Crate-less Streaming DAQ
July 8, 2019

Benjamin Raydo
Electronics Group (Physics Division)

Goals

Demonstrate a streaming DAQ based on the JLab FADC250 & VTP hardware

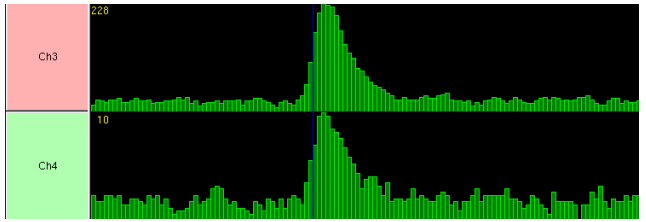
- **No expensive front-end hardware costs for development**
- **Create a viable streaming readout option based on existing Jlab hardware**
  - **Makes it easier for existing DAQ hardware in streaming mode without throwing away a lot of hardware they already have**
  - **Get existing users of Jlab DAQ hardware experienced using streaming DAQ modes**
- **Use as a testbed to help determine future streaming DAQ needs**
  - **Jlab FADC250 is fairly generic and can be used to emulate ASIC requirements/options in beam test setups**
  - **Real streaming data feed that can feed into other String Readout Grand Challenge projects**

JLAB DAQ: Existing Crate of FADC250

VXS Crate Configuration

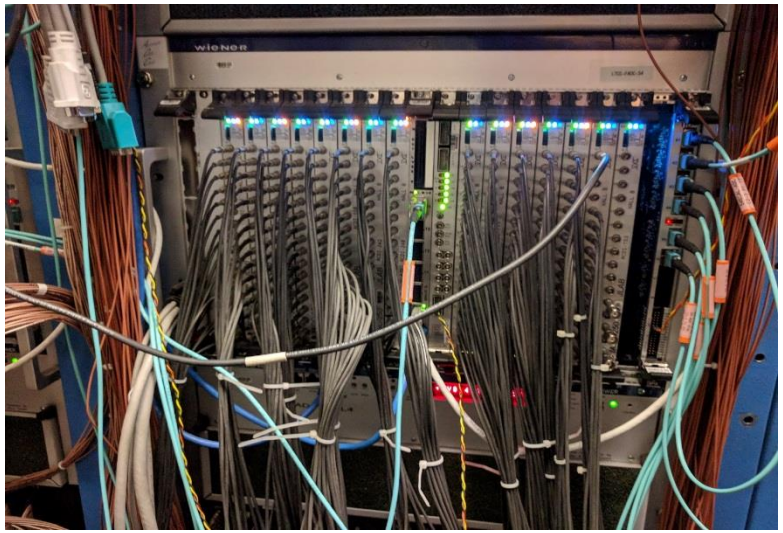
- FADC250 Modules (up to 16 per crate)
  - 16 channels per module, 256 channels per crate
  - 12b resolution, 250MHz sample rate
  - 12b * 250MHz * 256ch = 768Gbps
  - up to 20Gbps from each FADC250 module to VTP module

Example of detector pulses captured by FADC250:



- VTP Module
  - accepts up to 320Gbps from 16 FADC250 modules
  - 4x 34Gbps QSFP outputs (40Gbps for -2 speed grade)
  - 1x 40GbE QSFP (or 4x 10GbE)
  - 4GByte DDR3 (200Gbps bandwidth)
  - Trigger module (can use as concentrator and uplink)

FADC250 DAQ Crate



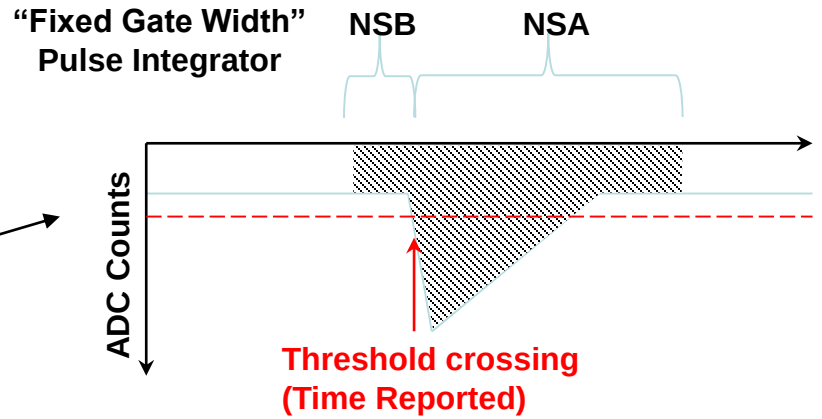
FADC250:

VTP:

FADC Raw Data Rates, Feature Extraction

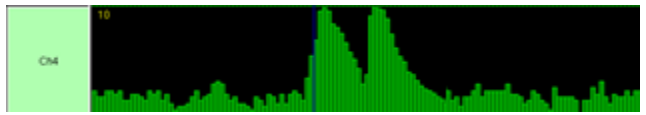
Raw FADC data rates

- 12b 250MHz: 3Gbps per channel
- 14b, 500MHz, 1GHz – way more data!
- CLAS12 & Hall D >3k FADC channels => ~10Tbps each



Feature extraction is a focus of this development

- Extract time and charge of pulses in detected FADC
  - Makes data rate from FADC independent of sample rate
  - Can describe hits FADC hits with 64bit words for high resolution time and charge
    - 10Gbps link: ~150M hits/s (600kHz average hit rate for 256 channel module)
    - 40Gbps link: ~600M hits/s (>2MHz average hit rate for 256 channel module)
    - Add more links as needed
- Plan to detect special cases and report raw waveform:
  - E.g. pile-up – a lot more data for hits, but if not common then overhead would be small



- Plan to add high resolution timing (i.e. FADC250 Mode 7/10 like)

Streaming “Event Builder”

The “Event Builder” is just building a message to send over TCP that contains FADC data corresponding to a programmable time window

- A programmable window can be set from: 32ns up to 524288ns
- All FADC hits with timestamps falling within its time window are packaged in a TCP message and sent
- An ‘end of frame’ flag tells the event builder when no more hits will arrive so it can send the message without further delay
- No trigger is required – these windows of data are continuously built and sent

