GR Operation: Streaming

GNU Radio Runtime

Options
- ID: top_block
- Generate Options: QT GUI

Variable
- ID: rate
- Value: 32k

File Source
- File: IQ_DATA/32
- Repeat: No
- Add begin tag: ()

Complex to Mag^2

QT GUI Time Sink
- Number of Points: 1.024k
- Sample Rate: 32k
- Autoscale: No

msg()
{...

work()
{
    do_some_dsp()
    out=result

“item”

Input buffer read pointer
Upstream write pointer
Downstream read pointer
Output buffer write pointer
GR Asynchronous Message Passing / PDUs

**GNU Radio Runtime**

```
handle_msg()
{
  do_other_dsp()
  publish_pdu()
}
```

**GR PDU**

```
{key1: val1, key2: val2, key3: val3}
```

```
[0, 1, 2, 3, 4, 5, 6, 7]: 8
```

**Complex to Mag^2**

```
FSK CFO Estimate
Search Depth: 2.56k
Start Offset: 0
Samples to Average: 500
Threshold: 500m
Start Finder Taps: fir_.....
```

**Burst Downmix**

```
Decimation: 16
Taps: fir_..low_pass_2/1...
```

**PDU FIR Filter**

```
Decimation: 2
FIR Taps: fir_..low_pass_.....
```
Motivation
Motivation for the PDU Utilities

- Receive a burst of information, respond with burst based on the received data
- Current tools exist but have limitations:
  - gr-uhd Stream Tags
  - Tagged Stream Blocks
  - GR Packet Communications API
  - gr-eventstream
- Complicated by real-world radio protocol constraints:
  - Minimum RF turnaround time (latency)
  - Other TDMA system constraints such as relative burst timing
  - Frequency agility, RF tuning and tracking
- ‘We tried GNU Radio but there was too much latency so we used
  \[
  \begin{align*}
  \text{libuhd} \\
  \text{C++} \\
  \text{RFNoC} \\
  \text{X-Midas}
  \end{align*}
  \]
  instead’
Required Capabilities

- **RX:** Convert received bursts of energy into discrete datasets for further processing
  - ~Sample accurate time estimate of bursts
  - Fixed or arbitrary frequency operation

- **TX:** Convert data into bursts of modulated data
  - ~Sample-accurate transmission time
  - Minimize RX/TX latency through SDR system
  - Fixed or arbitrary frequency operation

- Portable: minimize processing hardware dependencies
- Robustness: recover from overflows or dropped data
- Efficient: design for lower power embedded hardware
Existing Toolset
Existing In Tree Tools

- GR Tagged Stream API
  - Support for burst processing within the streaming architecture of GR
  - Going to be deprecated?

  “there’s a lot of bugs and inefficiencies, and quite some grief¹ and undocumented contracts connected to TSBs” - M. Müller, 2017

- gr-uhd Stream Tags
  - ‘rx_time’ provides for RX time tracking, ‘tx_time’ allows synchronized, timed transmissions
  - ‘tx_sob’ and ‘tx_eob’ tags allow for single port half-duplex transceiver
  - Not a complete solution, still streaming based
  - Since this relies on hardware, simulation can be difficult (weird SW emulation)
  - Still very useful – leveraged by gr-pdu-utils and extended to Sidekiq hardware
Existing In Tree Tools: Packet Communications API

- Powerful packet processing schema, somewhat complicated (makes use of TSBs…)
- Appears to use open loop timing, lots of rework to meet our requirements
- Interesting Module Example: https://www.gnuradio.org/deptofdefense/gr-uaslink
- Reasonable option for custom protocols and more complicated modulations
- We found the PDU Utilities to be easier for existing / simple protocols
OOT Modules: gr-eventstream

- Allows burst processing with closed loop timing by interleaving bursts into TX stream
- Relies on conventional GR backpressure: buffers will impact latency
  - Minimize number of streaming blocks between ES source and HW Sink

- Limitations for our use: T/R latency, different overall approach
  - Possible that we will integrate ES concepts in future
Theory of Operation
Usage of the PDU Utilities

- Robust set of blocks for converting between streaming and async message (PDU)

