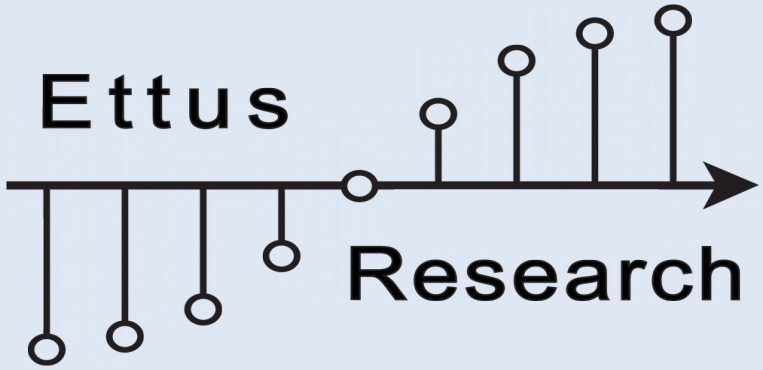
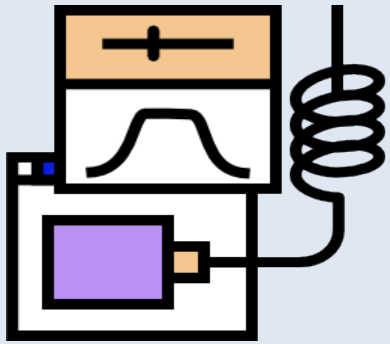
UHD - USRP Hardware Driver

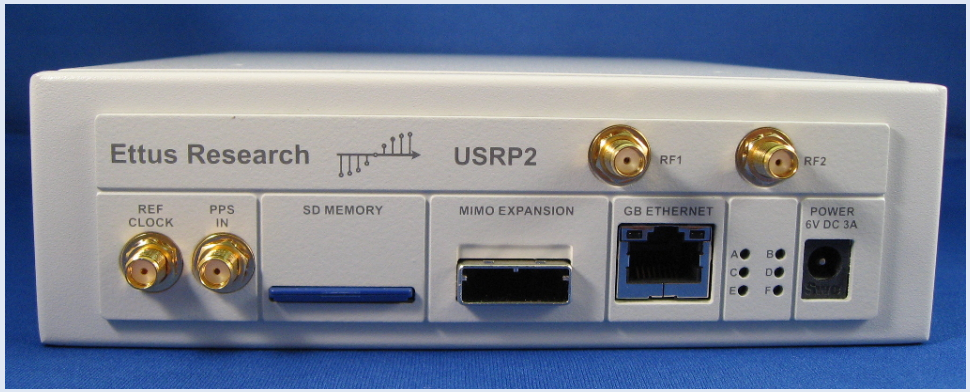
Universal Software Radio Peripheral Hardware Driver





