# Upstream DAQ Technology Review

- 100s Buffer – Firmware
- Criteria based assessment.

David Cussans

29/07/2020

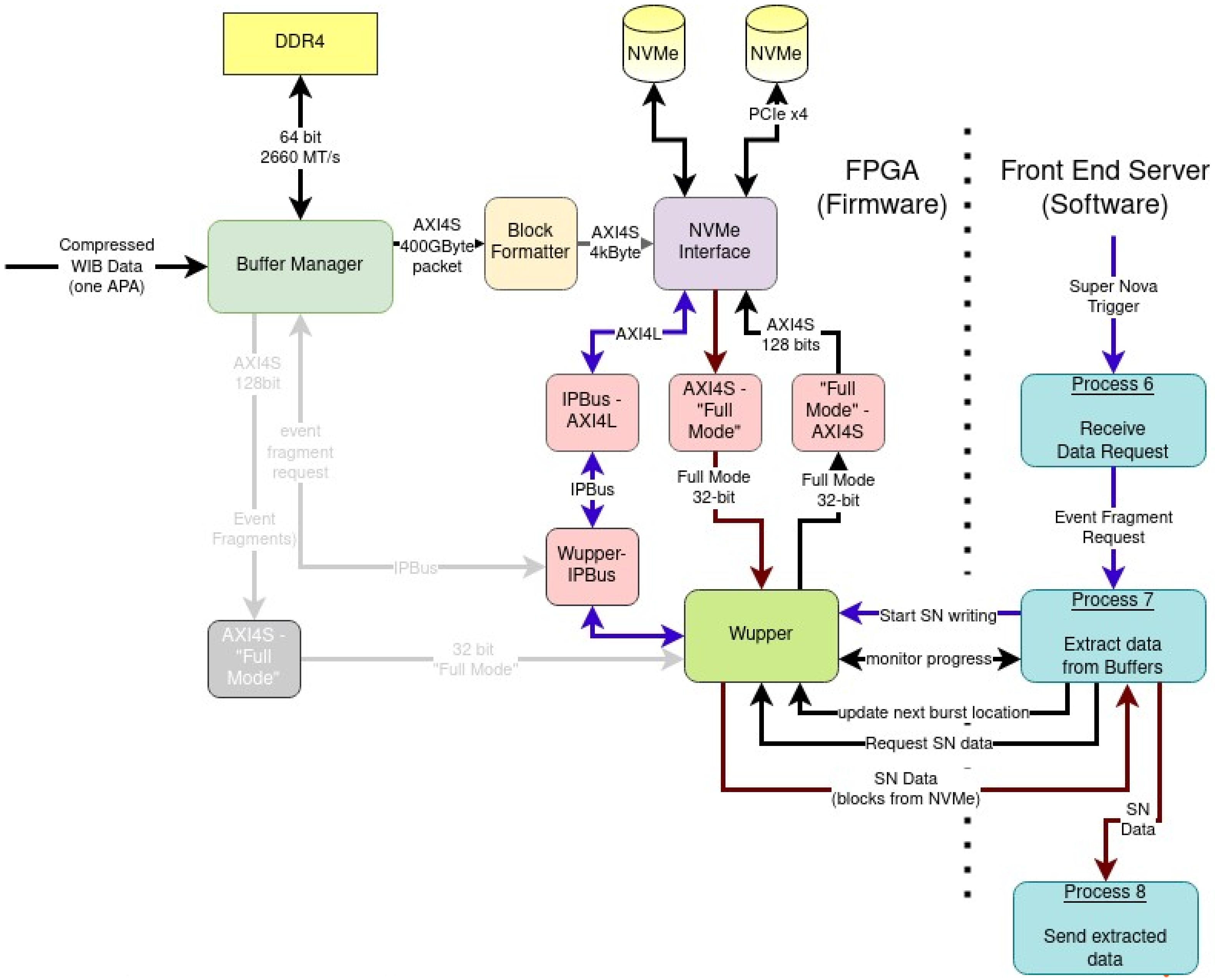
## Outline

- Introduction to 100s buffer
- Stream Formatter
    - Split ~ 320 GByte packet from buffer manager into 4kByte packets for storage
- NVMe Storage
    - Compressed data written to two drives in 100 sec.
- Integration with DUNE DAQ
- Readout Operation Modes
- Response to "Known Failure" scenarios
- Criteria Based Assessment
    - Features
    - Adaptability
    - Reliability
    - Maintenance
    - Resource

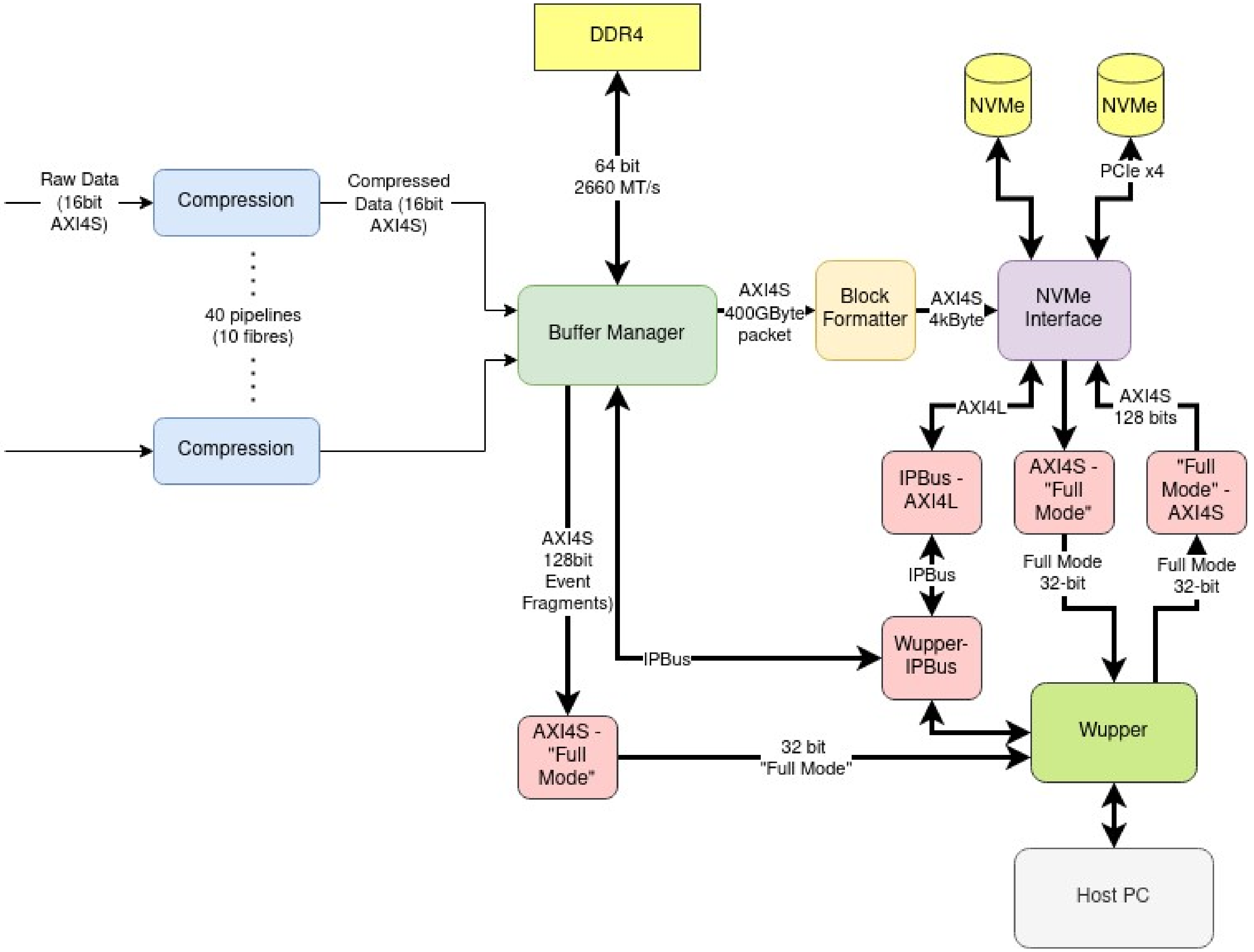
# Bandwidth Requirements (TPC)

- 2590 ADC channels per APA
- 2 Msample/s, 12-bit
- → 60Gbit/s raw data
- ~ 10% overhead for headers etc.
- Factor ~ 2.6 compression (on limited ProtoDUNE study)
- → 3250 MBytes/s data rate to buffer

# Message/Data flow for 100s buffer



## Firmware Part



## Software Part

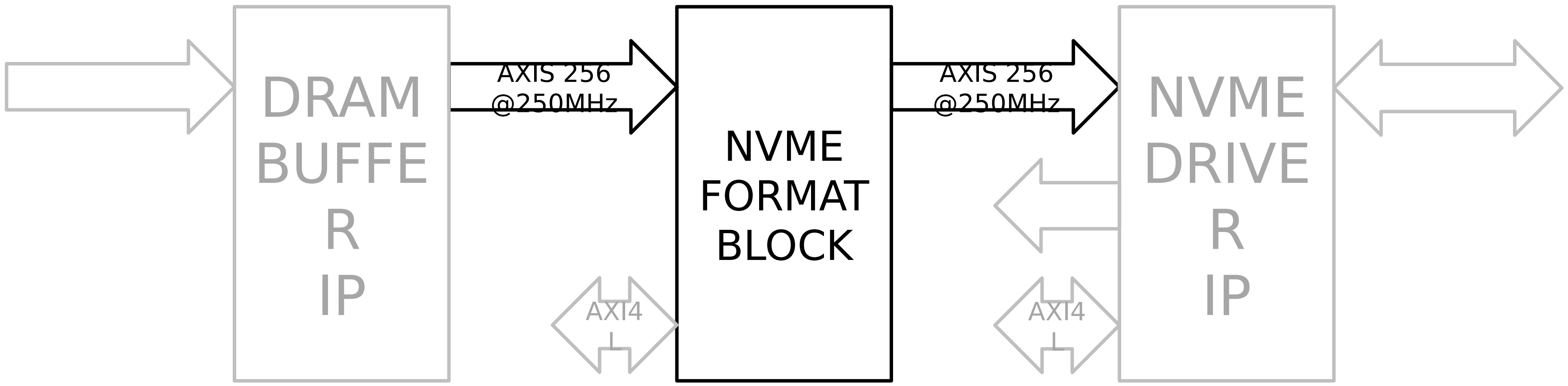
- At initialization:
  - Set up size of Super Nova capture
    - Write to IPBus register over Wupper interface (using Felix routines)
  - Set start location of Super Nova data in NVMe
    - Write to IPBus register over Wupper
- During run, on receipt of SN trigger:
  - Write to IPBus register over Wupper to initiate Super Nova capture
  - Monitor progress of capture
  - When complete, update start location of SN data in NVMe, ready for next trigger
  - Read SN data from NVMe drives
    - Write to IPBus register, receive AXI4 packet converted to full-mode packet.
  - Combine data from two NVMe drives, strip headers, check CRCs, transmit to back-end DAQ.