- Receive energy, basic preprocessing
- Identify start/stop of burst
- Convert required data elements to PDU
- Keep track of time
- Do higher layer processing
- Convert PDU to stream if required
- Emit RF energy at specified time
PDU Utilities Time Management

Uses the ‘tx_time’ tag key
The Timing Utilities Module

- Separated from gr-pdu_utils to contain various time tracking functions
  - Some complexity is now OBE

- Lots of time tuple related code
  - Tracking UHD Time Tuples
  - Adding periodic Time Tuples

- Does this need to be a separate module?
  - No, BUT…

```
UHD Time Tuple: (rx_time {1432 0.912341})

Timing Utils Tuple: (rx_time {1432 0.912341 243123411 1000000.0})
```
Extending The ‘Basic FSK Receiver’ Example

Channel Filter

Decimating FIR Filter
Decimation: 2
Taps: firdes.low pass 2/20...

Quadrature Demod
Gain: 2.54648

Symbol Sync
Timing Error Detector: Gardner
Samples per Symbol: 4
Expected TED Gain: 500m
Loop Bandwidth: 60m
Damping Factor: 300m
Maximum Deviation: 100m
Output Samples / Symbol: 1
Interpolating Resampler: Polyphase Filterbank, MMSE
Filterbank Arms: 128

Binary Slicer

Correlate Access Code - Tag
Access Code: 11001...10011000
Threshold: 0
Tag Name: BURST

Burst Detection

BURST: 0

BITSTREAM

Clock Recovery

Quadrature Demod
Extending The ‘Basic FSK Receiver’ Example

- We have a bitstream and know where bursts start…now what?
  - Encapsulate a complete burst of data
    - End-of-burst sequence detection
    - Encoded length field in header
    - Blind (fixed length after burst detection)
  - Create a PDU from received data
  - Pass this information to a higher-layer processor
Building a Simple TDMA FSK Modem
Building a Simple TDMA FSK Modem

Higher layer radio processing, interface to application layers, etc
“Low Latency”…?

- 'Low latency' is a key design consideration
  - Minimize time from burst RX to burst TX
- Blame the OS / Scheduler?
  - It's usually buffering...
  - Tradeoff...small buffers often means less efficient processing

\[ TR_{\text{latency}}_{\text{min}} \geq \left( \frac{\text{data_buffer_size}}{\text{sample_rate}} \right)_{\text{max}} \]

- ✓ Reduce Data Buffer Sizes
- ✓ Increase Sample Rate
- X More CPU Utilization
“Low Latency”…?

- ‘Low latency’ is a key design consideration
  - Minimize time from burst RX to burst TX

- Blame the OS / Scheduler?
  - It’s usually buffering…
  - Tradeoff…small buffers often means less efficient processing

\[ TR_{latency_{min}} \geq \left( \frac{data\_buffer\_size}{sample\_rate} \right)_{max} \]

- ✓ Reduce Data Buffer Sizes
- ✓ Increase Sample Rate
- X More CPU Utilization
Characterizing Round Trip Latency

- Flowgraph included in Timing Utilities
  - PULSE $\rightarrow$ RX $\rightarrow$ PDU $\rightarrow$ [...] $\rightarrow$ TX $\rightarrow$ SCOPE latency requires external equipment
  - TRIG $\rightarrow$ PDU $\rightarrow$ [...] $\rightarrow$ TX $\rightarrow$ RX $\rightarrow$ TIME SINK is simpler (terminate TX port of USRP)

Low Latency UHD Arguments:
- recv_frame_size = 256, num_recv_frames = 256, send_frame_size = 256, num_send_frames = 256

$\gg 4.0\text{ ms}$
Characterizing Round Trip Latency

- Flowgraph included in Timing Utilities
  - PULSE $\rightarrow$ RX $\rightarrow$ PDU $\rightarrow$ [...] $\rightarrow$ TX $\rightarrow$ SCOPE latency requires external equipment
  - TRIG $\rightarrow$ PDU $\rightarrow$ [...] $\rightarrow$ TX $\rightarrow$ RX $\rightarrow$ TIME SINK is simpler (terminate TX port of USRP)

Low Latency UHD Arguments:
```
"recv_frame_size=256, num_recv_frames=256, send_frame_size=256, num_send_frames=256"
```

Generate a ‘trigger pulse’ waveform that is coherent with...

