BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FRAMEWORK FOR HARDWARE ACCELERATION OF 400 GB NETWORKS

FRAMEWORK PRO HARDWAROVOU AKCELERACI 400 GB SÍTÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. VÁCLAV HUMMEL**
AUTOR PRÁCE

SUPERVISOR **Ing. JAN KOŘENEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2017

Zadání diplomové práce/19303/2016/xhumme00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Hummel Václav, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Framework pro hardwarovou akceleraci 400Gb sítí**
Framework for Hardware Acceleration of 400Gb Networks
Kategorie: Počítačová architektura

Pokyny:

1. Nastudujte možnosti hardwarové akcelerační karty COMBO.
2. Seznamte se s možnostmi hardwarové akcelerace v oblasti virtualizace síťových funkcí.
3. Navrhněte framework, který bude umožňovat hardwarovou akceleraci virtalizovaných síťových funkcí a to až do rychlosti 400 Gb/s. Zaměřte se na komunikaci mezi akcelerační kartou a hostitelským počítačem.
4. Proveďte implementaci navrženého frameworku.
5. Ověřte funkčnost a vlastnosti vytvořené implementace.
6. V závěru diskutujte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kořenek Jan, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstract

The NetCOPE framework has proven itself as a viable framework for rapid development of hardware accelerated wire-speed network applications using Network Functions Virtualization (NFV). To meet the current and future requirements of such applications the NetCOPE platform has to catch up with upcoming 400 Gigabit Ethernet. Otherwise, it may become deprecated in following years. Catching up with 400 Gigabit Ethernet brings many challenges bringing necessity of completely different way of thinking. Multiple network packets have to be processed each clock cycle requiring a new concept of processing. Advanced memory management is used to ensure constant memory complexity with respect to the number of DMA channels without any impact on performance. Thanks to that, even more than 256 completely independent DMA channels are feasible with current technology. A lot of effort was made to create the framework as generic as possible allowing deployment of 400 Gigabit Ethernet and beyond. Emphasis is put on communication between the framework and host computer via PCI Express technology. Multiple Ethernet ports are also considered. The proposed system is prepared to be deployed on the family of COMBO cards, used as a reference platform.

Abstrakt

Platforma NetCOPE již prokázala svou životaschopnost jako framework pro rychlý vývoj hardwarově akcelerovaných síťových aplikací. Tyto aplikace využívají virtualizované síťové funkce (NFV). Aby platforma v nejbližších letech nezastarala, tak se musí přizpůsobit požadavkům souvisejících s příchodem 400 Gigabitového ethernetu. Příchod 400 Gigabitového ethernetu sebou přináší velké množství výzev, které vyžadují nutnost kompletně změnit dosavadní myšlení. Během jediného hodinového cyklu musí být zpracováno několik síťových paketů, což vyžaduje nový koncept zpracování. Je použita pokročilá správa paměti, aby byla dosažena konstantní paměťová složitost vzhledem k počtu DMA kanálů. Díky tomu je možné realizovat se současnou technologií i více než 256 kompletně nezávislých DMA kanálů. Mnoho úsilí bylo kladeno na vytvoření co nejobecnějšího frameworku i pro vyšší rychlosti přesahující 400 Gb/s. Práce se zaměřuje na komunikaci mezi frameworkem a hostitelským počítačem skrze PCI Express rozhraní. Byla vzata do úvahy i varianta s více síťovými rozhraními. Navržený systém je připraven na nasazení na kartách rodiny COMBO, které jsou použity jako referenční platforma.

Keywords

400 Gigabit Ethernet, hardware acceleration, FPGA, Network Functions Virtualization (NFV), PCI Express, Direct Memory Access (DMA), SZE, DPDK

Klíčová slova

400 Gigabitový ethernet, hardwarová akcelerace, FPGA, Virtualizované síťové funkce (NFV), PCI Express, Přímý přístup do paměti (DMA), SZE, DPDK

Reference

HUMMEL, Václav. *Framework for Hardware Acceleration of 400 Gb Networks.* Brno, 2017. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Kořenek Jan.

Framework for Hardware Acceleration of 400 Gb Networks

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Mr. Ing. Jan Kořenec Ph.D. The supplementary information was provided by the whole team Liberouter by Cesnet group. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Václav Hummel
May 23, 2017

Acknowledgements

Hereby I would like to thank my supervisor Mr. Ing. Jan Kořenek Ph.D. for thorough supervising of my master's thesis. Moreover, the whole team Liberouter by Cesnet group, especially to Mr. Ing. Viktor Puš Ph.D., for providing me vital information and hardware support. Special thanks belong to my wife Dani and my family for their unlimited support during my whole studies.

Contents

Glossary

400 GMII 400 Gigabit Media Independent Interface. 13

ADT Abstract Data Type. 19

ASIC Application-Specific Integrated Circuit. 7, 12, 13, 24

AXI Advanced eXtensible Interface. 42, 47, 48

BER Bit Error Rate. 13

BRAM Block Random Access Memory. 31–33, 51, 52, 55

CAPEX CApital EXpenditure. 18

CLB Configurable Logic Block. 24

CPU Central Processing Unit. 6, 7, 14, 16, 18, 19, 25–28, 30, 37, 38, 56

CRC Cyclic Redundancy Check. 11, 15, 23, 47, 48

CSMA/CD Carrier Sense Multiple Access with Collision Detection. 11

DC Direct Current. 21

DDoS Distributed Denial of Service. 17, 39

DF Data Fetch. 36

DIC Deficit Idle Count. 12

DLLP Data Link Layer Packet. 21

DMA Direct Memory Access. 25, 28–33, 35, 36, 38, 40, 42–45, 47, 48, 51–53, 56

DPDK Data Plane Developement Kit. 8, 18, 19, 26, 56

DPI Deep Packet Inspection. 17

DSP Digital Signal Processing. 24

EAL Environment Abstraction Layer. 19

EX EXecution. 36

FEC Forward Error Correction. 13

FIFO First In First Out. 35, 42, 43, 45

FLU FrameLink Unaligned. 29, 34

FPGA Field Programmable Gate Array. 7, 12, 13, 18, 23–25, 30, 34, 36, 37, 39, 42, 47, 56

GPS Global Positioning System. 25

GPU Graphics Processing Unit. 14

HBM High Bandwidth Memory. 26

HDL Hardware Description Language. 24

HMC Hybrid Memory Cube. 26

HTTP HyperText Transfer Protocol. 9

IBUF Input BUFfer. 26

IDS Intrusion Detection System. 15, 16

IEEE Institute of Electrical and Electronics Engineers. 10

IP Internet Protocol. 6, 10, 15, 16

IPS Intrusion prevention System. 15, 16

IPv6 Internet Protocol version 6. 15, 16

ISO International Organization for Standardization. 8–11, 15, 16, 20, 21, 38, 39, 41, 47

ISP Internet Service Provider. 16

JTAG Joint Test Action Group. 25

LAN Local Area Network. 10

LPM Longest Prefix Match. 16

LSB Least Significant Bit. 10–12

LUT LookUp Table. 24, 31, 33, 51

MAC Media Access Control. 10

MI32 Memory Interface 32-bit. 27

MII Media Independent Interface. 13, 14, 25, 29, 33, 40, 41, 46–48

MPLS MultiProtocol Label Switching. 38

MSB Most Significant Bit. 10–12

MTU Maximum Transmission Unit. 41, 43, 45

NAT Network Address Translation. 18

NFV Network Functions virtualization. 6, 7, 17, 18

NIC Hardware Accelerated Network Interface Card. 25

NIC Network Interface Card. 6, 8, 18, 19, 25, 30, 35–39

NRZ Non-Return-to-Zero. 12, 13

NUMA Non-Uniform Memory Access. 37

OBUF Output BUFfer. 26

OPEX OPerating EXpenditure. 18

OSI Open Systems Interconnection. 8–11, 15, 16, 20, 21, 38, 39, 41, 47

OVS Open Virtual Switch. 6, 19

PAM4 4 level Pulse Amplitude Modulation. 12, 13

PAM8 8 level Pulse Amplitude Modulation. 13

PC Personal Computer. 8, 19

PCI Peripheral Component Interconnect. 20

PCI Express Peripheral Component Interconnect Express. 8, 19–25, 37, 38, 40, 42–49, 51, 56

PCS Physical Coding Sublayer. 12, 26

PMA Physical Medium Attachment. 12, 26

QDR Quad Data Rate. 25, 26

QoS Quality of Service. 9, 15, 16, 38

RF Register Field. 36

RX Reception. 8, 13, 19, 26, 30–35, 39, 40, 42, 43, 48, 49, 51–53

SDRAM Synchronous Dynamic Random Access Memory. 26

SoC System on Chip. 19

TCP Transmission Control Protocol. 10

TLP Transaction Layer Packet. 21, 22

TX Transmission. 8, 13, 19, 26, 30–32, 34, 39, 40, 42, 48, 49

UH Unified Header. 38

VHDL Very high speed integrated circuit Hardware Description Language. 25

VLAN Virtual Local Area Network. 11, 38

VM Virtual Machine. 18

WAN Wide Area Network. 10

WB Write Back. 36

WLAN Wireless Local Area Network. 10

Chapter 1

Introduction

The throughput of network links is growing very fast. According to the white paper, published by Cisco Systems, Inc [4], the annual global IP traffic will more than double by the year 2020, compared to 2016. Network traffic amount is growing exponentially just like the world population. Nobody knows where it stops. The Internet is everywhere and increasing number of people are using it on daily basis – not just for entertainment but also for many critical operations such as bank transfers, sensitive data sharing within companies and governments, home and industry automation, public transportation control, etc. Dependence on the Internet is a dangerous precedent – collapse may cause a global disaster. This master's thesis addresses future problems related to increasing amount of network traffic and its global importance.

Increasing popularity of Cloud computing in data centers [5] causes network traffic is being concentrated in relatively small areas. To solve problems, mentioned in the previous paragraph, means to solve problems of data centers and backbone networks. In this work, network traffic related problems will be considered such as increasing amount of data and importance of security.

Typical data center consists of thousands of servers. Each of these servers contains multiple CPUs (typically 2) which have multiple cores (typically 16-32, likely to grow more) each of them capable of running multiple simultaneous threads (typically 2). Moreover, the popularity of virtualization is increasing. One server can be used as a host to many virtual machines which compete for shared resources e.g. Network Interface Card (NIC). Virtual machines are run by hypervisor – usually software layer responsible for running virtual machines. These virtual machines communicate with each other via a network. Since these virtual machines share typically just one NIC it is not possible to switch network packets in a usual way outside server in a network switch – hypervisor has to implement switch as an additional (software) layer. This problem is addressed by Open Virtual Switch (OVS or Open vSwitch) [8].

Since network traffic and computation in data centers is rapidly increasing there is an increasing demand for performance. Unfortunately, this demand cannot be fulfilled by general CPUs since their performance increases at much lower rate than network traffic. Moreover, a huge emphasis is put on power consumption. With current technologies, these requirements can be met only with hardware acceleration. This master's thesis introduces the framework for rapid development of hardware accelerated network applications, able to be easily deployed in data centers. Moreover, the framework is very flexible since is it easily possible to change accelerated network functions on the fly. In other words, it is prepared for hardware acceleration of Network Functions Virtualization (NFV). NFV is a new

concept in data centers addressing unsustainable increasing of different proprietary hardware appliances. Instead, it standardizes network hardware to just three types – standard servers, storages, and routers. Everything is orchestrated by flexible software.

While common speeds in household are relatively low, in data centers and backbone networks multiple network lines with throughput from tens to hundreds of Gbps are common. Some data centers are already desperately waiting for 400 Gigabit Ethernet and beyond. Data centers provide a lot of important services and contain sensitive data. Therefore, they have to be highly protected to ensure their availability, integrity, and confidentiality under any circumstances.

Hardware acceleration is necessary to achieve throughput 400 Gbps and beyond. Today, two main hardware acceleration technologies are used – ASIC and FPGA. ASICs are custom chips providing excellent performance and power consumption but it completely lacks flexibility since its function is determined in a time of manufacture. Unfortunately, the lack of flexibility does not fit data center's needs – network environment is everything but fixed. On the other hand, FPGAs still provide much more performance and better power consumption than general purpose CPUs and they provide relatively good flexibility – they are unlike ASICs reprogrammable.

Purpose of this master's thesis is to analyze current situation in high-speed networks, design a framework capable of hardware acceleration of NFV with throughput up to 400 Gbps and beyond, implement such a design and verify its function.

The thesis is divided into 7 chapters. The fist chapter introduces the reader into the problem domain 1. The second gives theoretical background for the whole thesis 2. The third describes related work 3. The fourth analyses the current situation, explain why a new framework is needed and specifies its features in detail 4. The fifth proposes multiple clean sheet architectures of the whole framework 5. The sixth summarizes achieved results 6. And finally, the seventh chapter discusses the contribution of this master's thesis in the world of high-speed network applications 7.

Chapter 2

Theoretical Background

In this chapter theoretical background with respect to this master's thesis is given. Firstly, it is explained what the Internet actually is and how it can be described by a layered model 2.1. Secondly, Ethernet, the most widespread technology implementing Physical and Link Layer over a physical medium is discussed in detail. It's a key technology for implementing high-speed computer network. Therefore, it is primarily considered in this thesis and its features and behavior are exploited for an even more effective design of the framework 2.1.1. Thirdly, new emerging de facto standard of high-speed network packet transfers between NIC and the host computer and subsequent fast processing – DPDK is described 2.3. Fourthly, PCI Express description is provided. This technology is used for high-speed data transfers over short distances within PCs and servers. It uses layered protocol model 2.3.1.

2.1 ISO/OSI Model

The ISO/OSI model, sometimes referenced as a reference model of the Internet, is an international standard ISO number 7498-1:1994 [1]. It standardizes communication between various heterogeneous systems. In computer network's scope, these systems typically are PCs, routers, switches, etc. The main idea of the standard is to divide communication into multiple layers. Each layer has its purpose and responsibilities. Upper layers always work on a higher level of abstraction and vice versa. In general case, system operating on a layer `N` communicates with another system on the same layer without even knowing anything else about underlying layers. Computer network's devices typically work up to some specified layer `N`. A simple example of communication is given in following Figure 2.1 to understand a layered concept of communication more properly:

In the Figure 2.1, it can be seen communication between `PC 1` (TX only, operating up to the L7 layer) and `PC 2` (RX only, operating up to the L7 layer) via a `Router` (both RX and TX, operating up to the L3 layer). This example was not chosen by accident but on purpose to show simplified, but very typical communication over the Internet. Actually, as in the Figure 2.1, the ISO/OSI model consists of 7 layers. It is important to notice that some specific layer `N` communicates with a layer `N+1` (RX direction) and/or a layer `N-1` (TX direction), skipping the layers in communication is not possible. Further explanation of each layer is provided in top bottom order:

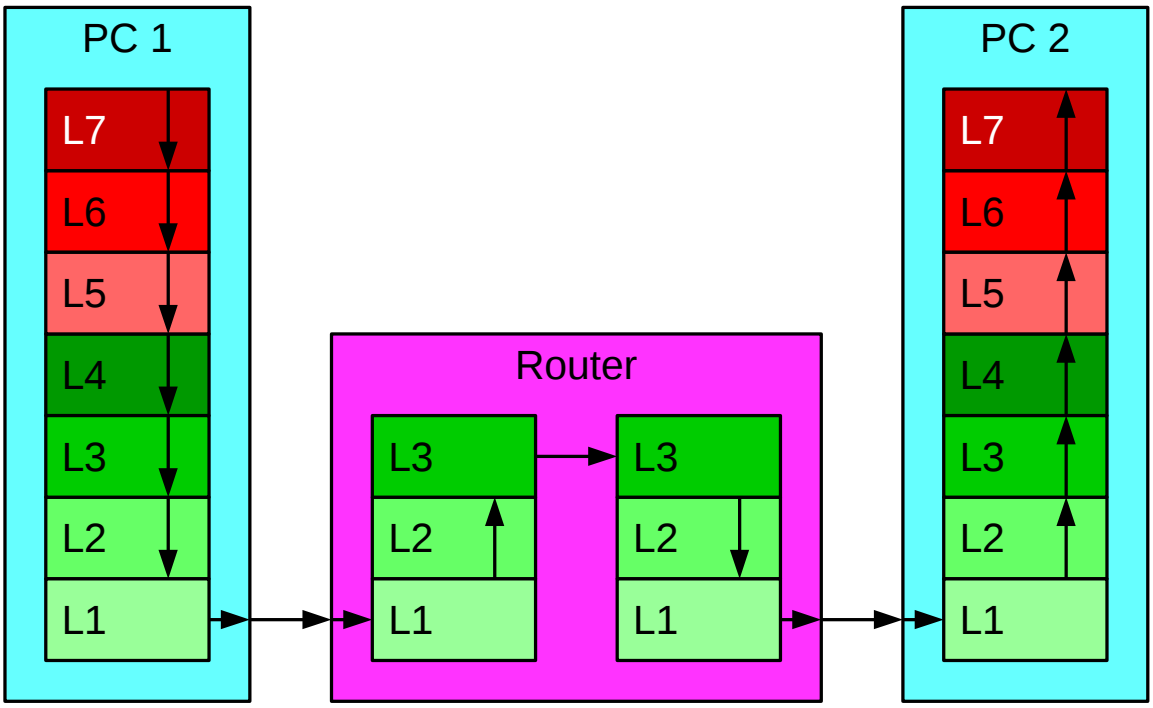

Figure 2.1: Example of communication according to the ISO/OSI model

Layer 7 – Application Layer is the uppermost layer of ISO/OSI model. Typical computer applications, communicating over a computer network, operate at this layer. Abstraction by underlying layers is guaranteed to easily design and implement network application. HTTP protocol is the most common example.

Layer 6 – Presentation Layer provides transparent data representation for Application Layer. It deals with various encodings and/or encryptions of data.

Layer 5 – Session Layer maintains dialogs (connections on a higher level of abstraction) for Application Layer. Dialogs are no longer data units like in underlying layers but logical data.

Layer 4 – Transport Layer provides demanded QoS transfers over an unreliable Network Layer, usually using connection (stateful) mode. Transport addresses distinguish between multiple connections.

Layer 3 – Network Layer is responsible for establishing, maintaining and terminating of connections between network devices. Network addresses are introduced for routing data units from a source to destination over a network with various underlying technologies.

Layer 2 – Data Link Layer provides functional and procedural means for a connectionless (stateless) mode. Data are transferred between network entities with Data Link Layer addresses. Error detection during transfer is mandatory.

Layer 1 – Physical Layer is the lowest layer of ISO/OSI model. It provides a physical connection between entities on a bit level. Physical medium typically used are e.g. copper, fiber cable or radio frequency.

