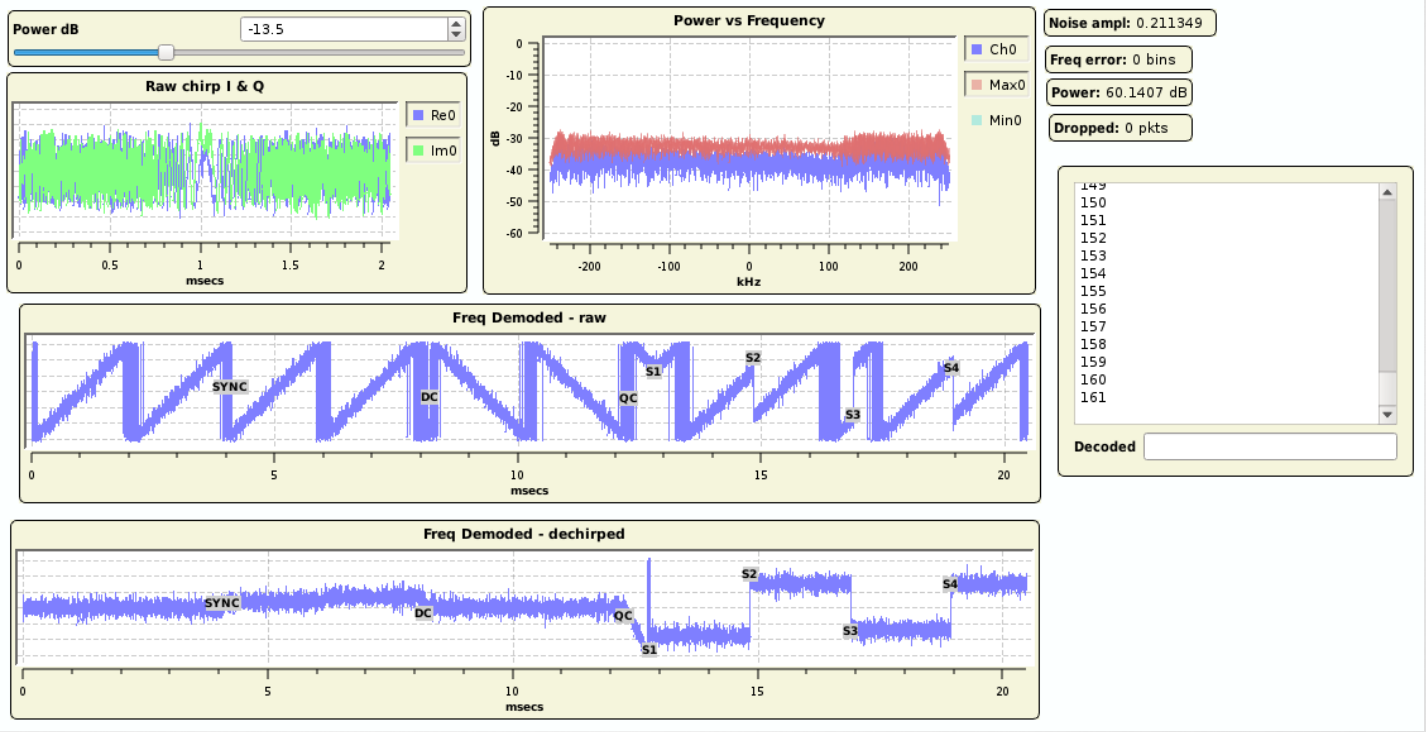
Pothos - http://pothosware.com/

- Interesting features
  - Feedback loops
  - Polymorphic streams
  - Signals slots
  - Remote topologies
  - Live reconfiguration
- Object introspection
  - Block registration
  - Block descriptions
- Wrapping Gnuradio (gr-pothos)
  - Buffer and executor integration
  - Scanning headers and GRC XML
- Proposed changes to Gnuradio
  - Factory and block registration
  - Scanning and module generation
  - Future Ideas for blocks + GRC
- Mention-ables
  - LiquidDSP



