

# ECE 598HH: Special Topics in Wireless Networks and Mobile Systems

## Lecture 17: Software Defined Radio Tutorial Haitham Hassanieh



# Software Defined Radios

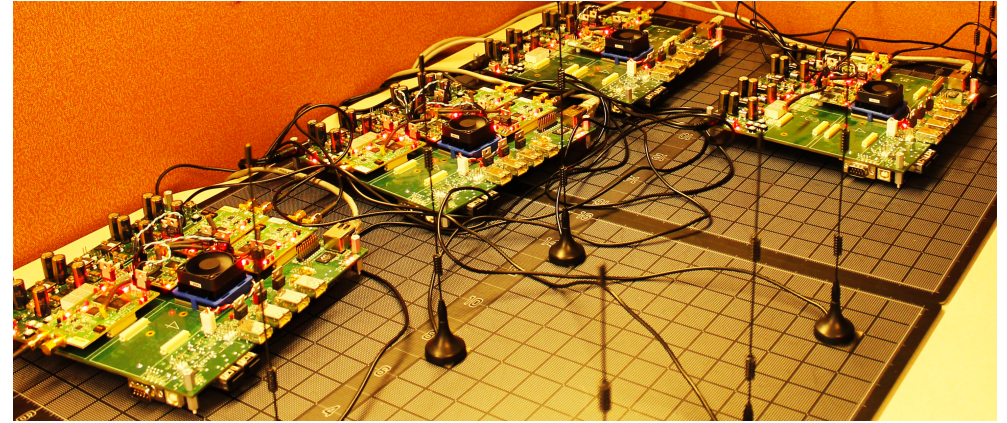
- Idea: Flexibility, Portability, Multifunction ...
  - Programmable Radio Frontend:
    - Change Center Frequency, Bandwidth, Sampling Rate, Gain, ....
  - Baseband Processing in Software:
    - Implement baseband functions on host in software (C++, Python, ...)
    - Implement some function on FPGA.

# Many Open Source SDR Platforms

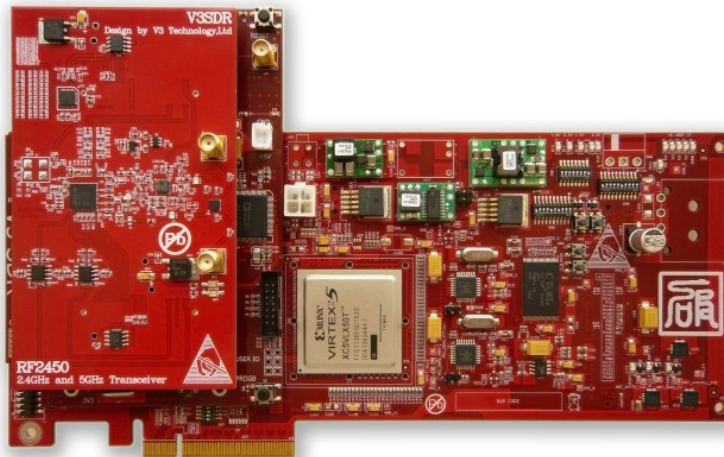
USRP: Universal Software Radio Peripheral (Ettus)



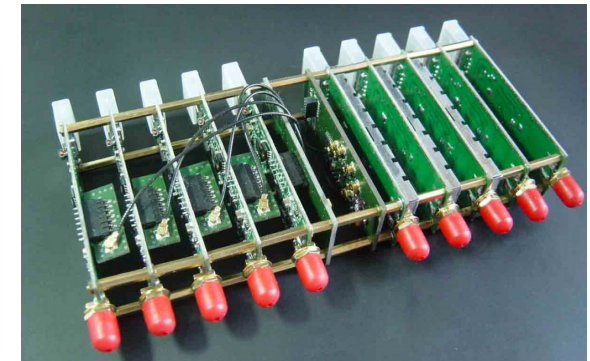
Warp: Wireless Open Access Research Platform (Rice)



SORA: Software-defined Radio (Microsoft)



RTL-SDR



# GNU-Radio USRP Platform

USRP: Universal Software Radio Peripheral (Ettus)



- What is GNU Radio?
- Basic Concepts
- Developing Applications
- UHD

- What is GNU Radio?
- Basic Concepts
- Developing Applications
- UHD

**BASEBAND  
PROCESSING**

**DIGITAL & RF  
FRONTEND**

**ANTENNA**



GNU Radio Host  
Application

Up to 25 Msps  
Complex Baseband  
Sample Stream

USB 2.0 – 8 Msps  
GbE – 25 Msps  
I/Q



Ettus Research  
USRP (FPGA-based)

Up to ~80 MHz  
Passband,  
DC - ~6GHz  
(db dependent)



