GNU Radio

Code Walkthrough for benchmark_rx.py

Mohd Adib Sarijari
Telematics Research Group, UTM

benchmark_rx.py overview

- Code example to received data.
- Passed to user MAC layer packet and above. PHY Layer had being processes including the synchronization (using preamble and start delimiter).
- The status of CRC32 also passed to user to indicate any error in the received packet.

benchmark_rx.py code

```
from gnuradio import gr, gru, modulation_utils
from gnuradio import usrp
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

import random
import struct
import sys

# from current dir
from receive_path import receive_path

#import os
#print os.getpid()
#raw_input('Attach and press enter: ')

class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()

    # Create Options Parser:
    parser = OptionParser (option_class=eng_option,
conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")

    parser.add_option("-m", "--modulation", type="choice",
choices=demods.keys(),
                      default='gmsk',
                      help="Select modulation from: %s [default=%default]"
                           % (', '.join(demods.keys()),))

    receive_path.add_options(parser, expert_grp)

    for mod in demods.values():
        mod.add_options(expert_grp)

    (options, args) = parser.parse_args ()

    if len(args) != 0:
        parser.print_help(sys.stderr)
        sys.exit(1)

    if options.rx_freq is None:
        sys.stderr.write("You must specify -f FREQ or --freq FREQ\n")
        parser.print_help(sys.stderr)
        sys.exit(1)


    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

benchmark_rx.py code walkthrough

```
#!/usr/bin/env python
#
# Copyright 2005,2006,2007,2009 Free Softwar
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#
```

Makes the python code executable by executing
$ chmod+x ‘filename.py"’

benchmark_rx.py code walkthrough

```
#!/usr/bin/env python
#
# Copyright 2005,2006,2007,2009 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#
```

GNU Public Licence (in comment)

benchmark_rx.py code walkthrough

```
from gnuradio import gr, gru, modulation_utils
from gnuradio import usrp
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

import random
import struct
import sys

# from current dir
from receive_path import receive_path

#import os
#print os.getpid()
#raw_input('Attach and press enter: ')
```

Importing Necessary Modules, modules from GNU library

benchmark_rx.py code walkthrough

```
from gnuradio import gr, gru, modulation_utils
from gnuradio import usrp
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

import random
import struct
import sys

# from current dir
from receive_path import receive_path

#import os
#print os.getpid()
#raw_input('Attach and press enter: ')
```

Importing Necessary Modules, modules from Python

benchmark_rx.py code walkthrough

```
from gnuradio import gr, gru, modulation_utils
from gnuradio import usrp
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

import random
import struct
import sys

# from current dir
from receive_path import receive_path

#import os
#print os.getpid()
#raw_input('Attach and press enter: ')
```

Importing Necessary Modules, modules from Current Directory

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callb
        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

Top block class for constructing the receiver flowgraph

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

Global variables declaration

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

Main Function

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

Global variables declaration (for function to be able to modify the global variable value)

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

Global variables initialization

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)

    demods = modulation_utils.type_1_demods()
```

- Received packet function implementation (local function inside main)
- This function is a call back function which will be called when there is data received.

benchmark_rx.py code walkthrough

```
    demods = modulation_utils.type_1_demods()

    # Create Options Parser:
    parser = OptionParser (option_class=eng_option, conflict_
    expert_grp = parser.add_option_group("Expert")

    parser.add_option("-m", "--modulation", type="choice", ch
(),
                      default='gmsk',
                      help="Select modulation from: %s [default=%%default]"
                            % (', '.join(demods.keys()),))

    receive_path.add_options(parser, expert_grp)

    for mod in demods.values():
        mod.add_options(expert_grp)

    (options, args) = parser.parse_args ()

    if len(args) != 0:
        parser.print_help(sys.stderr)
        sys.exit(1)

    if options.rx_freq is None:
        sys.stderr.write("You must specify -f FREQ or --freq FREQ\n")
        parser.print_help(sys.stderr)
        sys.exit(1)


    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)
```

