Faculty of Engineering Sciences

Heidelberg University

Master's Thesis

in Computer Engineering

submitted by

Bc. VLADISLAV VÁLEK

born in Šumperk, Czech Republic

2026

SHARED CPU FPGA FILESYSTEM ACCESS ON FAST NVME DEVICES

This Master's Thesis has been carried out by

Bc. Vladislav Válek

at the

Institute of Computer Engineering (ZITI)

under the supervision of

Prof. Dr. Dirk Koch

ABSTRACT

Abstract in English.

KEYWORDS

PCIe, PCI Express, DMA, Direct Memory Access, host-bypassing, peer-to-peer, SPDK, NVMe, FPGA

Typeset by the `thesis` package, version 4.13; `https://latex.fekt.vut.cz/`

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg . . . .

Unterschrift

ACKNOWLEDGEMENT

I would like to thank my supervisor, Prof. Dr. Dirk Koch for his valuable feedback when writing this thesis as well as in my research work at the Heidelberg University. I would also like to thank my colleague, Dr. Riadh Ben Abdelhamid, for providing necessary user feedback for the current course of research. Further, I give many thanks to my colleagues at CESNET, a.l.e., Ing. Jakub Cabal and Ing. Martin Špinler, for mentoring me in the development of PCIe-compliant IPs and software. Finally, I thank my mom, Ing. Marta Válková, for supporting me throughout my studies.

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Introduction

Here comes the introduction of the thesis, for example…

This thesis is devoted to **DSP!** (**DSP!**), especially it analyses the effect happening when the Nyquist condition for ***symfs!*** (**symfs!**) is not satisfied.[1]

The template is set to twoside printing by default. Do not be surprised that you find empty pages in your PDF. They are there to make the chapters and other important stuff begin on the right side when the document is printed. Having a serious reason to print one-sided, please switch the option `twoside` to `oneside`!

[1]This sentence is only to demonstrate how abbreviations can be used and typeset.

1 PCI Express

The *Peripheral Component Interconnect Express* (PCIe) is the interconnect standard that emerged in 2003 from the *Peripheral Component Interconnect* (PCI) standard. The PCIe addressed the drawbacks of the PCI that became increasingly challenging when more devices and more throughput were required. These include:

- Shared-bus topology that, inspite of its simplicity, was rendered unscalable since with increasing amount of devices, the bus gets longer which weakens the signal for the farthest device.
- Common clock which does not fit well with parallel data transmission model especially when lane skew takes place, meaning the bits of data travel unevenly fast over the wires.
- Half-duplex communication allows only one device to communicate at one point at a time.
- Reflected-wave signaling which was used to reduce the power consumption on the bus. This uses an interference of the sent wave from the master (which had driven the signal to the lower voltage than needed for the digital circuits on the bus) and its reflection that originated on the end of a bus (there is no termination on the bus' end). By the addition of these two waves, the signal reaches the required voltage level for the digital circuits (W: cite PCIe book).

The PCI protocol did not keep pace with increasing demand of computer peripheral communication structures and needed to be changed. The PCI Express protocol significantly changed the strategy for both the communication topology and the physical properties of the interconnect.

1.1 Communication model

The basic architecture is shown in Fig. XXX (host topology). Connections are implemented as point-to-point links between ports on different device types organized in a tree. The center of the topology is the *Root Complex* (RC), which provides *Root Ports*. Each device communicating over the infrastructure (such as today's GPU/FPGA accelerators) is called *Function* and is the major source of traffic within the system[1]. Each Function contains an *Upstream port* for exchanging data. Since the number of Root Ports is limited by the processor's interconnect, fan-out is provided by *PCIe Switches*, which supply multiple *Downstream ports* and a single Upstream port. Each port supports full-duplex communication and introduces asynchronous clocking model. Using this model significantly improves the possible throughput of the interconnect where clock gets reconstructed from the patterns in the incoming data on the receiving port. Using Switches, the number of devices can be increased arbitrarily, making PCIe a switched network (W: cite PCIe spec). Although the PCIe specification introduces many advanced features, this thesis presents mostly the necessary ones for its purposes.

Devices communicate using units called *Transaction Layer Packets* (TLPs), forming a packetized transfer. Each TLP contains a PCIe header used for identification followed by an optional payload. There are many TLP types, but the ones mentioned in this work are *Memory Read Request* (MRd) and *Memory Write Request* (MWr). Requests may require a response (called *non-posted*, as with MRds) in the form of *Completion* (Cpl) presenting a third TLP type. Such requests are called *non-posted* (like MRds) while requests that do not require a response are called *posted* (like MWrs). The device creating a request is called *Requester*, while the device 'completing' the request (sending a response or just accepting the payload) is called *Completer*.

The RC is located on the CPU die and tightly integrated with the processing cores to provide fast access to the system's I/O devices. The RC also provides the connection to host memory, which is the primary communication path used by DMA engines for *Host-to-Device* (H2D) and *Device-to-Host* (D2H) transfers. Multiple Root Ports or Downstream ports enable direct *Device-to-Device* (D2D) communication, as shown in Fig. XXX.

1.2 Link configuration

The PCIe specification gets published in versions called *generations*. With every new specification, the throughput of the bus gets doubled while keeping backward and forward compatibility between its standards (this is an important feature of PCIe). This is the reason, why the physical implementation of the protocol stack remains so complicated, especially on the physical layer. The physical connection between two ports consists of multiple differential pairs called *lanes* that are bundled together into a *link*. Each lane allows for full-duplex communication, where each direction is communicated using a separate differential pair. The link can either contain 1, 2, 4, 8 or 16 lanes.

