Chapter 17

GNU Radio for Cognitive Radio Experimentation
Outline

- What is GNU Radio?
- Basic Concepts
- GNU Radio Architecture & Python
- Dial Tone Example
- Digital Communications
- Cognitive Transmitter
What is GNU Radio?

- Software toolkit for signal processing
  - Software radio construction
  - Rapid development
  - Cognitive radio

- USRP (Universal Software Radio Peripheral)
  - Hardware frontend for sending and receiving waveforms

“Cognitive Radio Communications and Networks: Principles and Practice”
GNU Radio Components

<table>
<thead>
<tr>
<th>Hardware Frontend</th>
<th>Host Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF Frontend (Daugtherboard)</td>
<td>GNU Radio Software</td>
</tr>
<tr>
<td>ADC/DAC and Digital Frontend (USRP)</td>
<td></td>
</tr>
</tbody>
</table>

“Cognitive Radio Communications and Networks: Principles and Practice”
GNU Radio Software

- Opensource software (GPL)
  - Don't know how something works? Take a look!
  - Existing examples: 802.11b, Zigbee, ATSC (HDTV), OFDM, DBPSK, DQPSK

- Features
  - Extensive library of signal processing blocks (C++/ and assembly)
  - Python environment for composing blocks (flow graph)

“Cognitive Radio Communications and Networks: Principles and Practice”
GNU Radio Hardware

- Sends/receives waveforms
- **USRP Features**
  - USB 2.0 interface (480Mbps)
  - FPGA (customizable)
  - 64Msps Digital to Analog converters (receiving)
  - 128Msps Analog to Digital converters (transmitting)
  - *Daughterboards* for different frequency ranges

- **Available Daughterboard**
  - 400-500Mhz, 800-1000Mhz, 1150-1450Mhz, 1.5-2.1Ghz, 2.3-2.9Ghz

“Cognitive Radio Communications and Networks: Principles and Practice”
GNU Radio Hardware Schematic

“Cognitive Radio Communications and Networks: Principles and Practice”
Outline

- What is GNU Radio?
- *Basic Concepts*
- GNU Radio Architecture & Python
- Dial Tone Example
- Digital Communications
- Cognitive Transmitter

"Cognitive Radio Communications and Networks: Principles and Practice"
Basics: Blocks

- **Signal Processing Block**
  - Accepts 0 or more *input streams*
  - Produces 0 or more *output streams*

- **Source**: No input
  - *noise_source*, *signal_source*, *usrp_source*

- **Sink**: No outputs
  - *audio_alsa_sink*, *usrp_sink*
Basics: Data Streams

Blocks operate on *streams* of data
Basics: Data Types

- Blocks operate on certain data types
  - char, short, int, float, complex
  - Vectors

- Input Signature:
  - Data types for input streams

- Output Signature:
  - Data types for output streams

Diagram:
- Two streams of float
- One stream of complex
Basics: Flow Graph

- Blocks composed as a *flow graph*
  - Data stream flowing from *sources* to *sinks*
Outline

- What is GNU Radio?
- Basic Concepts
- GNU Radio Architecture & Python
- Dial Tone Example
- Digital Communications
- Cognitive Transmitter

“Cognitive Radio Communications and Networks: Principles and Practice”
GNU Radio Architecture

- GNU radio has provided some useful APIs
- What we are interested in at this time is how to use the existing modules that has been provided in GNU radio project to communicate between two end systems

"Cognitive Radio Communications and Networks: Principles and Practice"
GNU Radio Architecture - software

- How these modules co-work?
  - Signal processing block and flow-graph
    - C++: Extensive library of signal processing blocks
      - Performance-critical modules
    - Python: Environment for composing blocks
      - Glue to connect modules
      - Non performance-critical modules
Python scripting language used for creating "signal flow graphs"

C++ used for creating signal processing blocks
- An already existing library of signaling blocks

The scheduler is using Python’s built-in module threading, to control the ‘starting’, ‘stopping’ or ‘waiting’ operations of the signal flow graph.
GNU Radio Architecture – software(3)

“Cognitive Radio Communications and Networks: Principles and Practice”
Python - running

- Why Python?
  - Object-oriented
  - Free
  - Mixable (python/c++)

- Python scripts can be written in text files with the suffix .py

  Example:
  - $ python script.py
  - This will simply execute the script and return to the terminal afterwards
Python - format

- Module: a python file containing definitions and statements
  - `from pick_bitrate import pick_tx_bitrate`
    (from file import function)
  - `from gnuradio import gr, (or *)`
    (from package import subpackage or all)
  - Some modules are built-in e.g. sys (`import sys`)

- Indentation: it is Python’s way of grouping statements
  - Example:
    - `while b < 10:
      print b
      return`
  - Body of loop has to be indented by the same amount to indicate statements of the loop

"Cognitive Radio Communications and Networks: Principles and Practice"
Python – function & class (1)

- Function definitions have the following basic structure:
  ```python
def func(args):
    return values
  ```

- Regardless of the arguments, (including the case of no arguments) a function call must end with parentheses.

