FADC250 Streaming over Ethernet using TCP/IP
May 23, 2019

Benjamin Raydo
Electronics Group (Physics Division)

TCP/IP FADC Readout Goals

- Satisfy LDRD commitment
  - Demonstrate a streaming DAQ based on the Jlab FADC250 & VTP hardware

- 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 options in beam test setups

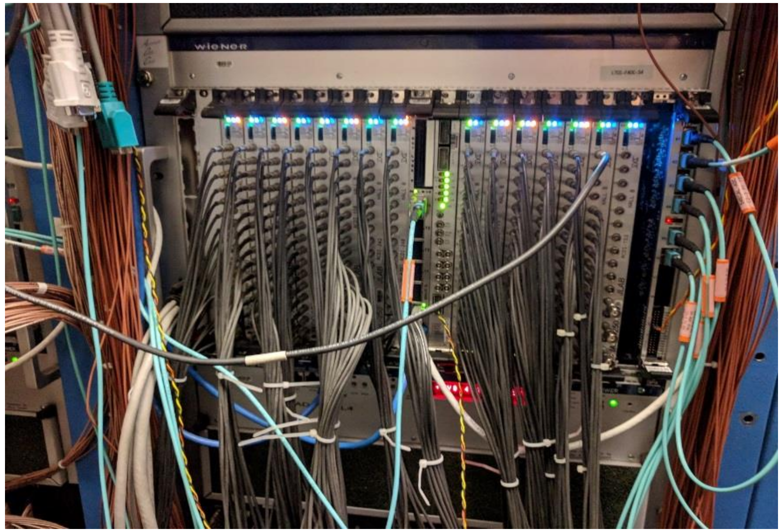
The goal does not include using this implementation as a solution if we must build/purchase new hardware!

JLAB DAQ Crate: FADC250

VXS Backplane

- 21 Slots: 2 VXS switch, 18 VXS/VME payload, 1 VME
- VME CPU for event readout (up to 200MB/s)
- 16 VXS payload slots for front-end modules
- FADC250 crate configuration shown
  - 16 channels per module, 256 channels per crate
  - 12b resolution, 250MHz sample rate
  - 10 to 20Gbps from each FADC250 module to VTP module for triggering
- VTP Trigger Module
  - accepts up to 320Gbps from 16 FADC250 modules
  - 4x QSFP trigger outputs (34Gbps due to -1 FPGA)
  - 1x 40GbE QSFP (or 4x 10GbE)
  - 4GByte DDR3 (200Gbps bandwidth)

