Sustained, High-Speed Data Recording with NVMe SSDs

MLE TB20201012

1. Introduction

MLE has been providing NVMe Streamer, an FPGA-based technology which enables users to directly stream onto NVM Express (NVMe) SSDs data to and from Programmable Logic (PL). The objective behind NVMe Streamer was to provide a solution for data recording (and re-play) without any CPUs involved, either because your FPGA does not have an embedded CPU or because you are looking for a solution with deterministically high read/write bandwidth and performance scalability.

Data Center Computational Storage uses similar concepts and as a matter of fact certain open-source portions of the “NoLoad” engine from Xilinx Alliance partner Eideticom are part of NVMe Streamer.

Most of MLE’s customers use NVMe Streamer for (high-performance) Embedded Systems, for example to record data in Test & Measurement or in Automotive Test Equipment applications. These applications typically require to record data at high data rates over long periods of time, sometimes using up the full SSD capacity - which, by the way, is not that long because recording at 2 GB/s will fill up a 1 TB SSD in 500 seconds or less than 10 minutes!

In working with our customers to support and optimize their systems we came across some findings about NVMe SSDs that we felt are worth sharing with you, including

- the significant performance differences of reading/writing small data sets compared to large recordings,
- environmental effects that impact your recording performance, such as Thermal Throttling, for example.

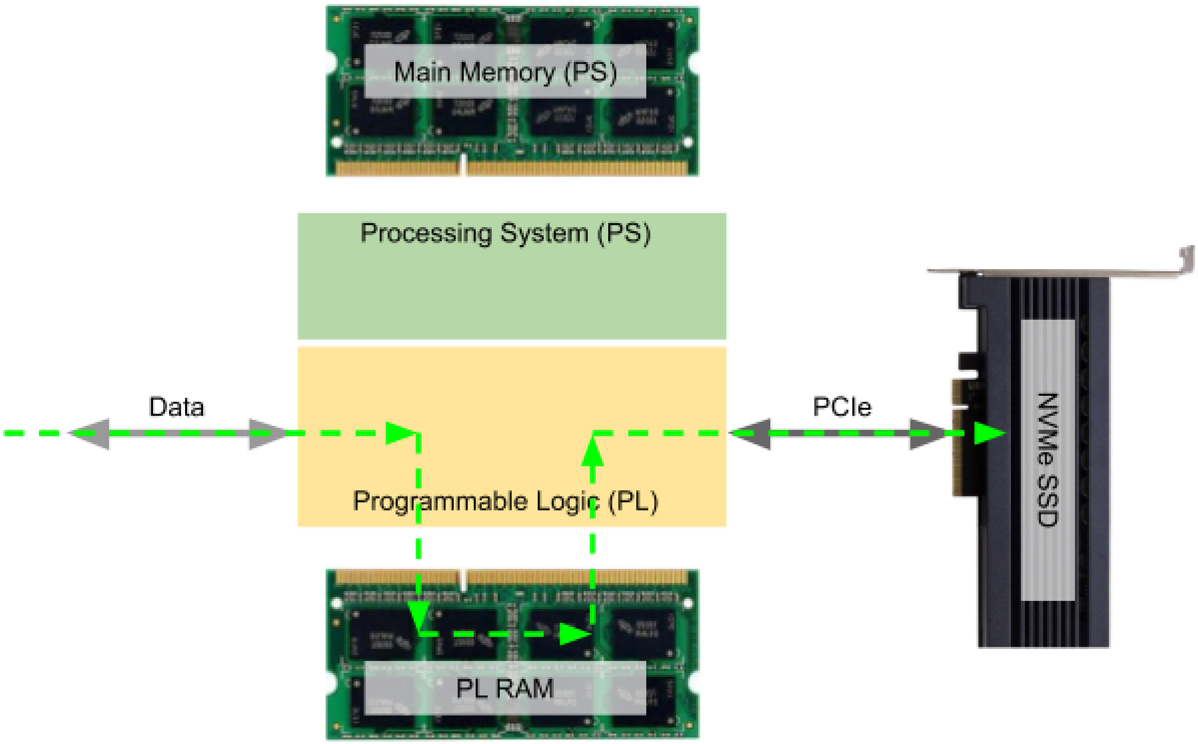
In no way this is meant to be a benchmark of different NVMe SSDs or SSD vendors, just a guide for the embedded systems engineers who are implementing an FPGA-based SSD recorder. Over the following pages we will describe

- MLE’s NVMe Streamer architecture and how it works,
- our test setup using a Xilinx Zynq UltraScale+ MPSoC and various NVMe SSDs,
- and an analysis of the performance results obtained.

Enjoy reading...

2. NVMe Streamer Technology

MLE's NVMe Streamer has been optimized for both reasonably high performance and for reasonable FPGA resource costs. Key differentiator is that NVMe Streamer allows writing data directly from an AXI4 Stream onto an NVMe SSD (and, obviously the opposite direction as well, reading from NVMe SSD into an AXI4 Stream), as the system-level diagram below shows:



Data flows between PL (for example entering the FPGA via GTX or GTH or GTY Multi-Gigabit Transceivers) and the PCIe direct-attached NVMe SSD, with an buffer in between using PL-attached DDR4 RAM (please refer to the FAQ section of our products page [http://MLEcorp.com/nvme] for details regarding this buffer).

