BEAM

Dune NVMe Storage Status

2020-08-17: Release 1.0.2

We fixed a few minor issues as seen with the HTG K800 port. The changes included:

1. The PeakLatency statistics values were sometimes invalid.
2. Read data block errors were occasionally seen.
3. Updated documentation.
4. Added documentation on how to use a different bus interface to the AXI4-lite used in the examples.
5. Added some information on the Seagate FireCuda write performance issues observed.

We believe the completes all of the contract work and the extra work we agreed to do apart from a few man days of on-going support.

2020-08-11: 1.0.2: Extra Items

The system was ported to a HTG-K800 board by Bristol University as their KCU105 board proved faulty. This worked but there is a minor issue with the peakLatency register being read incorrectly sometimes. We added some improved CDC crossing code for the register access and to make it easier for Wupper support but this did not fix the issue. We are awaiting a HTG-K800 board so we can investigate.

We have also performed some extra to contract work on the project, adding a complete read area NVMe trim system to simplify this operation as from NVMe performance testing it was noted that some NVMe's require the trim to be performed well before the write process. We have also improved the read performance of the system in case this is useful in the final system.

1. Improved CDC crossing system when reading registers.
2. Added support to the NvmeRead module to trim/deallocate a set of data blocks.
3. Fixed an issue in NvmeWrite if the optional trim function was enabled to trim the correct number of blocks (was doing 8x less).
4. Disabled Trim in NvmeWrite as support is now in NvmeRead where it is better placed for Dune usage and the way current NVMe's operate.
5. Improved NvmeRead performance to a Linux program. This was done by collating the small 128 Byte PCIe packets into a pseudo 4k block PCIe packet within the NvmeRead engine and sending these through the streams to the Linux driver. This reduces packet processing requirements in the Linux driver and application. Measured read performance is now in the order of 400 MBytes/sec on the test system. The NvmeRead engines SendFullBlock boolean can enable/disable this feature.

2020-07-18: 1.0.1: Second release of the System

Some minor updates to aid testing by Bristol.

2020-06-16: 1.0.0: First release of the System

We have made the first release of the Dune NvmeStorage system available. Recent work included:

1. Testing the performance with various NVMe's. We determined that there were performance issues with the Seagate FireCuda 520's and so have used the Samsung 970 Pro's for our testing. The performance figures are in the DuneNvmeStorageManual.
2. Creating a release compatible with the Opsero NVMe carrier card as the Dune project had difficulty sourcing the AB17-M2FMC carrier board.

The project work is basically complete apart from testing by the Dune project and any issues/missing features that are found added. There is a small amount of time left to do some additional work as needed. This might be to improve the read data speed but we will see how the testing work goes first.

2020-06-10: FPGA Bug Fixes and Testing

Main focus has been in testing the NvmeStorage module and fixing the issues found. At this stage all is working well and we can make the first external release of the system when wanted.

We are working on performance testing and optimisation. We have found some performance issues in initial testing that may require some further investigations.

1. When writing 200 GByte chunks continuously to separate NVMe areas the write performance drops from around the 4300 MBytes/s level to the 1000 MBytes/s level. We believe the trim/deallocate function in the NVMe's we are testing is slower than the block write. At the moment we start the trim while performing the write blocks. We may need to do the trim just after a 200 GByte chunk of data has been read to give the NVMe time to perform the block erasing after deallocation. We are investigating this and might add a function to the NvmeRead system to allow the host to trim/deallocate a data chunk with a simple register driven function.
2. The NvmeRead performance to the host is around 15 MBytes/s. The reason for the slow performance is the small PCIe packets in use (128 Bytes) and the hosts overhead with processing these. The project did not require a fast read of data but we could likely improve this to around 100 MBytes/s by adding an FPGA module to concatenate the small Pcie packets into a larger one to reduce host processing requirements.

```
nvmeCapture: Write FPGA data stream to Nvme devices. nvme: 2 startBlock: 0x00000000 numBlocks: 52428800
20:38:38.282: StartBlock:        0 ErrorStatus: 0x0, DataRate: 4530.139 MBytes/s, PeakLatancy:    10535 us
20:39:41.226: StartBlock: 52428800 ErrorStatus: 0x0, DataRate: 4582.337 MBytes/s, PeakLatancy:     3764 us
20:41:53.715: StartBlock:        0 ErrorStatus: 0x0, DataRate: 1782.111 MBytes/s, PeakLatancy:    25342 us
20:43:00.634: StartBlock: 52428800 ErrorStatus: 0x0, DataRate: 4390.204 MBytes/s, PeakLatancy:    11379 us
20:45:14.781: StartBlock:        0 ErrorStatus: 0x0, DataRate: 1761.116 MBytes/s, PeakLatancy:   216792 us
20:46:19.346: StartBlock: 52428800 ErrorStatus: 0x0, DataRate: 4425.614 MBytes/s, PeakLatancy:    12385 us
```

We will likely play with the Nvme'd DataSet Management Command to deallocate blocks and set performance parameters to see what difference this has.

