Google Summer of Code 2017

`gr-bokehgui`: A web based GUI for GNU Radio

Kartik Patel

April 3, 2017

1 Introduction

Currently, GNU Radio works on a local system (many times connected to a hardware). The display of GNU Radio is based on QT GUI framework. In addition to that, various input widgets of QT framework are used to interact with the ongoing simulations. The QT framework constrains the input/output operations from the system running the simulation.

In software development, the new paradigm is moving towards web based systems because of simple usability for the user and wide range of available frameworks for developers. In this project, an OOT module for web-based GUI is proposed for GNU Radio. The primary focus of the project will be a display mechanism which will be used to interact with ongoing simulation based on the parameters provided through interactive HTML inputs.

In the proposal, the focus is on the flow of the final OOT module and implementation details. The details of minute tasks like the arrangement of plots & widgets inside the output, the color, and labeling of the plots etc. have been intentionally left out but will be implemented taking *gr-qtgui* module as a guideline.

1.1 Primary features of the project

1. Alternative output mechanism other than Python QT framework
2. Real-time visualization and interaction with the program remotely
3. Simultaneous real-time multi-user interaction with the program
4. Flexible module to incorporate the future development in direction web-based software

2 Proposed workflow of the module

At present, GNU Radio plots various plots in the window based on QT Framework. Similarly, the proposed module will show various plots through HTML page served from the system running the simulations. This section explains working of the module from perspectives of the User and GNU Radio.

The working prototype of entire project is available here [**?**]. The file time_sink_f.py implements basic sink for floating point input and the file top_block.py is an example on how the sinks will be used. You can review the output here.

2.1 Using the module - User's perspective

The usage of module will be similar to the current QT based GUI. The steps will be as follows:

1. The user selects the *BokehGUI* from the options menu in generation options.
2. A optional variable `session_id` will allow the user to assign a ID to the session of the program. The program will be identified by the session ID.
3. The sinks and widgets from *BokehGUI* module will be included in flowcharts according to requirements.
4. Upon running the simulation, a server process will be started.
5. The user opens the URL `server:port/?session-id=session_id` through the web browser over the network. The response will be the *Document* instance corresponding to the session identified by `session_id`.

Note: As of now, the layout of HTML will be a default layout. If time permits, I am planning to include a jQuery based library [**?**] to arrange the plots and widgets from the frontend.

2.2 Interaction with GR - GR's perspective

In this subsection, the overall backend workflow of module is explained in main GNU Radio software. In addition to that, the working of major features of the module is introduced.

In particular, this OOT module uses `Bokeh` [**?**] library will be used to setup the server, sessions, documents and plots. The overall structure and workflow of Bokeh based GUI is similar to the structure and workflow of the QT based GUI. The figure 1 provides comparison of proposed mechanism with the current structure.

