GNU Radio 3.10

and other technical updates
FOSDEM 2022
Josh Morman, co-maintainer
jmorman@gnuradio.org
chat.gnuradio.org: @mormj
GNU Radio 3.10

- Released Jan 14, 2022 → 3.10.0.1 ~Jan 24
  - Packaging window for Ubuntu 22.04
- Much smaller 'irritation factor' 3.9-3.10 (than 3.7 or 3.8 to 3.9)
  - API changes, but likely little OOT rework
- Two new Modules
  - gr-pdu
  - gr-iio
- Support for Hardware Accelerators
- Updated Logging Infrastructure
- 92 Unique Contributors since v3.9.0.0
Getting 3.10

- maint-3.10 branch of https://github.com/gnuradio/gnuradio
- Radio Conda
  - https://wiki.gnuradio.org/index.php/CondaInstall
- Ubuntu PPA (for 20.04)
  - sudo add-apt-repository ppa:gnuradio/gnuradio-releases
  - sudo apt-get update
  - sudo apt install gnuradio
  
  **NOTE:** PPA doesn't play well with apt install other OOTs - install OOTs from source

- The goal is that for Ubuntu 22.04, should just be (thanks @mait)
  - sudo apt install gnuradio
  - Of course, Conda or the ppa will be better to get maintenance releases

- → Always looking for more help with packaging maintenance
PDUs (protocol data units) are the standard GNU Radio way for dealing with packetized data
- Special PMT type with metadata and data
- Upstreamed from Sandia National Labs (Jacob Gilbert)
  - Tools for manipulation of PDU objects

https://www.youtube.com/watch?v=bT60hVVte48
https://www.youtube.com/watch?v=YPfkjWtFXs
gr-iio

- Upstreamed from **Analog Devices**
- IIO is an industry standard for interacting with a wide range of devices
- Source/Sinks for iio based devices
  - PLUTO SDR
  - FMCOMMS 2/3/4

![PLUTO SDR](https://www.youtube.com/watch?v=2gKbollW6wg)
Logging Infrastructure Overhaul

Revamped Logging Infrastructure (Marcus Müller)

**spdlog** replaces Log4Cpp - which had become an unwieldy dependency

- Persistently difficult dependency to maintain and use
- **spdlog** provides much nicer and more modern logging facilities
- **libfmt** for boost::format string formatting replacement
  - Instead of GR_LOG_DEBUG(d_logger, boost::format("logging value %d") % x)
  - ... d_logger->debug("logging value {}", x)
gr_modtool OOT restructuring

- OOT structure more closely resembling the in-tree structure
- **OLD:**
  - import myoot
  - #include <myoot/myblock.h>
- **NEW:**
  - from gnuradio import myoot
  - #include <gnuradio/myoot/myblock.h>
- **WHY?**
  - We were being very presumptuous about package naming
  - Losing the association of an OOT with GNU Radio
  - Consistency
- OOTs created with 3.9 modtool (old structure) should be handled as normal with 3.10 modtool (the reverse is not true)


OOTs now installed *same level* as in-tree modules
Active Backporting

Thanks to the active backporting efforts of Jeff Long (co-maintainer), we are seeing many bug fixes and even large features show up in maintenance releases

https://github.com/gnuradio/gnuradio/blob/maint-3.9/CHANGELOG.md

- gr-soapy
  - provides access to Soapy hardware drivers using the SoapySDR driver framework
  - "out of the box" (if SoapySDR installed) support for
    - rtl-sdr
    - hackrf
    - limesdr
    - bladeRF
    - SDRPlay
    - Airspy
    - ...

3.9.5.0 and 3.8.5.0 available now!!!
Custom Buffers

- Feature introduced by **David Sorber** at **Black Lynx** via the **DARPA SDR 4.0** project
  - Working Status presented last year at FOSDEM
- Device compatible buffer structure (single mapped)
- Data able to remain in accelerator memory
  - Streamlined data movement

Prior to 3.10 using custom buffers, each connection between CUDA enabled blocks would require ingress/egress to/from device memory (expensive)

[https://www.youtube.com/watch?v=VO1zMXowezg](https://www.youtube.com/watch?v=VO1zMXowezg)
Single vs Double Mapped Circular Buffers

- GNU Radio Double Mapped Buffers
  - Contiguous read/write for the entire buffer size
  - Simple pointer arithmetic
- Device (e.g. DMA) Memory - Single Mapped
  - Readers/Writers need to handle wraparound
  - Shuffle the data when necessary
CUDA Block Example

Old way to write CUDA enabled GR blocks (pre 3.10)

constructor()

Allocate CUDA Device memory in the constructor (or worse, in the work function if necessary)

work()

Copy from the GNU Radio buffers (input_items) to the device memory

Execute kernel and check for errors

Copy from the device memory back out to the GNU Radio buffers (output_items)
Custom Buffers Example

**NEW WAY** to write CUDA enabled GR blocks in 3.10

**constructor()**

```cpp
#include <gnuradio/cuda/cuda_buffer.h>

g::io_signature::make(1, 1, itemsize, cuda_buffer::type),
g::io_signature::make(1, 1, itemsize, cuda_buffer::type),
```

**work()**

```cpp
load_cu::exec_kernel(in, 
  out, 
  gridSize, 
  d_block_size, 
  noutput_items * d_itemsize, 
  d_iterations, 
  d_stream);
check_cuda_errors(cudaPeekAtLastError());

cudaStreamSynchronize(d_stream);
```

Use the cuda_buffer class as part of the io_signature - the details of ingress/egress happen behind the scenes

Execute kernel and check for errors (in and out pointers are *already in device memory* due to cuda_buffer)

For both old and new, we need to block on the stream used for this block

- Contains CUDA Buffer Class that can be re-used in your OOT
- Similar custom buffers available for AMD HIP ([https://github.com/BlackLynx-Inc/gr-hip_buffer](https://github.com/BlackLynx-Inc/gr-hip_buffer))
Why is this Important?

- One of the value propositions of GNU Radio is the ability to accomplish a complex signal processing task by connecting a bunch of general purpose blocks together.
- Don't always want to program a special purpose block, though this can be computationally efficient.
- For accelerators, this paradigm means a lot of data movement in and out of devices unless handled properly.
- GRC already has the concept of "domains" to represent ingress/egress - applied effectively to RFNoC.
Accounting for CPU Time

- Compare GPU profile activity in the context of overall available CPU processing time (# of cores * flowgraph execution time)
- Much time is spent either idle - waiting for GPU operations to queue or complete, or synchronization and OS overhead
- Reducing the CPU overhead/idle can come through scheduler advancements → GR 4.0
For small kernels, the gains from streamlined data transfer are huge → 90% reduction
Benefit diminishes the more GPU processing that is happening in a single block
Getting Involved

If there are features you would like to see in GNU Radio, join the conversation

- chat.gnuradio.org (Matrix)
  - #gnuradio - general chat
  - #development
  - #scheduler - discussion of 4.0 / future development

- Monthly Developer Calls
  - 3rd Thursday of Every Month at 12PM Eastern - twitch.tv/gnuradio

- Scheduler Working Group
  - A number of us get together every month or so to discuss more forward looking features and ideas - working towards GNU Radio 4.0
  - Meetings announce in #scheduler