Demodulation initialization (pointer passing for modulation data buffer)

benchmark_rx.py code walkthrough

```
    demods = modulation_utils.type_1_demods()

    # Create Options Parser:
    parser = OptionParser (option_class=eng_option, conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")

    parser.add_option("-m", "--modulation", type="choice", c
(),
                      default='gmsk',
                      help="Select modulation from: %s [d
                            % (', '.join(demods.keys()),))

    receive_path.add_options(parser, expert_grp)

    for mod in demods.values():
        mod.add_options(expert_grp)

    (options, args) = parser.parse_args ()

    if len(args) != 0:
        parser.print_help(sys.stderr)
        sys.exit(1)

    if options.rx_freq is None:
        sys.stderr.write("You must specify -f FREQ or --freq
        parser.print_help(sys.stderr)
        sys.exit(1)


    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)
```

- Input setting from user for receiver.
- Setting by user is performed during python file execution.
- This is using OptionParser class (class provided by python - /usr/lib/python2.5/optpasrse.py).
- Note that this is not only the option available for user, there is other option which is declared in the receive_path file.

benchmark_rx.py code walkthrough

```
    demods = modulation_utils.type_1_demods()

    # Create Options Parser:
    parser = OptionParser (option_class=eng_option, conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")

    parser.add_option("-m", "--modulation", type="choice", choices=demods.keys
(),
                      default='gmsk',
                      help="Select modulation from: %s [defaul
                            % (', '.join(demods.keys()),))

    receive_path.add_options(parser, expert_grp)

    for mod in demods.values():
        mod.add_options(expert_grp)

    (options, args) = parser.parse_args ()

    if len(args) != 0:
        parser.print_help(sys.stderr)
        sys.exit(1)

    if options.rx_freq is None:
        sys.stderr.write("You must specify -f FREQ or --freq F
        parser.print_help(sys.stderr)
        sys.exit(1)


    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)
```

- Include the option selected by user for receiver in the receive_path file.
- This is where the others receiver user setting are located.
- Go through the receive_path code for details.

benchmark_rx.py code walkthrough

```
    demods = modulation_utils.type_1_demods()

    # Create Options Parser:
    parser = OptionParser (option_class=eng_option, conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")

    parser.add_option("-m", "--modulation", type="choice", choices=demods.keys
(),
                      default='gmsk',
                      help="Select modulation from: %s [default=%%default]"
                            % (', '.join(demods.keys()),))

    receive_path.add_options(parser, expert_grp)

    for mod in demods.values():
        mod.add_options(expert_grp)

    (options, args) = parser.parse_args ()

    if len(args) != 0:
        parser.print_help(sys.stderr)
        sys.exit(1)

    if options.rx_freq is None:
        sys.stderr.write("You must specify -f FREQ or --freq FREQ\n")
        parser.print_help(sys.stderr)
        sys.exit(1)


    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)
```

Pass options selected to all demodulator file (dqpsk, d8psk, cpm, qam8, dbpsk and gmsk)

benchmark_rx.py code walkthrough

```
    demods = modulation_utils.type_1_demods()

    # Create Options Parser:
    parser = OptionParser (option_class=eng_option, conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")

    parser.add_option("-m", "--modulation", type="choice", choices=demods.keys
(),
                      default='gmsk',
                      help="Select modulation from: %s [default=%%default]"
                            % (', '.join(demods.keys()),))

    receive_path.add_options(parser, expert_grp)

    for mod in demods.values():
        mod.add_options(expert_grp)

    (options, args) = parser.parse_args ()

    if len(args) != 0:
        parser.print_help(sys.stderr)
        sys.exit(1)

    if options.rx_freq is None:
        sys.stderr.write("You must specify -f FREQ or --freq F
        parser.print_help(sys.stderr)
        sys.exit(1)


    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback,
```