External  
RF Electronics

**Baseband  
Modulation/Demodulation**

**Up/Down Conversion  
ADC/DAC  
Interpolation/Decimation**

**Low Noise Rx  
Power Amp for TX**

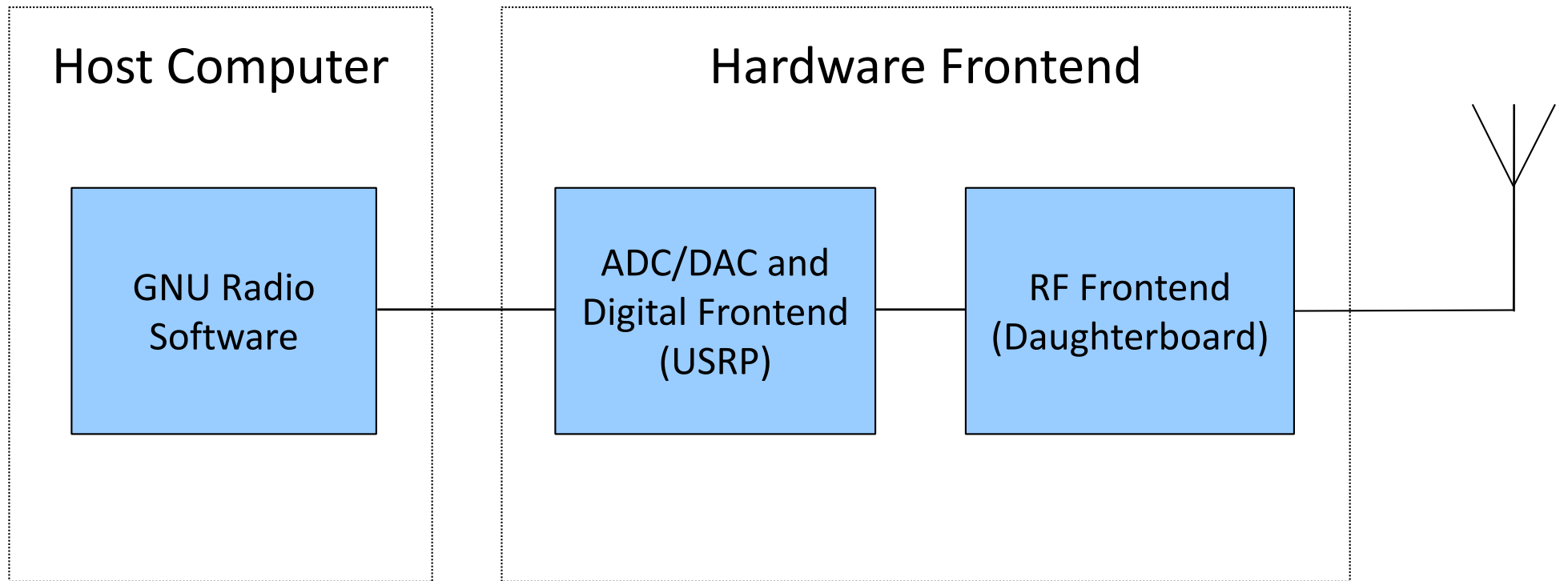
# What is GNU Radio?

- Software toolkit for signal processing
  - Providing a fast, low-cost way to test algorithms and waveform design
- USRP (Universal Software Radio Peripheral)
  - Hardware TX/RX frontend

