University of BRISTOL

DUNE

Bristol NVMe Firmware

ADAM GILLARD

ag17009@bristol.ac.uk

HiTech Global K800

NVMe firmware port to HiTech Global K800 initial testing complete

- Error in reading *Peak Latency* values in test software are suspected to be due to a clock domain crossing error
- This is further backed up by 14 critical warnings in Vivado CDC Report
- “Bug Report” has been written up and sent to BEAM

Build files can be found at https://github.com/DUNE/pl-nvme/tree/HTG-K800

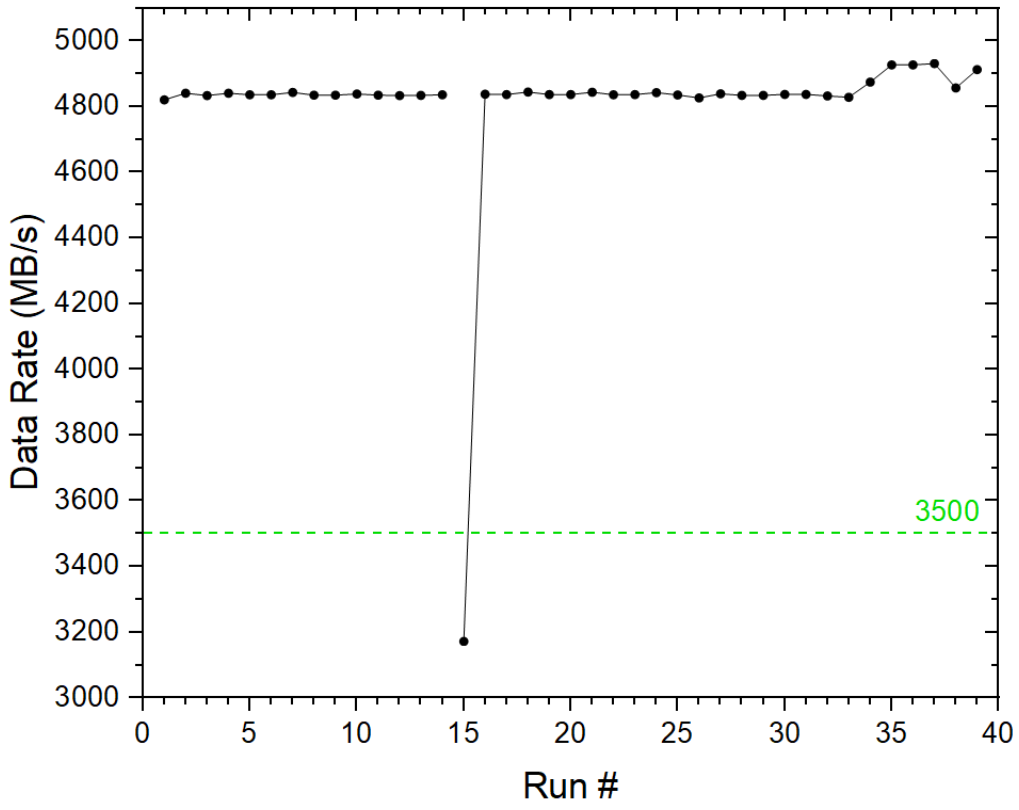
- Includes build instructions
- *Full step-by-step instructions from KCU105 to HTG-K800 can be provided upon request*