- Parse the command line option found in 'args'.
- Check for error in the parsing process.
- Return two value:
  - Value -> option (passing all the option value)
  - args -> args (report on any error)

benchmark_rx.py code walkthrough

```
    demods = modulation_utils.type_1_demods()

    # Create Options Parser:
    parser = OptionParser (option_class=eng_option, conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")

    parser.add_option("-m", "--modulation", type="choice", choices=demods.keys
(),
                      default='gmsk',
                      help="Select modulation from: %s [default=%%default]"
                            % (', '.join(demods.keys()),))

    receive_path.add_options(parser, expert_grp)

    for mod in demods.values():
        mod.add_options(expert_grp)

    (options, args) = parser.parse_args ()

    if len(args) != 0:
        parser.print_help(sys.stderr)
        sys.exit(1)

    if options.rx_freq is None:
        sys.stderr.write("You must specify -f FREQ or --freq FR
        parser.print_help(sys.stderr)
        sys.exit(1)


    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)
```

- Reporting error if any.
- Exit the program is error occur.

benchmark_rx.py code walkthrough

```
    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

- Constructing the receive flowgraph and pass the pointer to a variable named tb.
- Receive function name for callback is passed to the flowgraph using this passing.

benchmark_rx.py code walkthrough

```
# build the graph
tb = my_top_block(demods[options.modulation], rx_callback, options)

r.enable_realtime_scheduling()
gr.RT_OK:
```

• Constructing the r

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callbac
        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////
#                                   main
# ////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0
```

Receive parameter from the called command:

- demods[options.modulation] -> demodulator
- rx_callback -> rx_callback
- options -> options

benchmark_rx.py code walkthrough

```
# build the graph
tb = my_top_block(demods[options.modulation], rx_callback, options)

r.enable_realtime_scheduling()
gr.RT_OK:
```

• Constructing the r

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback,
        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////
#                                   main
# ////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0
```

- Pass the option to receiver (receive_path file).
- Receiver is setting up using the selected setting.
- Passed the receiver pointer to self.rxpath.

benchmark_rx.py code walkthrough

```
    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r.enable_realtime_scheduling()
    if r == gr.RT_OK:
```

• Constructing the r

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, op
        self.connect(self.rxpath)

# /////////////////////////////////////////////////////////////////
#                                   main
# /////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0
```

Connect and construct the flowgraph using connect class.

benchmark_rx.py code walkthrough

```
    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

Enable the real time scheduling for flowgraph to be able to run.

benchmark_rx.py code walkthrough

```
    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

Checking error in the status of real time scheduling enabling.

benchmark_rx.py code walkthrough

```
    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

- Start the execution of the constructed flowgraph using start method.
- Flowgraph properties in include start, stop and wait.

benchmark_rx.py code walkthrough

```
    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

- This is to make the flowgraph keep running until user ask to stop.
- Note that main is only one thread, there might be another thread running in the program (in this code, the receive thread is still running which is outside main)

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# /////////////////////////////////////////////////////////////////////////////
#                                   main
# /////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_r
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

- The call back function which will be called when there is a packet received.
- This function receive two parameters:
  - ok -> the status of the CRC32 (True if no error, False if there is an error).
  - Payload -> the received data from MAC and above (the transferred data from benchmark_tx.py).

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

- Unpack the payload.
- Buffer the payload of byte 0 and 1 (2bytes) to a variable called packetno.

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# /////////////////////////////////////////////////////////////////////////////
#                                   main
# /////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

- Increase the value of n_rcvd to indicate the total number of received packet.

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# /////////////////////////////////////////////////////////////////////////////
#                                   main
# /////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_righ
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

- Check the status of the packet either it is error free or not.
- If it is error free, then increase the value of n_right to indicate the total number of correctly received packet.

benchmark_rx.py code walkthrough

```
class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)
        self.rxpath = receive_path(demodulator, rx_callback, options)
        self.connect(self.rxpath)

# /////////////////////////////////////////////////////////////////////////////
#                                   main
# /////////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()
```

print to show the value of every variable for debugging.

benchmark_rx.py code walkthrough

```
    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

- This is where python code will go at the first time during the execution (after the import and include work).
- It is like a main function in C programming.

benchmark_rx.py code walkthrough

```
    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

Main is called by this command.

benchmark_rx.py code walkthrough

```
    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

Command to capture user interrupt command (cntrl+c) to stop the code execution properly.

Q & A

Thank You for Your Attention