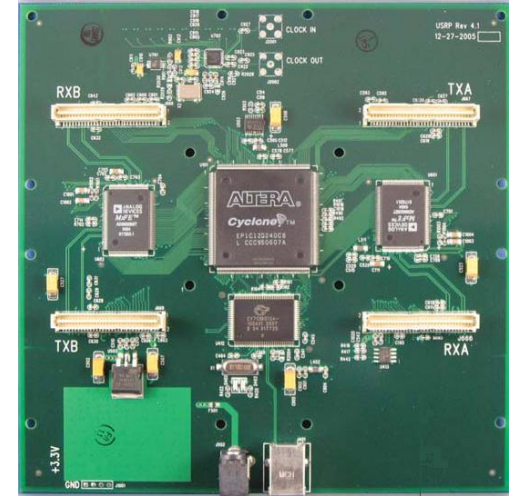




# GNU Radio Components



# USRP1

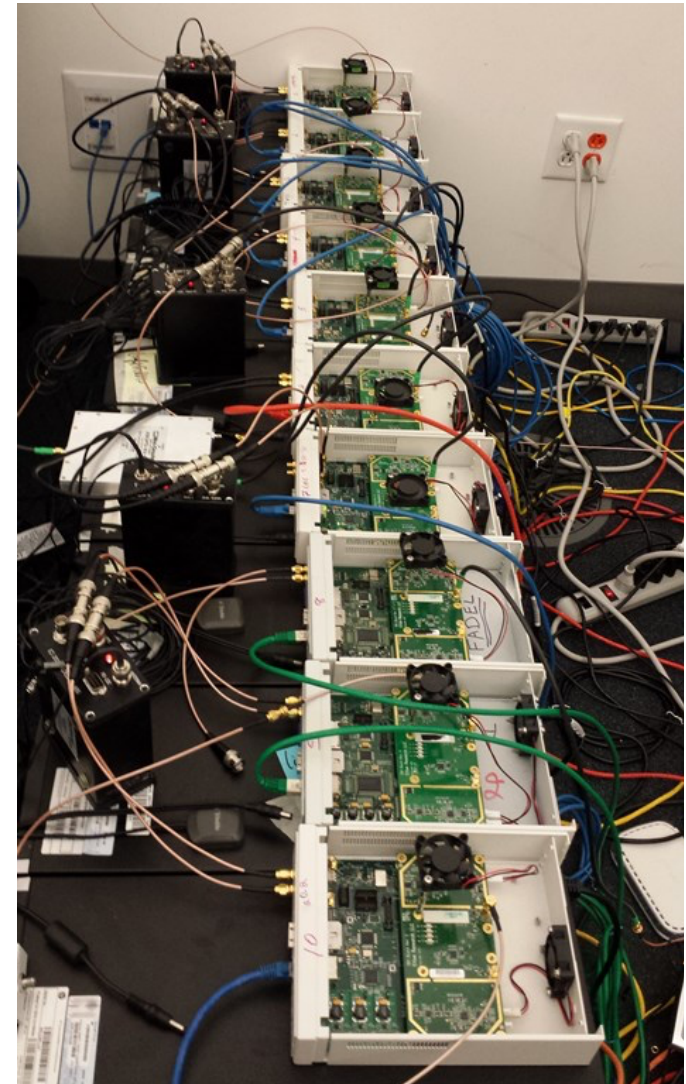


- Features:
  - Frequency Range: DC-6 GHz
  - Connectivity for Two, Complete Tx/Rx chains
  - Two Dual 64 MS/s 12-bit ADCs
  - Two Dual 128 MS/s, 14-bit DACs
  - Up to 16 MS/s USB Streaming
  - Altera Cyclone FPGA

# USRP2/N200/N210



- Features:
  - Frequency Range: DC-6 GHz
  - 100 MS/s 14-bit ADC
  - 400 MS/s, 16-bit DAC
  - Up to 25 MS/s through Gigabit Ethernet Streaming (50 MS/s at 8 bits per sample)
  - Xilinx Spartan XC3SD3400A FPGA: 2368K Memory, 53K logic cells
  - MIMO Extension/ External Clock Synchronization



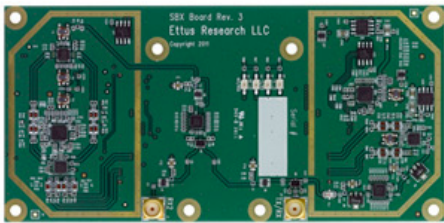
# USRP X300/310

- Features:
  - Frequency Range: DC-6 GHz
  - Two 200 MS/s 14-bit ADC
  - Two 800 MS/s, 16-bit DAC
  - Up to 200 MS/s through 10 Gigabit Ethernet or PCI Express
  - Xilinx Kintex-7 FPGA: 28620K memory, 406K logic cells
  - MIMO Extension External Clock Synchronization

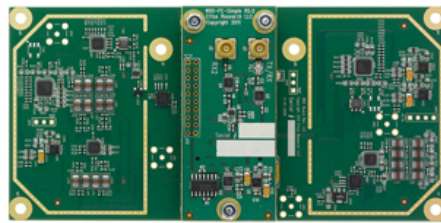


# USRP Daughter Boards

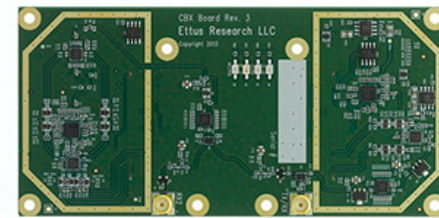
- Define Bandwidth, Frequency Range, Gain



SBX: 400MHz-4.4GHz  
(40MHz/120MHz)



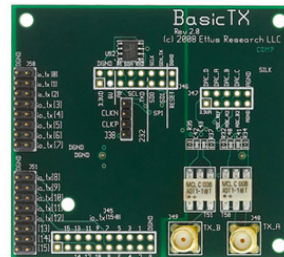
WBX: 40MHz-2.2GHz  
(40MHz/120MHz)



CBX: 1.2GHz-6GHz  
(40MHz/120MHz)



UBX: 10MHz-6GHz  
(40MHz/160MHz)