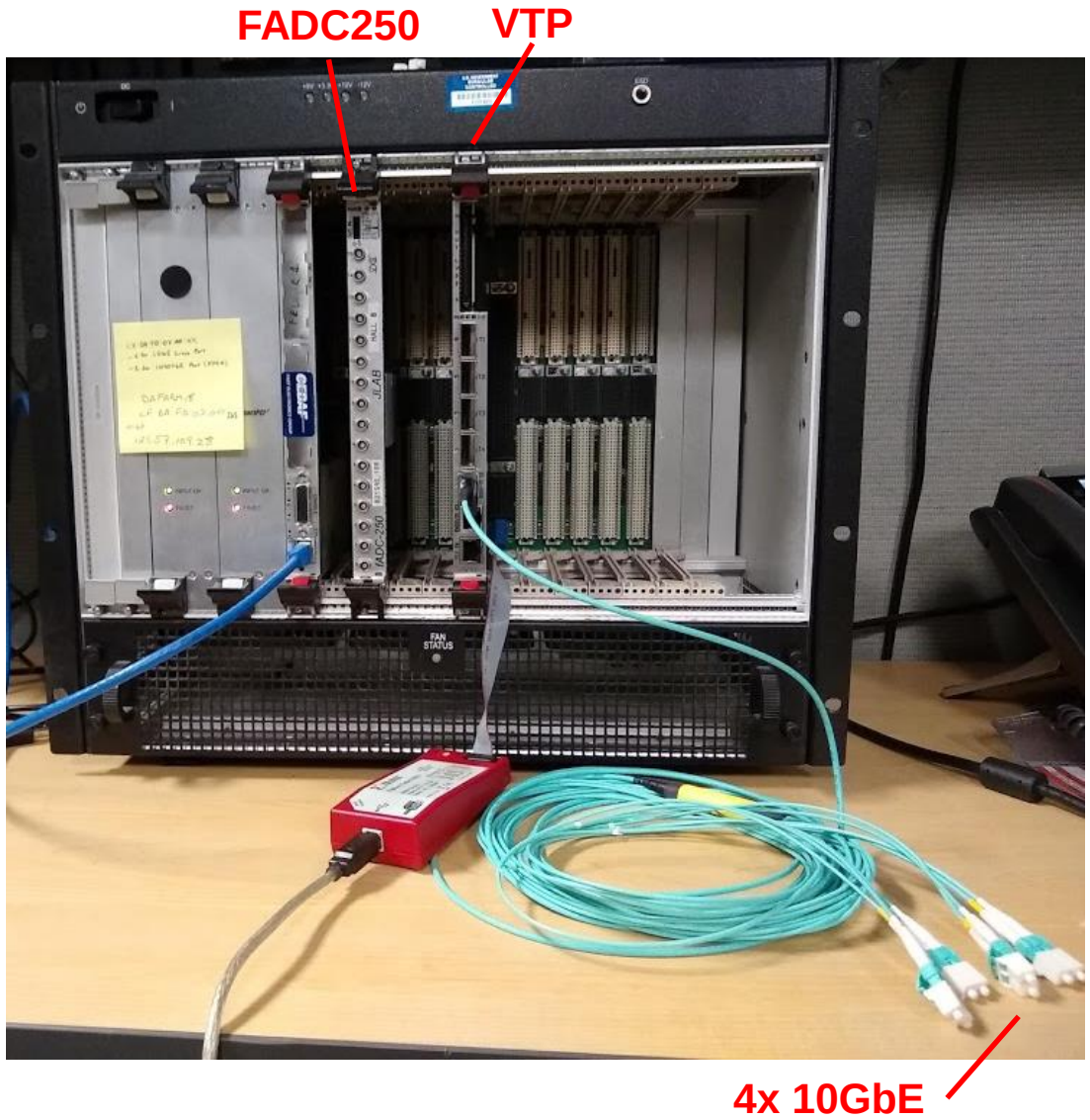
Current Test Setup

Current test setup sits in my office:

- Small VXS Crate
- 1 FADC250
- 1 VTP
- Old PC w/10GbE (Mellanox ConnectX-3)

Will move to INDRA-ASTRA lab soon

- Expanding to 16 FADC250 modules
- High performance servers

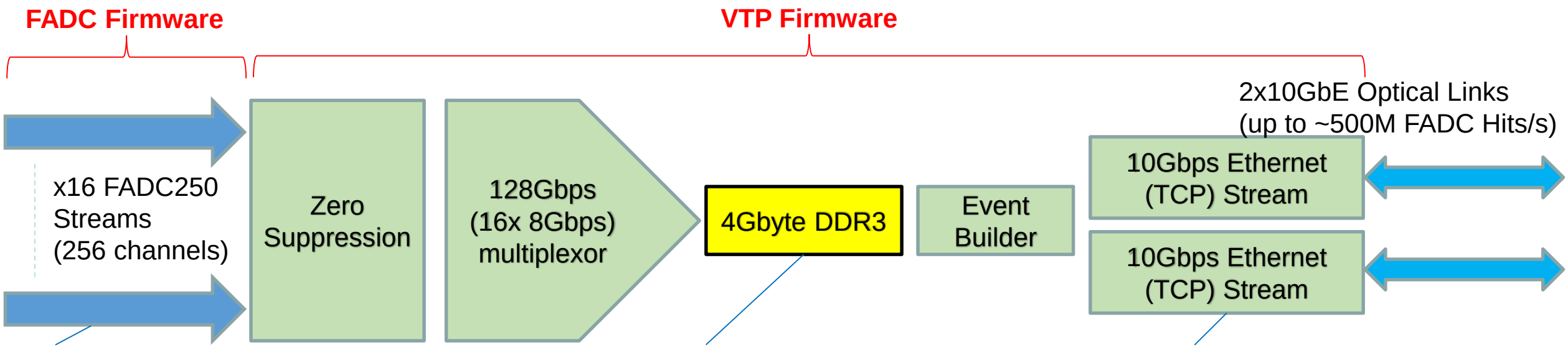


Firmware Development

FADC250 – no firmware development needed

- Reusing trigger path, which discriminates and provides pulse time and charge

VTP – nearly all firmware completed



16x FADC 8Gbps streams, each:

- 16 channels
- 32ns double pulse resolution
- 4ns leading edge timing
- Pulse integral

Large buffer:

- 200Gbps bandwidth
- Allows high burst rates
- (doesn't need to be this big, but it's what the hardware has, so might as well use it)

TCP Stack:

- Hardware accelerated
- Up to 4 links can be used
- Could be a single 40GbE (if we could afford the TCP stack IP – not worth it for R&D)