...received data from the SDR source block

On the rising edge of the ‘trigger pulse’ waveform, emit an RF pulse PDU for immediate transmission

Tag both the trigger pulse and received pulse locations

Visualize everything, print time differences to STDOUT

RF Pulse waveform definition
Notable Blocks in the PDU Utilities Module

- **PDU/Stream Conversion:**
  - Tags to PDU
  - PDU to Bursts
  - Take/Skip to PDU
  - Tag Message Trigger

- **In-Tree Streaming Analog:**
  - Complex to Mag^2
  - Keep 1 in N
  - Binary Arithmetic
  - Up/Downsample
  - Pack/Unpack
  - PDU Alignment
  - Commutator
  - Data Encoding
  - PFB Arb Resampler
  - FIR Filter
  - FM Modulator
  - Quadrature Demod

- **New PDU Manipulation Blocks:**
  - PDU Counters
  - QT PDU Source
  - PDU Message Gate
  - Random Drop

- **Debugging Tools:**
  - Extract Field
  - PDU WPCR
  - PDU Logger
  - Flow Controller
Notable Blocks

- PDU / Stream Conversion have been covered pretty well

- Emits messages based on certain conditions
  - Robust arming/triggerring architecture based on messages or stream tags
  - Re-trigger holdoff, arm timeout, TX limits, etc
  - Can emit fixed Messages or Timed PDUs

- Converts data within U8 PDU formats
  - Stuff U8 bits into U8 bytes, bit swap, etc
  - Requires work for higher order PDU translation

- Whole Packet Clock Recovery Symbol Synch
  - Based on M. Ossmann’s WPCR project\(^1\)
  - Operates well between 4 and 60 Samples/Symbol

- Asynchronous message passing: no flow control
  - Block queues are checked, PDUs block if >Max
  - Helpful for highly variable input situations

\(^1\) https://github.com/mossmann/clock-recovery/blob/master/wpcr.py
Other Sandia GR Modules
FHSS Utilities

- Built for ISM FHSS receiver applications
  - Bursts of energy on arbitrary frequencies
- Derived from CCC’s gr-iridium
  - Similar approach, generalized
  - Efficient implementation, good sensitivity to ~6dB C/N
Real World Example:
STFT of 900 ISM Band

30 MHz

400 ms

Strong in-band CW signal
Sandia Utilities

- Assorted useful blocks that don’t fit well into other modules
- Holding area for in-tree improvements we have yet to get released
- Mostly oriented at debug applications
- Usually minor documentation if any
- Blocks are generally not extensively tested…may be dangerous

- Sometimes things get stuck in here before being moved or upstreamed…
  - Useful to install if you are using Sandia’s GR Toolkit
[Yet another] gr-sidekiq

- Coming soon…
- Extends many UHD-style features to the Sidekiq hardware
  - DDC / DUCs in FPGA
  - Command queues in FPGA (separate for instantaneous and timed commands)
  - Time tags generated by source, support for TX tag based flow/time/frequency control
- Supports direct PDU-based transmission
- Efficient implementation (thread-per-channel)
- Currently works on the M.2 and VPX2 Sidekiq variants
Summary
Takeaways

- PDU’s are underutilized, and very useful…With only a bit of frustrating PMT behavior…

```cpp
481   // OK, here comes the horrible part. Pairs pass is_dict(), but they’re not dicts. Such
482   // dicks.
483   try {
484       // This will fail if msg is a pair:
485       pmt::pmt_t keys = pmt::dict_keys(msg);
```

- The PDU Utilities help bridge GNU Radio’s streaming and PDU APIs
- There are straightforward tools available for developing bursty transceivers within GR
- Significant enhancement to GR’s in-tree PDU capabilities are available
- GNU Radio can be used for ‘reliable’ ‘low latency’ applications
- The FHSS Utilities module has a robust and efficient arbitrary burst detector and associated signal processing
Thank you!

Questions?