As for the physical attachment interface, this thesis works with two of them, namely *Card Electromechanical* (CEM) and *M.2*. The CEM standard is the oldest

[1]The PCIe specification (with concordance to the PCI) defines three *System elements* to identify its devices, namely *Bus*, *Device* and *Function*. Every point-to-point connection in the topology makes a separate Bus together with internal (virtual) buses of the Switches and the Root Complex. The Device identifier stays as a remnant of the PCI standard and every Bus within the PCIe topology contains only one. Within each Device, one or multiple Functions can be present. The Function is basically and adressable entity in the configuration space where *Endpoint* is the most important type of a Function. Using the term 'device' in this thesis will refer to a general device in the PCIe topology that can contain one or multiple Functions (or Endpoints).

one used for *Add-in Cards* with GPU chips, FPGA accelerators or *Network Interface Cards* (NICs). It supports from 1 to 16 lanes with with wider link widths supporting lower ones and vice versa (these however with specially carved connectors that fit longer connector of the attached card). This form-factor allows for connection of the most power consuming peripherals where the connector can supply up to 75 W. The accelerators can require additional power by introducing external power supply attachements. The most common ATX connector adds another 75 W for its 6-pin configuration or 150 W for the 8-pin configuration (W: cite Intel's ATX standard).

The second type of physical attachement, *M.2*, is designed for ultra-light, power efficient platforms such as wireless modules or (as in the case of this thesis) *Non-volatile Memory Express* (NVMe) drives. This connector has a fixed set of 4 lanes and a variable module length ranging from 30 mm to 110 mm. These connectors present a detachable form factor for PCIe links but the standard allows for permanently attached devices to be connected as well.

As it is obvious from the variety of link configurations in terms of speeds and the amount of lanes, the multitude of PCIe devices inside a compute node needs to agree on common parameters in order to facilitate data transfer. This process of negotiation is called *link training* and it takes place after the network get powered up. The training is done on per-link-basis so there are multiple differently configured links in the system at a given point in time. This is presented on Fig. XXX where the M.2 SSD needs to communicate with the processor (where Root Ports can usually utilize up to 16 lanes) or with a NIC which supports up to 16 lanes. Other than link's lane count and transfer rate, other configured parameters include lane polarity and lane reversal. Since the training and the negotiated link parameters are tha matter of the physical layer, the link properties remain transparent for the user.

1.3 Device configuration

After the initial link training is done and every link is active, the host system does device discovery and configuration upon system boot in order to map the network. The process of discovery is called *enumeration* and involves a CPU communicating through the Root Complex in order to find every connected device in the topology and assign its BDF identifier (this includes assigning numbers also to buses between ports that do not necessarily connect to an Endpoint, like two Switches or a Switch with the Root Complex). The search is done in a depth-first way where each device is registered if it responds with a valid *Vendor ID* that is located within device's configuration registers. After assigning all BDFs in the topology, the configuration software further configures registers on the discovered devices. The configuration is done solely by the Root Complex in order to avoid the complexity of management when multiple devices (e.g. Endpoints) would try to configure each other. The initial assigning of configuration attributes is done using an IO-based transfer but further setting is done using *Memory-mapped I/O* (MMIO).

Each Endpoint contains configuration registers in a 64-byte *PCI Configuration Header*, which holds the most important attributes controlling the Function's ability to initiate transactions over PCIe and to be addressable within a host system. Firstly, since devices cannot generate Requests on their own by default, the *Bus Master* attribute enables this feature. Secondly, in order for adjacent devices to locate the current Function and to access its internal, user-defined registers/memory space, at least one of the *Base Address Registers* (BARs) must be initialized. By using BARs, each Endpoint requests a memory space within a host system for MMIO access. This is needed to access the application logic from the outside since each Endpoint accepts only those transactions whose addresses are in the ranges specified by its BARs. Each Function can register up to six 32-bit BARs which provides memory segmentation needed by some host applications. When 64-bit addressing is requested, two consecutive BARs are used. Throughout this work, the observation has been made that most devices allocate one to two BARs and, when not exceeding 4 GiB of space, an operating system maps them to 32-bit address ranges even when 64-bit addressing has been requested. First of all, this addressing has to be enabled in the UEFI settings of the host system.

Besides than the sheer capability for devices to be localizable in the system, other parameters are configured and need to be abided by by the user applications (e.g. the user-defined FPGA logic). A function defines its supported *Maximum Payload Size* (MPS) in the *Device Capabilities register*. Seeing this value, the configuration software sets its preferred value in a field in the *Device Control regiser*. The value of this field prohibits the Transmitter to dispatch TLP with greater payload than MPS specified and the Receiver to process them. The configuration allows furter adjustement using the *Maximum Read Request Size* (MRRS) in the *Device Control register* which prohibits the Transmitter to dispatch a Read Request greater than the specified size. Available values for these two parameters range from 128 to 4096 bytes (in powers of two steps). The configuration software balances these parameters based on latency/bandwidth requirements in the network. The configuration has to take account of the pairs of adjacent devices communicating with each other since receiving a TLP on a function's port whose size is larger than its set MPS results in the function rejecting this packet and reporting a fatal error[2].

[2]The PCIe standard does not require System Elements to perform segmentation of large TLPs.

