University of BRISTOL

DUNE

Readout/Buffering Update

ADAM GILLARD

ag17009@bristol.ac.uk

David Cussans, Kunal Kothekar

19/08/20

HiTech Global K800

K800 testing has been a great success in finding bugs that were unapparent in KCU105 testing

- Terry Barnaby was sent K800 test system to investigate bugs, on its way back to Bristol now
- ReadEngine error causing reports of non-physical Peak Latency values
  - Source: Logical error in VHDL code for statistics counter. Trivial fix
- More complex read error where valid blocks are being reported as invalid
  - This was seen only on Seagate FireCuda drives on both K800 and KCU105
  - Occurs when read requested very shortly after write – SLC NAND cache still moving data into TLC NAND storage after read complete
  - The FIFOs that collate data into one block ordered stream for each NVMe drive were overflowing when one drive was working faster than the other
  - Fix: Increased FIFO size and added bounds check with error report printf
  - This will need to be addressed when the NVMe readout software is designed

HiTech Global K800

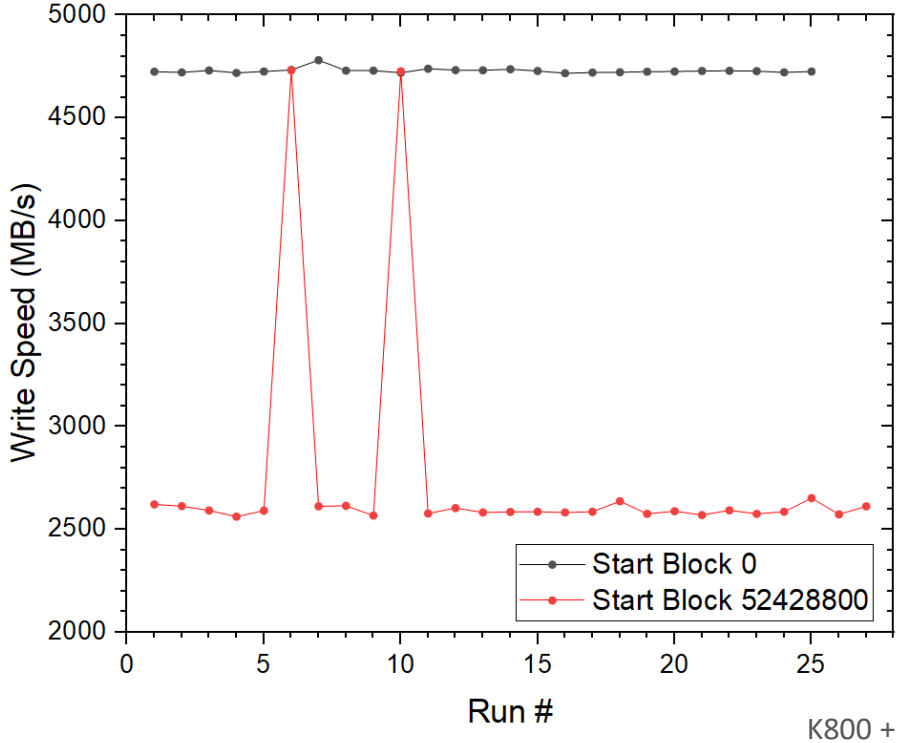
Example output of block validate error:

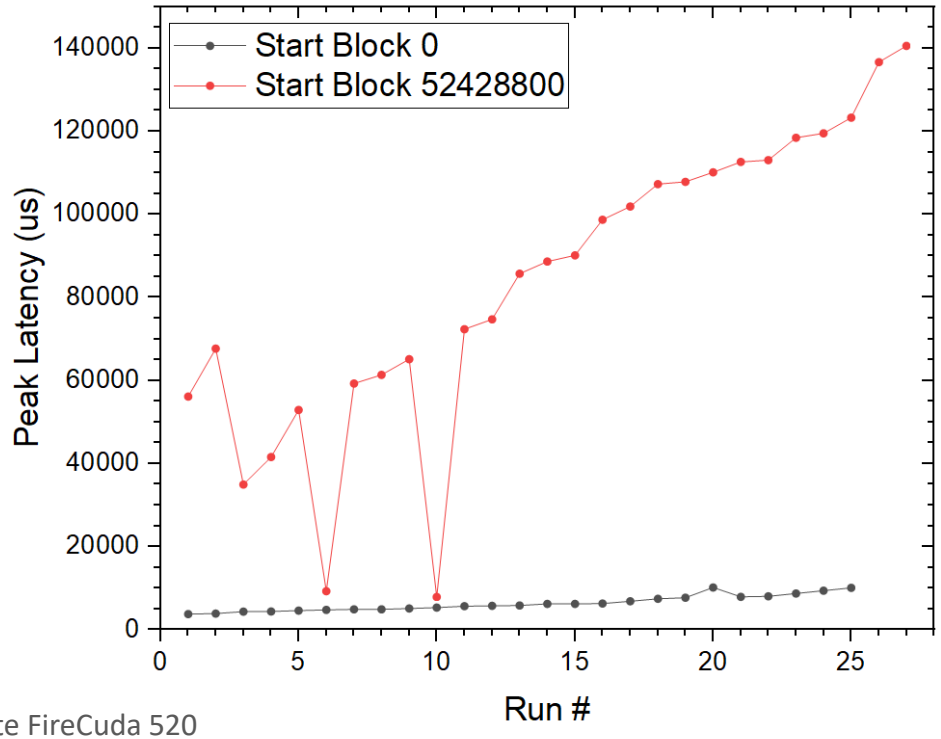
```
sh ./test1.sh
nvmeCapture: Write FPGA data stream to Nvme devices. nvme: 2 startBlock: 0 numBlocks: 52428800
13:34:08.407: ErrorStatus: 0x0, StartBlock:       0, DataRate: 4872.673 MBytes/s, PeakLatancy:     3359 us
NvmeRead: nvme: 2 startBlock: 0 numBlocks: 10000
Validate Error: Block: 3 Position: 0 has 0x00080c00 != 0x00000c00
Error in block: 3 startAddress(0x00000c00)
00080c00 00080c01 00080c02 00080c03 00080c04 00080c05 00080c06 00080c07
...
00080ff8 00080ff9 00080ffa 00080ffb 00080ffc 00080ffd 00080ffe 00080fff
Validate Error: Block: 5 Position: 0 has 0x00101400 != 0x00001400
Error in block: 5 startAddress(0x00001400)
00101400 00101401 00101402 00101403 00101404 00101405 00101406 00101407
...
001017f8 001017f9 001017fa 001017fb 001017fc 001017fd 001017fe 001017ff
Validate Error: Block: 7 Position: 0 has 0x00101c00 != 0x00001c00
Error in block: 7 startAddress(0x00001c00)
00101c00 00101c01 00101c02 00101c03 00101c04 00101c05 00101c06 00101c07
...
00101ff8 00101ff9 00101ffa 00101ffb 00101ffc 00101ffd 00101ffe 00101fff
```

HiTech Global K800

By performing a 200GB write loop (test 3 in BEAM test.sh) an interesting observation is found:

- By sorting the data by descending peak latency or ascending write speed, a very clear correlation can be found between the starting write block and the performance of the write - evidence of dynamic SLC cache size decrease