Status & Conclusion

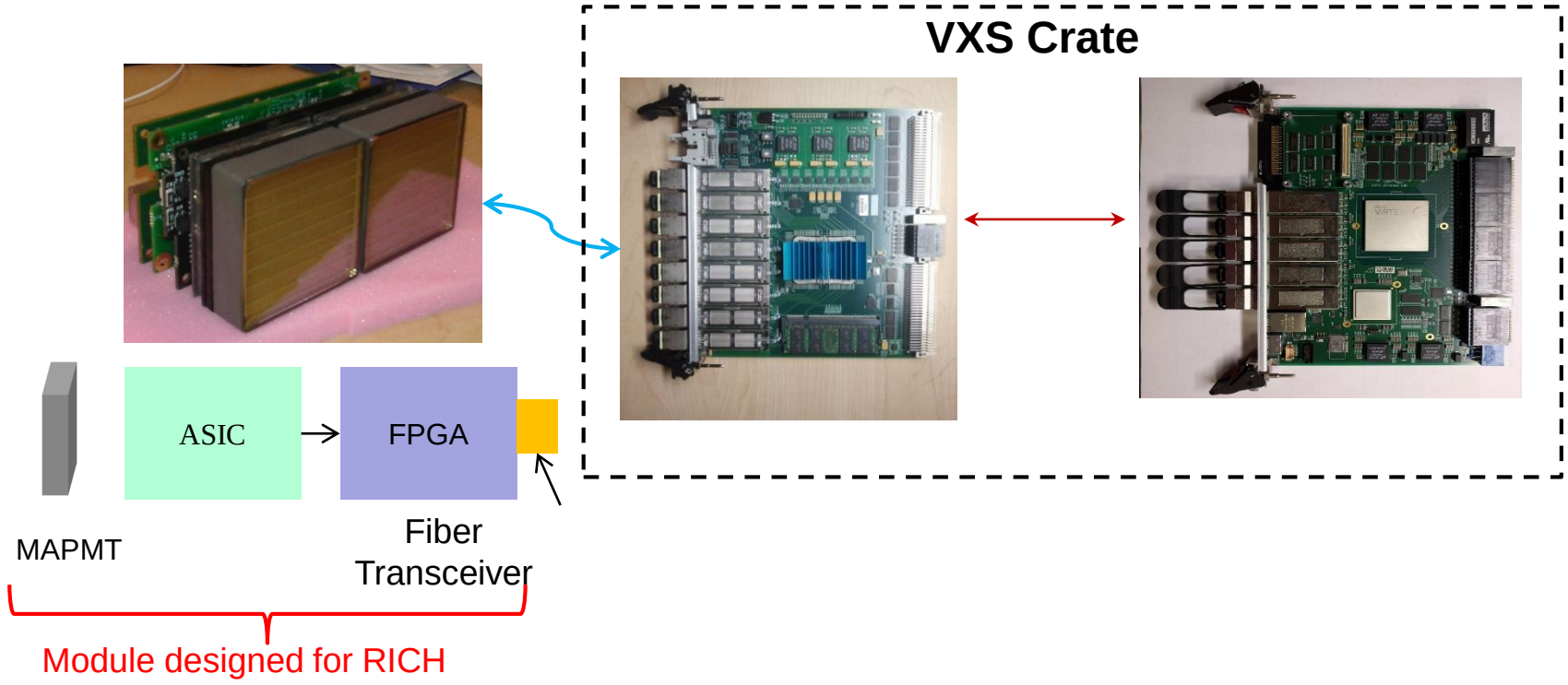
Good news: HPS commitments work is done, so I'm back on this project!

- **Here's what's been done so far:**
  - **FPGA TCP/IP hardware accelerated stack running for 10GbE interface**
    - **Still some reliability & performance issues here, but doesn't prevent remaining testing/development**
  - **FADC decoding, buffering, "event" formatting code written (not tested in hardware)**

- **What's needed to be finished:**
  - **Tie together TCP/IP interface to FADC "event" buffering**
  - **Write some scripts for automate configuration for testing**
  - **Test, debug, measure performance limitations**

- **Planning to be running in INDRA-ASTRA lab end of July**
  - **Simple testing initially: measuring benchtop pulser signals**

Misc…


VXS Crate
ASIC
FPGA
MAPMT
Fiber
Transceiver
Module designed for RICH

FADC Message Format

JLab Graham's "stream_buffer" header is used to wrap the message, making it compatible with his ZeroMQ messaging system:

```
typedef struct stream_buffer {
   uint32_t source_id;
   uint32_t total_length;
   uint32_t payload_length;
   uint32_t compressed_length;
   uint32_t magic;
   uint32_t format_version;
   uint64_t record_counter;
   struct timespec timestamp;
   uint32_t payload[];
};
```

The FADC hit information is defined within the payload[] element of the above structure:

- **First payload word is the fadc_header, followed by 1 or more fadc_hit words:**

```
typedef struct fadc_header {
   uint32_t slot:5;                 // 3 – 20 (slot number)
   uint32_t reserved0 : 10;         // 0
   uint32_t payload_type : 16; // 0x0001 (fadc hit payload type)
   uint32_t header_flag : 1;     // 1 (this is a header)
};
```