FADC250 DAQ Crate

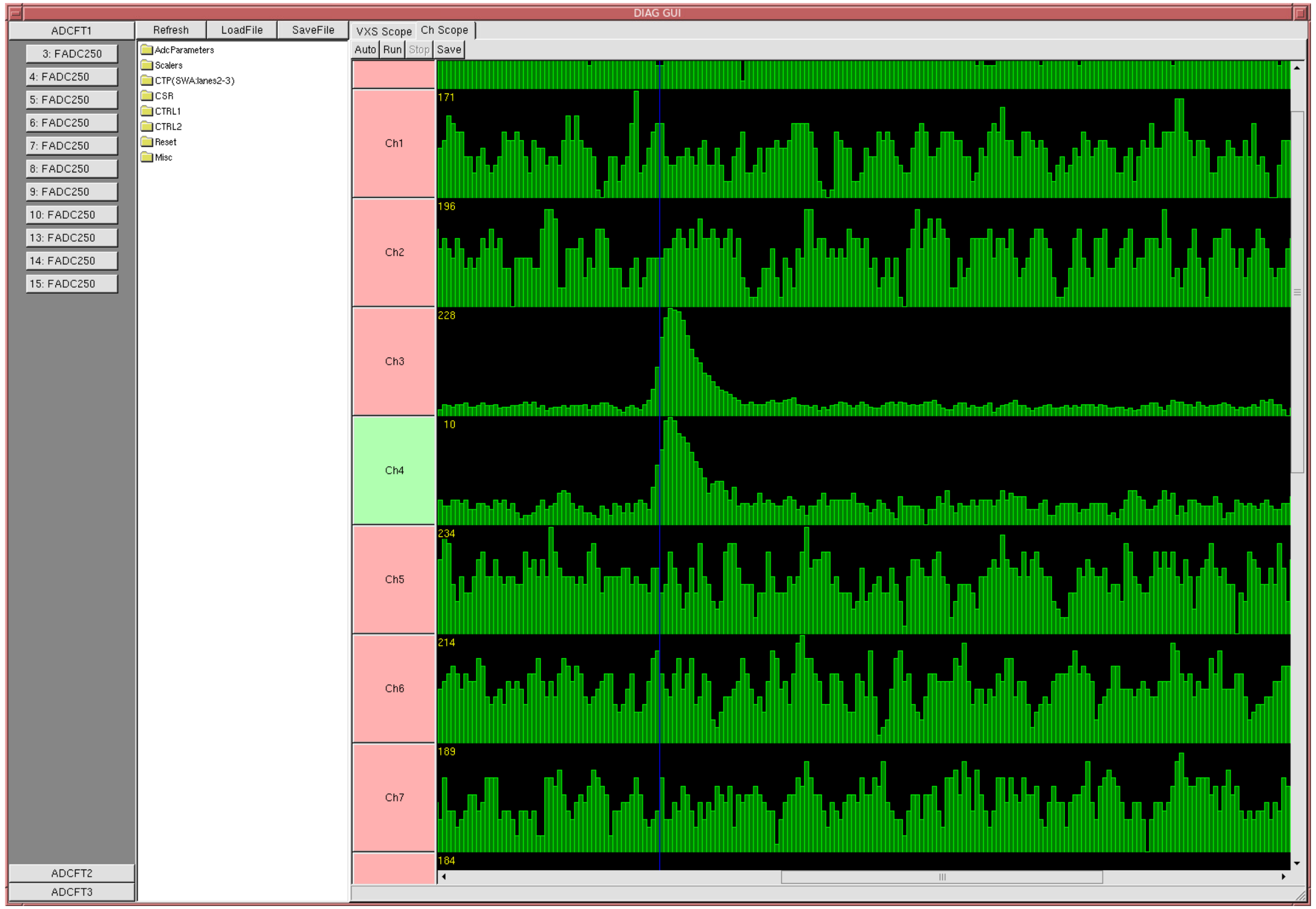


FADC250:

VTP:

FADC250 Waveforms

Typical pulses digitized by the FADC250 modules (CLAS12 FT Calorimeter cosmic pulses)



FADC250 Feature Extraction

Readout Path (only runs when triggered)

- Raw samples around threshold crossing
- Charge & coarse time (4ns leading-edge)
- Charge & high resolution time (62.5ps constant-fraction)

Trigger Path (continuously running)

- Charge & coarse time (4ns leading-edge)

