GRUG 11: Lots of gnuradio work

- Coding guide wiki page, follow along
  - http://gnuradio.org/redmine/projects/gnuradio/wiki/BlocksCodingGuide
  - Work here: git://gnuradio.org/jblum.git
- UHD complex-int8 samples
- Building with cmake
- In-place buffer optimizations
- Message passing
- Coding blocks in python
- New components
- Volk integration

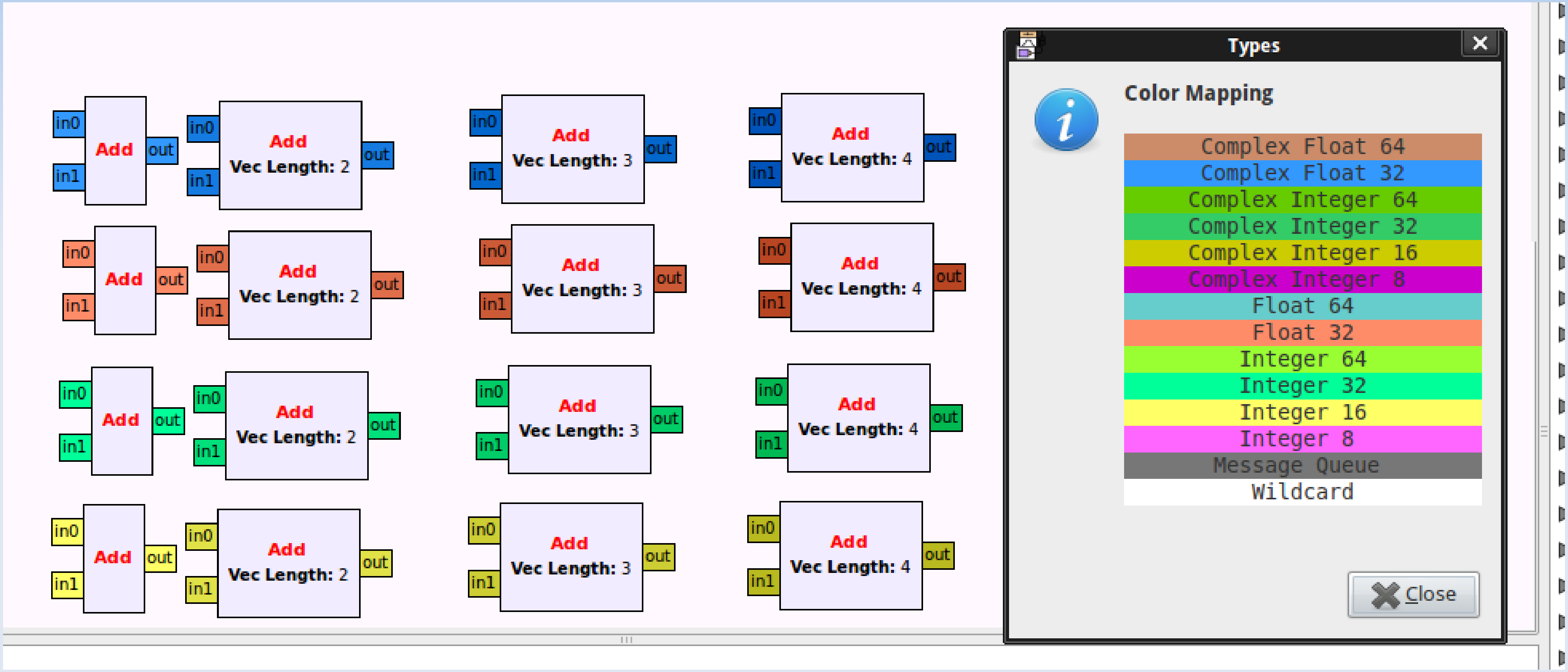
UHD complex-int8

- Doubles RX bandwidth at expense of dynamic range
- New UHD API to support alternative stream types
- Gr-uhd blocks support for new API
  - Select host data type
  - Select over-the-wire type
- GRC core changes
  - Support all basic real and complex types
  - Checks IO size not type
  - Float32 → byte w/ vlen 4

UHD complex-int8

- Doubles RX bandwidth at expense of dynamic range
- New UHD API to support alternative stream types
- Gr-uhd blocks support for new API
  - Select host data type
  - Select over-the-wire type
- GRC core changes
  - Support all basic real and complex types
  - Checks IO size not type
  - Float32 → byte w/ vlen 4

Look at the colors!