Received test data from Rob Hallsal (RAL) using KCU105 + Samsung 970 Evo Drives

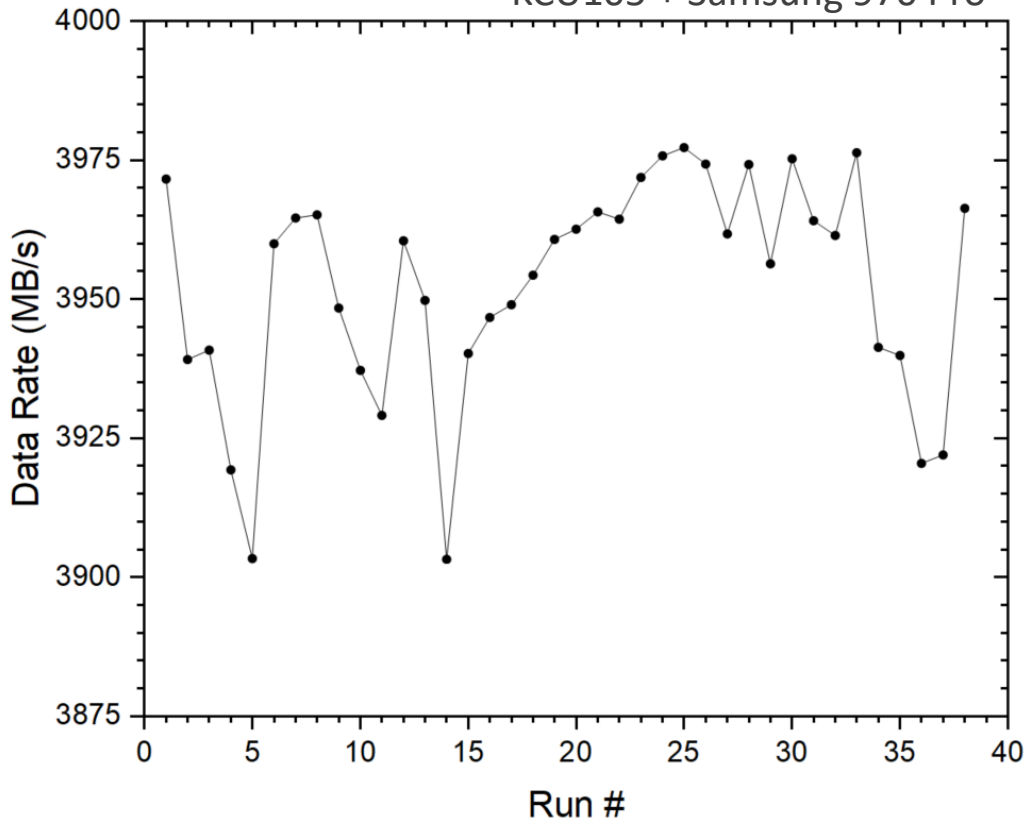
Testing Data – 20GB Write

Written to 4K Blocks