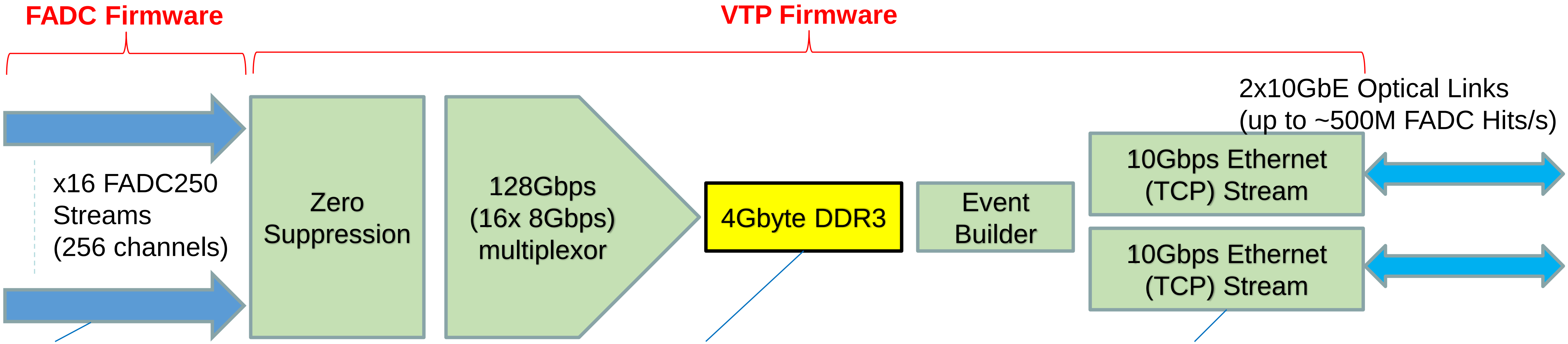
For the FADC250 streaming readout test, only “Trigger Path” is planned for test (which means no FADC250 firmware changes are necessary at the moment). Supporting other feature extraction methods can be done later (which currently isn’t the main focus of the FADC250 streaming readout at the moment)

Firmware Development

FADC250 – no firmware development needed

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

VTP – nearly all firmware completed


FADC Firmware
VTP Firmware
x16 FADC250 Streams (256 channels)
Zero Suppression
128Gbps (16x 8Gbps) multiplexor
4Gbyte DDR3
Event Builder
2x10GbE Optical Links (up to ~500M FADC Hits/s)
10Gbps Ethernet (TCP) Stream
10Gbps Ethernet (TCP) Stream

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)

“Event Builder”

The “Event Builder” is just building a message to send over TCP that contains all FADC hits 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**

