GNUradio Python Programming

KC Huang

Outlines

- Python Introduction
- Understanding & Using GNU Radio
  - What is GNU Radio architecture?
  - How to program in GNUradio? (Python and C++)
- An example: dial-tone
- Useful Resources

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

Python – function & class (1)

- Function definitions have the following basic structure:

```
def func(args):
    return values
```

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

```
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

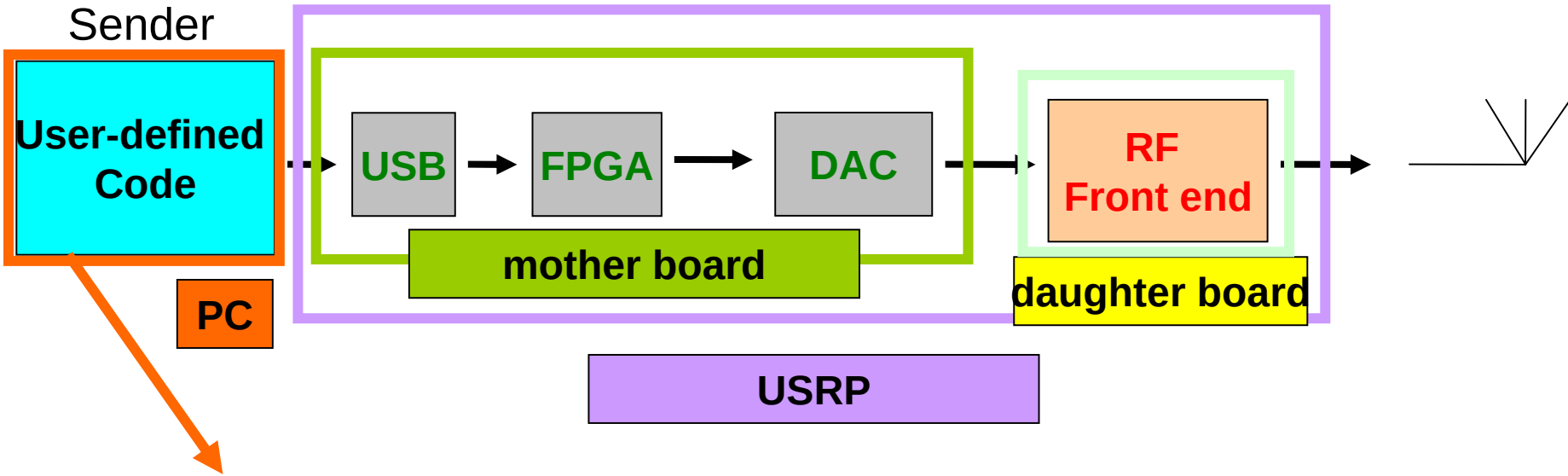
  - Initial state: for instantiation and parameter pass

```
def __init__(self):
    <statement-1>
```

Python – function & class (3)

