IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)
e-ISSN: 2278-1676,p-ISSN: 2320-3331, Volume 15, Issue 2 Ser. I (Mar – Apr 2020), PP 54-58
www.iosrjournals.org

Enabling Fast Decision Criteria in NVMe SSDs through NVMe-based Switch

Vinícius G. Linden[1], Rodrigo M. Figueiredo[1], Cassiano S. Campes[1,2], Lúcio R. Prade[1]
[1]*(Polytechnic School, Unisinos University, Brazil)*
[2]*(College of Information and Communication Engineering, Sungkyunkwan University, South Korea)*
Corresponding Author: Vinícius G. Linden

Abstract:
***Background**: Solid-state drives (SSDs) have increased throughput and bandwidth drastically with the introduction of Non-Volatile Memory Express (NVMe) protocol, profiting from the state-of-the-art Peripheral Component Interconnect Express (PCIe) interface. Moreover, SSDs are perceived by the host as a black-box where they simply service the write and read requests. This inhibits the host to acknowledge the device's internal architecture. Additionally, the SSD does not distinguish between data and metadata requests. Thus, the host still lacks opportunities to get the best performance from SSDs due to the data semantic gap.*
***Materials and Methods**: This paper proposes a hardware solution to enable fast decision criteria in NVMe SSDs. Our proposed model is flexible enough to be configured for specific workloads by the host. The solution enables switching decisions to be transferred into the hardware, minimizing CPU usage.*
***Conclusion:**The data semantic gap is shortened in the hardware level, consequently reducing the write requests to flash and widening the life-span of the device.*
***Key Word**: NVMe; Switch; SSD; HDL.*

Date of Submission: 12-03-2020 Date of Acceptance: 27-03-2020

I. Introduction

The Non-Volatile Memory Express (NVMe) specification has allowed higher bandwidth through the use of Peripheral Component Express (PCIe). It creates an interface for the host to communicate with the Solid-State Drive (SSD). Like any other storage interface, it takes addresses and data to either write or read, making the device to be perceived as a black-box. As an attempt to reduce the data semantic gap in NVMe version 1.3, the multi-stream concept has been introduced[1], enabling the separation of data inside the SSD. Nevertheless, stream separation is not enough to transfer the data semantic to the storage, leaving room for improvements. These enhancements would greatly benefit systems that are characterized by a massive use of SSDs.

State-of-the-art data-centers are increasingly employing SSDs for storage [2, 3]. They require a reliable File System (FS) that is able to handle a massive storage space, such as the *ext4*. This FS is prevalent in data-centers [4] and Linux systems in general. It separates the storage space into multiple block groups segments, reserving the first 1024 bytes for the bootloader. Each block group is further divided in superblock, group description table, block bitmap, inode bitmap, and data blocks. These regions are updated according to specific workloads and, consequently, metadata may have a much higher update frequency than data blocks. Because SSDs do not allow in-place-updates, a Flash Translation Layer (FTL) is employed to make the logical to physical addressing transparent to the host. However, excessive physical writes into the storage shorten the life-span of the flash cells due to wearout [5, 6]. This feature is characteristic of flash technology, so an SSD array system will still lose flash cells due to wearout caused by repetitive writes.

A system that has an array of NVMe-based SSDs is always routed through a PCIe switch, but since PCIe packets are not the same as NVMe commands, NVMe fields are not accessible to the PCIe switch in this manner. Thus, a PCIe switch does not allow command separation in NVMe. Therefore, a switch for NVMe commands is beneficial, because it allows multiple implementations to specific workloads [7]. The NVMe-based switch can be configured to make decisions for a given workload, in a given system. It is more advantageous to process data in- or near-storage with hardware for reduced latency and reduced CPU utilization in I/O processing [8, 9, 10]. To the best of our knowledge, there is no proposal for an NVMe switch solution that allows routing criteria based on a configurable aspect of the NVMe command itself. This paper proposes a novel packet-based NVMe switch architecture, capable of separating different NVMe commands based on the content of one or more of NVMe fields. This can be accomplished in Hardware Description Language (HDL), enabling the implementation on a Field-Programmable Gate Array (FPGA) [11]. The main contributions are: a shortening of the data semantic gap between host and SSD by allowing in-hardware NVMe-based command switching; a way to generate designer-defined NVMe packets out of NVMe commands; a basic platform for NVMe system manipulation; and the NVMe switch flexible architecture.

