POLITECNICO DI TORINO

Master degree course in Electronic Engineering

Master Degree Thesis
in
Integrated System Architecture

Next Generation Hardware Acceleration opportunities in Data Centers

Supervisor
prof. Maurizio Martina

Candidate
Giuseppe Dongiovanni
Mancino 245205

External Supervisor
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD)
prof. Alberto Dassatti

Anno Accademico 2019-2020

To my beloved family and friends

Abstract

The world has entered in the Era of Data. The role of data centers is more and more relevant due to the increased amount of data produced that have to be computed and stored. Therefore the scientific and engineering interest focuses on the research of new technologies to increase the computational power and storage capacity of data centers, and at the same time on the necessity to reduce their energy footprint.

Some of the available technological solutions, that can be used to meet the requirements for performance and energy efficiency, are presented in the first part of this thesis. They are the well-established NVMe protocol, designed to become an industrial standard that exploits the performance given by the PCIe it leverages, and the computational storage. The combination of these two technologies is the enabler framework for the near data computational paradigm also known as smart storage. The idea is simple: move the computing close to the data reducing the data movement, main source of energy dissipation, without compromising in processing performance.

Then the attention is focused on the development of a computational storage based on the NVMe protocol: the prototype is built on a Xilinx FPGA board, on the basis of an open-source project of an NVMe SSD controller.

The created prototype represents a scalable and standard compliant solution: it has been developed to experience and explore capabilities of the used technologies and standard, and the possible benefits that they can provide to data centers. Extensive benchmark tests have been carried out to characterize the device and discover its performance limits, in terms of both data rate and latency. At the same time, a general test application has been integrated in the prototype to evaluate, as real-world example, the complexity that the deployment of hardware accelerators involves.

Acknowledgements

The candidate would like to warmly thank the REDS institute, attached to the Department of Information and Communications Technology (ICT) of the HEIG-VD, for the resources provided, the professor Dassatti and the whole team for their guidance and help during this months. Moreover, the candidate would like to thank Mr. Kibin Park, one of the PhD students that works at the OpenSSD project, for his help in the early stages of the thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Despite never being in a data center, we live in a data-driven society and are depend on the services provided by data centers, almost as much other primary services, such as the water supply.

Every time we use the social networks or check our bank balance, or we create a back up of our data in a cloud, we are interacting with a data center: nowadays living without their existence seems to be nearly impossible.

A data center, or also called server farm, is a structure that hosts a large number of networked servers, routers and storage. It is used by governmental organizations and companies for remote storage and large-scale data handling and processing, and operates 24/7 to provide secure and continuous service[1].

The requirements that a data center should meet to provide an optimal service are:

- High performances (in terms of latency, IOPS and bandwidth);
- High memory capacity, to match the increase of data that have to be stored and/or computed;
- Non-volatile capabilities, to protect data even in case of power outage (backup server);
- Lower power consumption, to reduce costs for energy supply and cooling;
- Efficient management, to use the resources in the best way adapting to the workload;
- Flexibility, to easily deploy new technologies and/or application while, at the same time, leaving room for growth.

Through the years, the required processing capabilities of the data centers increased in order to match the increase in demand of bandwidth and computing.

This increase in performance can be provided in two main ways:

- increasing the workload on server processors, resulting in an excess of consumed power and an increase in temperature;
- increasing the elasticity, the ability to respond to workload changes, through the over-provisioning: underutilising hardware, in order to be able to meet peaks in demand, is not sustainable since facility consumes a lot of energy even when idle[2].

However, as demand for Internet traffic grows exponentially, the information and communications technology could lead to an explosion in energy consumption if eco-sustainable strategies are not implemented.

As shown in Figure 1.1, the data centers use more than one-third of the worst-case expected consumption for ICT[3].

The Storage Networking Industry Association (SNIA) has been working to search for improvements to increase the energy efficiency of data centers, regarding server and storage devices, through its Green Storage Initiative. Companies are now paying more attention to the type of technology they are going to deploy. As a matter of fact, they are in need of lower power consumption as much as higher performance and additional capacity [4].

1.1 Possible Solutions

To deal with these issues, current data centers must adopt new technologies and resources.

A first solution is to exploit hardware accelerators to offload the processors for different tasks, resulting in lower power consumption and higher energy efficiency: in addition, they can offer higher throughput and lower latency compared to common server processors[5].

Common hardware accelerators can be divided in 3 main groups, based on the platform used for their integration[6]:

- ASIC: integrated circuit designed for a specific application, improving the overall system speed as it focuses on performing just only or few functions;
- GPU: Originally designed for handling images, GPUs have more flexibility and programmability in respect to the ASIC: nowadays they are able to support different application with intensive computing.

  This specialization results into GPUs consisting of a large number of simpler processing cores with simple control logic, due to the handling of data that have little to no branching conditions or data dependences, leading to a very high parallelism.

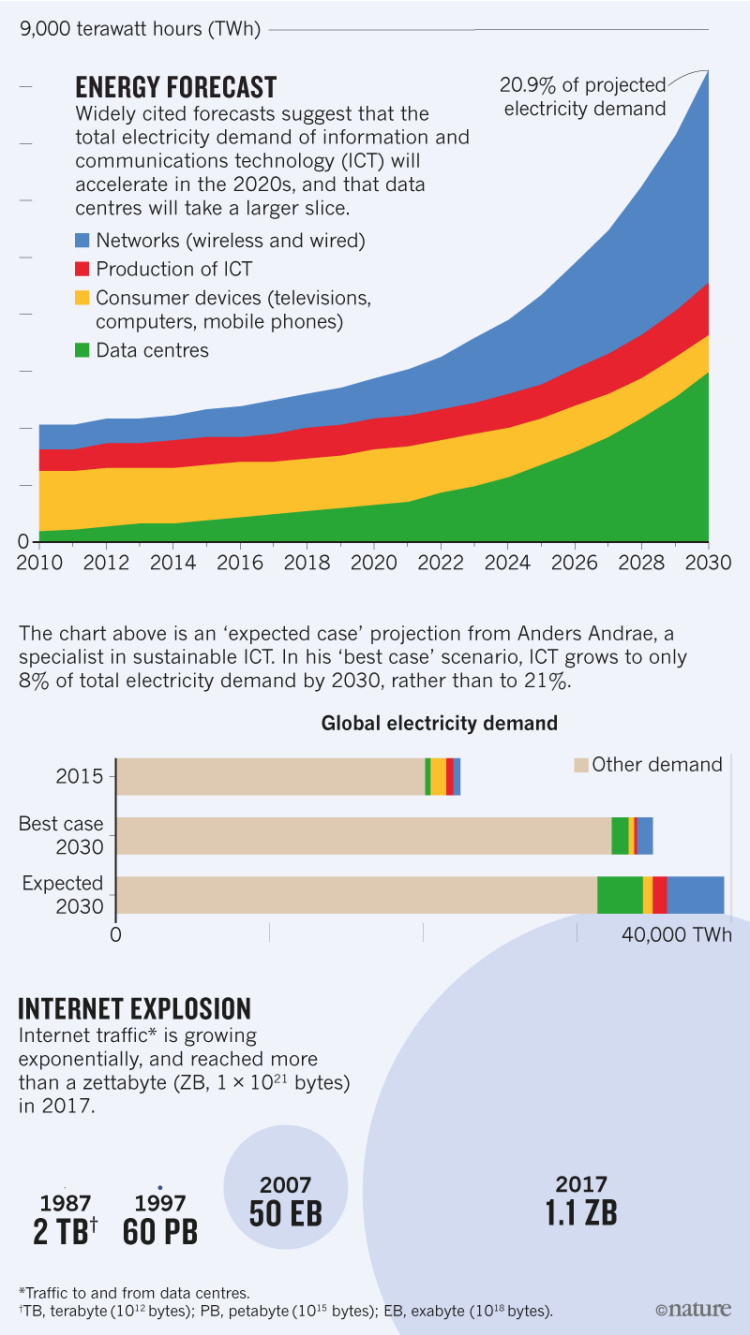


Figure 1.1: Energy Forecast - Nature, How to stop data centres from gobbling up the world's electricity

Different GPU organizations exhibit different performance and energy characteristics, according to the needs of the user: for example, high-end GPUs are mostly designed for hardware acceleration, with an architecture that can

be more complex internally, but with less general diversity thanks to the organization in cluster (GPC), large number of cores and vast on-chip memory resources, useful to achieve high performances.

- FPGA: as opposed to the fixed design of the ASIC, FPGAs consist of an array of logic blocks, DSPs, on-chip BRAMs, and routing channels, resulting in an extreme flexibility of configuration. It can be customized to fit the needs of specific computations, with an efficiency that can reach the one of a custom architecture.

  In FPGA, custom data paths can transfer data directly between computing units and exploit data locality thanks to the distributed on-chip BRAMs, which results in less access to the external memory.

  This extreme flexibility, however, comes at a cost. First, FPGAs are not as space efficient as its ASIC or GPU counterparts, which have higher internal density. Then, with FPGAs, developers must verify that their design complies with timing and space requirements.

  Finally, FPGA require, lacking a fixed structure, the use of hardware synthesis tools to create a configuration file that defines the architecture logic on the device.

  An alternative approach to realize reconfigurable hardware accelerators is the use of architectures with higher abstraction level, such as CGRAs (Coarse-Grained Reconfigurable Architecture), offering shorter compilation times but lower configuration flexibility.

- FPGA w/ SoC: Recent commercial SoCs also include reconfigurable hardware fabrics, which are able to implement custom functions, such as hardware accelerators.

  With these devices it is possible to achieve hardware/software solutions in a single chip, with the need to apply codesign approaches and hardware/software partitioning. This means that the developer needs to separate the application in parts that run on the CPU and those in the reconfigurable hardware.

  The development of embedded applications targeting these reconfigurable systems is speed up by the progress of the hardware synthesis tools, shortening the time to market of the final product.

However, moving data can be more expensive than processing them, or it can be even worse if the transfers take place through an interconnect network. In fact, transferring data from storage systems to processors (and viceversa) is one of the major obstacle toward meeting performance and energy efficiency requirements.

To remove this obstacle, it is necessary to change a basic concept from “move data to the process” to ”move process to data”. This approach, called In-situ processing

(ISP) and already present in framework such as Hadoop, can be fully exploited thanks to the modern solid-state drives (SSDs) architecture and the availability of powerful embedded processors, leading to the creation of computational storage. The embedded processor has access to the data stored in the NAND flash memory through an high speed and low power bus, avoiding both workload for the processor and transfer from and to the memory and so reducing the energy consumption[7].

Computational storages and how they are implemented will be dealt with in detail in section 2.3.

In order to fully exploit SSDs potential, the PCI Express, the interconnect bus that is closest to the host CPU, has been widely deployed in the data center, due to the ability to provide higher bandwidth and lower latency. The PCIe is currently one of the fastest I/O data highway available for storage and capable of supporting fast processors with high number of cores and heavy traffic[8]. However, the software protocols, that defines the traffic flow, need improvement to match the high performance of the PCIe.

A new storage protocol, that exploits the PCIe and is called Non-Volatile Memory Express (NVMe)[9], has been designed, with its first specifications published on March 2011, to take full advantage of the capabilities of SSDs.

NVMe allows applications that require the high performance servers and access to local storage via fast I/O data highways to reach their performance potential.

NVMe provides what data centers and hardware accelerators require. In particular:

- Low latency, given by the direct CPU connection;
- High throughput;
- Low CPU overhead;
- Multi-core awareness, and so capable to withstand the hundred of cores in data centers;
- Management at scale.

A Datacenter needs to manage everything on a single network: if the management is not efficient and network traffic is heavy, the management would take a part of the bandwith that could be sold to the costumer, resulting in a loss of profit. A simple network, easy to control and analyse, is needed.

NVMe can easily manage any number of storage devices and hardware accelerators, if they are all NVMe devices, thanks to NVMe Admin commands, that can for example update the firmware, format or repair the drive and management devices.

From the point of view of the performance, NVMe protocol reduces the wait time and latency through a more effective use of the PCIe data highway. Infact, while SATA only allows a single command queue that holds 32 commands, NVMe enables 64K queues with 64K commands each[10].

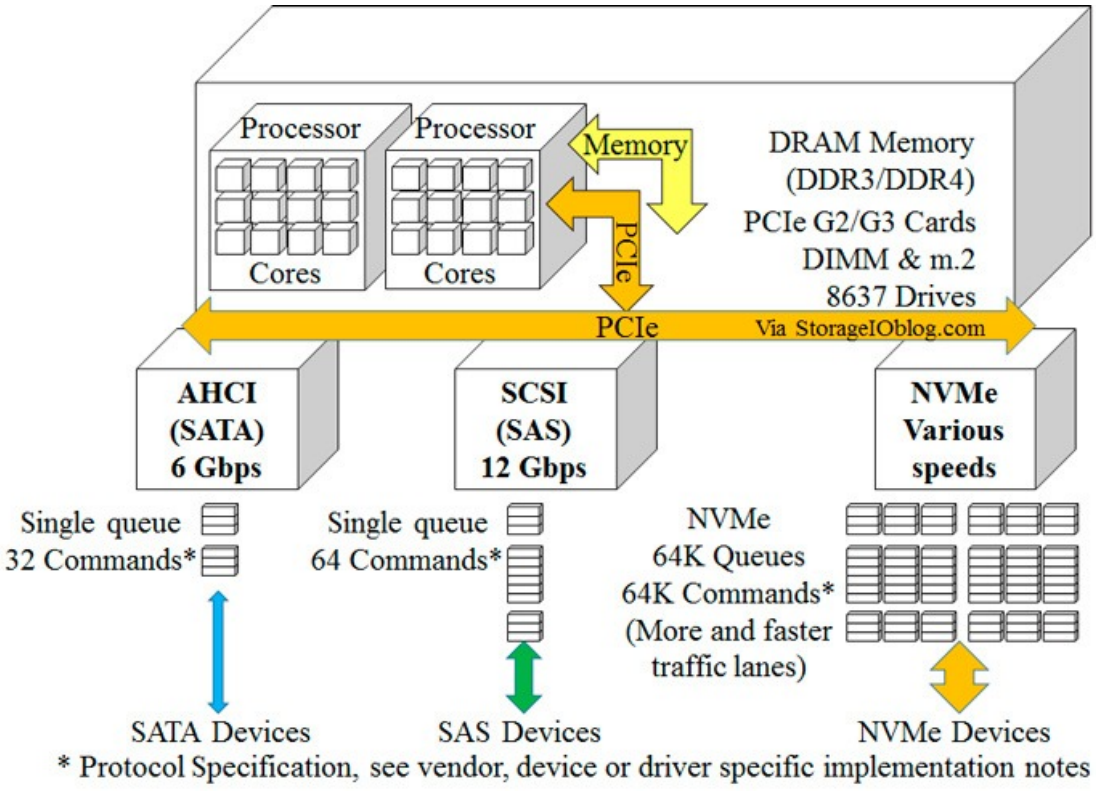


Figure 1.2: Comparison between SATA,SAS and NVMe - StorageIOblog.com

NVMe has been designed with more and deeper queues, supporting a larger number of commands in those queues. In this way the SSD are able to receive a large number of commands, exploiting the internal parallelism, and better optimize command execution, achieving much higher concurrent IOPS.

Moreover, the specifications of the NVMe protocol includes many features for power management: in particular, the presence of non-operational power states is of great significance to further reduce the idle power of the devices[11], thus providing an opportunity to reduce the over-provisioning energy impact.

Summing up, NVMe provides both flexibility and compatibility, provides lower latency and allows higher number of concurrent I/O operations to be completed, thanks also to the PCIe interface it leverages, while having higher power efficiency. And last but not the least, the NVMe protocol has been developed with the idea of creating a free and standard that would be equal for everybody: for all these reasons, more and more large and small companies invest in the development of the NVMe protocol.

Internet-of-Things and, in a near future, the Artificial Intelligence will totally change the technology landscape: their requirements for data and processing power are massive, leading to a huge increase in power consumption. NVMe is a first step

that can have a big impact on the future of ITC: shared solutions must be found in order to satisfy the demands but, at the same time, stop data centers and all the ICT infrastructure *from gobbling up the world's electricity*[3].

1.2 Thesis Structure

The thesis is organized in 4 main chapters. A background section follows this introduction and presents the relevant aspects of NVMe and Computational Storage. In particular, this first chapter describes in detail the reasons that led to the adoption of the PCIe and NVMe protocol. Then the discussion will move to the state of art of computational storages.

The third chapter describes all the steps that were necessary to create an NVMe hardware accelerator, starting from the base project Cosmos+ OpenSSD to the final hardware accelerator, and the results of the various tests.

Then the final chapter sums up the obtained results, presenting the encountered problems and the possible developments for future work.

Chapter 2

State of Art

2.1 PCI Express

PCI Express (Peripheral Component Interconnect Express) is a high-speed serial computer expansion bus standard, directly connected to the motherboard.

Its direct competitor was the Serial ATA (SATA)[8], a computer bus interface that connects host bus adapters to mass storage devices: initially designed for interfacing with hard-disk drives, the SATA interface became the IOPS bottleneck with the coming of the new solid-state drives.

SATA was not able, in order to catch up with increasing speed of the SSDs, to overcome the upper-limit of 6Gb/s (about 750MB/s), without any major and time consuming changes and with solutions that would be less power efficient and more expensive.

Thanks to better performance, due to low latency and high bandwidth, PCIe is becoming the predominant interface for storage devices[12].