1.4 Peer-to-peer transfer

An important aspect of PCIe (with its predecessor as well) and the structure of its network allowed to introduce *Direct Memory Access* (DMA) that allows to copy large amounts of data withouth the host CPU facilitating such copying. The usual approach it the exchange of data between PCIe devices and the host CPU through buffers in the host memory. However, the network infrastructure to address other devices in the topology as well. This type of D2D transport is called *Peer-to-peer* (P2P) or *Host Bypassing* and makes a fundamental part of this thesis. There are several implications of running such transport in the host system. The specification makes the support for each Switch mandatory (for transport between its Downstream ports) but optional for the Root Complex (for transport between its Root Ports) (source to spec). The routing of TLP differes based on its type with MWrs and MRds being routed by memory address while Cpl being routed by Requester's BDF identifier.

Today's host CPUs untilize *I/O Memory Management Unit* (IOMMU) to translate physical address to virtual ones in order to securely split I/O peripherals into groups. This unit is crucial in multi-tenant environments (like guests as virtual machines) where only devices in one group can perform P2P transfers between each other. Otherwise, where communication between two groups is required, the data need to be analyzed by the CPU. Without IOMMU a potentially malicious device can write and read from every region in the host system. The unit works hand in hand with the PCIe's *Access Control Services* (ACS) which forces every traffic to pass through the RC even if the devices are connected to the single Switch (depicted in Fig XXX). Otherwise, all devices under each Root Port are put to a common IOMMU group which does not play well with the need to decide isolation on per-device basis.

Considering the test server used in this work, the support for P2P transferes has been discovered as well as the presence of multiple IOMMUs and enabled ACS on the Root Ports. Since security is not a topic of this thesis nor are the system considerartions in the multi-tenant environments, both of these funcionalities have been disabled in server's UEFI. Therefore, the PCIe devices communicate using physical addresses.

2 Integrated core for PCIe on FPGAs

Modern FPGAs provide hardened IP core that implements the whole PCIe protocol stack (generally implementing a Port) that can be configured either as a Root Port (not as a RC though) or an Upstream Port (mostly for Endpoint, sometimes for Switch on some devices). The IP utilizes high-speed serial transceivers located on the edge of the chip that connect directly to the CEM-compliant Edge connector on the PCB's side. This work utilizes the *Alveo U55C* data-center acceleration card which is equipped with `xcu55c-fsvh2892-2L-e` chip (this corresponds to the serial chip `vu57p`). The available integrated core called *PCIE4C* allows to configure the interface to run on Gen3 transfer rat on 16 lanes (Gen3x16) or Gen4 transfer rate on 8 lates (Gen4x8) providing the same throughput but reducing logic utilization. Since two of these cores are provided close to each other and a total of 16 lanes is provided, these cores can split the available lanes and the acceleration card can be configured to run in a *PCIe bifurcation* mode. This mode utilizes a single PCIe link running two physically separated devices (each containing its own set of Functions). Therefore, the acceleration card can reach a cummulative data trasfer rate of PCIe Gen4x16.

2.1 Interfaces to user logic

Since the described core implement the whole PCIe protocol stack (meaning from Physical to Transaction layer), the user logic does not have to deal with task of the lower layers, such as transaction buffering, flow control credit management, data integrity check, lane deskew or clock recovery. In terms of the PCIe topology, this core implements an Upstream port (specifically as a PCIe Endpoint). The core provides four separate interfaces for the user logic where each follows the *AXI4-Stream* specification. As mentioned before, each PCIe port allows for full-duplex communication, allowing each device to act both as a Requester and as a Completer at the same time. The core contains these separate interfaces exactly in concordance with this behavior which include:

***Requester Request* (RQ)** allows user logic to send requests to other components in the PCIe hierarchy.

***Requester Completion* (RC)** transfers completions for non-posted requests dispatched on the RQ interfaces.

***Completer Request* (CQ)** receives requests from other devices in the PCIe hierarchy and provides them for the user logic.

***Completer Completion* (CC)** is used by the user logic to send responses for non-posted requests received on the CQ interface.

Other than that, the core provides additional interfaces for transferring status information either from the link or from user logic, dispatching and receiving interrupts, sending messages, etc. The PCIe IP maintains its set of status registers as every other PCIe Endpoint that are accessible by an operating system. A separate interface for the Requester side can be used to manage flow control credits but this is not used in this work and the management is left on the PCIe IP.

The width of the AXI4-Stream interfaces depends on the configured transfer rate of the IP. This work implements the DMA engine using 8-lane Gen4 PCIe endpoint which results in a 512-bit wide bus running on 250 MHz. All of the interfaces are running on thesame clock that is provided as the 'user clock' output from the IP (cite PCIe IP spec). Data units communicated on these buses are adjusted TLPs where every is formed as a *TLP header* followed by a payload of user data. The header uniquely identifies a payload within the PCIe topology with a physical address to the reserved space in the host memory or in another PCIe BAR. TLP header has a fixed size of 3 DW (12 B) for RC/CC interfaces and 4 DW (16 B) for RQ/CQ interfaces.

Using wide buses carries a significant drawback when data do not fill whole bus words. For example, transferring 1 B of data using a 512b bus utilizes only about 1.5 % of the bandwidth (1 B of payload plus a 16 B TLP header). To mitigate this inefficiency, the bus word is separated into two segments of equal size. Each segment can contain up to one TLP start and/or one TLP end. This doubles the throughput for packets that fit entirely within a single segment and generally improves throughput when a packet ends in the first segment (the second packet can then begin in the second segment).