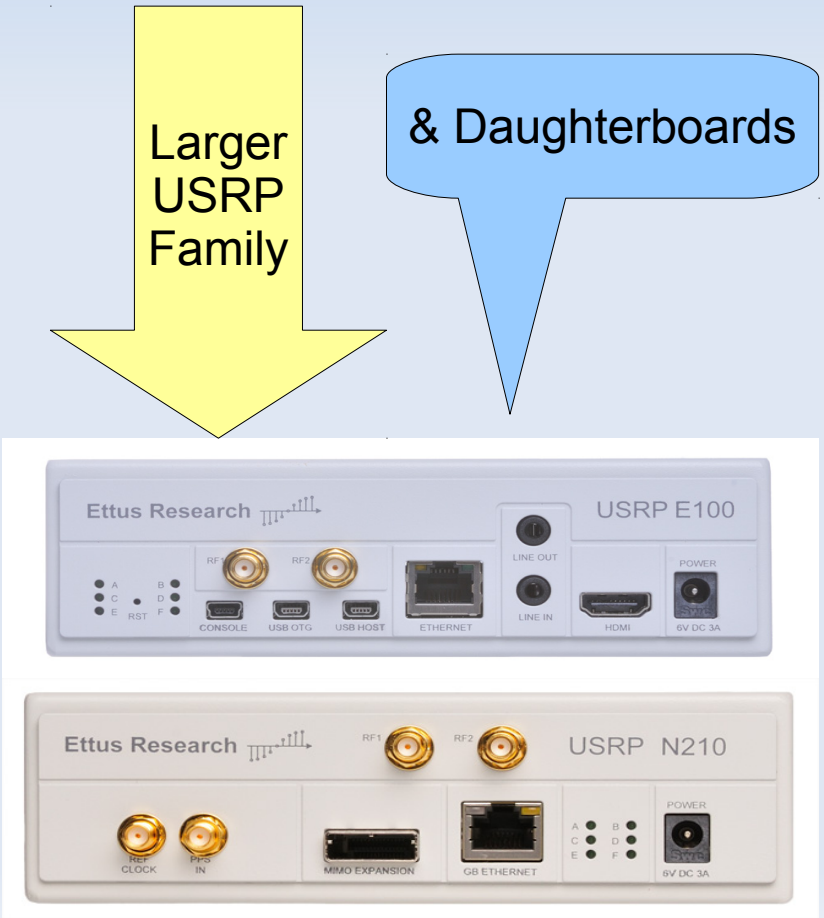
A Brief USRP Driver History

- USRP
  - Libusrp
  - Libusrp-gnuradio
  - Python dboard code
  - C++ dboard code
  - Usrp_* examples and utils
- USRP2
  - Libusrp2 (linux only)
  - libusrp2-gnuradio
  - C dboard code in FW
  - Usrp2_* examples and utils
- USRP N+1?
  - N drivers isnt going to scale...



UHD Intro

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

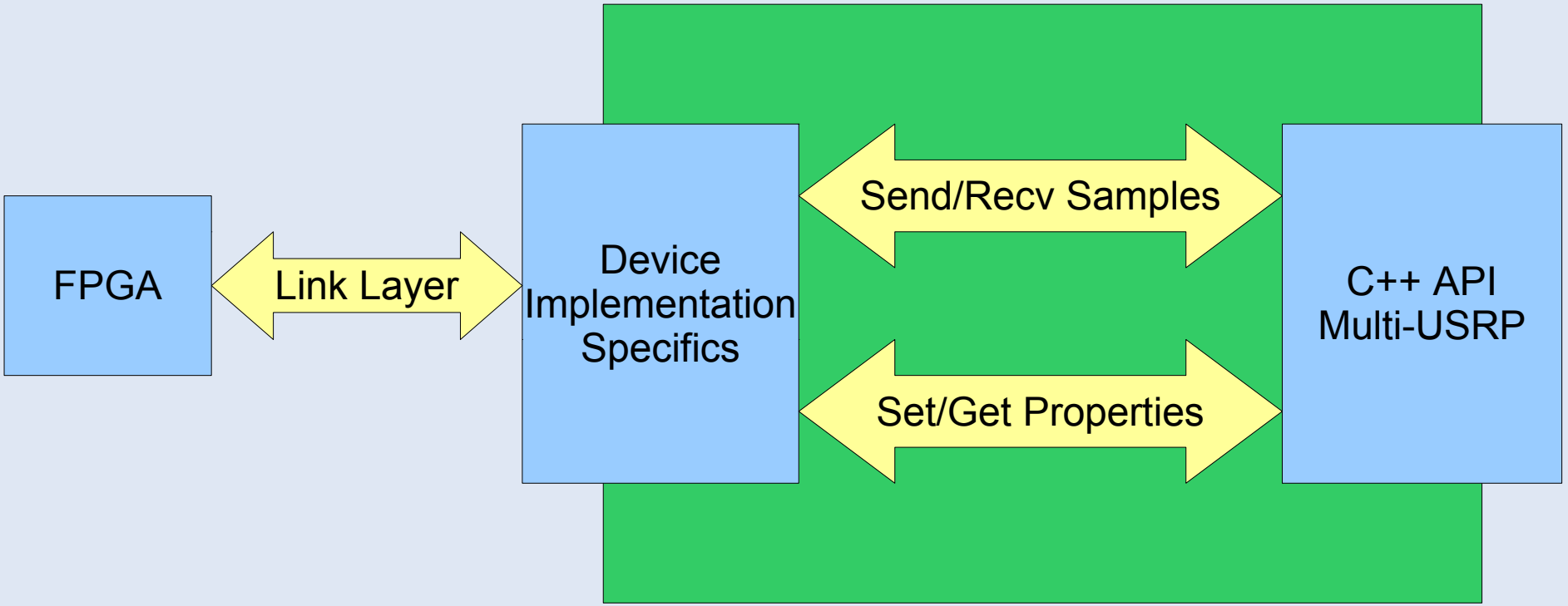


Cross Platform

- Linux, Machintosh, Windows
- Compilers
  - GCC (all OS)
  - Clang
  - MSVC
- Cmake
  - Cross platform make
  - Generates native build system
- Boost
  - Cross platform C++ awesome library
  - ASIO, Math, Unit testing, Program options

Whats in UHD?