- Example:
  ```python
def f1(x):
    return x*(x-1)
f1(3) = 6
  ```
Python – function & class (2)

☐ Classes

class ClassName:
    <statement-1>
    ...
    <statement-N>

☐ Objects

x = ClassName() creates a new instance of this class and assigns the object to the variable x

Initial state: for instantiation and parameter pass

def __init__(self):

"Cognitive Radio Communications and Networks: Principles and Practice"
Creating Your Own Signal Block

- Basics
  - A block is a C++ class
  - Typically derived from gr_block or gr_sync_block class

- Three components
  - my_block_xx.h: Block definition
  - my_block_xx.cc: Block implementation
  - my_block_xx.i: Python bindings (SWIG interface)
    (SWIG, a tool that generates wrapper code around your C++ functions and classes, so that they are callable from Python)
Outline

- What is GNU Radio?
- Basic Concepts
- GNU Radio Architecture & Python
- Dial Tone Example
- Digital Communications
- Cognitive Transmitter

"Cognitive Radio Communications and Networks: Principles and Practice"
Dial Tone Example

Sine Generator (350Hz)

Sine Generator (440Hz)

Audio Sink
class my_top_block(gr.top_block):
    def __init__(self):
        gr.top_block.__init__(self)

        parser = OptionParser(option_class=eng_option)
        parser.add_option("-O", "--audio-output", type="string", default="",
                         help="pcm output device name.  E.g., hw:0,0")
        parser.add_option("-r", "--sample-rate", type="eng_float", default=48000,
                         help="set sample rate to RATE (48000)")

        (options, args) = parser.parse_args ()
        if len(args) != 0:
            parser.print_help()
            raise SystemExit, 1

        sample_rate = int(options.sample_rate)
        ampl = 0.1

        src0 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 350, ampl)
        src1 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 440, ampl)
        dst = audio.sink (sample_rate, options.audio_output)
        self.connect (src0, (dst, 0))
        self.connect (src1, (dst, 1))

if __name__ == '__main__':
    try:
        my_top_block().run()
    except KeyboardInterrupt:
        pass
Dial Tone Example (1)

```
from gnuradio import gr
from gnuradio import audio
from gnuradio.eng_option import eng_option
from optparse import OptionParser
```
Dial Tone Example (2)

class my_top_block(gr.top_block):
    def __init__(self):
        gr.top_block.__init__(self)

Define container for Flow Graph;
gr.top_block class maintains the graph
Define and parse command-line options

```python
define_and_parse_command_line_options:
    parser = OptionParser(option_class=eng_option)
    parser.add_option("-O", "--audio-output", type="string", default="",
                      help="pcm output device name. E.g., hw:0,0")
    parser.add_option("-r", "--sample-rate", type="eng_float", default=48000,
                      help="set sample rate to RATE (48000)")
    (options, args) = parser.parse_args ()
    if len(args) != 0:
        parser.print_help()
        raise SystemExit, 1
```
Dial Tone Example (4)

Create and connect signal processing blocks

```
sample_rate = int(options.sample_rate)
ampl = 0.1

src0 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 350, ampl)
src1 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 440, ampl)
dst = audio.sink (sample_rate, options.audio_output)
sel.connect (src0, (dst, 0))
sel.connect (src1, (dst, 1))
```
Dial Tone Example (5)

```python
if __name__ == '__main__':
    try:
        my_top_block().run()
    except KeyboardInterrupt:
        pass
```

Run the flow graph when the program is executed
Outline

- What is GNU Radio?
- Basic Concepts
- GNU Radio Architecture & Python
- Dial Tone Example
- *Digital Communications*
- Cognitive Transmitter
Digital Communications: Transmitter

Packet Modulator → Amplifier → USRP Transmitter

Top level block of digital transmitter
Digital Transmitter

• **Building the Radio**
  • Modulating the data
  • Setting up the USRP

• **Running the Transmitter**
  • The main function
  • Starting the program
Digital Communications: Receiver