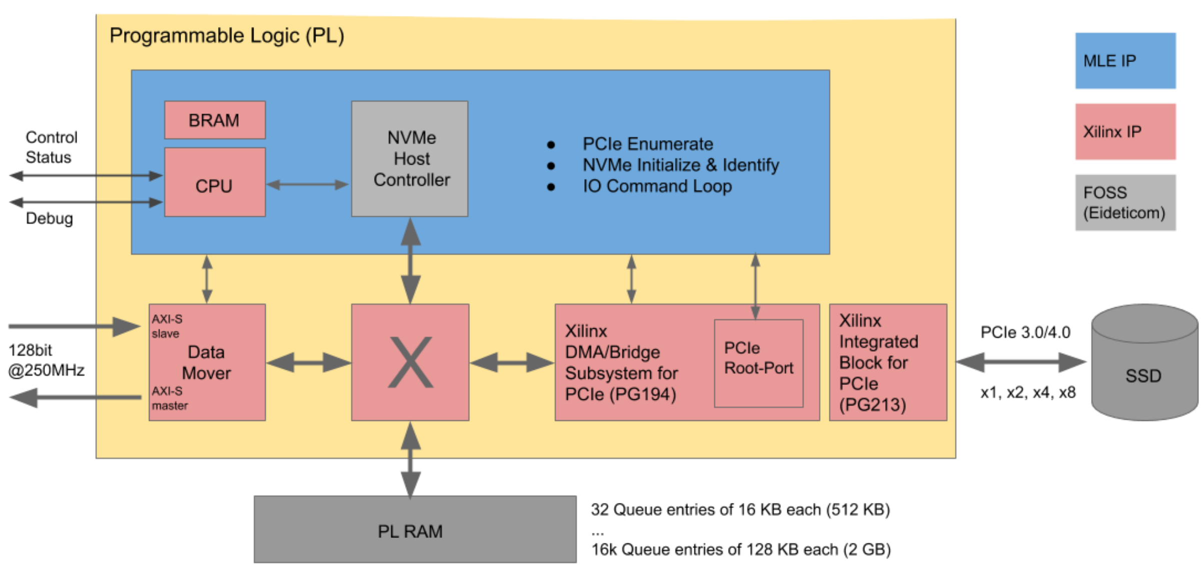
2.1 Using the Xilinx PCIe Hard IP

For efficient use of the FPGA resources NVMe Streamer instantiates the "Xilinx Integrated Block for PCIe" (refer to Xilinx PG213 [https://www.xilinx.com/support/documentation/ip_documentation/pcie4_uscale_plus/v1_3/pg213-pcie4-ultrascale-plus.pdf]) which we have configured to operate as a so-called PCIe Root-Port. This allows direct attachment of the NVMe SSD using up to 8 lanes each at 8 GT/s, according to PCI Express Base Specification 3.1.

Many Xilinx devices feature this PCIe Hard IP block, but not all devices, sometimes depending on the particular device package. We suggest to refer to the Xilinx device family overview, or just contact MLE [https://www.missinglinkelectronics.com/index.php/menu-contact].

2.2 NVMe Streamer Architecture

The following block diagram shows the key functions inside NVMe Streamer. The data path is horizontally drawn at the bottom of the diagram, controlled by the MLE NVMe Streamer:



Key components are:

- The open-source NVMe Host Controller (from Eideticom) which handles the NVMe command queues
- An instance of the Xilinx MicroBlaze CPU running software for PCIe/NVMe enumeration, initialization, setup and the command queues
- Said Xilinx PCIe Hard IP block
- Xilinx AXI4-Stream Data Movers and DMA PCIe Bridge Subsystem

In this architecture, the Xilinx MicroBlaze CPU takes care of the more complex but low speed functionality like PCIe bus enumeration. At startup, the Microblaze enumerates the NVMe SSD, sets up the NVMe host controller and hands the PCIe data over to the NVMe host controller. Furthermore the Micro Blaze controls the data movement and puts entries in the NVMe Host Controller's command queues.

2.3 Cost and Performance Based NVMe Architecture

We have optimized this architecture with regards to FPGA resources, yet supporting the performance expectations. Typical for Embedded Systems is the use of M.2 NVMe SSDs with PCIe Gen 3 x4 (four lanes). Theoretically, depending on payload size, the maximum read/write bandwidth is approximately 3.3 GiB/s.