Basic TX/RX  
1MHz-250MHz



LFTX/LFRX  
0MHz-30MHz

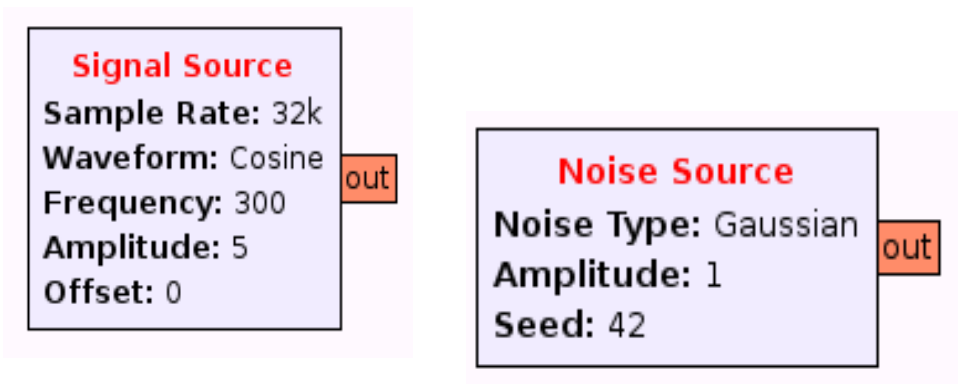
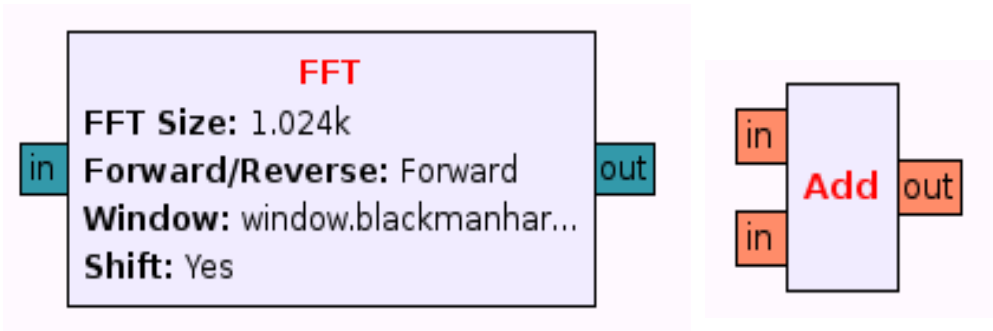
# GNU Radio Software

- Open source (GPL)
- Existing examples: 802.11b, Zigbee, OFDM, DBPSK, DQPSK ...
- Extensive library of signal processing blocks (C++)
- Modular design (flow graph)
- GNU Radio + UHD

- What is GNU Radio?
- Basic Concepts
- Developing Applications
- UHD

# Blocks

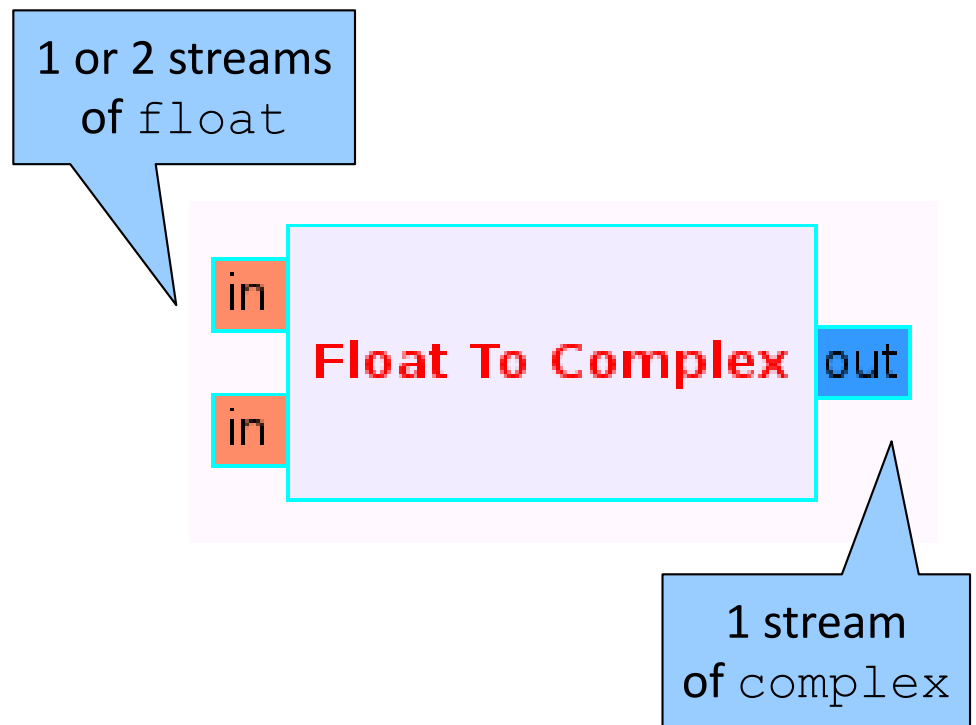
- Signal Processing Block
  - Accepts input streams
  - Produces output streams
- Source: No input
  - noise\_source,  
signal\_source,  
usrp\_source
- Sink: No outputs
  - audio\_alsa\_sink,  
usrp\_sink