As mentioned above ISO/OSI model is a reference model of the Internet. As usual, reference models are often too complicated so other models are in practice used instead. TCP/IP [15] model dominates today in practical implementations. Nevertheless, TCP/IP model can be easily mapped to ISO/OSI model so it is not considered in further text.

2.1.1 Ethernet

Ethernet is the most widespread and thus successful technology for data transmission over longer distances. In fact, on this technology is based on a backbone of today's Internet. It is defined by IEEE 802.3 standard [25].With respect to ISO/OSI model, it is an implementation of Physical and Data Link Layer. Ethernet is used to build both LAN and WAN networks – which is unusual considering LAN and WAN networks are in principle very different and Ethernet is still widely used in both of them. To ensure high standards of sustainable throughput and stability, dedicated lines used by Ethernet are preferred to radio frequencies used in WLAN. The secret behind the success of Ethernet is a lower price per 1 bit per second compared to its competitors. The price is always kept in mind while developing new standards.

As demand for a higher throughput over longer distances is growing fast, so is a throughput of Ethernet. Currently, 100 Gigabit Ethernet is the fastest standardized Ethernet. Despite standard of 400 Gigabit Ethernet has not been released yet it is about time to come up with an appropriate architecture of its implementation in order to stay ahead of competitors. The goal is to deploy this architecture as soon as the standard is released and proper technology is available. Before it happens, the architecture may be running inside adequate simulations.

Frame

Ethernet frame is a data unit on the Data Link layer of ISO/OSI model. They are processed by network devices called network switches. All fields of Ethernet frame are given and described in Figure 2.2 (for accuracy and better understanding fields of frame belonging to Physical layer are also mentioned):

Preamble (7 B) is the beginning of Ethernet frame. Each of its bytes contains binary value `10101010b` (MSB downto LSB). Bits are not chosen by accident – their purpose is to synchronize two asynchronous clocks between transmitter and receiver. Ethernet does not have any dedicated clock signal – clocks are asynchronous (since they have a different source). However, their frequencies are very close together (the maximum difference is specified by the standard), originating from two different very accurate crystals – asynchronous.

Start of frame delimiter (1 B) is the last synchronizing byte. Its value is `10101011b` (MSB downto LSB). Following bytes are already Ethernet frame (according to the newest IEEE 802.3 standard is referred as MAC frame in this context).

MAC destination address (6 B) is an address on the Data Link Layer of ISO/OSI model. It specifies target destination of Ethernet frame. Every network device on the Data

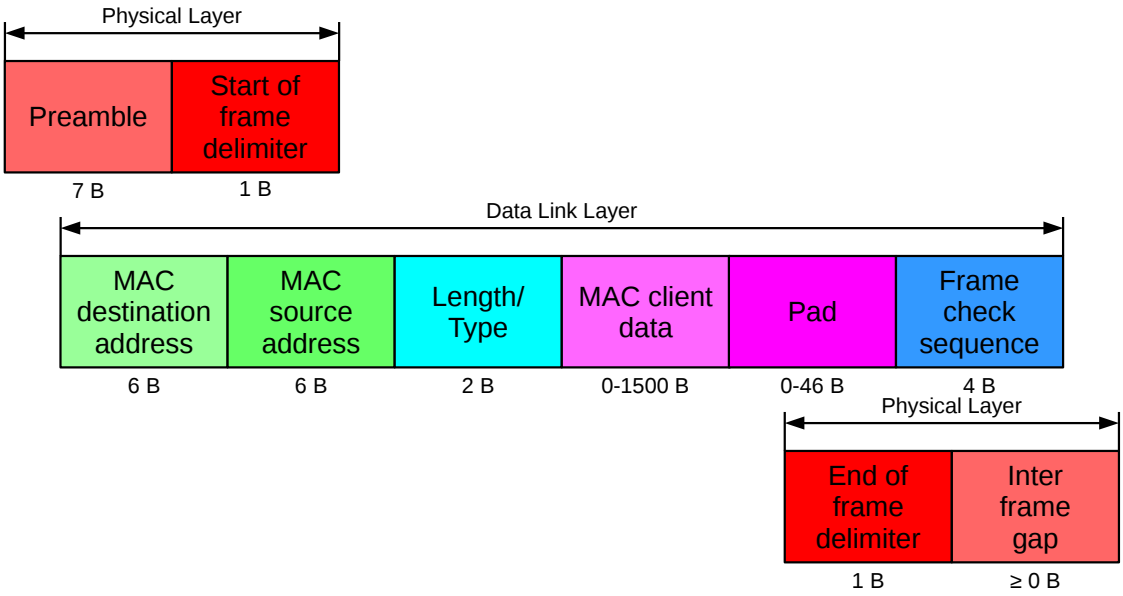

Figure 2.2: Description of Ethernet frame

Link Layer is supposed to have unique MAC address (which is no longer 100 % true). There are three types of destination addresses – unicast, multicast and broadcast.

MAC source address (6 B) is a primary key (in database meaning) of network transmitter device so a receiver is able to determine an origin of Ethernet frame.

Length/Type (2 B) has two meanings. The First meaning is the length of Ethernet frame in bytes (uses Big-Endian format) for values less or equal 1500 ($values \leq 1500$). Values greater or equal 1536 ($values \geq 1536$) mean this field specifies protocol in MAC client data field (e.g. it says the next field is VLAN).

MAC client data ($46\,B \leq data \leq 1500\,B$) is a payload of Ethernet frame. It contains data of the upper layer of ISO/OSI model. Standard says payload has to be greater or equal 46 B and less or equal 1500 B ($46\,B \leq payload \leq 1500\,B$). The minimum length is a relict of the past when Ethernet lines were not dedicated as in the most of the cases today, but shared. Collision detection algorithms had to be used (CSMA/CD) requiring minimum transmission's time of each Ethernet frame.

Pad ($0\,B \leq pad \leq 46\,B$) ensures the minimum length of the MAC client data. When a payload is too short, Pad is added accordingly.

Frame check sequence (4 B) ensures data integrity of received Ethernet frame. Checksum (CRC) is computed from fields MAC source address, MAC destination address, Length/Type, MAC client data, and Pad.

End of frame delimiter (1 B) marks the end of Ethernet frame. It is the first byte of Inter frame gap – its value is `11111101` (MSB downto LSB). While transmitting

a minimum Inter frame gas is 12 B on average (using DIC). While receiving the minimum Inter frame gap is just 1 B long which is slightly different from transmission because clocks of transmitter and receiver are asynchronous – it is allowed to add/remove bytes to/from Inter frame gap.

Inter frame gap (on average 11 B – the first byte is already End of frame delimiter) is a gap between Ethernet Frames. A minimum number of bytes is specified for receipt and transmission – the maximum is not limited. Each byte's value is `00000111` (MSB downto LSB).

Physical Layer

Physical Layer of Ethernet technology is nowadays already much closer to physics and analog rather than digital circuits as it is shown further. It is divided into other sublayers. The most important are PCS (where data is en/decoded) and PMA (connection to physical medium). Since the standard of 400 Gigabit Ethernet is expected to be released by the end of the year 2017 not everything is fixed yet. In general, it has been already agreed on upper layers of Ethernet standard but a lot of discussions are still being held about lower layers. Further text is based on the most recent draft of Ethernet standard [17].

Since 40 Gigabit and 100 Gigabit Ethernet is data transmitted over physical medium (typically fiber optics) in multiple lines using different wavelengths. Therefore, there internally are more than one possible realizations of e.g. 100 Gigabit Ethernet. The first generation was implemented as $10 \times 10\,Gigabit$ and the second as $4 \times 25\,Gigabit$. Why multiple lanes are actually used despite more complicated implementation? Why is simply not possible to make it as $1 \times 100\,Gigabit$? The answer is because of technology limitation (unfortunately so common for high-speed networks). Inside FPGAs or ASICs are used very wide data signals (hundreds or even thousands of bits), but before transmission over fiber optics data has to be serialized to just one bit (in a case of $1 \times 100\,Gigabit$ technology). Let us use a simple equation to calculate the target frequency 2.1:

$$f = \frac{speed}{data\_width} = \frac{100\,Gbps}{1\,b} = 100\,Gps = 100\,GHz \tag{2.1}$$

To sustain demanded throughput of 100 Gigabit over 1 line, 100 GHz transceivers will be necessary. For comparison, 25 GHz transceivers are today cutting-edge technology. Another simple equation for 400 Gigabit Ethernet's number of lines 2.2:

$$lines = \frac{speed}{f} = \frac{400\,Gbps}{25\,GHz} = 16\,lines \tag{2.2}$$

To realize 400 Gigabit Ethernet, 16 lines will be needed. This is exactly how the first generation is/will be implemented. Is there a trick to reduce the number of lines without increasing frequency of transceivers? There actually is and will probably be used in the second generation of 400 Gigabit Ethernet– PAM4 signaling [26]. The idea behind is simple – transfer two bits instead of just one in one clock cycle (4 levels are recognized instead of usual 2 in digital design). This technology brings somehow development back to analog circuits. For better understanding, the difference between NRZ (2 levels – 0 or 1) and PAM4 (4 levels – 00 or 01 or 10 or 11) signaling is illustrated in the Figure 2.3:

Transitions are in a red color, logic levels in a blue and values of the signal in a green. In digital circuits two logical values 0 or 1 are used. PAM4 encoding is a step forward towards

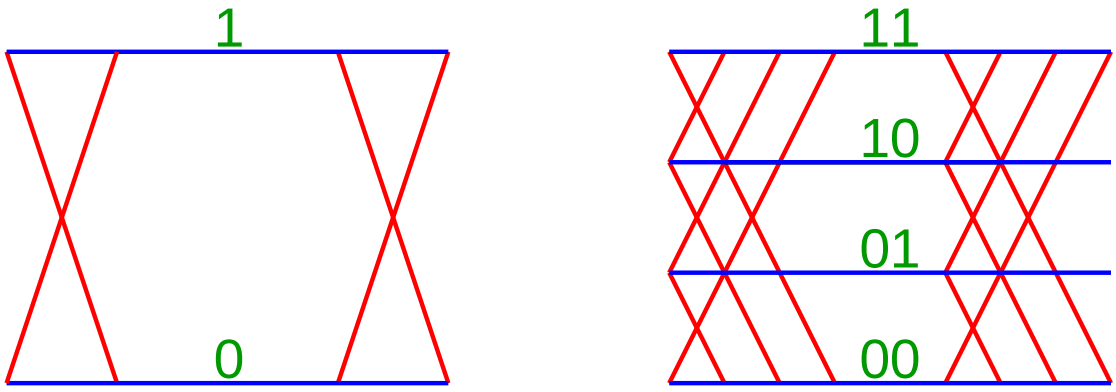

Figure 2.3: Comparison between NRZ and PAM4 signaling.

analog circuits (PAM8 encoding may be used in the future). This multi-level signaling has a clear advantage of effectively multiplying the throughput over a line using the same frequency. Disadvantage is that it decreases significantly signal-to-noise ratio and therefore it increases BER on lines. These are very real problems which are strongly addressed in Ethernet standard and are not completely solved yet. For 100 Gigabit Ethernet is use of Forward Error Correction (FEC) optional but for 400 Gigabit Ethernet it is mandatory (for transmission over 10 km it is a must, at much shorter distances may not be necessary). It is believed that the PAM4 signaling will be used in the second generation of 400 Gigabit Ethernet. Now the previous equation 2.2 may be updated 2.3:

$$lines = \frac{speed}{f \times bits\_per\_clock} = \frac{400\,Gbps}{25\,GHz \times 2} = 8\,lines \tag{2.3}$$

Using as least lines as possible increases throughput per price criteria which is vital for the success of Ethernet technology. In not far future, transceivers with frequency over 50 GHz are expected to make it possible to implement 400 Gigabit Ethernet technology in just 4 lines or to give birth to 1 Terabit Ethernet. These high-frequency multi-level signals are inputs for FPGAs or ASICs and at the beginning of processing pipeline they are deserialized to be clocked at much lower frequencies.

That was a short excursion to the lowest layers of Ethernet. Nevertheless, for this master's thesis are much more important upper layers. Among them is 400 GMII protocol – it is an interface and protocol between Physical and Data Link Layer of Ethernet technology. A throughput of this MII is described by an equation 2.4:

$$throughput = data\_width \times f \tag{2.4}$$

As in any other interface – if frequency or data_width is increased throughput gets higher. The challenge is to set up a right balance between these parameters for target technology. Signals of 400 GMII (basically any MII) interface are given in the following Table 2.1 (both RX and TX direction) and their meaning (protocol) is further described in more detail:

TX_CLK and RX_CLK are synchronization signals. Frequency of these signals determines throughput.

Name	Width [b]	Direction
TX_CLK	1	in
RX_CLK	1	in
TXD	DATA_WIDTH	out
TXC	DATA_WIDTH / 8	out
RXD	DATA_WIDTH	in
RXC	DATA_WIDTH / 8	in

Table 2.1: Definition of MII interface

TXD and RXD are data signals. Format of data exactly corresponds to Ethernet frames as defined in 2.1.1. Preamble and consequently beginning of Ethernet frame itself is always aligned to 8 B. End of Ethernet frame is unaligned (aligned to 1 B).

TXC and RXC are control signals of MII interface. Bits in TXC and RXC determine meaning of the corresponding bytes in TXD and RXD signals. If a bit is set then corresponding byte is a control character with special meaning. Control characters are explained in another Table 2.2.

TXC/RXC bit	TXD/RXD byte	Meaning
0	0x00h – 0xFFh	Data
1	0x07h	Idle (Inter frame gap)
1	0xFBh	Start (Start of frame delimiter)
1	0xFDh	Terminate (End of frame delimiter)
1	0xFEh	Error

Table 2.2: Control symbols of MII interface

2.2 Hardware Acceleration in Computer Networks

Since network traffic is growing very fast, hardware acceleration has become a must. There has been a long history of hardware acceleration in video processing – GPUs. History of hardware acceleration in computer networks is shorter but not less important. There has been an increasing pressure on network throughput. The demand could not be at some point met by general purpose CPUs so hardware acceleration had to be used instead. This claim is supported by a graph showing available time for processing the shortest 64 B (without overhead) network packet in the Figure 2.4. Fortunately, hardware acceleration in computer networks has been intensively studied over decades and lots of scientific articles have been written. E.g. already since 2003 a research group at Faculty of Information Technology, Brno University of Technology is devoted to hardware acceleration of computer networks [11]. Therefore, this master's thesis does not have to start from scratch but can benefit from previous research.

2.2.1 Set of Functional Primitives

The problem of hardware acceleration may be divided into subproblems. It is important to correctly identify functional primitives to be hardware accelerated in computer networks.

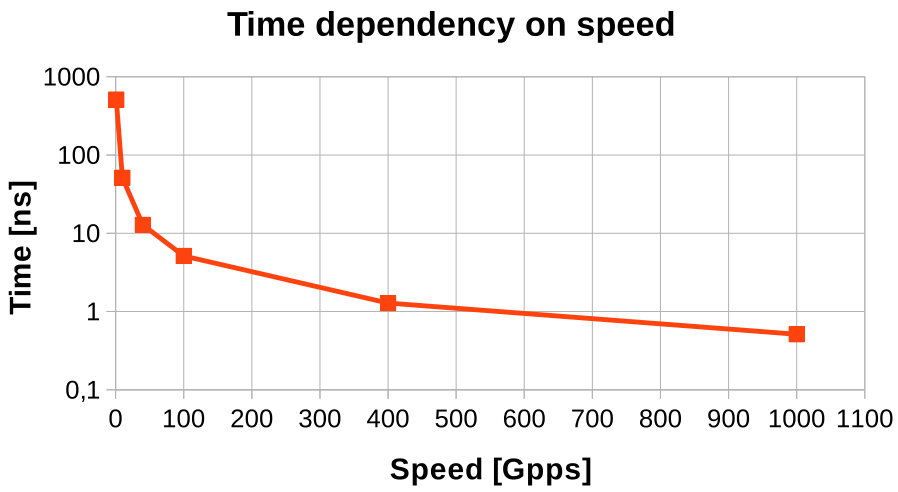

Figure 2.4: Available time for processing the shortest 64 B network packet at different speeds

Typically, these functional primitives are time critical, executed for each network packet and used as building stones for more complex network functions. According to [13] following functional primitives are well suited for hardware acceleration.

Packet Header Analysis

Upon arrival, each network device has to extract and analyze all packet's header – e.g. to check CRC on L2 layer. Considering packet rate nearly 600 Mpps for 400 Gb networks, this task is very demanding. Moreover, it has to be performed in a real-time manner due to the limited size of network buffers. Lack of performance at any time may lead to packet loss and thus negatively affect QoS. Depending on a network device, headers up to L7 layer have to be considered.

Packet Classification

Packet classification is a very important network function and it is essential as a building stone for other more complex network functions such as firewalls, IDS or IPS. Upon arrival of each network packet, it has to be decided what action to take with respect to value, range or prefix of specified header fields. Therefore, each packet has to be classified and a corresponding rule has to be found. Each network packet is identified by a 5-tuple – network flow:

$$Network\_flow = (SRC\_IP, DST\_IP, SRC\_PORT, DST\_PORT, PROTOCOL) \tag{2.5}$$

SRC_IP and DST_IP (32 b for IP or 128 b for IPv6 each) is unique source and destination identification numbers at Network Layer of the ISO/OSI model.

SRC_PORT and DST_PORT (16 b each) are source and destination indentification numbers at Transport Layer of the ISO/OSI model.

PROTOCOL (8 b) is a number at Network Layer of the ISO/OSI model specifying a protocol of upper layer.

Let us calculate how many different network flows there are for IP protocol:

$$IP\_network\_flows = 2^{32+32+16+16+8} = 2^{104} \tag{2.6}$$

And for IPv6 protocol:

$$IPv6\_network\_flows = 2^{128+128+16+16+8} = 2^{296} \tag{2.7}$$

In the other words packet classification is a lookup in N-dimensional space. Unfortunately, the state space of network flows is too enormous to specify exact match rule for each flow (sometimes used for the heaviest network flows). Range or prefix rules are usually used instead. For more information please refer to external sources such as [18] since this master's thesis is not about packet classification.

Longest Prefix Match (LPM)

LPM is an algorithm used for lookup in one-dimensional space. It selects the most specific entry (prefix) matching given value. This algorithm may be used for packet classification 2.2.1 especially when lookup in just one dimension is sufficient – e.g. routers lookup typically by destination IP addresses.

Pattern Matching

Pattern matching is used for advanced IDS and IPS for detection malicious network traffic described by strings or regular expressions. Lookup is conducted in packet's payloads. This task is extremely performance demanding thus a lot of research has been done to propose dedicated hardware accelerated solution – general purpose CPUs are insufficient for high-speed networks.