II. Proposed System

A top-level architecture view is presented in **Figure 1**. There, the switch logic is between the PCIe and NVMe interfaces for communication with a host and the SSD. The NVMe command can be sent either to the SSD or to a *generic NVMe* device. The last being the main reason for this architecture. As discussed in the Introduction, the designer may choose the *generic NVMe* to implement as any possibility to enable near-storage processing. The switch's configuration is asynchronously introduced via an external interface. This configuration port is used to adapt the switch to specific workloads.

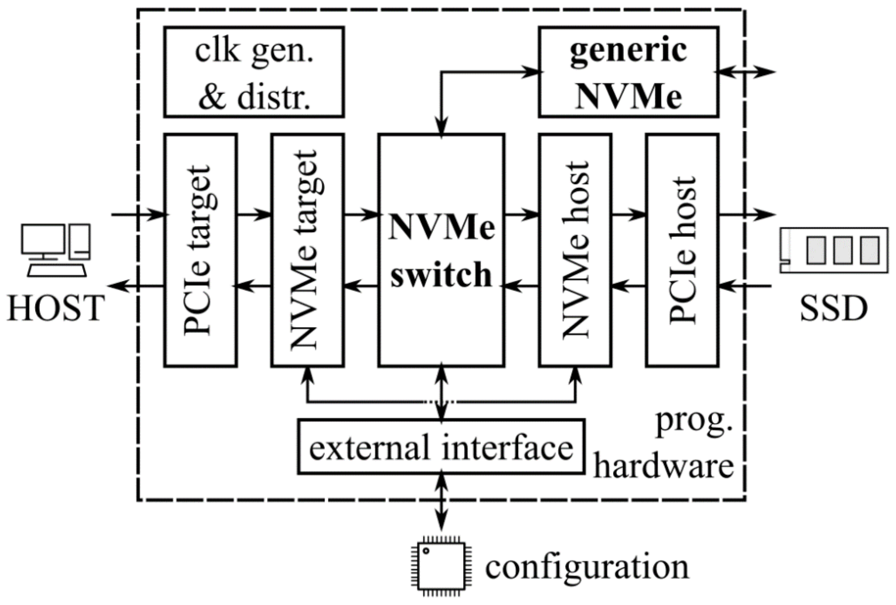

Fig. 1. Top-level architecture: it shows the layout of the modules with the NVMe switch in the middle

The *PCIe target* communicates with the host, and the *PCIe host* with the SSD. Similarly, the *NVMe target* communicates with the host, and the *NVMe host* with the SSD. The words “target” and “host” convey the sense of what interface is perceived by the host and SSD, respectively. The PCIe is required by the NVMe specification. NVMe modules deal with NVMe control, data flow, and PCI configuration. They translate the data to and from the NVMe streaming standard. This allows a flexible bridge for NVMe, which may be expanded later as needed. By virtue of the packet stream standard, the switch is presented in the data-path configuration.

The NVMe streaming standard uses byte-oriented frames. The first byte is acknowledged with the handshake, using the*valid* and *ready* signals. The rest of the bytes are automatically transferred. The burst length will be known according to the type of command: Submission Command (SC), 64 bytes; or Completion Commands (CC), 16 bytes. The *valid* signal comes from the source, indicating that the first byte is valid; and the *ready* signal is from the sink, indicating that the sink is ready to receive a full NVMe packet. The data transfer can only be peer-to-peer without a bus splitter module. This standard is the basis for the whole architecture. **Figure 2** shows a detailed view of the proposed switch architecture. The emphasis is on SC. The CC processing changes depending on the specific implementation of the *generic NVMe*. The relevant modules are explained in more detail below.