Science & Technology Facilities Council

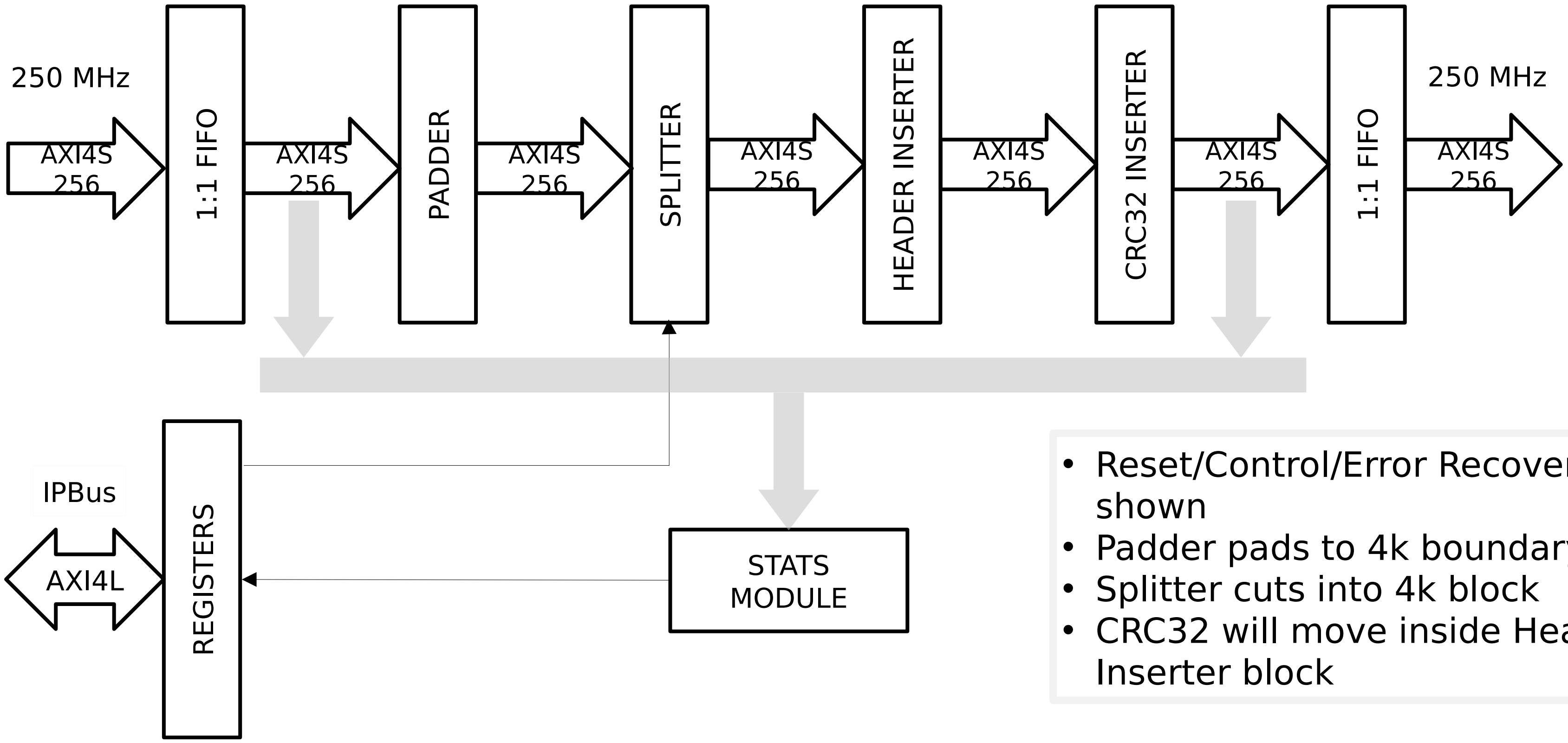
UK Research and Innovation

# DUNE Block Formatting Firmware

# NVME Format Block

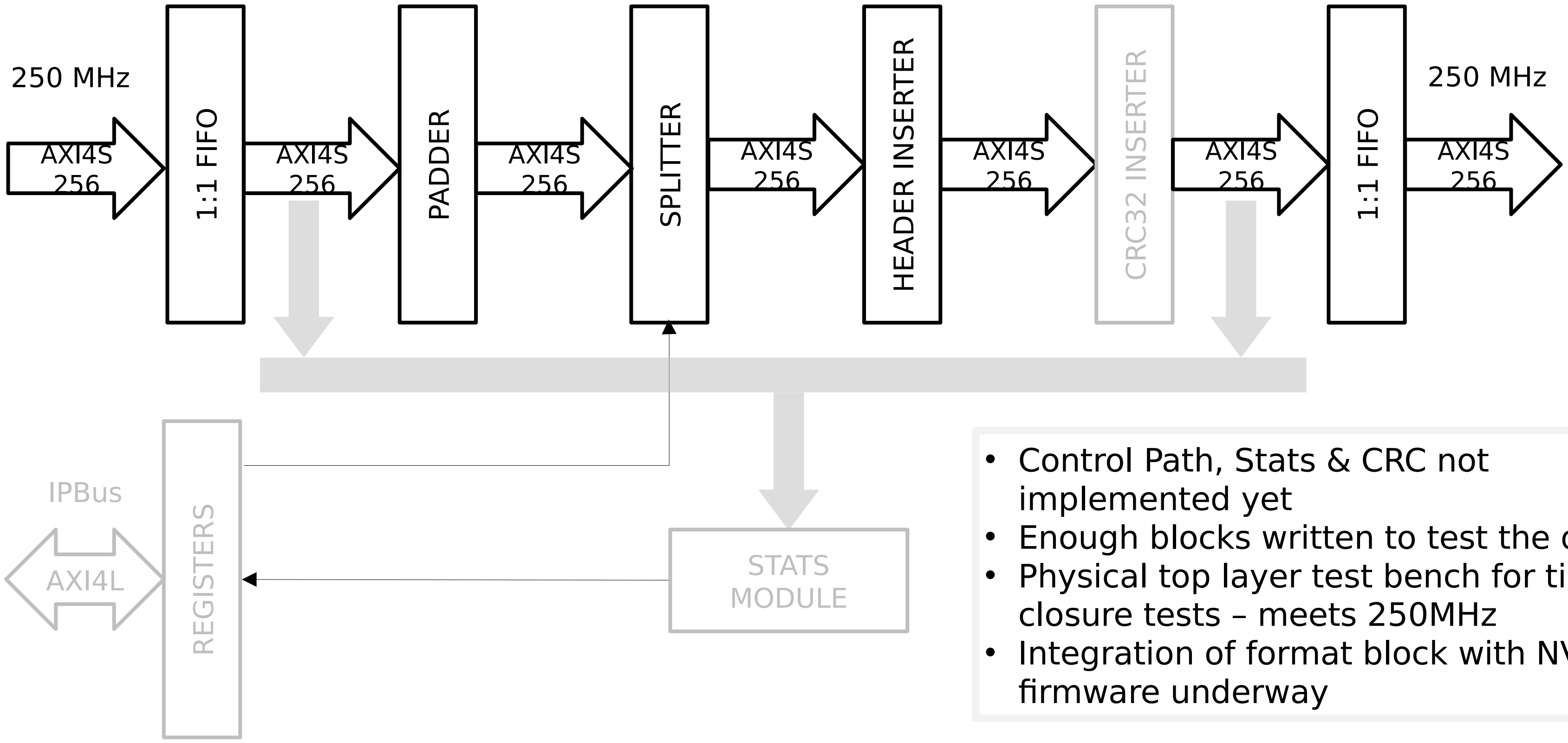


## NVME Format Block Internal



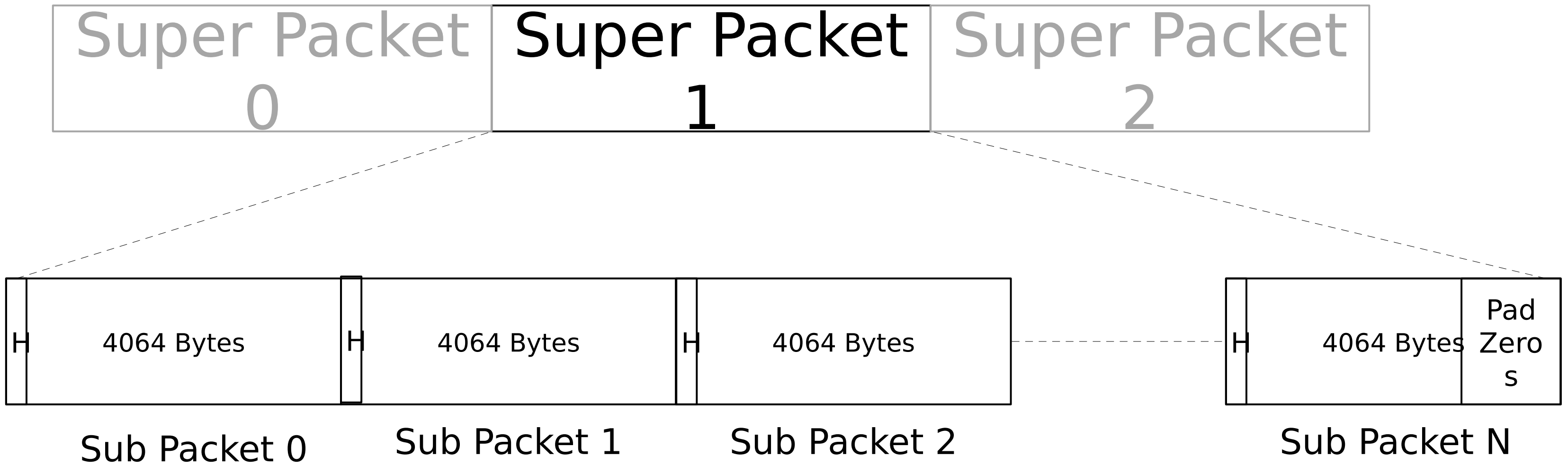
- Reset/Control/Error Recovery not shown
- Padder pads to 4k boundary.
- Splitter cuts into 4k block
- CRC32 will move inside Header Inserter block

# NVME Format Block Internal - implemented



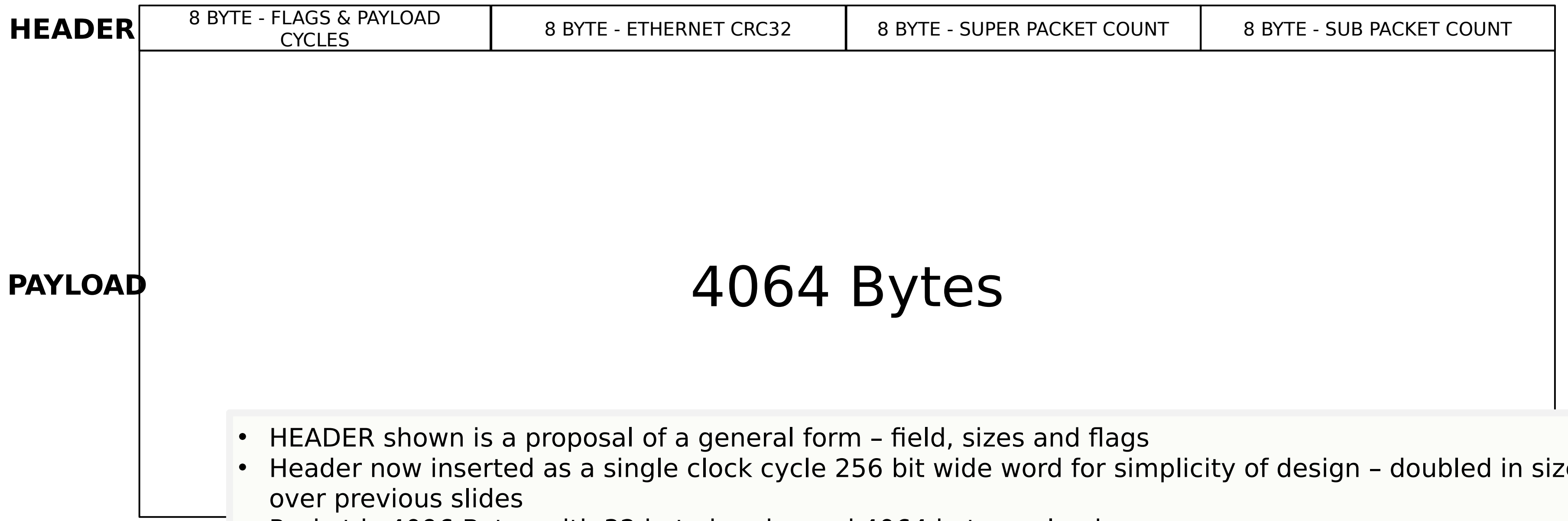
- Control Path, Stats & CRC not implemented yet
- Enough blocks written to test the design
- Physical top layer test bench for timing closure tests – meets 250MHz
- Integration of format block with NVMe firmware underway

# NVME Super Packet to Sub Packet



- Super Packet (Large) is broken into multiple sub packets which are 4K Bytes and are labelled with a Header (32B) containing super packet no, sub packet number, CRC32, payload cycles and flags.
- The super packet number is constant for all its sub packets and the sub packet number increments fro 0 to N where is the number of sub packets -1 contained within the super packet.
- The last packet may have padding (zeros) added and payload cycles will be lower in size in the header of the last packet to reflect this

## NVME Sub Packet Format

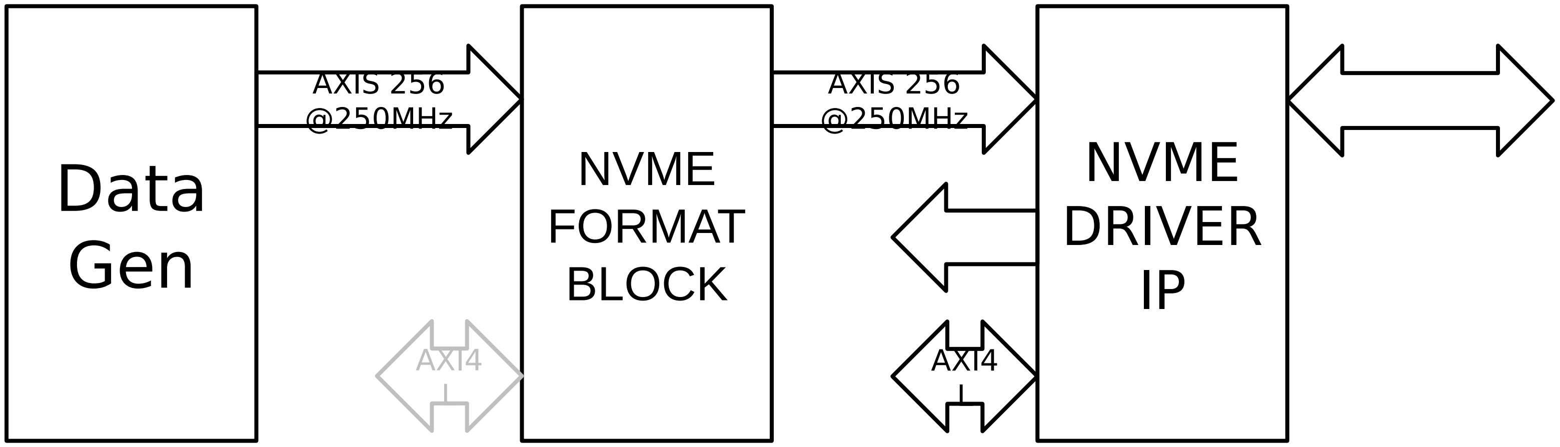

HEADER
8 BYTE - FLAGS & PAYLOAD CYCLES
8 BYTE - ETHERNET CRC32
8 BYTE - SUPER PACKET COUNT
8 BYTE - SUB PACKET COUNT
PAYLOAD
4064 Bytes

- HEADER shown is a proposal of a general form – field, sizes and flags
- Header now inserted as a single clock cycle 256 bit wide word for simplicity of design – doubled in size over previous slides
- Packet is 4096 Bytes with 32 byte header and 4064 byte payload
- Some flexibility in field sizes and header contents
- Uniquely identify packets & detect errors

Science & Technology Facilities Council

UK Research and Innovation

## NVME + Formatter Block Test Bench

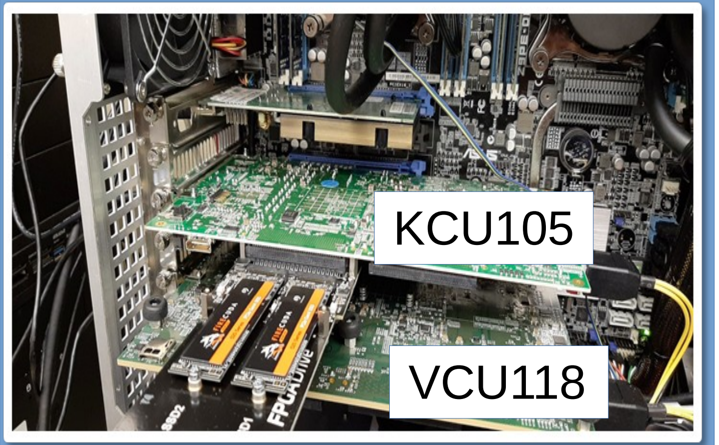


- In Progress

- Insert Data Gen and NVEM Formatter Block into Beam test design
- Use to write our data stream and format into NVME drives via Beam IP
- Added ILAs to AXIS buses in and out of NVME Formatter block
- Design Compiles – debug in progress
- Needs test software to confirm operation

29/07/2020

## Summary

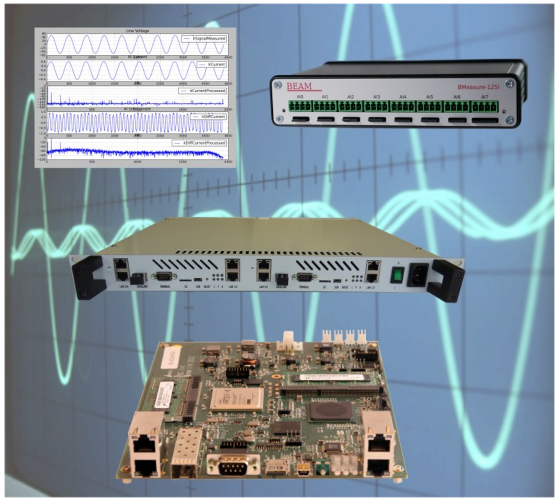


- *Initial version of formatter Block - on STFC Gitlab and CERN Gitlab*
  - https://gitlab.cern.ch/DUNE-SP-TDR-DAQ/nvme-block-formatting
- *Timing closure at 250 MHz standalone*
- *Integration with Beam test design in progress*
- *Use our data gen and formatter as data source*
- *Some minor bugs to resolve in formatter/checker*
- *Add features such as CRC, monitoring, control plane as we move forward and integrate designs*
- Needs integration with firmware build tool (ipbb) and rest of project

## NvmeStorage – Beam Ltd

Terry Barnaby of Beam Ltd

- Beam: Electronics and Software Engineering
- Instrumentation a focus
- Electronics design
- Software design
- Unit/product design
- Scientific systems
- Based in Yate, near Bristol in the UK
- https://portal.beam.ltd.uk/support/dune