3 Non-volatile Memory Express

The NVMe standard emerged in 2011 as a way of integration of non-volatile storage into the PCIe domain[1]. The standard emerged as a successor of *SAS* and *SATA* specifications in order to meet the requirements of consumer systems in terms of latency, throughput and scalability that fits with the flash-based storage. All of these standards introduced a queue mechanism where commands (like write and read) are present and processed by the controller of the attached *Non-volatile Media* (NVM). Both, SATA and SAS support one command queue allowing up to 32 and 256 commands respectively. NVMe enhances this mechanism by allowing to instantiate up to 2^{16} queues where each can contain up to 2^{16} commands. Allowing to have many queues results in hardware-aided parallel access to the NVM where, for example, each CPU core receives its own queue. The schema of the NVMe device is depicted in Fig. XXX. The *NVMe controller* sits on top of a PCIe controller, managing the instantiation of queues and processing of commands. The non-volatile storage is presented as a *Namespace* which is the basic referencable unit of storage in the NVMe standard[2]. A namespace is split into multiple sectors marked by *Logical Block Addresss* (LBAs) which are at least 512 B in size (which is mostly used one). The controller is equipped with a DMA engine that performs a read and write of raw data between memory regions in the host system. This implies that the NVMe device behaves as a Bus Master and can therefore generate its own MWrs and MRds which is suitable for the P2P communication in the PCIe domain.

The command mechanism provides two types of queues that always occur in pairs, the *Submission queue* and the *Completion queue*. A Submission queue contains commands that need to be processed by the NVMe controller which, after a command's execution, publishes a result of an operation into a Completion queue. These queues are allocated as circular buffers in a host-addressable memory. The most important pair is the *Admin queue-pair* that is used for additional configuration. The controller appears in the PCIe system as a single Function and, initially, the kernel is able to only reach the standard registers in the PCIe Configuration Space and its BARs. The NVMe driver allocates Admin queue-pair and publishes base addresses and sizes of the queues to controller registers located in BAR0.

For data trasmission to/from the NVMe namespace, the configuration driver in-

[2]This thesis works with specification version 1.2b since it describes only its basic concepts and not every currently working NVMe device supports the newest standard. The NVMe specification is designed to be backward compatible though. Using older specifiactions may be advised for boarding users to understand the basic concepts quickly.

[2]This is a simplified view in the referenced version of the NVMe specification. Later versions define a more complex and fine-grained classification beyond namespaces. Moreover, there can be many namespaces maintained by a single NVMe controller as well as one namespace shared by many controllers.

stantiates one or more *I/O queue-pairs* which are used by the consumers/producers of data. Same as for the Admin queue-pair, these queues are allocated circular buffers in host-addressable memory. A fully initialized system with a 4-core processor and an NVMe device is shown in Fig. XXX. The controller can associate multiple I/O Submission queues with one I/O Completion queue in order to save resources (this is not possible for the Admin queue-pair though since only one can be instantiated). Each queue pair is identified by its ID where Admin queue-pair has a fixed ID of 0 and every other I/O queue pair reserves an ID of a higher index.

3.1 Principles of operation

One queue 2 pointers for its operation, a *Head doorbell* and a *Tail doorbell.* A consumer of entries from the queue operates a Head doorbell whereas every producer operates a Tail doorbell. This pair of pointers ensures that a producer does not overwrite unprocessed entries and that a consumer reads only valid ones. A queue is considered empty if values of these pointers are equal, and full if a value of the Tail doorbell is equal the value of the Head doorbell minus one modulo a size of the queue as is shown in Fig. XXX.

A master submits a command to a submission queue by using a template for *Submission Queue Entry* (SQE) described further. Upon storing such entry in the queue, the master updates the *Submission Queue Tail Doorbell* (SQTDBL) pointer in controller registers (in BAR0) in order to signalize the controller that new command has been submitted. This is followed by the controller fetching and executing the command followed by dispatching a completion into a completion queue that follows the *Completion Queue Entry* (CQE) template. If a command operates with additional data (like reading/writing data from/to the NVM or fetching commands from a submission queue), the controller engages the DMA engine on the device. To detect a received completion of a command, the master does not have to read the *Completion Queue Tail Doorbell* (CQTDBL) value (since this is not available and remains internal to the NVMe controller anyways). Rather, every CQE contains a *Phase tag (P)* that indicates that a position in the Completion queue contains a valid entry.

3.1.1 Command submission

The SQE is created by a master by following a format in Fig. XXX (the grey marked fields are either unused or reserved) and it uniquely defines a command within a single NVMe controller. The entry has a length of 64 B (16 DW) with necessary information for a controller to process a command. There are two sets of commands defined by the NVMe specification, namely *Admin Command Set* and *NVM Command Set*[3]. The mostly used used commands in this thesis on runtime are read and write from/to the NVM that are described in the following paragraph.

An SQE begins with the *Opcode* (OPC) field that distinguishes between different types of commands, the *PSDT* field that specifies that *Physical Resource Pages* (PRPs) are going to be used as copy buffers and the *Command Identifier* (CID) field that presents a unique identifier when combined with the Submission Queue ID. This combination serves as a tag to match the dispatched command with a received completion. The first command DWord is followed by *Namespace Identifier* (NSID) to index a specific namespace that is managed by the adjacent NVMe controller (the index is 1-based, meaning that the first namespace receives NSID 1). The following *Data Pointer* (DPTR) field serves as a pointer to the host-adressable memory either from which data need to be fetched (as for the write command) or to which the data are copied from the NVM (as for the read command). The data pointer is split into two PRP entries depending on how many memory pages contain the operated data in host memory, thus how big the transfer is going to be. If the transfer fits into one page, only *PRP Entry 1* (PRP1) is used and it points to the start of the user data. In case a transfer, thus operated data, spans over 2 memory pages, the *PRP Entry 2* (PRP2) is used as well to point to the second page. If a transfer spans more than 2 pages, the PRP2 changes its meaning to point to the *PRP List* which contains pointers to every other page that contains to the user data. The following two Dwords (DW10 and DW11) contain the *Starting LBA* (SLBA) field in the namespace the last usable field in DW12 contains the *Number of Logical Blocks* (NLB) to copy. This size determines a size of a transfer for a currently processed command. The size of a transfer is limited by the NVMe controller and every master has to consult controller parameters in order to not submit larger transfers. When larger data chunks are required than a transfer can process, multiple commands have to be submitted.