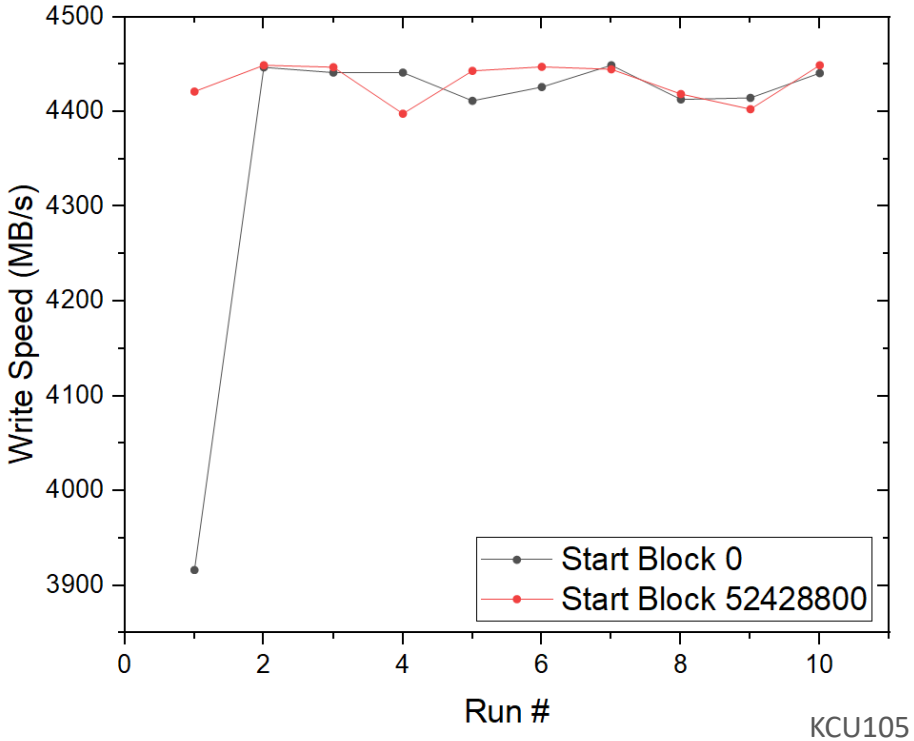


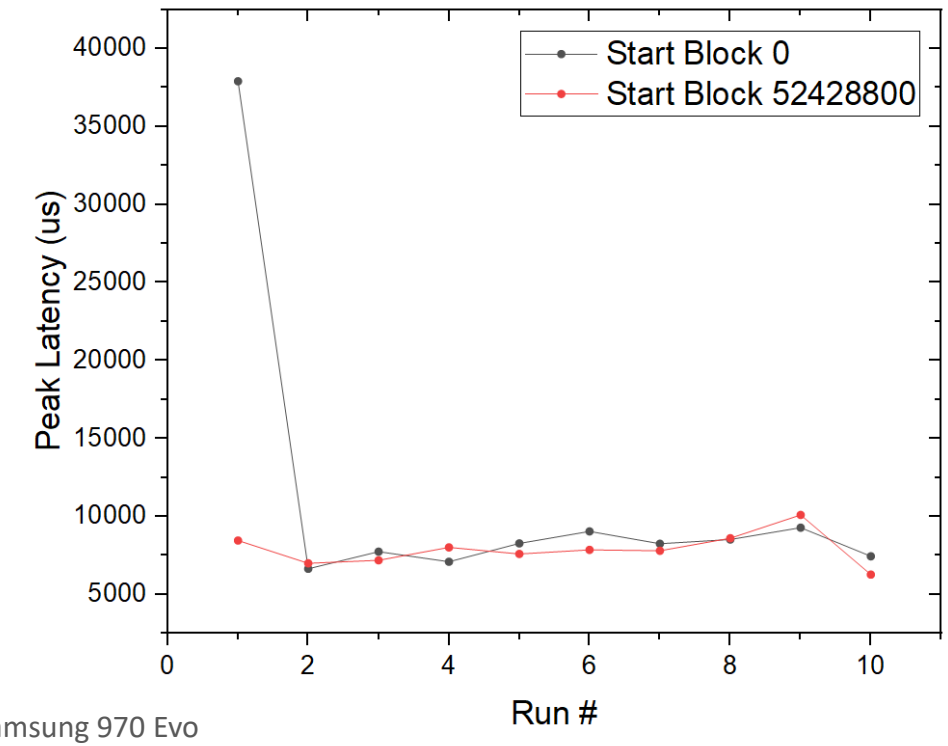


K800 + Seagate FireCuda 520

HiTech Global K800

We can compare this against data taken from test 3 on a KCU105 + Samsung 970 drive system to check that it is likely to be due to SLC cache drive architecture:



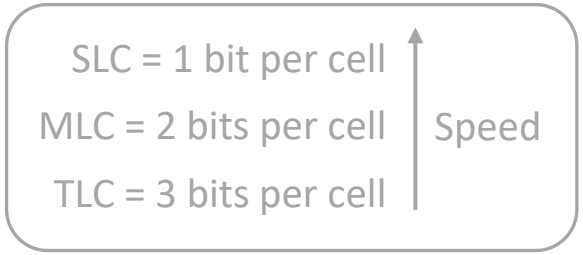


KCU105 + Samsung 970 Evo

HiTech Global K800

NVMe Device specific results:

- Seagate FireCuda 520
  - RAM cache to Dynamic SLC NAND cache (~2GB/s) to slow TLC NAND storage (~500MB/s) is unsuitable for large file sizes
  - Dynamic SLC NAND cache uses TLC storage cells over 1/3 of device free space and only stores 1 bit in those cells
  - SLC caching improve sequential write performance for files smaller than the cache size.
  - Full SLC cache = severe decrease in write speed = data still being moved within drive after write has been “completed”
  - Unsuitable for DUNE
- It is expected that this architecture will continue as commercial devices develop as it’s much cheaper
- Samsung 970 Evo
  - Write performance much more stable since SLC caching is not used – Straight to MLC NAND storage
  - MLC faster and have higher endurance than TLC hence cost vs type vs longevity trade off
  - NVMe form of “spin up” effect seen where devices that have been idle will perform at slower speeds on initial writes, could be due to APST (Autonomous Power State Transition) power saving. This should be able to be deactivated. Further investigation required.