Top level block of digital receiver
Digital Receiver

• **Building the Radio**
  • Demodulating the data
  • Setting up the USRP
  • Packet callback function (different from the transmitter)

• **Running the Receiver**
  • The main function
  • Starting the program
Outline

- What is GNU Radio?
- Basic Concepts
- GNU Radio Architecture & Python
- Dial Tone Example
- Digital Communications
- Cognitive Transmitter
Cognitive Transmitter

Top level block of cognitive transmitter
Cognitive Transmitter

• **Building the Radio**
  • Modulating the data
  • Setting up the USRP

• **Running the Transmitter**
  • *Timer control of flow graphs*
  • The main function
  • Starting the program
Cognitive Transmitter: Timer Control

Time division of cognitive transmitter
Chapter 17 Summary

This chapter introduces the reader to GNU radio for cognitive radio experimentation. The Introduction section provides an overview of software defined radio and the GNU Radio toolkit. The hardware commonly used with this software, the Universal Software Radio Peripheral (USRP), is also introduced.

The analog communication section is focused on the Python level, introducing some Python basics and how Python is used in GNU Radio to connect signal processing blocks and control the flow of the digital data.

The most common type of modern communication is digital. Building upon the analog communication section, a digital radio is developed to transmit a randomly generated bitstream modulated using DBPSK.

In the last section, a time division protocol is implemented to enable channel noise calculations by the transmitter. This time division is similar to the one specified by the 802.22 standard and outlines how cognitive engines can be added to GR.
## Appendix 1: GNU Radio modules

**from gnuradio import MODULENAME**

<table>
<thead>
<tr>
<th>MODULENAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr</td>
<td>The main GNU Radio library. You will nearly always need this.</td>
</tr>
<tr>
<td>usrp</td>
<td>USRP sources and sinks and controls.</td>
</tr>
<tr>
<td>audio</td>
<td>Soundcard controls (sources, sinks). You can use this to send or receive audio to the sound cards, but you can also use your sound card as a narrow band receiver with an external RF frontend.</td>
</tr>
<tr>
<td>blks2</td>
<td>This module contains additional blocks written in Python which include often-used tasks like modulators and demodulators, some extra filter code, resamplers, squelch and so on.</td>
</tr>
<tr>
<td>eng_notation</td>
<td>Adds some functions to deals with numbers in engineering notation such as `100M' for 100 * 10^6'.</td>
</tr>
<tr>
<td>eng_options</td>
<td>Use <code>from gnuradio.eng_options import eng_options</code> to import this feature. This module extends Python's <code>optparse</code> module to understand engineering notation (see above).</td>
</tr>
<tr>
<td>gru</td>
<td>Miscellaneous utilities, mathematical and others.</td>
</tr>
</tbody>
</table>
## Appendix 2: GNU Radio scheduler

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>run()</code></td>
<td>The simplest way to run a flow graph. Calls <code>start()</code>, then <code>wait()</code>. Used to run a flow graph that will stop on its own, or to run a flow graph indefinitely until SIGINT is received.</td>
</tr>
<tr>
<td><code>start()</code></td>
<td>Start the contained flow graph. Returns to the caller once the threads are created.</td>
</tr>
<tr>
<td><code>stop()</code></td>
<td>Stop the running flow graph. Notifies each thread created by the scheduler to shutdown, then returns to caller.</td>
</tr>
<tr>
<td><code>wait()</code></td>
<td>Wait for a flow graph to complete. Flowgraphs complete when either (1) all blocks indicate that they are done, or (2) after <code>stop()</code> has been called to request shutdown.</td>
</tr>
<tr>
<td><code>lock()</code></td>
<td>Lock a flow graph in preparation for reconfiguration.</td>
</tr>
<tr>
<td><code>unlock()</code></td>
<td>Unlock a flow graph in preparation for reconfiguration. When an equal number of calls to <code>lock()</code> and <code>unlock()</code> have occurred, the flow graph will be restarted automatically.</td>
</tr>
</tbody>
</table>
Appendix 3: Useful Links

- Homepage (download, more links, etc)
  - http://gnuradio.org/trac/

- More comprehensive tutorial
  - http://gnuradio.org/trac/wiki/Tutorials/WritePythonApplications

- Available Signal Processing Blocks

- GNU Radio Mailing List Archives

- CGRAN: 3rd Party GNU Radio Apps
  - https://www.cgran.org/

- OFDM Implementation Presentation
  - http://gnuradio.org/trac/wiki/Wireless

"Cognitive Radio Communications and Networks: Principles and Practice"