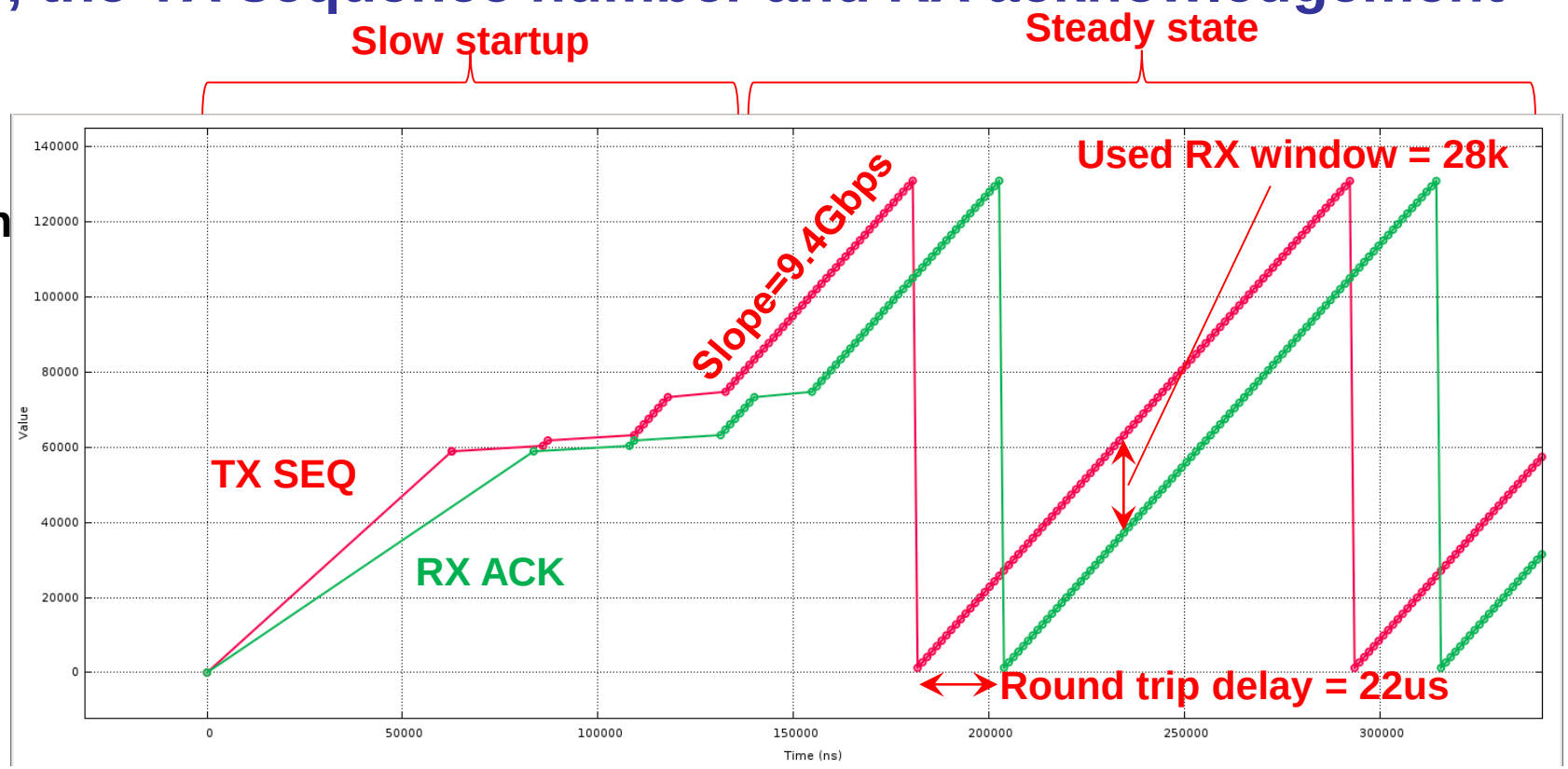
```
typedef struct fadc_hit {
   uint32_t q   : 13;               // pedestal subtracted & gained "charge"
   uint32_t ch : 4;                 // 0-15 channel number
   uint32_t t    : 14;              // 0 – 16363 hit time (in 4ns steps in window)
   uint32_t header_flag : 1;     // 0 (not a header)
};
```

- **Additional fadc_header and fadc_hit words may follow**
- **This is a fairly simple format that in the future would be expanded to handle higher resolution/dynamic range charge & time as well as raw waveform sampling.**

Simulation Performance of TCP/IP VHDL Stack

Testbench consists of a hardware accelerated TCP client and server instance. Each sends data as fast as is can to the other.