Flow Cache Management

Flow cache management keeps a context of network flows. The task is to collect metadata such as source and destination IP addresses and ports, protocol, an amount of transmitted bytes, etc. This metadata is used for network monitoring, network troubleshooting, improving QoS, for law enforcement institution (duty of ISPs), etc. Due to an excessive amount of flows in backbone networks the task is challenging.

2.2.2 P4 Language

P4 language is a new promising approach for network packet processing. It is trying to address volatile computer network environment by allowing to easily specify new network protocols in a high-level language. It is based on a Match+Action forwarding model. High level description in P4 language of network packet processing is subsequently compiled into target technology – the language is technology independent. For more detailed information please refer to [14]. In the Figure 2.5 is given a block diagram of P4 pipeline.

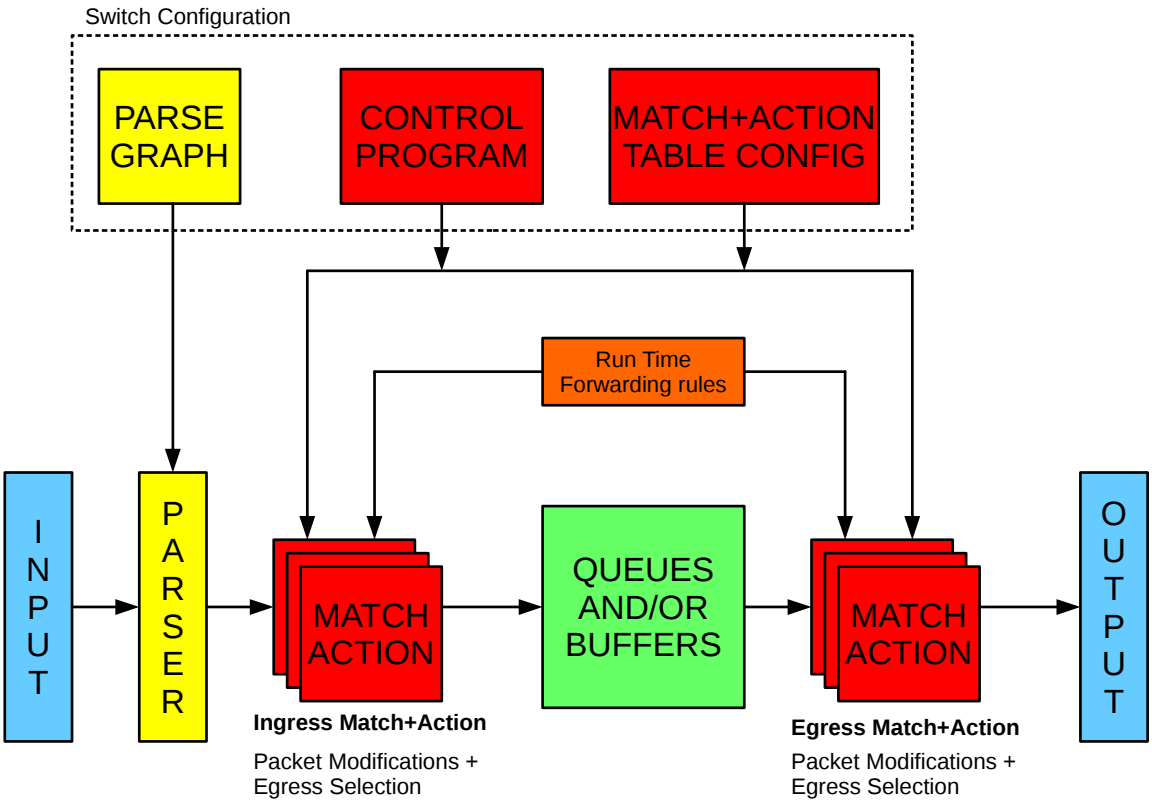

Figure 2.5: P4 pipeline block diagram

2.2.3 Network Functions Virtualization (NFV)

Network Functions Virtualization (NFV) is trying to address unsustainable increase of proprietary hardware appliances in data centers. Instead of using many kinds of hardware boxes for a specialized purpose, NFV reduces them to just standard servers, storages and switches [7]. These three standard boxes may provide all kinds of network functions in virtualized manner such as (they use functional primitives described here 2.2.1):

- Packet filter
- Load balancer
- Packet routing and switching
- Deep Packet Inspection DPI

Packet filter is becoming more and more important network function due to increasing harmful network traffic such as DDoS attacks. It is a good practice to filter network traffic at entry points of data centers in order to protect them. Packet filtering is based on packet classification. Corresponding rules may be static or dynamic depending on the current network traffic and possibly reputation databases. For high-speed networks, this function has to be hardware accelerated because pure software solution lacks performance.

Load balancer may be based e.g. on packet classification, hash of network flows or round-robin algorithms. Each of these algorithms has advantages and disadvantages – fortunately, they can be combined to achieve the best results depending on the target use case. Load balancers are particularly useful in data centers consisting of thousands of servers, especially at the entry points, and also within the servers itself. As already mentioned, a typical server consists of multiple CPUs, each of them consists of multiple cores, each of them is capable of running multiple threads 1. To avoid performance demanding inter-process communication, network traffic can be distributed upon arrival without CPU's contribution to the right software buffers by NICs.

Packet routing and switching is redirecting of network traffic based on Data Link Layer's addresses (switching) and Network Layer's addresses (routing). Many other variables may be considered such as NAT.

Deep Packet Inspection (DPI) is unlike so far mentioned network functions based on analysis of both network packet's headers and payload instead of just headers. It can be realized as a regular expression pattern matching. It is very performance demanding.

According to the authors of NFV, using this technology would bring many advantages:

- reduction of CAPEX since there will not be a need to buy specialized hardware anymore
- reduction of OPEX due to decreased space, power and cooling requirements and simplification of management of network devices
- reduced time to market due to much faster rolling out of new network services
- added flexibility for customers allowing them to pay according to their current needs

The original idea about NFV was to utilize just three standard devices and everything else would be orchestrated by a flexible software layer. NFV technology is actually an evolution of virtual machines (VMs) and server virtualization but this time as a Network Functions Virtualization (NFV). The technology of VMs and server virtualization comes with some performance overhead but over the years it was minimized to just a few percents and it is today widely and successfully used. Unfortunately, NFV via a pure software solution lacks performance and power consumption efficiency – both nowadays so desperately needed in the world of high-speed computer networks. Hardware acceleration of NFV may be a solution to address these problems. Since flexibility is one of the main columns of NFV, FPGA technology may be a right candidate for such hardware acceleration.

2.3 Data Plane Development Kit (DPDK)

DPDK is becoming a de facto standard for high-speed network packet transfers between NIC and the host computer and its subsequent fast processing [10]. Behind DPDK are big companies such as 6WIND, Calsoft Labs, Intel, Tieto and others and so far it seems as a very successful project. DPDK is a set of drivers and libraries for fast packet transfers and processing avoiding Linux network stack in kernel space and sticking to packet zero-copy philosophy to gain as much performance as possible. Since DPDK is a de facto standard for fast packet transfers and processing there are many implementations of various network

functions such as packet classification, packet filter, packet distributor, Open Virtual Switch (OVS) [8] and many others [6]. Support of DPDK seems to be vital for any high-speed NIC.

As it has been already mentioned, DPDK is a set of drivers and libraries. Let us have a closer look at some core ones:

Environment Abstraction Layer (EAL) provides abstraction from low-level resources to DPDK libraries and applications. EAL is e.g. responsible for DPDK loading and launching, core affinity, system memory reservation, PCI Express address abstraction, trace and debug functions, utility functions, CPU feature identification, interrupt handling, alarm functions and others [6].

Ring Library is an implementation of ring buffer Abstract Data Type (ADT). Unlike an implementation as a linked list, this ring buffer is implemented as a static array of pointers (pointing to mbuf structures, possibly linked) with lockless multiple enqueue and dequeue functions. The static implementation provides more performance but it is less flexible.

Mempool Library manages memory allocations and frees for DPDK. It includes special services to ensure that cache of CPU, core affinity and memory modules are used effectively.

Mbuf Library is an implementation of mbuf ADT. It is responsible for storing network packets of variable size. This is achieved by using fixed size mbuf structures as a linked list. Actual size of the mbuf structure is set up during a launch of DPDK application. The mbuf structure is very important from NIC's perspective because NIC directly reads/writes from/into this structure via PCI Express. Descriptors of these mbufs (memory addresses and lengths) have to be provided to NIC. The mbuf structures are allocated and freed via Mempool Library and their pointers are stored in a ring buffer structure implemented by Ring Library. An example of linked mbuf structures is illustrated in the Figure 2.6.

Poll Mode Driver is managing RX/TX memory descriptors from/to NIC via pooling instead of interrupts – for high-speed NICs would interrupts consume a tremendous amount of performance just for interrupt's overhead itself, pooling technique is much more appropriate here.

2.3.1 PCI Express

PCI Express is a widespread and successful technology used as a system expansion bus for computer systems. It is defined by PCI-SIG, nowadays already in its third generation [2]. The most important component of computers is CPU (today better to say SoC). This CPU is soldered to a motherboard containing other vital components such as memory, power supply, etc. They usually also contain expansion slots for various peripherals (in PC and server segment). These peripherals have to be somehow physically and logically connected to CPU. When high-throughput dedicated connection is required PCI Express comes into play (so far it is the highest throughput technology in this context). System buses went through a long evolution and PCI Express generation 4 is so far the latest step – it has not been standardized yet. Let us pinpoint its features:

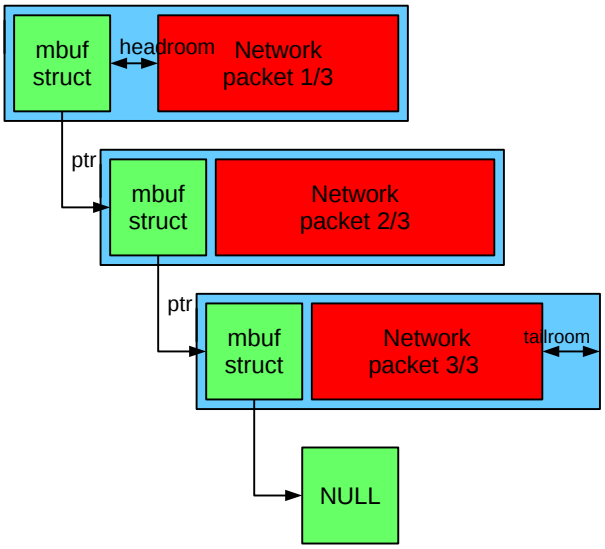

Figure 2.6: Example of linked mbuf structures

- full-duplex operation
- high-frequency, serial, low-voltage transceivers using differential pairs
- copper as a physical medium (it may change in the future)
- transmission over short distances (tens of centimeters)
- hierarchical topology (using switches)
- dedicated lines (Original PCI was a bus, PCI Express is actually not but still the word „bus“ is used from a higher perspective)
- raw throughput of 8 Gbps per line
- multiple lines, (usually) up to 16
- Physical, Data Link and Transaction layer model (similar to ISO/OSI)
- transmission in data units (packets)
- reliable delivery (ensured by protocol)

There are many similarities between PCI Express and Ethernet technology (mentioned above 2.1.1) on higher abstraction level – e.g. both use a layered model or transmit data units. Let us calculate total throughput of PCI Express generation 4 for (usual) maximum of 16 lines 2.8:

$$throughput = throughput\_per\_line \times lines = 16\,Gbps \times 16 = 256\,Gbps \tag{2.8}$$

The calculated raw throughput is just 256 Gbps but the desired throughput would exceed 400 Gbps (speed of 400 Gigabit Ethernet). To overcome this significant obstacle, PCI Express would either has to speed up or another workaround has to be found. Let us have a closer look at PCI Express layers:

Transaction Layer is the uppermost layer of PCI Express protocol. At this level of abstraction, communication is done via transactions, may consist of multiple Transaction Layer Packets TLPs. Transactions are mapped over different address spaces (the host computer main memory, I/O, configuration, and message). There are read, write and event type of transactions. Request packets (such as read) share the same ID with corresponding response packets – being a single transaction.

Data Link Layer is responsible for reliable delivery over an unreliable channel. This is done with methods such as data error detection, data error correction and retransmitting (similarities with Transport layer of ISO/OSI model). This layer also produces and consumes packets that are used for Link management functions (such as TLP acknowledgment). These packets are called Data Link Layer Packet (DLLP).

Physical Layer is responsible for transmitting and receiving DLLP packets. That means it contains drivers, input buffers, serializers, deserializers and all other circuits. 128b/130b encoding is used in generation 4 of PCI Express to balance DC level on lines. Most of the technological improvements of PCI Express will be reflected in this layer (such as replacing physical copper medium with fiber optics).

Layers of PCI Express were so far described in more detail. Let us continue with description of TLP packet of PCI Express generation 3 (differs from generation 2) in the Figure 2.7 now (additional source of information [19]):

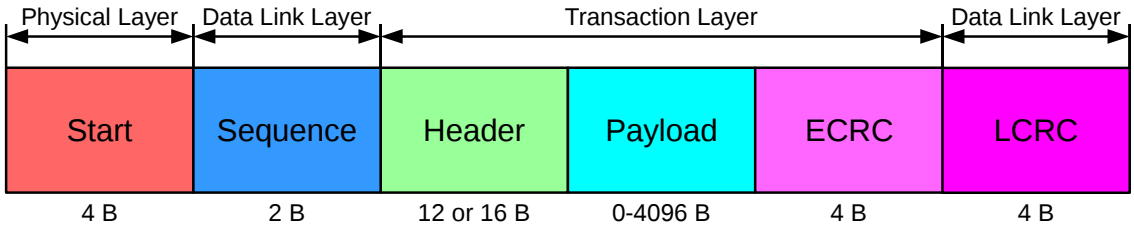

Figure 2.7: PCI Express packet (TLP) fields

Start (4 B) is a starting sequence at Physical Layer of PCI Express. It is used for synchronization between transmitter and receiver.

Sequence (2 B) is a number uniquely identifying DLLP packets on Data Link Layer.

Header (12 or 16 B) specifies a type of TLP packet at Transaction Layer. There are many types of packets such as read/write from memory or I/O and various messages used as interrupts. An example of memory request TLP packet is given in the Figure 2.8. If it is a read request, completer process this packet and return data from memory as a payload.

Payload (0–4096 B) is data itself. For memory read requests this field is not used. Lengths up to 4096 B are defined in standard but in real case scenarios the length is often limited to e.g. 256 B. Obviously, with increasing payload length the effectivity of PCI Express gets higher due to a lower overhead of the protocol.

ECRC (4 B) is a checksum at the Transaction Layer to ensure data integrity and it is optional – due to a redundancy with LCRC it is often not used.

LCRC (4 B) is a checksum at the Data Link Layer to ensure data integrity. Unlike ECRC this field is mandatory.

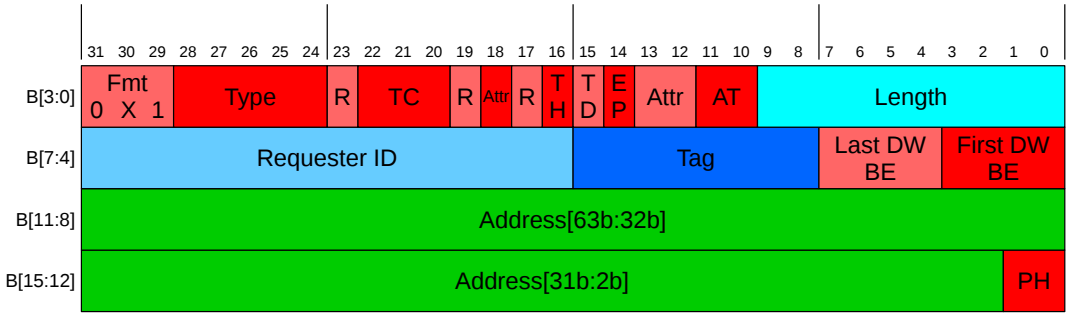

Figure 2.8: PCI Express packet's (TLP's) header for memory request transaction

For more details please refer directly to the PCI Express standard [2].

Chapter 3

Related Work

Theoretical background was summed up in the previous chapter 2 – in this is provided related work about hardware acceleration of high-speed networks. It has to be said right at the beginning that there is a very little related work regarded to 400 Gb network processing. Two sources have been found after long searching on the Internet [16] by Xilinx and [24] by Altera (today part of Intel). They address problems of lower Ethernet layers (such as CRC computation) and explain why only FPGA is the most suitable for these problems. Nevertheless, they do not address high-level processing of network traffic.

Since there is no relevant related work about 400 Gb network processing (at least not publicly accessible) let us look closer at 100 Gb network processing. The most relevant sources for this thesis seems to be either a NetFPGA project [12] or a Liberouter project [11]. Both use FPGA technology for high-speed network processing. When it comes to 100 Gb network processing the Liberouter project is more relevant. The truth is that the NetFPGA project never actually mastered 100 Gigabit Ethernet and their latest framework is NetFPGA SUME [27] which theoretically has capability up to 100 Gb network processing but in fact, it has just four 10 Gigabit Ethernet ports. On the other hand, the Liberouter project offers mature 100 Gigabit Ethernet FPGA solution both in hardware [9] (COMBO cards family 3.1) and firmware and software [23] (the NetCOPE framework 3.2). Therefore, the Liberouter project will be considered in the following text and used as a reference platform in this master's thesis. Nevertheless, it is important to keep in mind that 400 Gb and 100 Gb network processing may be very different although basic principles probably remain.