Color Mapping

Complex Float 64
Complex Float 32
Complex Integer 64
Complex Integer 32
Complex Integer 16
Complex Integer 8
Float 64
Float 32
Integer 64
Integer 32
Integer 16
Integer 8
Message Queue
Wildcard

UHD calibration stuff

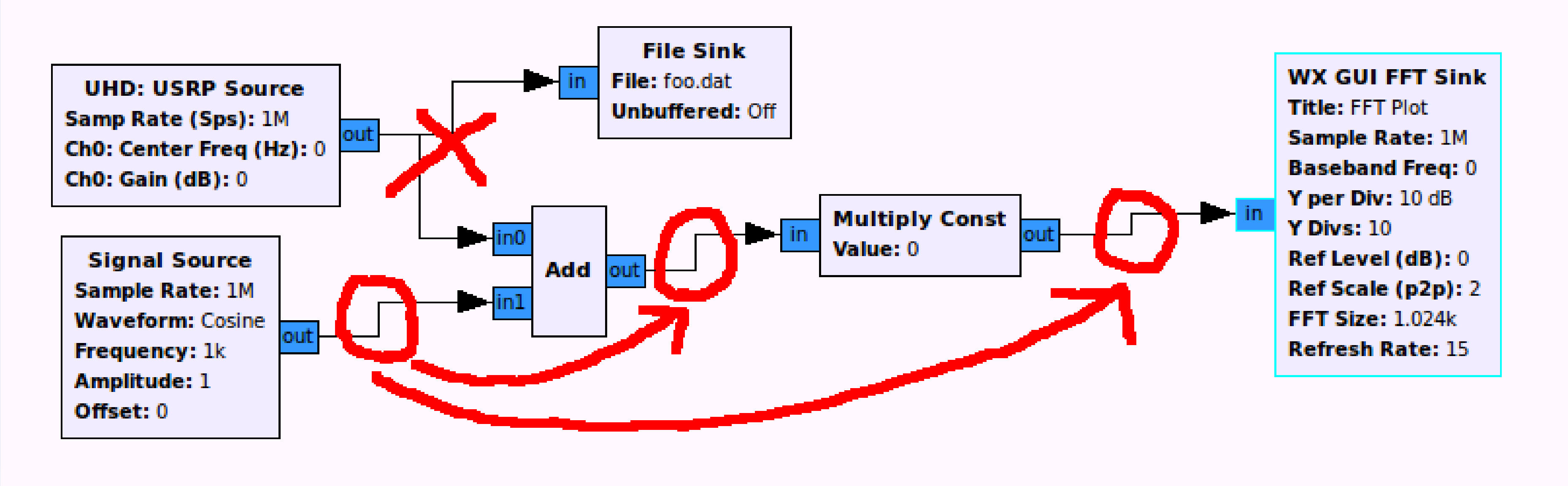
- API for dc offset correction and iq imbalance
- Available in raw uhd API and gr-uhd python/c++
- Self-calibration utils for SBX and WBX
  - Sweeps LO across frequency
  - Drags IQ imbalance and TX DC into noise
  - Saves calibration table, auto loaded at runtime
- Find basic usage documentation here:
  - http://files.ettus.com/uhd_docs/manual/html/calibration.html

Building gnuradio w/ cmake

- Build gnuradio with cmake
  - Easier to express complex build rules
  - Builds 100% out of tree
  - No checked-in generated file
  - Compilers/generators msvc, gcc
  - Builds packages (debs, rpms, exes)
  - More work for multi-deb, multi-rpm
- Instructions:
  - http://gnuradio.org/redmine/projects/gnuradio/wiki/CMakeWork

In-place buffer optimizations

- Share gr-buffers (input memory = output memory)
  - take advantage of caching
  - Save precious memory bandwidth
- Share gr-buffers when certain rules apply
  - Matching io size
  - Fixed rate (sync block)
  - Buffer has only one reader
  - User set this->set_inplace(true, inport_index)