- **On TCP client sender, the TX sequence number and RX acknowledgement numbers are plotted:**

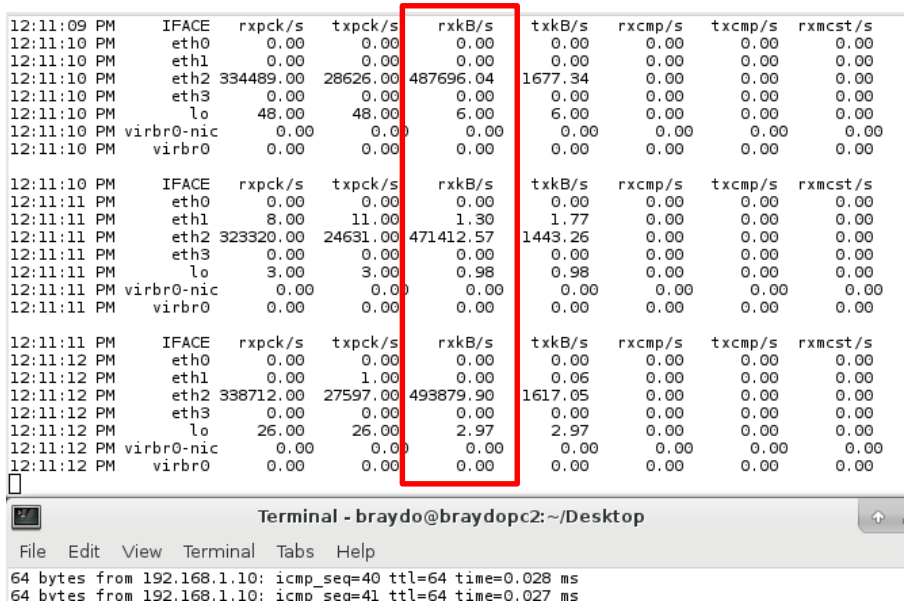


- **10µs added to each TX->RX, in attempt to overestimate delay on PC setup (that's 20µs of the shown round trip delay)**
- **VHDL stack only running with 32kBytes of TCP TX buffering. Reduction in bandwidth if round trip latency exceeds ~25µs**
- **TCP Ethernet rate: 9.4Gbps**

Performance of TCP/IP VHDL Stack sending to PC

Previous simulation rates measured would be absolutely correct if the FPGA accelerated TCP stack was used on both ends in the actual setup (realistically this is an option), but for now we're focused on using a typical Linux PC (RHEL7 64bit) to receive the TCP stream from a HW accelerated TCP stack:

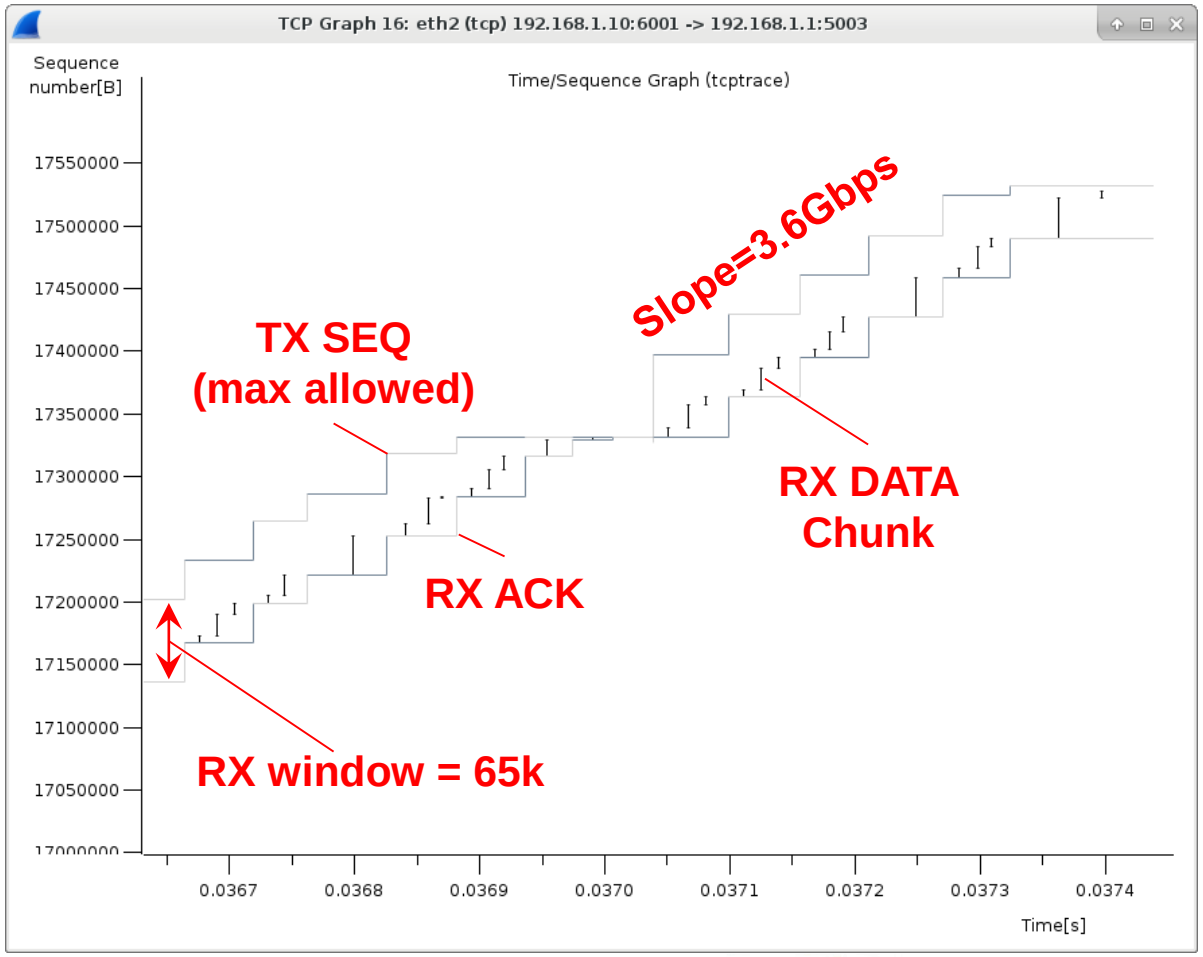
- Once again, the sender sends as fast as possible
- On the PC, the RX rate is measured: ~480MB/s
- That's only 3.8Gbps!
- Ping is showing latency slightly past the point were bandwidth will be reduced – ping is also less protocol layers and likely is faster than actual TCP processing



TCP SEQ & ACK on PC

Similar to FPGA simulation showing TCP TX SEQ and RX ACK numbers, this can be measured on the PC (not ideal, but still is useful):

- Slope of the measurement indicates 3.6Gbps
- RX DATA Chunk slope is much higher (probably >9Gbps)
- You can see that RX DATA Chunks never fill past 32kByte of the window – it stops and waits until the RX ACK increments…So this confirms the 32kByte TCP TX Buffer is causing the slow down, but this is also a result of the large latency
- Strangely a 1GbE NIC card on the same machine gives a ping of <=10µs when talking to a similar FPGA TCP stack, making me suspicious of the 10GbE driver or settings
- In any case, we'll likely look at increasing the TX buffer size to deal with this issue.