K800 + Seagate FireCuda 520



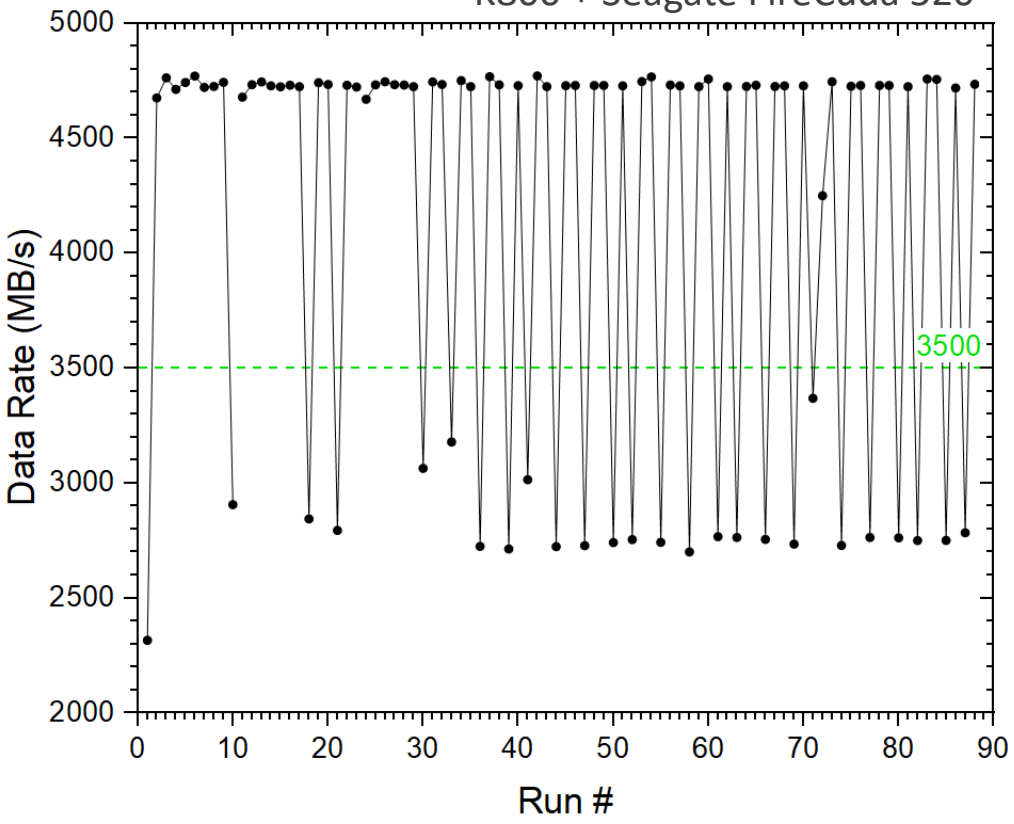
KCU105 + Samsung 970 Pro



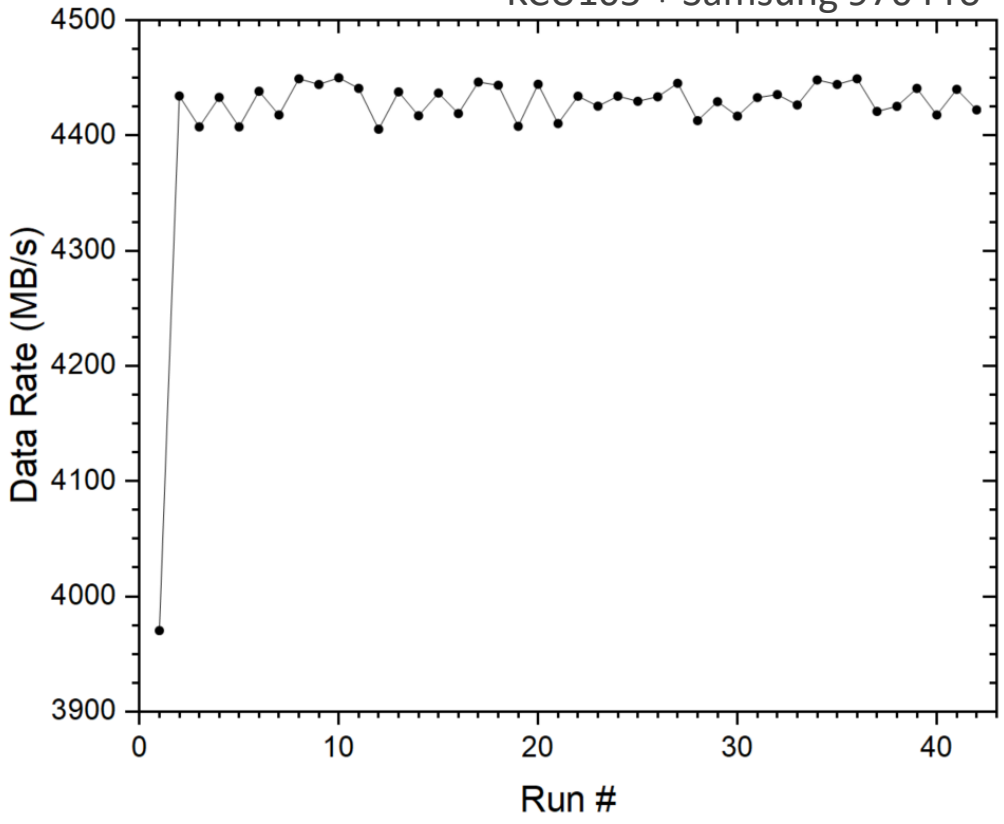
Testing Data – 200GB Write

Written to 4K Blocks

