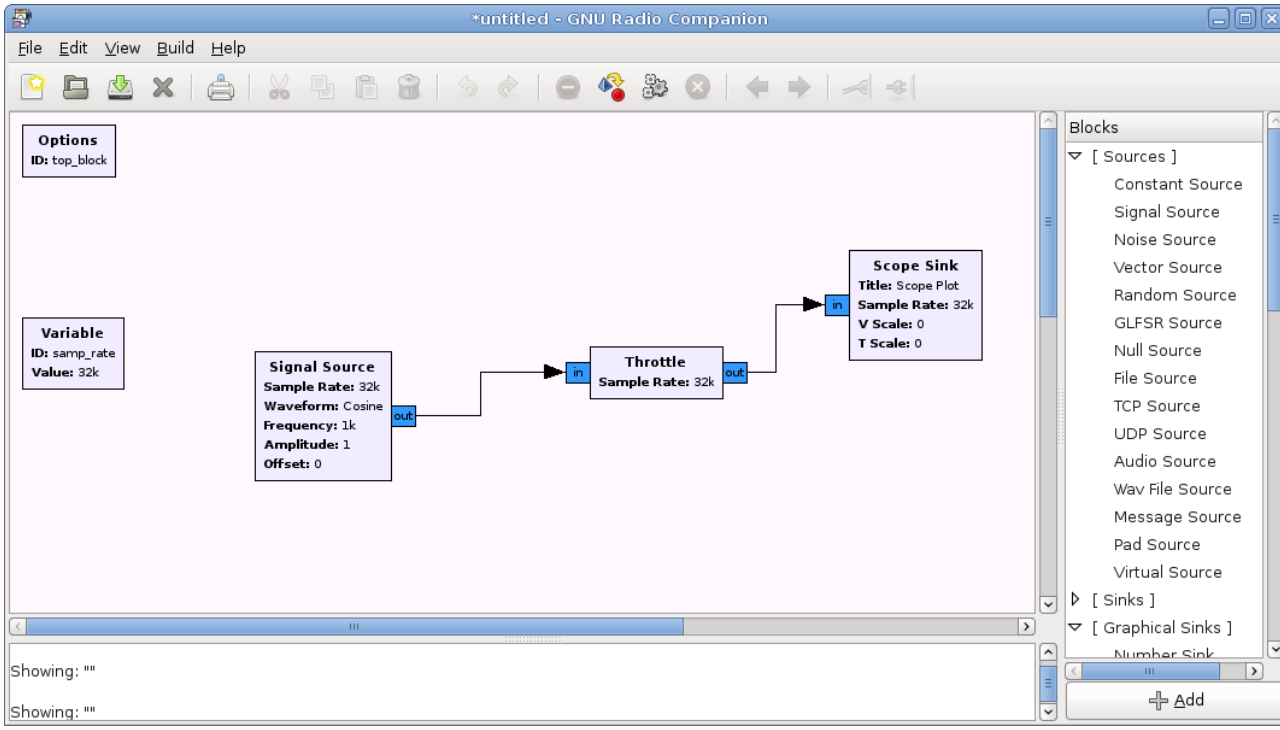

Figure 1: GRC Intro Image

1.1 Generated Code

GRC takes a flowgraph and generates the equivalent python code. GRC generates quality python code that someone could look at, learn from, and modify for custom uses.