We are now nearing the end of the project so it would be good to have a review from the Dune project to make sure all is ok, we haven't missed anything and if there are any changes desired.

2020-05-30: FPGA Timing improvements and Documentation updates

We have been tidying up the VHDL code, improving the timing and the constraints and extending the documentation of the system.

2020-05-21: Complete all functionality

We have now completed all of the functionality required. This includes:

- Read data blocks function.
- Trim/deallocate blocks function.
- Drop data blocks on input.
- Now supports separate PCIe clocks for the NVMe devices if needed.
- Write performance with two Seagate Firecuda NVMe's is 4644 MBytes/s.
- GitId: 9b9dd3c95f308480a40390eb40a32983f355cf5e
- Utilisation: 7010 LUT, 32 RAMB36, 26 RAMB18

We are now working on:

- Improving documentation.
- Tidying up the system prior to testing work.
- Testing read speed performance.
- Moving to the 250 MHz clock by default.
- Testing/improving error handling.
- Tidying up timing constraints.
- Improving test software.

For the testing work:

- Test write performance (and validity of writes) with 200 GByte writes (10 x) with all NVMe drives we have.
- Test read performance.
- Test write performance to destruction of 2 NVMe drives logging write performance an peak latencies.

2020-05-11: Tidied up code and tested with 100 GByte writes

- NvmeWrite now supports 8 concurrent block writes (configurable) and queues blocks in block number order.
- NvmeWrite now implements DataChunkSize limit as well as enable/disable.
- NvmeWrite now supports register writes for DataChunkSize and DataChunkStart locations.
- NvmeQueues now uses Blockram for queue storage and increased to 16 queue entries.
- Using 250 MHz clocks and Gen3 PCIe speeds for both the host and Nvme we get a speed of 2309 MBytes/sec on average when writing 100 GBytes of data to one Nvme so the system is now supported above the write performance needed. This was with the Seagate FireCuda 520 Nvme. We haven't looked at peak latency yet. There are more performance optimisations that could be done, but for the sake of keeping the code simple we will not do these.
- Our next actions include:
  - Simplify GIT tree. As we have been able to support the full Nvme test mode in the main NvmeStorage core we will simplify the GIT tree to just the sources/DuneNvme tree.
  - Firm up the top-level API and document more fully.
  - Implement the new host register system and modify the code to start to be able to support dual Nvme storage units.
  - Implement the Nvme Trim/Deallocate system (currently done manually in host software).
  - Implement NvmeRead system (although the host can read using Pcie commands at the moment).

GitId: 33013ede2fcb670e4ef8d471a0ff2de89bafa920

2020-05-04: Now writing data stream to the Nvme at needed data rate

- Added simple NvmeWrite multiple block write system writing 4 x 4k blocks at a time. This is very simple and is used for a performance check. With this we get write speeds of 1127 MBytes/s with 125 MHz clocks.
- Using 250 MHz clocks and Gen3 PCIe speeds for both the host and Nvme we get a speed of 2030 MBytes/sec so the system is on course for the write performance needed.

GitId: 8dcbade6d8a70a44c7f71d85b001de198e437747

2020-05-01: Now writing data stream to the Nvme

We have now implemented a first pass NvmeWrite engine that is able to accept an input data stream from a test signal generator (TestData) and write the resulting stream to the Nvme. We have tested this by writing 1 GBytes of test data to the Nvme. Data rates are a bit low at the moment, but the code is still pretty basic, is running at 125 MHz with slower NvmePcie interface speeds and only involves writing a single 4k block at a time ATM. Current data rate is: 617 MBytes/s .

BEAM

GitId: 4fd2b475bbf4f760c5668f6f0be2b642a4d4cfe2

2020-04-27: First DuneNvme Code produced

From the DuneNvmeTest example we have now progressed to what will be the final system layout, the DuneNvme code.