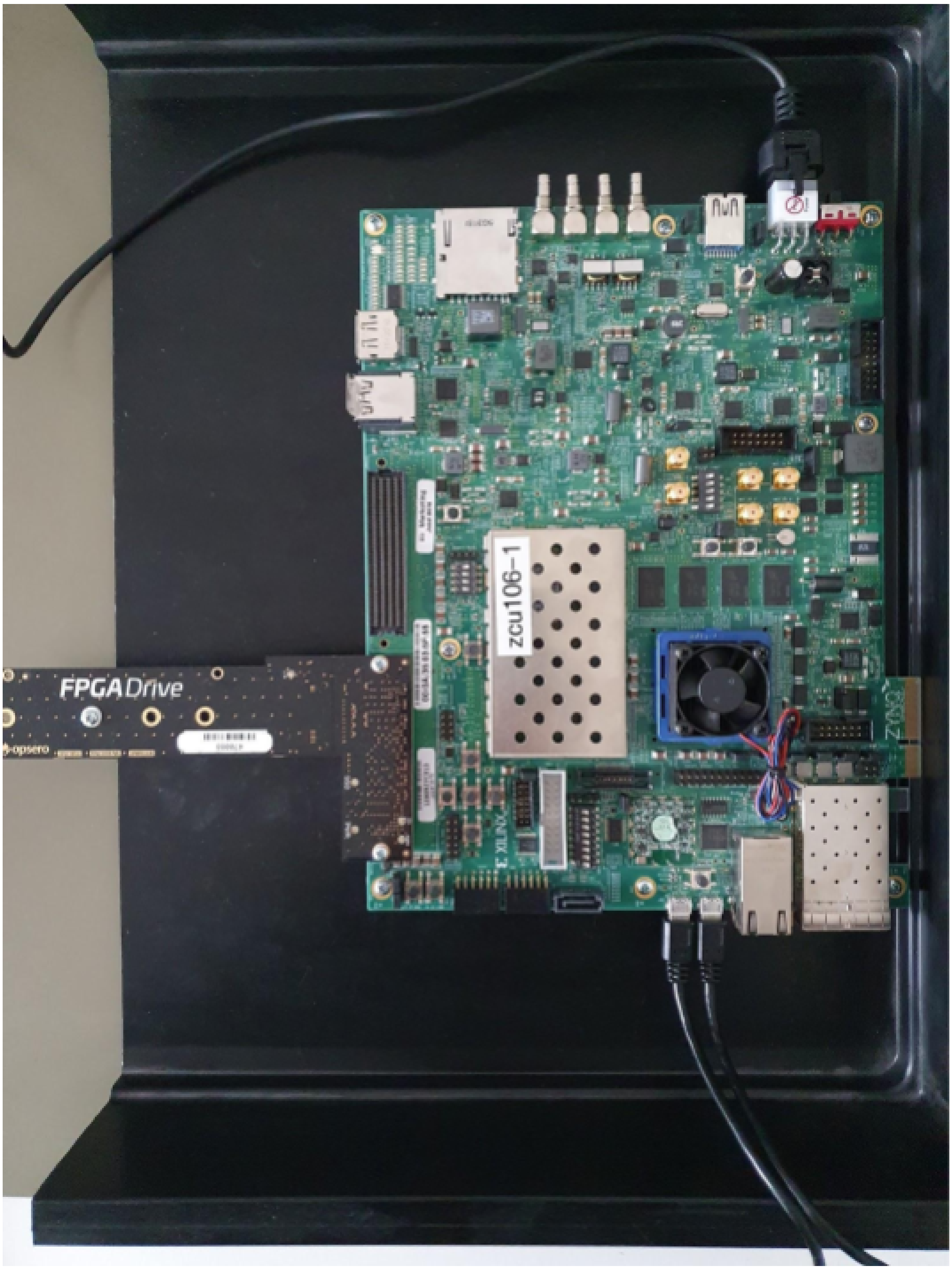
Therefore, we configured the AXI4 components to 128 bits wide running at 250 MHz, resulting in 3.7 GiB/s, sufficient for the maximum NVMe bandwidth. Similarly, we optimized the "bare metal" software on the MicroBlaze for AXI4-Stream performance based on certain features of MicroBlaze.

Hence with the standard release of NVMe Streamer you can expect bandwidth performance close to 3.1 GiB/sec. If your application needs to scale up in terms of performance we will be happy to discuss with you the several options for increasing bandwidth, for example a wider PCIe link with multiple, parallel instantiations with multi-SSD support.

3. Experimental NVMe Setup

For running our experiments we are using the standard Xilinx ZCU106 Development Kit [link https://www.xilinx.com/products/boards-and-kits/zcu106.html] together with the M.2 FMC adapter card from our fellow Xilinx Alliance partner, Opsero [link https://opsero.com/product/fpga-drive-fmc-dual/] and a subset of M.2 NVMe SSDs that we use in-house for quality assurance.

We are using the standard NVMe Streamer Evaluation Reference design (Release 1.1.0 GIT:geebd484). Additionally, as a Testbench, we have instantiated a second, separate Microblaze which "drives" the tests together with a test pattern generator, which has been implemented in PL to be fast enough to generate test data at high data rates to test the disk performance for writing. The test pattern generator also has a comparator to verify the correctness of the data which has been read back from the SSD. Here is a picture of one of the hardware setups we have been using in MLE's lab:



3.1 Selection of NVMe SSD

Besides the “big four” SSD vendors there are many other SSD vendors who offer SSD products specialized for certain application domains. This includes technical optimizations to meet environmental requirements, ambient temperature for example, as well as they address certain compliance or longevity aspects.

The selection of NVMe SSDs we picked for this Technical Brief was to show certain performance aspects in general, but not to benchmark one SSD vs the others. That is why we anonymized the SSDs in our results section. However, please note that our selection represents a combination of consumer and enterprise quality SSDs as well as SSDs targeted to industrial/automotive use.

3.2 Operating NVMe Streamer

The following explains the boot-time and run-time steps that the entire test setup (NVMe Streamer driven by the Testbench) goes through during the performance analysis:

1. The NVMe Streamer receives instructions from the Testbench regarding the amount of data which shall be transferred to (SSD write) or from (SSD read) the SSD together with the NVMe origin address.
2. The NVMe Streamer then sets up the Data Mover which, again, starts the transfer of the data to/from the PL RAM.
3. The NVMe Streamer then puts entries into the submission queue of the NVMe Host Controller.
4. The NVMe Streamer notifies the SSD, via the PCIe root complex, about available tasks in the submission queue.
5. The SSD reads one submission queue entry and starts reading the data from the PL RAM.
6. After all data is read by the SSD, and written to the Flash memory, the SSD sends a completion to the completion queue.
7. The NVMe Host Controller cleans the submission and completion queue and notifies NVMe Streamer about the completed task.