```
class cs_mac(object):
    def __init__(self, tun_fd, verbose=False):
        self.tun_fd = tun_fd
        self.verbose = verbose
        self.fg = None
    def main_loop(self):
          min_delay = 0.001 # seconds
        while 1:
           payload = os.read(self.tun_fd, 10*1024)
          if not payload:
             self.fg.send_pkt(eof=True)
            break
          if self.verbose:
             print "Tx: len(payload) = %4d" % (len(payload),)
             delay = min_delay
             while self.fg.carrier_sensed():
                    sys.stderr.write('B')
                    time.sleep(delay)
                    if delay < 0.050:
                        delay = delay * 2 # exponential back-off
            self.fg.send_pkt(payload)
```

GNUradio Architecture

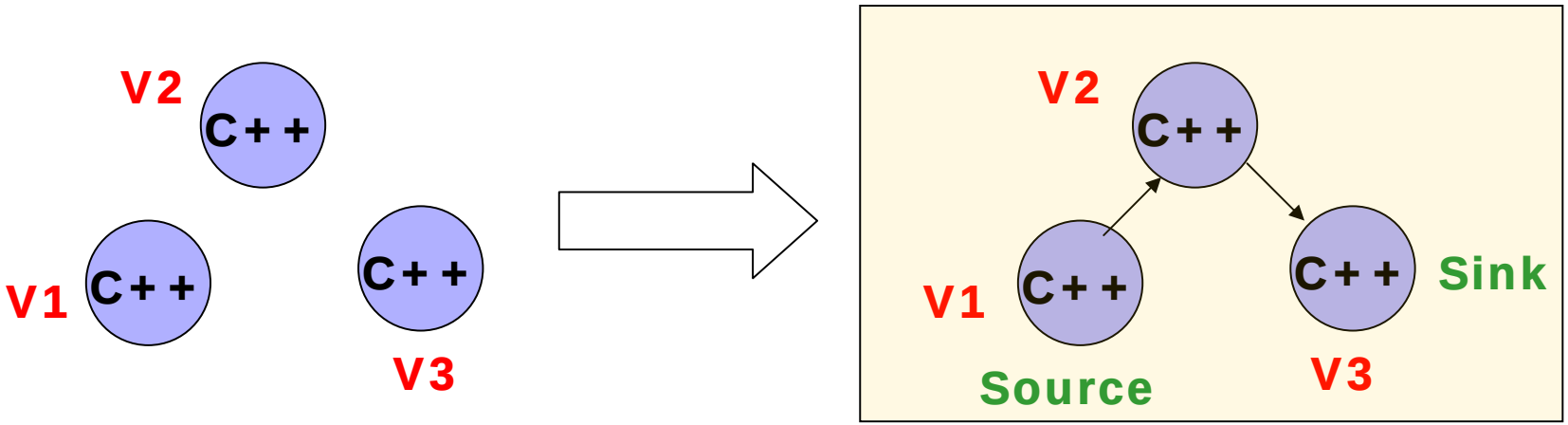


- GNU radio has provided some useful APIs
- What we are interested in at this time is how to use the existing modules that has been provided in GNU radio project to communicate between two end systems

GNUradio 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
- Signal Processing Block
  - Source: No input
    - **noise_source, signal_source, usrp_source**
  - Sink: No outputs
    - **audio_alsa_sink,usrp_sink**
  - Processing blocks: one or more inputs/outputs

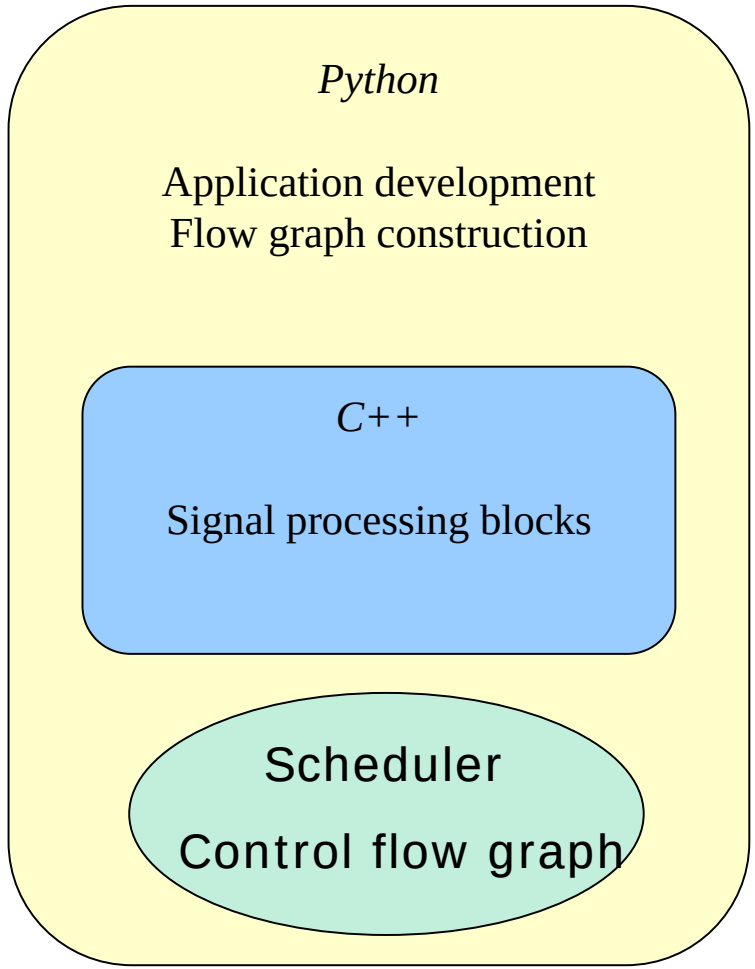
GNUradio Architecture – software(2)



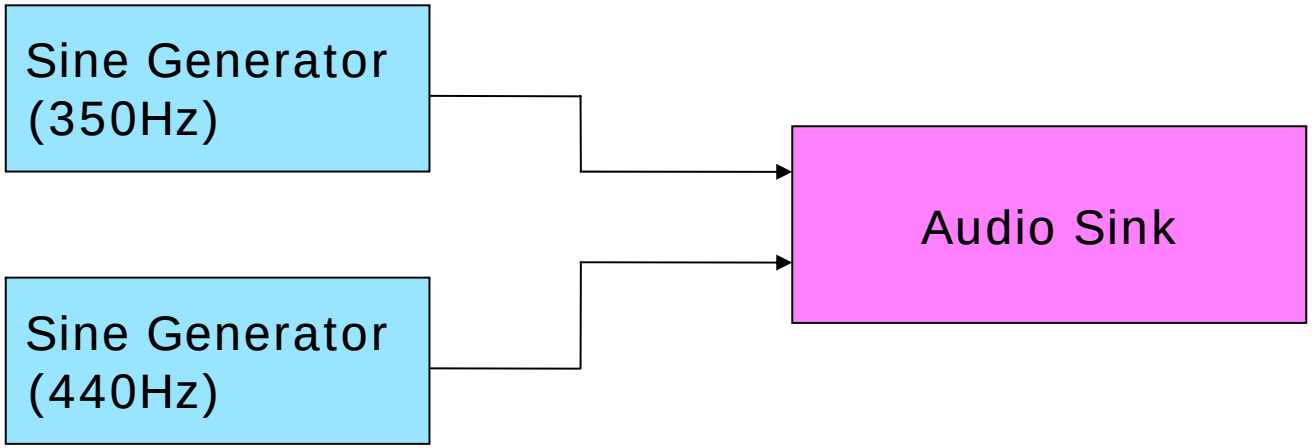
At python level, what we need to do is always just to **draw a diagram** showing **the signal flow from the source to the sink** in our mind.

GNUradio Architecture – software(3)

- 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.



Dial Tone Example (1)