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Top Block
# Generated: Sun Sep 20 02:31:13 2009
##################################################

from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import forms
from gnuradio.wxgui import scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class top_block(grc_wxgui.top_block_gui):

    def __init__(self, ampl=1, off=0):
        grc_wxgui.top_block_gui.__init__(self, title="Top Block")
        _icon_path = "/usr/local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        ##################################################
        # Parameters
        ##################################################
        self.ampl = ampl
        self.off = off

        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 32000
        self.freq = freq = 2000

        ##################################################
        # Controls
        ##################################################
        _freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_freq_sizer,
            value=self.freq,
            callback=self.set_freq,
            label="Frequency",
            converter=forms.float_converter(),
            proportion=0,
        )
        self._freq_slider = forms.slider(
            parent=self.GetWin(),
```

```
52              sizer=_freq_sizer,
53              value=self.freq,
54              callback=self.set_freq,
55              minimum=-samp_rate/2,
56              maximum=samp_rate/2,
57              num_steps=100,
58              style=wx.SL_HORIZONTAL,
59              cast=float,
60              proportion=1,
61          )
62          self.Add(_freq_sizer)
63
64          ##################################################
65          # Blocks
66          ##################################################
67          self.gr_sig_source_x_0 = gr.sig_source_c(samp_rate, gr.GR_COS_WAVE, freq, ampl, off)
68          self.gr_throttle_0 = gr.throttle(gr.sizeof_gr_complex*1, samp_rate)
69          self.wxgui_scopesink2_0 = scopesink2.scope_sink_c(
70              self.GetWin(),
71              title="Scope Plot",
72              sample_rate=samp_rate,
73              v_scale=0,
74              t_scale=0,
75              ac_couple=False,
76              xy_mode=False,
77              num_inputs=1,
78          )
79          self.Add(self.wxgui_scopesink2_0.win)
80
81          ##################################################
82          # Connections
83          ##################################################
84          self.connect((self.gr_sig_source_x_0, 0), (self.gr_throttle_0, 0))
85          self.connect((self.gr_throttle_0, 0), (self.wxgui_scopesink2_0, 0))
86
87      def set_ampl(self, ampl):
88          self.ampl = ampl
89          self.gr_sig_source_x_0.set_amplitude(self.ampl)
90
91      def set_off(self, off):
92          self.off = off
93          self.gr_sig_source_x_0.set_offset(self.off)
94
95      def set_samp_rate(self, samp_rate):
96          self.samp_rate = samp_rate
97          self.wxgui_scopesink2_0.set_sample_rate(self.samp_rate)
98          self.gr_sig_source_x_0.set_sampling_freq(self.samp_rate)
99
100     def set_freq(self, freq):
101         self.freq = freq
102         self._freq_slider.set_value(self.freq)
103         self._freq_text_box.set_value(self.freq)
104         self.gr_sig_source_x_0.set_frequency(self.freq)
105
106 if __name__ == '__main__':
107     parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
108     parser.add_option("", "--ampl", dest="ampl", type="eng_float", default=eng_notation.num_to
```

```
109             help="Set Amplitude [default=%default]")
110         parser.add_option("", "--off", dest="off", type="eng_float", default=eng_notation.num_to_s
111             help="Set Offset [default=%default]")
112         (options, args) = parser.parse_args()
113         tb = top_block(ampl=options.ampl, off=options.off)
114         tb.Run(True)
```

2 New Features

These are features which have been added within the last 6 months or so.

2.1 WX Notebook

GRC now has a notebook block that abstracts a wxgui notebook. WX Gui elements like scope windows, and sliders can be added to the notebook. Notebooks can be nested! So your notebook can contain another notebook and so on.