3.1 COMBO Card Family

COMBO card family is a set of hardware acceleration FPGA cards for high-speed network processing such as 100 Gigabit Ethernet using PCI Express expansion slot for fast communication between the acceleration card and the host computer [9] developed by the Liberouter project [11]. A typical COMBO card consists of one or multiple cages for optical transceivers used to implement Physical Layer of Ethernet technology, a large FPGA chip where the most of the processing is done, PCI Express interface for high-speed transfers between the acceleration card and the host computer, external memories for large tables needed by stateful network processing and/or for packet buffers and other support components such as hardware for precise timestamps generation [21]. High-level block diagram

of the most recent COMBO-100G card with two 100 Gigabit Ethernet ports follows in the Figure 3.1:

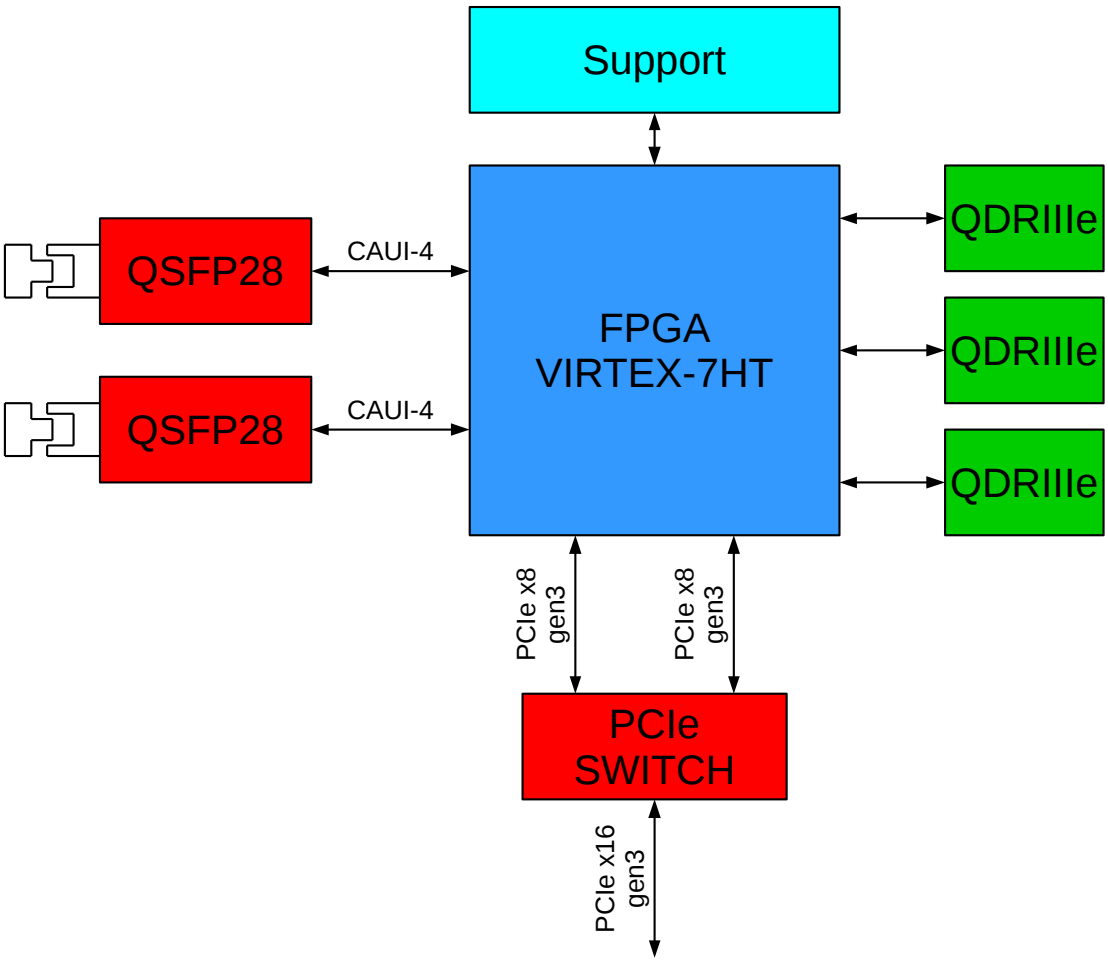

Figure 3.1: High-level diagram of COMBO-100G card

QSFP28 is a standardized size of an optical transceiver. This transceiver works between electrical and optical domain. CAUI-4 interface is used for connection to FPGA – these are four differential, low-voltage, high-frequency, electrical pairs signals each operating at frequency 25 GHz.

FPGA VIRTEX-7HT is a core of the COMBO card. FPGA is a regular structure of CLB blocks over a large area. These CLB blocks consist of LUTs, registers, carry chains and multiplexers. Synthesis tools map logical functions described in HDL languages into them. Moreover, FPGAs also contain small ASICs such as internal memory, DSPs, transceivers and other useful fixed functions adding much more features and performance to FPGAs.

PCIe SWITCH provides PCI Express generation 3 with 16 lines capability for the COMBO card. The reason for usage of this component is behind absence of appropriate integrated blocks in FPGA VIRTEX-7HT. It contains multiple integrated blocks for PCI

Express generation 3 with just 8 lines. PCIe SWITCH joins these 2 integrated blocks to just 1 with desired capability. Actually, the COMBO card does not contain PCIe SWITCH as an external component – its function is provided by the host computer's motherboard and CPU by bifurcation technology [3].

QDRIIIe are QDR static external memories, 72 Mb each. Operating frequencies are up to 675 MHz with throughput up to 48.6 Gbps, in total 145.8 Gbps.

Support is a set of additional modules and functions such as power supply, GPS input connector for GPS signal, JTAG connector for debugging and others.

COMBO-100G is fully prepared for a rapid development of hardware accelerated network applications for 100 Gigabit Ethernet. Nevertheless, it may also be used as a testing platform for this master's thesis because as stated in abstract the emphasis is put on to propose a generic framework for hardware acceleration of high-speed networks from 100 Gb to over 1 Tb. Merging both Ethernet ports together into one single MII stream 2.1.1 gives even 200 Gb.

3.2 The NetCOPE Framework

The NetCOPE framework is a framework for hardware accelerated high-speed network applications [20]. Its reference hardware platform is a family of FPGA COMBO cards. The newest generation of the framework is the NetCOPE-100G framework for the COMBO-100G card, capable of processing of two 100 Gigabit Ethernet ports. The NetCOPE framework and family of the COMBO cards is a technological leader in FPGA-based solutions for high-speed network processing. The closest competitor is a NetFPGA framework being developed by famous institutions such as Stanford University, University of Cambridge, Xilinx, and others [12]. The NetFPGA framework is already deprecated today. The newest generation provides just four 10 Gigabit or one 40 Gigabit Ethernet ports (without obscure adapter) [27]. The NetCOPE framework is a firmware layer over FPGA and a set of drivers, libraries and applications for the host computer providing high-level abstraction.

3.2.1 Firmware Layer

Firmware part of the NetCOPE framework is an abstraction layer of hardware including high-level functions such as DMA transfers over PCI Express between acceleration card and the host computer, implementation of Physical and Data Link Layer of Ethernet technology, memory controllers and others. Developer can focus on its network application and not to waste time with low-level functions. Block diagram of the NetCOPE-100G framework and description of its components in the Figure 3.2:

Application is a high-level firmware layer implementing hardware accelerated network application. Developer can rely on underneath layers providing both basic and more advanced functionality and focus on the application itself. The application may be described by low-level languages such as VHDL or Verilog, or by previously mentioned high-level language for packet processing P4 2.2.2. Example of such applications are NIC, Packet capture and reply, NIC [23] (load balancer with filter), pattern matching (regular expression pattern matching) and others. An application consists both of firmware and software layer – now firmware part is being described.

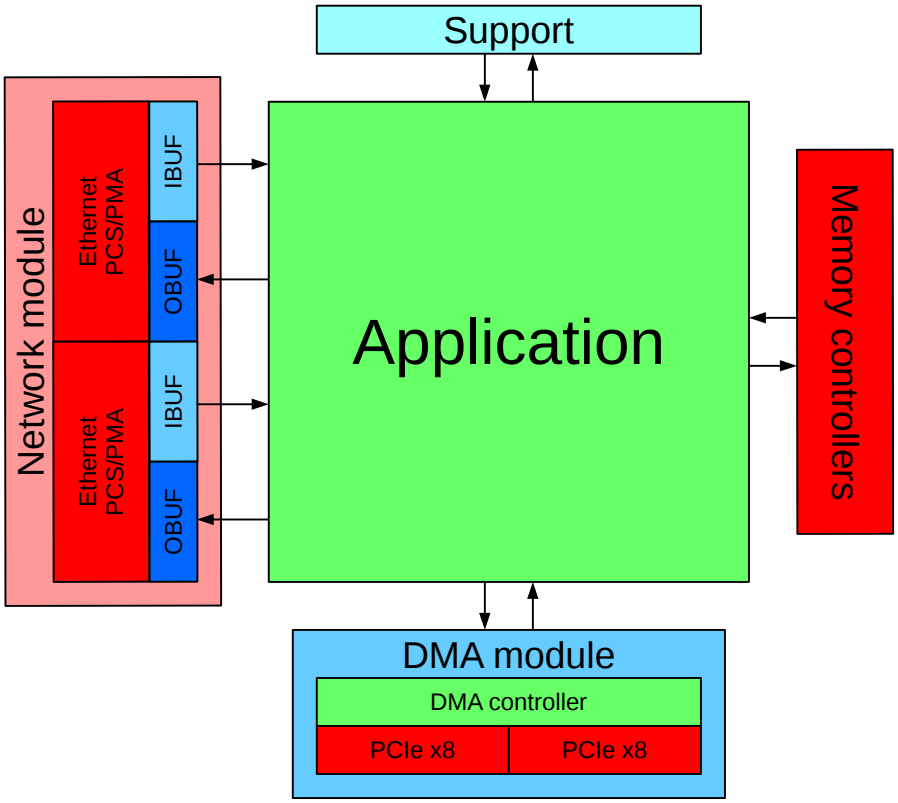

Figure 3.2: Block diagram of the NetCOPE-100G framework

Network module is an implementation of Ethernet Physical (Ethernet PCS/PMA) and Data Link (IBUF and OBUF) layers. The main purpose is to provide abstraction of Ethernet protocol by Ethernet protocol encoding/decoding, integrity checking, statistics of incoming and outcoming network traffic computation and buffering. Towards application, network packets are transferred via point-to-point protocol with source and destination ready signal, towards Ethernet transceivers via stream protocol.

DMA module implements high-speed bi-directional transfers between acceleration card and the host computer. The host computer has an additional relatively cheap performance for further packet processing. To effectively utilize multiple CPU's cores, DMA module contains multiple completely independent RX and TX queues. Queue selection is made by an application – either statically or dynamically. Support of DPDK protocol 2.3 was added to the NetCOPE framework 2.3 recently. However, its original protocol was/is different. It is similar to the Ring Buffer structure mentioned in DPDK's section 2.3. Network packets are directly stored in a fixed size ring buffer (instead of pointers in DPDK). Each network packet's header contains a pointer to the next one. This approach has both advantages and disadvantages and its name is SZE. The ring buffer data structure is illustrated in the Figure 3.3.

Memory controllers provides firmware abstraction layer over external memory chips such as QDRIIIe static and SDRAM dynamic (missing in latest COMBO-100G card) memory. In the future, HMC and HBM memories may be also supported.

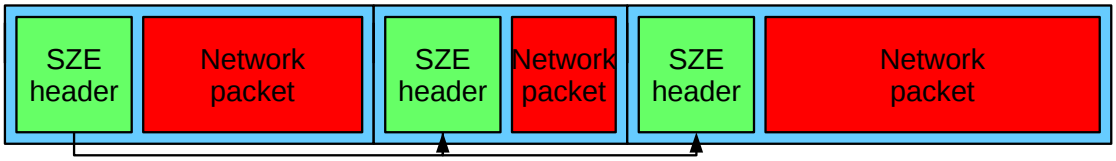

Figure 3.3: SZE network packet's storage format in the host computer memory

Support contains additional components and functionality such as precise timestamps generator, ID component, MI32 protocol implementation for configuring state and control registers, etc.

3.2.2 Software Layer

The software layer is an integral part of the NetCOPE framework. Software layer consists of three sublayers – drivers, libraries, and applications. Drivers provide basic software functionality for COMBO cards family such as memory descriptors management, ring buffer's pointers update, etc. Libraries implement further reusable functionality used by applications such as `sze2read` for reception of network traffic, `sze2write` for transmission and others. Standard Linux is used running on x86/ADM64 CPUs.

Chapter 4

System Analysis

Theoretical background and related work has been described in previous chapters. The knowledge is used for detailed system analysis of framework for hardware acceleration of 400 Gb networks. Real world's requirements and constraints are always kept in mind.

The main goal of this chapter is to justify the necessity of the new clean sheet framework. Thorough analysis and subsequent criticism of the current state is given. As mentioned in previous chapter 3, COMBO-100G card 3.1 and the NetCOPE framework 3.2 are the reference platform. Few experiments with respect to high-speed transfers between the acceleration card and the host computer are performed.

Furthermore, system requirements for 400 Gb network processing and beyond are discussed. It is explained why current framework NetCOPE-100G 3.2 is inappropriate. Real world use case scenarios are taken into account, subsequently functional and performance requirements are enumerated, these are further studied and their impact on a system architecture is considered. This is an important milestone for the next chapter of the system design 5.

4.1 Analysis Current State

This master's thesis introduces a clean sheet new architecture. Therefore, it is right at the beginning essential to answer one question: Is it really necessary to come up with a new architecture?

The current solution, the NetCOPE-100G framework, will be soon deprecated. The platform was designed in 2008 and it does not reflect current needs of network applications. Requirements and processing techniques were slightly different. Multicore CPUs contained just a few cores, there was no need for a large number of completely independent DMA channels. Multiple clock cycles were available for even the shortest network packets (see Figure 2.4). Each time Ethernet increased its speed, limits of the framework were pushed further but the principle remained.

For 400 Gigabit Ethernet, the framework has to be significantly changed. First, multiple network packets has to be processed in one clock cycle to achieve wire-speed throughput. CPUs contain today many cores – much more completely independent DMA channels are needed. Moreover, the speed of network links is growing faster than the síze of FPGA chips. Thus, the utilization of hardware resources has to be much more efficient. Let us sum up limitations of the current framework in order to eliminate them in the proposed framework.

- Missing support to process multiple packets per clock cycle

- Linear scalability with respect to number of DMA channels
- Resource demanding processing pipeline

4.1.1 Missing Support to Process Multiple Packets per Clock Cycle

The performance of the NetCOPE framework is determined by a width of the main data bus. 8 b data bus at frequency 125 MHz is used for 1 Gigabit Ethernet, 64 b@156 MHz for 10 Gigabit Ethernet and 512 b@195 MHz for 100 Gigabit Ethernet. Throughput is described by the equation 2.4. Due to technological reasons, it is not possible to increase frequency as fast as the speed of Ethernet and because of Ethernet protocol 2.1.1 is very difficult to increase a width of the main data bus over 512 b (explained in further text).

The NetCOPE-100G framework internally uses FLU protocol with 512 b wide data bus (see Table 4.1). FLU protocol is similar to MII stream 2.1.1. Moreover, it contains additional information such address of start and end of frame delimiter within the data word (see 2.1.1). FLU protocol does not support to transfer multiple network packets per clock cycle within the data word (to be accurate, multiple start of frame delimiters). Therefore, increasing data width of FLU interface over 512 b leads to limited throughput for short network packets. Unfortunately, short network packets are common in computer networks.

Name	Width [b]	Meaning
DATA	512	Data bus
SOP_POS	3	Start of packet within data word
EOP_POS	6	End of packet within data word
SOP	1	Start of packet valid bit
EOP	1	End of packet valid bit
SRC_RDY	1	Source ready signal for flow control
DST_RDY	1	Destination ready signal for flow control

Table 4.1: Definition of FLU interface

Increasing data width further means to transfer multiple start of frame delimiters within one data word. Let us compare MII protocol with data width 512 b (Figure 4.1) and 2048 b (Figure 4.2) with respect to a number of network packets (start of frame delimiters) per clock cycle. Using 2048 b data width instead of 512 b means to transfer multiple start of frame delimiters within one data word. From the comparison is clear that a way of processing multiple network packets per clock cycle has to be found in order to catch up with the future speeds of Ethernet. Throughput of the NetCOPE framework has been perceived as data width times frequency so far (see equation 4.2). The equation cannot be used for data width over 512 b because support to process multiple packets per clock cycle has to be also considered (see equation 4.1). Both equations have to be taken into account in order to calculate throughput.

$$throughput = \frac{network\_packets}{clock\_cycle} \tag{4.1}$$

$$throughput = data\_width \times f \tag{4.2}$$

Using fixed data width in the NetCOPE-100G framework means a very little flexibility. The proposed solution should use a number of packets per clock cycle as a parameter making

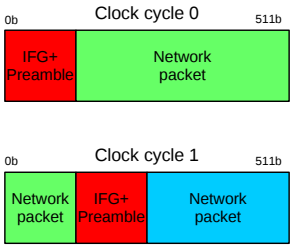

Figure 4.1: Example of network packets on 512b wide data bus

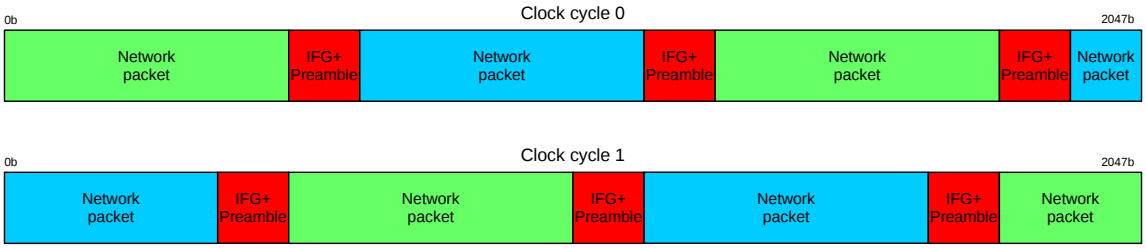

Figure 4.2: Example of network packets on 2048b wide data bus

it very flexible for greater or equal 100 Gb network solutions (more accurately, for internal data widths greater of equal 512 b with step 512 b).

4.1.2 Linear Scalability with Respect to Number of DMA Channels

Another significant problem of the NetCOPE-100G framework is just linear scalability with respect to a number of completely independent DMA channels. This problem is becoming more and more significant with each new generation CPUs because the trend is to increase a number of cores. Both Intel and AMD mention today in their roadmaps around 32 cores per CPU, CPUs based on ARM technology may have even much more. The NetCOPE-100G framework provides up to the 16 DMA channels. To fully utilize the current and near future CPUs, the desired number would be at least 64. It means a new architecture has to be prepared for at least 256 and possibly more.

Inter-process communication significantly decreases the throughput of network applications. Therefore, these applications do not scale with number of cores with respect to the throughput. In ideal case there is none inter-process communication. Fortunately, this is often the case of network packet processing. Network packets may be divided into disjunctive sets based on network flows or possibly by another key (see 2.2.3). These sets may be processed by different cores without any inter-process communication. Considering NICs, these sets are in ideal case mapped to completely independent DMA channels (each core has an exclusive access to one or more of them). This approach is used in the NetCOPE framework.

Problem of the current NetCOPE framework is that it does not scale well with number of DMA channels. Not well means that space complexity is $\mathcal{O}(n)$. To support this claim, a small research has been done with the current DMA module in both RX and TX directions. Synthesis results by Xilinx Vivado for FPGA UltraScale+ (xcvu9p-flgb2104-2-i) are summed up in separated Tables 4.2 (for RX) and 4.3 (for TX).