```
#!/usr/bin/env python
from gnuradio import gr
from gnuradio import audio
from gnuradio.eng_option import eng_option
from optparse import OptionParser
```

It tells the shell that this file is a Python file and to use the Python interpreter to run this file.

Import modules from GNU Radio library

1. The import command is similar to the #include directive in C/C++.
2. gr is the basic GNU Radio module

Dial Tone Example (2)

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

- The class called my_top_block is derived from another class, gr.top_block
- gr.top_block class maintains the graph and also provides all the functions to build and connect blocks

- Instantiating a flow graph object and the parent constructor is called

Dial Tone Example (3)

```
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
```

Python dial_tone.py –r 50000

Define and parse command-line options

Dial Tone Example (4)

```
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))
```

- **Setting up sinewaves at 350 and 440 Hz and sampling rate by command-line**
- **Create signal generating blocks**
- **defining destination**
- **connecting source and destinations, left and right channel (it specifically connects src0 to port 0 of dst)**

Dial Tone Example (5)

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

Run the flow graph when the program is executed

GNUradio modules

- **from** gnuradio **import** MODULENAME

gr	The main GNU Radio library. You will nearly always need this.
usrp	USRP sources and sinks and controls.
audio	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.
blks2	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.
eng_notation	Adds some functions to deals with numbers in engineering notation such as `100M' for 100 * 10^6'.
eng_options	Use from gnuradio.eng_options import eng_options to import this feature. This module extends Pythons optparse module to understand engineering notation (see above).
gru	Miscellaneous utilities, mathematical and others.

GNUradio scheduler

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

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)

Useful Resource

- GNUradio:
  - Homepage (download, more links, etc)
    - http://gnuradio.org/trac/
  - More comprehensive tutorial
    - http://gnuradio.org/trac/wiki/Tutorials/WritePythonApplications
  - Available Signal Processing Blocks
    - http://gnuradio.org/doc/doxygen/hierarchy.html
  - GNU Radio Mailing List Archives
    - http://www.gnu.org/software/gnuradio/mailinglists.html
  - CGRAN: 3rd Party GNU Radio Apps
    - https://www.cgran.org/

- Python:
  - http://docs.python.org - online version of built-in Python function documentation
  - http://laurent.pointal.org/python/pqrc - Python Quick Reference Card
  - http://rgruet.free.fr - long version of Python Quick Reference Card
  - http://mail.python.org - extensive Python forum