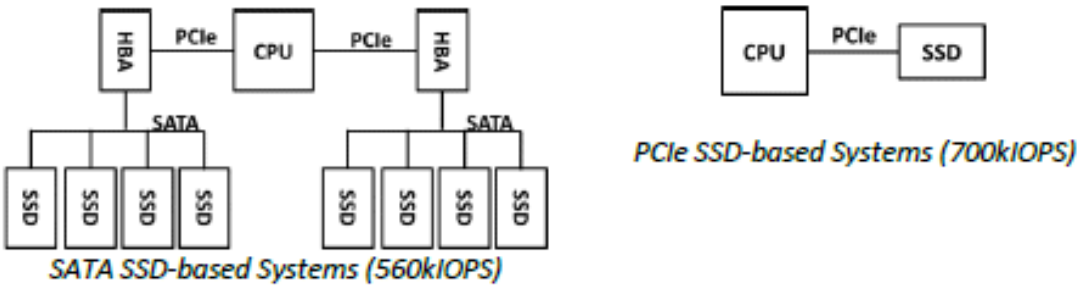


Figure 2.1: Comparison between SATA and PCIe in number of IOPS - Design & Reuse

However, nowadays SATA and PCIe still coexist: for example, SATA SSDs has lower cost and better performance than the SATA HDD, making it widely used in

consumer applications; on the other hard, PCIe SSD are more expensive but are much more performing, meeting the requirements of industial applications as the data centers.

Defined by its number of lanes, the PCIe has undergone several revision, reaching very high throughput with the newer versions, as it can be seen in the Figure 2.2.

PCI Express link performance[35][36]

PCI Express version	Introduced	Line code	Transfer rate[i]	Throughput[i]				
				×1	×2	×4	×8	×16
1.0	2003	8b/10b	2.5 GT/s	250 MB/s	0.50 GB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s
2.0	2007	8b/10b	5.0 GT/s	500 MB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s	8.0 GB/s
3.0	2010	128b/130b	8.0 GT/s	984.6 MB/s	1.97 GB/s	3.94 GB/s	7.88 GB/s	15.75 GB/s
4.0	2017	128b/130b	16.0 GT/s	1969 MB/s	3.94 GB/s	7.88 GB/s	15.75 GB/s	31.51 GB/s
5.0	2019	128b/130b	32.0 GT/s[ii]	3938 MB/s	7.88 GB/s	15.75 GB/s	31.51 GB/s	63.02 GB/s
6.0 (planned)	2021	128b/130b	64.0 GT/s	7877 MB/s	15.75 GB/s	31.51 GB/s	63.02 GB/s	126.03 GB/s

Figure 2.2: PCIe Generations - Wikipedia, PCI Express

NAND chips in consumer SSDs usually have a bus bandwidth of around 200MB/s: consisting of 4-8 chips, a typical SSD can reach for up to 1.6GB/s transfer speeds, that can easily supported with a PCIe Gen2.0 x4.

2.2 NVM Express

However, PCIe as storage protocol is not enough: Non-Volatile Memory Express is a communication transfer protocol, designed to address the needs of both Enterprise and Client systems, that has been especially developed for PCIe-based SSDs, supporting all form factors (U.2, M.2, AIC, EDSFF) and providing the capabilities to meet the demands of cloud, internet portal data centers and other high-performance computing environments.

2.2.1 Overview

The benefits that NVMe provides are[12, 13]:

- Driver standardization: NVME is open source and supported by the major operating systems, and future NVMe features, such as vendor specific commands, can be integrated in the standard driver;
- Performance increase, since a multi-queues management and submission of practically unlimited number of commands are possible, as shown in Figure 1.2, and the SATA bottleneck has been removed;

- Scalability, with headroom for improvements;
- Optimized register interface and command set, to simplify host software and device firmware;
- Reduced power consumption, resulting in a lower Total Cost of Ownership (TCO) and carbon footprint.

Another important feature for data centers is the possibility to exploit the parallel processing capabilities of multi-core processors: the ownership of queues, their priority and the arbitration mechanisms can be shared between different CPU cores, achieving higher IOPS and lower data latency.

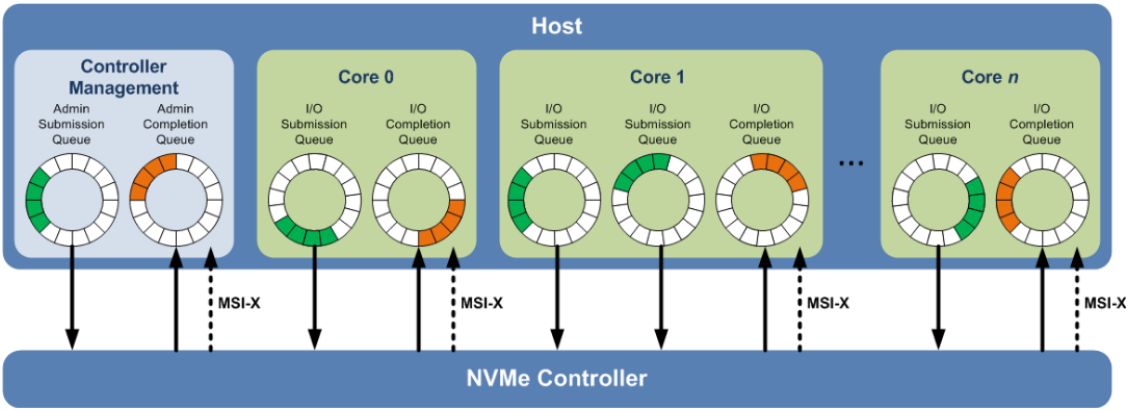


Figure 2.3: General diagram of a Multi-core Host NVMe Management - IDF13, Optimized Interface for PCI Express SSDs

As shown in Figure 2.3, each core has one or more I/O submission queues, a completion queue, and the MSI-X interrupt.

While the cores manage the I/O commands, in the submission and completion queues of the controller management takes place the management of the Admin commands, that are used to obtain informations about the NVMe device, modify its configuration or create the Submission and Completion Queues.

A write access from the host to the NVMe controller of a device can be described in the following way[15]:

- The host submits new commands in the Submission Queue and sets the Submission doorbell register (tail/head mechanism) of the NVMe controller in order to inform it that there is a new submission queue ready.
- The NVMe controller fetches the commands, including all the necessary information (source and destination address, data size, priority, ect.), from Submission Queue into the host memory processes them;

- The NVMe controller manages the data transfer and write the completion of the commands in the host Completion Queue. As all the commands are executed, the NVMe controller generate the MSI-X interrupt;
- Finally the host processes the completition and updates the Completion doorbell register.

2.2.2 NVMe Hierarchy

The subsystem of an NVMe device subsystem consists of different elements:

- Namespaces – array of logical blocks;
- NVM Sets – groups of one or more namespaces;
- Endurance Groups – consisting of a fixed or variable number of NVM Sets;
- Domains – consisting of endurance groups, one or more NVMe controllers, etc.

The two type of endurance group identify two methods of management [16]:

- fixed capacity management, as shown in 2.4 for drives to satisfy the requirements of an hyperscalar system. If the number of namespace and NVMe set for endurance group is fixed to 1 (NVMe sets are not needed anymore), the host will have a high workload but will be able to optimize the wear leveling of the storage;
- variable capacity management for an higher customization.

An NVMe namespace is a storage volume of non-volatile memory formatted into logical blocks. Its range from the LBA 0 to LBA (n-1), where LBA stands for Logical Block Address and n is the size of the namespace, and is backed by some capacity of non-volatile memory.

Namespaces may be created and deleted using the Namespace Management of the NVMe Controller of the device, and each namespace is indipendent of other namespaces.

An identifier (NSID) is provided by the controller, using Namespace Attachment Commands, in order for the host to have access to a namespace. A namespace, with a given NSID, can be accessed by multiple NVMe controllers[17], as shown in Figure 2.5.

2.2.3 NVMe-oF

The NVMe over PCIe is limited to the local use. Therefore, the natural evolution of the NVMe protocol is the possibility to access an external NVMe device through a network (Figure 2.6).

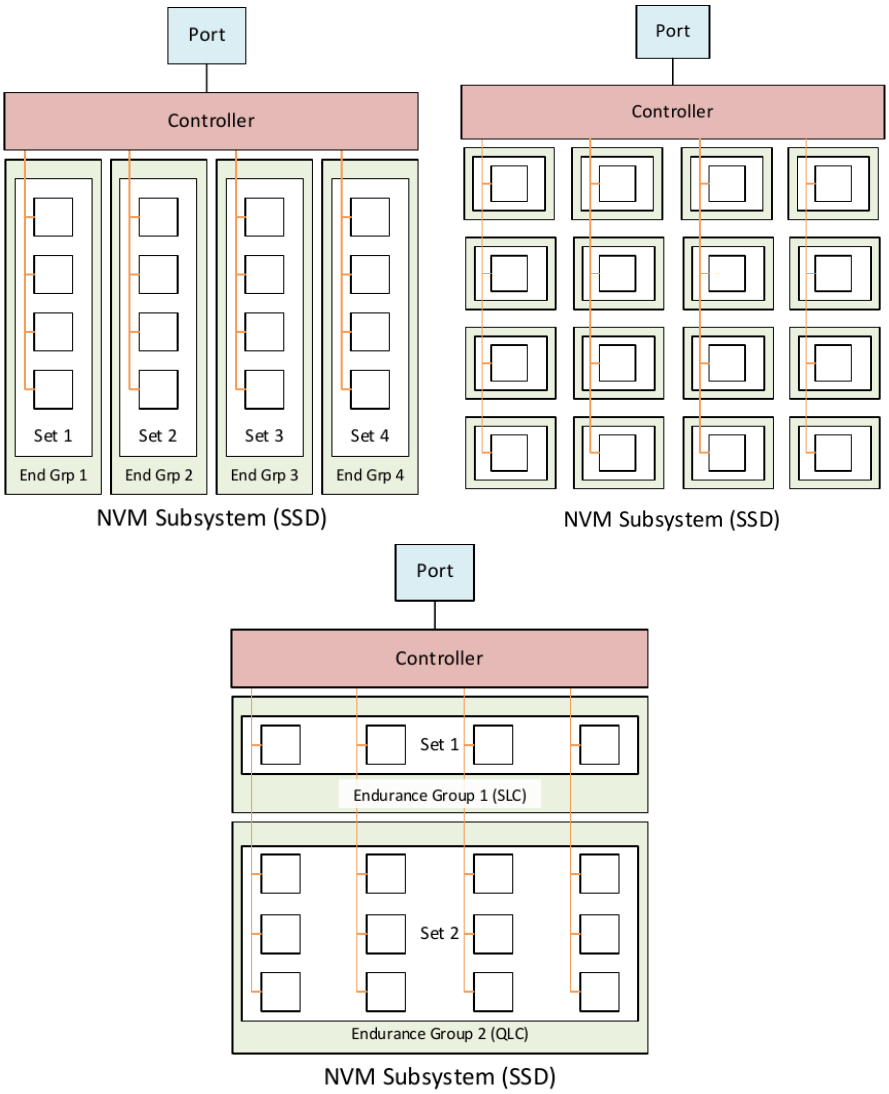


Figure 2.4: NVMe Hierarchy Types - SDC19, Managing Capacity in NVM Express SSDs

This evolution allows to access a shared network, keeping the benefits of the NVMe protocol. The additional advantages that the NVMe-oF provides are:

- sharing and provisioning;
- data and workload migration;
- better efficiency;
- better data protection.

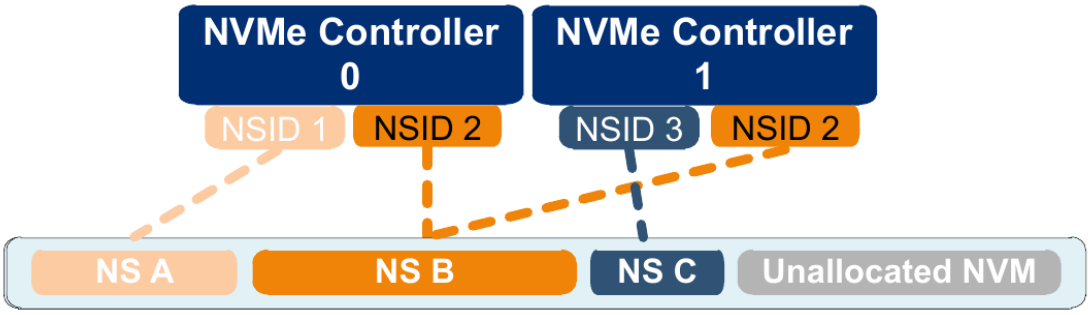


Figure 2.5: Namespace Management and Attachment Commands - SNIA DSI Conference 2015, Creating Higher Performance Solid State Storage with Non-Volatile Memory Express

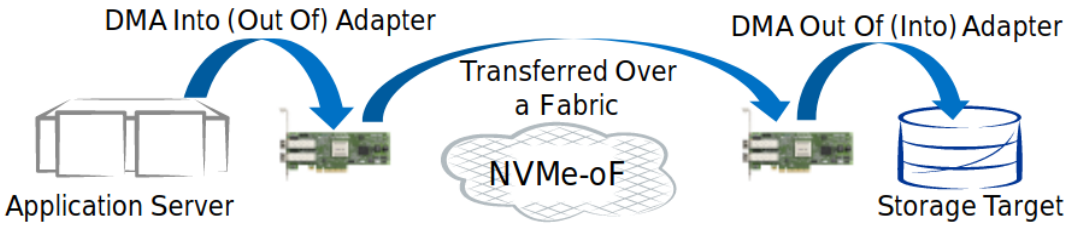


Figure 2.6: NVMe-oF General Scheme - NVM Express, NVMe Over Fabrics

The direct consequence of the NVMe-oF is the further consolidation of the position of the data centers as single shared and efficient infrastructure for both SAN (storage area network) and DAS (direct-attached storage). Moreover, being able to access any NVMe device over a network, it is even possible to use hardware accelerator to transfer part of the local CPU workload, that can even be performed in a more efficent way[20].

NVMe-oF is transport agnostic: it means that NVMe-oF supports all transport protocols, like RoCE, TCP and Fibre Channel.

For Enterprise Storage, Fibre Channel fabric is the best choice: the Fibre Channel Protocol is stable, reliable, mature, very efficient and high speed, and it offers consistently high performance.

However, not all organizations and costumers can use means like Fibre Channel or RoCE: the TCP is the most diffused and simple, does not require special hardware or networks, being based on Ethernet fabric, and can provide high performance if the network design is properly set up[21, 22].

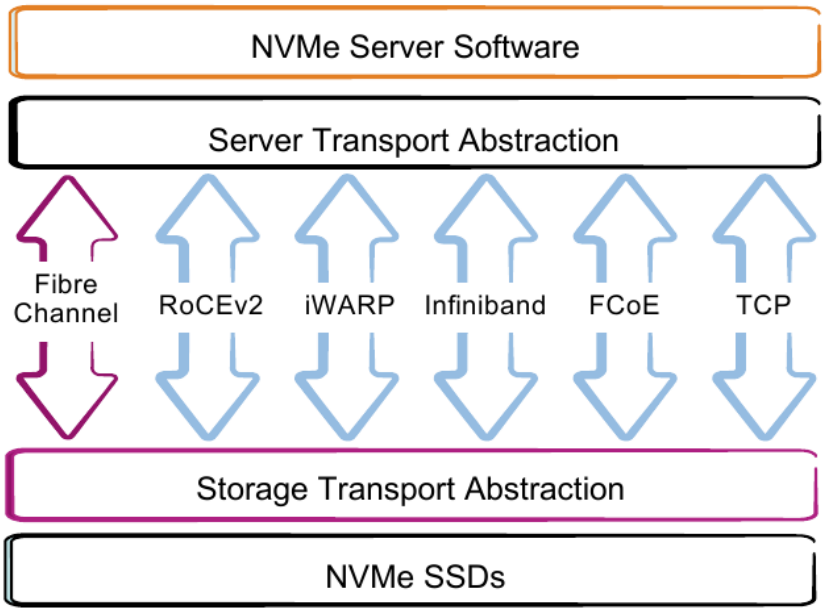


Figure 2.7: NVMe-oF: Protocol Options - Flash Memory Summit 2020, NVMe-oF™ Enterprise Appliances

2.3 Computational Storage

A computational storage is an architecture that couples hardware accelerators to the storage. The benefits of this architecture are

- CPU offload, with the reduction of the host processor workload;
- increase in performance, due to the hardware acceleration;
- near storage computing, with the reduction of the amount of data that must move between the storage plane and the compute plane, increasing efficiency.

As future prospective for the NVMe protocol, there is the possibility to create a new namespace type, the NVMe Computation Namespaces. Operating Systems will be able to treat these computation namespaces in a different way in respect to storage namespaces. For instance, the computational storage will be seen as </dev/nvmeXcsY> and, if possible, the user-space will not know or care if it is local (over PCIe) or remote (over Fabrics)[24, 25].

Computational storage, generally referred as CSx, can be mainly implemented, as shown in Figure 2.8, in three different ways:

- Computational Storage Processors (CSP);
- Computational Storage Drive (CSD);

- Computational Storage Array (CSA).

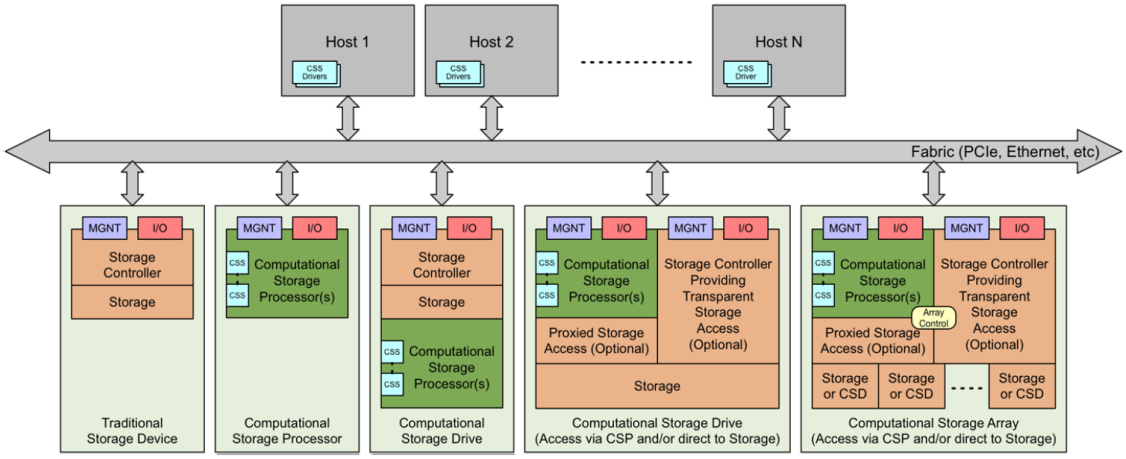


Figure 2.8: Different implementations of Computational Storage - SNIA Computational Storage 2019, What happens when Compute meets Storage?