## NVMe Documentation

- Beam: https://portal.beam.ltd.uk/support/dune/
- Git repository (clone URL)
  https://portal.beam.ltd.uk/git/DuneNvme.git
- General documentation is at:
- Doxygen documentation at:
  https://portal.beam.ltd.uk/support/dune/files/doc/DuneNvme/fpga/html/

## NvmeStorage - Introduction

Dune DAQ – NVMe event storage

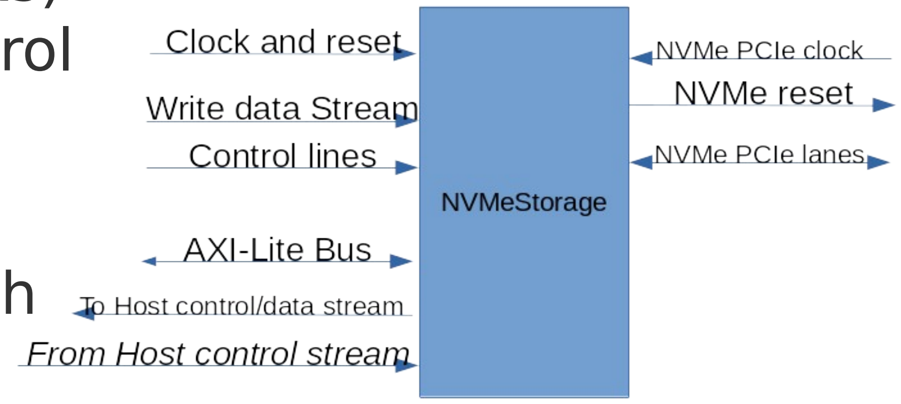
FPGA module to store 20 – 200 GBytes of event data
4 GBytes/s data rate for 100 seconds
512 Gbyte drives – so two 200 GByte events per drive

## NvmeStorage: Data structure

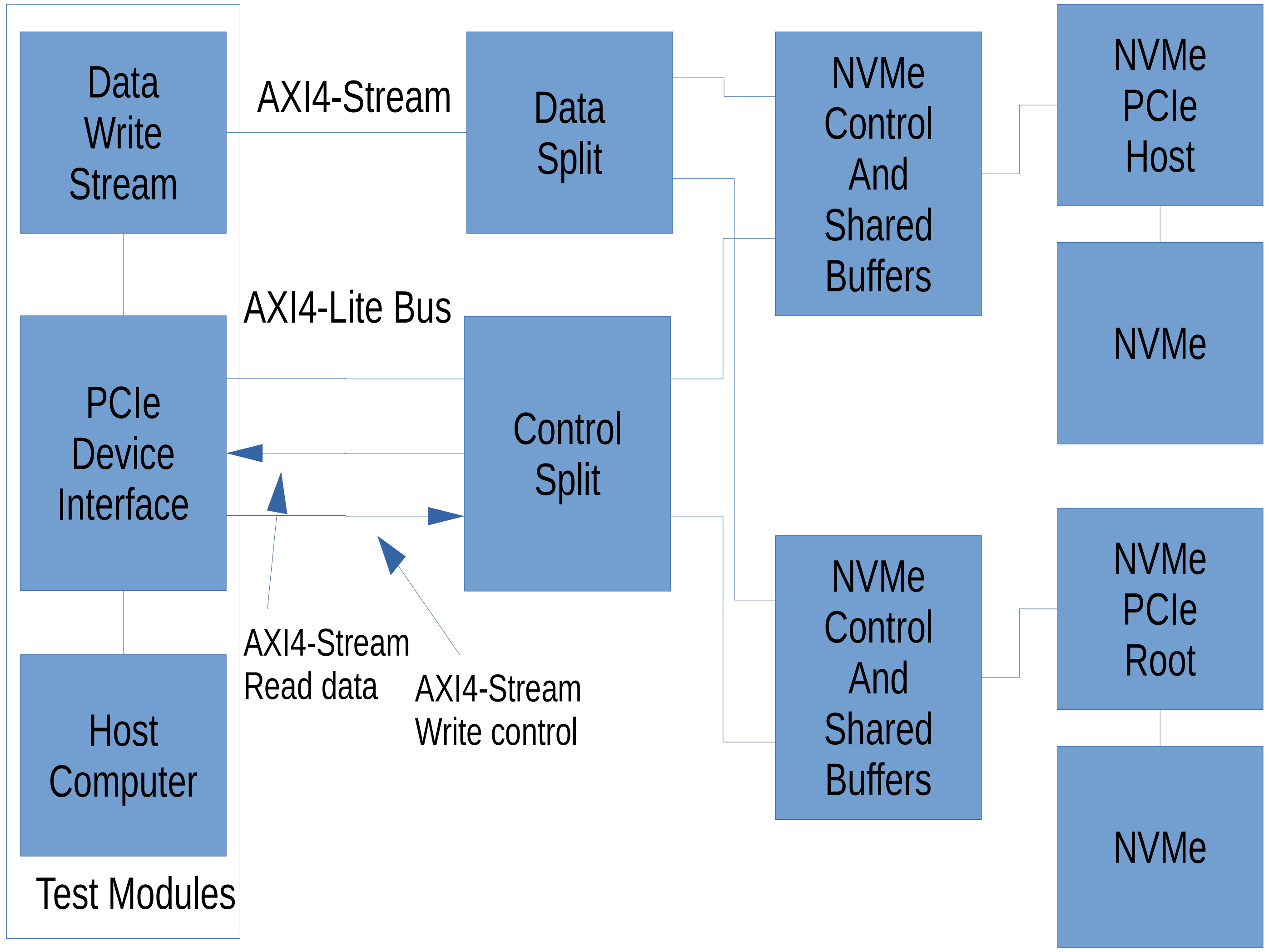
- The Dune DAQ system generates a “super” packet of around 128 kBytes of data (variable due to compression)
- The NvmeStorage system simply stores 4 kByte blocks of data by block number (super block padded to 4k).
- The Dune DAQ provides around 1 second of buffering to handle NVMe latencies (10% of buffer).
- Back-pressure from NVMe will result in packets being dropped by buffer manager.
- Blocks have header/footer to handle lost blocks.

# NvmeStorage: API

- AXI4 Data Stream 256 Bits at 250 MHz.
- AXI4 Lite “bus” for register access by host (may change to IPBus)
- AXI4 data stream (128 bits) for reading data and control replies.
- Optional AXI4 control stream providing host with access to NVMe drives



## NvmeStorage: Internals



## NvmeStorage: Test Rig

- Xilinx KCU105 board Ultrascale
- Kintex XCKU040-2FFVA1156E
- AB17-M2FMC dual NVMe board
- Basic PC
- Fedora 31 as OS
- Using Beam’s bfpga PCIe driver
- RAL, Bristol have duplicate test rigs
- Samsung 970, Seagate FireCuda NVMe under test
- Simple incrementing 32 bit value FPGA data source

# NvmeStorage: NVMe’s

- Have limited endurance ~600 TBytes 512 GByte device
- Can trade off cost v.s. lifetime ( large NVMe devices cost more, but will last longer).
  - 1 x 200 GByte chunk per day – 2 x 512G ~8 Years
  - 1 x 200 GByte chunk per day – 2 x 1T ~16 Years
- Peak write latency is usually not defined in specs. , depends on many factors including NVMe engine and firmware, device structure and age etc.
- We will test at least 3 types of devices, we will run one set of drives to destruction storing metrics (about a week at full data rate).

# NvmeStorage: Status

- Firmware block complete.
  - Waiting for verification by DUNE before final invoice.
- Test firmware available for KCU105 (KU040)
- Ported to K800 in Bristol (KU115)
- Ported to VCU118 (VU9P)
  - Needs testing.
- Will be ported to VMK180 (VM1802) in Oxford.
- Tests under way with different NVMe devices

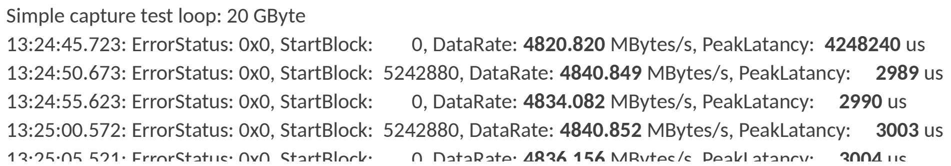
## NvmeStorage: Resource Usage (2 x NVMe)

- One PCIe x4 “hard IP” block per NVMe drive
- Small FPGA resource usage compared to any FELIX FPGA
  - Fractions of KU115 (FPGA on FLX712)
    - CLB LUB 10629  (1.5%)
    - LUTRAM 1896    (0.6%)
    - LUT FF   13776  (1%)
    - BRAM    37        (2.3%)
- (See Erdem’s talk for usage of block formatter)
- Needs optimization
  - Buffers everywhere….
  - Compression BRAM usage seems much too high

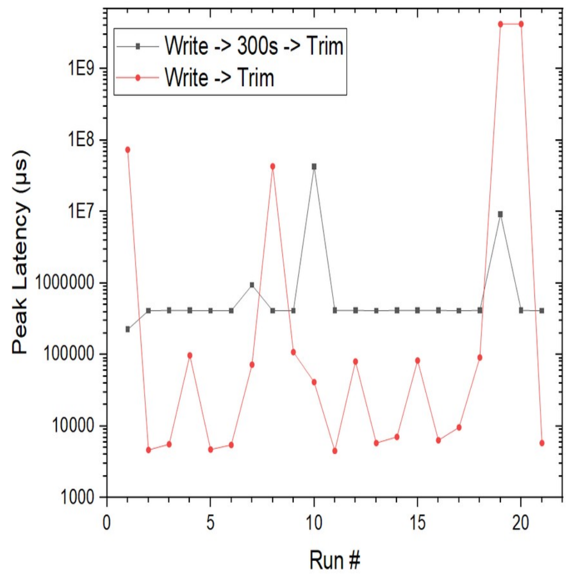
# Test Results – 200GByte with Seagate FireCuda

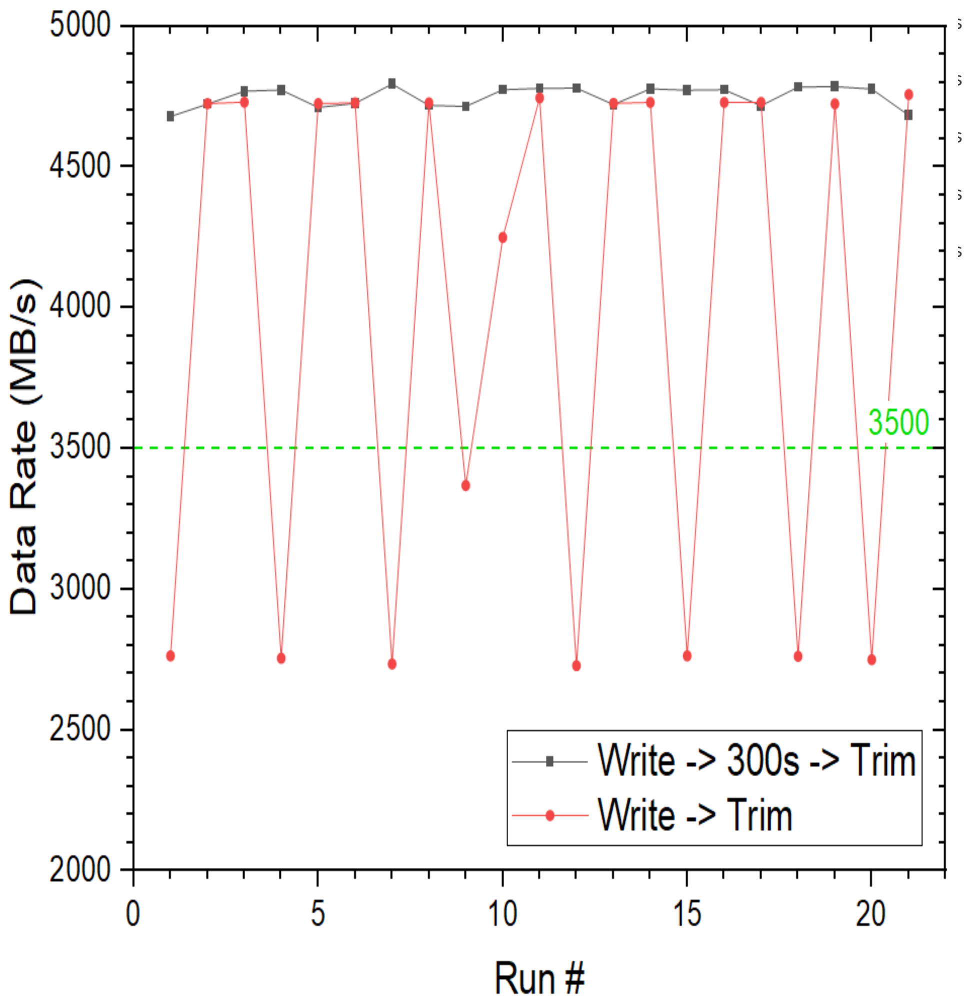
Capture test loop: 200 Gbyte, wait 300s then trim“

- Mean *Write Data Rate*: 4749.30 MB/s
  - Minimum: 4678.82 MB/s
  - Median: 4772.84 MB/s
  - Maximum: 4794.35 MB/s
- Mean *Peak Latency*: 2874922.71 us
- Median *Peak Latency*: 416566 us