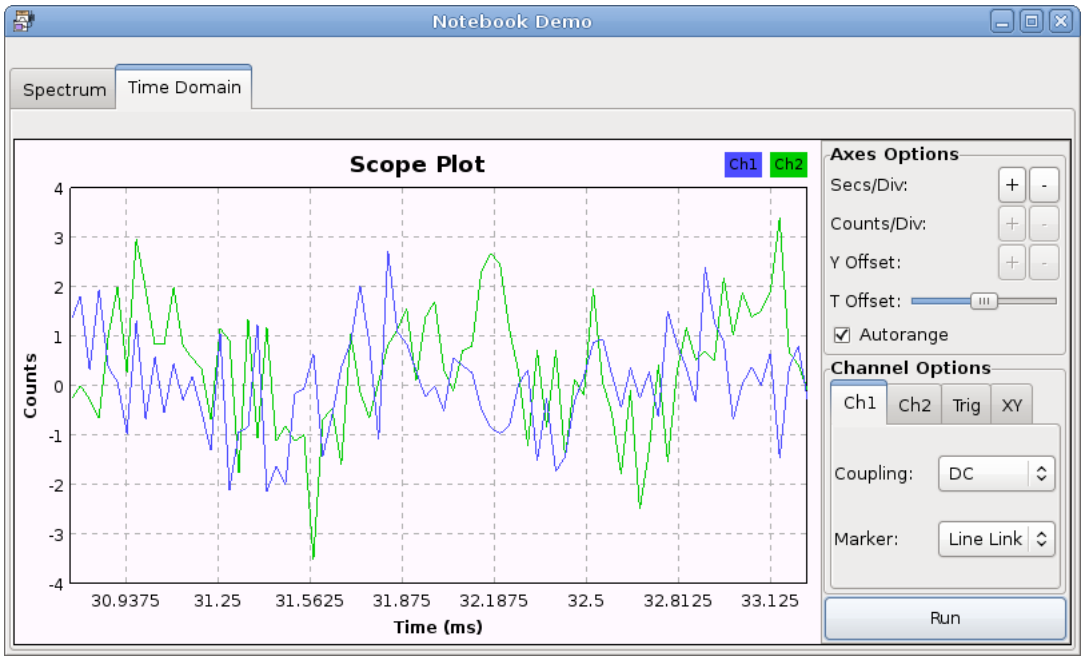

Figure 2: Notebook Demo

2.2 Implementation

Each wx element in grc has gained a "notebook" parameter. The notebook parameter is a tuple of notebook id, tab index. Where the notebook id is the unique block id of a notebook block. The notebook block itself has a notebook parameter so it can be nested within another notebook.

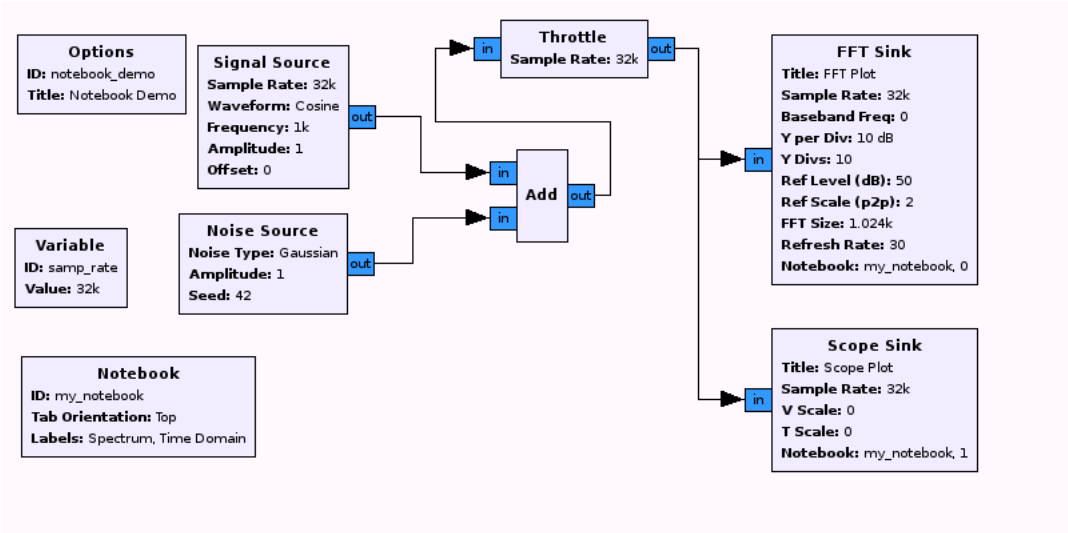

Figure 3: Notebook Flow Graph

2.3 Forms Module

2.3.1 Brief Introduction

The forms module is a gr-wxgui component that wraps standard wx python forms. Used by: wx-gui GL sinks, gr-utils (usrp siggen gui), GRC, and various other apps.