Pothos – features of interest for gr-folks

- Core stuff
  - Reconfiguration while running
  - Feedback loops
  - Signals and slot
  - Polymorphic streams
  - Single Block API
  - Buffer forwarding
  - Remote distribution
  - https://github.com/pothosware/PothosCore/wiki
  - https://github.com/pothosware/PothosCore/wiki/SchedulerExplained
- GUI stuff – PothosFlow
  - Supports features above, etc…
  - Widgets/potters are live in the graph
  - Export design to JSON topology (*new*)
  - Saving widget state (*new*)
  - Overlay support (*new*)
  - https://github.com/pothosware/PothosFlow/wiki

Pothos – block descriptions

- Block descriptions are used for the GUI (kind of like GRC XML)
- Inline markup format that looks like doxygen (in cpp or hpp)
  - Show example source
- Generated into a JSON format consumed by the GUI
- PothosLiquidDSP and gr-pothos generate the JSON
- https://github.com/pothosware/PothosCore/wiki/BlocksCodingGuide
- https://github.com/pothosware/PothosCore/wiki/BlockDescriptionMarkup

Pothos features – in the works – TODO

- Code generation
  - Qt support for JSON topology
  - Python and C++ generation
- New GUI modes
  - Going headless + reconnecting
  - Operate without live objects
- Better API support for block interaction
  - Output stream/msg data without topology
  - Access to streams outside of framework

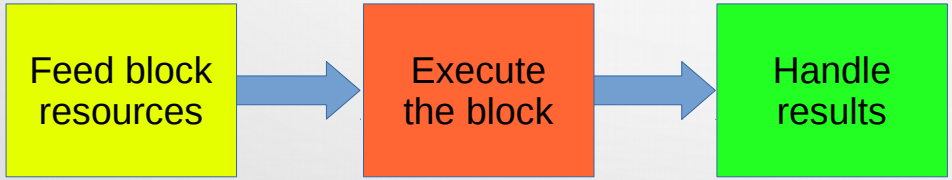
GNU Radio bindings - gr-pothos

- Ties in with gr-buffers, msgs, and tags
- Parses all of your headers and xml files
- Generates JSON block descriptions
- Generates modules with object registration
- Show the build/generator – it has colors
- How to plug into gr? Show some code
- https://github.com/pothosware/gr-pothos/wiki

gr-ideas: API for buffer, tag, msg access

Pothos

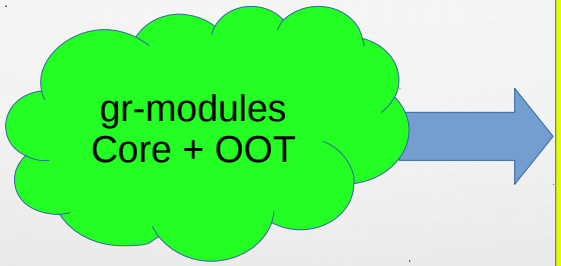
- It works in gr-pothos, how-about a formal API?
- Expose direct buffer, tag, msg injection and extraction
  1) Pass in buffers and lengths in and out
  2) Pass in input tags and input messages
  3) Existing executor: run_one_iteration()
  4) Look at buffer lengths consumed and produced
  5) Read out/pop any produced tags and messages



- Use blocks like kernels – numpy, volk
- Enables better unit tests – in some cases
- Experiment with custom scheduling OOT
- Buffers could come from DMA/OpenCl/etc
- Or better integration with some hardware
- Finally, cleanup gr-pothos
  - no special friend class
  - Use the API and gr-modules
  - Just make block descriptions

gr-ideas: block factory + class registration

- Register my_block::make function under a unique name
- Register class methods of my_block
- Generalized APIs takes vector of pmt args
  - Cleanup with C++ template wrapper
  - Cleanup with pythonic wrapper
- Build system generates block factory modules (automatic)
- Factory loads these modules by scanning the install path
- The precedent here is that python + swig does this….




```
class block
{
        static block::sptr make(pmt_args…);
        pmt_t call(name, pmt_args);

        //make API nice with C++11 below
        template<A…>
        block::sptr make(const A &a…)
        {
                args = pmt_t(a…);
                return block::make(args);
        }

        template<R, A…>
        R call(name, const A &a…)
        {
                args = pmt_t(a…);
                pmt r = this→call(name, args);
                return pmt_to<T>(r);
        }
};
```