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.**

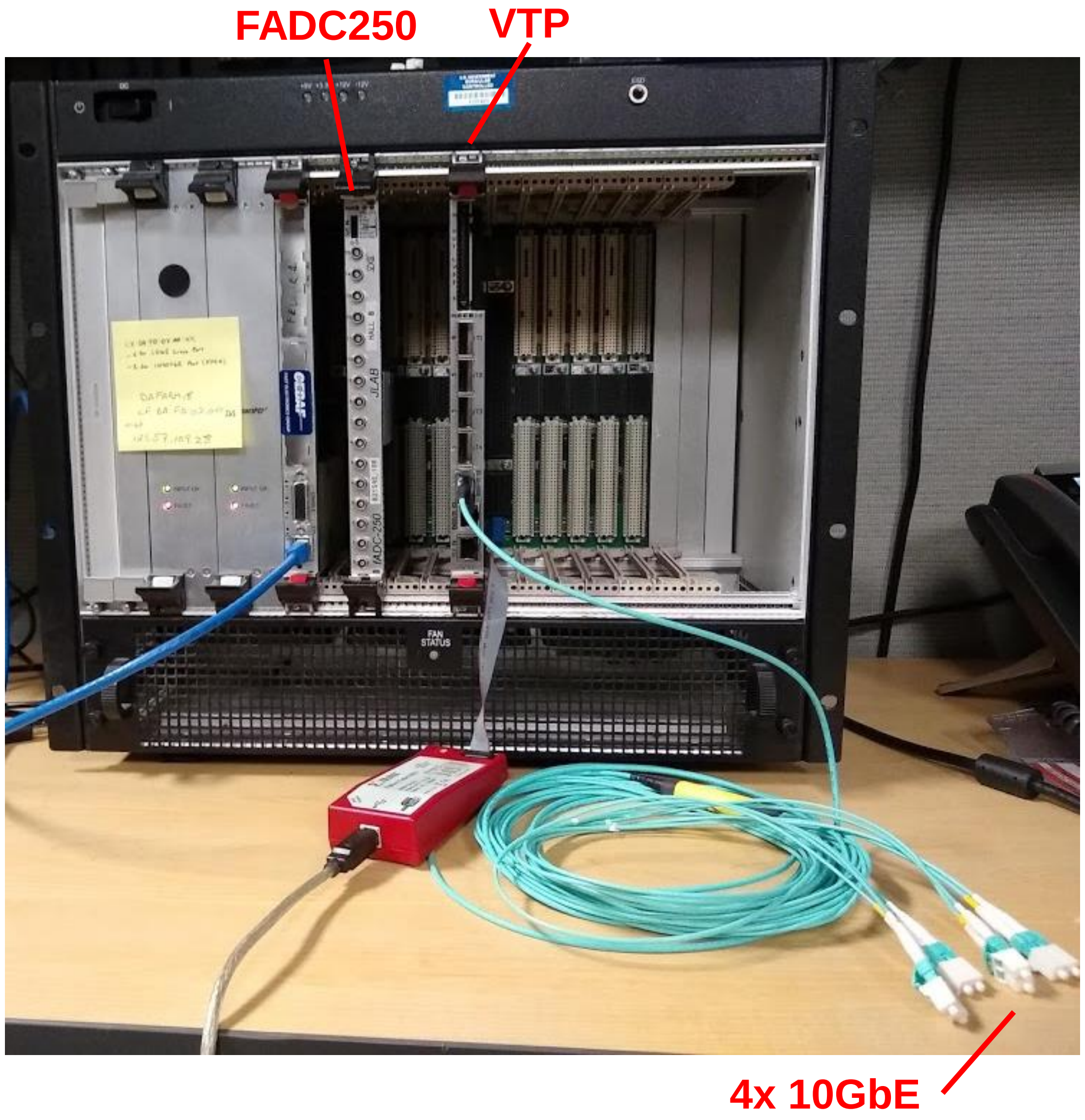
Current Test Setup

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

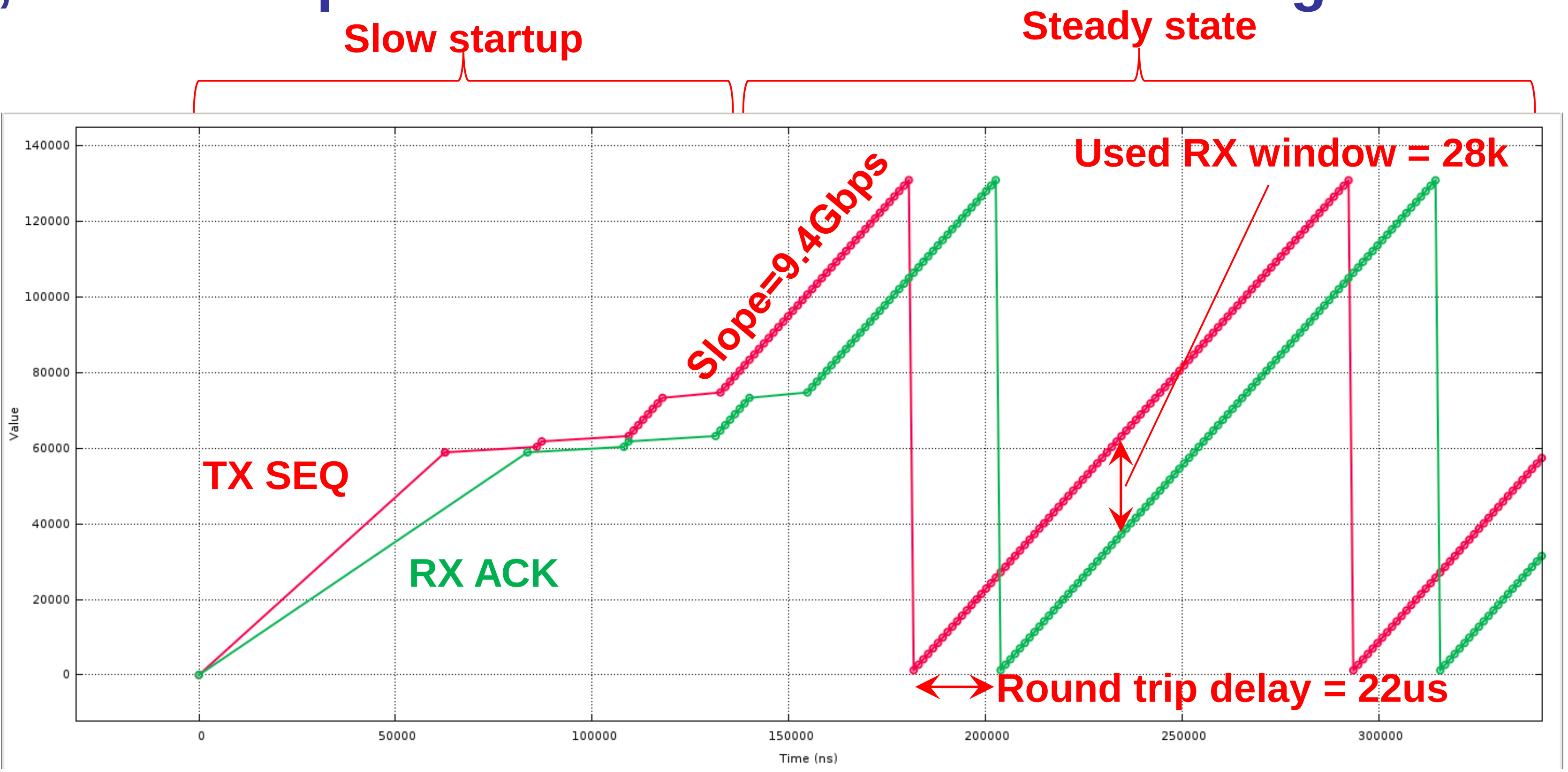


Simulation Performance of TCP/IP VHDL Stack

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:**