RX DMA channels	LUTs	Regs	BRAMs	Frequency [MHz]
1	8750	6031	16	469
2	10054	5677	16	443
4	10202	5934	32	398
8	10924	6440	64	430
16	12142	7429	120	419
32	14903	9382	232	376
64	20494	13878	456	324
128	33911	22222	912	293
256	54059	37311	2048	215

Table 4.2: Synthesis results of RX DMA module of the NetCOPE-100G framework for Xilinx UltraScale+

TX DMA channels	LUTs	Regs	BRAMs	Frequency [MHz]
2	11203	8451	16	364
4	12025	8997	32	430
8	14715	10056	64	430
16	20034	12143	120	360
32	31773	16271	232	380
64	53123	24490	456	321
128	97750	40863	912	241
256	187288	74622	2048	219

Table 4.3: Synthesis results of TX DMA module of the NetCOPE-100G framework for Xilinx UltraScale+

These results are visualized in three different graphs (each of them shows data for both RX and TX direction to save space). The first graph (Figure 4.3) shows number of consumed LUTs and registers with respect to a number of DMA channels. The linear scalability is clearly visible (be aware of the logarithmic scale of the x-axis). Consumed resources are enormous for 256 DMA channels. Obviously, it is not completely possible to avoid linear scalability. The goal is to minimize the number of components with linear scalability in favor of components with constant scalability.

The second graph (Figure 4.4) shows number of consumed BRAMs with respect to number of DMA channels. Space complexity is again linear. The amount of available on-chip memory is always limited (256 DMA channels already exceeds its size in this case). Therefore, it is worth to examine whether is possible to move from space complexity $\mathcal{O}(n)$ to $\mathcal{O}(1)$. In the the NetCOPE-100G framework, there is a separate buffer (each made of 8 BRAMs with data width 64 b) for every DMA channel in order to avoid writing collisions. This solution (Figure 4.5) is relatively simple but causes $\mathcal{O}(n)$ space complexity.

Writing collision is a situation when multiple producers demand the same consumer at the same time. In other words, writing collision occurs when multiple data blocks (data blocks are mapped to BRAM's words 1:1) are scheduled to be written in one clock cycle into the same BRAM (address is not significant). This scheduling has to be serialized. If writing collisions are solved in a more sophisticated way than buffer replication, it is possible to reduce space complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$. The writing collisions are solved by maximal

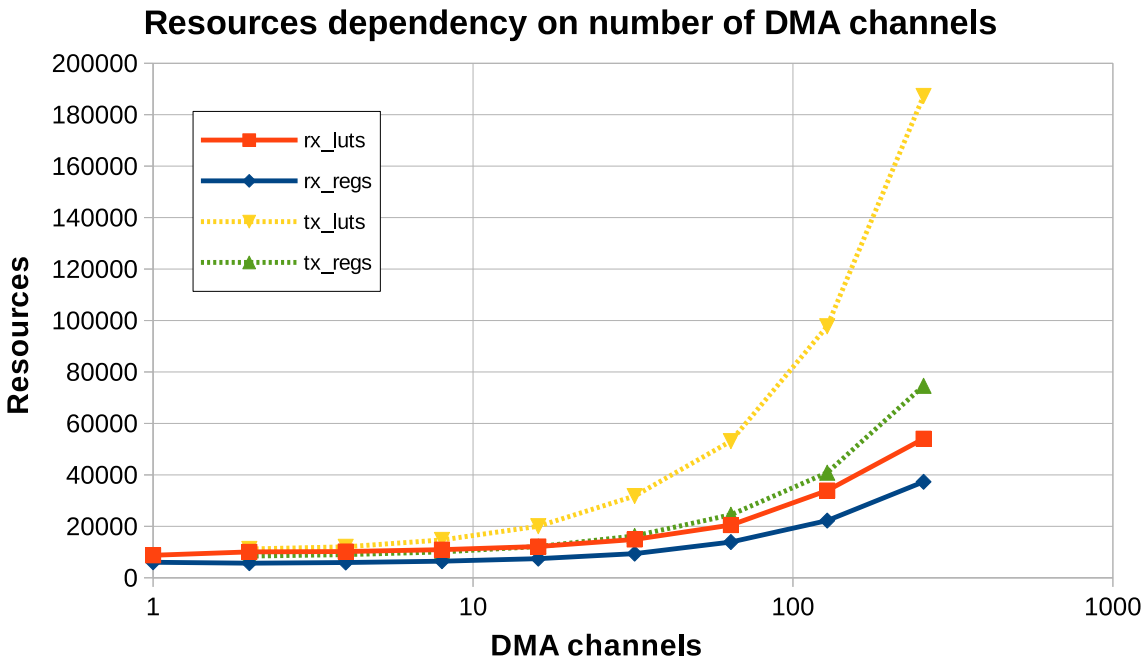

Figure 4.3: Resources dependency on number of DMA channels in both RX and TX in the NetCOPE-100G framework

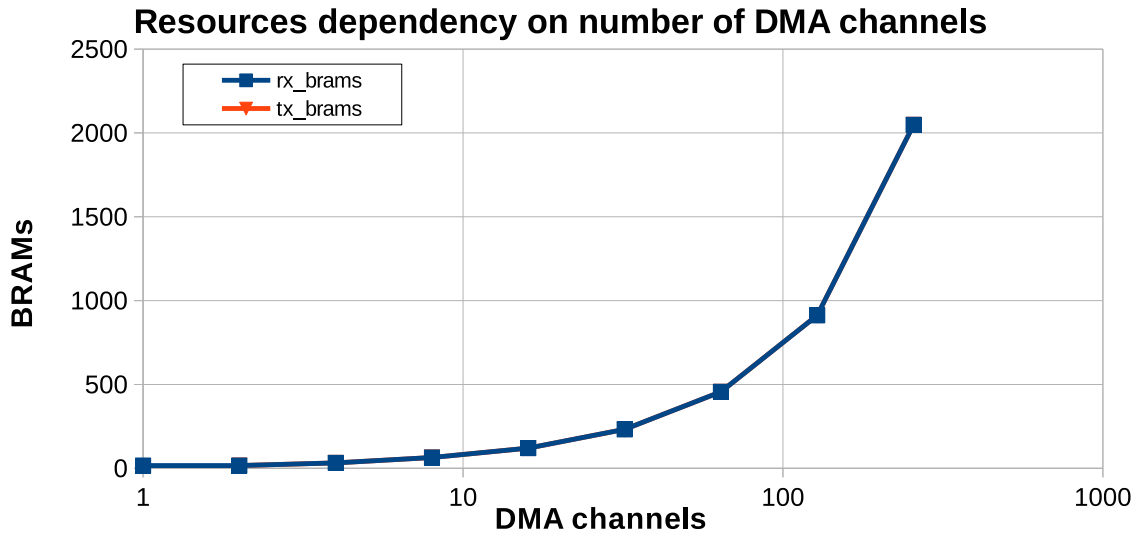

Figure 4.4: BRAMs dependency on number of DMA channels in both RX and TX in the NetCOPE-100G framework – both are identical

pair matching algorithm or its simplification (e.g. [22]). Modified block diagram of the current DMA buffer (Figure 4.5) would look like the block diagram in the Figure 4.6.

This solution with constant space complexity with respect to number of DMA channels shares just one buffer (made of 8 BRAMs with data width 64 b) for all DMA channels. A

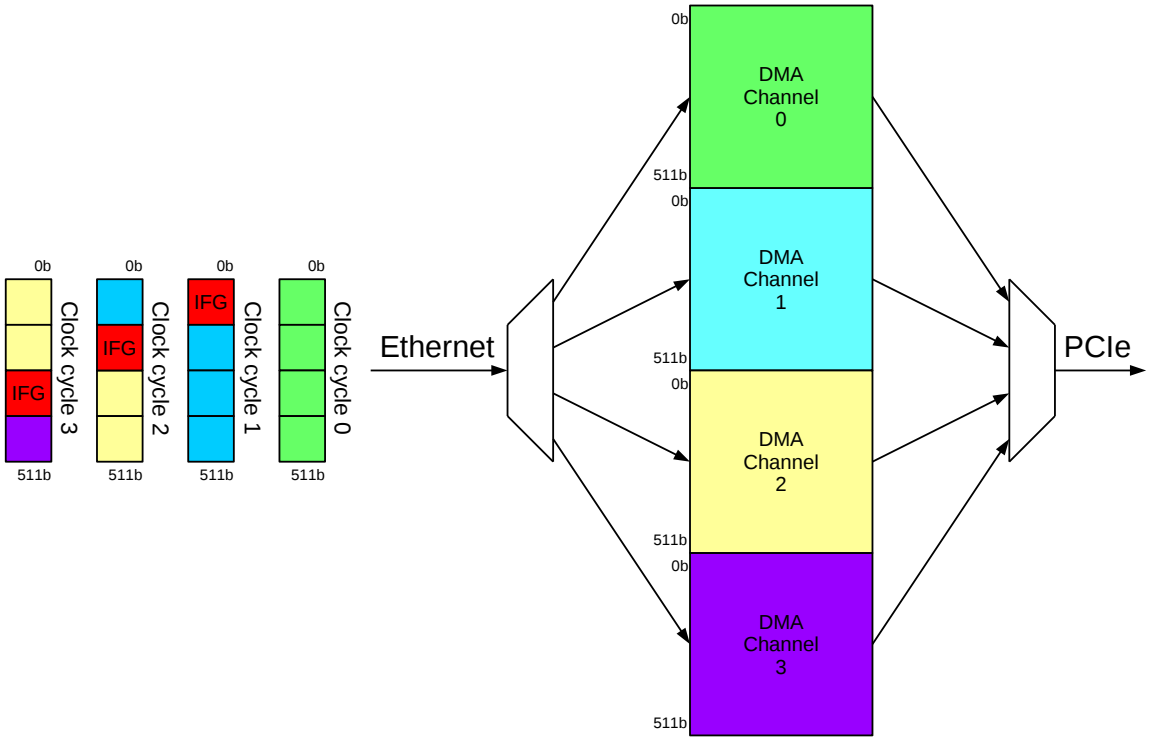

Figure 4.5: High-level diagram of the current RX DMA module architecture of the NetCOPE-100G framework

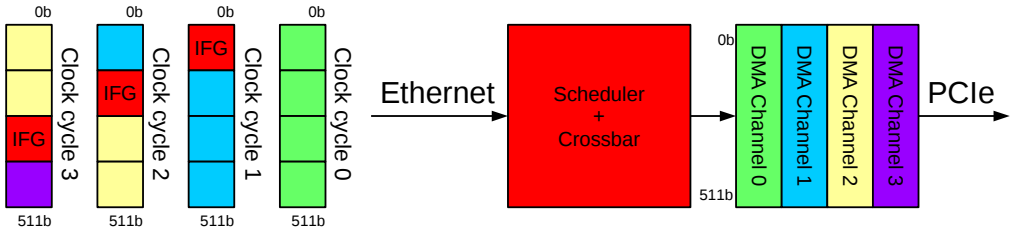

Figure 4.6: High-level block diagram of proposed RX DMA module architecture

number of DMA channels is theoretically unlimited. There are two possibilities of logical organization of the buffer. The first is to statically allocate memory for each DMA channel (the number of DMA channels is limited by a capacity of the buffer). The second is to allocate memory dynamically (number of DMA channels is theoretically unlimited).

Maximal pair matching algorithm works with granularity one data block – bigger the data block more simple the algorithm. The convenient size of the data block is 64 b for two reasons. Firstly, the start of frame delimiter is aligned on the current MII (see 2.1.1) protocol to 64 b and secondly, the data width of BRAM is also 64 b (in its widest configuration without parity bits). Using bigger data blocks would cause problems with alignment, using smaller would not be effective.

The third graph (Figure 4.7) shows the maximum frequency with respect to a number of DMA channels. Critical path should grow logarithmically, therefore maximum frequency decreases by multiplicative inverse of logarithm. Base of the logarithm is a number of inputs of target LUT – for Xilinx UltraScale+ it is 6.

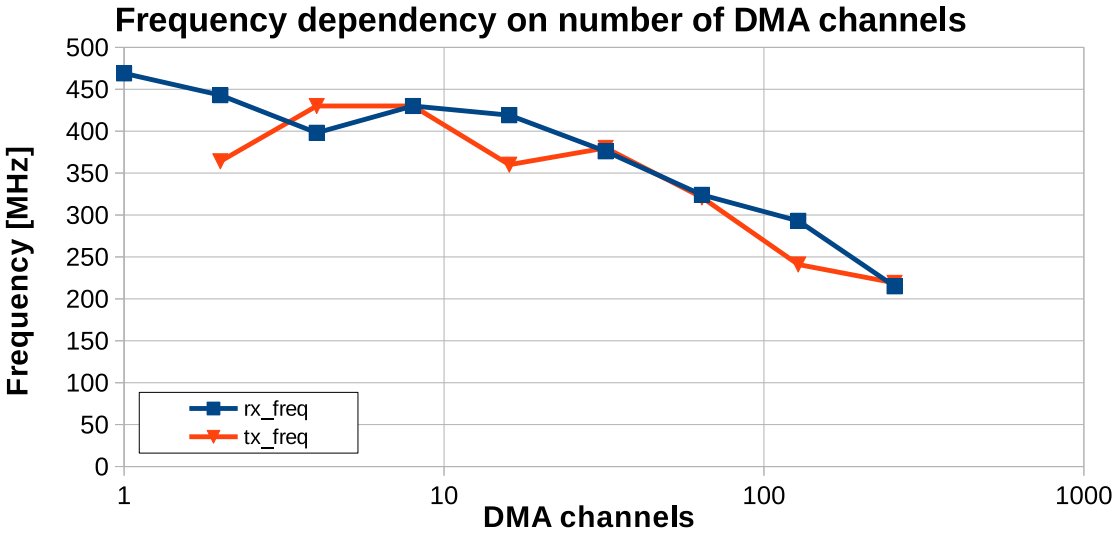

Figure 4.7: Frequency dependency on number of DMA channels in both RX and TX in the NetCOPE-100G framework – Xilinx Ultrascale+

4.1.3 Resource Demanding Processing Pipeline

Consumed resources grow quadratically with respect to increasing data width of the main data bus. As has been already mentioned, the speed of Ethernet grows faster than a frequency of FPGAs. Therefore, using wider data buses is inevitable – that means processing in FPGA has to be done more efficiently in order to fit in. The goal is to minimize resource demanding operations in a processing pipeline. Usually, the most resource demanding operation is data reordering on a main data bus (as already seen in the Figure 4.5).

Let us scrutinize the RX NetCOPE-100G framework processing pipeline (in the Figure 4.8) in order to find its weaknesses.

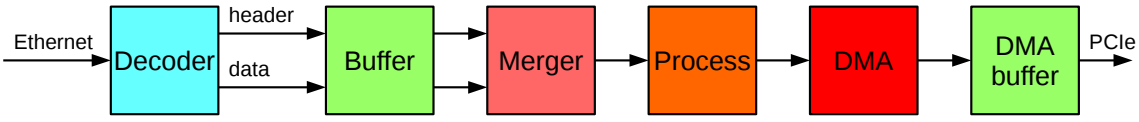

Figure 4.8: High-level block diagram of the current RX processing pipeline

It is clearly visible that data path goes through the whole pipeline. FLU protocol is used between processing blocks (see Table 4.1). Each processing block has to ensure consistency of FLU protocol at its output. Sometimes it means resources demanding data reordering on main data bus. In fact, in the processing pipeline is data reordered in `Merger`, `DMA` and maybe also in `Process` (depends on application) components. Is it possible to reorganize the processing pipeline in order to avoid or reduce the number of data reordering? Proposed solution is in the Figure 4.9.

Obviously, the length of the pipeline is smaller than in the previous case (see Figure 4.8). Design of hardware components is always divided between data and control path. In the proposed solution is this separation done on the top level entity (top-down approach).

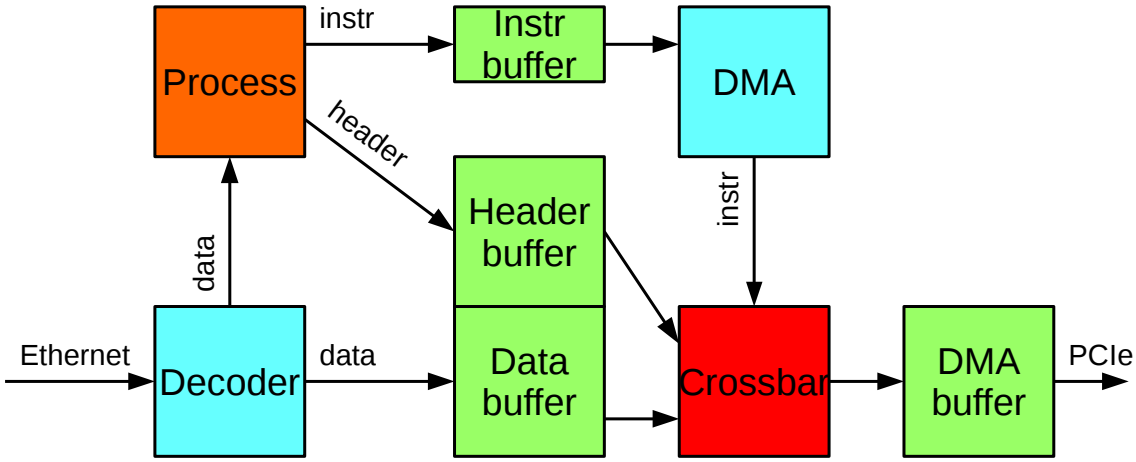

Figure 4.9: High-level block diagram of proposed RX processing pipeline

Result is that data reordering on the main data bus is performed just once in `Crossbar` component. This global optimization saves a substantial amount of resources.

Another local, but with global impact, optimization is to remove destination ready signal from the processing pipeline or its parts. Individual components may use destination ready signal to control input data stream. Destination ready signal is cleared thus the processing pipeline stopped when the component is unable to process the current input data word in the current clock cycle. Destination ready signal has to be propagated to all previous pipeline stages. Therefore, it has high fanout and it is often a part of a critical path. Critical path in a processing pipeline with a destination ready signal cannot be broken by a register because destination ready signal has to be propagated to all previous pipeline stages immediately. It can be broken by a component `Pipe`. `Pipe` is a FIFO with two items. `Pipe` clears its destination ready signal as soon as there is one item in the `Pipe`. Therefore, destination ready signal can be registered to break the critical path and thus delayed by one clock cycle because the `Pipe` has still one free item to store. Common processing pipeline with flow control ensured by destination ready signal works according to the Figure 4.10. The whole processing pipeline has to use flow control with destination ready if just one pipeline stage may clear its destination ready signal by itself. The optimization lies in separating the processing pipeline into parts with and without flow control with destination ready signal. The parts with flow control have to still use `Pipe` components to break critical path. The parts without flow control do not have to use destination ready signal and can break its critical path using registers. Parts with and without destination ready signals are joined together via a FIFO and its almost full signal is used for flow control. In this way, processing pipeline without flow control saves a large number of `Pipe` components and thus save resources. Number of items in FIFO is equal to a number of pipeline stages plus one. In an updated block diagram in the Figure 4.11 are two `Pipe` components (possibly much more) substituted by one FIFO component. Individual pipeline stages do not use flow control with destination ready signal. Flow control for multiple pipeline stages is provided by one FIFO component. This approach may increase the maximum frequency and reduces implementation complexity.