- Find devices on system
- Instantiate device objects
  - Set/get properties
  - Send/receive samples



Device Properties

- Set/get gain
  - Overall chain or individual elements
- Set center frequency
  - Overall chain or individual elements
- Aribitrary readback w/ sensors
  - Is the RF LO locked?
- Set/get device time
- Set/get sample rate
- Antenna selection
- Frontend selection

* See doxygen or <uhd/usrp/multi_usrp.hpp> for more details *

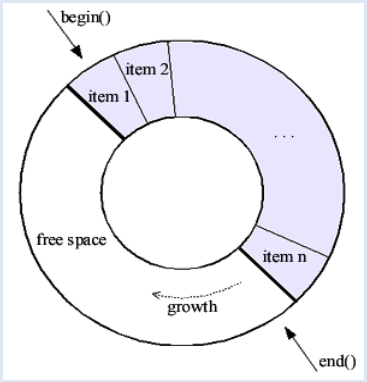
Streaming Interface

- **Streaming samples**
  - device->send(...) and device->recv(...)
  - Inherinitly multi-channel
    - Vector of pointers just like gnuradio work()
  - Metadata → aka sample decoration
    - Timestamps, Burst flags
- **Messages**
  - Inline messages for receive (recv call)
    - Overflow, stream command error
  - Async messages for transmit (recv async message call)
    - Underflow, sequence error, other...

* See doxygen or <uhd/device.hpp> for more details *

Transport Layers

- USB 2.0
  - USRP1
  - B100
- UDP/IPv4
  - USRP2
  - N2XX
- Device Node
  - E1XX

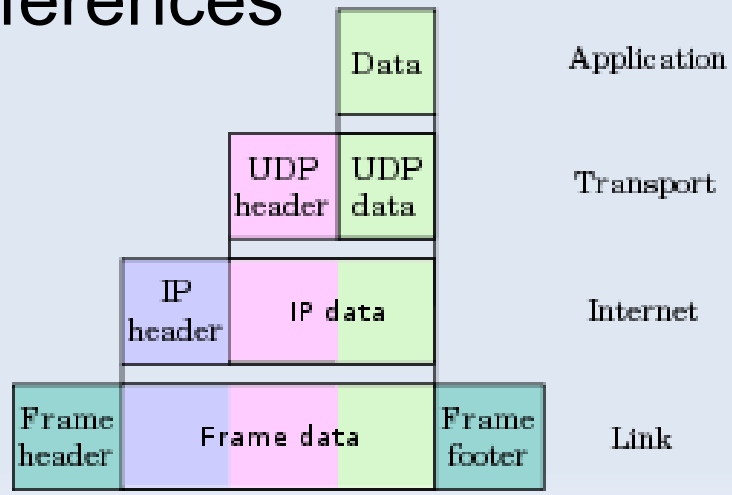


The USB 2.0 Transport

- 480Mbps theoretical, practically 256Mbps
  - 8 Msps @ 32 bits per sample
  - 16 Msps @ 16 bits per sample
- LibUSB 1.0
  - Support on all OS
  - Synchronous control transfers
  - Asyncrhonous bulk transfers
- Windows support via WinUSB
  - http://www.libusb.org/wiki/windows_backend

The UDP/IPv4 Transport

- 1 Gbps theoretical
  - 25 Msps @ 32 bits per sample
- Userspace socket implementation
  - Berkely sockets send()/recv()
  - Very portable/works everywhere
  - Boost ASIO handles platform differences

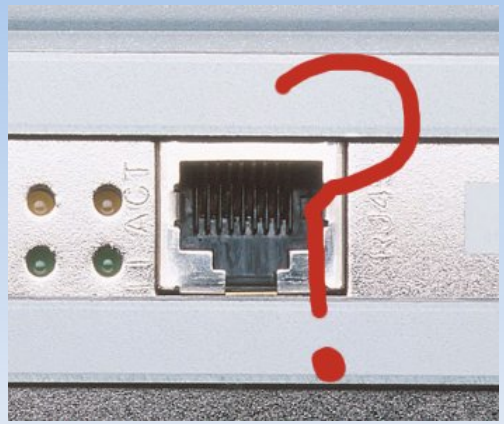


UDP Socket Tweaks

- Use massive receive socket buffer (50MB)
  - Kernel buffers receive data for you
  - Buffer size severly limited on OSX (1MB)
- Do something with the send socket buffer
  - Too big on linux, hurts performance
  - Too small on windows, hurts performance
- Latency optimization
  - Configure ”Interrupt Coalescing”
  - Use smaller packet sizes