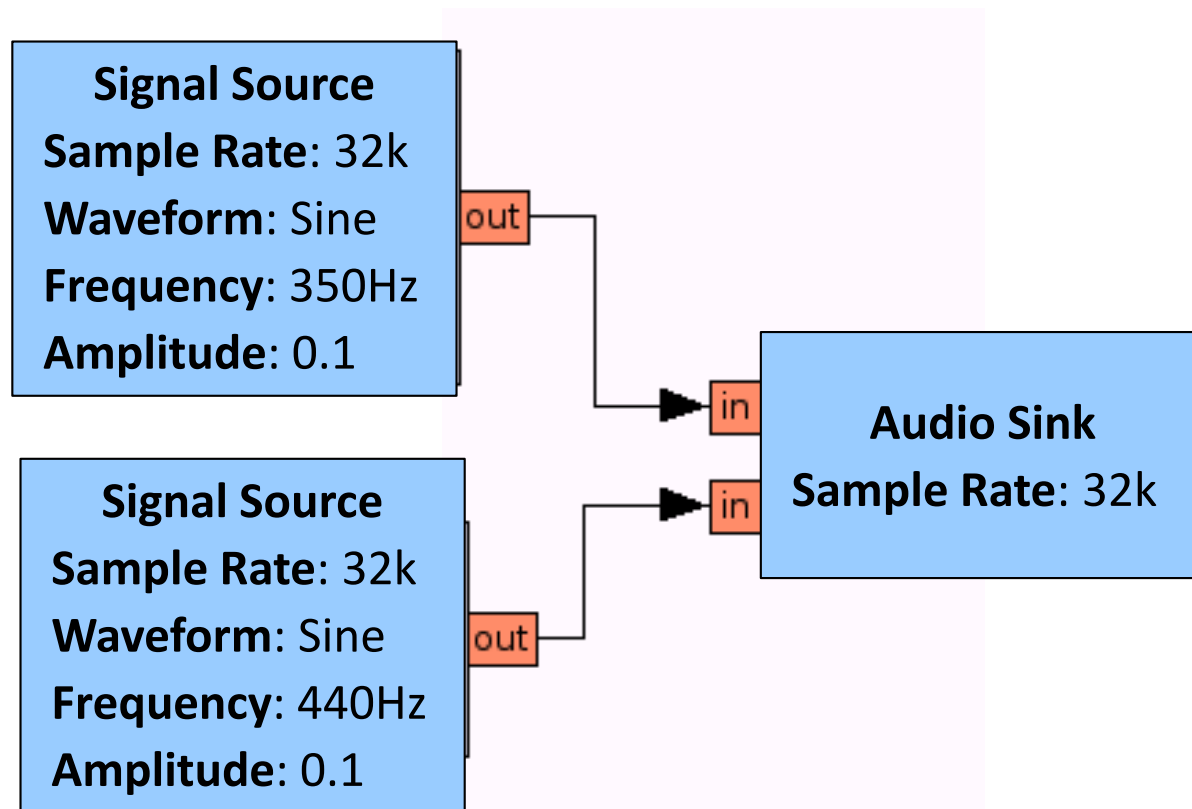
# Data Types

- Blocks operate on certain data types
  - float, complex, int, char ...
  - Vectors
- IO Signature:
  - Number of input ports
  - Number of output ports
  - Item size of each port