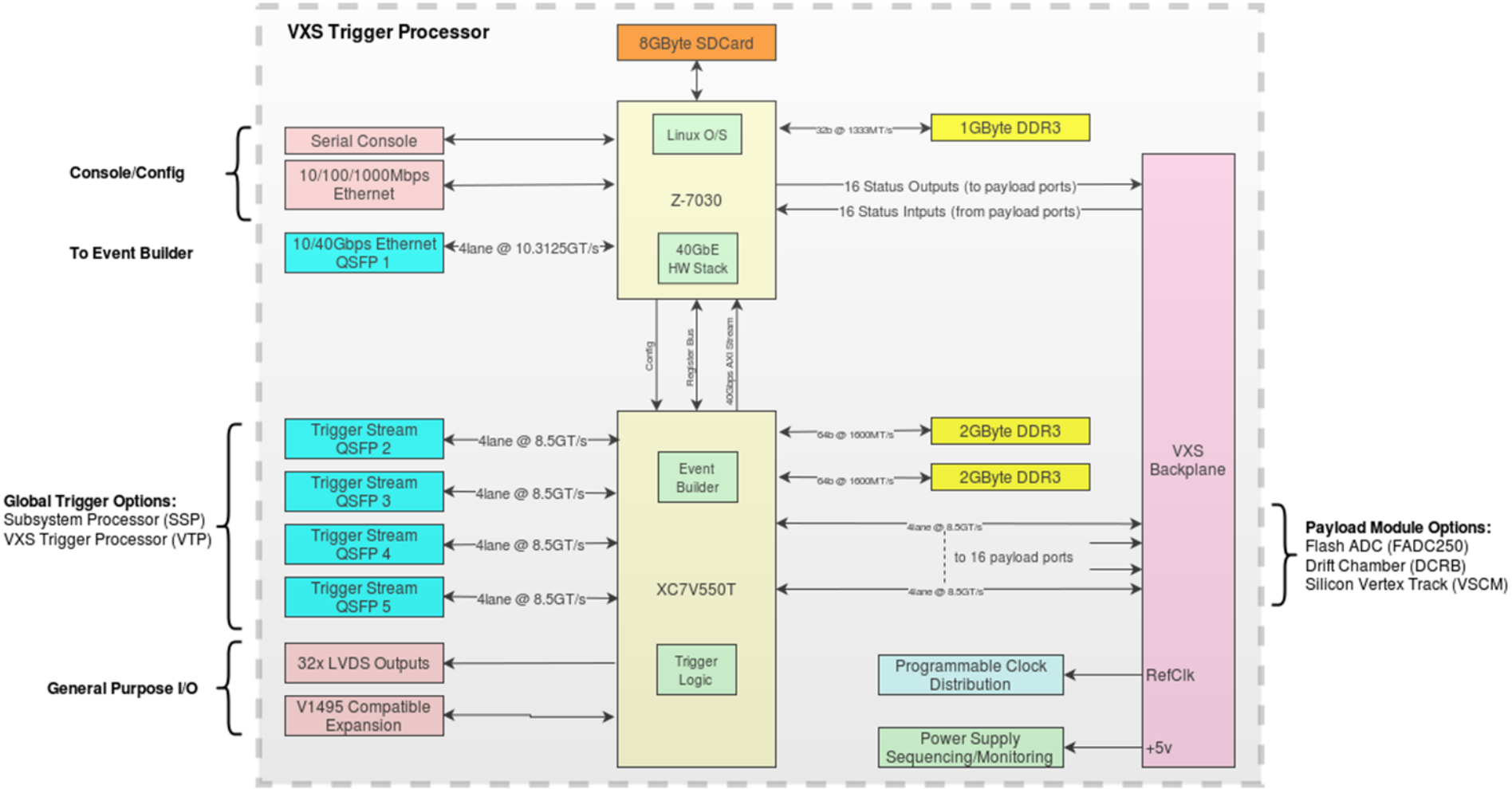
Scheduling…

Got a slow start on this so far due to CLAS12 operations…

- ~1 week of effort done so far in FY2019

- Here's what's been done so far:
  - FPGA TCP/IP hardware accelerated stack running for 10GbE interface
    - Still some reliability & performance issues here, but doesn't prevent remaining testing/development
  - FADC decoding, buffering, "event" formatting code written (not tested in hardware)

- What's needed to be finished:
  - Tie together TCP/IP interface to FADC "event" buffering
  - Write some scripts for automate configuration for testing
  - Test, debug, measure performance limitations


VXS Trigger Processor
8GByte SDCard
Linux O/S
32b @ 1333MT/s
1GByte DDR3
Serial Console
10/100/1000Mbps Ethernet
Console/Config
Z-7030
16 Status Outputs (to payload ports)
16 Status Intputs (from payload ports)
10/40Gbps Ethernet QSFP 1
4lane @ 10.3125GT/s
To Event Builder
40GbE HW Stack
Config
Register Bus
40Gbps AXI Stream
VXS Backplane
Trigger Stream QSFP 2
4lane @ 8.5GT/s
Trigger Stream QSFP 3
4lane @ 8.5GT/s
Trigger Stream QSFP 4
4lane @ 8.5GT/s
Trigger Stream QSFP 5
4lane @ 8.5GT/s
Event Builder
64b @ 1600MT/s
2GByte DDR3
64b @ 1600MT/s
2GByte DDR3
4lane @ 8.5GT/s
to 16 payload ports
4lane @ 8.5GT/s
Global Trigger Options:
Subsystem Processor (SSP)
VXS Trigger Processor (VTP)
XC7V550T
Payload Module Options:
Flash ADC (FADC250)
Drift Chamber (DCRB)
Silicon Vertex Track (VSCM)
32x LVDS Outputs
Trigger Logic
General Purpose I/O
V1495 Compatible Expansion
Programmable Clock Distribution
RefClk
Power Supply Sequencing/Monitoring
+5v