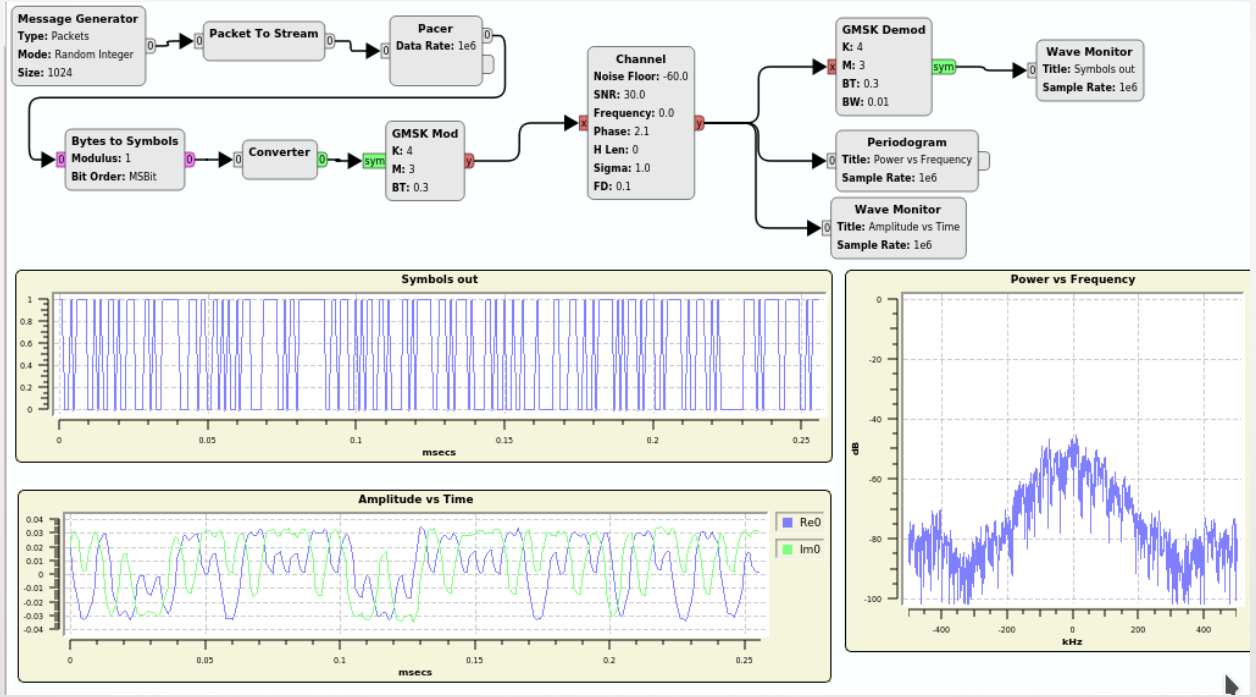
gr-ideas: block factory what ifs...

Pothos

- Create gr-blocks and gr-topologies only using the runtime API
  - In c++, we don’t need headers for devel libs from OOT projects
  - In python, we don’t need to generate or import swig bindings
- Build a topology entirely from a list of all blocks, parameters, connections
  - Example: gr-util –execute-topology=my_topology.yaml
- Remote stuff
  - Hey remote server: here is a JSON, run my topology
  - Make blocks into transparent RPC objects (serialize pmts)
- Generalized API calls into factory blocks can be made thread-safe automatically
- Downsides...
  - More complicated blocks with objects for parameters?
  - Dealing with enums, do we need a string representation?

PothosLiquidDSP

- Generates block wrappers and GUI descriptions
- Need YAML files for processing cores of interest
- Official JSON wrapper for liquidDSP in the works...
- https://github.com/pothosware/PothosLiquidDSP/wiki



Thanks!

Pothos

- https://github.com/pothosware/pothos/wiki/Support
  - https://groups.google.com/d/forum/pothos-users
  - https://twitter.com/pothosware
  - #pothos on freenode
- Questions?