PMT Extentions

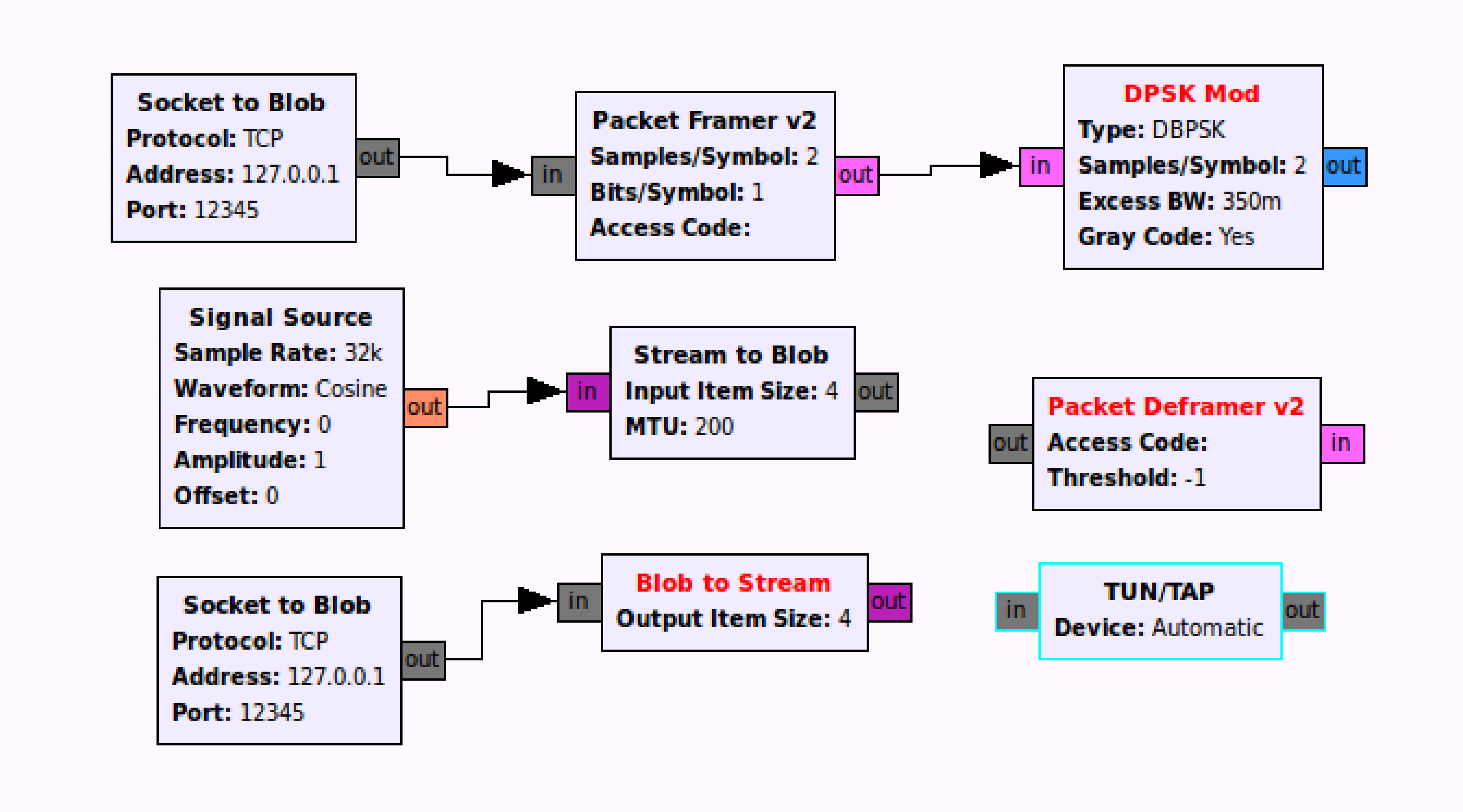
- PMT (polymorphic types)
  - serializable, reference counted objects
  - Used in stream tags
- Extensions to pmt blob
  - RO and RW pointers
  - Allocate + manage memory
- PMT Manager
  - Memory re-use for pmt objects
  - Backpressure for upstream consumers

Message Passing

- Pass message between blocks
  - Messages are gr_tag_t (key, value, srcid)
  - Implement mac layers, control planes
- Blocks have 1 input message queue
- Arbitray number of message destinations
- From the work function
  - can pop incoming messages (upstream)
  - Post to downstream subscriber group
- Scheduler has msg_connect(src, group, dst)
- http://gnuradio.org/redmine/projects/gnuradio/wiki/BlocksCodingGuide#Messages

Message passing cont...