Any type of computational storage provides computational storage services (CSS) that can be fixed (FCSS) or general/programmable purpose (PCSS)[26].

The CSP is the basic implementation of a computational storage: it is a component that provides CSS to a storage, but does not provide any persistent storage internally. The CSD, instead, is a component that provides CSS and persistent storage, with the possibility to access directly both or only one between the CSP and the storage. Finally, the CSA is a collection of computational storage drives, computational storage processors and/or storage, managed by a control software.

Moreover, Peer-to-Peer (P2P) operations can be achieved using computational storage.

As shown in Figure 2.9, the usual route followed to perform a data computation has the following step:

- copy data from the storage to the DDR;
- compute data in the CPU;
- write back the data in the storage.

Using computational storage, to process the data instead of the CPU, and an NVMe Controller Memory Buffer(CMB), to store the Submission and Completition Queues, it is possible to compute a huge amount of data without any workload for the host CPU, apart from the usual tasks like security.

The benefits that can be achieved are[27]:

- reduced data movement,

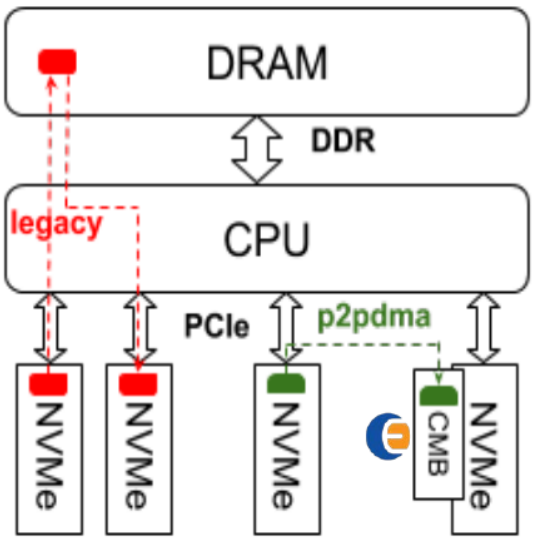


Figure 2.9: Peer-to-Peer with an NVMe CSx Device - MSST 2019, How NVM Express and Computational Storage can make your AI Applications Shine!

- CPU offload of processing and DMA traffic;
- power efficiency.

2.3.1 Commercial Hardware Accelerators

Some of the companies involved in the development of the NVMe protocol works on creating computational storage.

The product that is going to be presented has been create by Eideticom, founded in 2016 with the only objective of developing Computational Storage solutions for cloud and data centers. It is called NoLoad® CSP[1], the first nvme-based one to be created in August 2019, as certified by the UNH-IOL.

Eideticom's NoLoad CSP purpose is to accelerate storage and intensive workloads, reducing the utilization of the host CPU. The CSP is Plug-and-Play: it utilizes drivers that are available on all major operating systems.

Moreover, it supports all types of form factors, P2P and CMB, NVMe-oF and provides different type of computation services, like compression, encryption or data analytics.

Different demonstrations were carried out by partners like Bittware and Xilinx, respectively with the FPGA platforms 250-U2 and Alveo U50.

[1]Eideticom NoLoad: https://www.eideticom.com/uploads/images/NoLoad_Product_Spec.pdf

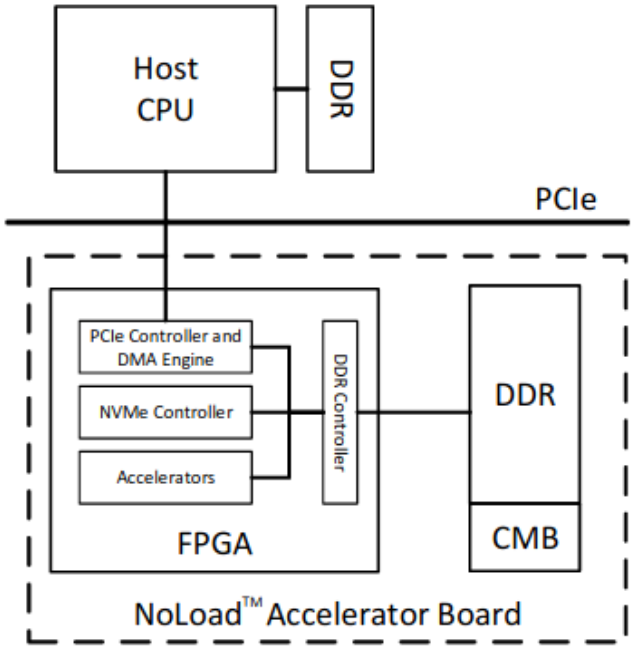


Figure 2.10: NoLoad® CSP - SNIA SDC2017, An NVMe-based Offload Engine for Storage Acceleration

On the other hand, there are several vendors developing CSDs not based on NVMe. An example of FPGA-based CSD is the ScaleFlux 2000 Series[2] over PCIe, or the Samsung SmartSSD drive[3], produced by Samsung.

However, while the SmartSSD memory is managed by a Samsung SSD controller, the ScaleFlux 2000 is practically an open-channel SSD: the flash translation layer (FTL) is not implemented in the FPGA, but it runs in the software on the host system[29].

[2]ScaleFlux 2000 Series: http://scaleflux.com/product.html

[3]Samsung SmartSSD: https://www.nimbix.net/samsungsmartssd

Chapter 3

Computational Storage Project

In this chapter all the steps necessary to build an FPGA-based NVMe Computational Storage are going to be analyzed. However, due its complexity, it is necessary to first obtain the NVMe communication interface: an NVMe Controller. Therefore, the open source project Cosmos+ OpenSSD has been chosen as the basis for our goal due to matching the requirements.

3.1 Cosmos+ OpenSSD Project

The first step consists of analysis and adaptation of the OpenSSD project from the original custom board to the Xilinx Zynq-7000 SoC ZC706.

3.1.1 Overview

Cosmos OpenSSD is an open source and FPGA-based SSD controller project that has been developed since 2014 by the HYU ENC Lab of the Hanyang University in South Korea, with research and education purposes[30].

A first version of the project was based on Indilinx Barefoot, a SoC over SATA2.

The project version that will be analyzed is the Cosmos+ OpenSSD: developed in 2016, this version of the SSD controller supports the NVMe protocol. The project has been developed using Xilinx Developer Tools, Vivado Design Suite and SDK.

The custom Cosmos+ FPGA board has the following main features:

- FPGA Xilinx Zynq-7000 with a Dual ARM Cortex-A9 1GHz Core;
  - 1GB of DDR3;
  - AXI4-lite bus width of 32 bits;
  - AXI4 bus width of 64 bits;
- dual PCIe Gen2 x8 End-Points (Cabled PCIe Interface);

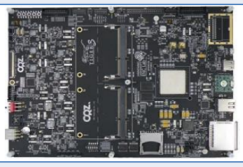
	Jasmine OpenSSD	Cosmos OpenSSD	Cosmos+ OpenSSD
Released in	2011	2014	2016
Main Board			
SSD Controller	Indilinx Barefoot (SoC)	HYU Tiger3 (FPGA)	HYU Tiger4 (FPGA)
Host Interface	SATA2	PCIe Gen2 4-lane (AHCI)	PCIe Gen2 8-lane (NVMe)
Maximum Capacity	128 GB (32 GB/module)	256 GB (128 GB/module)	2 TB (1 TB/module)
NAND Data Interface	SDR (Asynchronous)	NVDDR (Synchronous)	NVDDR2 (Toggle)
ECC Type and Strength	BCH, 16 bits/512 B	BCH, 32 bits/2 KB	BCH, 26 bits/512 B

Figure 3.1: OpenSSD project History - Cosmos+ OpenSSD 2017 Tutorial

- additional interfaces (JTAG,USB,Ethernet);
- up to 2 NAND Flash Modules, with 8 flash packages slot each.

In Figure 3.2 it is illustrated the internal system overview of the project.

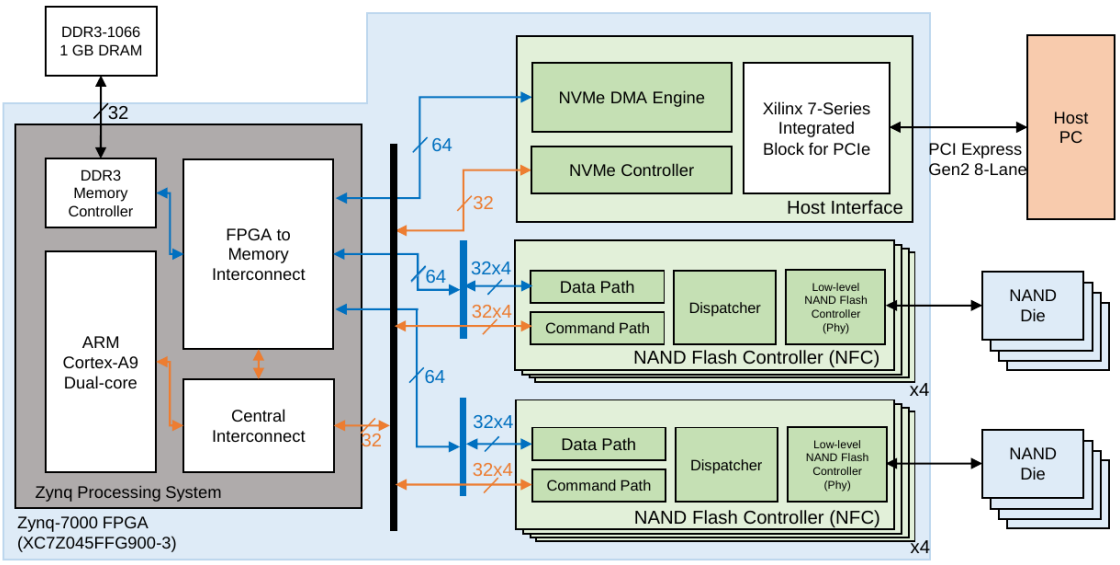


Figure 3.2: Cosmos+ OpenSSD project System Overview - Cosmos+ OpenSSD 2017 Tutorial

The Zynq processor is connected to the host through the Host Interface, called NVMe Host Controller, which is responsible of:

- handling of the data from the host to the buffer with a DMA engine;

- automated completion of the NVMe IO Command, without involving the Flash Transition Layer (FTL), that will be described later.

The NAND Flash Controller is the interface between the NAND Flash and the processor. It consists,as shown in the system block desing in Figure 3.3, of three different hardware IP blocks:

- Tiger4 NSC;
- Tiger4 Shared KES;
- V2NFC.

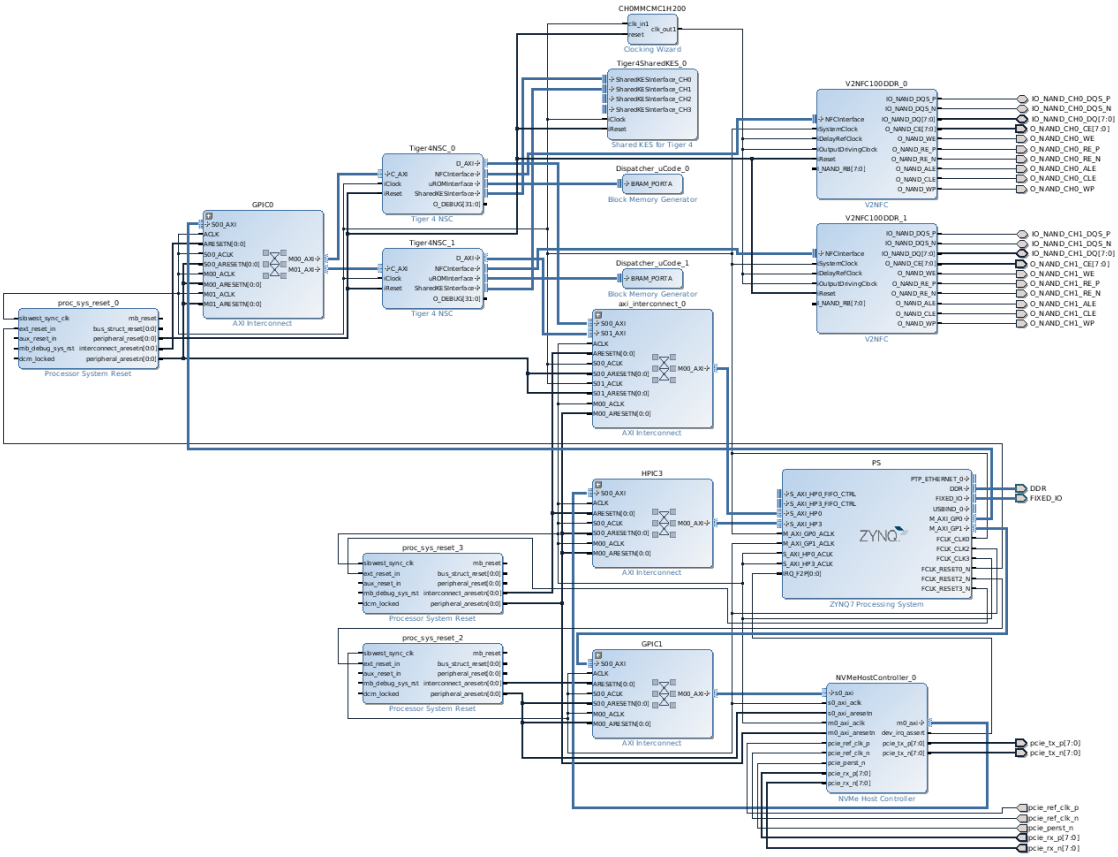


Figure 3.3: Cosmos+ OpenSSD project System Design

The IP Tiger4 NSC is the responsible of the handling of command and data from the processing system: the commands, consisting of information such as source and destination of the operation, are written by the firmware driver in the Tiger4 NSC registers and then elaborated.

The data, instead, undergo different manipulations: in particular they pass through a module responsible of the Error Detection and Correction, the Tiger4 Shared KES.

Finally, the data are handled by the V2NFC block, that physically performs the low-level I/O operations in the NAND Flash Modules.

The operations have to be scheduled in order to be performed on the right SSD package and die. As shown in Figure 3.4, each NAND module has up to 4 available channels, one every two packages, and each channel has 8 maximum ways, corresponding to the maximum number of connected dies. The number of channels is equal to the number of NAND Flash Controllers.

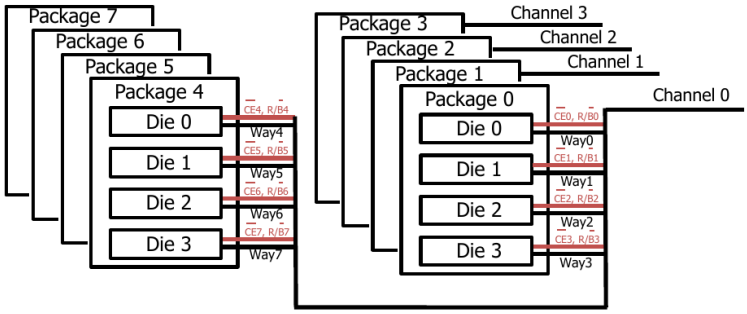


Figure 3.4: Nand Modules Organization - Cosmos+ OpenSSD 2017 Tutorial

In the first version of the project, Cosmos, the way scheduling was managed by the NFC block, while the channel one by the firmware Flash Transition Layer (FTL): with the Cosmos+ one, both channel and way scheduling is managed by the FTL, providing more flexibility.

The other main features of the FTL are:

- Least Recently Used (LRU) data buffer management;
- priority command scheduling, as shown in Figure 3.5, with the aim of enhancing the multi-channel and way parallelism;
- on demand garbage collection, triggered only when there is no more free user block in each die;

The garbage collection is needed to recover free blocks for write requests: a victim block with invalid data is selected, then the valid data are copied in a free block while the victim one is erased.

However, while supporting the garbage collection, the firmware does not support the wear leveling of the flash memory.

A schematic description of the firmware execution is shown in Figure 3.6.

In order to be executed, a received IO command is first transformed in Slice Requests, which number depends on the number of logic blocks requested. Then

Command	Priority
LLSCommand_RxDMA	0
LLSCommand_TxDMA	0
V2FCommand_StatusCheck	1
V2FCommand_ReadPageTrigger	2
V2FCommand_BlockErase	3
V2FCommand_ProgramPage	4
V2FCommand_ReadPageTransfer	5

Figure 3.5: Command Priority - Cosmos+ OpenSSD 2017 Tutorial

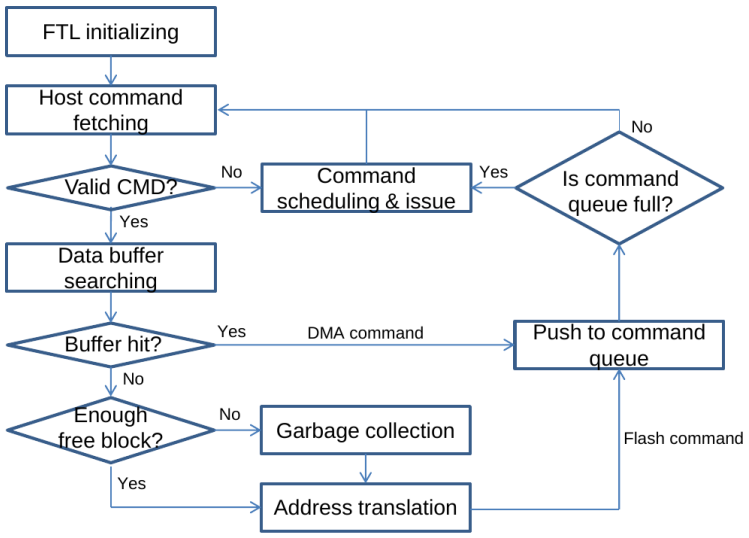


Figure 3.6: Firmware Overall Sequence - Cosmos+ OpenSSD 2017 Tutorial

each Slice Request undergoes a second transformation in DMA and, if there is no buffer hit for read operations, NAND Requests. These requests are organized in different queues: one for the free requests, three for the requests that are going to be executed, one for each aforementioned type of requests, and two for the blocked requests, either for buffer or row address dependencies. Then the requests are, if not blocked, finally scheduled and executed.