# Flow Graph

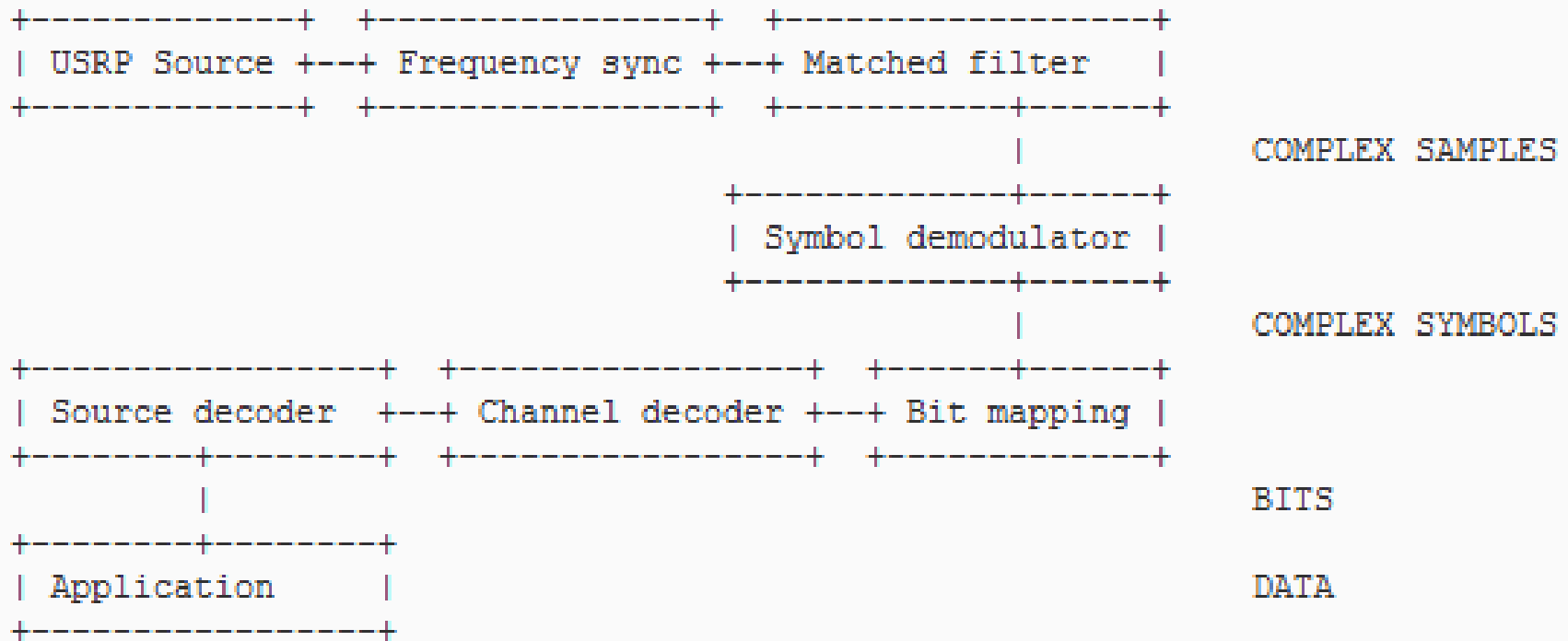
- Blocks composed as a flow graph
  - Data stream flowing from sources to sinks



# Example Code

```
1  #!/usr/bin/env python
2
3  from gnuradio import gr
4  from gnuradio import audio
5
6  class my_top_block(gr.top_block):
7      def __init__(self):
8          gr.top_block.__init__(self)
9
10         sample_rate = 32000
11         ampl = 0.1
12
13         src0 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 350, ampl)
14         src1 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 440, ampl)
15         dst = audio.sink (sample_rate, "")
16         self.connect (src0, (dst, 0))
17         self.connect (src1, (dst, 1))
18
19  if __name__ == '__main__':
20      try:
21          my_top_block().run()
22      except [[KeyboardInterrupt]]:
23          pass
```