8. NVMe Streamer then refills the submission queue with new tasks or goes in standby.

Below you can find an exemplary screenshot of the NVMe Streamers console log:

```
INFO: Start ...

INFO: AXI4-Stream Pattern Generator/Checker IP Version 1.1.0
INFO: NVMe Subsystem IP Version 1.1.0
INFO: NVMe Subsystem SW Version 1.2.0
INFO: NVMe Subsystem IP Status:
INFO: NVMe Subsystem IP Status:  RP_Link_Up
INFO: NVMe Subsystem IP Status:  RP_Link_Up  PCIe_Enumerated
INFO: NVMe Subsystem IP Status:  RP_Link_Up  PCIe_Enumerated
SSD_Found  SSD_Gen3_x4
INFO: NVMe Subsystem IP Status:  RP_Link_Up  PCIe_Enumerated
SSD_Found  SSD_Gen3_x4  SSD_Initialized
INFO: NVMe Controller: [...]
INFO: NVMe Subsystem Media Size: 960197124096 (0xdf90356000) bytes
****************************************************************
TEST #00: test link status
****************************************************************
Test checks PCIe Root Port and NVMe SSD link widths and speeds.
The test is passed when both links are Gen3 x4.
********
INFO: NVMe Subsystem IP Status:  RP_Link_Up  RP_Gen3_x4
PCIe_Enumerated  SSD_Found  SSD_Gen3_x4  SSD_Initialized
****************************************************************
TEST #00: PASS
****************************************************************
****************************************************************
TEST #01: test flush rest of beats that have not been transfered
****************************************************************
After setting up Data Mover to expect one chunk to be transfered,
and the pattern generator to send two chunks, the one extra chunk
needs to be flushed.
The test is passed if after setting up start flush the pattern
generator DONE is set
and after setting up stop flush the flush finished flag is set
********
INFO: AXI4-Stream pattern generator not DONE
INFO: AXI4-Stream pattern generator DONE
INFO: Flush finished
****************************************************************
TEST #01: PASS
****************************************************************
****************************************************************
TEST #02: test pattern checker
****************************************************************
Test writes some data which then suppose to be read and checked
against different
pattern checker modes: wrong LFSR seed, wrong packet length, no
errors applied
The test is passed if proper flags of pattern checker status are set
and none of error flags is set when no errors expected
********
INFO: Writing 10 chunks to NVMe SSD
INFO: Reading 10 chunks from NVMe SSD with length error expected
INFO: Expected pattern checker length error set
INFO: Reading 10 chunks from NVMe SSD with LFSR error expected
INFO: Expected pattern checker LFSR error set
INFO: Captured LFSR_ERROR_BEAT: 0x0 does match expected: 0x0
INFO: Reading 10 chunks from NVMe SSD with no errors expected
****************************************************************
TEST #02: PASS
****************************************************************
****************************************************************
TEST #03: test flush after write
****************************************************************
Test sets FLUSH CONTROL flush after write and writes some data
It is expected that flush will start automatically after write
operation
The test is passed if after write check for FLUSH_STATUS in progress
is set
********
INFO: Setting FLUSH_CONTROL flush after write bit
INFO: Writing 100 chunks to NVMe SSD
INFO: Expected flush in progress set
INFO: AXI4-Stream pattern generator DONE
INFO: Flush finished
****************************************************************
TEST #03: PASS
****************************************************************
****************************************************************
TEST #04: test early tlast assertion
****************************************************************
After setting up the pattern generator to transfer chunks with size
smaller than 128 KiB an error should be reported in the WRITE_STATUS
register.
The test is passed if the WRITE_STATUS register has both 'finished'
and 'early tlast' flags asserted.
********
INFO: Flush rest of beats from Data Mover
INFO: Flush finished
****************************************************************
TEST #04: PASS
****************************************************************
****************************************************************
TEST #05: test write stop/abort functionality
****************************************************************
```

```
****************************************************************
TEST #07: PASS
****************************************************************
****************************************************************
TEST #08: test performance with different chunk counts per session
****************************************************************
Write and read data with incrementing number of chunks,
and measure the transfer speed while doing so.
********
INFO: Writing 1 chunks to NVMe SSD @ 0x20000
INFO: Writing 131072 bytes took 5715235 ns => 21 MiB/s
INFO: Writing 2 chunks to NVMe SSD @ 0x40000
INFO: Writing 262144 bytes took 135943 ns => 1839 MiB/s
INFO: Writing 4 chunks to NVMe SSD @ 0x80000
INFO: Writing 524288 bytes took 232893 ns => 2146 MiB/s
INFO: Writing 8 chunks to NVMe SSD @ 0x100000
INFO: Writing 1048576 bytes took 426080 ns => 2346 MiB/s
INFO: Writing 16 chunks to NVMe SSD @ 0x200000