Every OS is special

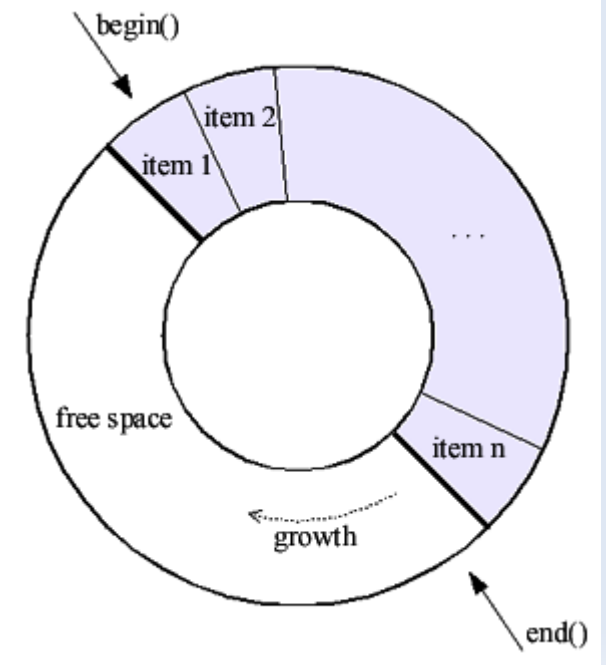
UDP Socket Tweaks cont...



- Bandwidth optimization
  - Use jumbo frames (4096 bytes)
  - Network hardware specific
- Windows transmit performance
  - registry magic: FastSendDatagramThreshold
- Crappy network hardware
  - Confused network switches
  - Bad network drivers
  - Packets > MTU size

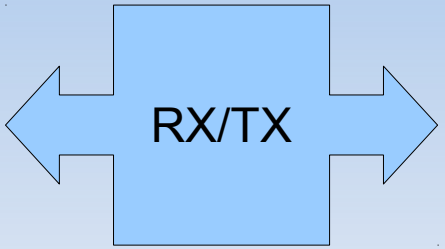
The USRP Embedded Transport

- Special kernel module and device node
  - /dev/usrp_e
  - Call ioctl() for FPGA control
  - DMA between FPGA and kernel
- Memory-mapped ring buffers
  - 1 send buffer ring
  - 1 recv buffer ring
- 8 Msps @ 32 bits per sample



Sample Framing - VITA49

- VITA49 standard for sample framing
  - Layer between samples and USB/UDP/Kernel
  - Bidirectional → frames RX and TX packets
  - Stream IDs, Timstamps, sequence count...

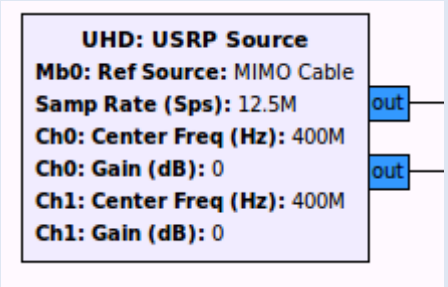


VRT / VITA49

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Header (1 Word, Mandatory)																															
Stream Identifier (1 Word, Optional)																															
Class Identifier (2 Words, Optional)																															
Integer-seconds Timestamp (1 Word, Optional)																															
Fractional-seconds Timestamp (2 Words, Optional)																															
Data Payload (Variable, Mandatory)																															
Trailer (1 Word, Optional)																															

GNU Radio + UHD

- Wrapped UHD functionality into gnuradio
  - Source and sink blocks
  - Source work() calls device->recv()
  - Metadata passed via stream tags
  - Sink work calls device->send()
- Handles multi-channel
  - Sample alignment
  - Time synchronization



GNU Radio + UHD (API)

```
#include <gr_uhd_usrp_source.h>

uhd::device_addr_t addr;
addr[”name”] = ”Lab USRP11”;

boost::shared_ptr<uhd_usrp_source> usrp = uhd_make_usrp_source(
        addr,
        uhd::io_type::COMPLEX_FLOAT32,
        1
);

usrp->set_gain(10.0);
```