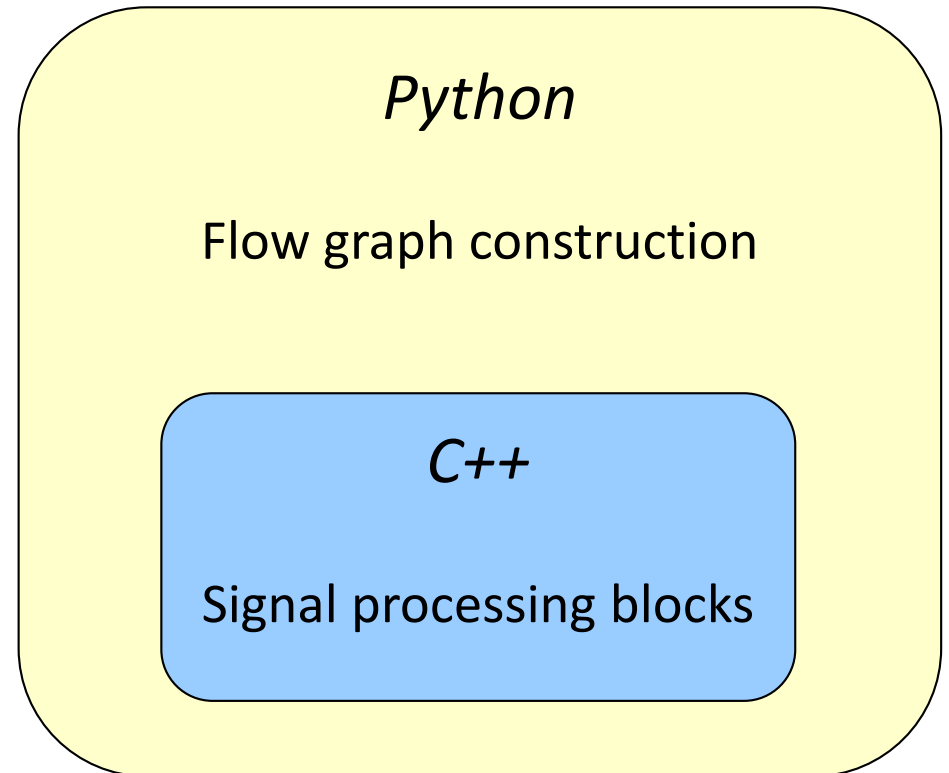
# QPSK Demodulator Example



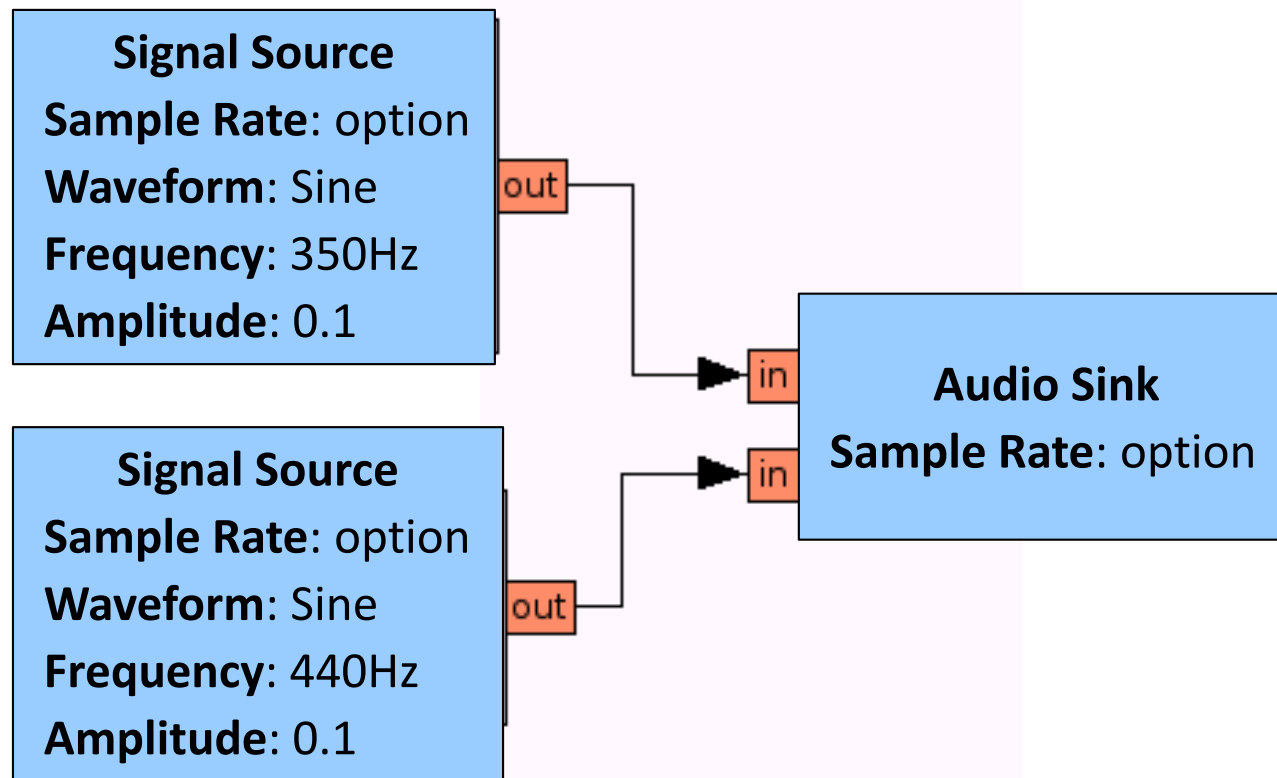
- What is GNU Radio?
- Basic Concepts
- Developing Applications
- UHD

# Development Architecture

- Python
  - Flow graph construction
- C++
  - Signal processing blocks



# Python Example



```

#!/usr/bin/env python

from gnuradio import gr
from gnuradio import audio
from gnuradio.eng_option import eng_option
from optparse import OptionParser

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

```



## Import modules from GNU Radio library

```
from gnuradio import gr
from gnuradio import audio
from gnuradio.eng_option import eng_option
from optparse import OptionParser
```

```
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

```
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
```

## 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)
self.connect (src0, (dst, 0))
self.connect (src1, (dst, 1))
```

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

Run the flow graph when the program is executed

# Creating Your Own Block

- Basics
  - A block is a C++ class
  - Typically derived from `gr_block` class
- Three components
  - `my_block_xx.h`: Block definition
  - `my_block_xx.cc`: Block implementation
  - `my_block_xx.i`: Python bindings (SWIG interface)

# Block Definition

```
#include <gr_sync_block.h>
```

```
class my_block_cc;
```

```
typedef boost::shared_ptr<6829_my_block_cc> 6829_my_block_cc_sptr;
```

```
6829_my_block_cc_sptr 6829_make_my_block_cc();
```

Create instances of block

```
class my_block_cc : public gr_sync_block
```

```
{
```

```
private:
```

```
unsigned int d_count;
```

```
friend 6829_my_block_cc_sptr 6829_make_my_block_cc();
```

```
6829_my_block_cc();
```

```
public:
```

```
int work(int noutput_items,  
         gr_vector_const_void_star &input_items,  
         gr_vector_void_star &output_items);
```

```
};
```