3.1.2 Project Adaptation

The available platform is the Xilinx ZC706: in comparison with the original custom board, it has:

- same FPGA Zynq-7000 SoC;

- 4-line Gen2 PCIe Connector, instead of the 8-lane of the custom board;
- no NAND module.

The first step to adapt the project is to modify the hardware[1]: there is no NAND module, therefore the entire NAND Flash Controller hardware is not necessary.

At the same time, the NVMe Host Controller has to be changed from the 8-lane PCIe to the 4-lane one: this can be done by modifying the configuration of the Xilinx PCIe Core IP. The result is shown in Figure 3.7. Likewise, the constraint files have to be modified or removed, in order to match the pinout of the Xilinx ZC706[31].

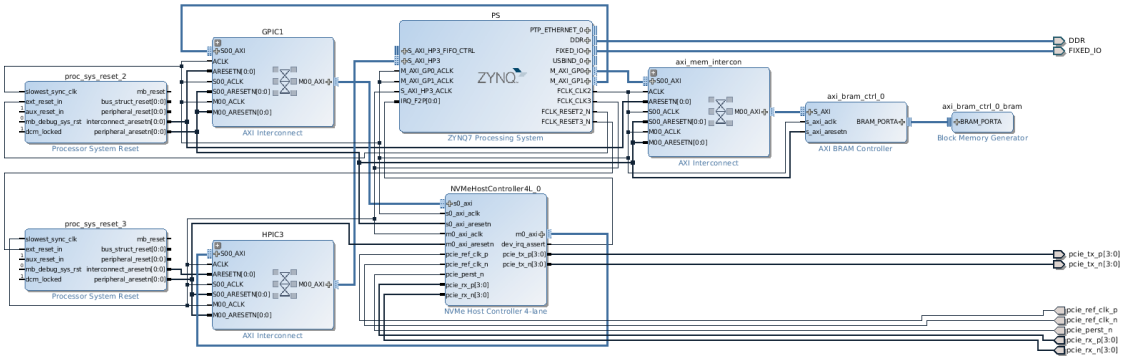


Figure 3.7: First Adaptation of the Cosmos+ OpenSSD project System Design

A BRAM Controller has been added to provide a destination address for the firmware channel, substituting the Tiger4NSC one: however, it has no active role in the operations.

After exporting the new hardware file (.hdf), it is necessary to create a new project with the new platform specifications.

Some modifications have also been made to the firmware:

- allocation of the memory arrays <MemSpace> in the DDR: the storage capacity is of 64 MB, due to the DDR already being used by the firmware FTL;
- different organization of the memory management unit (MMU) table and the memory mapping, given the presence of the memory array;
- variation to the memory dimensions, number of channel and way;
- bypass of the status check and ECC functions;

[1]This modified project and all the following ones have been uploaded on the GitHub repository <https://github.com/giuseppedongiovanni/nvme_comp_storage>

- replacement of the NAND operations, represented by the write operation of the commands in the Tiger4 registers, with the MemCpy function.

The memory array <MemSpace> is the replacement for the SSD: however, the available storage is much smaller than what the firmware expects. Varying the FTL configuration parameters, that should be left untouched, is necessary to avoid that important memory location useful for the firmware execution are corrupted.

3.1.3 Functionality Tests

The first test that has been carried out is the functionality one, consisting of a routine of power-up, partitioning and reset from both developer and host side, as shown respectively in Figures 3.8, 3.9 and 3.10. Then the data correctness, that corresponds to writing a file and reading it back, is verified in Figure 3.11 by the matching MD5 values (practically the digital fingerprint of a file).

Finally, it is possible to send some NVMe Admin Commands to verify the presence of the namespace and, as well, obtain information about the device. As it can be seen in Figure 3.13, the namespace id is set to 14740, equal to the storage capacity in terms of NVMe blocks (4096 bytes): this namespace, defined by the project creators, has been attached by the controller to the NSID 1, being the device seen as <nvme0n1>.

3.1.4 Performance Tests

Different performance tests are carried out: <dd> function and the software Iometer are used to evaluate IOPS and bandwidth; to measure the latency instead, timers are used both in the firmware and in a c-file on the host side.

The Figures 3.14 to 3.16 refer to the performed tests for the read operation.

The results obtained are unexpected: performing operations from a DDR to an host device through a PCIe bus should imply very high performance, in the order of GB/s for the bandwidth. Instead, the obtained one is around 100 MB/s, with only 7000 IOPS.

More information can be obtained from the analysis of the latency test results, shown in Figure 3.17.

The pie chart is divided in two main parts: the green one is related to the time spent in the device firware, while the orange one includes all the other contributions, in particular the host, the PCIe bus and the NVMeHostController of the device.

The firmware execution takes up the 73% of the total latency: in particular, the <MemCpy> function employs more than half of this time period to be performed, slowing the entire execution. The obtained speed is of about 150 MB/s, while a single-port DDR3, with a 32-bit bus-width at 533 MHz, has a maximum theoretical bandwidth of 4 GB/s.

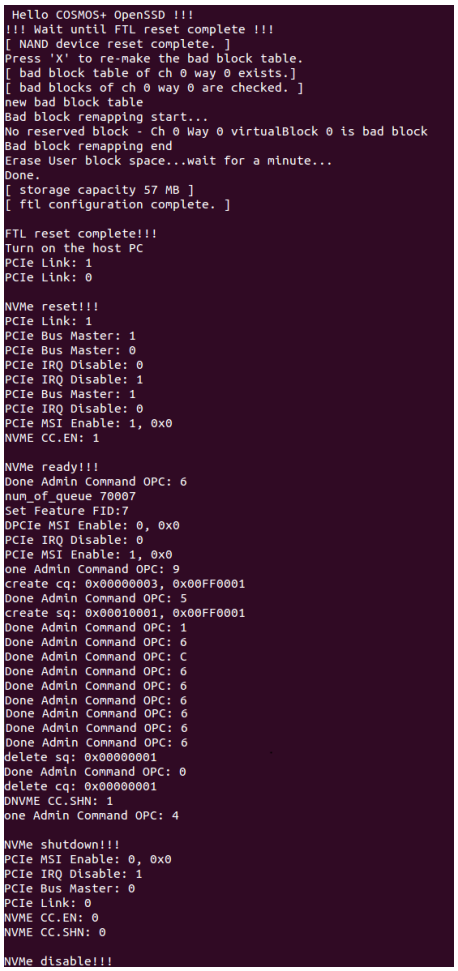


Figure 3.8: Functionality test - Power-up, Partitioning and Reset - Development PC

The cause of this problem has to be found in the project itself: both the processing system and the NVMeHostController IP are running with a heavy memory access loading through the same port of the DDR, that for this reason has become the bottleneck of the project.

On the other hand, no write test are available: if the device is stressed with long or continuous write operations, the DMA of the NVMe Host Controller freezes, resulting in a Timeout Abort Error on the host.

In Figure 3.18 different variables were printed in the terminal to backtrack the cause of the error: in particular, head.autoDmaRx is the hardware counter of the completed DMA request, while tail.autoDmaRx is the software counter of the submitted DMA request: when the two counters coincide, the DMA operation is completed. It is possible to see that the DMA is stuck at head.autoDmaRx = 0xAA, although other 2 requests are present in the queue, being tail.autoDmaRx = 0xAC.

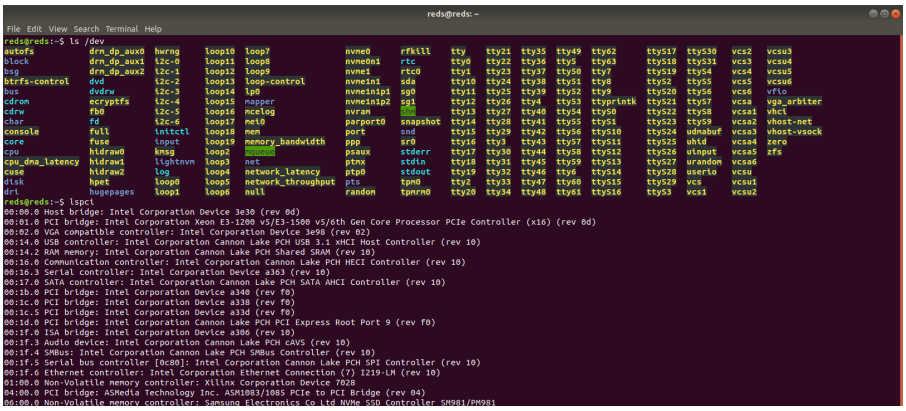


Figure 3.9: Functionality test - Power-up - Host PC

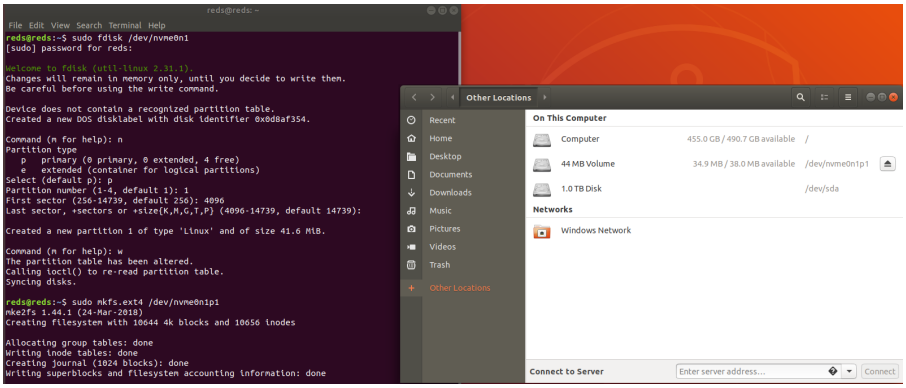


Figure 3.10: Functionality test - Formatting operation - Host PC

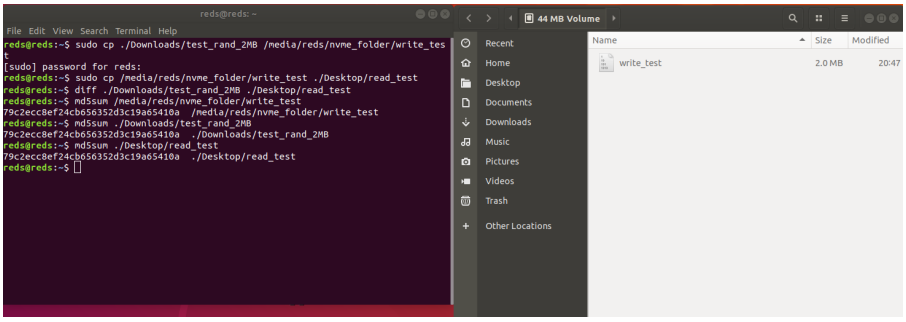


Figure 3.11: Functionality test - Data Correctness - Host PC

As it can be seen in Figures 3.19 and 3.20, the counter X of the submitted request is incremented up to 0x30, however the last increment of head.autoDmaRx is due to the count 0x2e.

Given multiple factors, between which the difficulty to easily reproduce and backtrack the error and the altered timing due to the modifications, a solution to

```
reds@reds:~$ sudo nvme id-ctrl -H /dev/nvme0
NVME Identify Controller:
vid     : 0x1edc
ssvid   : 0x1edc
sn      : SSDD515T
mn      : Cosmos+ OpenSSD
fr      : TYPE0005
rab     : 0
ieee    : 5cd2e4
cmic    : 0
  [2:2] : 0     PCI
  [1:1] : 0     Single Controller
  [0:0] : 0     Single Port

mdts    : 8
cntlid  : 9
ver     : 0
rtd3r   : 0
rtd3e   : 0
oaes    : 0
  [8:8] : 0     Namespace Attribute Changed Event Not Supported

ctratt  : 0
  [0:0] : 0     128-bit Host Identifier Not Supported

oacs    : 0
  [8:8] : 0     Doorbell Buffer Config Not Supported
  [7:7] : 0     Virtualization Management Not Supported
  [6:6] : 0     NVMe-MI Send and Receive Not Supported
  [5:5] : 0     Directives Not Supported
  [4:4] : 0     Device Self-test Not Supported
  [3:3] : 0     NS Management and Attachment Not Supported
  [2:2] : 0     FW Commit and Download Not Supported
  [1:1] : 0     Format NVM Not Supported
  [0:0] : 0     Security Send and Receive Not Supported

acl     : 3
aerl    : 3
frmw    : 0x3
  [4:4] : 0     Firmware Activate Without Reset Not Supported
  [3:1] : 0x1   Number of Firmware Slots
  [0:0] : 0x1   Firmware Slot 1 Read-Only

lpa     : 0
  [2:2] : 0     Extended data for Get Log Page Not Supported
  [1:1] : 0     Command Effects Log Page Not Supported
  [0:0] : 0     SMART/Health Log Page per NS Not Supported

elpe    : 8
npss    : 0
avscc   : 0
  [0:0] : 0     Admin Vendor Specific Commands uses Vendor Specific Format
```

Figure 3.12: Functionality test - NVMe Identify Command - Host PC

```
reds@reds:~$ sudo nvme get-ns-id /dev/nvme0n1
nvme0n1: namespace-id:14740
reds@reds:~$ sudo nvme id-ns /dev/nvme0n1
NVME Identify Namespace 14740:
nsze    : 0x3994
ncap    : 0x3994
nuse    : 0x3994
nsfeat  : 0
nlbaf   : 0
flbas   : 0
mc      : 0
dpc     : 0
dps     : 0
nmic    : 0
rescap  : 0
fpi     : 0
nawun   : 0
nawupf  : 0
nacwu   : 0
nabsn   : 0
nabo    : 0
nabspf  : 0
noiob   : 0
nvmcap  : 0
nguid   : 00000000000000000000000000000000
eui64   : 0000000000000000
lbaf  0 : ms:0   lbads:12 rp:0x2 (in use)
```

Figure 3.13: Functionality test - NVMe Namespace List - Host PC

```
                                                        reds@reds: ~
File  Edit  View  Search  Terminal  Help
reds@reds:~$ sudo dd if=./Downloads/test_rand_8MB of=/dev/nvme0n1 bs=4096
[sudo] password for reds:
2000+0 records in
2000+0 records out
8192000 bytes (8.2 MB, 7.8 MiB) copied, 0.0530143 s, 155 MB/s
reds@reds:~$ sudo dd if=/dev/nvme0n1 of=./Desktop/read_test bs=4096 count=1500
1500+0 records in
1500+0 records out
6144000 bytes (6.1 MB, 5.9 MiB) copied, 0.0644825 s, 95.3 MB/s
```

Figure 3.14: Performance test - dd function, write and read - Base Version

the problem was not found.

3.1.5 Optimized Firmware Version

At first, the cause for the poor performance has been attributed to the scheduling of the NAND requests, having taken for granted that the <MemCpy> function was extremely fast. Therefore an optimized version of the firmware was developed in which the converted slice requests are directly executed, not creating any NAND request.

Even if working correctly, performance were just slightly better than the ones

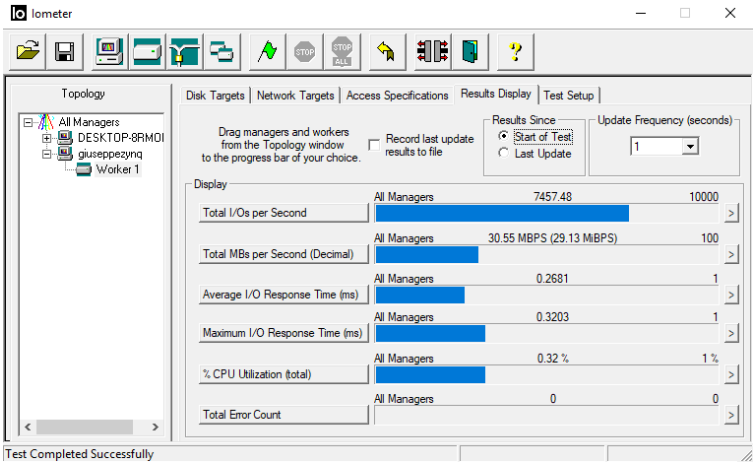


Figure 3.15: Performance test - Iometer Sequential Read transfer size = 4k, block size = 4k - Base Version

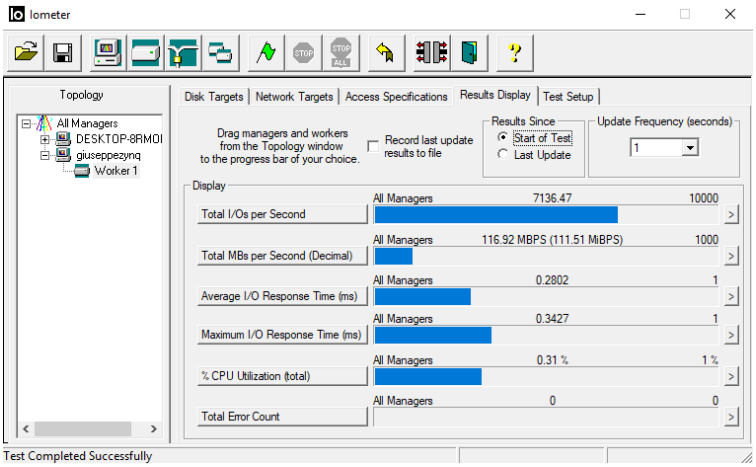


Figure 3.16: Performance test - Iometer Sequential Read transfer size = 16k, block size = 4k - Base Version

of the original firmware, as shown in Figures 3.21,3.22 and 3.23: this helped to discover the real problem, but has lead to no significant improvements. For this reason and to modify the original code the least possible to preserve stability, this version was discarded and no further optimization was carried out.