- provides convenience parameters for labels and positioning
- provides publish/subscribe interface to de-couple presentation/control
- standardizes the coding and presentation of gnuradio wx-gui apps

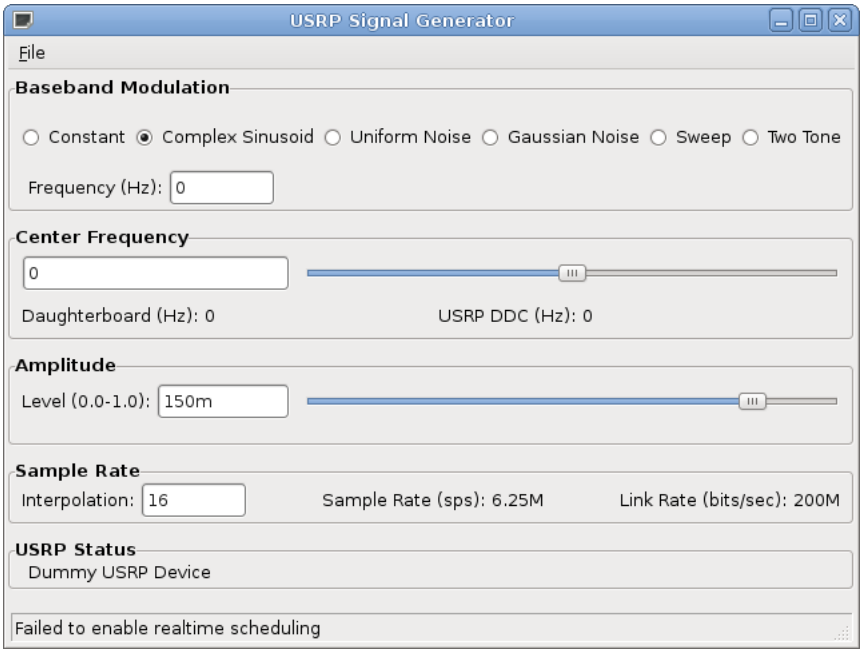

Figure 4: USRP Siggen GUI

2.3.2 Forms as GRC Variables

GRC used to come with its own special wx forms-like module called "callback controls". After integrating the new forms module into gr-wxgui, the callback controls were removed, and GRC's variable blocks were modified to generate forms code. In addition, the static text and checkbox variable blocks were added.

- Variable Slider - combines text box and linear slider form
- Variable Chooser - button, drop down, or radio buttons form
- Variable Check Box
- Variable Text Box
- Variable Static Text

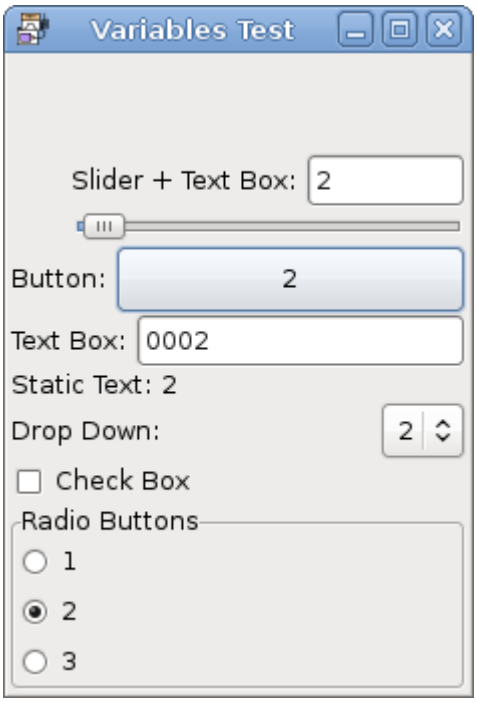

Figure 5: Variables Demo

2.4 Virtual Connections

Virtual connections allow us to connect the IO ports of blocks without an explicit, solid-line connection. This allows sections of a flow graph to be functionally separated while actually remaining connected.

The *virtual source* and *virtual sink* block allow us to create virtual connections in GRC. To create a virtual connection, the source port of a block must be connected to a virtual sink block. Then, the sink port of another block must be connected to a virtual source block. The virtual connection is created when a matching stream ID is entered into both the virtual source and virtual sink's parameters.