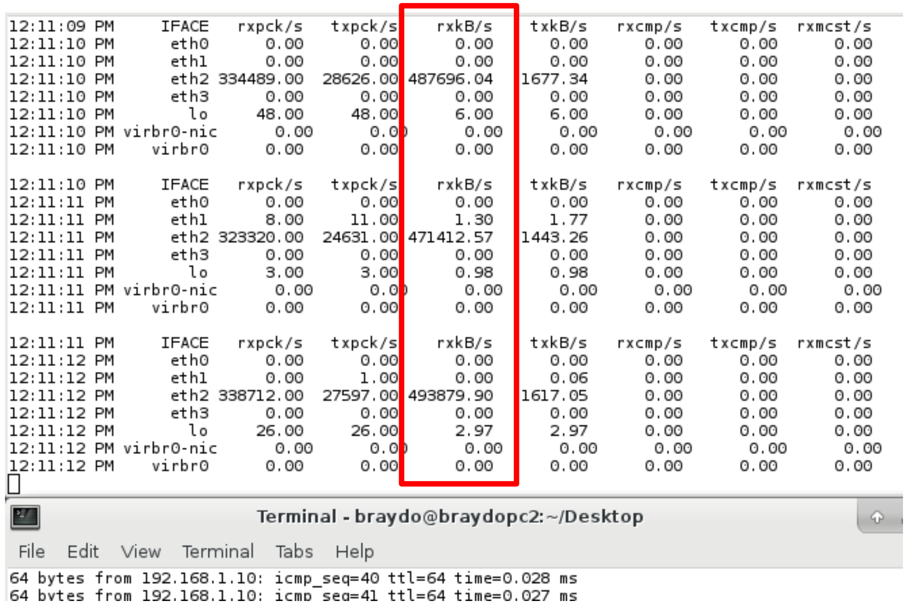
- **10µs added to each TX->RX, in attempt to overestimate delay on PC setup (that's 20µs of the shown round trip delay)**
- **VHDL stack only running with 32kBytes of TCP TX buffering. Reduction in bandwidth if round trip latency exceeds ~25µs**
- **TCP Ethernet rate: 9.4Gbps**