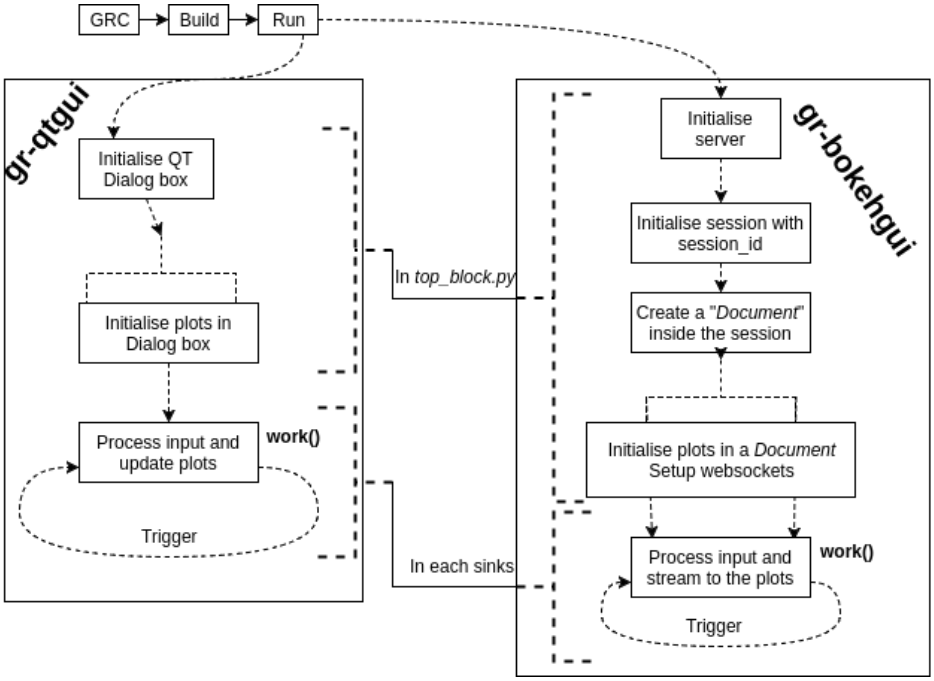

Figure 1: Comparison of `gr-qtgui` with `gr-bokehgui`

2.2.1 Understanding `top_block.py`

First of all, a GRC file builds *top_block.py* which initializes all the sinks and blocks. A example of *top_block.py* having a signal source, a throttle block and BohekGUI time sink is given in [**?**]. Important snippets of the code are explained here.

```
1 class top_block(gr.top_block):
2   def __init__(self):
3     gr.top_block.__init__(self, "Top Block")
4
5     self.doc = curdoc()
6     self.session = push_session(self.doc, session_id=session_id)
```

Here, the line #5 and #6, initialize a *Document* instance and a *Session* instance. All the clients connected to the session identified by `session_id`, will have a single *Document* instance. Hence, there is only one Document and session instance on the server. It implies changing the parameters or plot configurations on one client will ensure the change is displayed to all clients viewing the same document.

```
1 self.htmlgui_time_sink_f_0 = htmlgui.time_sink_f(self.doc, 0.1)
2
3 # Followed by code to configure the plots of the sink
```

It initializes and configures the time sink. More details on the sink

implementation is explained in following section.

2.2.2 Proposed implementation of sinks and widgets

Consider a sample floating point `time_sink_f.py` in the following snippet.

```
class time_sink_f(gr.sync_block):
  def __init__(self, doc, update_time):
    gr.sync_block.__init__(self,
            name="time_sink_f",
            in_sig=[numpy.float32],
            out_sig=None)

    self.doc = doc # the Document instance of the program

    self.plot = None
    self.ds = None # datasource instance to the clients
    self.initialize()
    # other configurations

  def initialize(self): # initialize the plot and datasource
    self.plot = figure()
    self.ds = ColumnDataSource(data=dict(x=[], y=[]))
    self.plot.line(x='x', y='y', source = self.ds)
    self.doc.add_root(self.plot) # Add plot to the document

  # Other methods to configure the plots
```

A constructor of the sink that connects the sink to *Document* instance. Initialize the plots and data source.

```
  def work(self, input_items, output_items):
    new_data = dict()
    # Processing the input_items
    self.ds.stream(new_data, rollover = 1000)
            # streaming to the Document instance
    return len(input_items[0])
```

The function `work()` in the sink instance processes the data and stream the data to the Document instance through the datasource.

In the sample program, only Python is used to explain the idea. But in general, Python is much slower than C++. Hence, the OOT module will have both Python and C++ code as structured in figure 2. Python will be used to interact with the plots and other blocks whereas C++ will be used to pre-process the data before sending to the plots. This will ensure fast processing and easy implementation.

1. **sink_impl** : C++ class which will be inherited. Contains functions related to processing of the data. Note that the processing part of the sink will be almost same as the data processing in QT based GUI.