Method for processing  
input streams and  
producing output streams

# Block Implementation

```
#ifndef HAVE_CONFIG_H
#include "config.h"
#endif

#include <my_block_cc.h>
#include <gr_io_signature.h>

6829_my_block_cc_sptr 6829_make_my_block_cc()
{
return 6829_my_block_cc_sptr(new 6829_my_block_cc());
}

6829_hw_sample_block_cc::6829_hw_sample_block_cc()
: gr_sync_block("my_block_cc",
    gr_make_io_signature(1, 1, sizeof(gr_complex)),
    gr_make_io_signature(1, 1, sizeof(gr_complex))),
  d_count(0)
{
}
```

Define input and  
output signatures



# Block Implementation

```
int 6829_my_block_cc::work(int noutput_items,  
                           gr_vector_const_void_star &input_items,  
                           gr_vector_void_star &output_items)  
{  
    const gr_complex* in = (const gr_complex*)input_items[0];  
    gr_complex* out = (gr_complex*)output_items[0];
```

```
    memcpy(out, in, sizeof(*out) * noutput_items);
```

Copy input stream  
to output stream

```
    for (int i = 0; i < noutput_items; ++i)  
        fprintf(stderr, "%u\t<%.6f, %.6f>\n",  
                d_count++, in[i].real(), in[i].imag());
```

Echo samples  
stderr

```
    return noutput_items;  
}
```

Return number  
of items processed

- What is GNU Radio?
- Basic Concepts
- Developing Applications
- UHD

# UHD (USRP Hardware Driver)

- Why UHD?

- USRP
  - Libusrp
  - Libusrp-gnuradio
  - Python dboard code
  - C++ dboard code
  - Usrc\_\* examples and utils
- USRP2
  - Libusrp2 (linux only)
  - libusrp2-gnuradio
  - C dboard code in FW
  - Usrc2\_\* examples and utils
- USRP N+1?



- Having N drivers isn't going to scale ...

# UHD (USRP Hardware Driver)

- Single API for all USRP devices
  - C++ based API
  - All daughterboards
  - Inherent multi-channel support
    - Synchronization
    - Channel alignment
- Gnuradio–UHD Blocks
  - Source Block, Sink Block
  - Python, C++

# UHD (USRP Hardware Driver)



- Send/receive samples
  - `device->send(...)` and `device->recv(...)`
- Set/get properties

# Device Properties

- Set/get gain
- Set center frequency
- Set/get device time
- Set/get sample rate
- Antenna selection
- Frontend selection
- $\vdots$

# Example: MIMO TX

```
sudo ./tx_samples_multi_from_file_repeat  
--args="addr0=192.168.10.2,addr1=192.168.20.2"  
--file_prefix="TX"  
--rate=1000000  
--freq=922000000  
--ant="TX/RX"  
--gain=10  
--ref="external"  
--num_transmits=10  
--scale=0.5
```

# Example: MIMO TX

```
sudo ./tx_samples_multi_from_file_repeat  
--args="addr0=192.168.10.2,addr1=192.168.20.2"
```

---

**tx\_samples\_multi\_from\_file\_repeat.cpp**

```
uhd::usrp::multi_usrp::sptr usrp = uhd::usrp::multi_usrp::make(args);
```



# Example: MIMO TX

```
sudo ./tx_samples_multi_from_file_repeat
--args="addr0=192.168.10.2,addr1=192.168.20.2"
--file_prefix="TX"
--rate=1000000
--freq=922000000
--ant="TX/RX"
--gain=10
```

---

## **tx\_samples\_multi\_from\_file\_repeat.cpp**

```
usrp->set_tx_rate(rate, i);
usrp->set_tx_freq(freq, i);
usrp->set_tx_antenna(ant, i);
usrp->set_tx_gain(gain, i);
```

# Example: MIMO TX

```
sudo ./tx_samples_multi_from_file_repeat  
--args="addr0=192.168.10.2,addr1=192.168.20.2"  
--file_prefix="TX"  
--rate=1000000  
--freq=922000000  
--ant="TX/RX"  
--gain=10  
--ref="external"
```

---

**tx\_samples\_multi\_from\_file\_repeat.cpp**

```
usrp->set_clock_config(uhd::clock_config_t::external());
```

- USRP

<http://www.ettus.com/resource>

- GNU Radio

<http://gnuradio.org/redmine/projects/gnuradio/wiki>

- Available Signal Processing Blocks

<http://gnuradio.org/doc/doxygen/hierarchy.html>

- GNU Radio Mailing List Archives

<http://www.gnu.org/software/gnuradio/maillinglists.html>

- USRP Mailing List

<http://lists.ettus.com/pipermail/usrp-users> [lists.ettus.com/](http://lists.ettus.com/)

- UHD

<http://ettus-apps.sourcerepo.com/redmine/ettus/projects/uhd/wiki>