K800 + Seagate FireCuda 520



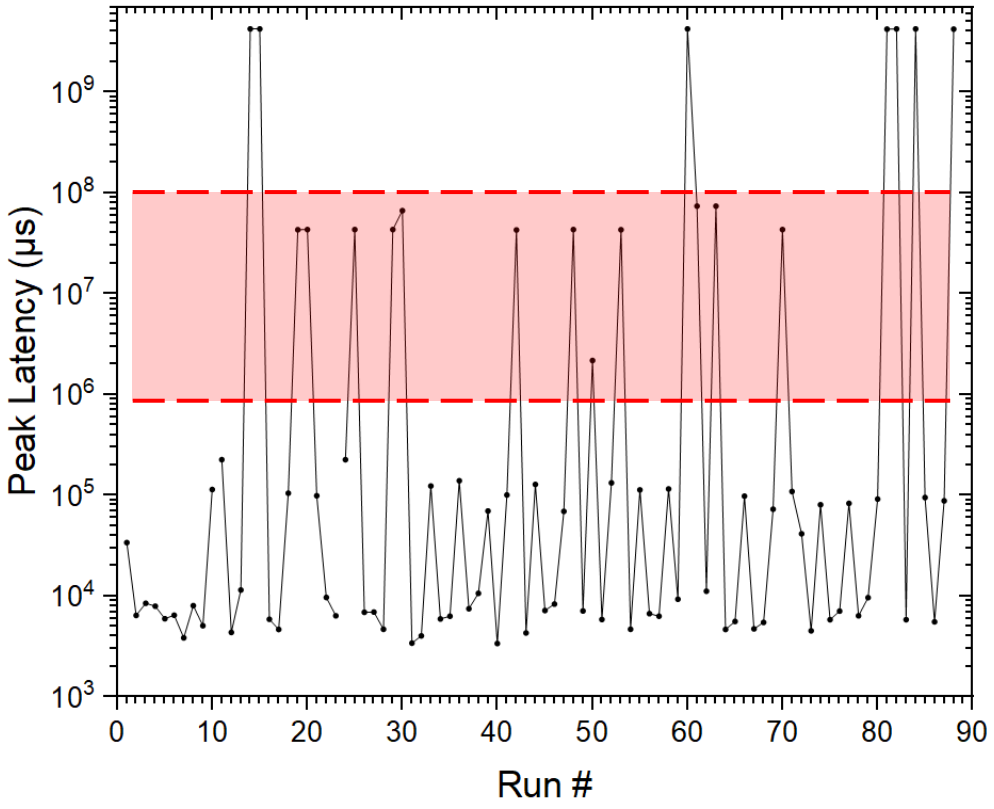
KCU105 + Samsung 970 Pro



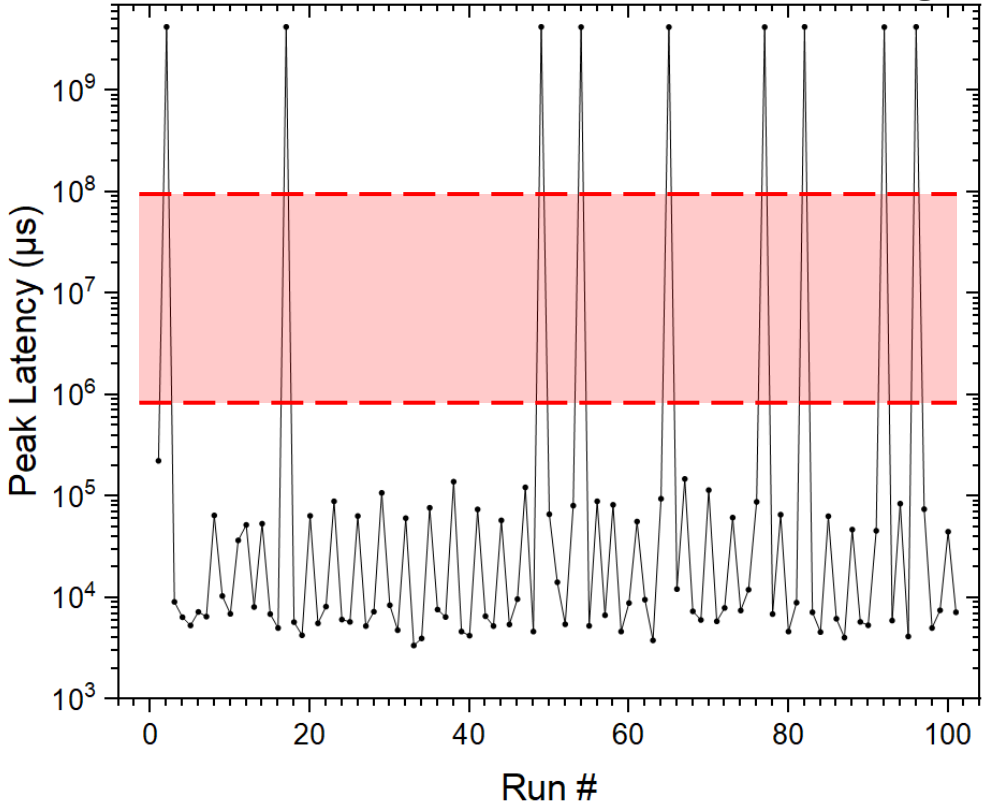
Testing Data – Peak Latency (200GB)