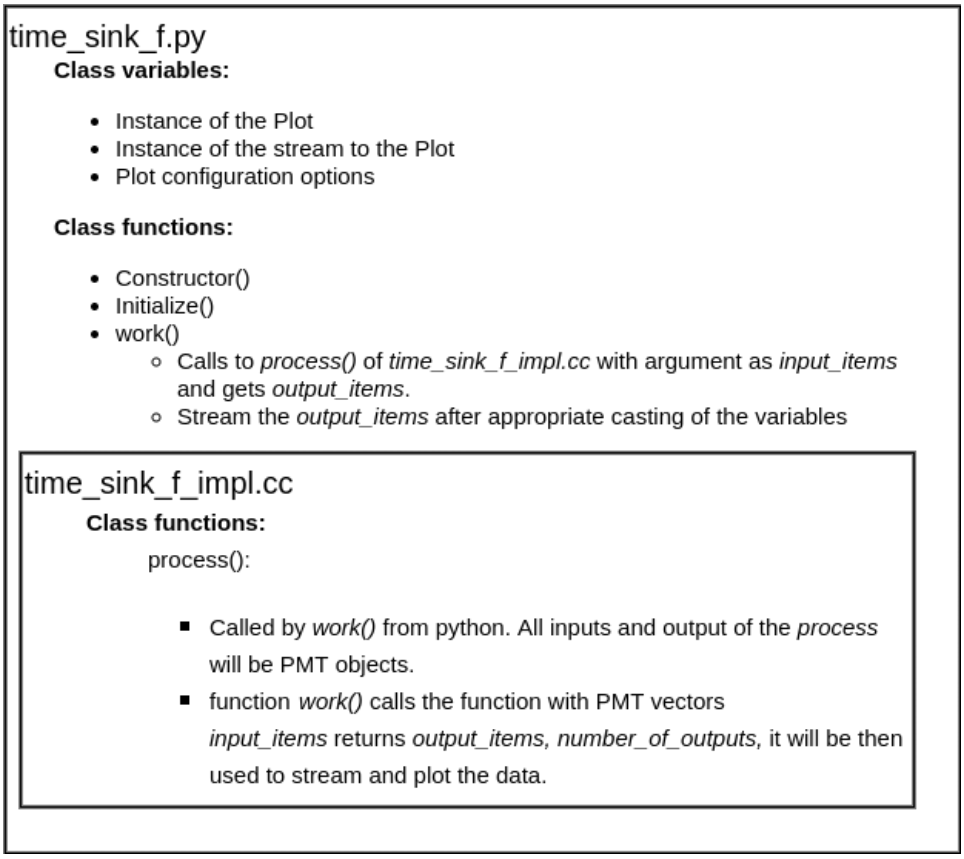

Figure 2: Proposed structure and flow of the program

2. **sink** : Python class inherited from **sink_impl** class containing methods to configure and communicate with the plots. All processing of the data will be done in functions of **sink_impl**.

3. **widgets** : Bokeh provides all required input widgets with appropriate mechanism for javascript as well as backend callbacks. Hence, this class will take care of all such widgets and their corresponding python callback functions.

4. **forms/toolbars** (If time permits) : The group of widgets that forms a toolbar. For an instance: in frequency plots, one can have a range slider for the range of frequency of interest, button for "autorange" and similarly more. Grouping all these makes the system modular and easy to extend.

Note: The *sink_impl* and *sink* is a generic term to point all sinks. e.g. in case of floating point time sink, it will be *time_sink_f_impl* and *time_sink_f*.

Based on this, overall flow of the OOT module `gr-bokehgui` is as follows:

1. GRC creates a *top_block.py*.
2. *top_block.py* initialize the server, session, doc and sinks.
3. The sinks have instance of the corresponding doc, and initialises plot.
4. The *work()* function process the *input_items* and stream the *output_items* to the plot.
5. Connect the system through the browser using specific URL.

Please note that, at present, it is considered that the output plots are displayed on the same system or on the small network. In other words, the application for the output over the Internet is not considered for now. Hence, it can be safely assumed that the samples will not be required to be dropped. Also, during streaming of the data, re-streaming will not be necessary.

3 The proposed work during GSoC 2017

A summary of proposed features are as follows:

1. Following plots will be implemented for the web based GUI. Most plots are directly available in Bokeh library. Others can be developed by providing the formatted input values to existing plots in Bokeh library.
    - Constellation Display
    - Histogram Display
    - BER Sink
    - Frequency Display
    - Time Display
    - Waterfall Display (If time permits)
2. Following input widgets will be implemented for the web based GUI. All these widgets are already available in Bokeh library.
    - Checkbox
    - Chooser (Drop-down menu in HTML)
    - TextBox
    - Label (Non-editable textbox)
    - Push Button
    - Range Slider

- Slider

3. Add GRC blocks for the Sinks
   - Add option in *Options* block and define building of *top_block.py*
   - Add GRC blocks for each sinks and input widgets mentioned above

4 Timeline

The timeline provided by Google suggests a 1 month of community bonding period. But since, the registration for graduate studies at University of Texas at Austin are around end of August, I will be available for the duration of May-August except 3-4 days sometimes in June or July to complete my visa process. Although I will keep on contributing to GNU Radio, I would like to start coding from 20th May (10 days earlier than suggested date), so that the three month duration ends at 20th August to avoid any loss of work during the registration process at the University.

The necessary documentation will be done in parallel to to the development. According to the timeline of GSoC 2017, there are 13 weeks of coding period. I scheduled the deliverables into 1-week period, planning to work full-time from Monday to Saturday and using the Sunday as additional time buffer. My tentative GSoC timeline is given below:

Table 1: Timeline

May 4 - May 20	Define minute details of the projects like plot configuration parameters
	Understanding why *Cyberspectrum is the best spectrum.*
May 21 - May 27	Initial setup: Define "Generate Options" for Bokeh GUI
	Complete coding `time_sink_f_impl.cc`
	Work on Python part of time sink for float inputs
May 28 - June 3	Complete time sink for float inputs
	Conclude time sink for complex inputs
	Create GRC block for Time Sink