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



```
12:11:09 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
12:11:10 PM      eth0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:10 PM      eth1      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:10 PM      eth2 334489.00  28626.00 487696.04   1677.34      0.00      0.00      0.00
12:11:10 PM      eth3      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:10 PM        lo     48.00     48.00      6.00      6.00      0.00      0.00      0.00
12:11:10 PM virbr0-nic      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:10 PM    virbr0      0.00      0.00      0.00      0.00      0.00      0.00      0.00

12:11:10 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
12:11:11 PM      eth0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:11 PM      eth1      8.00     11.00      1.30      1.77      0.00      0.00      0.00
12:11:11 PM      eth2 323320.00  24631.00 471412.57   1443.26      0.00      0.00      0.00
12:11:11 PM      eth3      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:11 PM        lo      3.00      3.00      0.98      0.98      0.00      0.00      0.00
12:11:11 PM virbr0-nic      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:11 PM    virbr0      0.00      0.00      0.00      0.00      0.00      0.00      0.00

12:11:11 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
12:11:12 PM      eth0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:12 PM      eth1      0.00      1.00      0.00      0.06      0.00      0.00      0.00
12:11:12 PM      eth2 338712.00  27597.00 493879.90   1617.05      0.00      0.00      0.00
12:11:12 PM      eth3      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:12 PM        lo     26.00     26.00      2.97      2.97      0.00      0.00      0.00
12:11:12 PM virbr0-nic      0.00      0.00      0.00      0.00      0.00      0.00      0.00
12:11:12 PM    virbr0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

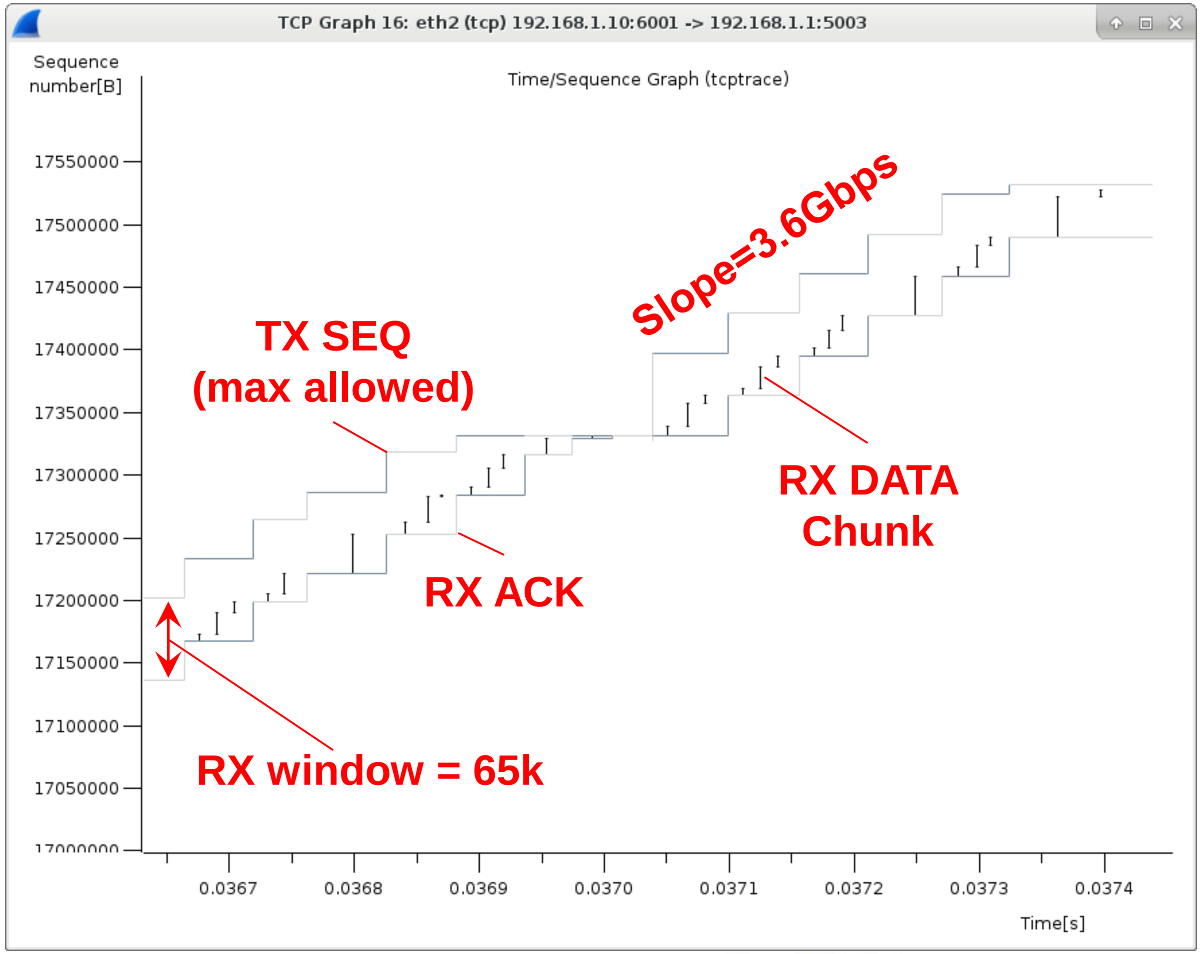
Terminal - braydo@braydopc2:~/Desktop
File Edit View Terminal Tabs Help