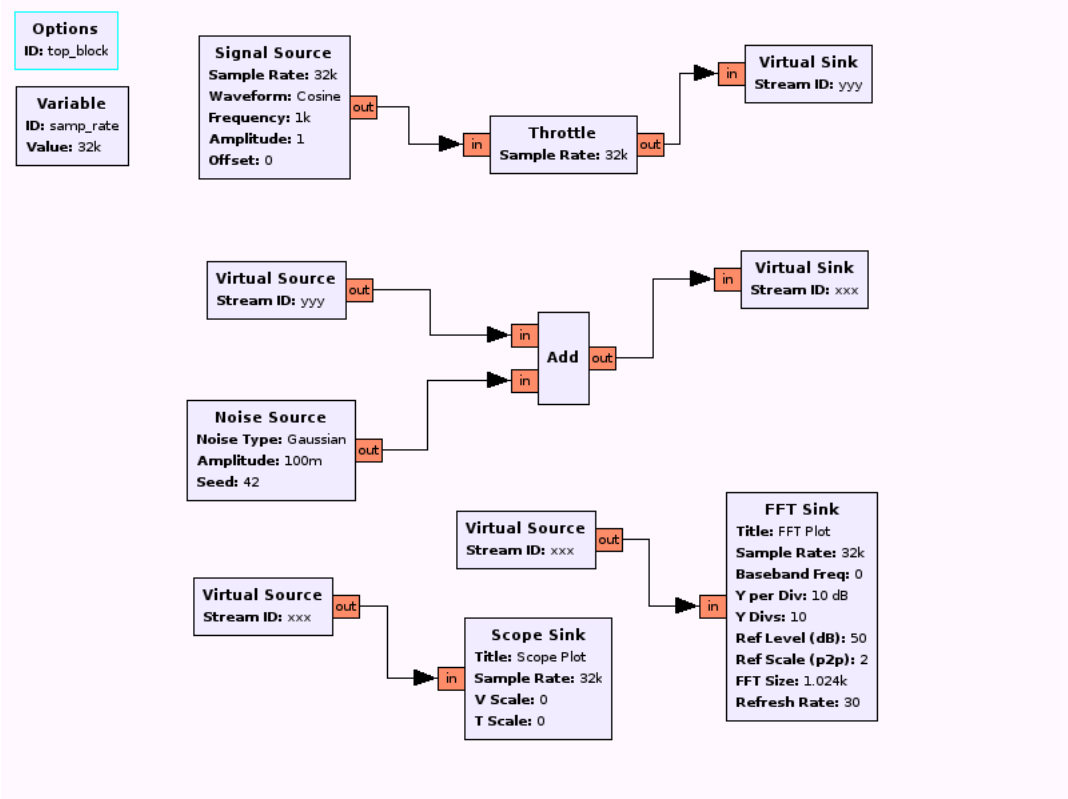

Figure 6: Virtual Connection Demo

2.4.1 Notes

- Each virtual sink block must have a unique stream ID.
- This stream ID can be re-used in any number of virtual source blocks.
- Virtual connections are resolved recursively, and therefore, may be nested.
- Virtual connections can be used split a single flowgraph among multiple pages (not yet implemented).

2.5 Message Queue Ports

GRC has gained the ability to use a message queue as a block input or output. This enabled adding a message source and message sink block wrapper to GRC. In addition, users can now create block wrappers for blocks with message queue IO.