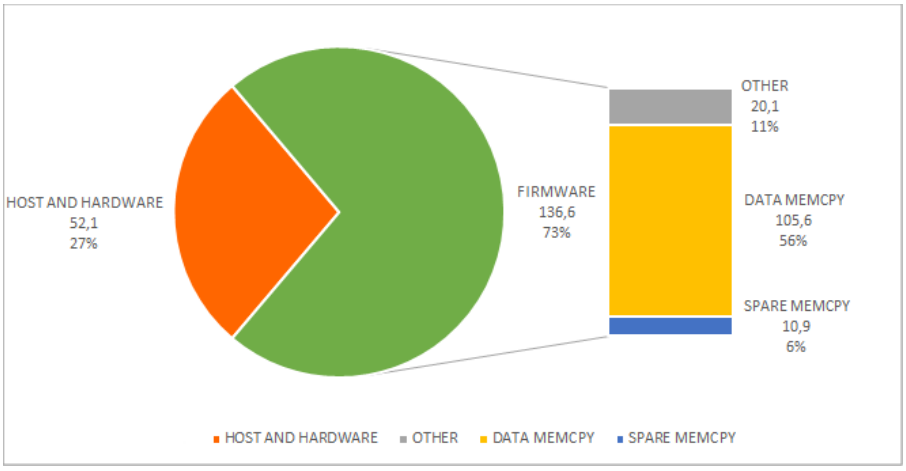


Figure 3.17: Performance test - Read Latency - Base Version

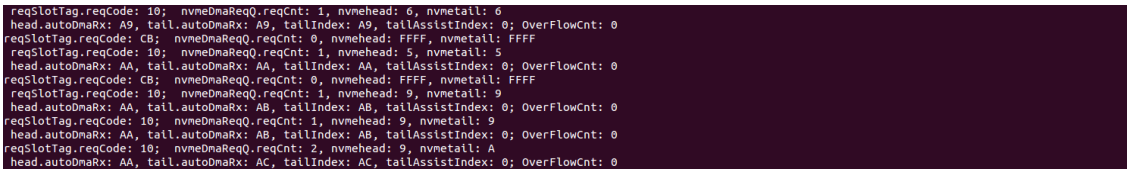


Figure 3.18: Terminal Error - head.autoDmaRx stuck

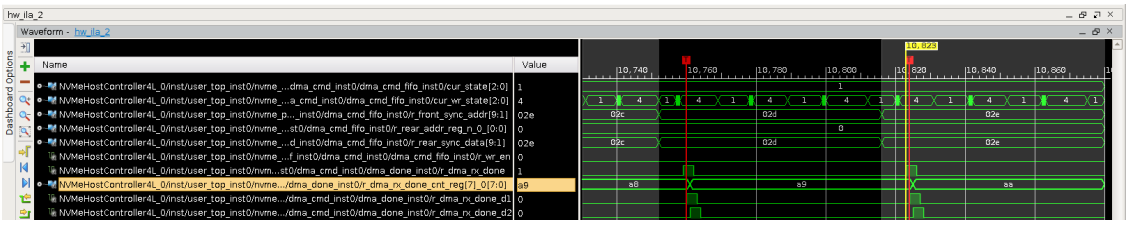


Figure 3.19: Waveform Error - head.autoDmaRx stuck

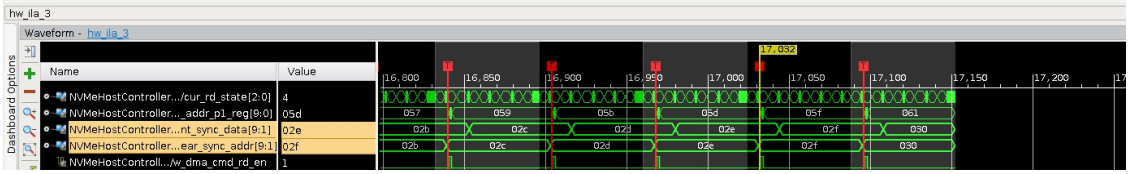


Figure 3.20: Waveform Error - Submitted DMA Request

Figure 3.21: Performance test - dd function, write and read - Optimized Version

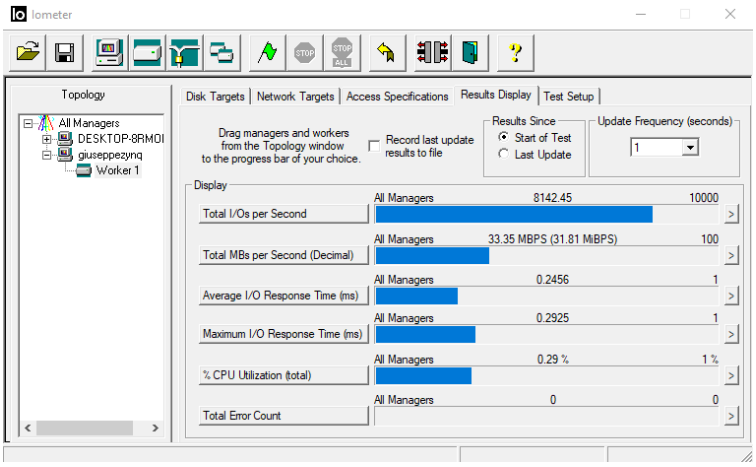


Figure 3.22: Performance test - Iometer Sequential Read transfer size = 4k, block size = 4k - Optimized Version

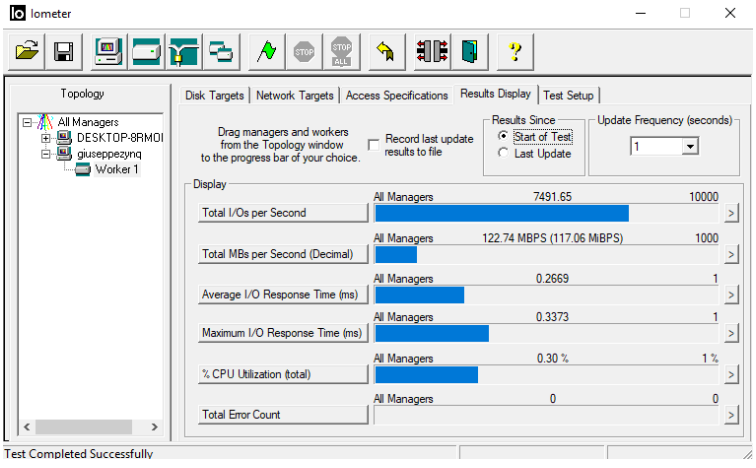


Figure 3.23: Performance test - Iometer Sequential Read transfer size = 16k, block size = 4k - Optimized Version

3.2 Computational Storage Development

The objective is to develop an hardware accelerator consisting of a wrapper that can host different types of acceleration core. However, before starting with the development of the hardware accelerator, it is necessary to modify the project with the addition of the AXI DMA IP, in order to match the general architecture of a FPGA-based Computational Storage Drive 3.24.

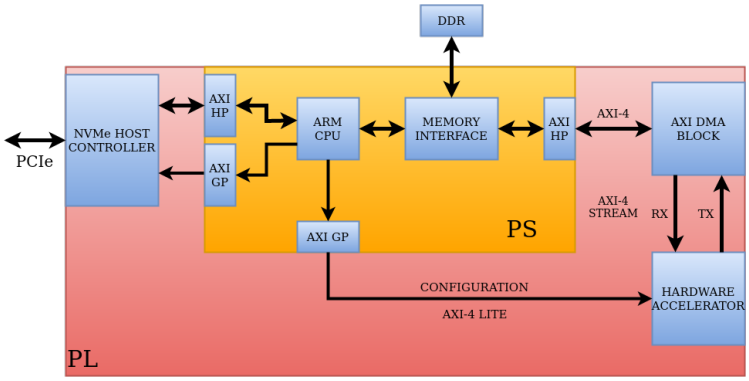


Figure 3.24: General Architecture of a FPGA-based Computational Storage

3.2.1 32-bit AXI DMA

At first, the AXI DMA has been used as a replacement for the MemCpy function, being simply closed on a loop-back. The only firmware modifications concern the configuration of the DMA and, as mentioned above, the substitution of the MemCpy with a DMA transfer.

Some preliminary performance tests are carried out to compare this version to the one with <MemCpy> function.

In Figure 3.26 the <dd> test has been performed: however, due to the small amount of data transferred, the results are not accurate. Instead, as it can be seen in Figures 3.27 to 3.29, despite an increment in the latency, there is a slight increase in both IOPS and bandwidth due to the introduction of the DMA, that reduces the workload of the processing system and favors the increase of throughput.

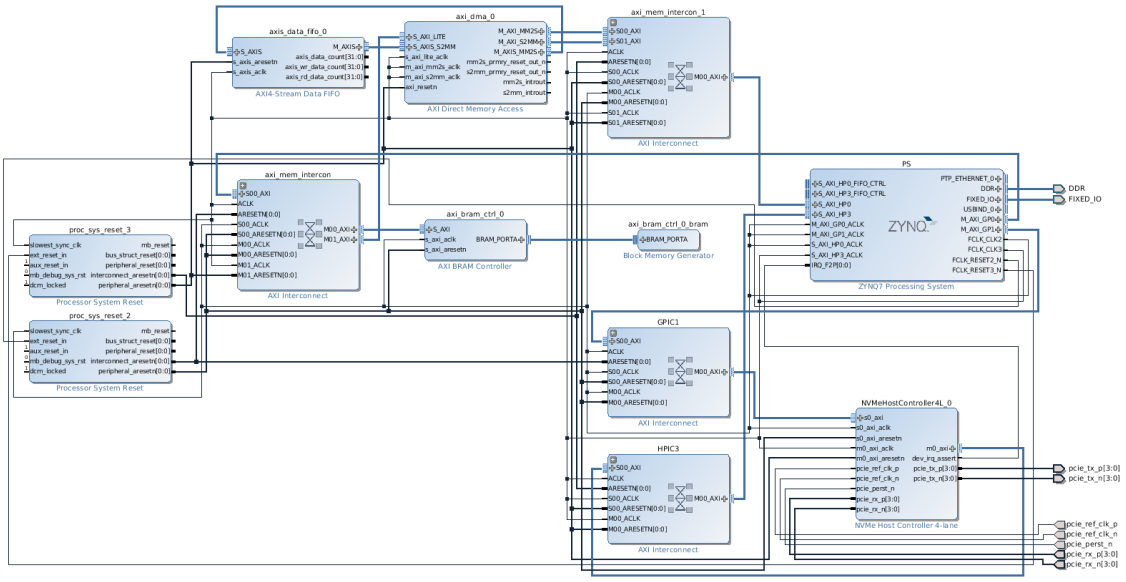


Figure 3.25: 32bit AXI DMA Adaptation of the Cosmos+ OpenSSD project System Design

Figure 3.26: Performance test - dd function, write and read - 32bit AXI DMA Version

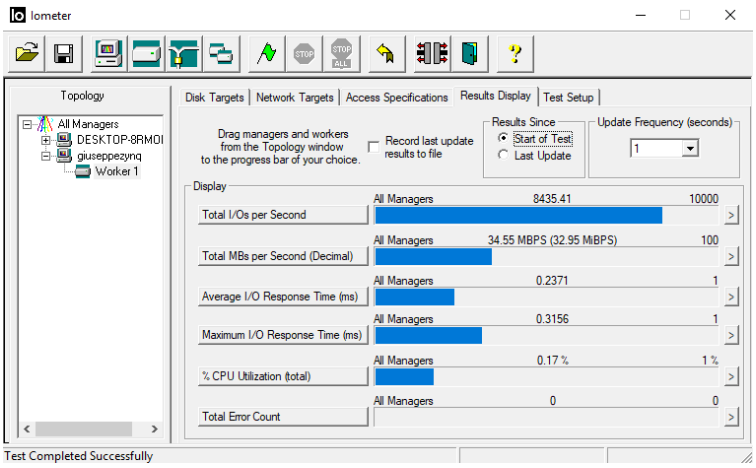


Figure 3.27: Performance test - Iometer Sequential Read transfer size = 4k, block size = 4k - 32bit AXI DMA Adaptation

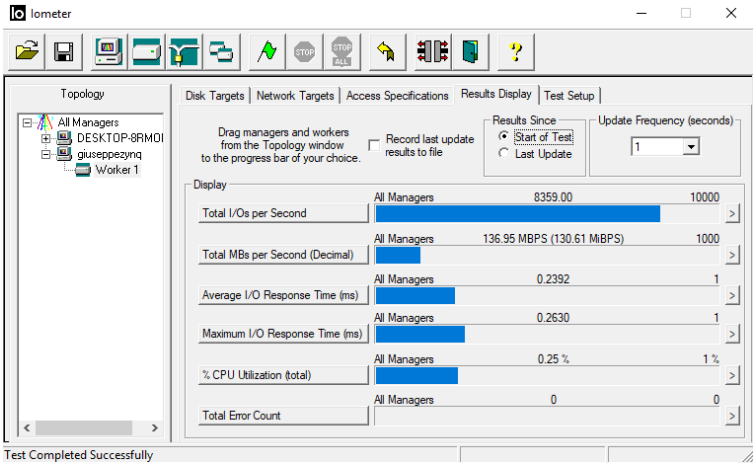


Figure 3.28: Performance test - Iometer Sequential Read transfer size = 16k, block size = 4k - 32bit AXI DMA Adaptation

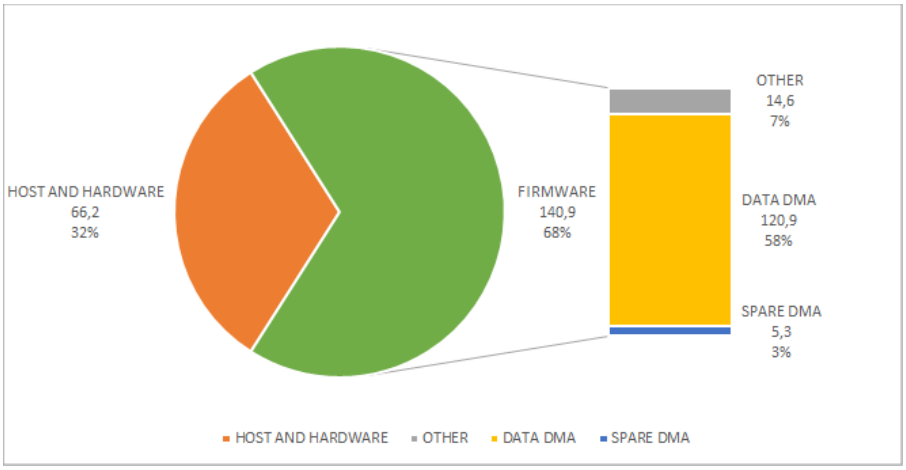


Figure 3.29: Performance test - Read Latency - 32bit AXI DMA Adaptation

3.2.2 First Prototype

The first prototype of hardware accelerator is a simple block which adds the value of a parameter to the data that need to be processed. Its primary goal was to provide a better understanding of the AXI-4 lite and stream protocols and the NVMe Admin Command.

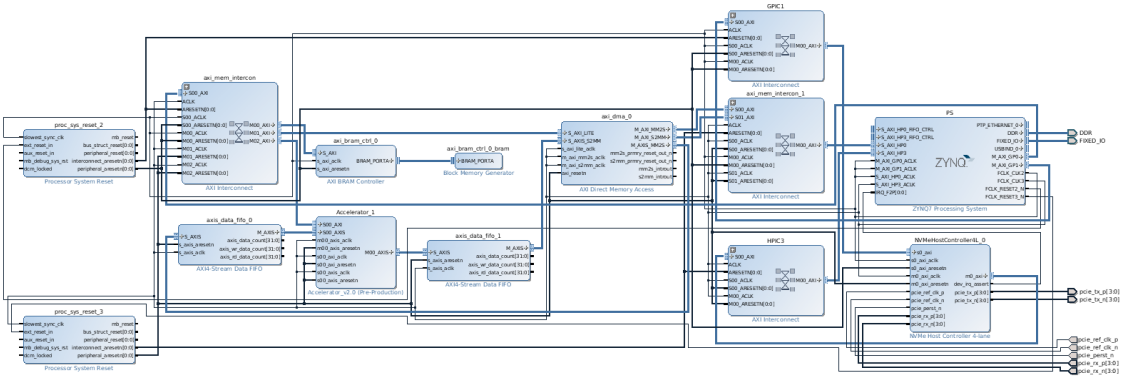


Figure 3.30: System Design of the first version of the Computational Storage based on the Cosmos+ OpenSSD project

As shown in Figure 3.30, the accelerator is enclosed by two Data FIFO, in order to decouple the DMA transfer from the computation.

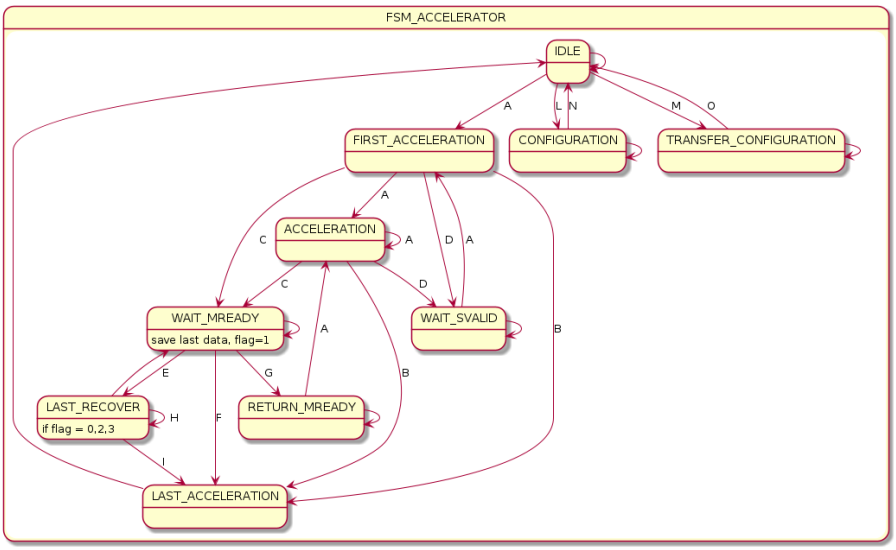
The designed Finite State Machine (FSM) is shown in Figure 3.31.