3.1.2 Command completion

After a command is processed the controller dispatches a CQE to the Completion queue to notify the master about the status of a command. The template of every entry is shown in Fig. XXX. A CQE is at least 16 bytes in size. The first DWord is used by some commands to publish command-specific information (the

[3]In terms of more recent specification revisions, these sets use a *memory-based* transport which differs from *message-based* transport. While the first type of transport is typical for PCIe, newer specifications allow to attach NVM devices to other fabrics, like RDMA or Ethernet, that use the second type of transport. These are part of the *NVMe over Fabric* (NVMe-oF) standard published separately.

Read and Write commands do not use them though) while the second DWord is reserved. An NVMe controller indicates the internal value of *Submission Queue Head Doorbell* (SQHDBL) through every completion entry which notifies the transmitter of commands about how many commands have been read by the controller (thus providing a backpressure). The *SQ Identifier* (SQID) and CID fields serve a unique identifier of a command in the communication chain. The SQID is present because the NVMe specification allows to associate multiple Submission queue with one Completion queue (this applies for one NVMe controller though since every controller maintains its own set of queues). A *Phase Tag (P)* field follows and its value indicates that the current position in the Completion queue is valid (meaning the position to which *Completion Queue Head Doorbell* (CQHDBL) points to). The value that is considered valid is flipped upon every CQTDBL rollover, starting as logical 1 for the initial pass through queue (i.e. after queue initialization), then flipping to 0 when the pointer rolls back to 0, then to 1 again and so on.

The last bits of a CQE are occupied by status information about the successful/unsuccessful execution of a command. The *Do Not Retry* (DNR) bit indicates that if the same command is going to be resubmitted, it is expected to fail. The submitter has to wait till this bit clears to 0 in order to receive valid data. The *More (M)* bit indicates that a completion for a command is going to be composed out of multiple CQEs. A more specific information about a command's status can be found in the *Status Code Type* (SCT) and *Status Code* (SC) fields. A NVMe controller provides coarse-grained category of status within the first field whereas a more fine-grained in the second one. Here are some examples of possible status codes received:

- A successfull completion is indicated by the SCT field set to 0h (i.e. *Generic Command Status*) and the SC field set to 00h (i.e. *Successful Completion*).
- When a submitter dispatches an NVMe command with a specified LBA range that exceeds the capacity of the Namespace, the SCT field is set to 00h and the SC field to 80h (i.e. *LBA Out of Range*).
- Submitting two NVMe Write/Read commands with the same CID results in a completion status where SCT is set to 1h (i.e. *Command Specific Status*) and SC set to 80h (i.e. *Conflicting Attibutes*).

3.1.3 Ordering

An important note has to be made with regards to ordering of submitted commands. An NVMe controller arbitrates between its own set of submission queues in a round-robin fashion and can fetch multiple commands out of every queue[4]. The order of execution where many commands are fetched is not defined by the NVMe specification and can be arbitrary. This is the reason why a tagging of commands using the CID field is important so the submittier can bound a completion to a dispatched command. The order of execution can therefore be derived from the order in which the completions for mutliple commands have been received. If a strict ordering of commands is required, a submitter needs to implement its own for that. However, by the means of the NVMe specification, there are two ways of how to ensure ordering. The standard allows to use *Fused operations* in order to force ordering of two subsequent commands. Such two commands can be executed atomically, resulting in the success only if both of the commands succeed. However, this feature is optional and, therefore, implementation specific. The second way that is always available, is to submit only one command to the submission queue and wait for its completion before submitting another commands. This can be guarded by the submitter itself or by initializing the submission queue with the size of 2 (since this can contain at most 1 valid SQE) and using the blocking by the doorbell pointers. The second way is used in the implementation proposed by this thesis.

3.2 Multi-pathing

Since multiple queues can be initialized for a single NVMe controller, multiple submitters can execute commands on the controller. Since the proposed implementation in this thesis introduces a direct NVMe device access from an FPGA acceleration card using the PCIe P2P transfer, it contains a potential to access multiple NVMe devices (the 1-to-M topology) as well as to access multiple NVMe devices from multiple acceleration cards, thus implementing an N-to-M topology[5]. These can be seen in Figure XXX and XXX. This feature that allows to access one NVMe device from multipler masters is sometimes called *multi-pathing*.

[4]Advanced devices can implement a Weighted Round-robin or some vendor-specific mechanism.

[5]This topoolgy, however, is only virtual since it is made only by addressing and not by physical connections. When performance (like throughput) is critical, it should be adapted on the underlying PCIe topology which is organized as a tree by design.