INFO: Writing 2097152 bytes took 815091 ns => 2453 MiB/s
INFO: Writing 32 chunks to NVMe SSD @ 0x400000
INFO: Writing 4194304 bytes took 1585628 ns => 2522 MiB/s
INFO: Writing 64 chunks to NVMe SSD @ 0x800000
INFO: Writing 8388608 bytes took 3135149 ns => 2551 MiB/s
INFO: Writing 128 chunks to NVMe SSD @ 0x1000000
INFO: Writing 16777216 bytes took 16936226 ns => 944 MiB/s
INFO: Writing 256 chunks to NVMe SSD @ 0x2000000
INFO: Writing 33554432 bytes took 30950221 ns => 1033 MiB/s
INFO: Writing 512 chunks to NVMe SSD @ 0x4000000
INFO: Writing 67108864 bytes took 55906558 ns => 1144 MiB/s
INFO: Writing 1024 chunks to NVMe SSD @ 0x8000000
INFO: Writing 134217728 bytes took 116737721 ns => 1096 MiB/s
INFO: Writing 2048 chunks to NVMe SSD @ 0x10000000
INFO: Writing 268435456 bytes took 229073083 ns => 1117 MiB/s
INFO: Writing 4096 chunks to NVMe SSD @ 0x20000000
INFO: Writing 536870912 bytes took 499179910 ns => 1025 MiB/s
INFO: Writing 8192 chunks to NVMe SSD @ 0x40000000
INFO: Writing 1073741824 bytes took 970274279 ns => 1055 MiB/s
INFO: Writing 16384 chunks to NVMe SSD @ 0x80000000
INFO: Writing 2147483648 bytes took 1967248132 ns => 1041 MiB/s
INFO: Writing 32768 chunks to NVMe SSD @ 0x100000000
INFO: Writing 4294967296 bytes took 6037909785 ns => 678 MiB/s
INFO: Reading 1 chunks from NVMe SSD @ 0x20000
INFO: Reading 131072 bytes took 252488 ns => 495 MiB/s
INFO: Reading 2 chunks from NVMe SSD @ 0x40000
INFO: Reading 262144 bytes took 344922 ns => 724 MiB/s
INFO: Reading 4 chunks from NVMe SSD @ 0x80000
INFO: Reading 524288 bytes took 598130 ns => 835 MiB/s
INFO: Reading 8 chunks from NVMe SSD @ 0x100000
INFO: Reading 1048576 bytes took 731130 ns => 1367 MiB/s
INFO: Reading 16 chunks from NVMe SSD @ 0x200000
INFO: Reading 2097152 bytes took 1417171 ns => 1411 MiB/s
INFO: Reading 32 chunks from NVMe SSD @ 0x400000
INFO: Reading 4194304 bytes took 2785818 ns => 1435 MiB/s
INFO: Reading 64 chunks from NVMe SSD @ 0x800000
INFO: Reading 8388608 bytes took 5531806 ns => 1446 MiB/s
INFO: Reading 128 chunks from NVMe SSD @ 0x1000000
INFO: Reading 16777216 bytes took 11017258 ns => 1452 MiB/s
INFO: Reading 256 chunks from NVMe SSD @ 0x2000000
INFO: Reading 33554432 bytes took 21714381 ns => 1473 MiB/s
INFO: Reading 512 chunks from NVMe SSD @ 0x4000000
INFO: Reading 67108864 bytes took 43113391 ns => 1484 MiB/s
INFO: Reading 1024 chunks from NVMe SSD @ 0x8000000
INFO: Reading 134217728 bytes took 85911954 ns => 1489 MiB/s
INFO: Reading 2048 chunks from NVMe SSD @ 0x10000000
INFO: Reading 268435456 bytes took 173178213 ns => 1478 MiB/s
INFO: Reading 4096 chunks from NVMe SSD @ 0x20000000
INFO: Reading 536870912 bytes took 342703666 ns => 1494 MiB/s
INFO: Reading 8192 chunks from NVMe SSD @ 0x40000000
INFO: Reading 1073741824 bytes took 686703702 ns => 1491 MiB/s
INFO: Reading 16384 chunks from NVMe SSD @ 0x80000000
INFO: Reading 2147483648 bytes took 1371531563 ns => 1493 MiB/s
INFO: Reading 32768 chunks from NVMe SSD @ 0x100000000
INFO: Reading 4294967296 bytes took 2744671825 ns => 1492 MiB/s
*** Performance Summary ***
| chunks    | write        | read         |
|       1   |    21 MiB/s  |   495 MiB/s  |
|       2   |  1839 MiB/s  |   724 MiB/s  |
|       4   |  2146 MiB/s  |   835 MiB/s  |
|       8   |  2346 MiB/s  |  1367 MiB/s  |
|      16   |  2453 MiB/s  |  1411 MiB/s  |
|      32   |  2522 MiB/s  |  1435 MiB/s  |
|      64   |  2551 MiB/s  |  1446 MiB/s  |
|     128   |   944 MiB/s  |  1452 MiB/s  |
|     256   |  1033 MiB/s  |  1473 MiB/s  |
|     512   |  1144 MiB/s  |  1484 MiB/s  |
|    1024   |  1096 MiB/s  |  1489 MiB/s  |
|    2048   |  1117 MiB/s  |  1478 MiB/s  |
|    4096   |  1025 MiB/s  |  1494 MiB/s  |
|    8192   |  1055 MiB/s  |  1491 MiB/s  |
|   16384   |  1041 MiB/s  |  1493 MiB/s  |
|   32768   |   678 MiB/s  |  1492 MiB/s  |
NOTE: The performance of transfering few chunks is significantly
impacted
by setup and teardown as well as potential debug printouts via the
NVMe Subsystem UART.
```

```
Test provides two cycles of write and read of 32768 chunks with
different LFSR seeds
The second write is stopped by WRITE_CONTROL stop/abort flags, the
WRITE_ADDR_LAST is captured
The test is passed when a pattern checker fails on second read first
beat of the chunk that has not been overwritten
It does suppose to have deprecated data from previous write
operation with different LFSR seed
Expect no data to have been overwritten or corrupted.
********
INFO: Writing 32768 chunks to NVMe SSD
INFO: Captured WRITE_ADDR_LAST: 0xfffe0000
INFO: Reading 32768 chunks from NVMe SSD
INFO: Writing 32768 chunks to NVMe SSD
INFO: Set WRITE_CONTROL abort
INFO: Write aborted: Flush should be run automatically
INFO: Waiting until flush in progress
INFO: Waiting until pattern generator done
INFO: Flush finished
INFO: Captured WRITE_ADDR_LAST: 0x364e0000
INFO: Reading 32768 chunks from NVMe SSD
INFO: Captured LFSR_ERROR_BEAT: 0x3650000 matches expected:
0x3650000
****************************************************************
TEST #05: PASS
****************************************************************
****************************************************************
TEST #06: test write stop/abort functionality
****************************************************************
Test provides two cycles of write and read of 32768 chunks with
different LFSR seeds
The second write is stopped by WRITE_CONTROL stop/abort flags, the
WRITE_ADDR_LAST is captured
The test is passed when a pattern checker fails on second read first