Space complexity grows by $\mathcal{O}(n)$ with respect to a number of DMA channels. As already mentioned it is not entirely possible to reduce the complexity to $\mathcal{O}(1)$. The goal is to minimize the number of components with $\mathcal{O}(n)$ complexity. NIC has to maintain a

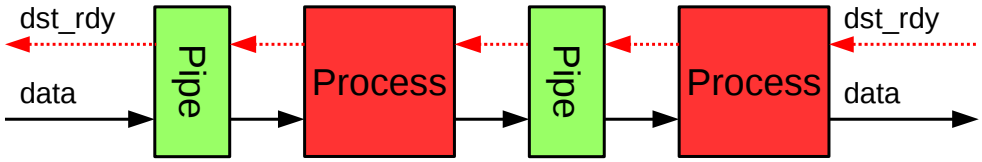

Figure 4.10: Principle of pipeline with destination ready signal

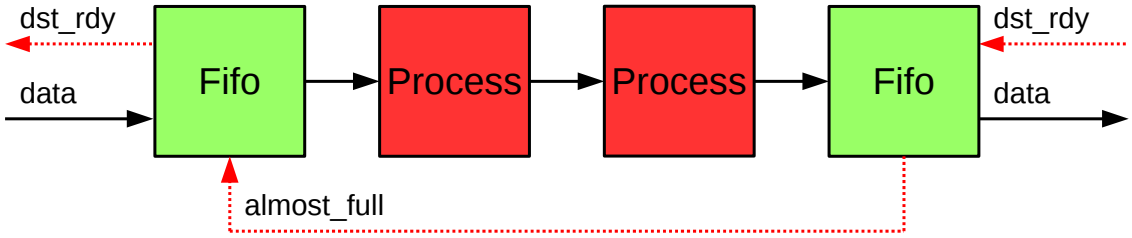

Figure 4.11: Principle of a pipeline with almost full signal

state of each DMA channel independently. The state is stored inside FPGA in registers. Multiple network packets (or its parts) may arrive each clock cycle (for 512 b width data bus it means two network packets or its parts). Therefore, just two `Process` components are required to handle any number of DMA channels for 512 b width data bus. The processing has three steps. The first step is DF (Data Fetch) – fetch a state of DMA channel from RF (Register Field). The second is EX (EXecution) – update the state of DMA channel with respect to the incoming network packet. The third is WB (Write Back) – write back updated state of DMA channel to RF. Described architecture is illustrated in the Figure 4.12.

Problem of this architecture is a very long critical path from RF via input multiplexer, `Process` component and output demultiplexer to RF. Size of the input multiplexer and output demultiplexer grows linearly with respect to the number of DMA channels. The critical path grows logarithmically. It is difficult to break this critical path because it contains a loop.

However, it is possible to process all three steps in pipelined manner. Unfortunately, the architecture is not as trivial as the previous one (Figure 4.12). Shortcuts in the pipeline from EX to EX and from WB to EX have to be created. Designed architecture is shown in the Figure 4.13.

The critical path from the previous architecture (Figure 4.12) is broken into three similar parts. However, the architecture is much more complicated, but the amount of resources does not grow significantly. This architecture (Figure 4.13) enables to increase a number of DMA channels even further and at the same time achieve a very high frequency. The Process component may be an adder, comparator or more complex logic.

4.2 System Requirements for 400 Gb Networks

This section is a milestone for the next chapter System Design 5. It sums up previously mentioned ideas into a list of requirements for the next generation of hardware accelerated NICs based on FPGA technology. These requirements are further described in detail:

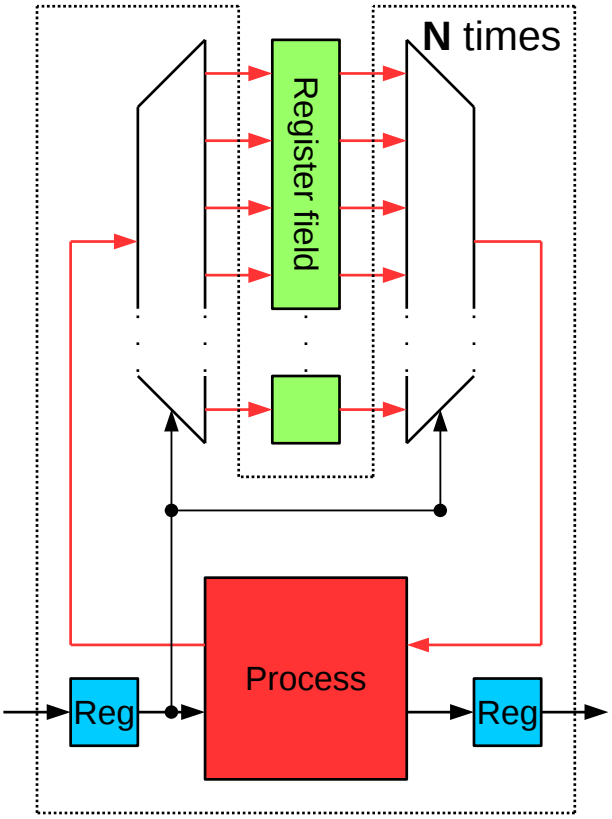

Figure 4.12: Block diagram of register field with multiple read and write ports

Very wide data bus $N \times 512\,b$ seems to be inevitable to catch up with 400 Gigabit Ethernet and beyond using FPGA technology. Data buses wider than 512 b brings many challenges. Multiple network packets may be transferred in just one clock cycle.

Multiple network packets per clock cycle processing corresponds to the necessity of usage wide data buses. A throughput of processing pipeline is measured by both frequency times data width (see equation 2.4) and also processed network packets per clock cycle (see equation 4.1).

Wire-speed reception and transmission via PCI Express to/from a host in order to take full advantage of new technology. Especially when it seems that PCI Express would be today a bottleneck of 400 Gb NIC (see 2.3.1). Upcoming PCI Express generation 4 with 16 lines reaches throughput up to 256 Gbps. Although PCI Express standard mentions also 32 lines connector, in practical implementations it does not exist. However, it is still possible to use two 16 lines connectors with throughput up to 512 Gbps. Moreover, each of these connectors may be served by different CPU withing computational node (typical node has, in fact, two CPU sockets). If network traffic is divided into two disjoint sets and each of them is served by a separated CPU, NUMA architecture and its features may be completely avoided and thus may significantly improve a performance of the system. Another approach to safe PCI Express bandwidth is to process part of network traffic within NIC without a participation of the host computer.

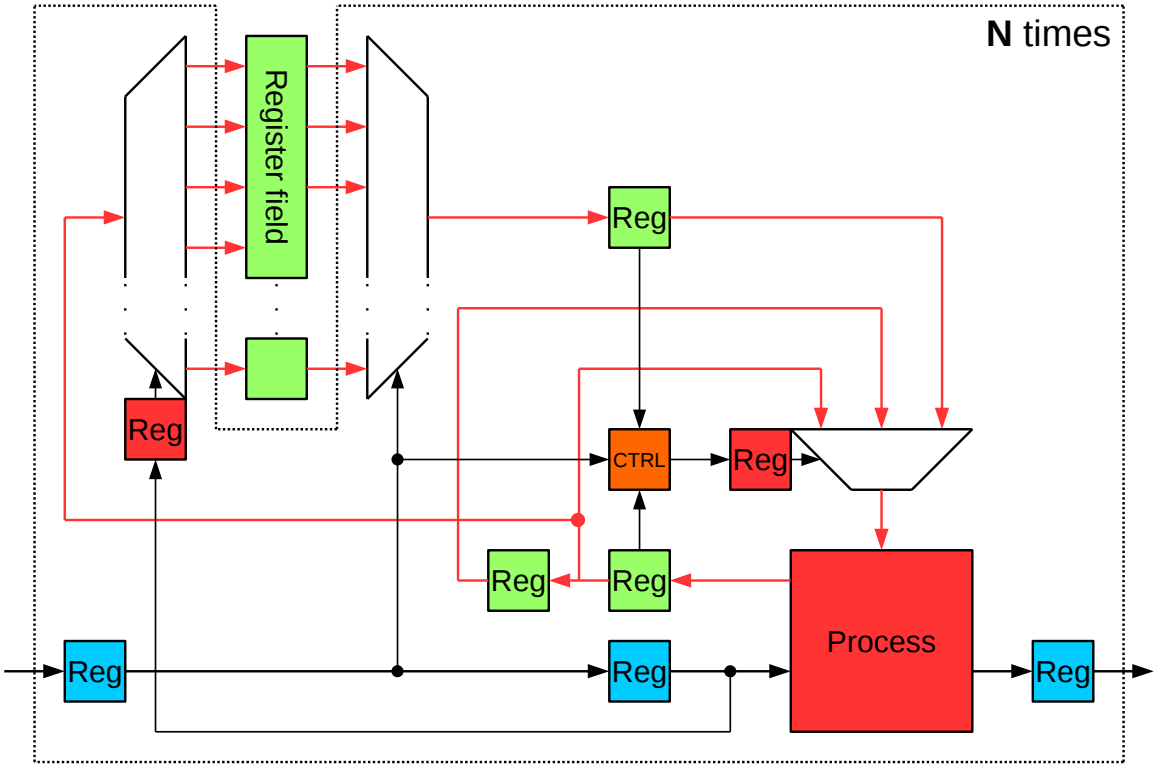

Figure 4.13: Block diagram of pipelined register field with multiple read and write ports

Scaling the number of DMA channels to hundreds and its importance was already mentioned as one of the biggest limitation of the NetCOPE-100G framework. A number of cores per CPU is increasing so has to number of DMA channels. Moreover, each core may utilize multiple queues based on QoS e.g. eight. Therefore, a number of DMA channels would be e.g. eight times larger than a number of cores. For two 32 cores CPUs, it means 512 DMA channels.

Editing and shortening network packets within NIC provides further functionality. Packet editing may be used to packing/unpacking network traffic to/from specified protocol e.g. adding/removing VLAN tag. It is very resources demanding because it is implemented as a data reordering on a main data bus. Packet shortening may be used increase packet throughput over PCI Express. To shorten network packet means to transmit just part of the packet and discart the rest. It does not consume many resources.

Assign metadata to every packet transferred via PCI Express to the host may be useful in many network applications. It is not always necessary to transfer the whole network packets to the host computer. For example network flow monitoring applications need only selected header fields up to the L4 layer of ISO/OSI model. Due to a usage of VLAN or MPLS tags, these headers do not always have static offsets. To ensure static offsets, Unified Header (UH) containing all headers with static offsets has to be used. These UH are then much more easily and with much greater performance processed by the host computer.

Retransmitting from RX to TX Ethernet within NIC allows network traffic to avoid the host computer. Traffic may be discarded or edited within NIC and retransmitted to TX ethernet port without the host computer would get directly involved. This may save significant resources of the host computer. A typical application may be a DDoS protector discarding malicious traffic. Be aware that simple retransmitting from RX to TX Ethernet port is not possible because upon reception may be inter frame gap shorter than is allowed upon transmission.

Wire-speed reception and transmission via Ethernet has to be supported by the proposed framework in order to achieve the wire-speed throughput for network applications based on the framework. Many high-speed NICs are limited by a number of processed network packets per second. Therefore, they are not able to process all of the shortest Ethernet packets at the wire-speed.

Multiple Ethernet ports support is an optional requirement. However, it enables to support high-density solutions. Despite e.g. second generation of 100 Gigabit Ethernet NICs has just one transceiver, they internally use multiple lines working together (see 2.1.1). These lines may also operate separately, each of them being independent Ethernet port with corresponding speed. The consequence is that the same hardware may be used for a single-port or a multi-port NIC. The mode is determined by FPGA firmware.

Parsing up to L4 of ISO/OSI model is a building block for other more sophisticated network functions. Standard NICs usually parse network packets up to L2 of ISO/OSI model. Hardware accelerated NICs operating up to L4 layer are able to distinguish between network flows. There are many important network functions based on network flows such as filter, classification, load balancer, rate limiter, etc. This function is not a part of the proposed framework but the framework has to enable its implementation and not vice versa.

Stateful processing within NIC is necessary for some network function such as network flow statistics or rate limiter. State processing is one of the reasons why very wide data bus cannot be easily replaced by multiple more narrow buses – multiple parallel processing pipelines. In order to ensure stateful processing, would have to these pipelines share states. This function is not a part of the proposed framework but the framework has to enable its implementation and not vice versa.

Chapter 5

System Design

This chapter proposes multiple novel frameworks for hardware acceleration of 400 Gigabit Ethernet. Frameworks were designed according to the previous chapter System Analysis 4. Emphasis is put on high frequency and performance, low resource consumption (especially of on-chip memory), a large number of completely independent DMA channels and acceptable implementation complexity with respect to the current technology.

RX and TX data paths (from the host computer perspective) frameworks are at first designed separately and subsequently joined together with added data path from RX toTX. Multiple Ethernet ports are also considered. All frameworks are characteristic by „`BUFFER->CROSSBAR->BUFFER`“ data path with sophisticated scheduler in order to significantly save on-chip memory.

5.1 RX Data Path

The first proposed framework is single RX data path, meaning the framework contains one Ethernet port without TX data path (from the host computer perspective). The framework is capable of reception ingress network traffic to the application core and high-speed transfers to the host computer via PCI Express in many completely independent DMA channels. Application core is a firmware application based on the proposed framework. Such application may e.g. implement packet header analysis, simple packet classification, filter, and load balancer distributing network traffic into a large number of completely independent DMA channels, subsequently processed by the host computer in a parallel matter.

In the following Figure 5.1, is shown a high-level block diagram of this framework. Ethernet stream MII (Table 2.1) is decoded by `Decoder` and subsequently stored in `Data buffer`. Simultaneously, the decoded Ethernet stream is processed by the application core (`Process`). The application has a full control over the network packets stored in `Data buffer`. It may assign also a metadata for every packet. This metadata is stored in `Header buffer`. Description of network packets and its metadata is sent to `DMA ctrl` as packet instructions (`Pkt. instr`). `Scheduler` transforms network packets into PCI Express transactions with respect to a DMA channel. It controls `Crossbar` in order to transfer network packets and its metadata into `DMA` buffer. These transfers have to be planned by maximal pair matching algorithm (see [22]) in order to avoid writing collisions. `AXI` reads PCI Express transactions from `DMA buffer`, adds PCI Express headers from `DMA ctrl` and sends transactions via PCI Express in the host computer memory.

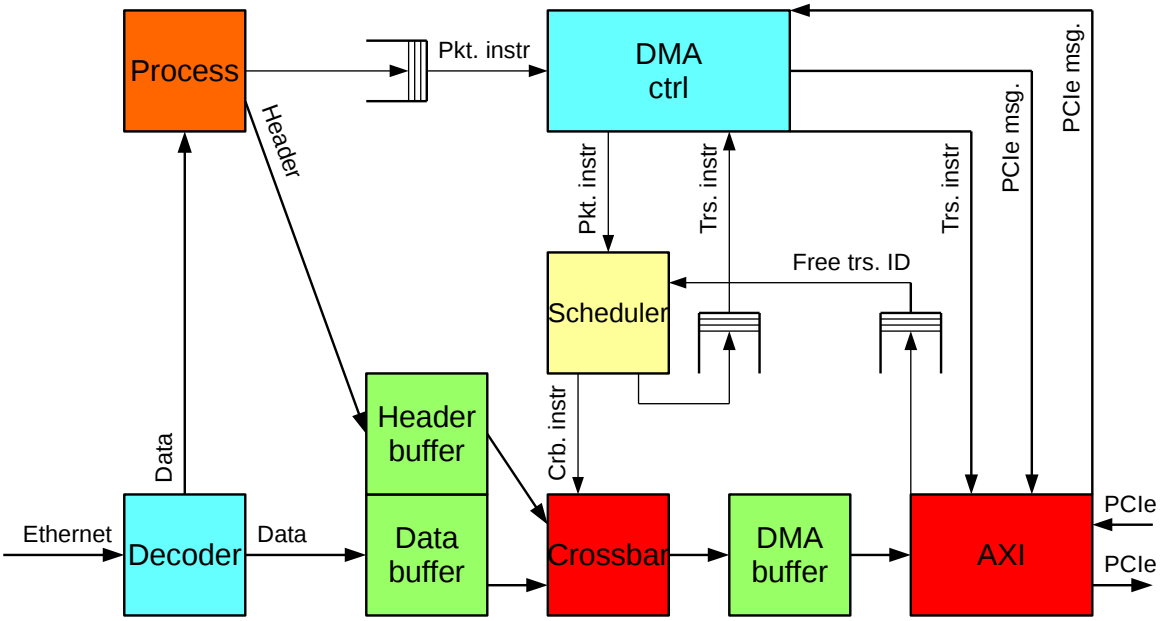

Figure 5.1: Block diagram of the proposed Framework of single RX data path

Decoder decodes raw MII stream which is not suitable for further processing. Its function is to find all valid start and end of frame delimiters in a potentially random data stream. Network packets may have invalid packet length or be damaged. It also maintains write pointer (an approximation of a read pointer into `Data buffer` is provided by `Scheduler`). To avoid write pointer rollback due to a buffer overflow, network packets are discarded if there is insufficient space in `Data buffer` for the longest valid network packet. Packets longer than MTU are terminated. `Decoder` produces metadata about network packets such as protocol error or MTU error to `Process`.

Data buffer stores network packets in store and forward manner because application core (`Process`) finishes packet processing at the end of the packet. It contains multiple on-chip memories. Data width of each memory and thus its number is determined by a size of alignment of the framework. Alignment is determined by properties of the Ethernet stream MII. Alignment just 1 B consumes enormous resources but allows to add/remove any packet headers on all layers of ISO/OSI model. Alignment 8 B is sufficient to receive network packets into the host computer because the beginning of network packets are also aligned on the Ethernet stream MII to 8 B. Alignment to 8 B saves significant resources but it is impossible to add/remove packet headers on layers of ISO/OSI model.

Header buffer stores network packet headers provided by application core (`Process`). The buffer is divided into uniform regions. A number of regions depends on a number of processed packets per clock cycle. The header for corresponding network packet in `Data buffer` is aligned with its start of frame delimiter (their regions and column addresses are the same in both buffers). Unlike a start of frame delimiter, packet header always starts at the beginning of the region.

Process is an application core based on the framework. It determines the functionality of the acceleration card. It consumes network packets and produces packet instructions (`Pkt. instr`) and packet headers (`Header`). This component is extremely complex. The component may be described by a high-level language P4 (2.2.2).