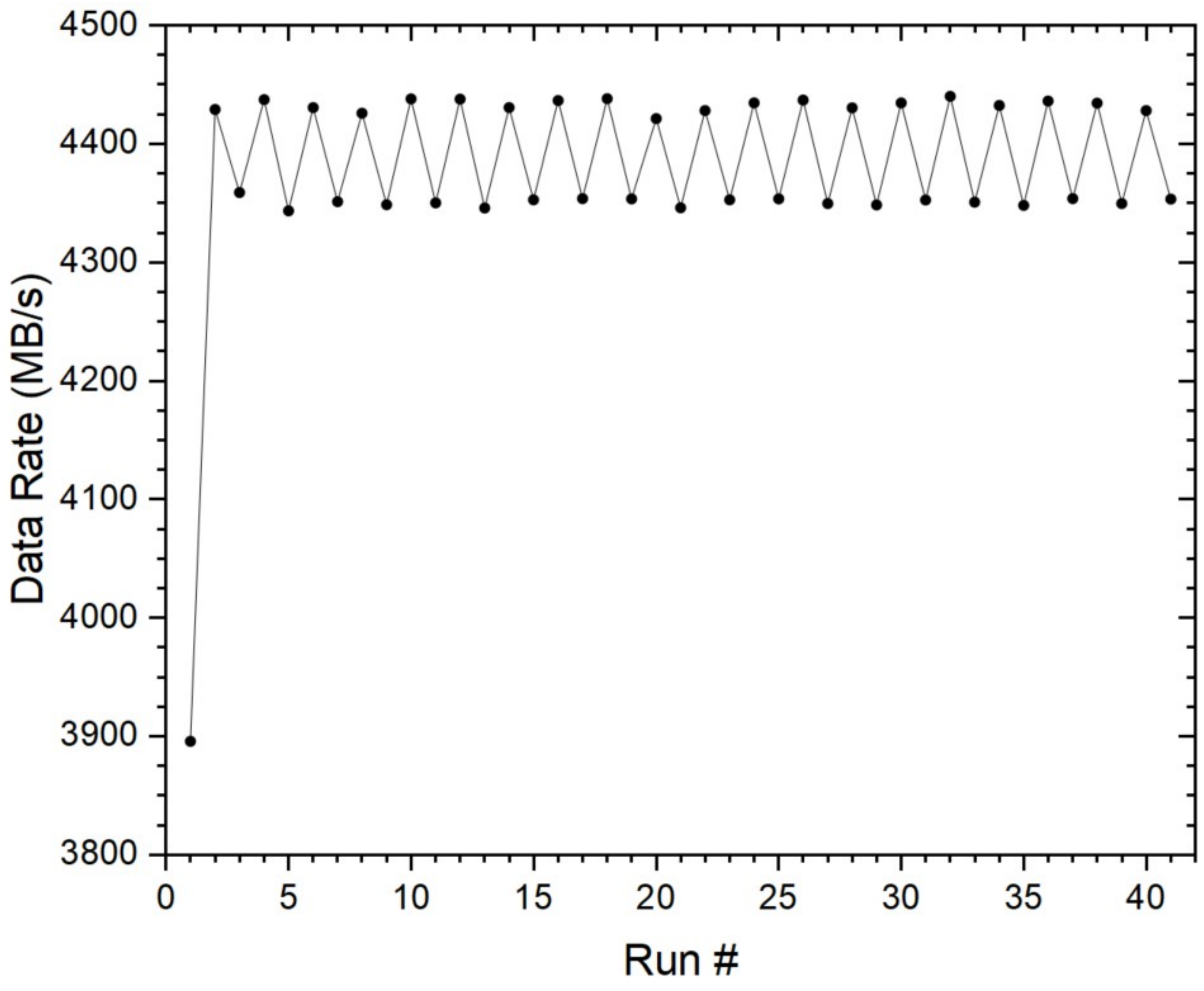
```
Simple capture test loop: 20 GByte
13:24:45.723: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4820.820 MBytes/s, PeakLatancy:  4248240 us
13:24:50.673: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4840.849 MBytes/s, PeakLatancy:     2989 us
13:24:55.623: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4834.082 MBytes/s, PeakLatancy:     2990 us
13:25:00.572: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4840.852 MBytes/s, PeakLatancy:     3003 us
13:25:05.521: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4836.156 MBytes/s, PeakLatancy:     3004 us
```





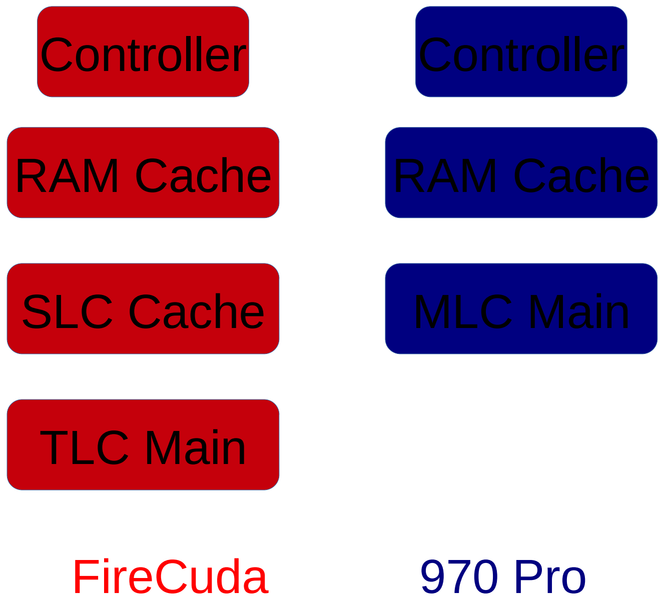
## Test Results – Samsung 970 Pro

- PCIe 3 interface ( c.f. gen 4 on FireCuda)
- Lower peak performance than FireCuda
    - ~4400 MBytes/s
- Much more "stable" – only small variations in transfer rate observed.
- Need more testing to ensure performance doesn't fall with age
    - As blocks start to fail and get mapped out.



## A Note about NVMe Devices

- Have a controller (specialized CPU)
- RAM cache
- Often have an additional cache of fast flash in addition to...
- Main flash memory
- FireCuda devices
  - Phison controller with PCIe 4
  - RAM cache
  - SLC NAND cache
  - TLC NAND
- Samsung 970 Pro devices
  - Controller with PCIe 3
  - RAM cache
  - MLC NAND



## Need / Will deliver:

- Integration with Felix firmware
    - Need AXI4S ↔ Full mode. Announced by Felix/Wupper team (or use adaptation of Simone Ponzio's CR Interface
- Need Wishbone/AIX4L interface. Either use IPBus over Wupper or Felix/Wupper Wishbone interface
- Need API for event fragment request.
- Need API for super Nova trigger request
- Need Felix low level driver ( "Felix Core" )
- Will provide interface to low-level IPBus register access
- Will provide software to keep track of SN data on disk and convert to file on front end server.
- Will provide software to monitor NVMe status ( interface to NVMe status registers )

# Readout Modes

## 1) Trigger requests for windows of few [us] to few seconds

- This is the main mode of operation. Handled by buffer manager (see previous talk).
  - ROI in time by choosing event fragment start/end time
  - ROI in channel number by unpacking on front-end server and selecting channels
    - Does not require decompression, but does require unpacking of memory structure read out from FPGA

## 2) Debug / calibration data streaming

- Debug streaming: keep existing firmware infrastructure in place to read out raw link data.
- Calibration data: Time of calibration events should be known → Can use a event fragment request with special type.

## 3) SNB

- Stream ~ 100s of compressed data to NVMe on receipt of trigger. Read out slowly. (see this talk)

## Known Failure Scenarios

### 1) Low level FE failure

- Data reception will perform check-sum / data integrity checks (see talk on hit finding)
  - Corrupt data not passed to hit-finding pipelines.
  - Corrupt data not passed to buffer manager
  - ProtoDUNE-1 infrastructure remains - corrupt data can be read out for debugging if it can be parsed by interface to “Central Router”.
  - It may be worth adding capture buffers capable of capturing a few WIB frames of data.
  - Data integrity checks only partially implemented so far.

### • 2) Link alignment error

- Currently data reception leaves gaps in data for missing WIB frames. Probably better to discard an entire “super packet” (set of 64 WIB frames) to avoid risk of spurious hits.

### • 3) Uncompressable data

- Will be able to turn off compression on a link-by-link basis (f/ware structure makes this straight forward)

## Features

- Existing:
  - Stand-alone buffer management
  - Stand-alone block formatting
  - Stand-alone NVMe writing.
    - Meets requirements ( 4300 MBytes/s , c.f. ~ 3250 MBytes/s )
- Missing:
  - Integration of buffer management
    - To Felix/Wupper framework
      - Event fragment requests
      - SN triggers
    - To block formatting + NVMe
  - Integration of NVMe interface
    - Start NVMe block for next burst needs to be written before SN trigger.
    - Need software to read contents of two NVMe drives, check for data integrity, build one 400GByte file from 2 x 200GBytes of NVMe blocks.
    - After read, unused blocks (probably) need to be TRIMed to mark as free.
  - Component selection
    - Need NVMe drives suitable for writing 200GByte continuously
  - Endurance testing
    - Need to check that NVMe drives will delivery acceptable performance for lifetime of DUNE

# Adaptability

- Allows flexibility of approach as to where buffer placed (FPGA, server).
  - (Existing “FELIX” infrastructure remains in place.)
- DUNE remains cost constrained →
  - Moving functionality closer to front end give possibility of more APAs per server and lower total hardware cost and power.
    - Only “triggered” data has to move across PCIe. 70GBit/s per APA → 100 Mbit/s (for DUNE, ProtoDUNE more ).
  - PCIe board with NVMe connectors and RAM doesn’t have to populate them if server cost drops more rapidly than expected and DUNE has more funding than expected.
    - i.e. can move to “server centric” if new requirements emerge and funding permits.
    - Adding connectors for NVMe, RAM small cost
      - May be an issue having sufficient MGTs for both ATLAS use (48 input links) and two NVMe interfaces.

## Adaptability

- Can be scaled to 2 ( or 1.5 ) APA per FPGA (dependant on hardware)
- Readout of PDS likely “straight to server”
  - Could implement similar system as for TPC
    - Perform hit finding on zero suppressed data
    - But …… PDS lower bandwidth – single 4.8GBit/s link, cf. ten 9.6Gbit/s

# Adaptability

- The firmware can be ported to other boards (Xilinx FPGA devices) easily
- Works with different DDR4 RAM physical connection schemes (e.g. ZCU102 and KCU105 boards are different)
- Porting to other FPGA devices is possible (e.g Intel/Altera devices)
- Currently there are FPGA devices having Gigabytes of on-chip RAM (Xilinx HBM technology)
- Can support uncompressed data (write/read access speed is adequate to RAM. Add NVMe drive(s) )
- Can support different readout modes (ROI based) either by adding extra filtering in the output selector (no performance penalty) or repacking in software

## Reliability

- Check data integrity at input to firmware
  - Still to be implemented
  - Drop data in controlled manner if downstream “stalls”
- Parts of the firmware can be turned on/off targeting faulty inputs
  - Debugging buffers inside FPGA for data capture and diagnostics.

## Support and Maintenance

- Firmware relies on Felix framework
  - Will be supported for ATLAS
  - Bandwidth to host sufficiently low with buffer-on-FPGA approach that other HEP-supported frameworks with lower bandwidth are possible.
    - e.g. IPBus over PCIe , supported by CMS
- NVMe firmware is commercial but open source.
  - Modifications/enhancements either in-house or purchasing support from BEAM
- NVMe devices are multi-vendor and currently still improving in performance
  - Need to select and evaluate correct device, but out of two devices evaluated one exceeds requirements
  - Multi-vendor preferable to single-vendor lock-in
- Have built a team that can develop and support firmware.
  - Verify all features on PD2 then go to “maintenance mode”

# Resource Requirements

- Firmware relies on Felix framework
  - Will be supported for ATLAS
  - Bandwidth to host sufficiently low with buffer-on-FPGA approach that other HEP-supported frameworks with lower bandwidth are possible.
    - e.g. IPBus over PCIe
- NVMe firmware is commercial but open source.
  - Modifications/enhancements either in-house or purchasing time from BEAM
- NVMe devices are multi-vendor and currently still improving in performance
  - Need to select and evaluate correct device, but out of two devices evaluated one exceeds requirements
  - Multi-vendor preferable to single-vendor lock-in
- Have built a team that can develop and support firmware.

# Summary

- Ability to keep data in FPGA drastically reduces bandwidth to upstream server
  - Flexibility to reduce cost, power if needed.
- Prototypes exist for the components needed for 10s + 100s buffer system.
- NVMe storage a compelling choice for 100s buffer
  - Required performance demonstrated with one (of two) models tested.
- Buffer management firmware (10s, 100s) needs to be integrated with FELIX and Hit-finding blocks
  - Can use evaluation boards for up to 1 APA
    - e.g. XUP3R , Virtex Ultrascale VU9P. Port of “Vanilla Felix”. 12 input links, Up to 512GByte RAM, waiting for release of expansion port → NVMe cable.
- Needs software integration with DAQ
  - Test at PD2

## Backup Slides

# Example NVMe Test Script

```
test3(){
echo "Simple capture test loop: 200 GByte with delayed trim"
./test_nvme -d 2 -s 0 -n 52428800 trim
./test_nvme -nr -d 2 -s 52428800 -n 52428800 trim
# Let NVMe's perform some trimming
sleep 20
while true; do
./test_nvme -nr -d 2 -s 0 -n 52428800 capture
sleep 10
./test_nvme -nr -d 2 -s 52428800 -n 52428800 capture
sleep 10
done
}
```

# Storage Costs

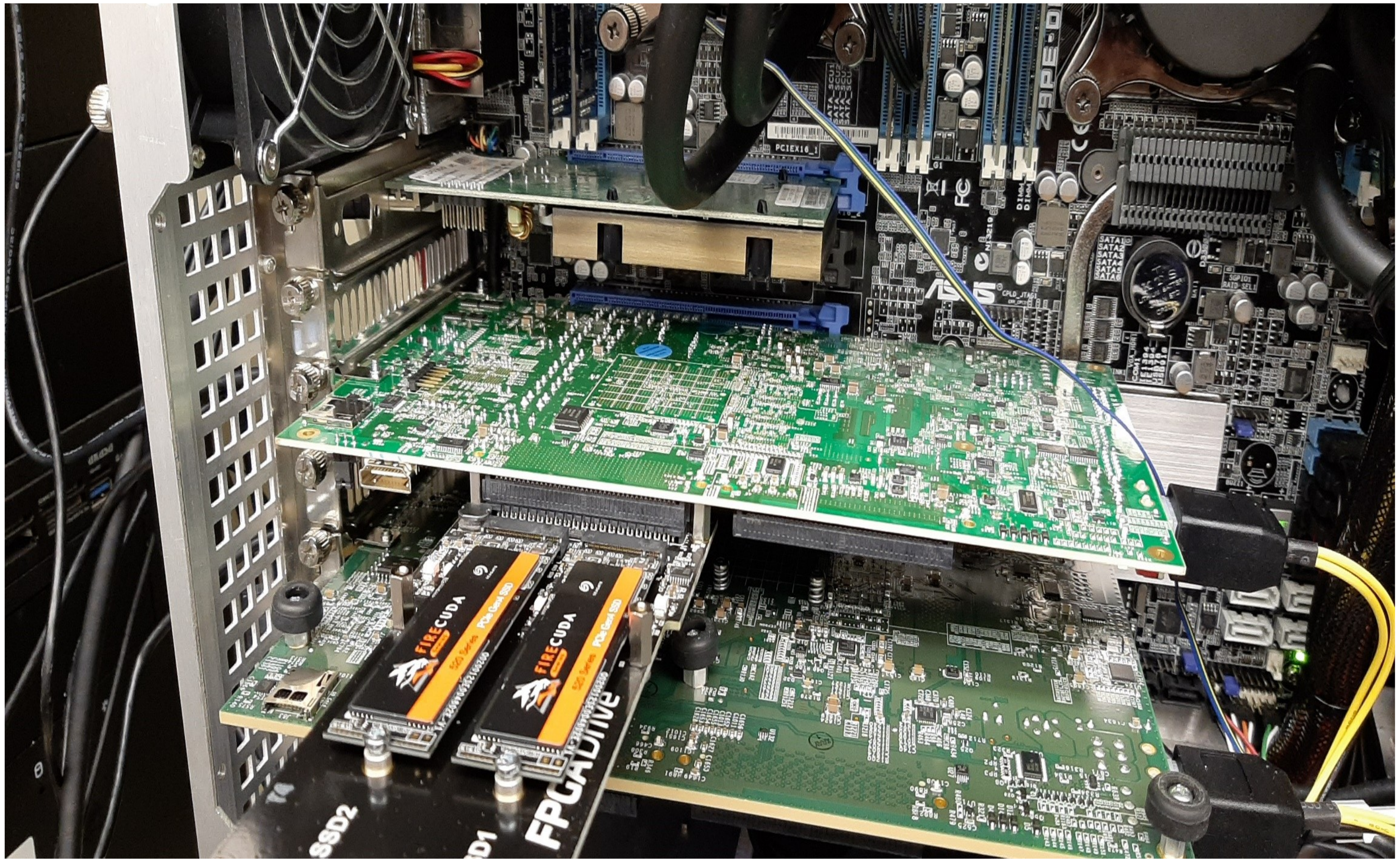
- Taken from https://thememoryguy.com/intels-optane-dimm-price-model/ (2019 data)

| Type | Density | Price | $/GB |
|---|---|---|---|
| DRAM | 32GB | $374.71 | $11.71 |
| DRAM | 64GB | $708.25 | $11.07 |
| DRAM | 128GB | $1,913.21 | $14.95 |
| DRAM | 256GB | $5,952.00 | $23.25 |
| Optane (DIMM) | 128GB | $577.00 | $4.51 |
| Optane (DIMM) | 256GB | $2,125.00 | $8.30 |
| Optane (DIMM) | 512GB | $6,751.00 | $13.19 |
| NVMe (Samsung) | 512GB | $169.99 | $0.33 |
| NVMe (Samsung) | 1TB | $349.94 | $0.35 |