4 DMA Iuventus

Since this thesis proposes a solution that is able to access the filesystem located on an NVMe device that is shared between the host OS and an FPGA accelerator a specific IP core is needed on the accelerator's side. Without this, the user logic inside an FPGA would deal with raw PCIe data not only from the adjacent NVMe device but from the host system as well. Therefore, the *DMA Iuventus* IP core has been developed to process requests between an accelerator and an NVMe device. The DMA description has been retained to highlight the mechanism of the direct copying of data to the adjacent storage without a host CPU facilitating such transfers (this is sometimes called *Host bypassing*). This IP sits on the integrated block for PCIe of the FPGA to accept NVMe-specific traffic alongside with a logic that processes transactions to *Configuration and Status* (C/S) registers.

4.1 TLP Headers

The DMA IP utilizes 3 of the 4 interfaces provided by the integrated block for PCI Express, namely RQ, CQ, CC. Each on of these uses its own template of the TLP header that the DMA core needs to create or process. The RQ header has a size of 4 DW and its format is depicted in Fig. XXX. The *Address* field is aligned with respect to DWords since its two bottom bits are reserved[1]. For MWrs, the *Dword Count* field specifies the size of the payload following the header, whereas, for MRds, it specifies the required amount of data that a Completer needs to send in its Completion. The type of a request is defined in the *Request Type* field. The *Function* field informs the PCIe IP which Function generated the request (more on that later). The last used field is the *NoSnoop* bit located in the *Attr* field. Asserting this bit instructs the the host CPU that the underlying address space the transaction is targeted to, does not require a hardware-enforced cache coherency (cite the PCIe spec). This means that the address space is not shared by any other device in the system. The RQ interface is used to dispatch SQTDBL and CQHDBL updates to the NVMe registers.

The CQ header (Fig. XXX) follows a similar format as the RQ header where the user logic acts as a PCIe Completer. The *Target Function* field indicates to which the Function is targeted. When multiple BARs are used, the core puts index of the specific one in the *BAR ID* field. The size of a targeted BAR is determined by the *BAR Aperture* which serves as a mask for the address (for example, the value of 12 signifies that the BAR has a size of 4 KiB and, therefore, the address bits [63:12]

[1]The x86_64 architecture addresses its physical memory space with resolution to bytes.

can be ignored). Other attributes of this header are either reserved or used to create a response for MRd.

For posted requests, such as MRd, the response is dispatched on the CC interface by using the 3 DW TLP header as in Fig. XXX. The header may be followed by a payload. The *Address* is a byte-level address of the first byte of the memory block being transferred. Each completion transferes a status of either *Successful Completion*, *Unsupported Request* or *Completer Abort* in the *Completion Status* field. The size of the payload is specified in the *Dword Count* field. Similar to the Target Function field in the RQ header, there is an index of a completing Function in the *Completer ID* fields. Furthermore, there are multiple fields that need to be copied from the Requester TLP header, such as *Address Translation* (AT), *Requester ID*, *Tag*, *Transaction Class* (TC) and *Attributes* in order that the Requester correctly classifies the received Completion. This applies especially for the Tag field which marks all Completions for a specific Request. A Completion can be dispatch in multiple TLPs, especially if the value of MRRS

Since there is only one PCIe IP per one physical connector, all of the requests pass through same interfaces to the user logic and have to be split according to the function index and the BAR ID.

Conclusion

Thesis conclusion.

Bibliography

[1] VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ. *Směrnice č. 72/2017, Úprava, odevzdávání a zveřejňování závěrečných prací.* Online. Brno: VUT v Brně, 2017. Úplné znění ke dni 11. 4. 2022. Dostupné z: `https://www.vut.cz/uredni-deska/vnitrni-predpisy-a-dokumenty/smernice-c-72-2017-uprava-odevzdavani-a-zverejnovani-zaverecnych-praci-d161410`. [cit. 2023-09-27].

[2] ÚŘAD PRO TECHNICKOU NORMALIZACI, METROLOGII A STÁTNÍ ZKUŠEBNICTVÍ. ČSN ISO 690:2022 (01 0197), *Informace a dokumentace – Pravidla pro bibliografické odkazy a citace informačních zdrojů.* Čtvrté vydání. Praha, 2022.

[3] ÚŘAD PRO TECHNICKOU NORMALIZACI, METROLOGII A STÁTNÍ ZKUŠEBNICTVÍ. ČSN ISO 7144 (010161), *Dokumentace – Formální úprava disertací a podobných dokumentů.* Praha, 1997.

[4] ÚŘAD PRO TECHNICKOU NORMALIZACI, METROLOGII A STÁTNÍ ZKUŠEBNICTVÍ. ČSN ISO 31-11, *Veličiny a jednotky – část 11: Matematické znaky a značky používané ve fyzikálních vědách a v technice.* Praha, 1999.

[5] FARKAŠOVÁ, B. et al. *Výklad normy ČSN ISO 690:2022 (01 0197) účinné od 1. 12. 2022.* Online. První vydání. 2023. Dostupné z: `https://www.citace.com/Vyklad-CSN-ISO-690-2022.pdf`. [cit. 2023-09-27].

[6] *Pravidla českého pravopisu.* 1. vydání. Olomouc: FIN, 1998.
ISBN 80-86002-40-3.

[7] WALTER, G. G. a SHEN, X. *Wavelets and Other Orthogonal Systems.* 2. vydání, Boca Raton: Chapman & Hall/CRC, 2000. ISBN 1-58488-227-1

[8] SVAČINA, J. Dispersion Characteristics of Multilayered Slotlines – a Simple Approach. *IEEE Transactions on Microwave Theory and Techniques.* 1999, vol. 47, no. 9, s. 1826–1829. ISSN 0018-9480.