Written to 4K Blocks

Original



Post Software & Firmware Changes



NVMe Readout

What do we need to do?

1. Take data (already in 4kByte blocks) – using incrementing counter at this stage
2. Write data to drives in parallel
3. Read data out to host PC
    1. Confirm all blocks written to drives
    2. Confirm all blocks have 4k Bytes
    3. Confirm data is written in correct sequence
4. Stitch blocks back together in the correct order to give a single data-stream
    1. Requires special treatment for missing blocks
5. Check the data-stream is correct (no missing blocks)

NVMe Readout

Initial sanity check

- Use BEAM test software to read 2 blocks to binary output file (.bin)
- Very primitive Python script to convert output to list of Bytes in integer format
- Output consisted of 8192 elements (Bytes) as expected
- Counter uses 4 Bytes: each of values 0 to 255

```python
f=open("testoutput.bin", "rb")
num=list(f.read())
print(num)
f.close()
```

Python

and so on…

NVMe Readout

Written simple Python script which piggybacks off of Beam *test_nvme* software

1. Uses *test_nvme* to write 20GB data
2. Uses *test_nvme* to read a specified number of 4kByte blocks to .bin output file
3. Interpret binary data to list of bytes in integer format
4. Checks the list contains expected number of Bytes
5. Compiles Bytes into a single data-stream string

```python
#import numpy as np
import subprocess


def readFile(filename):
    """ Read a binary file and output elements as a list
    """
    f = open(filename, "rb")
    num = list(f.read()) # List of all elements
    f.close()

    return num

def dataStream(elementList, numBlocksRead):
    """ Takes in list of binary output and compiles to single datastream
    """
    elementStream = ''.join(map(str, elementList))
    if numBlocksRead <= 2:
        print(elementStream)

    return elementStream

def bashProcess(command):
    """ Run a bash command to terminal
    """
    subprocess.run(command, shell=True)
    print("Command completed")

    return

def lenCheck(elementList):
    #if numBlocksRead <= 2:
    #    print(elementList)

    if len(elementList) == 4096 * numBlocksRead:
        print("No blocks missing. {} kBytes read.".format(len(elementList)/1024))
    else:
        print("Blocks missing")

    return


filename = "output.bin"
numBlocksRead = 2
command = """./test_nvme -d 2 -s 0 -n 5242880 capture
            ./test_nvme -d 2 -s 0 -n {0} -o {1} read""".format(numBlocksRead, filename)

bashProcess(command)
elementList = readFile(filename)
lenCheck(elementList)

elementStream = dataStream(elementList, numBlocksRead)
```