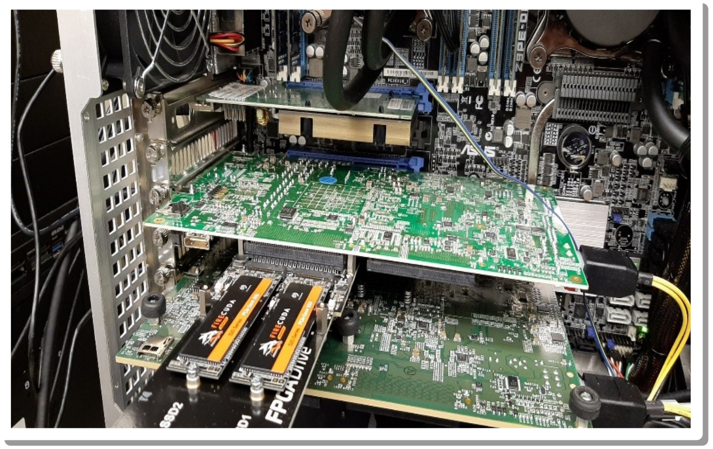
Science & Technology Facilities Council

UK Research and Innovation

# KCU105 and dual NVME



## KCU105 and dual NVME at RAL



***VCU118 below KCU105 …*

- 'Duplicate' of Beam test rig
  - Dual Xeon Server
  - KCU105 Dev Board
  - Opsero Dual NVME Carrier FMC
  - 2 x Firecuda 500GB shown
  - 2 x Samsung 970 Pro 512GB available
  - See Adams slides for results …

## NvmeStorage: Specifications

- Sustained data rate of 4 GBytes/s
- Able to store 2 x 200 GByte data chunks
- Able to handle NVMe latency issues
- Raw FPGA data stream input
- Low FPGA resource usage (No CPU cores etc.)
- Host CPU manages the system.
- Host CPU can read the data whilst writes are in progress.
- Target Xilinx Ultrascale[+] and Vesal.
- Open source. (Apache Licence)

## NvmeStorage: Structure

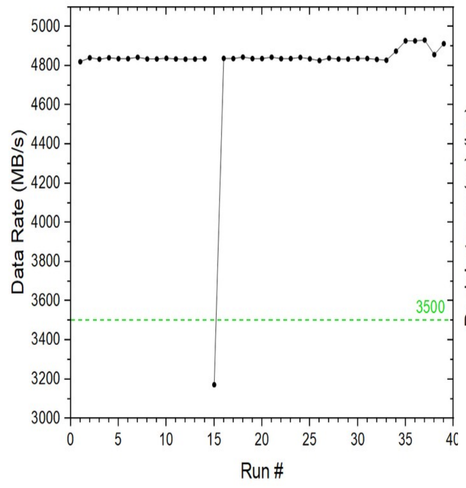
- PCIe Gen3 stable, Gen4 port in progress.
- 4 lane PCIe Gen3 has a peak data rate of around 4 GB/s
- Current commodity NVMe’s have a write data rate of ~2.4 GB/s
- Design uses two NVMe’s working in parallel. Blocks written to alternate drives.
- Uses two Xilinx PCIe hard blocks
- Implemented in FPGA state machines
- Hard coded NVMe parameters for simplicity (Block size, Doorbell stride and size)

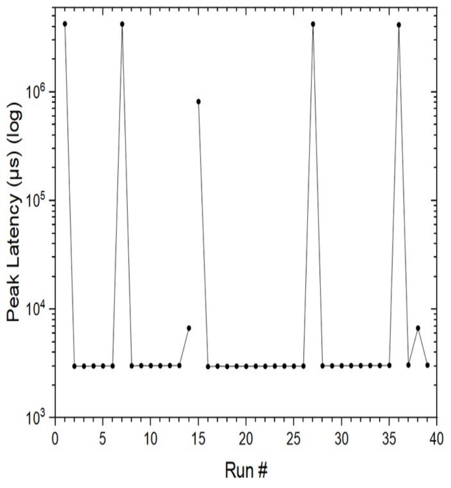
# Test Results – Small Chunks NVIDIA FireCuda

Test 1 - "Simple capture test loop: 20 GByte“

- N = 39
- Mean *Write Data Rate*: 4804.15 MB/s
  - Minimum: 3171.61 MB/s
  - Median: 4836.78 MB/s
  - Maximum: 4931.12 MB/s
- Mean *Peak Latency*: 456188 us
- Median *Peak Latency*: 3020 us
- Difference from mean to median shows instability
- Instability in FireCuda drives also found in testing by BEAM

```
Simple capture test loop: 20 GByte
13:24:45.723: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4820.820 MBytes/s,
PeakLatancy:  4248240 us
13:24:50.673: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4840.849 MBytes/s,
PeakLatancy:     2989 us
13:24:55.623: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4834.082 MBytes/s,
PeakLatancy:     2990 us
13:25:00.572: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4840.852 MBytes/s,
PeakLatancy:     3003 us
13:25:05.521: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4836.156 MBytes/s,
PeakLatancy:     3004 us
13:25:10.471: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4836.292 MBytes/s,
PeakLatancy:     3004 us
13:25:15.420: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4843.474 MBytes/s,
PeakLatancy:  4228370 us
13:25:20.370: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4835.163 MBytes/s,
PeakLatancy:     3013 us
```



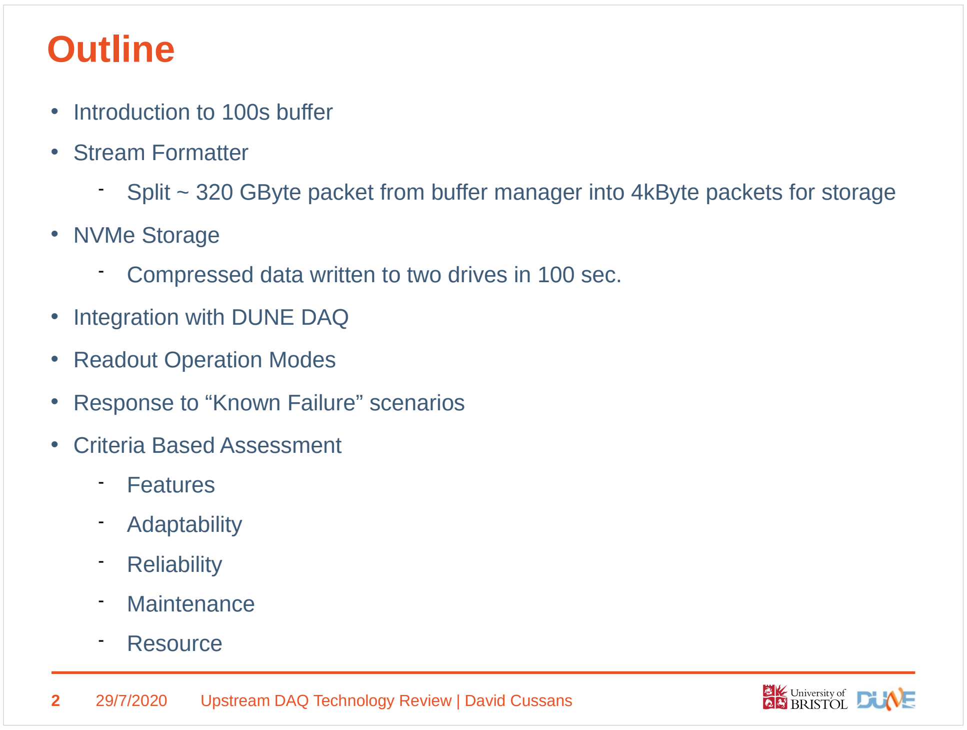


# Upstream DAQ Technology Review

- 100s Buffer – Firmware

- Criteria based assessment.

David Cussans
29/07/2020



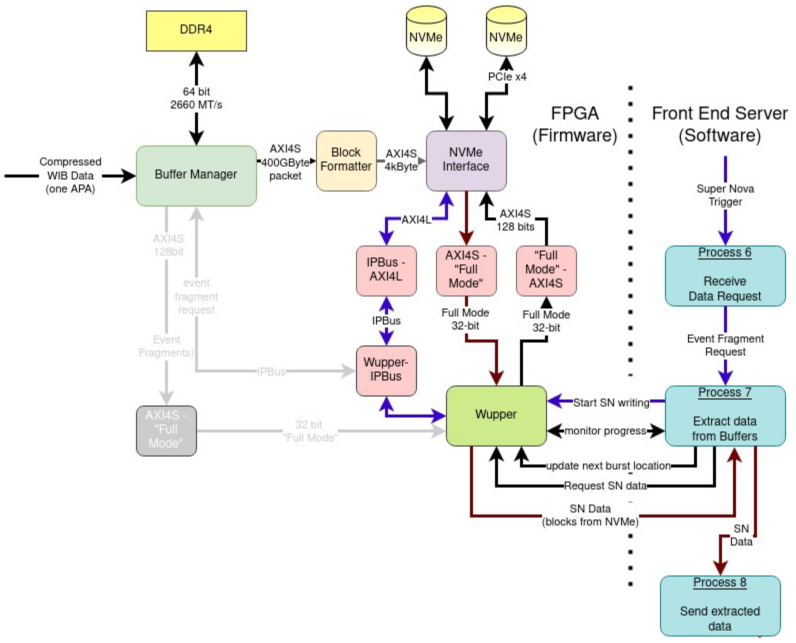
## Outline

- Introduction to 100s buffer
- Stream Formatter
  - Split ~ 320 GByte packet from buffer manager into 4kByte packets for storage
- NVMe Storage
  - Compressed data written to two drives in 100 sec.
- Integration with DUNE DAQ
- Readout Operation Modes
- Response to “Known Failure” scenarios
- Criteria Based Assessment
  - Features
  - Adaptability
  - Reliability
  - Maintenance
  - Resource

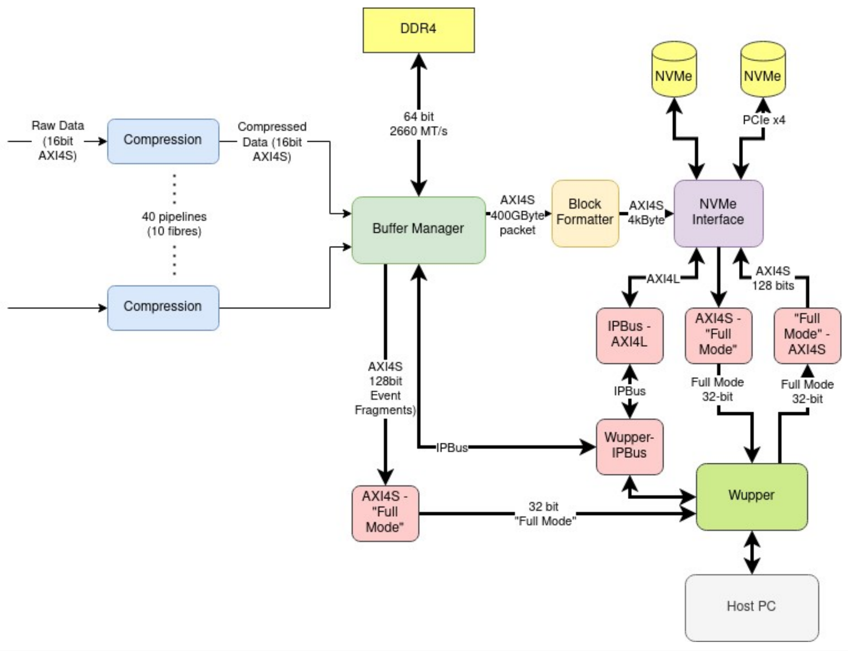
## Bandwidth Requirements (TPC)

- 2590 ADC channels per APA
- 2 Msample/s, 12-bit
- → 60Gbit/s raw data
- ~ 10% overhead for headers etc.
- Factor ~ 2.6 compression (on limited ProtoDUNE study)
- → 3250 MBytes/s data rate to buffer

## Message/Data flow for 100s buffer



## Firmware Part



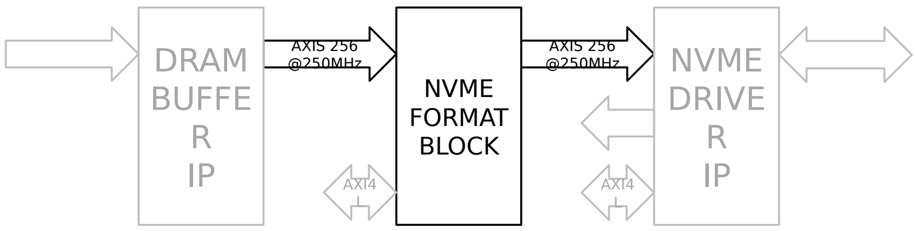
5 2

## Software Part

- At initialization:
  - Set up size of Super Nova capture
    - Write to IPBus register over Wupper interface (using Felix routines)
  - Set start location of Super Nova data in NVMe
    - Write to IPBus register over Wupper
- During run, on receipt of SN trigger:
  - Write to IPBus register over Wupper to initiate Super Nova capture
  - Monitor progress of capture
  - When complete, update start location of SN data in NVMe, ready for next trigger
  - Read SN data from NVMe drives
    - Write to IPBus register, receive AXI4 packet converted to full-mode packet.
  - Combine data from two NVMe drives, strip headers, check CRCs, transmit to back-end DAQ.