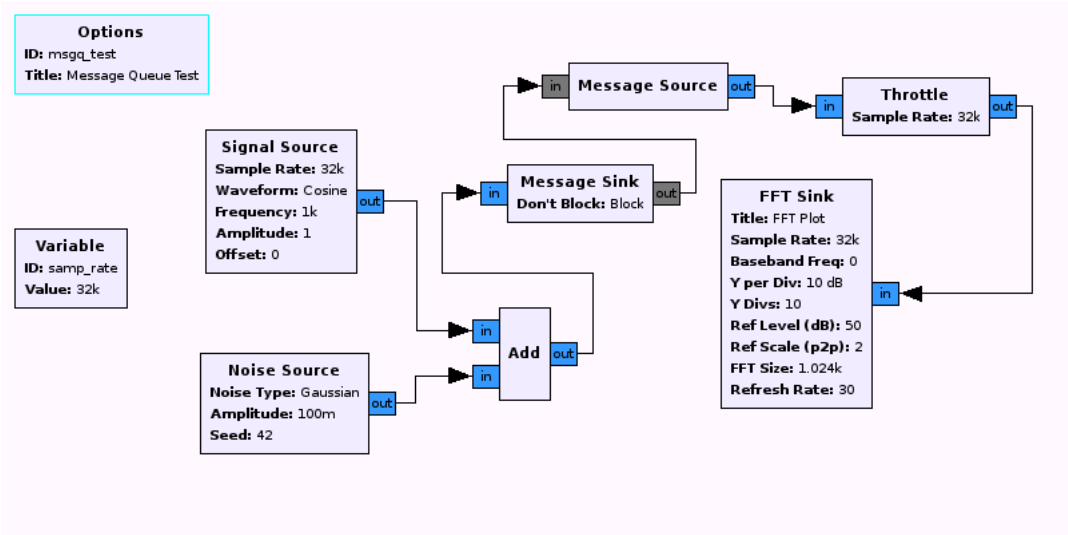

Figure 7: Message Queue Demo

2.5.1 Implementation

When the code is generated, a message queue is created for each message-based connection. The name of the queue expected by the block will be *block_id*_msgq. The block wrapper code is responsible for passing a parameter into a block constructor using a message queue with that variable name.

2.5.2 Pitfalls

- Only one message queue supported per block (one input or one output, not both)
- The message ports do not work with the virtual connections
- Message queue ports will be replaced by some PMT implementation

2.6 A few more mentionables

- Added variable config block that reads and writes to a config file
- Various speed improvements (especially when evaluating the inline python code)
- Added helpful dialogs:

  colors to data types mapping

  error summary dialog

3 Future Features

3.1 Mutiple Sheets

Currently, each flow graph has one editable window. The idea would be to allow for multiple editable windows, called "sheets". One could use multiple sheets to functionally split up the blocks in a flow graph.

Ex: A sheet for parameter blocks, a sheet for the signal processing, and another sheet for the wxgui blocks. Connected components of the flow graph can be split up as well using virtual connections.

3.1.1 Implementation

Each block will get a new hidden gui parameter called "_sheet" (other hidden gui params are _rotation and _coordinate). The value of this parameter will be a unique key identifying the sheet. This way, the underlying xml format for a saved flow graph need not change.

The gui will get a sheets notebook with tabs to select between sheets. The menu will get the following new options:

- Add Sheet - asks for sheet name and key
- Remove Sheet - removes currently selected sheet
- Rename Sheet - renames currently selected sheet

3.2 Projects

A project is a collection of flow graphs and properties. A project will contain multiple saved grc flow graphs (top blocks and hier blocks). A project will contain properties like name, key, directories, configuration files...

Projects will allow full utilization of grc's ability to produce hierarchical blocks. The project manager will automate generation and installation of the hierblock2 code. The python code will be installed into the site packages directory specified in the project. The project manager will also automate the generation of the "block wrapper" so the block will appear in grc's block tree, and can be used within grc flow graphs.

Once grc has a comprehension of the current hierarchical blocks in a project we can:

- Ctrl+DClick on a hierarchical block in a grc flow graph to open its corresponding flow graph in a new tab
- Update a top block as changes to its hierarchical blocks occur
- Rubberbanding - atomatically create hierarchical blocks from a section in a top block, the selection will be replaced with the resulting hier block

3.3 Support PMTs

The goal is to support a real message passing interface within the grc flow graph model. Graphically, this will look a lot like the message queue implementation.

Things to consider:

- What does a flow graph with PMTs look like
- How to deal with PMTs as block parameters
- Removing the message queue implementation

3.4 Adopt QtGui

GRC could support generating flow graphs with qtgui code. This would require adding new blocks for qtgui signal analysis sinks, adding new blocks for variable control (sliders, choosers, etc...), and modifying the flowgraph.tmpl file to handle generation of qtgui "setup code".

The generate options (found in the options block) would gain a new option, QT GUI. The current generate options are WX GUI, No GUI, and Hier Block.

WXGUI specific blocks

- WXGUI sinks
- Variable blocks
- Notebook block