beat of the chunk that has not been overwritten
It does suppose to have deprecated data from previous write
operation with different LFSR seed
Expect no data to have been overwritten or corrupted.
********
INFO: Writing 32768 chunks to NVMe SSD
INFO: Captured WRITE_ADDR_LAST: 0xfffe0000
INFO: Reading 32768 chunks from NVMe SSD
INFO: Writing 32768 chunks to NVMe SSD
INFO: Set WRITE_CONTROL stop
INFO: Write stopped: Flush should be run manually
INFO: Waiting until flush in progress
INFO: Waiting until pattern generator done
INFO: Flush finished
INFO: Captured WRITE_ADDR_LAST: 0x3f780000
INFO: Reading 32768 chunks from NVMe SSD
INFO: Captured LFSR_ERROR_BEAT: 0x3f7a000 matches expected:
0x3f7a000
****************************************************************
TEST #06: PASS
****************************************************************
****************************************************************
TEST #07: test for erroneous write outside specified boundaries
****************************************************************
Write data in multiple sessions with different chunk first and last
addresses.
Sessions place chunks at adjacent addresses and do so out of order.
Read back the chunks in multiple sessions with a different order
compared to writing.
Expect no data to have been overwritten or corrupted.
********
INFO: Writing 16 chunks to NVMe SSD @ 0x0
INFO: Writing 2097152 bytes took 15095520 ns => 132 MiB/s
INFO: Reading 16 chunks from NVMe SSD @ 0x0
INFO: Reading 2097152 bytes took 964410 ns => 2073 MiB/s
INFO: Writing 4 chunks to NVMe SSD @ 0x140000
INFO: Writing 524288 bytes took 232630 ns => 2149 MiB/s
INFO: Writing 4 chunks to NVMe SSD @ 0x40000
INFO: Writing 524288 bytes took 231036 ns => 2164 MiB/s
INFO: Writing 4 chunks to NVMe SSD @ 0xc0000
INFO: Writing 524288 bytes took 233086 ns => 2145 MiB/s
INFO: Reading 4 chunks from NVMe SSD @ 0x140000
INFO: Reading 524288 bytes took 359120 ns => 1392 MiB/s
INFO: Reading 4 chunks from NVMe SSD @ 0x40000
INFO: Reading 524288 bytes took 342565 ns => 1459 MiB/s
INFO: Reading 4 chunks from NVMe SSD @ 0xc0000
INFO: Reading 524288 bytes took 347455 ns => 1439 MiB/s
Just the AXI4-Stream transfers are much faster and almost constant
regardless of number of chunks.
****************************************************************
TEST #08: PASS
****************************************************************
****************************************************************
TEST #09: test read and write status error code
****************************************************************
Test checks WRITE_STATUS and READ_STATUS error codes are properly
assigned
after setting up operations with explicitly wrong first or last
addresses
********
INFO: Writing with WRITE_ADDR_LAST greater than the start address of
the last chunk
INFO: Write status error code address range greater than media size
has been set as expected
INFO: Writing with WRITE_ADDR_FIRST greater than WRITE_ADDR_LAST
INFO: Write status error code address first greater than address
last has been set as expected
INFO: Writing with unaligned WRITE_ADDR_FIRST
INFO: Write status error code address unaligned has been set as
expected
INFO: Reading with READ_ADDR_LAST greater than the start address of
the last chunk
INFO: Read status error code address range greater than media size
has been set as expected
INFO: Reading with READ_ADDR_FIRST greater than READ_ADDR_LAST
INFO: Read status error code address first greater than address last
has been set as expected
INFO: Reading with unaligned READ_ADDR_LAST
INFO: Read status error code address unaligned has been set as
expected
****************************************************************
TEST #09: PASS
****************************************************************
****************************************************************
TEST #10: test write read full disk
****************************************************************
Test performs write and read of maximum available number of chunks
Expect no data to have been overwritten or corrupted.
********
INFO: Writing all 7325722 chunks of NVMe SSD
INFO: Writing 7325722 chunks to NVMe SSD @ 0x0
INFO: Writing 960197033984 bytes took 2073003777126 ns => 441 MiB/s
INFO: Reading 7325722 chunks from NVMe SSD @ 0x0
INFO: Reading 960197033984 bytes took 538020405702 ns => 1702 MiB/s
INFO: Captured WRITE_ADDR_LAST: 0xdf90320000
INFO: Captured READ_ADDR_LAST: 0xdf90320000
****************************************************************
TEST #10: PASS
****************************************************************
****************************************************************
TEST #11: test link status
****************************************************************
Test checks PCIe Root Port and NVMe SSD link widths and speeds.
The test is passed when both links are Gen3 x4.
********
INFO: NVMe Subsystem IP Status:  RP_Link_Up  RP_Gen3_x4
PCIe_Enumerated  SSD_Found  SSD_Gen3_x4  SSD_Initialized
****************************************************************
TEST #11: PASS
****************************************************************

****************************************************************
****************************************************************
Overall Test Summary: PASS
   0 of 12 tests failed
****************************************************************
****************************************************************


INFO: ... End
```