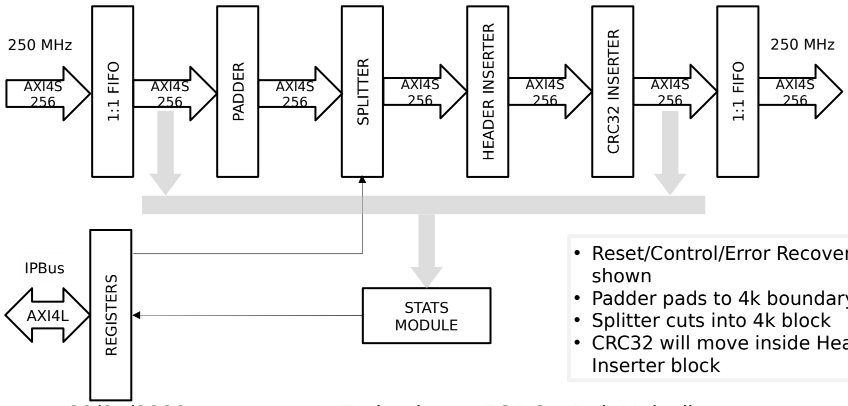
## DUNE Block Formatting Firmware

## NVME Format Block

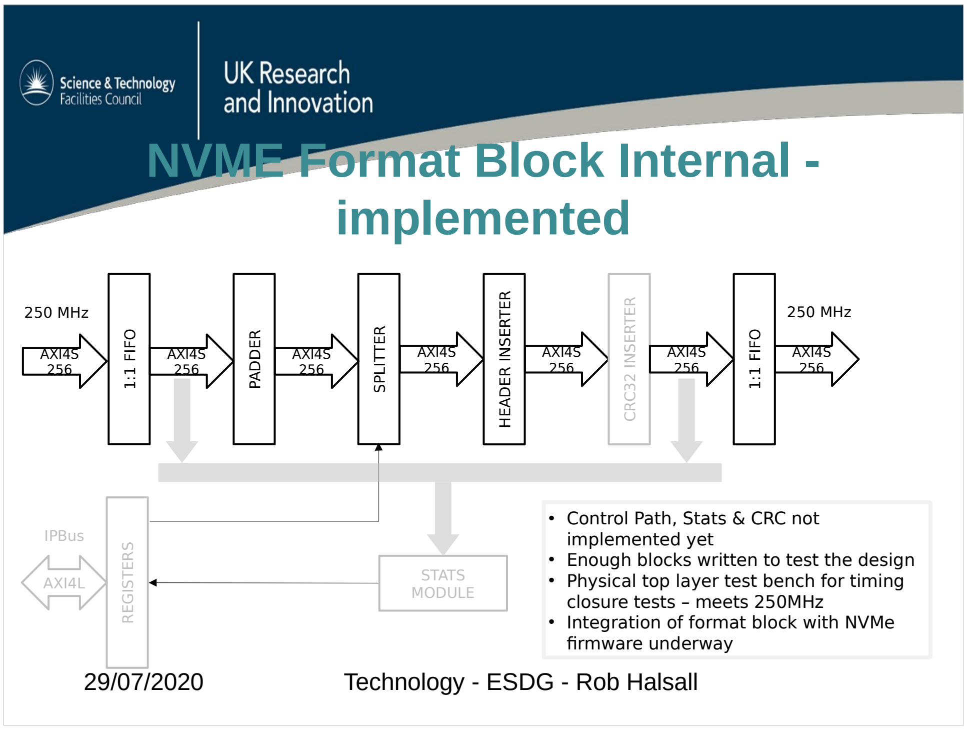


Science & Technology Facilities Council | UK Research and Innovation

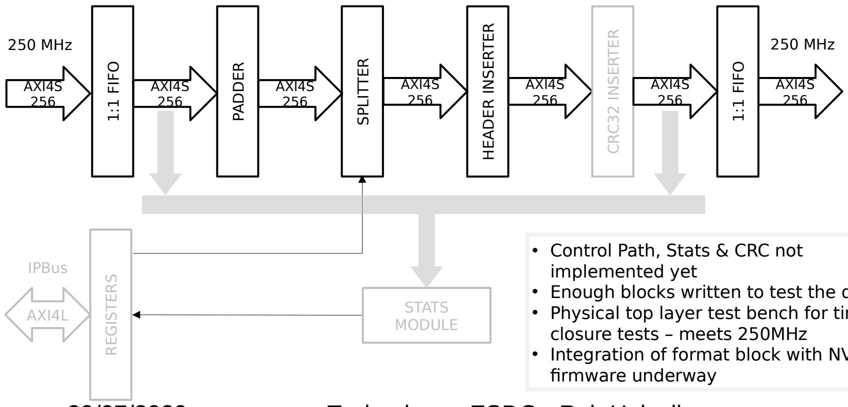
## NVME Format Block Internal



- Reset/Control/Error Recovery not shown
- Padder pads to 4k boundary.
- Splitter cuts into 4k block
- CRC32 will move inside Header Inserter block

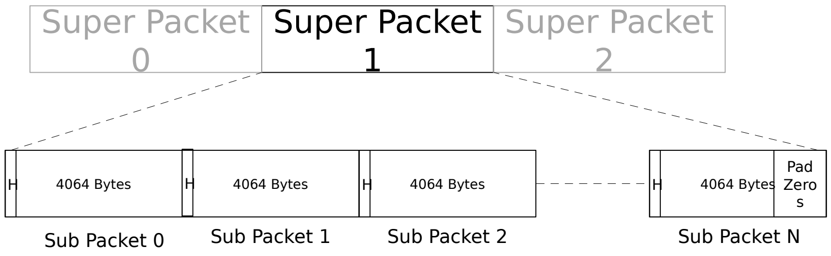


# NVME Format Block Internal - implemented



- Control Path, Stats & CRC not implemented yet
- Enough blocks written to test the design
- Physical top layer test bench for timing closure tests – meets 250MHz
- Integration of format block with NVMe firmware underway

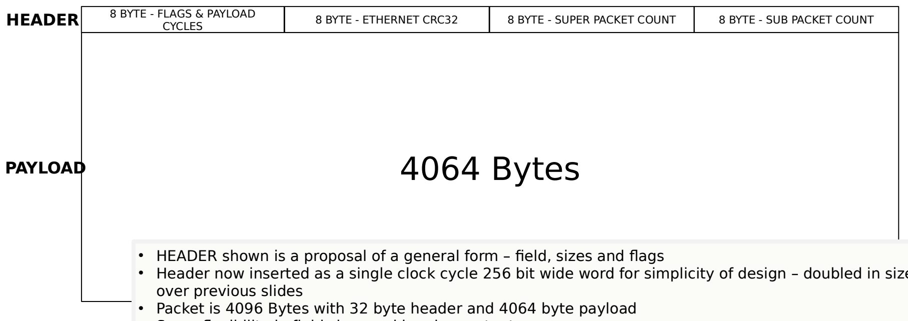
## NVME Super Packet to Sub Packet



- Super Packet (Large) is broken into multiple sub packets which are 4K Bytes and are labelled with a Header (32B) containing super packet no, sub packet number, CRC32, payload cycles and flags.
- The super packet number is constant for all its sub packets and the sub packet number increments fro 0 to N where is the number of sub packets -1 contained within the super packet.
- The last packet may have padding (zeros) added and payload cycles will be lower in size in the header of the last packet to reflect this

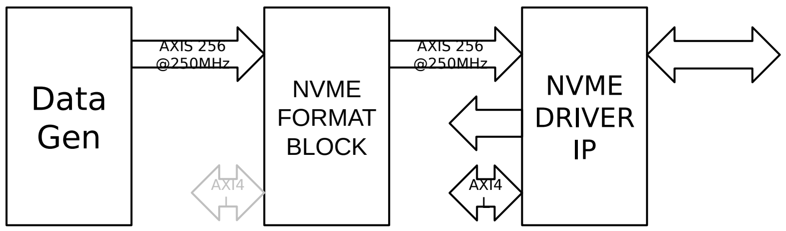
Science & Technology Facilities Council | UK Research and Innovation

## NVME Sub Packet Format


HEADER
8 BYTE - FLAGS & PAYLOAD CYCLES
8 BYTE - ETHERNET CRC32
8 BYTE - SUPER PACKET COUNT
8 BYTE - SUB PACKET COUNT
PAYLOAD
4064 Bytes

- HEADER shown is a proposal of a general form – field, sizes and flags
- Header now inserted as a single clock cycle 256 bit wide word for simplicity of design – doubled in size over previous slides
- Packet is 4096 Bytes with 32 byte header and 4064 byte payload
- Some flexibility in field sizes and header contents
- Uniquely identify packets & detect errors

## NVME + Formatter Block Test Bench

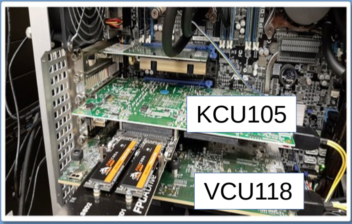


- In Progress

- Insert Data Gen and NVEM Formatter Block into Beam test design
- Use to write our data stream and format into NVME drives via Beam IP
- Added ILAs to AXIS buses in and out of NVME Formatter block
- Design Compiles – debug in progress
- Needs test software to confirm operation

29/07/2020

## Summary

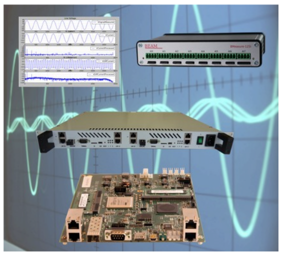


- *Initial version of formatter Block - on STFC Gitlab and CERN Gitlab*
  - https://gitlab.cern.ch/DUNE-SP-TDR-DAQ/nvme-block-formatting
- *Timing closure at 250 MHz standalone*
- *Integration with Beam test design in progress*
- *Use our data gen and formatter as data source*
- *Some minor bugs to resolve in formatter/checker*
- *Add features such as CRC, monitoring, control plane as we move forward and integrate designs*
- Needs integration with firmware build tool (ipbb) and rest of project

## NvmeStorage – Beam Ltd

Terry Barnaby of Beam Ltd

- Beam: Electronics and Software Engineering
- Instrumentation a focus
- Electronics design
- Software design
- Unit/product design
- Scientific systems
- Based in Yate, near Bristol in the UK
- https://portal.beam.ltd.uk/support/dune



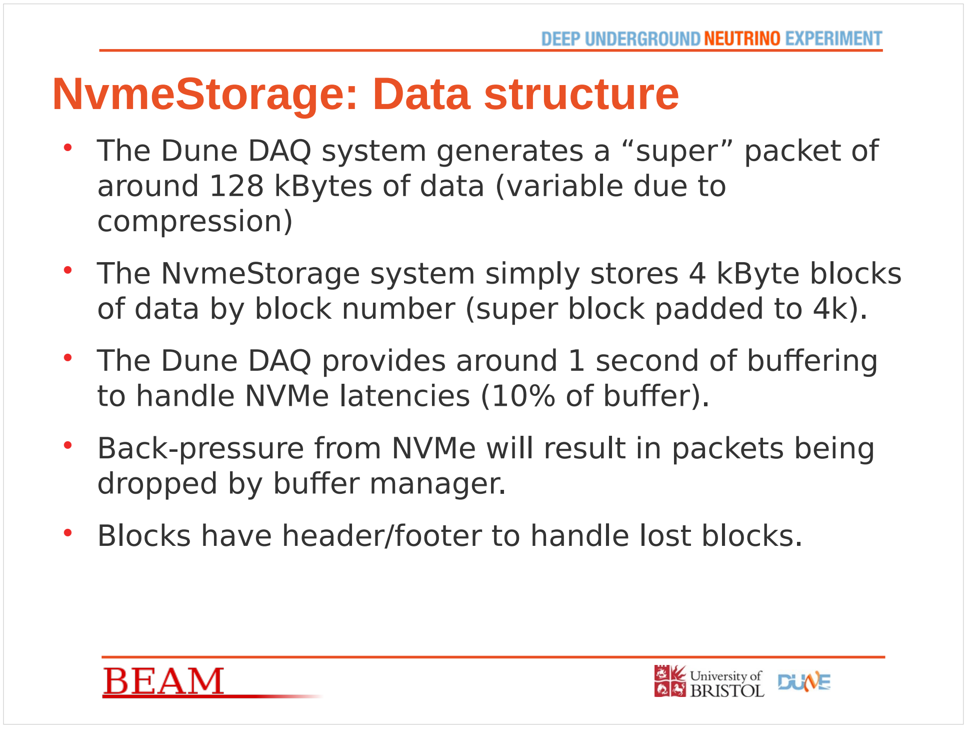
## NVMe Documentation

- Beam: https://portal.beam.ltd.uk/support/dune/
- Git repository (clone URL)
  https://portal.beam.ltd.uk/git/DuneNvme.git
- General documentation is at:
- Doxygen documentation at:
  https://portal.beam.ltd.uk/support/dune/files/doc/DuneNvme/fpga/html/

## NvmeStorage - Introduction

Dune DAQ – NVMe event storage

FPGA module to store 20 – 200 GBytes of event data
4 GBytes/s data rate for 100 seconds
512 Gbyte drives – so two 200 GByte events per drive

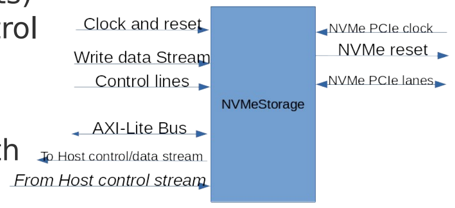


## NvmeStorage: Data structure

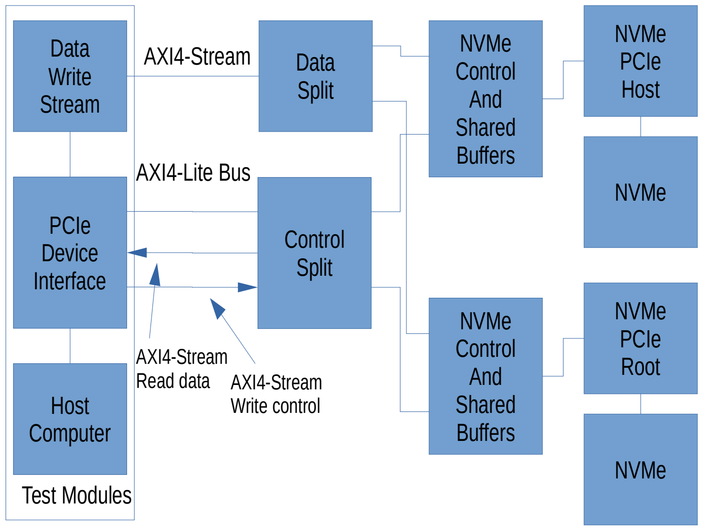
- The Dune DAQ system generates a “super” packet of around 128 kBytes of data (variable due to compression)
- The NvmeStorage system simply stores 4 kByte blocks of data by block number (super block padded to 4k).
- The Dune DAQ provides around 1 second of buffering to handle NVMe latencies (10% of buffer).
- Back-pressure from NVMe will result in packets being dropped by buffer manager.
- Blocks have header/footer to handle lost blocks.