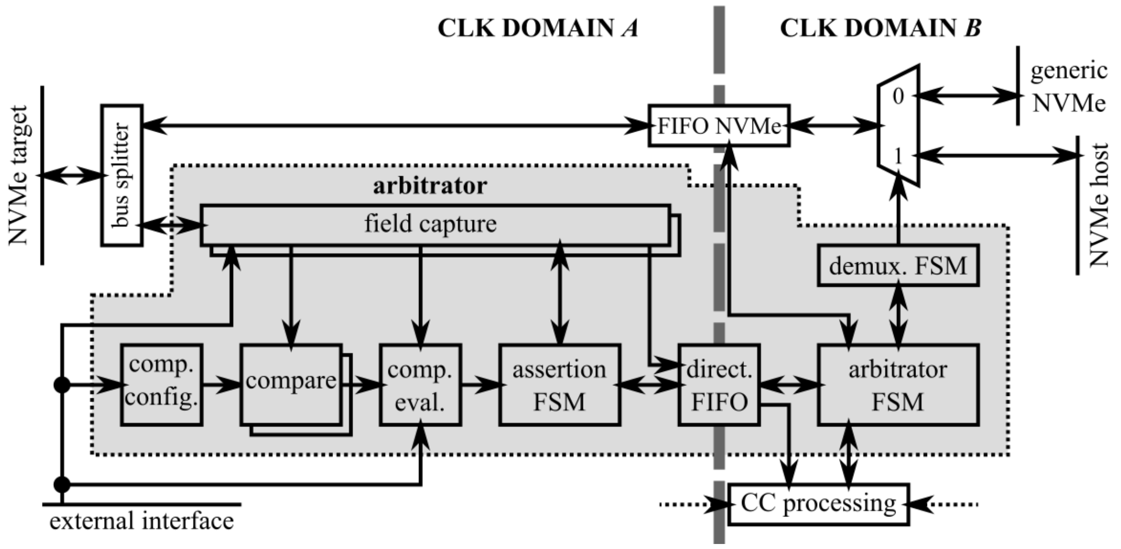

Fig. 2. Switch's internal architecture: it presents the SC packet flow side (above) and the CC processing side (below)

The *field capture* module observes the packets and cherry-picks NVMe fields based on a particular configuration and opcode. This module also separates the opcode of the command for use in other modules. It can accept multiple predefined configurations for arbitrator.

A *compare* module outputs a logical '1' or '0' depending on the evaluation. Comparisons are configurable for the type of evaluation (i.e. equality, inequality, boolean, etc) and comparison values. The module takes two evaluations and one evaluated value. Provided a configuration, the module will compare if the evaluated value is within a certain range, which is defined by one or two evaluation values. This is useful, for example, in taking a Start of Logical Block (SLBA) field and checking if this field hits a certain range in the address space. The designer may want to add more *compare* modules in parallel to check for more conditions. In this case, a *comparison evaluator* module will be required to logically combine the outputs. Moreover, the *comparison configurator* module is necessary to correctly translate between the arbitration configuration to the *compare* module. This arbitration configuration is acquired from the central processing unit (CPU), through the external interface.

The *FIFO NVMe* and *direction FIFO* are employed to completely separate clock domains. These clock domains are necessary in order to allow any halt by the SSD or *generic NVMe* without packet loss. There are two Finite State Machines (FSMs) in each clock domain: the *assertion* and the *arbitrator*. They are responsible for system management and synchronization. The main output of the *assertion* FSM is the direction of the command, alongside the FIFO's storage control signals. Because of the NVMe stream standard, the demultiplexer cannot change direction in the middle of a packet. Therefore, a *demultiplexer* FSM is a system to lock control over a module. It is used to resolve the dispute over the demultiplexer between the *generic NVMe* and *arbitrator* modules. Note that the packets are being stored in the *FIFO NVMe* while simultaneously being processed by the *arbitrator*, allowing for reduced latency. With this system in mind, one may proceed to analyze the simulation results presented in the next section.

III. Simulation Results

The proposed architecture shown in **Figure 2** has been developed in Very High Speed Integrated Circuit Hardware Description Language (VHDL). The implemented proof-of-concept is the switch, containing only one *compare* module. The SSD is linked to a logical '1' and the *generic NVMe* to a logical '0' (as shown in **Figure 2**) in the simulations below. The simulation was run in Vivado® Simulator 2019.1 and is available at GitLab (https://gitlab.com/storagesystemslaboratory/nvme-switch). The test configurations are shown in **Table no 1**. The AC_WORKLOAD configuration tests if the command's SLBA field access a certain address range, namely an *ext4*'s superblock.