NVMe Readout

Example output for 10 block readout

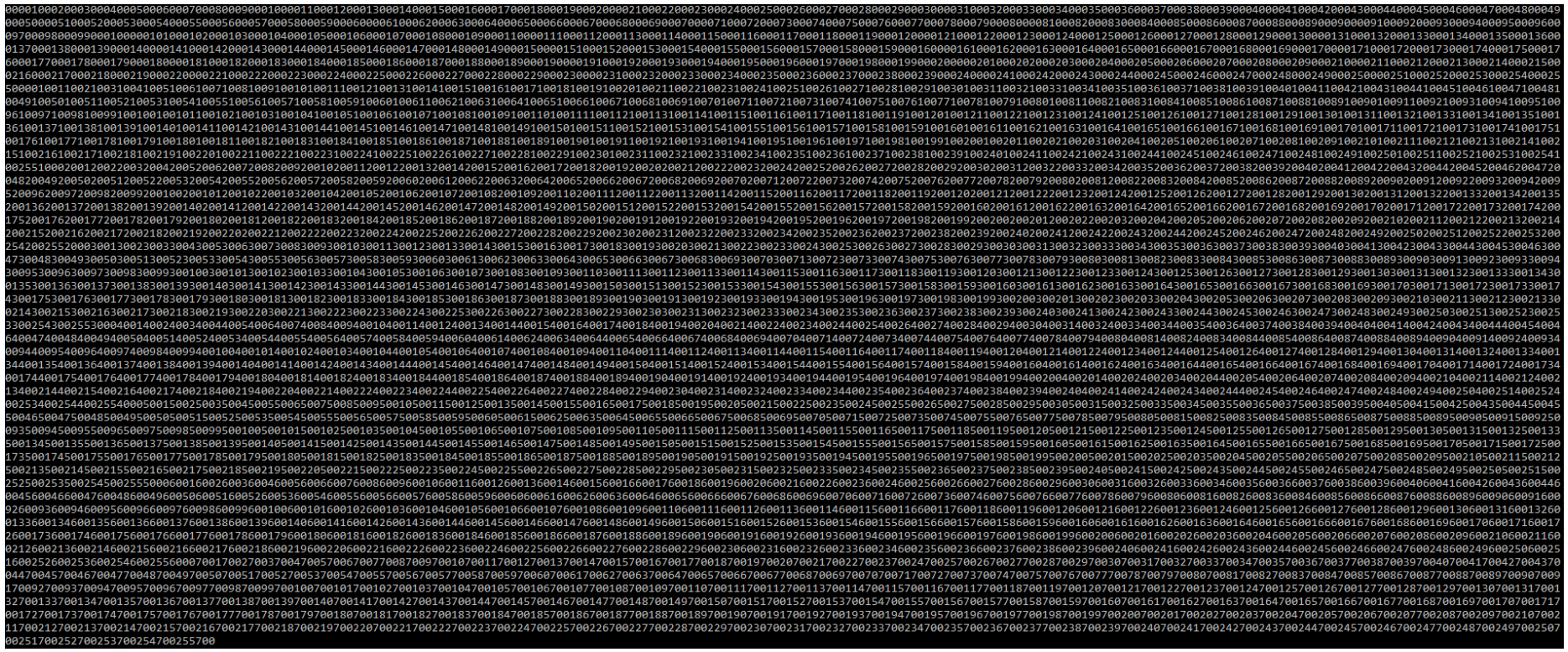
```
[ag17009@kairos test]$ python new_readout.py
nvmeCapture: Write FPGA data stream to Nvme devices. nvme: 2 startBlock: 0 numBlocks: 5242880
10:39:08.090: ErrorStatus: 0x0, StartBlock:        0, DataRate: 4832.916 MBytes/s, PeakLatancy:     3028 us
NvmeRead: nvme: 2 startBlock: 0 numBlocks: 10
Read complete at: 10 blocks
NvmeRead: rate: 4.684355 MBytes/s
Command completed
No blocks missing. 40.0 kBytes read.
```

Problem here is that the *test_nvme* software does its own error checking for missing blocks when it reads, so you'll never produce an error case when piggybacking off of it in this way.

- Need to write the readout solution into dedicated software that can perform the same tasks as Beam test software
- Need custom data sets with intentional errors for testing scenarios where blocks are missing/corrupt

NVMe Readout

Example of 8kByte data-stream output



These are still very primitive readout solutions performed mostly for sanity check and proof of concept purposes.