## NvmeStorage: API

- AXI4 Data Stream 256 Bits at 250 MHz.
- AXI4 Lite “bus” for register access by host (may change to IPBus)
- AXI4 data stream (128 bits) for reading data and control replies.
- Optional AXI4 control stream providing host with access to NVMe drives



## NvmeStorage: Internals

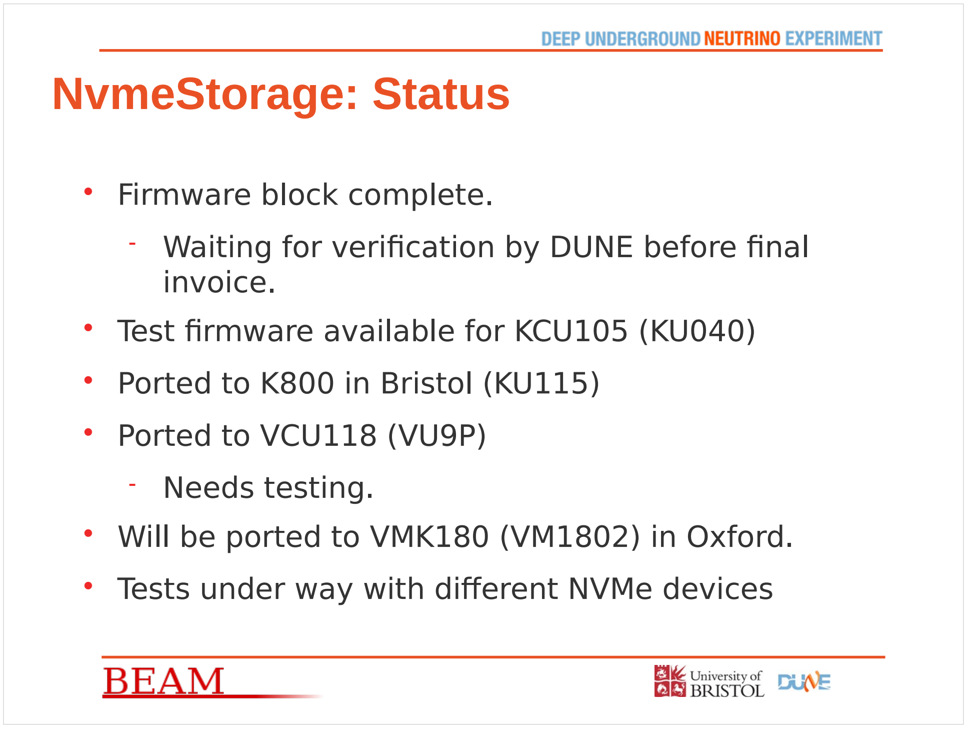


## NvmeStorage: Test Rig

- Xilinx KCU105 board Ultrascale
- Kintex XCKU040-2FFVA1156E
- AB17-M2FMC dual NVMe board
- Basic PC
- Fedora 31 as OS
- Using Beam’s bfpga PCIe driver
- RAL, Bristol have duplicate test rigs
- Samsung 970, Seagate FireCuda NVMe under test
- Simple incrementing 32 bit value FPGA data source

## NvmeStorage: NVMe’s

- Have limited endurance ~600 TBytes 512 GByte device
- Can trade off cost v.s. lifetime ( large NVMe devices cost more, but will last longer).
  - 1 x 200 GByte chunk per day – 2 x 512G ~8 Years
  - 1 x 200 GByte chunk per day – 2 x 1T ~16 Years
- Peak write latency is usually not defined in specs. , depends on many factors including NVMe engine and firmware, device structure and age etc.
- We will test at least 3 types of devices, we will run one set of drives to destruction storing metrics (about a week at full data rate).



## NvmeStorage: Status

- Firmware block complete.
  - Waiting for verification by DUNE before final invoice.
- Test firmware available for KCU105 (KU040)
- Ported to K800 in Bristol (KU115)
- Ported to VCU118 (VU9P)
  - Needs testing.
- Will be ported to VMK180 (VM1802) in Oxford.
- Tests under way with different NVMe devices

## NvmeStorage: Resource Usage (2 x NVMe)

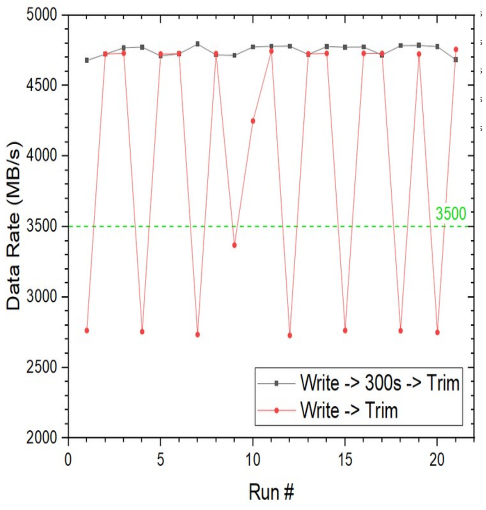
- One PCIe x4 “hard IP” block per NVMe drive
- Small FPGA resource usage compared to any FELIX FPGA
  - Fractions of KU115 (FPGA on FLX712)
    - CLB LUB 10629 (1.5%)
    - LUTRAM 1896 (0.6%)
    - LUT FF 13776 (1%)
    - BRAM 37 (2.3%)
- (See Erdem’s talk for usage of block formatter)
- Needs optimization
  - Buffers everywhere….
  - Compression BRAM usage seems much too high

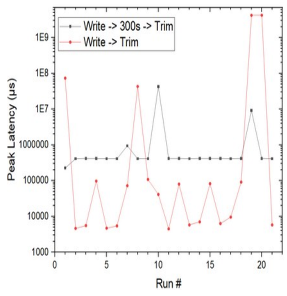
## Test Results – 200GByte with Seagate FireCuda

Capture test loop: 200 Gbyte, wait 300s then trim“

- Mean *Write Data Rate*: 4749.30 MB/s
  - Minimum: 4678.82 MB/s
  - Median: 4772.84 MB/s
  - Maximum: 4794.35 MB/s
- Mean *Peak Latency*: 2874922.71 us
- Median *Peak Latency*: 416566 us

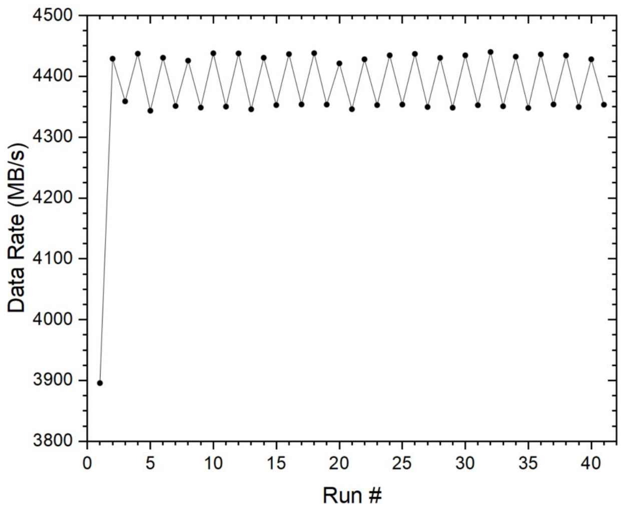
```
Simple capture test loop: 20 GByte
13:24:45.723: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4820.820 MBytes/s, PeakLatancy:  4248240 us
13:24:50.673: ErrorStatus: 0x0, StartBlock: 5242880, DataRate: 4840.849 MBytes/s, PeakLatancy:     2989 us
13:24:55.623: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4834.082 MBytes/s, PeakLatancy:     2990 us
13:25:00.572: ErrorStatus: 0x0, StartBlock: 5242880, DataRate: 4840.852 MBytes/s, PeakLatancy:     3003 us
13:25:05.521: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4836.156 MBytes/s, PeakLatancy:     3004 us
```





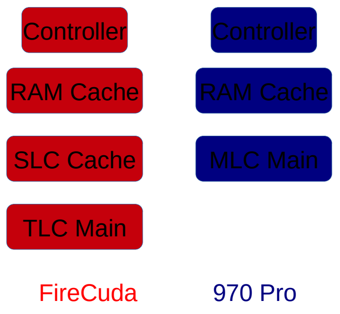
## Test Results – Samsung 970 Pro

- PCIe 3 interface ( c.f. gen 4 on FireCuda)
- Lower peak performance than FireCuda
  - ~4400 MBytes/s
- Much more “stable” – only small variations in transfer rate observed.
- Need more testing to ensure performance doesn’t fall with age
  - As blocks start to fail and get mapped out.



## A Note about NVMe Devices

- Have a controller (specialized CPU)
- RAM cache
- Often have an additional cache of fast flash in addition to...
- Main flash memory
- FireCuda devices
  - Phison controller with PCIe 4
  - RAM cache
  - SLC NAND cache
  - TLC NAND
- Samsung 970 Pro devices
  - Controller with PCIe 3
  - RAM cache
  - MLC NAND



## Need / Will deliver:

- Integration with Felix firmware
    - Need AXI4S ↔ Full mode. Announced by Felix/Wupper team (or use adaptation of Simone Ponzio's CR Interface
- Need Wishbone/AIX4L interface. Either use IPBus over Wupper or Felix/Wupper Wishbone interface
- Need API for event fragment request.
- Need API for super Nova trigger request
- Need Felix low level driver ( "Felix Core" )
- Will provide interface to low-level IPBus register access
- Will provide software to keep track of SN data on disk and convert to file on front end server.
- Will provide software to monitor NVMe status ( interface to NVMe status registers )

## Readout Modes

### 1) Trigger requests for windows of few [us] to few seconds

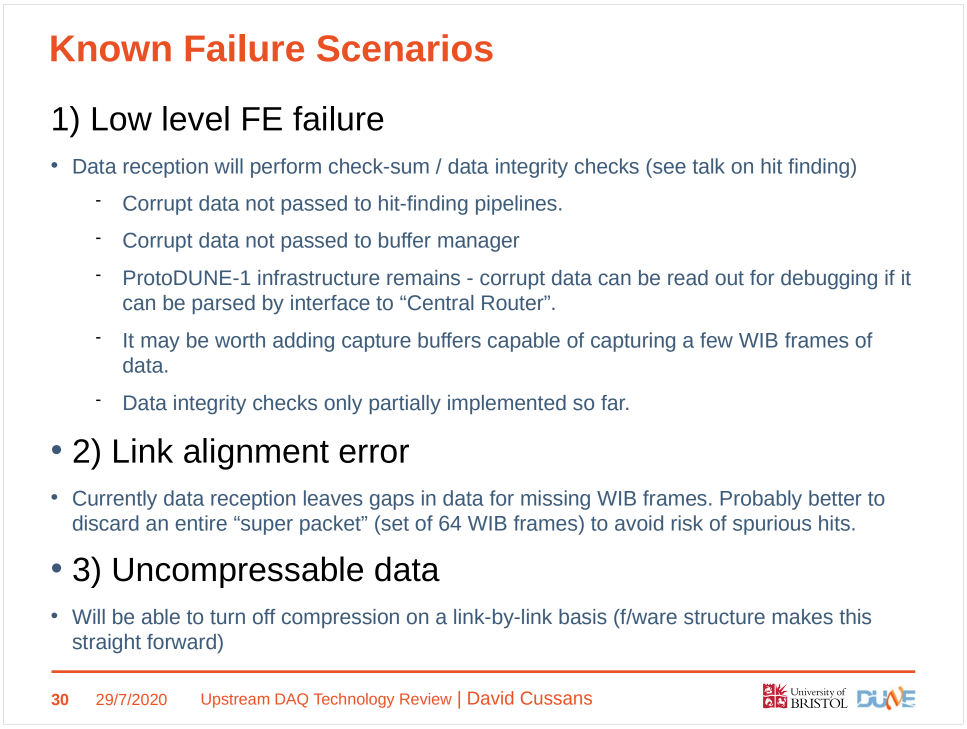
- This is the main mode of operation. Handled by buffer manager (see previous talk).
  - ROI in time by choosing event fragment start/end time
  - ROI in channel number by unpacking on front-end server and selecting channels
    - Does not require decompression, but does require unpacking of memory structure read out from FPGA

### 2) Debug / calibration data streaming

- Debug streaming: keep existing firmware infrastructure in place to read out raw link data.
- Calibration data: Time of calibration events should be known → Can use a event fragment request with special type.

### 3) SNB

- Stream ~ 100s of compressed data to NVMe on receipt of trigger. Read out slowly. (see this talk)



## Known Failure Scenarios

### 1) Low level FE failure

- Data reception will perform check-sum / data integrity checks (see talk on hit finding)
  - Corrupt data not passed to hit-finding pipelines.
  - Corrupt data not passed to buffer manager
  - ProtoDUNE-1 infrastructure remains - corrupt data can be read out for debugging if it can be parsed by interface to “Central Router”.
  - It may be worth adding capture buffers capable of capturing a few WIB frames of data.
  - Data integrity checks only partially implemented so far.

### • 2) Link alignment error

- Currently data reception leaves gaps in data for missing WIB frames. Probably better to discard an entire “super packet” (set of 64 WIB frames) to avoid risk of spurious hits.

### • 3) Uncompressable data

- Will be able to turn off compression on a link-by-link basis (f/ware structure makes this straight forward)

## Features

- Existing:
  - Stand-alone buffer management
  - Stand-alone block formatting
  - Stand-alone NVMe writing.
    - Meets requirements ( 4300 MBytes/s , c.f. ~ 3250 MBytes/s )
- Missing:
  - Integration of buffer management
    - To Felix/Wupper framework
      - Event fragment requests
      - SN triggers
    - To block formatting + NVMe
  - Integration of NVMe interface
    - Start NVMe block for next burst needs to be written before SN trigger.
    - Need software to read contents of two NVMe drives, check for data integrity, build one 400GByte file from 2 x 200GBytes of NVMe blocks.
    - After read, unused blocks (probably) need to be TRIMed to mark as free.
  - Component selection
    - Need NVMe drives suitable for writing 200GByte continuously
  - Endurance testing
    - Need to check that NVMe drives will delivery acceptable performance for lifetime of DUNE

## Adaptability

- Allows flexibility of approach as to where buffer placed (FPGA, server).
  - (Existing “FELIX” infrastructure remains in place.)
- DUNE remains cost constrained →
  - Moving functionality closer to front end give possibility of more APAs per server and lower total hardware cost and power.
    - Only “triggered” data has to move across PCIe. 70GBit/s per APA → 100 Mbit/s (for DUNE, ProtoDUNE more ).
  - PCIe board with NVMe connectors and RAM doesn’t have to populate them if server cost drops more rapidly than expected and DUNE has more funding than expected.
    - i.e. can move to “server centric” if new requirements emerge and funding permits.
    - Adding connectors for NVMe, RAM small cost
      - May be an issue having sufficient MGTs for both ATLAS use (48 input links) and two NVMe interfaces.

## Adaptability

- Can be scaled to 2 ( or 1.5 ) APA per FPGA (dependant on hardware)
- Readout of PDS likely “straight to server”
  - Could implement similar system as for TPC
    - Perform hit finding on zero suppressed data
    - But …… PDS lower bandwidth – single 4.8GBit/s link, cf. ten 9.6Gbit/s

## Adaptability

- The firmware can be ported to other boards (Xilinx FPGA devices) easily
- Works with different DDR4 RAM physical connection schemes (e.g. ZCU102 and KCU105 boards are different)
- Porting to other FPGA devices is possible (e.g Intel/Altera devices)
- Currently there are FPGA devices having Gigabytes of on-chip RAM (Xilinx HBM technology)
- Can support uncompressed data (write/read access speed is adequate to RAM. Add NVMe drive(s) )
- Can support different readout modes (ROI based) either by adding extra filtering in the output selector (no performance penalty) or repacking in software

## Reliability

- Check data integrity at input to firmware
  - Still to be implemented
  - Drop data in controlled manner if downstream “stalls”
- Parts of the firmware can be turned on/off targeting faulty inputs
  - Debugging buffers inside FPGA for data capture and diagnostics.