DMA ctrl controls high-speed transfers via PCI Express to/from the host computer. It maintains a state of DMA ring buffers in the host computer memory, downloads memory descriptors from the host computer, update read and write pointers to buffers in the host computer memory, initiates transfers in both RX and TX direction, etc. It consumes packet instructions from `Process` and with respect to free space in corresponding DMA ring buffer in the host computer memory. In order to be DMA channels completely independent, discarding has to be used. These packet instructions are subsequently sent to `Scheduler`. They are processed into transaction instructions (`Trs. instr`) and return back to `DMA ctrl`. Transaction instructions are completed (physical address into the host computer memory is added) and sent to `AXI`.

Scheduler transforms network packets into PCI Express transactions with respect to a DMA channel. It controls `Crossbar` in order to transfer network packets and its metadata into `DMA` buffer. These transfers have to be planned by maximal pair matching algorithm (see [22]) in order to avoid writing collisions. Details are described in dedicated section 5.1.1.

Crossbar is part of the data path implemented by multiplexers. It enables to transfer network packets and its metadata from `Data buffer` and `Header buffer` into `DMA buffer`. It is controlled by `Scheduler`, otherwise, it does not have any other functionality. A size of `Crossbar` depends on the alignment of the framework as already mentioned in 5.1.

DMA buffer stores PCI Express transactions. Transactions of all DMA channels share this buffer. Data width of the buffer depends on a data width of the data bus towards PCI Express. Space for transactions is allocated dynamically. Free transaction IDs (addresses to `DMA buffer`) are maintained by `AXI` and stored in FIFO. `Scheduler` consumes these IDs from the FIFO and `AXI` produces them.

AXI communicates with PCI Express integrated blocks in FPGA. It uses standard AXI protocol. It consumes transaction instructions (`Trs. Instr`) from `DMA ctrl` and PCI Express payloads from `DMA buffer`, subsequently completes PCI Express transactions and sends them towards the host computer memory via PCI Express.

5.1.1 Scheduler for Single RX Data Path

One of the core components used by single RX data path is `Scheduler`. `Scheduler` with modified functionality is also used by all other proposed frameworks. `Scheduler` completely controls component `Crossbar` which does not have any other functionality and is actually the control path of the „`BUFFER->CROSSBAR->BUFFER`“ data path. It reads data from an input buffer and transfers it via a crossbar into an output buffer in a different form. In other words, it efficiently implements a function of data reordering on the main data bus. It consumes a description of data in an input buffer and produces a description of data in an output buffer and control signals for crossbar.

A Description of data in an input buffer, in this case, means instructions of network packets (`Pkt. instr`), an output buffer is described by instructions of PCI Express transactions (`Trs. instr`). Data reordering is scheduled with respect to DMA channel (see Figure 3.3), memory paging in the the host computer and MTU of PCI Express protocol. Subsequently, network packets are divided into small data blocks and these are independently planned in order to avoid writing collisions into an output buffer by a maximal pair matching algorithm (see [22]).

In the Figure 5.2 is given detailed architecture of `Scheduler` for single RX data path. Packets instructions (`Pkt. instr`) describe network packets and its metadata in `Data buffer` and `Header buffer`. They are produced by an application core (`Process`). `Packet breaker` breaks these network packet instructions with respect to data words. A single network packet may be spread over multiple data words. This is not practical for further processing. Therefore, network packets are divided into smaller units subpackets. A subpacket does not spread over multiple data words. A subpacket may contain a network packet or its part. Subpackets are described by subpackets instructions (`Spkt. instr`). `Increment per DMA` computes increments of subpackets aggregated by DMA channels. An increment is a number of bytes of aggregated subpackets lengths by DMA channels. Each of these increments (`Inc. DMA`) may be breaked into two increments by `4K page breaker + barrier` with respect to memory paging in the host computer because PCI Express transactions cannot spread over two memory pages (defined by the standard). These increments are broken by `MTU breaker` with respect to MTU of PCI Express protocol. The output of `MTU breaker` is transaction instructions (`Trs. instr`) and subtransaction instructions (`Strs. instr`). Transaction instructions describe PCI Express transactions in `DMA buffer`. Subtransaction instructions are the description of a data word in a form of PCI Express transactions or its parts. `Crossbar instr generator` transforms the description of a data word in subtransaction instructions with respect to the description of the data word in subpacket instructions into crossbar instructions (`Crb. instr`). Each of crossbar instruction describes one fixed-size data block. These fixed-size data blocks are planned by `Planner` independently in order to avoid writing collisions into `DMA buffer`. Planner is an implementation of a maximal pair matching algorithm (see [22]) or its approximation. `Trans FIFO ctrl` releases transaction instructions after its transactions are already transferred into `DMA buffer`.

Packet breaker breaks packets into subpackets. Packets are described by address, packet length, and header length. They may exceed one data word (column) in `Data buffer` and therefore spread over multiple clock cycles. Transfers from `Data buffer` to `DMA buffer` are planned in fixed-size data blocks. It is not necessary to plan data blocks of the whole network packet in one clock cycle. The network packet may be divided into data words, exactly as it was received by Ethernet. This division is convenient for further processing. In other words, network packet may be processed into PCI Express transaction in multiple clock cycles. Subpackets are also described by address, packet length, and header length but they do not exceed one data word. Therefore, subpackets describe network packets or their parts. `Packet breaker` also takes care of flow control.

Increment per DMA aggregates all network subpackets with respect to DMA channel. The result is one increment per DMA channel (`Inc. DMA`). It also stores subpacket instructions (`Spkt. instr`) in FIFO. They are consumed by `Crossbar instr generator`.

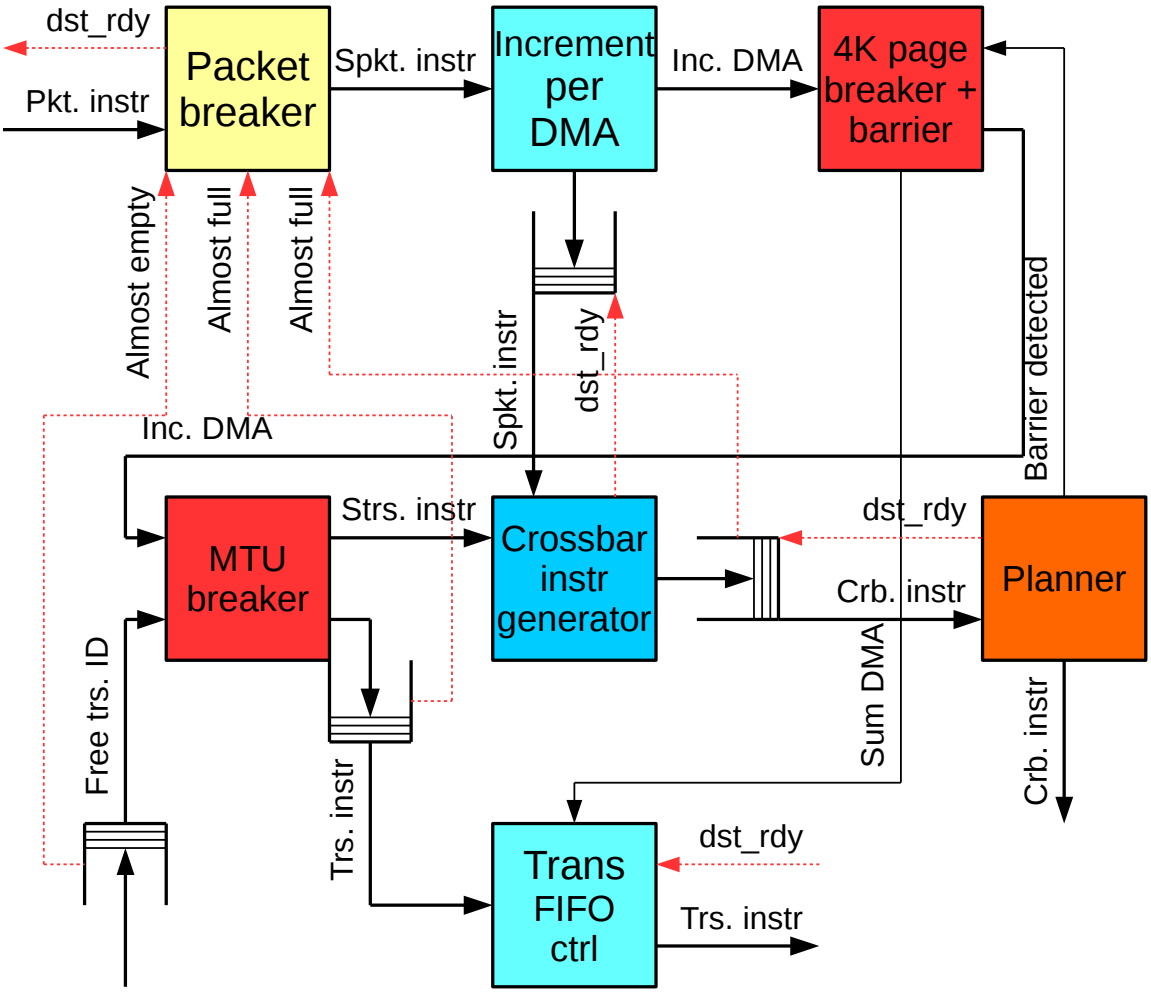

Figure 5.2: Block diagram of proposed detailed architecture of `Scheduler` for single RX data path

4K page breaker + barrier breaks increments per DMA (`Inc. DMA`) with respect to memory page size in the host computer – 4 KB. PCI Express transactions cannot be spread over two memory pages because the PCI Express standard does not allow it. The result is two increments per DMA channel. `4K page breaker` has to maintain a state of each DMA channel – it uses a register field (block diagram is in the Figure 4.12) or pipelined register field (block diagram is in the Figure 4.13). `4K page breaker` also takes care of synchronization. It produces increments either with black or white color. Each time it receives signal `Barrier detected` from `Planner`, changes the color. It maintains a confirmed state of a number of bytes already transferred into `DMA buffer` per DMA channel. The confirmed state is computed in three stages. In the first stage, a current state of a number of bytes already processed (increments are sent to output) per DMA channel is being updated. In the second stage when the color is changed, the current state is sampled in a sampled state. In the third stage when the color is changed again, the sampled state is stored into a confirmed state (`Sum DMA`). Each time `Barrier detected` signal is set, `Planner` ensures that all data blocks with the previous color have already been transferred

into DMA buffer. Be aware that data blocks may be potentially transferred out-of-order (depends on Planner).

MTU breaker breaks two increments per DMA channel into PCI Express transactions with respect to PCI Express MTU. Multiple transactions may be completed in just one clock cycle but also it may take multiple clock cycles to complete a transaction – depends on the current state of MTU breaker and inputs. It has to maintain a counter of bytes and current transaction ID per each DMA channel. It produces transaction instructions (Trs. instr) of already completed transactions and subtransaction instructions (Strs. instr) describing PCI Express transactions or its parts (unlike transaction instructions produced each clock cycle). Subtransaction instructions are consumed by Crossbar instr generator. Memory for transactions in DMA buffer is allocated dynamically. Free transaction IDs (Free trs. ID) are consumed from AXI. When transactions are read from DMA buffer by AXI its IDs are freed. MTU breaker contains timeout counter for each DMA channel in order to avoid its starvation – MTU breaker always tries to fill up PCI Express transaction up to its MTU in order to save PCI Express bandwidth but in some cases, a timeout has to terminate an incomplete transaction.

Transaction FIFO ctrl controls read from transaction instructions FIFO. Transactions instructions are produced by MTU breaker at the time they are scheduled but actual data appears in DMA buffer with delay. Transaction FIFO ctrl has access to confirmed state of number of bytes already transferred into DMA buffer per DMA channel from 4K page breaker + barrier (Sum DMA). If length of transaction on the top of the transaction instructions FIFO is less or equal to confirmed state of appropriate DMA channel minus actual state of appropriate DMA channel in Transaction FIFO ctrl, transaction instruction is freed. Therefore, Transaction FIFO ctrl has to maintain a counter for each DMA channel.

Crossbar instr generator generates crossbar instructions (Crb. instr). Crossbar instructions describe fixed size data blocks (for considerations about their size see 5.1) from subtransaction instructions (Strs. instr) with respect to subpacket instructions (Spkt. instr). Corresponding subtransaction and subpacket instructions both describe the same data word of Header buffer and Data buffer. At the end network packets are broken into many small data blocks described by crossbar instructions suitable to be planned in Planner independently.

Planner plans crossbar instructions (Crb. instr) in order to avoid writing collisions into DMA buffer by a maximal pair matching algorithm (see [22]) or its approximation. Each clock cycle, Planner should be able to plan transfers to fully utilize DMA buffer's bandwidth. Otherwise, performance is degraded. Maximal pair matching algorithms usually do not fully utilize output buffer's bandwidth. Either it is possible to increase a little bit frequency of Scheduler and Crossbar or Planner has to plan two transfers per clock cycle and Crossbar has to run at double frequency. Planner consumes either black or white (see 5.1.1) crossbar instructions. After reset, black crossbar instructions have priority. As soon as Planner detects there are no black crossbar instructions anymore, it sets Barrier detected bit and change priority in favor to white crossbar instructions.

5.2 Multiple RX Data Paths

This framework extends single RX data path (5.1) with multiple Ethernet ports instead of one. In fact, it joins multiple Ethernet streams into a single Ethernet stream. Processing of single Ethernet stream is already designed in the section 5.1. Ethernet streams are joined by `Scheduler` with „`BUFFER->CROSSBAR->BUFFER`“ data path. The output is one very wide data bus shared by all Ethernet ports. The output is an input of already proposed RX data path framework 5.1.

In the Figure 5.3 is shown architecture of joining multiple Ethernet streams. Multiple `Parser` components process multiple Ethernet streams. `Parser` extends the functionality of `Decoder` mentioned in section 5.1 by computing packet lengths. Decoded Ethernet streams are stored in multiple `Buffer` components. `Buffer` component is very similar as `Data buffer` already mentioned in section 5.1. `Scheduler` schedules transfers from `Buffer` to `Data buffer` via `Crossbar` in order to join all Ethernet streams into just one Ethernet stream. `Crossbar` is already described in section 5.1. `Data buffer` has the same properties as `Buffer` but it stores just one joined Ethernet stream.

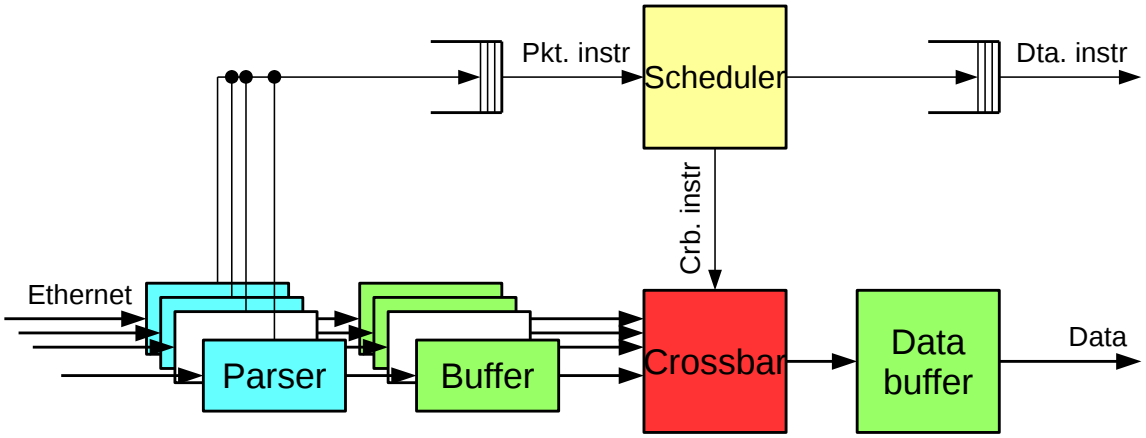

Figure 5.3: Block diagram of proposed Framework of multiple RX data paths

Parser extends the functionality of `Decoder` mentioned in 5.1. Moreover, it computes lengths of network packets. Extracted metadata is sent to `Scheduler` as packet instructions (`Pkt. instr`).

Scheduler schedules transfers from `Buffer` to `Data buffer` in order to merge all Ethernet ports into just one MII stream. It consumes packet instructions `Pkt. instr` from `Parser`. They contain addresses and lengths of network packets. It controls `Crossbar` and produces data instructions (`Dta. instr`) describing data in `Data buffer`. Since MII is a stream protocol, data instructions may be reduced to write pointer into `Data buffer`.

5.3 Single TX Data Path

Single TX data path contains one Ethernet port without RX data path (from the host computer perspective). The framework is capable of transmission egress network traffic from the host computer via PCI Express into the application core and its subsequent transmission

over Ethernet. The application core may e.g. modify, add/remove packet headers on all layers of ISO/OSI model. It can be used as a hardware accelerated network packets generator.

In the following Figure 5.4, is shown a high-level block diagram of the framework. Network packets are transferred from the host computer memory via PCI Express interface. `AXI` communicates with PCI Express integrated blocks in FPGA. Transfers from the host computer to the acceleration card are controlled by `DMA ctrl`. `DMA ctrl` and `AXI` are the same components as `DMA ctrl` and `AXI` already mentioned in the section 5.1. AXI stream, the output of `AXI`, is stored in `AXI buffer`. `AXI buffer` is similar to `Data buffer` already mentioned in the section 5.1. AXI stream contains raw data from PCI Express interface. PCI Express read transactions initiated by `DMA ctrl` may be split into many parts and transferred out-of-order. `AXI` parses this AXI stream and provides metadata to `AXI scheduler` as AXI instructions. Metadata contains number of DMA channel, PCI Express read transaction TAG, offset within the transaction and its length. `AXI scheduler` schedules transfers from `AXI buffer` to `Trans buffer` via `AXI crossbar`. Its purpose is to reorder AXI stream into PCI Express transactions as they were requested by `DMA ctrl`. `AXI crossbar` is very similar to `Crossbar` already mentioned in the section 5.1. `Trans buffer` stores PCI Express read transactions containing network packets. These network packets are processed by `Process`. `Process` is an application core based on the framework. `Process` may modify network packets, stores new packet headers into `Header buffer` and produces packets instructions (`Pkt. instr`) to `Trans scheduler`. Packets instructions describe already modified network packets by `Process`. Packets instructions are executed by `Trans scheduler`. `Trans scheduler` schedules transfers from `Trans buffer` to `Data buffer` via `Trans crossbar` in order to extract network packets from PCI Express transactions and create valid Ethernet stream with respect to inter frame gap (see 2.1.1). `Encoder` reads Ethernet stream from `Data buffer`, computes network packets CRC, and encodes valid Ethernet stream MII. `Header buffer`, `Trans crossbar`, and `Data buffer` components are very similar as `Header buffer`, `Crossbar`, and `Data buffer` already mentioned in the section 5.1.