```
64 bytes from 192.168.1.10: icmp_seq=40 ttl=64 time=0.028 ms
64 bytes from 192.168.1.10: icmp_seq=41 ttl=64 time=0.027 ms
```

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)**

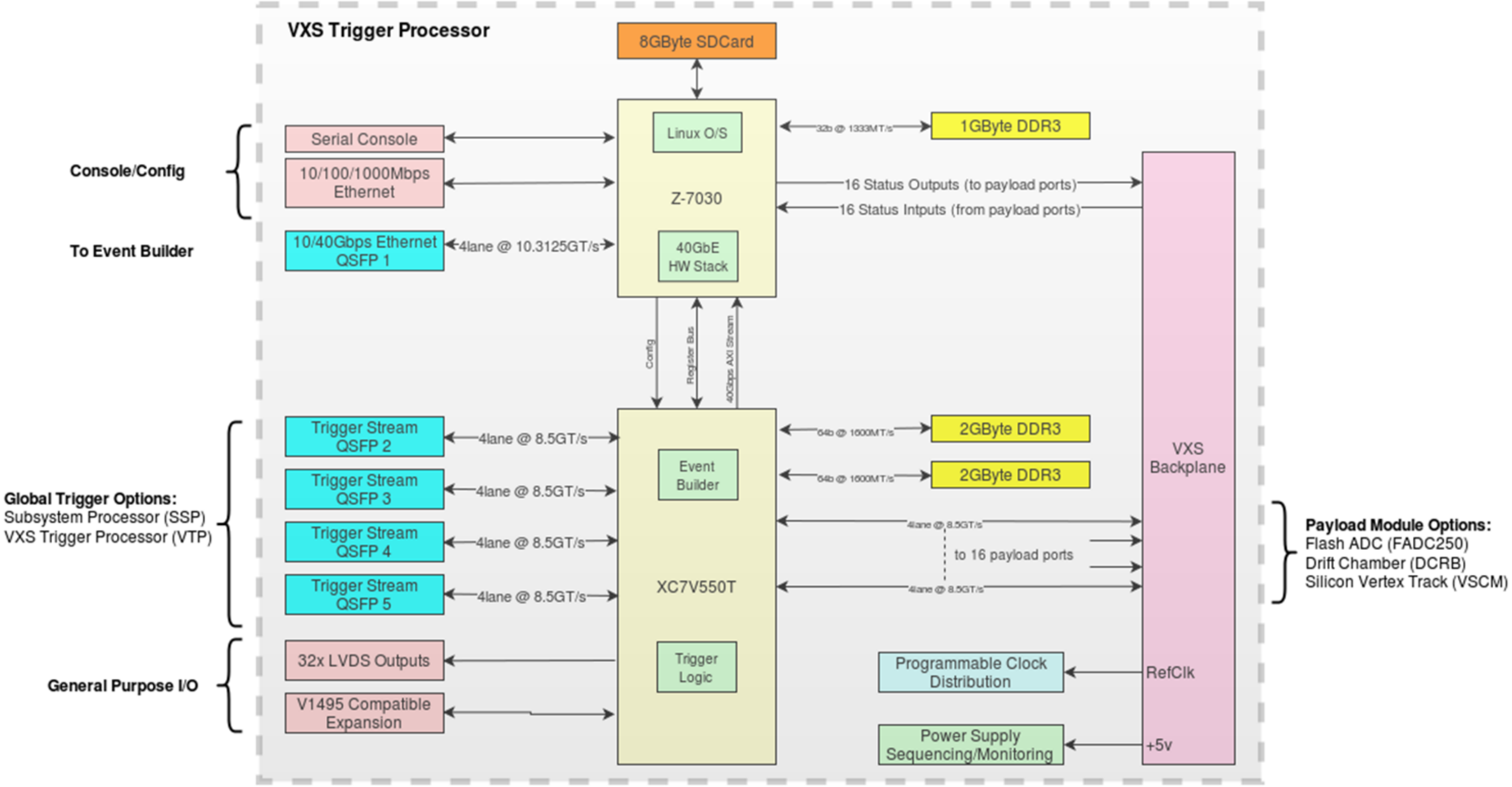
- **What's needed to be finished:**
  - **Tie together TCP/IP interface to FADC "event" buffering**
  - **Write some scripts for automate configuration for testing**
  - **Test, debug, measure performance limitations**

Conclusion

- TCP hardware accelerated stack is probably one of the trickier parts that luckily we have a vendor providing. We have found a number of issues with the IP, but the vendor has been working with us to resolve them – shouldn’t prevent us from reach test goals.

- Delays on my part due to CLAS12 & HPS experiment preparations, but these will be complete in the next few weeks so I can actually spend good time to wrap up this project!

- Making progress towards FADC250 crate streaming over Ethernet using TCP – expected to have a functional demonstration this summer!