- Some blocks that use the pmt blob type to pass bulk data
  - Socket to/from blob (preserves packet domain)
  - Stream to/from blob (stream to message domain)
  - Framer/Deframer (gr-digital's pkt.py operates on blobs)



Socket to Blob
Protocol: TCP
Address: 127.0.0.1
Port: 12345

Packet Framer v2
Samples/Symbol: 2
Bits/Symbol: 1
Access Code:

DPSK Mod
Type: DBPSK
Samples/Symbol: 2
Excess BW: 350m
Gray Code: Yes

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 0
Amplitude: 1
Offset: 0

Stream to Blob
Input Item Size: 4
MTU: 200

Packet Deframer v2
Access Code:
Threshold: -1

Socket to Blob
Protocol: TCP
Address: 127.0.0.1
Port: 12345

Blob to Stream
Output Item Size: 4

TUN/TAP
Device: Automatic

Some code for thought

```
int work(...){
    const gr_tag_t msg = this->pop_msg_queue();

    //work stuff here...
    //perhaps the message determines what we produce...
}
                              int work(...){
                                  //perhaps the input determines what messages we produce...

                                  pmt::pmt_t key = pmt::pmt_string_to_symbol("example_key");
                                  pmt::pmt_t value = pmt::pmt_string_to_symbol("example_value");
                                  this->post_msg("a message group", key, value);

                                  //work stuff here...
                              }

msg_src_block::sptr my_msg_src_block = msg_src_block::make();
msg_sink_block::sptr my_msg_sink_block = msg_sink_block::make();

gr_top_block_sptr tb = gr_make_top_block("some message flow graph");
tb->msg_connect(my_msg_src_block, "a message group", my_msg_sink_block);

tb->start(); //the flow graph is now running...
```

New component gr-blocks

- Dumping ground for misc blocks (not core)
  - Math operators, signal and noise source, delay block, stream selector
- Easy to add support for new data types
  - Complex float ok, howabout complex int16?
  - Avoid the gnuradio-core gengen paradigm
  - Templated implementations
  - Volk style naming convention
- SIMD optimized implementations
- also gr-filter

Using volk in a block

- Alignment issues
  - Tail cases, buffer alignment
  - set_output_multiple
    - Whoops finte cases
    - TODO set_input/output_alignment(...)
- See gr_blocks branch: gr-blocks/lib/add.cc
  - Generic implementation for most types
  - Implementation for floats calling volk
- Nick Foster should mention something...
  - ORC etc...

Make gnuradio blocks in python

- Make real gnuradio blocks in python
  - Overload work, general_work, start, stop...
  - Numpy types for io signatures for work
  - Stream tags and message passing too
  - Removes need for old gr_msg_queues
- Philosophy
  - Easier on user for rapid prototyping
  - Optmize for performance if you **need**

C++ / python block comparison

```
#include <gr_sync_block.h>

class my_adder_block : public gr_sync_block{
public:
    my_adder_block(...):
        gr_sync_block(
                "another adder block",
                gr_make_io_signature(2, 2, 4),
                gr_make_io_signature(1, 1, 4)
         ){}

    int work(
        int noutput_items,
        gr_vector_const_void_star &input_items,
        gr_vector_void_star &output_items
    ){
        //cast buffers
        const float* in0 = reinterpret_cast<const float *>(input_items[0]);
        const float* in1 = reinterpret_cast<const float *>(input_items[1]);
        float* out = reinterpret_cast<float *>(output_items[0]);

        //process data
        for (size_t i = 0; i < noutput_items; i++)
            out[i] = in0[i] + in1[i];

        //return produced
        return noutput_items;
    }
};
```

```
from gnuradio import gr
Import numpy

class my_basic_adder_block(gr.sync_block):
    def __init__(self, args):
        gr.sync_block.__init__(
           self,
            name="another_adder_block",
            in_sig=[numpy.float32, numpy.float32],
            out_sig=[numpy.float32],
      )

    def work(self, input_items, output_items):
        #buffer references
        in0 = input_items[0]
        in1 = input_items[1]
        out = output_items[0]

        #process data
        out[:] = in0 + in1

        #return produced
        return len(out)
```