The Configuration an Transfer_Configuration state handle respectively the modification and the reading of the configuration and parameter registers through the Set Parameters NVMe Admin Commands. The feature identifiers of the commands, a field that indicates for what feature the attribute are being specified for, are chosen between the group of the vendor specific ones[32], being the registers custom.

On the acceleration branch, the different states are used to take care of all the combination of the AXI-4 stream protocol signals and to recover any lost input data in the occurrence of particular conditions (e.g. output not ready on the last element).

The Accelerator has two configurations: the first is a simple pass-through, necessary for the power-on procedure, while the second is the acceleration configuration that, as previously said, is a simple addition of a parameter to the data. It is possible to change the values of the configuration and parameter registers through the NVMe Set Parameter Command, as shown in Figure 3.32.

The test in Figures 3.33 to 3.36 shows a degradation in performance for this solution due to the insertion of FIFOs and Accelerator block.



Letter	Signals
A	m_axis_tready && s_axis_tvalid
B	m_axis_tready && !s_axis_tvalid && flag_tlast
C	!m_axis_tready
D	!s_axis_tvalid && !flag_tlast
E	m_axis_tready && ((s_axis_tlast && s_axis_tvalid)
F	m_axis_tready && flag_tlast
G	m_axis_tready
H	m_axis_tready && flag
I	m_axis_tready && !flag
L	s_axi_awvalid && s_axi_wvalid
M	s_axi_arvalid
N	s_axi_bready
O	s_axi_rready

Figure 3.31: FSM of the first version of the Hardware Accelerator

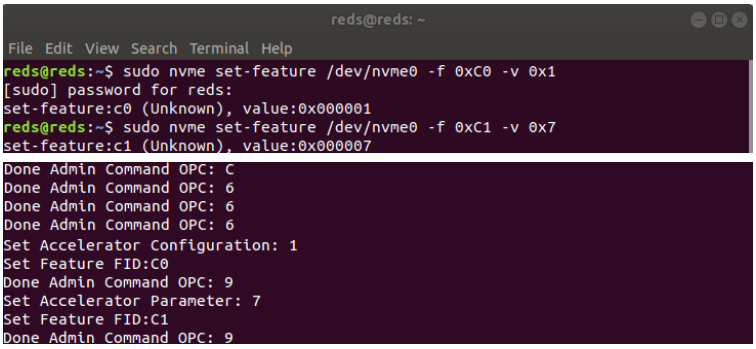


Figure 3.32: NVMe Set Parameter Admin Command - Addition of 0x7 to a 32-bit sequence - Host and Developer PCs

3.2.3 128-bit AXI DMA

As it will be treated in the next section, the chosen acceleration core works on 128bit data packet: in order not to add complexity to the accelerator and achieve higher performance, it is necessary to change the AXI DMA bus width from 32 to 128 bits.

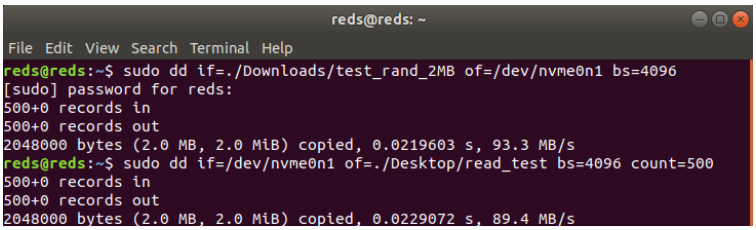


Figure 3.33: Performance test - dd function, write and read - Computational storage first prototype

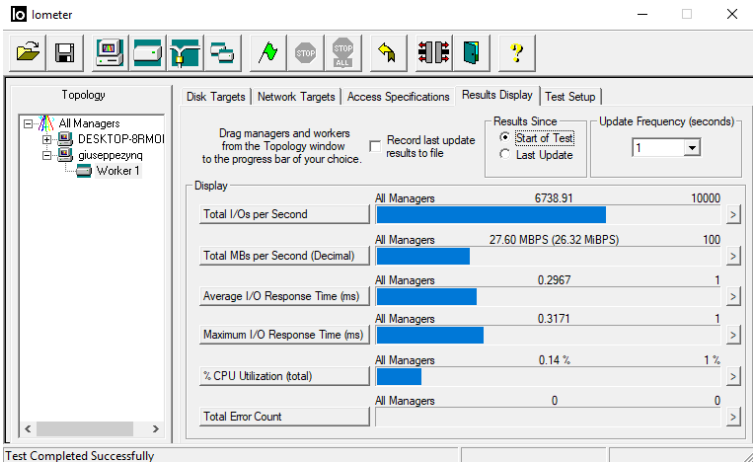


Figure 3.34: Performance test - Iometer Sequential Read transfer size = 4k, block size = 4k - Computational storage first prototype

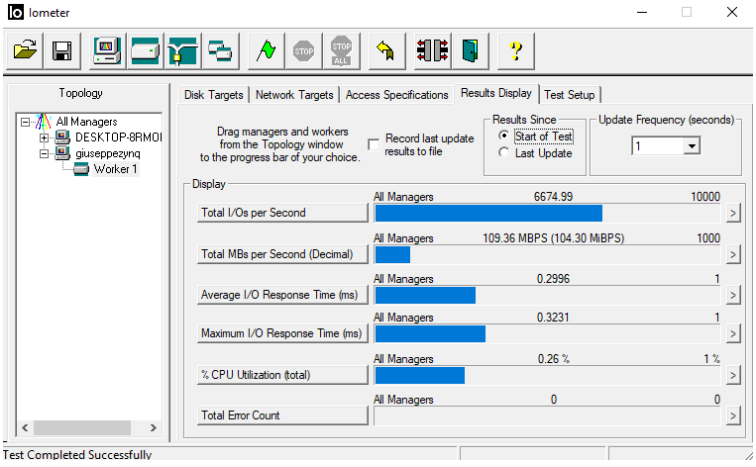


Figure 3.35: Performance test - Iometer Sequential Read transfer size = 16k, block size = 4k - Computational storage first prototype

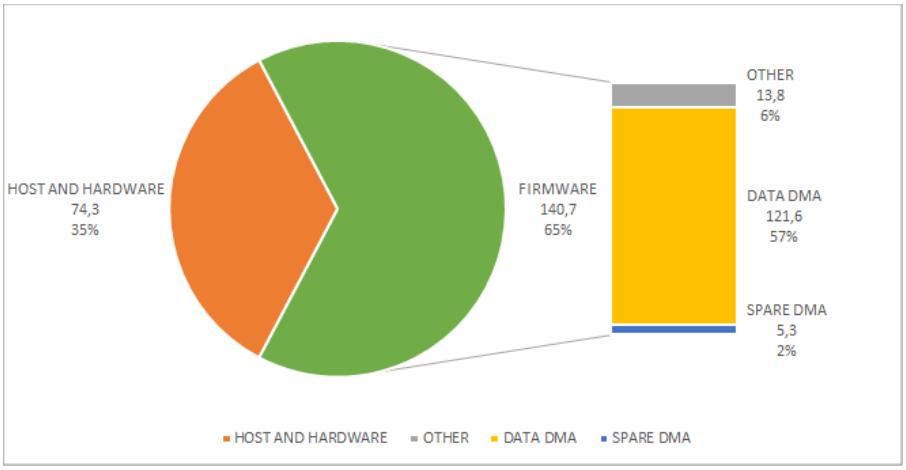


Figure 3.36: Performance test - Read Latency - Computational storage first prototype

Only two firmware modifications has to be carried out in order to perform the 128-bit DMA transfer:

- allocation of a new memory array for the spare data, called <SpaceArray>, to avoid conflicts for consecutive transfers;
- use of an attribute to specify the minimum alignment for the memory arrays, that has to be set to 16 bytes.

As for the previous versions, some preliminary tests are carried out to evaluate the performance of the new configuration.

Figure 3.37: Performance test - dd function, write and read - 128bit AXI DMA Version

The achieved performance, in terms of bandwidth, IOPS and latency, for the 128-bit AXI DMA version are better than those of the 32-bit one thanks to the higher speed of the data transfer from and to the PS. However, as seen in section 3.1.1, the PS can provide only a 64-bit AXI interface, limiting the possible performance increase.

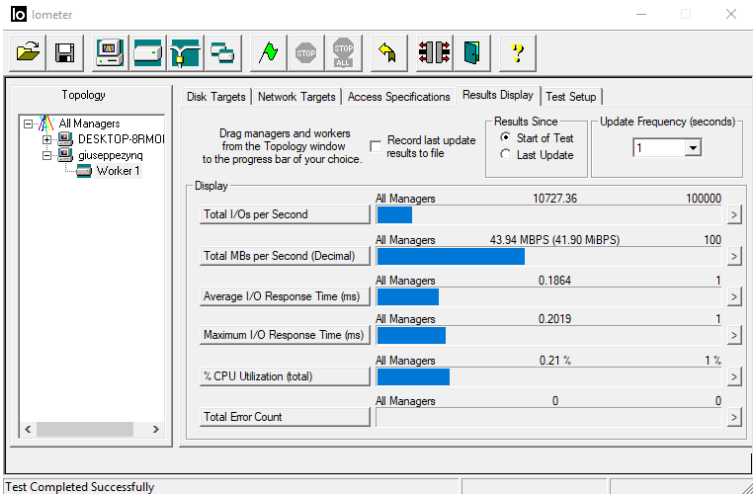


Figure 3.38: Performance test - Iometer Sequential Read transfer size = 4k, block size = 4k - 128bit AXI DMA Adaptation

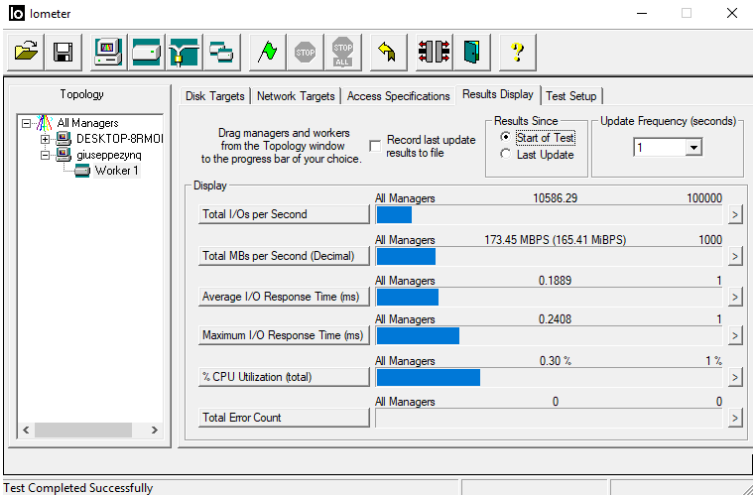


Figure 3.39: Performance test - Iometer Sequential Read transfer size = 16k, block size = 4k - 128bit AXI DMA Adaptation

3.2.4 Second Prototype

The first prototype is not very suitable as general wrapper because the AXI-4 stream protocol makes the management of cores with different timing difficult. Therefore, the previous external FIFOs are then included in the hardware of this second prototype, in order to separate the input and the output AXI-4 stream signals, as shown in Figure 3.41.

In this second prototype, there are two FSMs, as shown in Figure 3.42: the modification and the reading of the registers through NVMe Admin Commands is

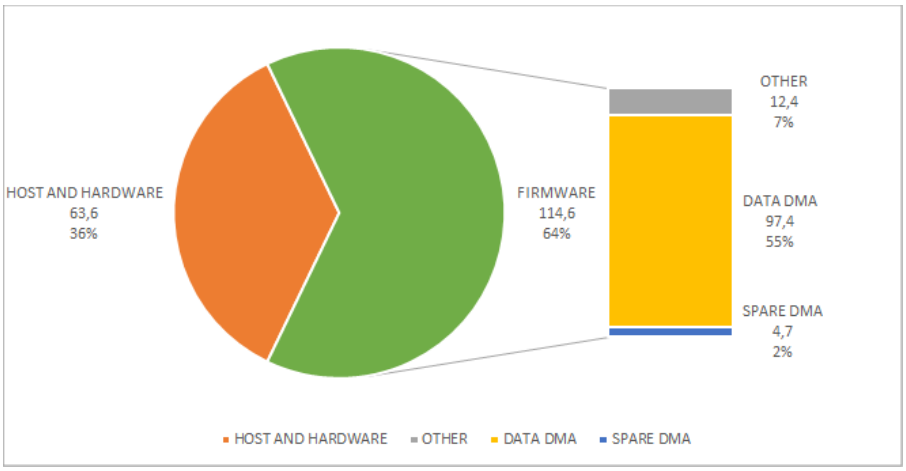


Figure 3.40: Performance test - Read Latency - 128bit AXI DMA Adaptation

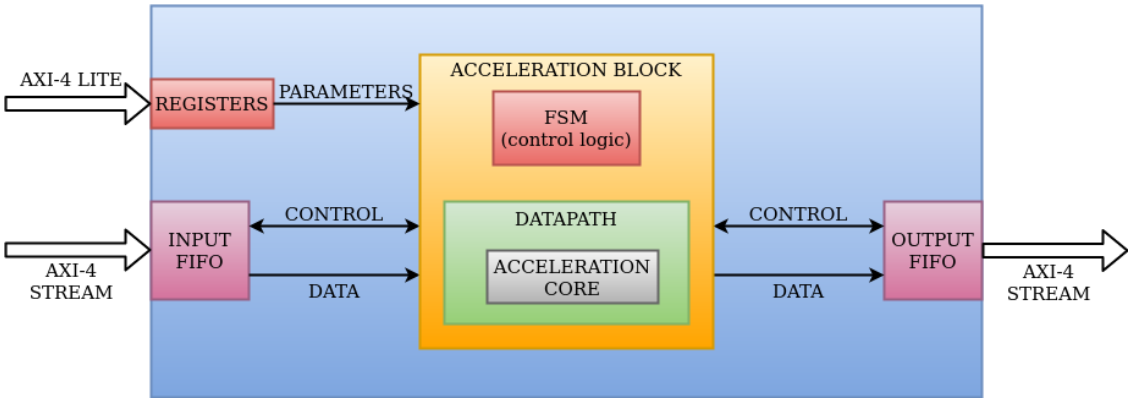


Figure 3.41: General diagram of the Hardware Accelerator

now divided from the acceleration branch.

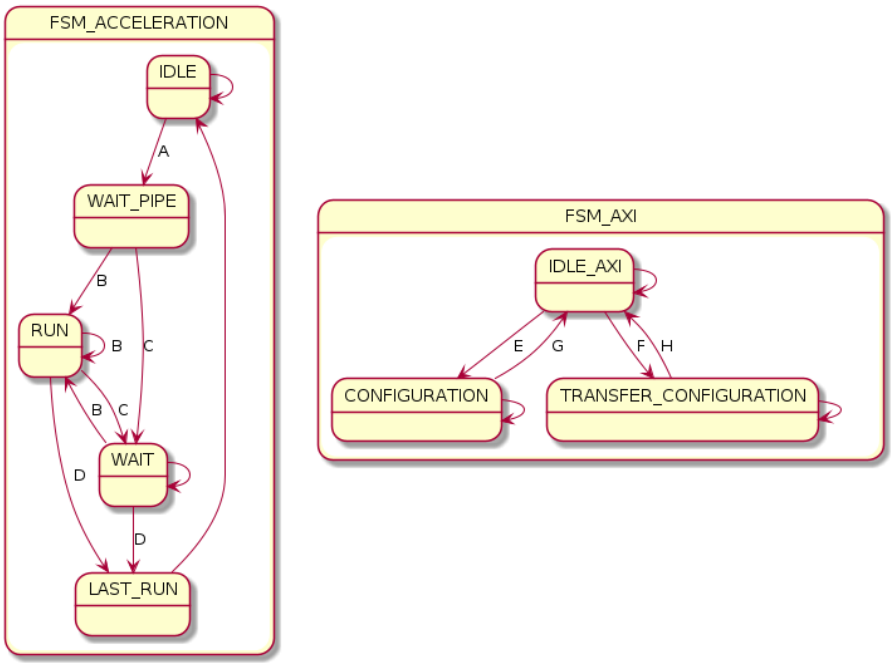
The acceleration branch is simpler than in the first prototipe, due to the acceleration process being indipendent from the AXI-4 stream protocol. The WAIT_PIPE state is used to wait for the core to be ready, in the case its output is not immediately available.

AES Core and CTR Configuration

In order to verify the correct functioning of the block, it was necessary to choose the acceleration function and, consequently, the core to be inserted.

The sought acceleration function has to be:

- widespread and standard;
- format-preserving: for construction limits (DMA transfer) the output data



Letter	Signals
A	!empty & !full
B	valid & !empty & !full
C	valid & !empty & !full
D	valid & !empty & !full & last
E	s_axi_awvalid && s_axi_wvalid
F	s_axi_arvalid
G	s_axi_bready
H	s_axi_rready

Figure 3.42: FSMs of the second version of the Hardware Accelerator

format has to be the same of the input one;

- used with a data stream: to achieve high performance.

Therefore, the chosen core is an encryption one based on the Advanced Encryption Standard (AES)[33], called "tiny_aes". The AES encryption algorithm, that is a specific implementation of the Rijndael one, has been adopted and standardized by the National Institute of Standards and Technology (NIST) and the US FIPS PUB in 2001 and is accepted all over the world. The AES is a symmetric encryption algorithm, using a single key for both encryption and decryption. It is a format-preserving encryption algorithm that works on a single fixed-length data block of 128-bit. On the other hand, the key size can be 128,192 or 256 bit.

The encryption process consists of different rounds of transformations, which number varies depending on the length of the key.