Table no 1: Tested arbitrator configurations.

Configuration name	**Value**	***Generic NVMe*?**
AC_IDLE	0x00	FALSE
AC_SEND_GENERIC	0x01	TRUE
AC_SEND_SSD	0x02	FALSE
AC_WORKLOAD	0xFE	$v1 \leq SLBA \leq v2$

The testbench itself is a VHDL code with an automatic file consulting system - with each file being a NVMe packet. Each one is described in a separated text file, having a title and an activation signal. The title corresponds to the command's opcode, for our understanding. Once the reading of these packets is activated, the testbench will send the whole packet. Note that, if a packet does not contain the SLBA field used in AC_WORKLOAD the system will send it to the default destination - configured as the SSD. There are two packets that contain the SLBA field: *read1* and *read2*. Only the *read2* is to be sent to the *generic NVMe*. There are also fixed direction configurations, to send any packet independently of its fields. **Figure 3** shows the whole system working in the AC_WORKLOAD configuration.

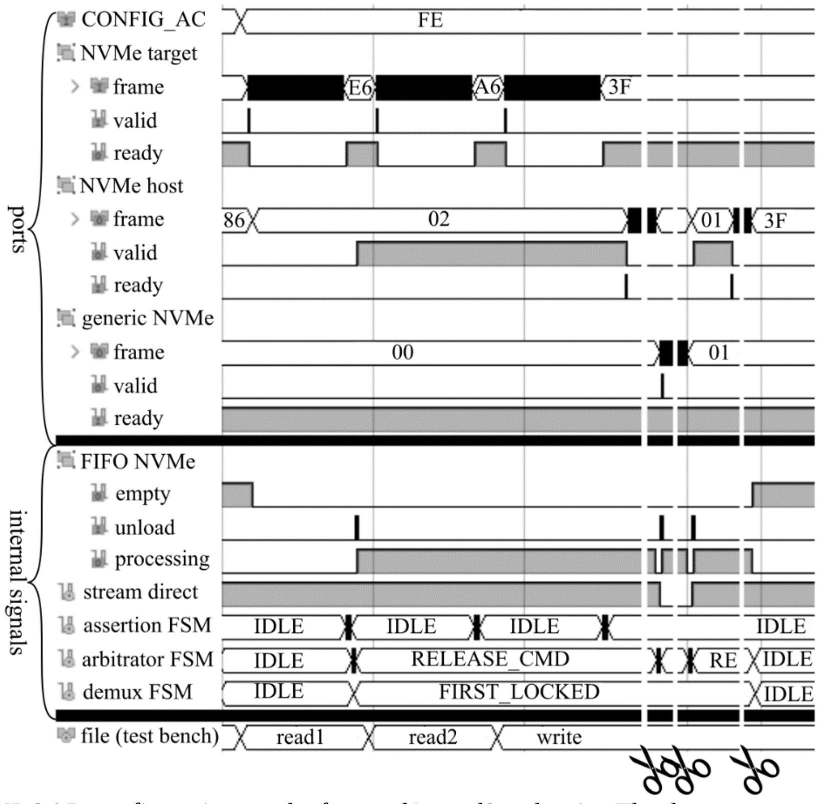

Fig. 3.AC_WORKLOAD configuration results for *read1*, *read2* and *write*. The three commands are buffered in the *FIFO NVMe*, displaying the switch's capability to hold the stream. The *read1* and *write* commands are sent to the SSD, whereas *read2* is sent to the *generic NVMe*, as expected. As indicated by the scissors, the figure was cut to present the most relevant parts of the simulation

IV. Conclusion

In this paper, we have proposed a solution for shortening the data semantic gap between host and SSD by allowing the host to gain more control over the SSD, without the burden of software processing. This has been accomplished through an NVMe-based switch with criteria configuration. This criterion can be configurated for different workloads at run-time via the external interface. The switch's proof-of-concept has been implemented in VHDL and validated in Vivado® Simulator 2019.1. The NVMe streaming standard facilitates the flow of packets throughout the design and it is the basis for the whole architecture. The results portrayed that the proposed design provides a flexible and expandable architecture that can be easily altered and/or increased in its functionality. The architecture also allows for other NVMe-based systems manipulation to run in parallel. Furthermore, in this design, any path may block the flow of packets without data loss. Thus, we have proved possible to arbitrate the flow of NVMe commands in hardware. The arbitration is configured for different workloads by the CPU, without in-datapath software intervention.