June 4 - June 10	Add Python and GRC example for time sink
	Complete input widgets: Label and textbox
June 11 - June 17	Define GRC blocks for Label and textbox
	Start frequency sink
June 18 - June 24	Conclude frequency sink
	Add GRC block for frequency sinks
	Add Python and GRC example for frequency sink
June 25 - July 1	Complete Constellation sink
July 2 - July 8	Add GRC block for Constellation sink
	Add Python and GRC example for Constellation sink
	Start BER sink
July 9 - July 15	Complete BER sink
	Add GRC block for BER sink
	Add Python and GRC example for BER sink
July 16 - July 22	Conclude Checkbox and add GRC block
	Conclude Chooser (dropdown) and add GRC block
July 23 - July 29	Conclude Push button and add GRC block
	Conclude slider and add GRC block
July 30 - August 5	Complete Historgram sink
	Add GRC block for Histogram sink
	Add Python and GRC example for histogram sink
August 6 - August 12	Conclude range slider and add GRC block
	Complete remaining tasks
August 13 - August 20	Wrap up the project
	Submit the final report

I am also planning to publish my weekly progress on *discuss-gnuradio forum* in order to keep my work transparent.

5 License

The entire project will be open-source, available on GitHub, included the GPLv3 licensed code of GSoC.

6 Acknowledgement

I have read the rules of conduct for GSoC of GNU Radio and acknowledge the tree strikes rule. Therefore I am going to intensively communicate with the mentor and keep my work transparent and my working progress up to date.

7 Personal background and previous experience

I am a final year undergraduate student at Department of Electronics and Communication Engineering, Indian Institute of Technology Roorkee, India. I will be joining University of Southern California for Ph.D. in Electrical and Computer Engineering with majors in Electrical Engineering. My area of interests revolve around Communication systems and I have developed web and software development as my hobby.

I am proficient in 3 human languages including English and many computer languages including Python, C++, Javascript, and HTML/CSS. I will be connected to Internet throughout GSoC period. Overall, I will be always available over the email for any discussion or questions. In addition to that, I can be available on Skype and Google Hangout whenever necessary.

My experience in programming and in particular open-source development is as follows:

- **Implementation of Bluetooth Low Energy module in NS3** (Documentation)
  Designed and implemented Bluetooth Low Energy protocol stack in NS3. Initially developed the basic idea and then implemented entire module within 4 weeks.
- **Chief Technical Lead, Information Management Group (IMG), IIT Roorkee** - IMG develops and maintains the IIT Roorkee Intranet & Internet systems. We manage the Institute website, Content Management System, Placement Portal and many other official applications for the institute.
  - I am responsible for all aspects of the backend stack including performance management and security.
  - Initiated changes in development cycle to optimize the resource usage of servers and reduce the load on servers and databases.
  - Developed and maintained the official placement portal of the institute containing more than 20,000 lines of codes. The student side interface which includes company information, application details and the interface for placement office which includes contact manager and related administrative tasks were developed based on Python/Django framework.

- Since January 2017, I contributed to GNU Radio in order to get familiar with organization and codebase. Following are my contributions to the code:
  - Pull request with the feature to *Duplicate flowgraph and Save a Copy*(PR: # 1188).
  - Pull request with the change `cout` to `gr::logger` (PR: # 1178)
  - Solved issue # 1124: Added plot configuration options for Line2 in case of input type Complex Messages in QT GUI Time Sink block.
  - Solved issue # 1192: Removed redundant configuration options in QT GUI Time Sink block.

  In addition to the contributions to the code, I have been involved in the GNU Radio mailing list to get in touch with the community. Although I had no need to ask doubts on the forum because of excellent tutorials and documentations available over the Internet, I have tried to answer the queries based on my knowledge and capacity. I am highly interested to continue the contributions to GNU Radio even after the GSoC period.

Note: Due to copyright and other constraints, the code for BLE stack in NS3 and code by IMG, IIT Roorkee can't be published.

I have also worked on several research projects. For details of projects and other details, please refer to my resume available here.

Contact details

Address	:	Ahmedabad, Gujarat, India
Email	:	kartikpatel1995@gmail.com
Website	:	http://kartikpatel.in
Github	:	https://github.com/kartikp1995/
Skype ID	:	kartikp1995
LinkedIn	:	https://www.linkedin.com/in/kkpatel95/

8 Conclusion

An overview of the module for web based GUI for GNU Radio is given in the previous sections. With example of a small scale implementation, the flow of development is explained. All tasks are divided into proper timeline so that mentor(s) and community can track the progress of the project.