Five modes of operation were defined[34]:

- Electronic Code Book (ECB) mode: the simplest mode and, for this reason,

generally not recommended. All the input data blocks, called plain text, are encrypted with same key, allowing parallel encryption but resulting in low level of security;

- Cipher Block Chaining (CBC) mode: the input block is xored with an initialization vector (IV) and then encrypted. The resulting cipher text is used as IV for the next block;
- Cipher FeedBack (CFB) mode: the key and the IV are the input of the encryption block, which result is xored with the plain text. As in the CBC, the resulting cipher text is used as IV for the next block;
- Output FeedBack (OFB) mode: as in the CFB, the key and the IV are the input of the encryption block, which result is xored with the plain text to give the cipher text. However the IV for the next block is given by the result of the encryption block;
- Counter (CTR) mode: in this mode the IV is a counter. As in the CFB, the key and the IV are the input of the encryption block, which result is xored with the plain text to give the cipher text. However the IV for the next block is given by the incremented counter.

Apart from the ECB, each mode has advantages and disadvantages: the choice on which to use depends on the application. Given the requirements for the accelerator, the best choice is the CTR mode: in fact in this mode the encryption and decryption can be prepared in advance and the parallel computing is supported, providing in this way the possibility to achieve high speed while ensuring security. Then the hardware accelerator is modified according to the scheme shown in Figure 3.43, in which each block is equivalent of a pipeline stage.

It is fundamental that a data block uses the same value of the counter in both encryption and decryption. Otherwise, the operation is not completed correctly.

As described in the project specification[33], there are three cores available, one for each key size: due to the limit in FPGA resources, only the AES-128 and AES-256 are included in the hardware accelerator.

Functionality Tests

To verify the correct behavior of the developed accelerator, the NIST provides test vectors[34] for both encryption and decryption operations.

By construction of the cores, input and output of the core have to be reversed in order to be processed. This occurs for the opposite orientation of the software and hardware vectors.

Accounting NIST test vectors to be in a reversed state, only the output of the core is inverted: the key has to be inserted as it is. As well, to verify that results match, both plain-text and cipher-text have to be reversed.

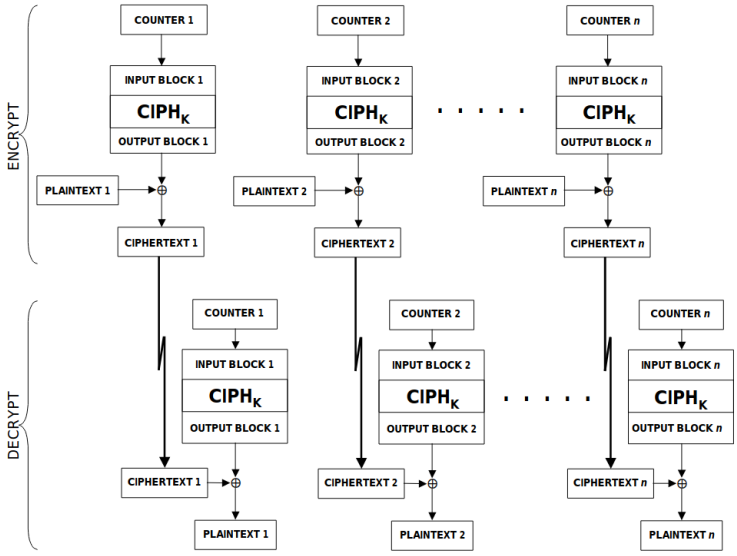


Figure 3.43: CTR Mode - NIST, Recommendation for Block Cipher Modes of Operation: Methods and Techniquesm

The test are performed in three different ways:

- Module verilog test-bench;
- Module execution;
- General execution.

As shown in Figures the first vectors of the test-bench waveform are, respectively, plain-text blocks, initial counter and key from NIST test vectors.

The plain-text is reversed in the <data_in_pipe> vector to ensure the correct behavior: the match of results can be verified by the vector <reverse_data>, inversion of the accelerator output.

For the module execution test, the comparison between the obtained cipher-text and NIST test vectors is directly executed by the software. In this type of test the configuration still takes place with direct writing of configuration and parameter registers.

Figures 3.48 and 3.51 shows the messages printed by the software, as a consequence of the correct completion of the operations, for both configurations.

In Figures 3.49, 3.50, 3.52 and 3.53 instead, the complete execution waveforms for the parameters configuration and the encryption phase only are shown.

As mentioned previously, it is possible to notice the difference between the time passed in the state WAIT_PIPE (code 2) in the two encryption waveforms: the process takes more rounds of transformations to complete the operations in the case of the AES-256 core rather than the AES-128 one.

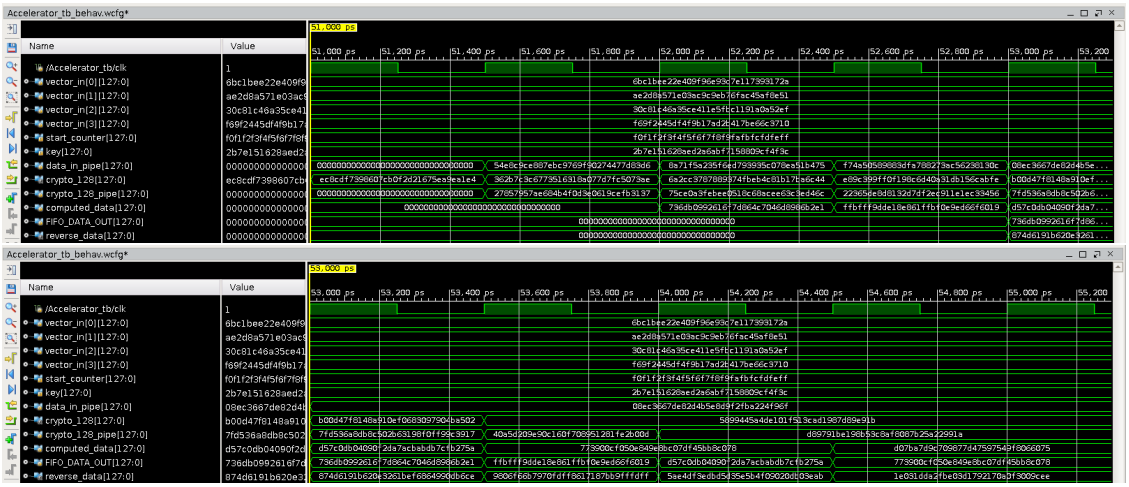


Figure 3.44: Verilog Test-bench - AES-128 Encryption

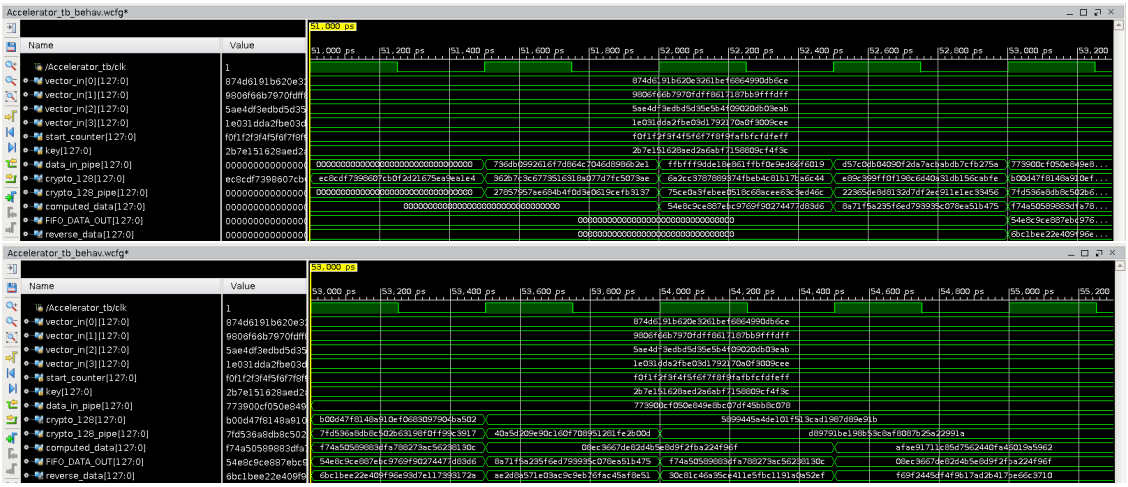


Figure 3.45: Verilog Test-bench - AES-128 Decryption

Before performing the general test, is necessary to define in the firmware the Feature Identifiers of the Set Feature command. Their assignation to the configuration and parameters registers is shown in Table 3.1.

Finally, it is possible to perform the general test. Figures 3.54 and 3.55 show, from both host and developer PC sides, the configuration phase and write operation to send the test vectors: <test_128.bin> is a binary file in which the NIST vectors have been reversed. The printed output, which are the hexadecimal values of the byte saved in the MemSpace array, are equivalent to test vectors reversed.

The write operations are always referred to a page (16kB): therefore, as it can be seen in Figure 3.31, the amount of computed data is much larger than the 64 bytes of binary file.

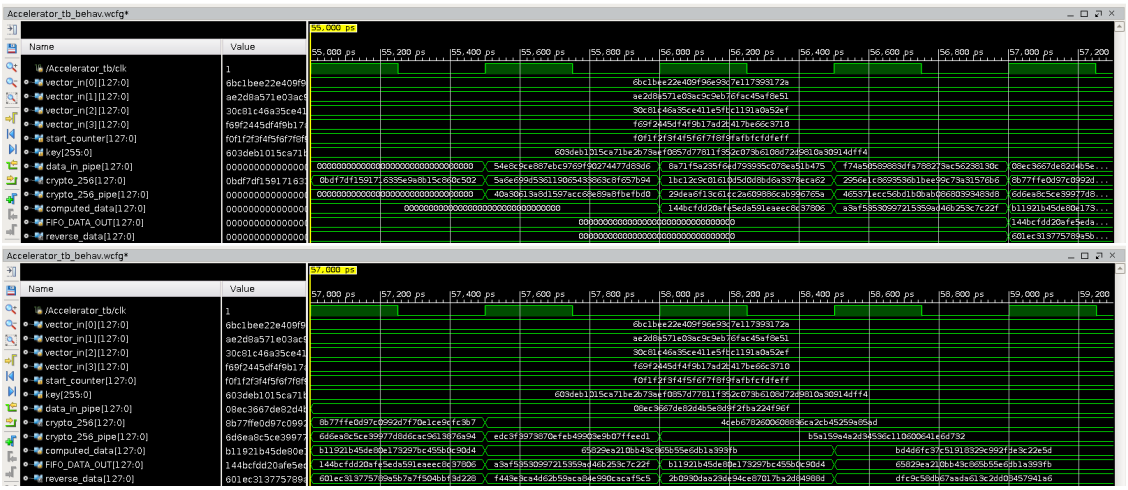


Figure 3.46: Verilog Test-bench - AES-256 Encryption

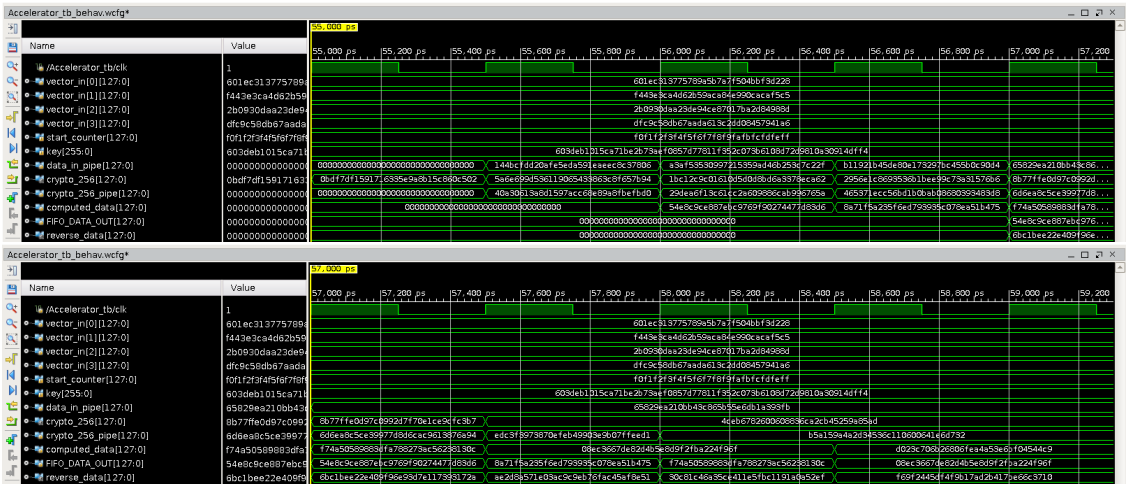


Figure 3.47: Verilog Test-bench - AES-256 Decryption

```
Terminal ready
◆[!] MMU has been enabled.

 Hello COSMOS+ OpenSSD !!!
Configuration - Press 'X' to start
Configuration: 02; Key 128bit = 2B7E1516-28AED2A6-ABF71588-9CF4F3C; IV: F0F1F2F3-F4F5F6F7-F8F9FAFB-FCFDFEFF
Encryption-Decryption - Press 'X' to start
transfer Dev-DMA successful
transfer DMA-Dev successful
Encryption completed successfully
transfer Dev-DMA successful
transfer DMA-Dev successful
Decryption completed successfully
done
```

Figure 3.48: Module execution - AES-128 Terminal

Same operations are performed for the AES-256 core in Figures 3.57 to 3.59.

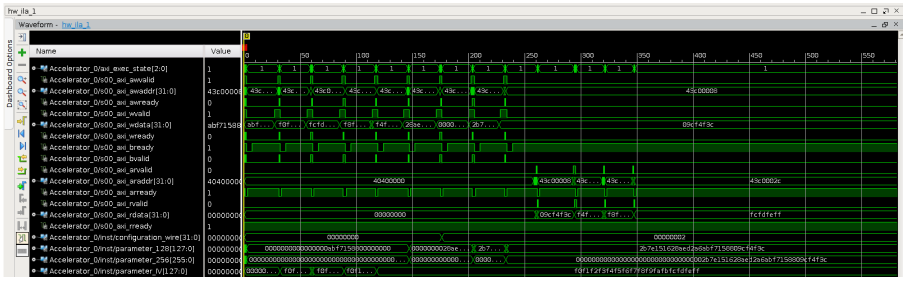


Figure 3.49: Module execution - AES-128 Configuration

Figure 3.50: Module execution - AES-128 Encryption

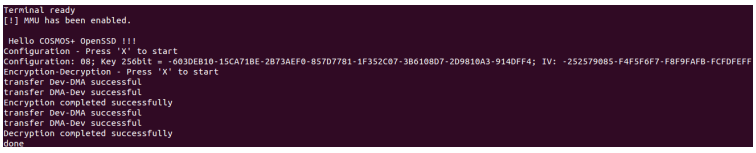


Figure 3.51: Module execution - AES-256 Terminal

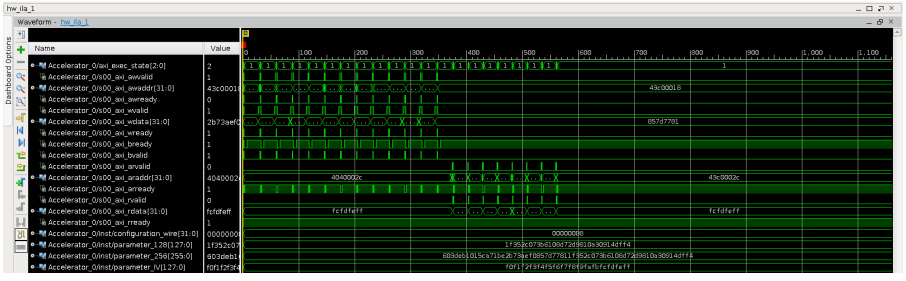


Figure 3.52: Module execution - AES-256 Configuration

Figure 3.53: Module execution - AES256 Encryption

Feature Identifier	Register Name	Notes
0xC0	Configuration Register	0 = Pass through; 1 = Adder; 2 = AES-128; 8 = AES-256
0xC1	Parameter 1 Register	LSW of the key vector
0xC2	Parameter 2 Register	
0xC3	Parameter 3 Register	
0xC4	Parameter 4 Register	MSW of the 128-bit key vector
0xC5	Parameter 5 Register	
0xC6	Parameter 6 Register	
0xC7	Parameter 7 Register	
0xC8	Parameter 8 Register	MSW of the 256-bit key vector
0xCA	Parameter A Register	LSW of the IV vector
0xCB	Parameter B Register	
0xCC	Parameter C Register	
0xCD	Parameter D Register	MSW of the IV vector