VXS Trigger Processor

8GByte SDCard

Z-7030

Linux O/S

40GbE HW Stack

32b @ 1333MT/s

1GByte DDR3

16 Status Outputs (to payload ports)

16 Status Intputs (from payload ports)

Console/Config

Serial Console

10/100/1000Mbps Ethernet

To Event Builder

10/40Gbps Ethernet QSFP 1

4lane @ 10.3125GT/s

Config

Register Bus

40Gbps AXI Stream

Global Trigger Options:
Subsystem Processor (SSP)
VXS Trigger Processor (VTP)

Trigger Stream QSFP 2

Trigger Stream QSFP 3

Trigger Stream QSFP 4

Trigger Stream QSFP 5

4lane @ 8.5GT/s

4lane @ 8.5GT/s

4lane @ 8.5GT/s

4lane @ 8.5GT/s

XC7V550T

Event Builder

Trigger Logic

64b @ 1600MT/s

2GByte DDR3

64b @ 1600MT/s

2GByte DDR3

4lane @ 8.5GT/s

to 16 payload ports

4lane @ 8.5GT/s

VXS Backplane

Payload Module Options:
Flash ADC (FADC250)
Drift Chamber (DCRB)
Silicon Vertex Track (VSCM)

General Purpose I/O

32x LVDS Outputs

V1495 Compatible Expansion

Programmable Clock Distribution

RefClk

Power Supply Sequencing/Monitoring

+5v