SLC storage devices exist in Enterprise stores at very high cost, often PCIe form factor rather than M.2

In dual NVMe system, MLC drives are best option.

In single PCIe gen4 NVMe system, SLC will likely be the only option.

HiTech Global K800

BEAM git tree has been updated to allow for building KCU105 and K800 from same directory tree

- Private Github (https://github.com/DUNE/pl-nvme/tree/allBuilds/master) contains my own version which allows building for KCU105, K800 and VCU118 (still requires further development) - gain access from David Cussans
- Includes command-line automation to build directly to bitfile using flow

A fundamental VHDL file (NvmeStorageUnit.vhd) has been altered so that it now supports multiple families of platform

- Supports Ultrascale and Ultrascale+, with easy implementation for Versal ACAP when necessary

Beam have updated their documentation to include all changes resulting from this testing

- Including advice on NVMe device suitability in the current environment

I will be unable to test with the K800 as my placement ends on Friday 21st August

- Will be providing David and Kunal with a full report of the HTG-K800 and VCU118 build

VCU118 NVMe Firmware

The aim for doing this was to test moving the firmware to newer platform family boards (Ultrascale+) as a stepping stone to implement the firmware on the upcoming Versal ACAP architecture.

- Port to US+ is, in general, relatively straight forward
- Required changing how some of the VHDL code was being handled due to signal width changes etc.
  - Changes to: *DuneNvmeTop.vhd* and *NvmeStorageUnit.vhd*
- VCU118 specifically had complications due to its clock alignment with the required daughter board FMC pins
  - Requires separate GBT reference clocks for GTY quads, but only single clock FMC connection to Design Gateway daughter board for NVMe clock synchronisation
  - Resulted in timing critical warnings that must be addressed
  - Using VCU118 + Opsero 047 daughter board may be better solution as it has dual clock connection via FMC

Rob Halsall (RAL) is handling testing

- Poor initial performance seen with Seagate FireCuda drives (dips to <1500MB/s)
- Awaiting Samsung 970 results

Block Formatting

Implementation of block formatting VHDL code (written by Rob Halsall) into NVMe VHDL code

- Consists of replacing the current (very simple) data generation block with:
  - Data generation (counter) + Formatting into blocks + Header/padding insertion + Error checking
- Hence required to make adjustments to block formatting code to extract data, take correct inputs etc

Managed to get to a stage where the system would synthesise into theNVMe code, but not correctly

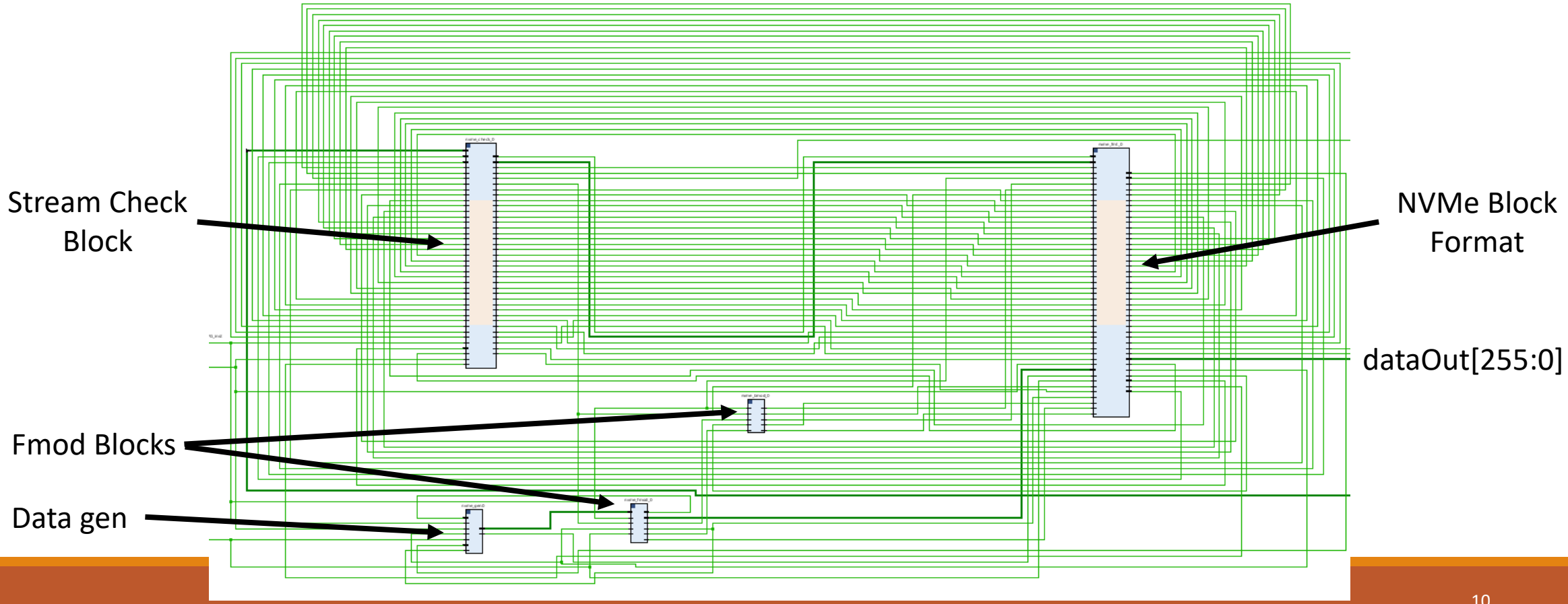
- Synthesis completes without error or critical warning
- The error checking block is getting optimized away every time, no matter what I try
- Hence requires further investigation and adjustment for full implementation