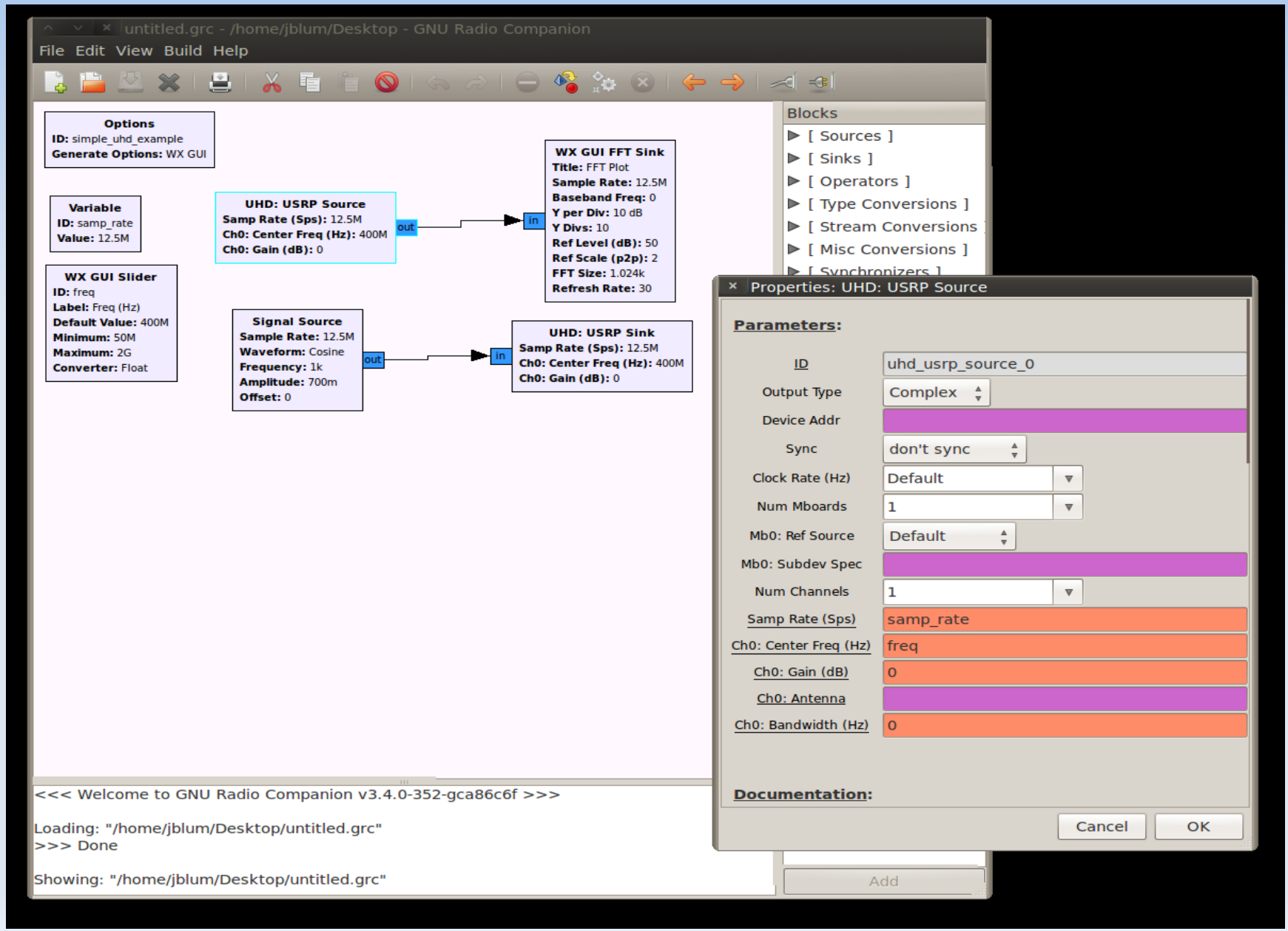
C++ API

```
from gnuradio import uhd

addr = uhd.device_addr()
addr[”name”] = ”Lab USRP11”

usrp = uhd.usrp_source(
        device_addr = addr,
        io_type = uhd.io_type.COMPLEX_FLOAT32,
        num_channels = 1,
)

usrp.set_gain(10.0)
```

- Code to the API in C++ or Python
- Data structures SWIG'd into python
- Code is basically identical

Python API

GNU Radio + UHD (GRC)



Future Features

- Support other over-the-wire types
  - 16 bit samples, 8-bit maybe too
  - A raw mode for custom FPGA stuff
- Calibration
  - Self calibration (IQ imbalance, DC offset)
  - Select full-scale power level
  - ...or transmit/receive absolute power level
- Support multi-channel, non-homogenous rates
- TX stream tags to control timed bursts

Conclusion

- USRP + UHD + GNU Radio + GRC = Awesome
- Questions? Comments?