## Support and Maintenance

- Firmware relies on Felix framework
  - Will be supported for ATLAS
  - Bandwidth to host sufficiently low with buffer-on-FPGA approach that other HEP-supported frameworks with lower bandwidth are possible.
    - e.g. IPBus over PCIe , supported by CMS
- NVMe firmware is commercial but open source.
  - Modifications/enhancements either in-house or purchasing support from BEAM
- NVMe devices are multi-vendor and currently still improving in performance
  - Need to select and evaluate correct device, but out of two devices evaluated one exceeds requirements
  - Multi-vendor preferable to single-vendor lock-in
- Have built a team that can develop and support firmware.
  - Verify all features on PD2 then go to “maintenance mode”

## Resource Requirements

- Firmware relies on Felix framework
  - Will be supported for ATLAS
  - Bandwidth to host sufficiently low with buffer-on-FPGA approach that other HEP-supported frameworks with lower bandwidth are possible.
    - e.g. IPBus over PCIe
- NVMe firmware is commercial but open source.
  - Modifications/enhancements either in-house or purchasing time from BEAM
- NVMe devices are multi-vendor and currently still improving in performance
  - Need to select and evaluate correct device, but out of two devices evaluated one exceeds requirements
  - Multi-vendor preferable to single-vendor lock-in
- Have built a team that can develop and support firmware.

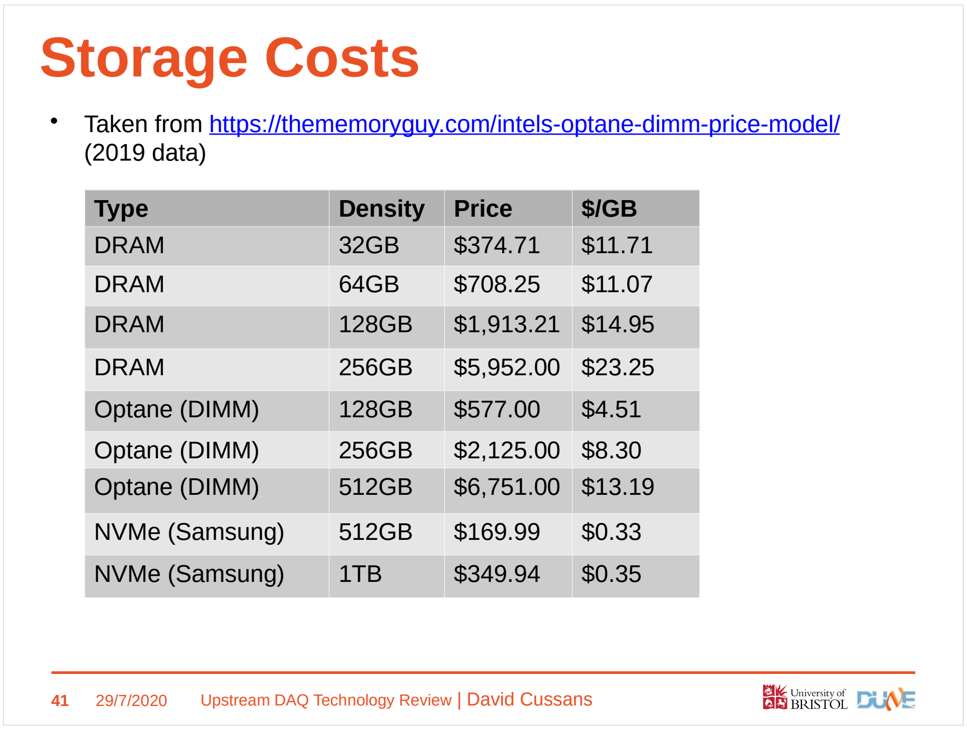
## Summary

- Ability to keep data in FPGA drastically reduces bandwidth to upstream server
  - Flexibility to reduce cost, power if needed.
- Prototypes exist for the components needed for 10s + 100s buffer system.
- NVMe storage a compelling choice for 100s buffer
  - Required performance demonstrated with one (of two) models tested.
- Buffer management firmware (10s, 100s) needs to be integrated with FELIX and Hit-finding blocks
  - Can use evaluation boards for up to 1 APA
    - e.g. XUP3R , Virtex Ultrascale VU9P. Port of “Vanilla Felix”. 12 input links, Up to 512GByte RAM, waiting for release of expansion port → NVMe cable.
- Needs software integration with DAQ
  - Test at PD2

## Backup Slides

## Example NVMe Test Script

```
test3(){
echo "Simple capture test loop: 200 GByte with delayed trim"
./test_nvme -d 2 -s 0 -n 52428800 trim
./test_nvme -nr -d 2 -s 52428800 -n 52428800 trim
# Let NVMe's perform some trimming
sleep 20
while true; do
./test_nvme -nr -d 2 -s 0 -n 52428800 capture
sleep 10
./test_nvme -nr -d 2 -s 52428800 -n 52428800 capture
sleep 10
done
}
```



## Storage Costs

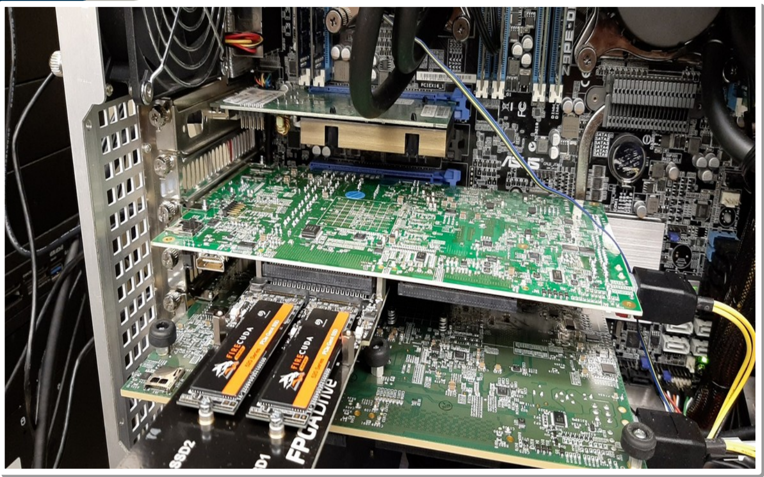
- Taken from https://thememoryguy.com/intels-optane-dimm-price-model/ (2019 data)

| Type | Density | Price | $/GB |
|---|---|---|---|
| DRAM | 32GB | $374.71 | $11.71 |
| DRAM | 64GB | $708.25 | $11.07 |
| DRAM | 128GB | $1,913.21 | $14.95 |
| DRAM | 256GB | $5,952.00 | $23.25 |
| Optane (DIMM) | 128GB | $577.00 | $4.51 |
| Optane (DIMM) | 256GB | $2,125.00 | $8.30 |
| Optane (DIMM) | 512GB | $6,751.00 | $13.19 |
| NVMe (Samsung) | 512GB | $169.99 | $0.33 |
| NVMe (Samsung) | 1TB | $349.94 | $0.35 |

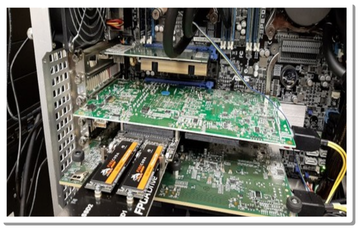
Science & Technology Facilities Council

UK Research and Innovation

## KCU105 and dual NVME

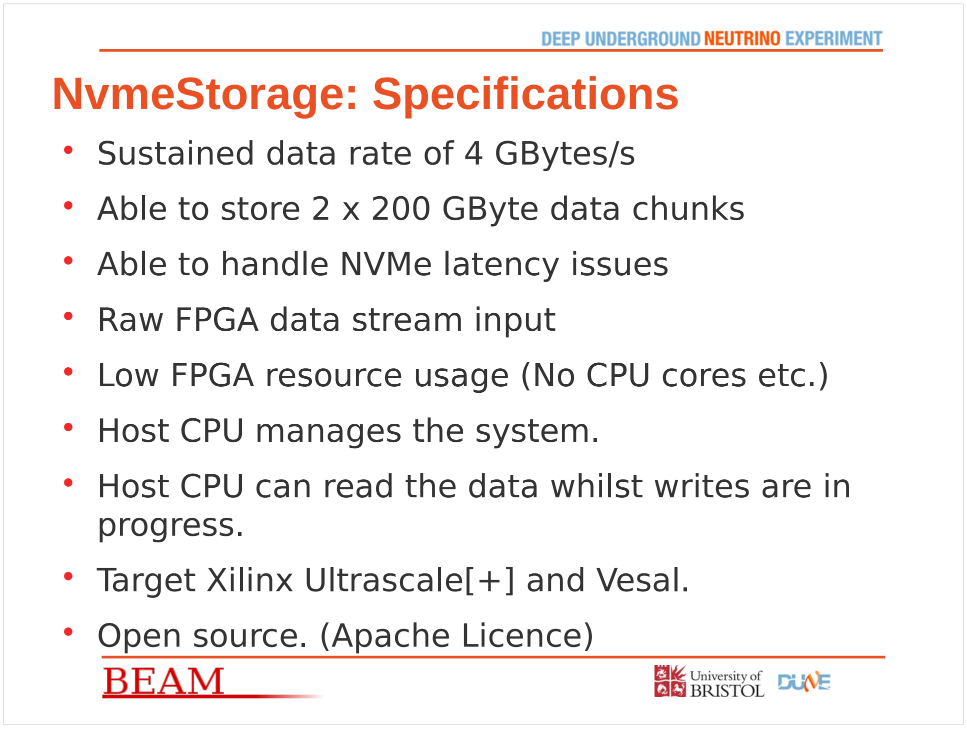


## KCU105 and dual NVME at RAL



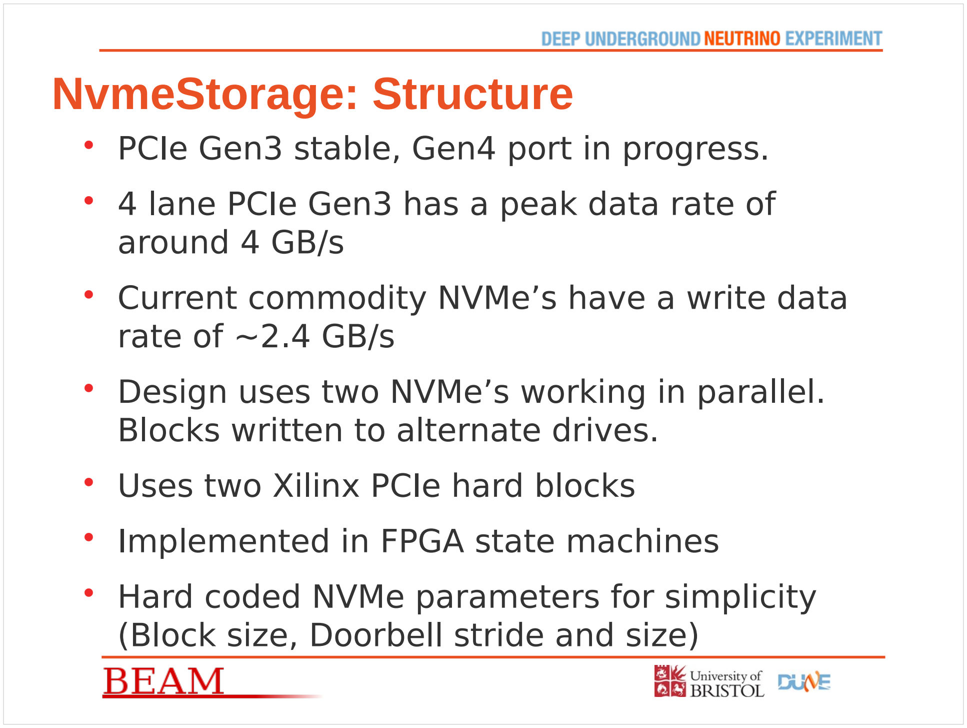
***VCU118 below KCU105 …*

- ‘Duplicate’ of Beam test rig
  - Dual Xeon Server
  - KCU105 Dev Board
  - Opsero Dual NVME Carrier FMC
  - 2 x Firecuda 500GB shown
  - 2 x Samsung 970 Pro 512GB available
  - See Adams slides for results …



## NvmeStorage: Specifications

- Sustained data rate of 4 GBytes/s
- Able to store 2 x 200 GByte data chunks
- Able to handle NVMe latency issues
- Raw FPGA data stream input
- Low FPGA resource usage (No CPU cores etc.)
- Host CPU manages the system.
- Host CPU can read the data whilst writes are in progress.
- Target Xilinx Ultrascale[+] and Vesal.
- Open source. (Apache Licence)



## NvmeStorage: Structure

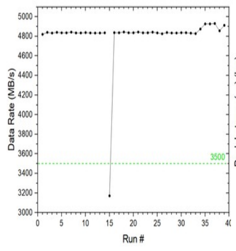
- PCIe Gen3 stable, Gen4 port in progress.
- 4 lane PCIe Gen3 has a peak data rate of around 4 GB/s
- Current commodity NVMe's have a write data rate of ~2.4 GB/s
- Design uses two NVMe's working in parallel. Blocks written to alternate drives.
- Uses two Xilinx PCIe hard blocks
- Implemented in FPGA state machines
- Hard coded NVMe parameters for simplicity (Block size, Doorbell stride and size)

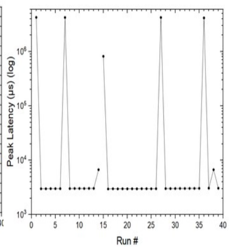
## Test Results – Small Chunks NVIDIA FireCuda

Test 1 - "Simple capture test loop: 20 GByte“

- N = 39
- Mean *Write Data Rate*: 4804.15 MB/s
  - Minimum: 3171.61 MB/s
  - Median: 4836.78 MB/s
  - Maximum: 4931.12 MB/s
- Mean *Peak Latency*: 456188 us
- Median *Peak Latency*: 3020 us
- Difference from mean to median shows instability
- Instability in FireCuda drives also found in testing by BEAM

```
Simple capture test loop: 20 GByte
13:24:45.723: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4820.820 MBytes/s,
PeakLatancy:  4248240 us
13:24:50.673: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4840.849 MBytes/s,
PeakLatancy:     2989 us
13:24:55.623: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4834.082 MBytes/s,
PeakLatancy:     2990 us
13:25:00.572: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4840.852 MBytes/s,
PeakLatancy:     3003 us
13:25:05.521: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4836.156 MBytes/s,
PeakLatancy:     3004 us
13:25:10.471: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4836.292 MBytes/s,
PeakLatancy:     3004 us
13:25:15.420: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4843.474 MBytes/s,
PeakLatancy:  4228370 us
13:25:20.370: ErrorStatus: 0x0, StartBlock:  5242880, DataRate: 4835.163 MBytes/s,
PeakLatancy:     3013 us
```


Data Rate (MB/s)
Run #


Peak Latency (µs) (log)
$10^6$
$10^5$
$10^4$
$10^3$
Run #