We have now thought about and designed a suitable architecture and design for the system. Using that we now have the FPGA configuring the Nvme and providing host access to the Nvme for data reads and writes. We are now implementing the first pass NvmeWrite engine that will perform the actual data writes. See the DuneNvmeStorageDesign.pdf for more info.

The architecture is now quite simple. It is based on passing Pcie TLP packets between the various processing engines (state machines) via a simple StreamSwitch. This allows a simple modular design and allows the host system full access to the Nvme at all times (suitable for debug but also getting status and logs etc).

The current VHDL and host test software is at:
NvmeStorageManual: https://portal.beam.ltd.uk/support/dune/files/doc//DuneNvmeStorageManual.pdf
NvmeStorageDesign: https://portal.beam.ltd.uk/support/dune/files/doc//DuneNvmeStorageDesign.pdf
Project layout and style: https://portal.beam.ltd.uk/support/dune/files/doc//DuneNvmeStorageProject.pdf
Doxygen: https://portal.beam.ltd.uk/support/dune/files/doc/DuneNvme/
Git: https://portal.beam.ltd.uk/gitweb/?p=dune.git;a=tree;f=source/DuneNvme

2020-04-07: Git repository Available

We have configured the git repository for the project. There is gitweb viewing available at: https://portal.beam.ltd.uk/gitweb/?p=dune.git and git cloning from: https://portal.beam.ltd.uk/git/dune.git. Both of these are linked to from the support website at: https://portal.beam.ltd.uk/support/dune/.

The only contents on here is the DuneNvmeStorageTest system that we have implemented to test basic FPGA NVMe access from a Linux host.

We have a suitable testdata source module implement and we are working on the NVMe VDHL data write system at the moment.

2020-03-19: Meeting with Bristol Physics

Video meeting describing progress and current status of project. Highlighted hardware requirements and presented proposed interface to the rest of the FPGA VHDL system. Described methods of NVMe configuration we could implement and possible software access to the system from the Linux host. We have setup meetings and actions for the following:

1. FPGA hardware requirements for NVMe access using Xilinx PCIe hard core blocks including external 100 MHz PCIe clock. This is described in the DuneNvmeStorageManual and DuneNvmeStorageDesign documents.

2. NvmeStorage data ingress stream, how to drop complete packets if latency is too high for external RAM buffer. Either NvmeStorage system does this with an external VHDL signal line or the external logic performs this function.
3. How to configure NVMe devices: VDHL or host software. Both have pros and cons. Host software would also allow interrogation of NVMe's statistics and other information.
4. VHDL Coding style discussions.
5. The Dune project intends to use the Opencores Wupper PCIe core and Linux driver. Beam will investigate its usage to see if it can be used as part of our test system and investigate methods of providing host to NVMe communications over this interface.
6. Git repository. We will shortly create a locally hosted and externally accessible Git repository for the project.

2020-03-11: We now have the test system fully accessing the NVMe device connected to the KCU105 FPGA board.

We have determined the minimum requirements for NVMe initialisation and writing and reading data blocks. In order to do this we ported our bfpga_driver to Fedora31 Linux as there were issues with the Xilinx xdma driver. We now have a test system that allows us to experiment with NVMe access via the Xilinx PCIe hardware blocks and can be use to help test and validate the NvmeStorage VHDL implementation.

2020-03-02: We now have a system where we can talk the the NVMe drive on the FPGA board from Linux.

We have configured the NVMe and accessed its registers and now have bus master mode working so the NVMe drive can access command queues from the Linux systems memory. This is allowing us to build a simple software model of accessing the NVMe drive in the simplest way that we will be able to move onto the FPGA in VHDL.

2020-02-27: The Test system components have been purchased and the FPGA development environment has been setup.

1. We have purchased all of the Xilinux and NVMe test hardware.

2. We have got the development/test environment all up and running with this environment and the latest Vivado Xilinx development software under Fedora31.

3. We have implemented some basic Host PCIe VHDL firmware and have managed to get the Xilinx XDMA Linux driver working (needed some changes) and communicating.

4. We have validated using the FMC NVMe adapter board with the KCU105, it seems like this should be fine for development.

2020-02-19: We have received the signed contracts and PO to start work.

We have initiated the purchase of the test/development hardware and started to get the test software environment setup and running.

2020-01-07: Started initial work

Although the contract has not been signed by the purchasing organisation and no PO has been generated, we started basic work on the project including setting up the support website.

2019-12-20: Awarded the contract and signed the contract.

2019-11-20: Tendered for contract

2018-11-26: Sent in a rough project proposal and estimate

2018-11-21: Originally started talking about the project