4. NVMe Performance Analysis

In our performance analysis we focus on two separate aspects (and for both aspects we looked at read and at write performance):

The first aspect is what we refer to as “Average Peak Performance”. Here we repeatedly read/write various sizes of data at a time, starting with one chunk of 128 KiB all the way up to 4096 MiB (yes, four Gigabytes). Each time we measured performance and then averaged the results over all test runs. Our motivation was to look at the effects of SSD-internal caches.

The second aspect is called by us “Average Continuous Performance”. In this case we repeatedly wrote the entire SSD until it was full and then read back and compared the data using our test pattern generator / comparator. Each time we measured performance and then averaged the results over all test runs. Idea was to emulate the record/replay use case.

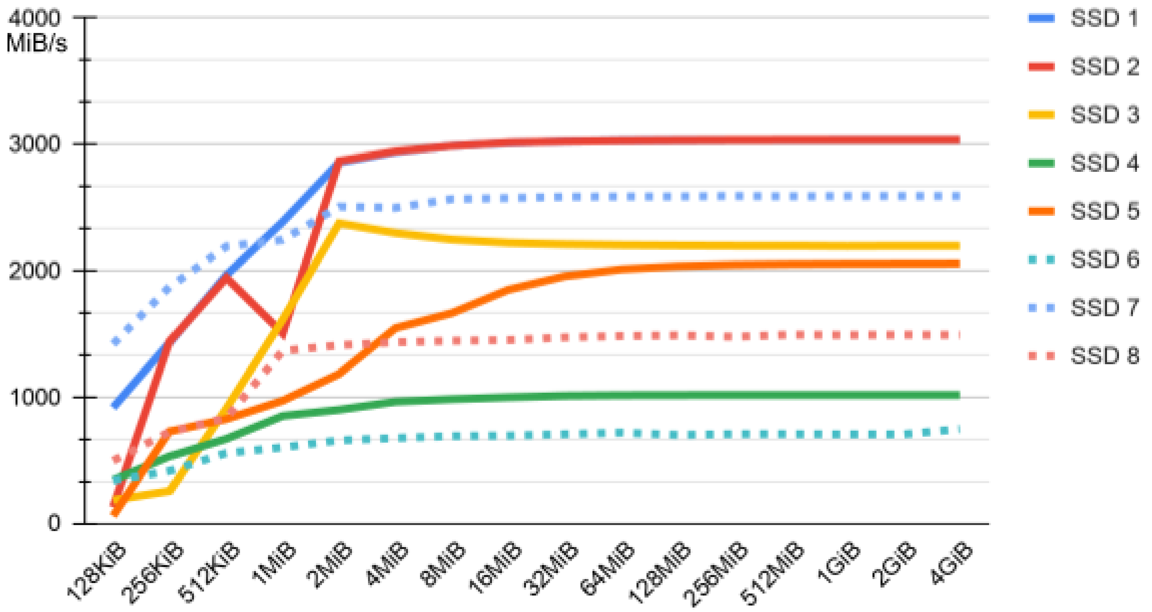
Here is the data...

4.1 NVMe SSD Average Peak Performance

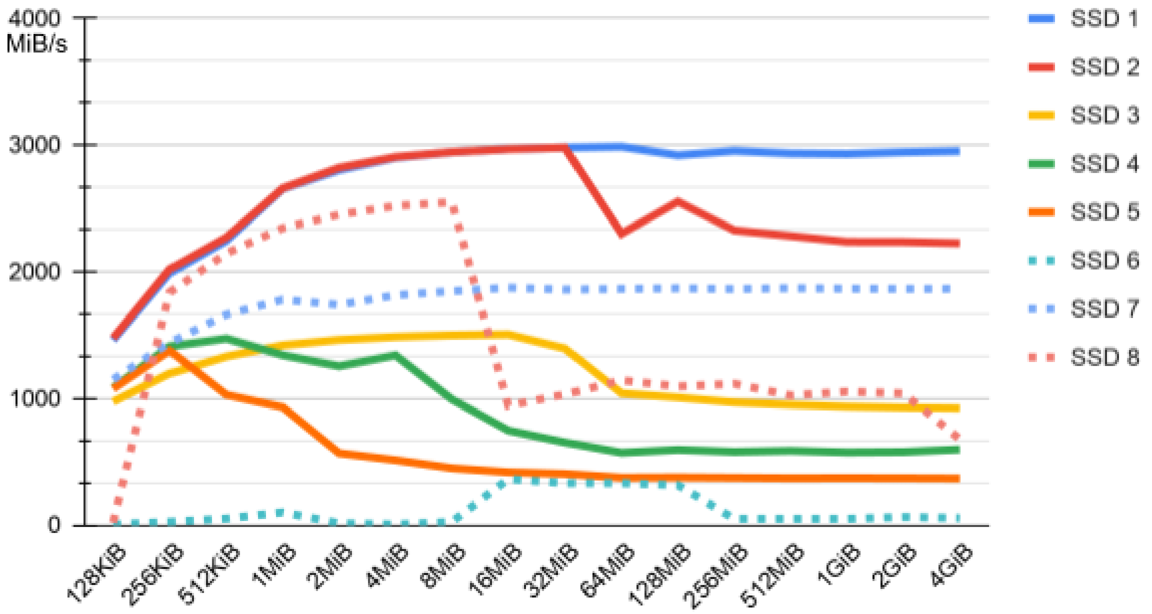
Again, NVMe Streamer measured the average peak performance with a sweep over a various data chunk sizes, from 1 to 32768. Each data chunk contains 128 KiB of data, which equals to a data size sweep from 128 KiB to 32768 * 128 KiB = 4096 MiB. The amount of data and the required time to read, or write, the data to the disk, averaged over several runs, results in an “Average Peak Performance”.