[9] RAJMIC, P. a SYSEL, P. Wavelet Spectrum Thresholding Rules. In: *Proceedings of the International Conference Research in Telecommunication Technology.* Žilina: Žilina University, 2002. s. 60–63. ISBN 80-7100-991-1.

Symbols and abbreviations

PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
DMA	Direct Memory Access
NVMe	Non-volatile Memory Express
FPGA	Field-programmable Gate Array
P2P	Peer-to-peer
RC	Root Complex
EP	Endpoint
TLP	Transaction Layer Packet
DLLP	Data-link Layer Packet
MRd	Memory Read Request
MWr	Memory Write Request
Cpl	Completion
H2D	Host-to-Device
D2H	Device-to-Host
C2H	Card-to-Host
H2C	Host-to-Card
D2D	Device-to-Device
CEM	Card Electromechanical
NIC	Network Interface Card
SSD	Solid-state Drive
MMIO	Memory-mapped I/O
BAR	Base Address Register
MPS	Maximum Payload Size

MRRS	Maximum Read Request Size
IOMMU	I/O Memory Management Unit
ACS	Access Control Services
RQ	Requester Request
RC	Requester Completion
CQ	Completer Request
CC	Completer Completion
NVM	Non-volatile Media
SQE	Submission Queue Entry
CQE	Completion Queue Entry
SQTDBL	Submission Queue Tail Doorbell
SQHDBL	Submission Queue Head Doorbell
CQTDBL	Completion Queue Tail Doorbell
CQHDBL	Completion Queue Head Doorbell
NVMe-oF	NVMe over Fabric
OPC	Opcode
PRP	Physical Resource Page
CID	Command Identifier
SQID	SQ Identifier
NSID	Namespace Identifier
LBA	Logical Block Address
DPTR	Data Pointer
PRP1	PRP Entry 1
PRP2	PRP Entry 2
SLBA	Starting LBA

NLB	Number of Logical Blocks
DNR	Do Not Retry
SCT	Status Code Type
SC	Status Code
C/S	Configuration and Status
AT	Address Translation
TC	Transaction Class

List of appendices

A Selected Commands of `thesis` Package

A.1 Quantities and Units

Table A.1: An overview of commands (use within the mathematical environments).

Command	Example	LaTeX code of example	Meaning
`\textind{...}`	β_{max}	`$\beta_\textind{max}$`	text-style index
`\const{...}`	U_{in}	`$\const{U}_\textind{in}$`	constant
`\var{...}`	u_{in}	`$\var{u}_\textind{in}$`	variable
`\complex{...}`	$\boldsymbol{u}_{\mathrm{in}}$	`$\complex{u}_\textind{in}$`	complex variable
`\vect{...}`	$\mathbf{y}$	`$\vect{y}$`	vector
`\mat{...}`	$\mathbf{Z}$	`$\mat{Z}$`	matrix
`\unit{...}`	kV	`$\unit{kV}$` or `\unit{kV}`	unit

A.2 Symbols

- `\E`, `\eul` – typesets the Euler number: e,
- `\J`, `\jmag`, `\I`, `\imag` – imaginary unit: j, i,
- `\dif` – the differential: d,
- `\sinc` – the function sinc,
- `\mikro` – typesets the *micro* symbol in roman type[1]: µ,
- `\uppi` – typesets π (greek pi in roman type, in difference to `\pi`, which typesets π).

All symbols are considered to be used within a math mode, except `\mikro` that is possible in the text mode as well.

[1]the symbol comes from package `textcomp`

B Next Appendix

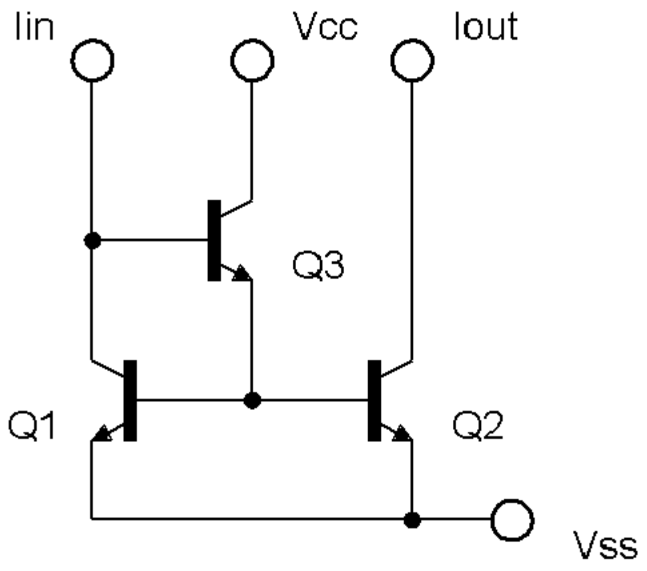

Fig. B.1: Improved Wilson current mirror.

For inclusion of the vector-based graphics directly via LaTeX, it is possible to use the `TikZ` package. Examples of use can be found at the TeXample site. TikZ graphics creation is supported in QTikz and TikzEdt software.

C Examples of Listing Computer Codes

C.1 Package `listings`

Listing computer codes can be handled efficiently via the `listings` package. This package introduces a new environment `lstlisting` for typesetting computer codes, as for example:

```
\section{Package lstlistings}
Listing computer codes can be handled efficiently
via the \texttt{listings} package.
This package introduces a new environment
\texttt{lstlisting} for typesetting computer codes.
```

The package supports a number of programming languages. The code to be typeset can be input directly from files on disk. The package allows row numbering and extracting only selected parts of the code. The following paragraph is an example of the use of `listings`:

Abbreviations are typeset with the `acronym` environment:

```
\begin{acronym}[SQTDBL]
```

The width of the input parameter, `HowMuchSpace`, determines the width of the first column. An example of the definition of abbreviation **symfs!** is in Listing C.1.