My progress is on the private Github (https://github.com/DUNE/pl-nvme/tree/BlkFmtDevelopment)

- Includes project files

Block Formatting

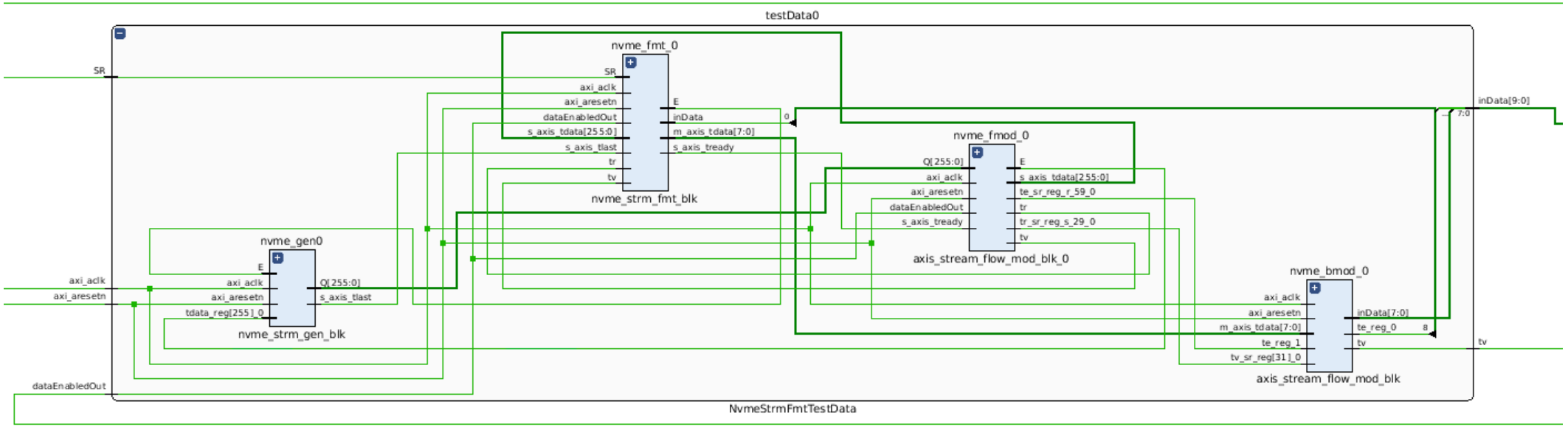
Expected output – Post synthesis schematic of block formatting project



Block Formatting

Schematic of Block Formatting Test Data block (post-synthesis):

- Missing *axis_stream_check_blk* (hence missing error output)
- dataOut is only 10 bit rather than 256 bit seen in block formatting project



Overall Outcomes

Migration of BEAM NVMe firmware to HiTech Global K800

- Findings from testing have helped identify and remove existing bugs in original design and subsequent changes have increased read-to-host speed from 17MB/s to over 400MB/s
- Given insight about versatility of firmware design across different platforms
- Given insight about NVMe device suitability requirements for DUNE

Migration of BEAM NVMe firmware to Xilinx VCU118

- Uncovered difficulties and restrictions of the firmware design
- Provided changes to firmware to give a stepping stone to migration to upcoming Versal ACAP

Adjusted original Makefile system to include all builds in single directory tree

- Provides build to bitfile from single command line for all existing build designs
- Provides an easy way to add new build designs – fully documented

Provided a starting point for Block Formatting integration to NVMe firmware

Performed a small investigation into block readout from NVMe to host