You can see how different each SSD behaves, and effects which we believe come with the behavior of SSD-internal caches: Some of those SSDs, according to their datasheets, have Flash memory based caches and others have DDR RAM based caches.

When looking at the Average Peak Read Performance you can notice that SSD 1 quickly approaches a read bandwidth of approx. 3 GiB/s once data sets are larger than 1 MiB, while SSD 2 shows a minor performance drop between data sizes of 512 KiB and 1 MiB.



When looking at the Average Peak Write Performance SSD 1 supports approx. 3 GiB/s for data sizes of 4 MiB, and larger. SSD 8, however, shows a significant performance drop once data sizes are larger than 8 MiB. We are not sure what may have caused the "weird" behavior of SSD 6.



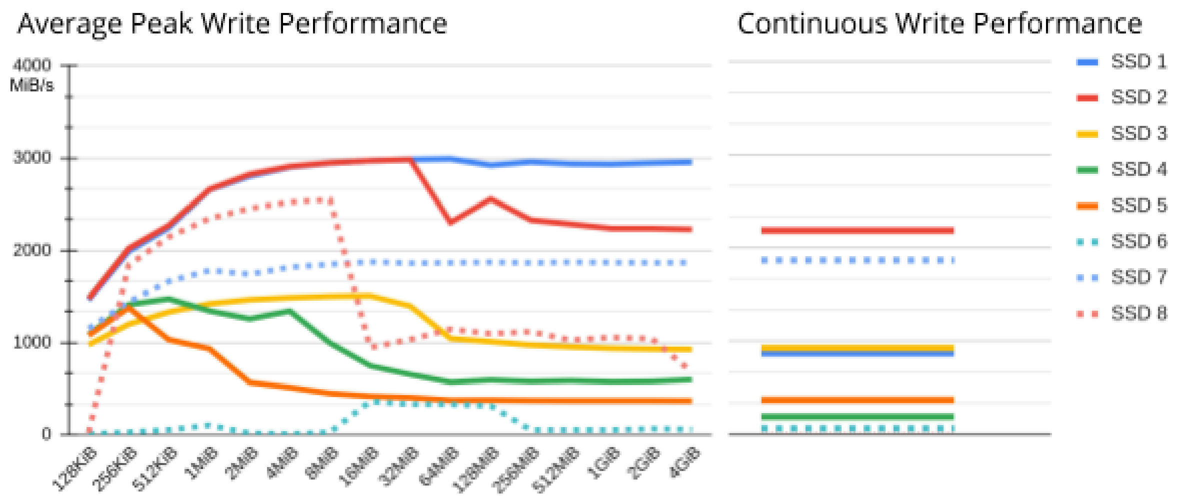
4.2 Average Continuous Performance

As many applications require a continuous datastream to the disk, MLE includes a test which mimics these kinds of applications. NVMe Streamer will write one continuous data stream until the SSD is full. Afterwards, the same data gets read back and compared with the internal pattern checker. As before, the amount of data and the required time result in an Average Continuous Performance. However as it is an average value which includes disk internal tasks, like wear leveling, or temperature throttling. Due to this, it isn't given this performance is given all the time during operation, at the beginning it could be faster as disk temperature is low. At the end the disk could be in Thermal Throttling and SSDs sometimes slow down when the SSD gets full.

However, what is more interesting to watch is the comparison between the Average Peak Performance and the Average Continuous Performance.

For the read performance comparison you can see, that SSD2 is very consistent, even when writing the full capacity (500 GB). SSD 1 does show a performance drop from approximately 3 GiB/s down to 2 GiB/s.

For the write performance comparison we see similar results: SSD 1 has a significant performance drop from approximately 3 GiB/s down to 1 GiB/s while SSD 2, 3, 5, and 6 show full write performance even when writing the full capacity (500 GB).



4.3 Disk consideration conclusion

Besides the access pattern and read/write speeds that your application requests from the SSD, the SSD also needs to handle the data written over live time (measured as in TeraBytes Written, TBW, or in Drive Writes per Day, DWPD; WDC has a great Cheat Sheet for this [link https://blog.westerndigital.com/ssd-endurance-speeds-feeds-needs/]). This so-called endurance differs a lot between consumer and industrial / data center grade disks. We can only recommend to review the datasheet of the SSD. If there is no detailed datasheet and your data matters to you, we suggest to switch to a "better" SSD.

5. Conclusion

We used the Evaluation Reference Design of MLE's NVMe Streamer to analyze the read/write performance of several NVMe SSDs for what we believe mimics some typical use cases. Despite a resource efficient implementation, NVMe Streamer delivers very high performance, which means that in most cases your SSD likely becomes the performance bottleneck. What we did not do (yet) is to consider environmental effects such as temperature and/or vibration.

Side note: No SSD was harmed during our testing!

This is not a joke. It means that, despite the fact that we repeatedly have been overwriting SSDs at their full capacity, none of our SSDs have died, nor showed bit errors when reading back and comparing the data. Apparently, Flash memory has become much more resilient over the last years!

Authors and Contact Information

Andreas Schuler, Dir. Application, Missing Link Electronics GmbH
Endric Schubert, PhD, CTO, Missing Link Electronics, Inc.

Missing Link Electronics, Inc.
2880 Zanker Road, Suite 203
San Jose, CA 95134, USA
+1-408-475-1490

Missing Link Electronics GmbH
Industriestrasse 10
89231 Neu-Ulm
Germany

www.missinglinkelectronics.com