Listing C.1: Example of code listing.

The list is finished with the end of the environment:

```
    \acro{rcpx}[RC]
```

Listing C.2 contains an example of code for Matlab, whereas in Listing C.3 you find an example in the C language.

Listing C.2: Example of the Schur–Cohn test of stability in Matlab.

```
%% Priklad testovani stability filtru

% koeficienty polynomu ve jmenovateli
a = [ 5, 11.2, 5.44, -0.384, -2.3552, -1.2288];
disp( 'Polynom:'); disp(poly2str( a, 'z'))

disp('Kontrola␣pomoci␣korenu␣polynomu:');
zx = roots( a);
if( all( abs( zx) < 1))
    disp('System␣je␣stabilni')
else
    disp('System␣je␣nestabilni␣nebo␣na␣mezi␣stability');
end

disp('␣'); disp('Kontrola␣pomoci␣Schur-Cohn:');
ma = zeros( length(a)-1,length(a));
ma(1,:) = a/a(1);
for( k = 1:length(a)-2)
    aa = ma(k,1:end-k+1);
    bb = fliplr( aa);
    ma(k+1,1:end-k+1) = (aa-aa(end)*bb)/(1-aa(end)^2);
end

if( all( abs( diag( ma.'))))
    disp('System␣je␣stabilni')
else
    disp('System␣je␣nestabilni␣nebo␣na␣mezi␣stability');
end
```

Listing C.3: Example of implementation of first canonical form in C.

```c
// first canonical form
short fxdf2t( short coef[][5], short sample)
{
  static int v1[SECTIONS] = {0,0},v2[SECTIONS] = {0,0};
  int x, y, accu;
  short k;

  x = sample;
  for( k = 0; k < SECTIONS; k++){
    accu = v1[k] >> 1;
    y = _sadd( accu, _smpy( coef[k][0], x));
    y = _sshl(y, 1) >> 16;

    accu = v2[k] >> 1;
    accu = _sadd( accu, _smpy( coef[k][1], x));
    accu = _sadd( accu, _smpy( coef[k][2], y));
    v1[k] = _sshl( accu, 1);

    accu = _smpy( coef[k][3], x);
    accu = _sadd( accu, _smpy( coef[k][4], y));
    v2[k] = _sshl( accu, 1);

    x = y;
  }
  return( y);
}
```

D Algorithms

For typesetting algorithms/pseudocode, the `algorithm2e` package can be used, offering rich options. Algorithms can be referenced by usual cross-references, as here: see Alg. 1.

Algorithm 1: B-PHADQ

```
Choose parameters $\tau, \sigma, > 0,\ \rho \in [0, 1]$
Initialize $x^{(0)}, p^{(0)}, q^{(0)}$ and $\omega_{\mathbf{s}}$
for $i = 0, 1, \dots$ do
    $q^{(i+1)} = \mathrm{clip}_\lambda(q^{(i)} + \sigma D R_{\omega_{\mathbf{s}}} G_g x)$
    $u = p^{(i)} - \tau G_g^* R_{\omega_{\mathbf{s}}}^* D^* q^{(i+1)}$   % auxiliary
    $p^{(i+1)} = \mathrm{proj}_\Gamma(u)$   % consistent
    $p^{(i+1)} = \frac{1}{\tau+1}(\tau\ \mathrm{proj}_\Gamma(u) + u)$   % inconsistent
    $x^{(i+1)} = p^{(i+1)} + \rho(p^{(i+1)} - p^{(i)})$
end
return $p^{(i+1)}$
```

E Content of the electronic attachment

An electronic attachment is often a part of the thesis. The attachment is uploaded in the BUT information system together with the thesis PDF. Please use an appropriate file format for the attachment.

It is suggested to comment on every folder, to specify which of the files contains main settings, to specify which is the main or executable file, what was the setting of the compiler etc. It is also valuable to specify in which version of the software the code has been tested (e.g. Matlab 2018b). In the case that hardware has been created within the thesis, the electronic attachment must contain all documentation (for example Eagle files with the printed circuit board layout).

If your attachment contains a lot of files or folders, LaTeX package `dirtree` can become handy, as in the following example.

```
/ ............................................ root of the attached archive
  logo ............................................................ logotypes
    BUT_abbreviation_color_PANTONE_EN.pdf
    BUT_color_PANTONE_EN.pdf
    FEEC_abbreviation_color_PANTONE_EN.pdf
    UTKO_color_PANTONE_EN.pdf
  pdf .......................... PDFs (generate them in the information system)
    assignment-example.pdf
    cover-example.pdf
    titlepage-example.pdf
  pict ................................................... other graphic files
    soucastky.png
    spoje.png
    ZlepseneWilsonovoZrcadloNPN.png
    ZlepseneWilsonovoZrcadloPNP.png
  text .......................................... LaTeX source codes of the text
    abbreviation.tex
    appendix.tex
    bibliography.tex
    conclusion.tex
    introduction.tex
    results.tex
    solution.tex
  template-thesis.tex ................................... main file of the thesis
  template-presentation.tex ........... main file of the slides for presentation
  thesis.sty ........................ package for typesetting final theses at BUT
```