Table 3.1: Set Parameter - Feature Identifiers Assignation

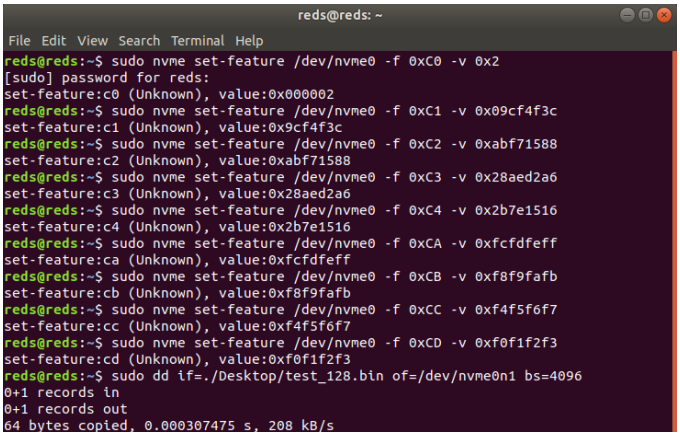


Figure 3.54: General Execution - AES-128 Host PC Terminal

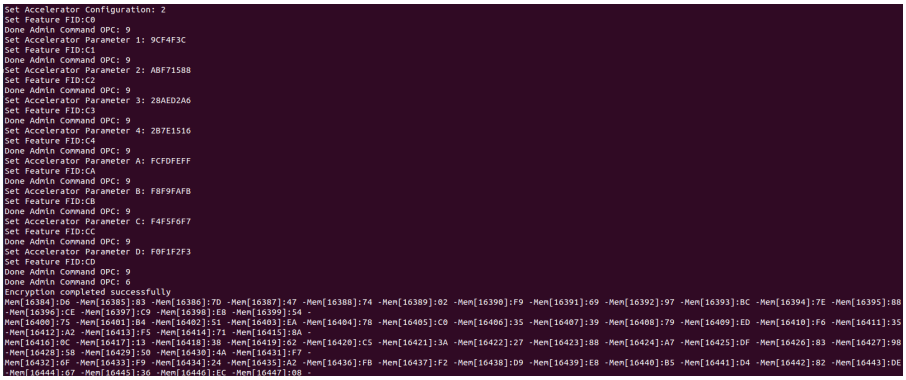


Figure 3.55: General Execution - AES-128 Developer PC Terminal

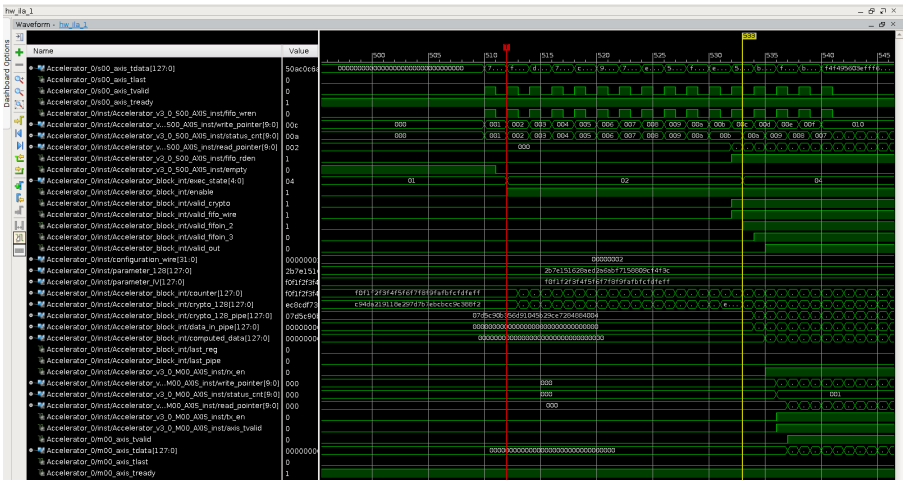


Figure 3.56: General Execution - AES-128 Encryption Waveform

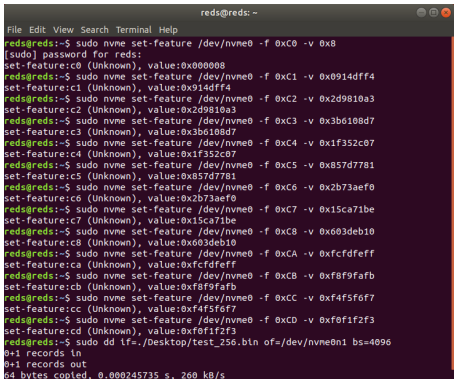


Figure 3.57: General Execution - AES-256 Host PC Terminal

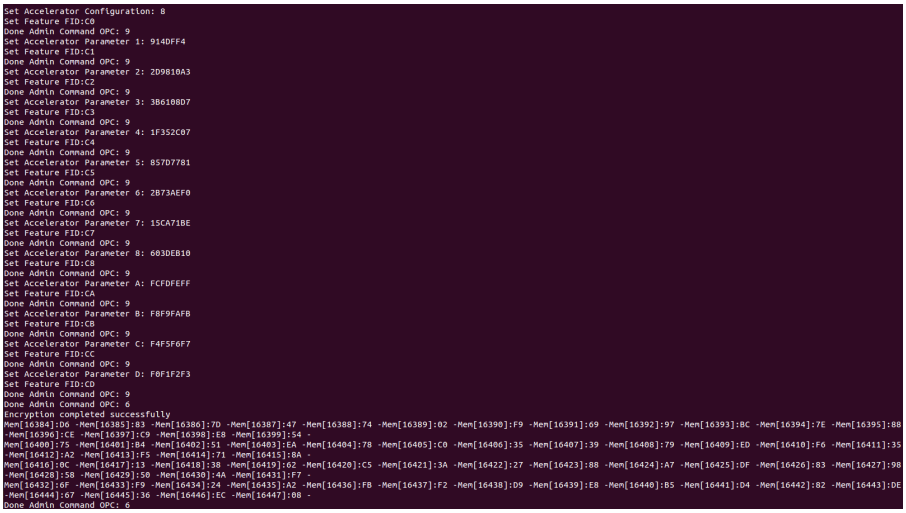


Figure 3.58: General Execution - AES-256 Developer PC Terminal

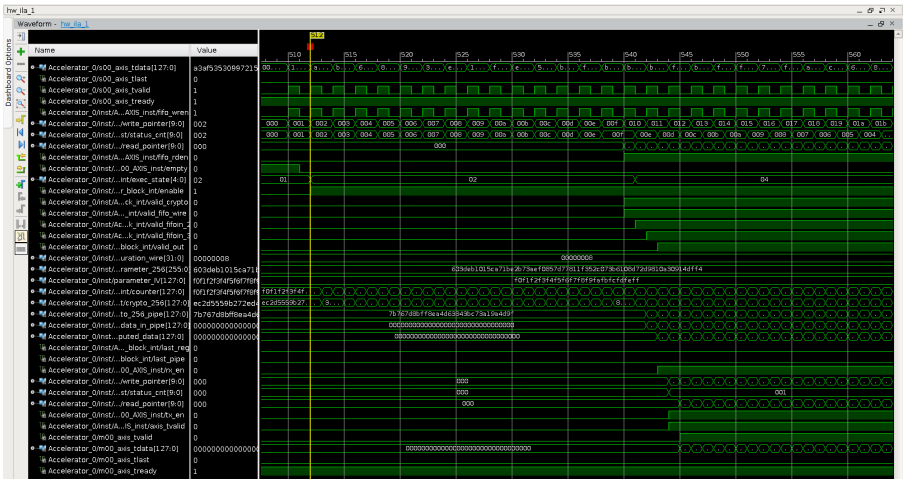


Figure 3.59: General Execution - AES-256 Encryption Waveform

Performance Tests

As in the previous cases, performance tests are carried out to evaluate IOPS, bandwidth and latency for the final computational storage configured to perform the cryptography.

As it can be seen in Figures 3.60 to 3.63, there is almost no degradation in performance after the insertion of the hardware accelerator.

Figure 3.60: Performance test - dd function, write and read - 128bit AXI DMA Version

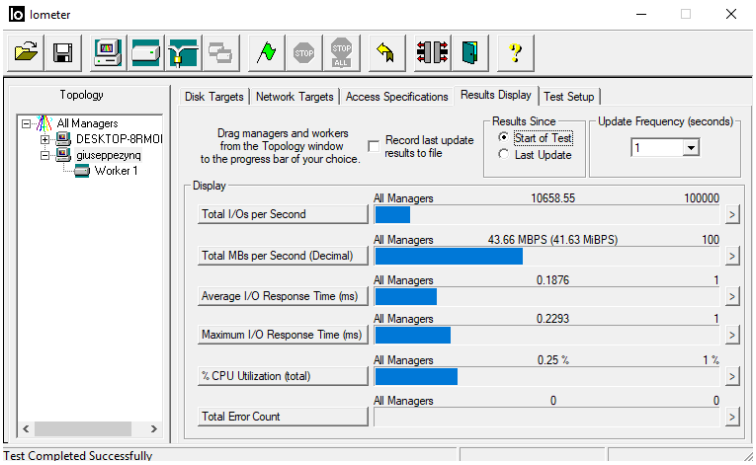


Figure 3.61: Performance test - Iometer Sequential Read transfer size = 4k, block size = 4k - Computational Storage

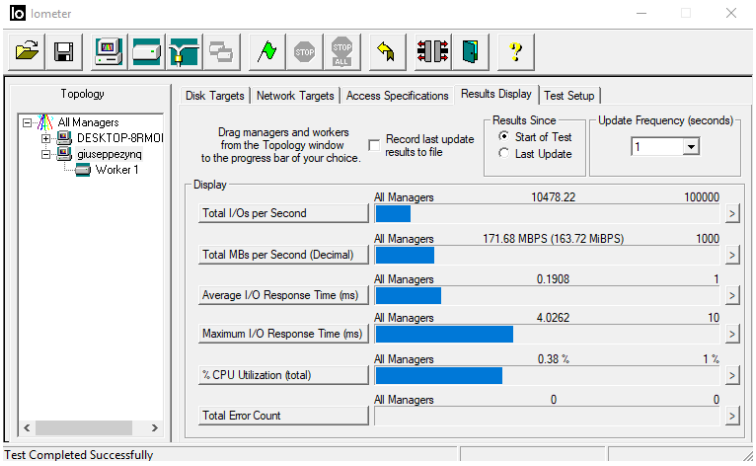


Figure 3.62: Performance test - Iometer Sequential Read transfer size = 16k, block size = 4k - Computational Storage

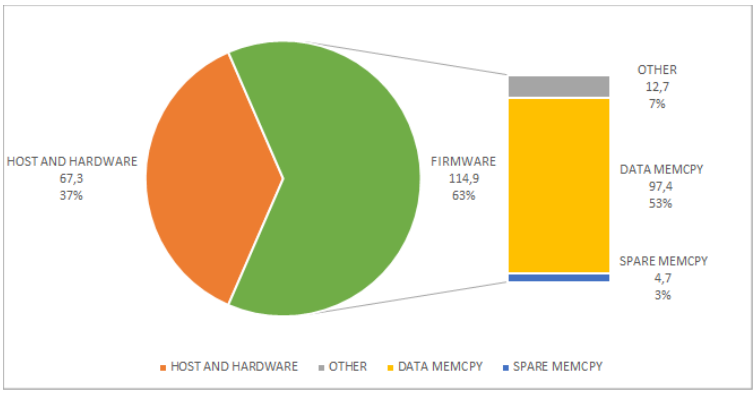


Figure 3.63: Performance test - Read Latency - Computational Storage

Chapter 4

Conclusion

This research aimed to experience and explore capabilities of an NVMe computational storage, as opportunity take data centers to the next level in terms of performance and energy efficiency.

The developed computational storage is easily configurable from the host side, thanks to the NVMe protocol on which is based: no additional software/driver installations is required due to the NVMe driver being open source and supported in all major distributions. This involves the development of an NVMe controller as interface block for the computational storage that, though, complies with specifications common to all the vendors. Moreover, the results provided by the performance test carried out, summarized in Tables tables 4.1 and 4.2, show that the developed hardware accelerator has good performance: in fact, it does not introduce significant losses with respect to the standard version (without the acceleration block).

However, it was not possible to make a detailed analysis of the possible effects which could result from the increase of the operating frequency for both DMA and accelerator blocks due to performance limits. In fact, the obtained performance results are much lower than the expected ones, being heavily affected by the high memory traffic in the DDR: the problem could be solved using a platform with more internal resources or expansion possibilities. Furthermore, it is necessary to find a solution to the problem affecting the write operation, in order to finally obtain a bidirectional device.

Further improvements can be carried out: the original project was not designed to perform this type of operations, therefore a dedicated firmware can enhance the performance, reducing the software latency. Lastly, the computational storage, thanks to the NVMe protocol, could work in a peer-to-peer network, which exploitation enables further reduction of the CPU workload and data movement, resulting in higher efficiency and lower energy consumption.

Version	**Iometer**			
	Transfer size 4kB		Transfer size 16kB	
	IOPS	Bandwidth [*MBPS*]	IOPS	Bandwidth [*MBPS*]
Base	7457	30.55	7136	116.92
32-bit AXI DMA	8435	34.55	8359	136.95
First Prototype	6738	27.60	6674	109.36
128-bit AXI DMA	10727	43.94	10586	173.45
Final Prototype	10658	43.66	10478	171.68

Table 4.1: Iometer Sequential Read Test - block size 4kB

Version	**Latency** [μs]			
	Firmware	Data Transfer	Host & Hardware	Total
Base	136.6	105.6	52.1	188.7
32-bit AXI DMA	140.9	120.9	66.2	207.1
First Prototype	140.7	121.6	74.3	215
128-bit AXI DMA	114.6	97.4	63.6	178
Final Prototype	114.9	97.4	67.3	182.2

Table 4.2: Latency Sequential Read Test - block size 4kB

Bibliography

[1] Roderick Bauer, *The Challenges of Opening a Data Center*, 2018, accessed June 2020, <https://www.backblaze.com/blog/choosing-data-center/>.

[2] Kachris, Christoforos, Falsafi, Babak, Soudris, Dimitrios, Energy-Efficient Servers and Cloud, *Hardware Accelerators in Data Centers*, 2019.

[3] Nicola Jones, *How to stop data centres from gobbling up the world's electricity*, Nature, September 2018 accessed June 2020, <https://www.nature.com/articles/d41586-018-06610-y>.

[4] Wayne M. Adams, *Power consumption in data centers is a global problem*, Data Center Dynamics (DCD), November 2018 accessed June 2020, <https://www.datacenterdynamics.com/en/opinions/power-consumption-data-centers-global-problem/>.

[5] Kachris, Christoforos, Falsafi, Babak, Soudris, Dimitrios, The Era of Accelerators in the Data Centers, *Hardware Accelerators in Data Centers*, 2019.

[6] João M.P. Cardoso, José Gabriel F. Coutinho, Pedro C. Diniz, High-performance embedded computing, *Embedded Computing for High Performance*, 2017.

[7] Mahdi Torabzadehkashi, Siavash Rezaei, Ali HeydariGorji, Hosein Bobarshad, Vladimir Alves, Nader Bagherzadeh, *Computational storage: an efficient and scalable platform for big data and HPC applications*, Journal of Big Data, 2019.

[8] Rino Micheloni, The Need for High Speed Interfaces, *Solid-State-Drives (SSDs) Modeling: Simulation Tools & Strategies fonte*, 2017.

[9] NVM Express, *What is NVMe™?*, accessed June 2020, <https://nvmexpress.org/>.

[10] Greg Schulz, Doug Rollins, *Why NVMe Should Be in Your Data Center*, Micron, accessed June 2020, <https://www.micron.com/solutions/technical-briefs/why-nvme-should-be-in-your-data-center>.

[11] NVM Express, *How does power management work with NVMe technology?*, accessed June 2020, <https://nvmexpress.org/faq-items/how-does-power-management-work-with-nvme-technology/>.

[12] IP-Maker, *NVMe IP for Enterprise SSD*, Design & Reuse, accessed June 2020, <https://www.design-reuse.com/articles/39493/nvme-ip-for-enterprise-ssd.html>.

[13] IP-Maker, *Conquering the challenges of PCIe with NVMe in order to deliver highly competitive Enterprise PCIe SSD*, Design & Reuse, accessed June 2020, <https://www.design-reuse.com/articles/34744/conquering-the-challenges-of-pcie-with-nvme.html>.

[14] Wikipedia, *PCI Express — Wikipedia, The free encyclopedia*, 2020, accessed June 2020, <https://en.wikipedia.org/wiki/PCI_Express>.

[15] Amber Huffman, *NVM Express: Optimized Interface for PCI Express SSDs*, Intel Developer Forum (IDF), 2013.

[16] Mark Carlson, Paul Suhler, *Managing Capacity in NVM Express SSDs*, Samsung Developer Conference (SDC), 2019.

[17] J Metz, *Creating Higher Performance Solid State Storage with Non-Volatile Memory Express*, Data Storage Innovation (DSI) Conference SNIA™, 2015.

[18] Brandon Hoff, *NVMe over Fabrics*, NVM Express, 2017.

[19] Kamal Hyder, Manoj Wadekar, Yaniv Romem, Nishant Lodha, *NVMe-oF™ Enterprise Appliances*, Flash Memory Summit, 2020.

[20] Pure Storage, *PERCHÉ NVMe-oF È IL FUTURO?*, accessed June 2020, <https://www.purestorage.com/it/resources/glossary/nvme-over-fabrics.html>.

[21] Mike Kieran, *When You're Implementing NVMe Over Fabrics, the Fabric Really Matters*, NetApp, March 2019, accessed June 2020, <https://blog.netapp.com/nvme-over-fabric/>.

[22] Franco Benuzzi, *NVMe e NVMe-oF: i nuovi protocolli di accesso ai dischi flash*, lantechlongwave, June 2019, accessed June 2020, <https://www.lantechlongwave.it/archivio/nvme-nvme-of-nuovi-protocolli-accesso-ai-dischi-flash/>.

[23] BittWare, *Traditional vs. Computational Storage*, accessed June 2020, <https://www.bittware.com/fpga/storage/>.

[24] Stephen Bates, *Accelerating RocksDB with Eideticom's NoLoad®*, Samsung Developer Conference (SDC), 2019.

[25] Stephen Bates, Richard Mataya, *Accelerating Applications with NVM Express™ Computational Storage*, NVMe® Annual Members Meeting and Developer Day, 2019.

[26] Scott Shadley, Nick Adams, David Slik, *What happens when Compute meets Storage?*, SNIA Computational Storage, 2019.

[27] Stephen Bates, *How NVM Express and Computational Storage can make your AI Applications Shine!*, Massive Storage Systems and Technology (MSST), 2019.

[28] Sean Gibb, Stephen Bates, *An NVMe-based Offload Engine for Storage Acceleration*, SNIA Storage Developer Conference (SDC), 2017.

[29] Daniel Robinson, *Computational Storage Winds Its Way Towards The Mainstream*, TheNextPlatform, February 2020, accessed June 2020, <https://www.nextplatform.com/2020/02/25/computational-storage-winds-its-way-towards-the-mainstream/>.

[30] Yong Ho Song, *Cosmos+ OpenSSD 2017 Tutorial*, HYU ENC Lab of Hanyang University, 2017.

[31] Xilinx, *ZC706 Evaluation Board for the Zynq-7000 XC7Z045 SoC, User Guide*, Page 46-47, 2019.

[32] NVM Express, *NVM Express™ Base Specification Revision 1.4*, Page 206-207, June 2019.

[33] Homer Hsing, *AES Core Specification*, OpenCores, February 2013, Accessed June 2020, <https://opencores.org/projects/tiny_aes>

[34] Morris Dworkin, *Recommendation for Block Cipher Modes of Operation: Methods and Techniquesm*, NIST Special Publication 800-38A, December 2001.