V. Future Work

In the future, we will run this design in a FPGA, alongside others functionalities such as NVMe *field capturing* and *opcode counting* for analysis purposes. This will allow us to make statistics out of the SSD. These statistics will serve as a tool for future research. A *generic NVMe* will be created to contain a cacheable address region with an automatic content flush, thus, extending the life-span of the SSD by reducing the number of writes in the flash cells. This will extend the life-span of the SSD by reducing the number of writes in the flash cells. This implementation will be tested with the *ext4* FS in journaling mode. This work is a subset of an ongoing project that targets a complete solution for NVMe-based SSD devices for industry and research.

References

[1]. NVMe Express Workgroup: 'NVM Express Revision 1.3' https://nvmexpress.org/wp-content/uploads/NVM_Express_Revision_1.3.pdf, accessed December 2019

[2]. Schroeder, B., *et al*.: 'Flash Reliability in Production: The Expected and the Unexpected', 15th USENIX Conference on File and Storage Technologies, Santa Clara, USA, February 2017, pp. 67-80, isbn: 978-1-931971-28-7

[3]. Stratikopoulos, A., *et al*.: 'FastPath: Toward Wire-speed NVMe SSDs', *International Conference on Field-Programmable Logic and Applications*, July 2018, pp. 170-177, doi: 10.1109/FPL.2018.00036

[4]. Yang, C. *et al*.: 'Reducing Write Amplification for Inodes of Journaling File System using Persistent Memory', *Design, Automation & Test in Europe Conference & Exhibition*, March 2019, pp. 866-871, doi: 10.23919/DATE.2019.8715068

[5]. Cai, Y., *et al*.: 'Error characterization, mitigation, and recovery in flash-memory-based solid-state drives', *Proceedings of the IEEE*, 2017, **105**, (9), pp. 1666-1704, doi: 10.1109/JPROC.2017.2713127

[6]. Kim, K., *et al*.: 'Effect of field oxide structure on endurance characteristics of NAND flash memory', *Electronic Letters*, 2014, **50**, (10), pp; 739-741, doi: 10.1049/el.2014.0522

[7]. Intelliprop Inc.: 'NVMe to NVMe Bridge' https://www.xilinx.com/products/intellectual-property/1-mpymke.html, accessed March 2020

[8]. Adjari, M., *et al.*: 'A Scalable HW-based Inline Deduplication for SSD Array', *IEEE Computer Architecture Letters*, September 2018, **17**, (1), pp. 47-50, doi: 10.1109/LCA.2017.2753258

[9]. Li, D., *et al*.: 'CISC: Coordinating Intelligent SSD and CPU to Speedup Graph Processing', *17th International Symposium on Parallel and Distributed Computing*, June 2018, pp. 149-156, doi: 10.1109/ISPDC2018.2018.00029

[10]. Jun, S., *et al*.: 'GraFBoost: Using Accelerated Flash Storage for External Graph Analytics', *ACM/IEEE 45th Annual International Symposium on Computer Architecture*, June 2018, pp. 411-424, doi: 10.1109/ISCA.2018.00042

[11]. Zhang, J., *et al*.: 'Design and Implementation of Optical Fiber SSD Exploiting FPGA Accelerated NVMe', *IEEE Access*, October 2019, **7**, pp. 152944-152952, doi: 10.1109/ACCESS.2019.2947181

Vinícius G. Linden,etal."Enabling Fast Decision Criteria in NVMe SSDs through NVMe-based Switch." *IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)*, 15(2), (2020): pp. 54-58.