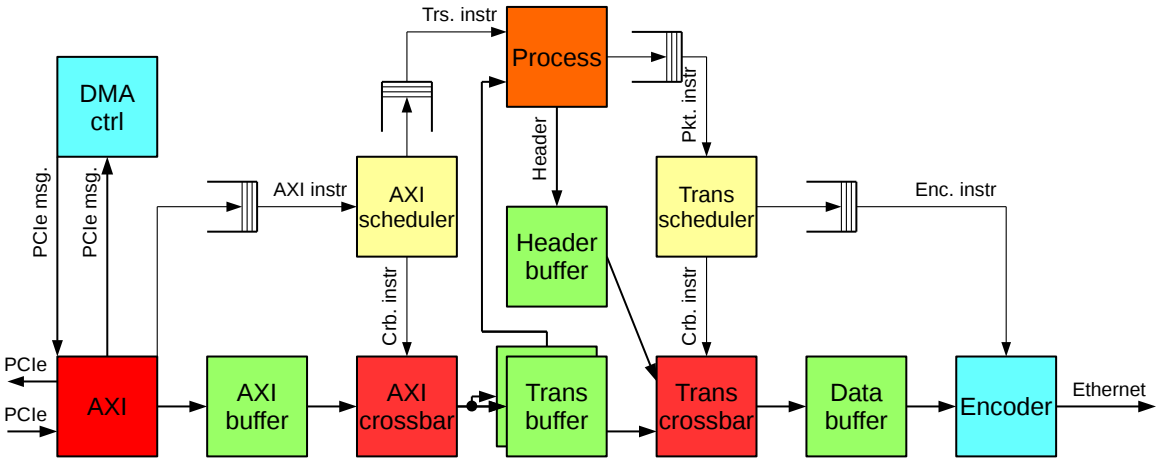

Figure 5.4: Block diagram of proposed Framework of single TX data path

AXI communicates with PCI Express integrated blocks in FPGA. It uses standard AXI protocol. It produces AXI instructions (`AXI instr`) to `AXI scheduler`. These in-

structions contain metadata of AXI stream. Completer (the host computer) may return requested read transactions potentially out-of-order and divided into many smaller answers. In one clock cycle AXI stream may contain multiple transaction parts of multiple read transactions. Transactions are identified by TAG and offset within a transaction. This metadata is vital to reorder data from AXI stream by `AXI scheduler` into originally requested read transactions.

DMA ctrl controls high-speed transfers via PCI Express to/from the host computer. It maintains state of DMA ring buffers in the host computer memory, downloads memory descriptors from the host computer, update read and write pointers to buffers in the host computer memory, initiates transfers in both RX and TX direction, etc. As already mentioned requested read transactions may be returned out-of-order and divided into many small parts.

AXI scheduler schedules transfers from `AXI buffer` to `Trans buffer`. Its purpose is to reorder AXI stream into PCI Express transactions as they were requested by `DMA ctrl`. It consumes AXI instructions (`AXI instr`) from `AXI`, controls `AXI crossbar` and produces transaction instructions (`Trs. instr`) to `Process`. These transaction instructions have to contain an address, a length, and a number of DMA channel.

Trans buffer stores read transactions as they were requested by `DMA ctrl`. The buffer is doubled because there is also stored metadata of network packets within transactions itself which has to be parsed by `Process` in order to transmit network packets by Ethernet (metadata contains address, length and DMA channel of network packets).

Process is an application core based on the framework. It parses PCI Express transactions in order to extract address, length and DMA channel of network packets. Optionally, it may implement packet classification and add/remove/modify packet headers before they are sent to Ethernet interface. It consumes transaction instructions (`Trs. instr`) and produces packet instructions (`Pkt. instr`) and network packet headers (`Header`). `Process` may be described by high-level language P4 (2.2.2).

Trans scheduler schedules transfers from `Trans buffer` to `Data buffer` in order to extract network packets from PCI Express transactions and create valid MII stream with respect to inter frame gap (see 2.1.1). It consumes packet instructions (`Pkt. instr`) from `Process`, controls `Trans crossbar` and produces encoder instructions (`Enc. instr`). Since in `Data buffer` is still not completely valid MII stream, encoder instructions cannot be reduced to just write pointer into `Data buffer` but they have to contain metadata such as address and length of network packets in order to encode MII stream.

Encoder encodes MII stream. It consumes encoder instructions `Enc. instr` from `Trans scheduler`, computes CRC of network packets and adds control characters of MII stream.

5.4 Multiple TX Data Paths

This framework in the Figure 5.5 extends single TX data path (Figure 5.3) with multiple Ethernet ports instead of one. Data path is the same as in single TX data path with few modifications. `Data buffer` and `Encoder` components are duplicated as many times as is the number of Ethernet ports. Component `Trans scheduler` has to schedule transfers from `Trans buffer` to `Data buffer` components via `Trans crossbar`. `Trans scheduler` has to ensure consistency of Ethernet protocol for each Ethernet port. Therefore, `Trans scheduler` has to maintain states of all Ethernet ports. Components `Process`, `Header buffer`, `Trans buffer`, and `Trans crossbar` are exactly the same as in single TX data path in the section 5.3.

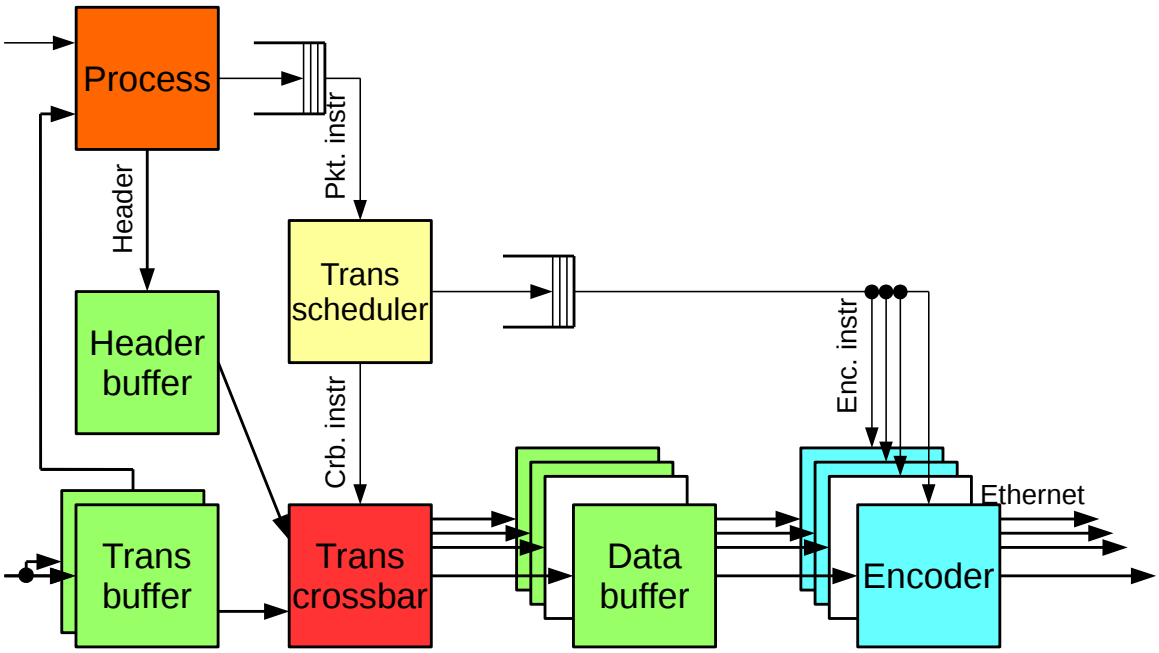

Figure 5.5: Block diagram of proposed Framework of multiple TX data paths

5.5 Multiple RX and Multiple TX Data Paths

This framework in the Figure 5.6 provides the most functionality. It covers both RX and TX data paths with multiple Ethernet ports. The framework consists of multiple RX data paths from the section 5.2, multiple TX data paths from the section 5.4, and added data path from RX to TX. Multiple Ethernet ports are not in the block diagram shown because extension from single RX and single TX data paths to multiple RX and multiple TX data paths framework was already explained in the sections 5.2 and 5.4. Added data path resulted in buffer replication – `RX header buffer` and `RX data buffer` are doubled. Component `Trans crossbar` has to be enlarged accordingly. Be aware that components `AXI` and `DMA ctrl` in the block diagram (Figure 5.6) for RX and TX data paths are identical.

Packets instructions (`Pkt. instr`) are extended to describe network packets either in `RX data buffer` or `TX data buffer`. `RX data buffer` stores Ethernet stream, `Trans buffer` stores PCI Express transactions. Nevertheless, they both contain network packets.

Packets instructions have to be able to describe network packets from multiple packet parts stored in different buffers. `Trans scheduler` has to be able to execute these instructions. Therefore, data reordering on the main data bus is performed just once. The framework is able to transmit network packets either from the host computer or from RX Ethernet ports.

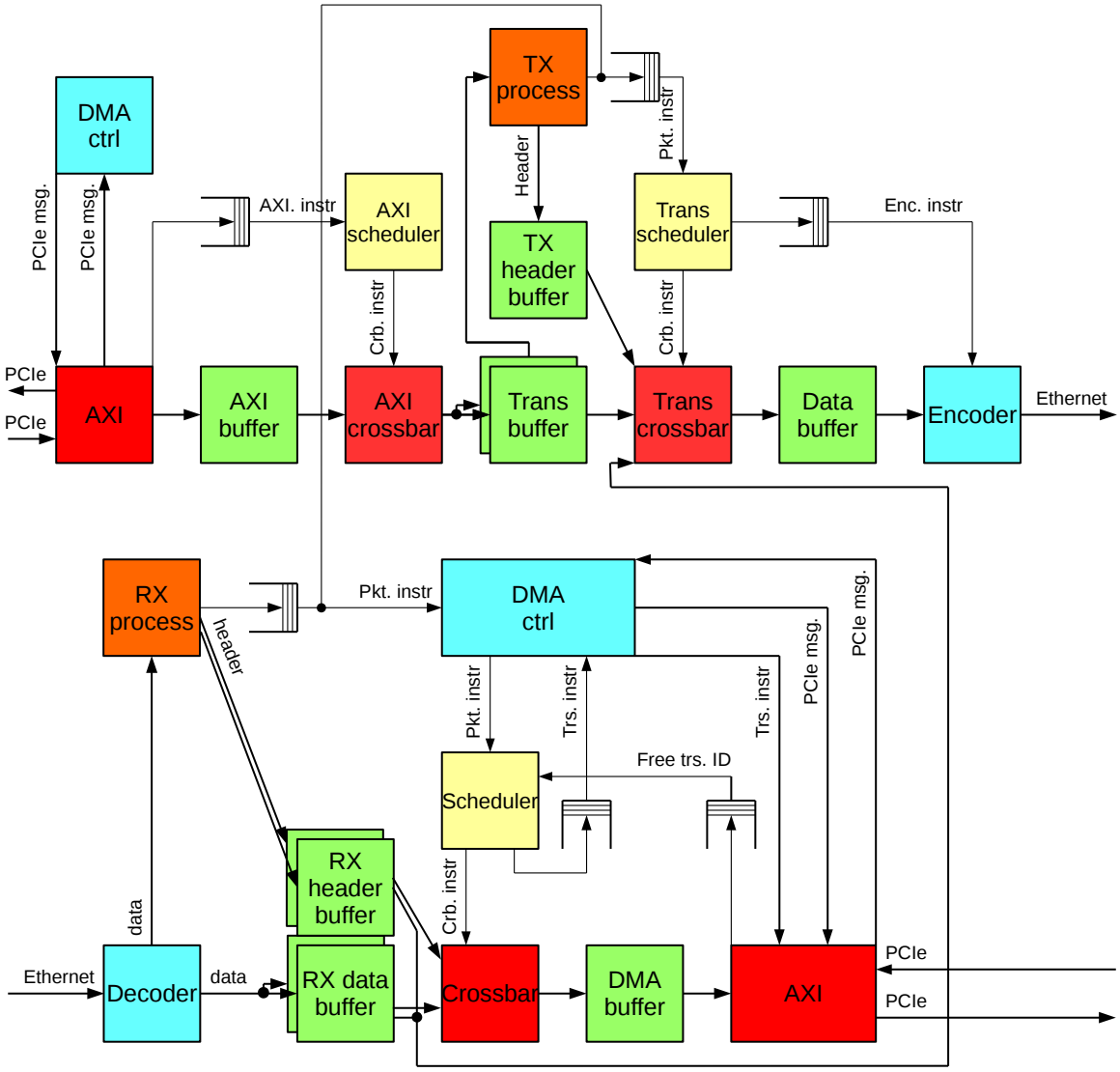

Figure 5.6: Block diagram of proposed Framework of multiple RX and multiple TX data paths

Chapter 6

Results Summary

In this chapter are discussed results of this master's thesis. In the previous chapter 5 are proposed multiple frameworks. The emphasis of this master's thesis is put on communication between an acceleration card and the host computer via PCI Express. Therefore, `Scheduler` for single RX data path was chosen to be implemented and its results evaluated. In the previous chapter 5 was shown that `Scheduler` is a controller of the `BUFFER->CROSSBAR->BUFFER` data path. This approach is used in all proposed frameworks in order to save resources, especially on-chip memory.

`Scheduler` for single RX data path was implemented and verified in simulations according to the design 5.1.1. A number of processed network packets per clock cycle is a generic parameter (PPC – Packets Per Clock). Synthesis results are summed up in Tables 6.1 (1 PPC, 512 b data width), 6.2 (2 PPC, 1024 b data width), 6.3 (4 PPC, 2048 b data width), and 6.4 (8 PPC, 4096 b data width) for Xilinx UltraScale+ (xcvu9p-flgb2104-2-i).

The implementation tries to address all disadvantages of the NetCOPE-100G framework (see 4.1). Disadvantages are missing support to process multiple packets per clock cycle, linear space complexity ($\mathcal{O}(n)$) of on-chip memory with respect to a number of DMA channels and resource demanding processing pipeline.

In the graph (Figure 6.1), is compared tge throughput of 2048 b data width processing pipeline with respect to a number of processed network packets per clock cycle. The NetCOPE-100G framework is able to process just one network packet per clock cyle. Its throughput is limited by 100 Gbps for short network packets despite using a large data width of the processing pipeline. On the other hand, the proposed solution may process multiple network packets per clock cycle. Therefore, its throughput is independent on network packet size. Unfortunately, short packets are common in computer network traffic. The throughput of the proposed solution goes far beyond 400 Gbps. In fact, throughput up to 800 Gbps seems to be feasible with the current technology.

The results are visualized in three graphs (Figure 6.2), (Figure 6.3), and (Figure 6.4). In the graphs are added results of the NetCOPE-100G framework for comparison despite the results are not completely comparable due to architectural differences.

In the first graph (Figure 6.2), is shown resources (LUTs and registers) consumption dependency on a number of DMA channels of `Scheduler` for single RX data path. Resources grow linearly (space complexity is $\mathcal{O}(n)$), same as in the current solution. Due to architectural differences, the results may not be completely compared with the NetCOPE-100G framework.

In the second graph (Figure 6.3), is shown BRAMs consumption dependency on a number of DMA channels of `Scheduler` for single RX data path. BRAMs consumption

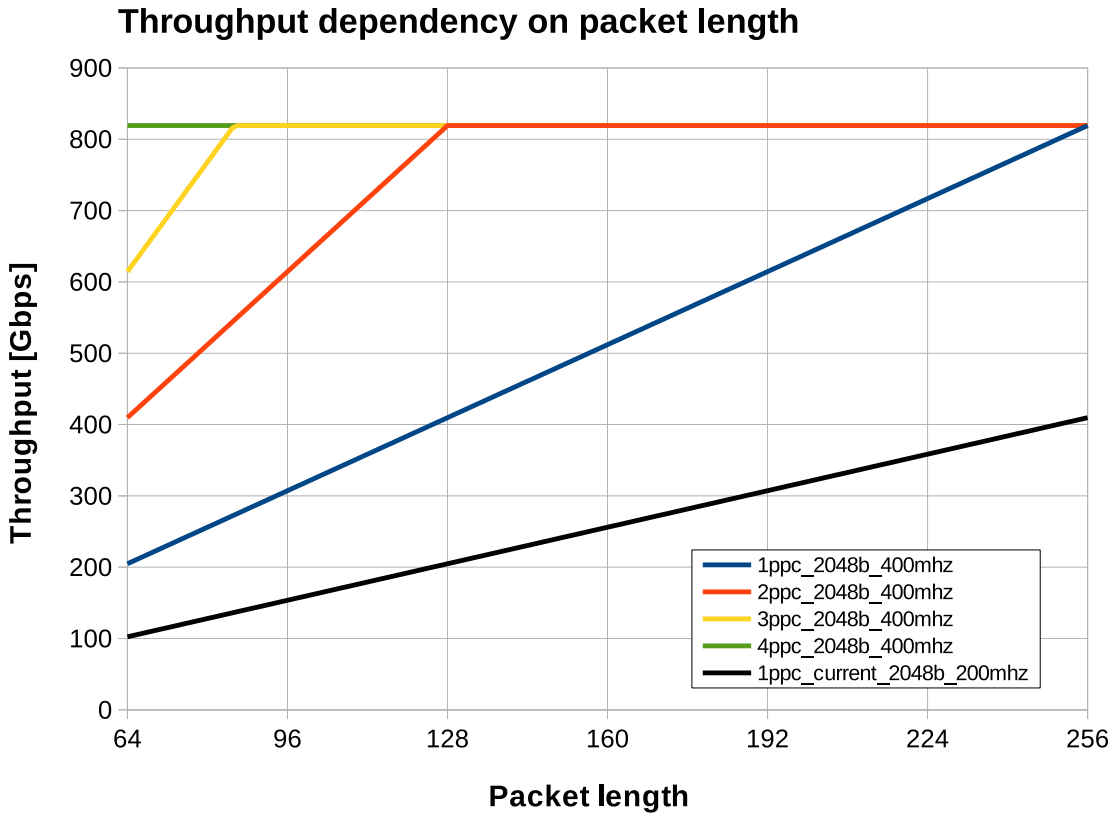

Figure 6.1: Throughput dependency (PPC – Packets Per Clock) on packet length of `Scheduler` for single RX data path

RX DMA channels	LUTs	Regs	BRAMs	Frequency [MHz]
2	4060	2449	8	561
4	4175	2599	8	573
8	4485	2891	8	575
16	4961	3451	8	560
32	5964	4517	8	560
64	7988	6664	8	560
128	12457	11022	8	572
256	21517	19598	8	543

Table 6.1: Synthesis results of `Scheduler` for single RX 512 b width data path for Xilinx UltraScale+

grows constantly (space complexity is $\mathcal{O}(1)$). This is a significant improvement over the NetCOPE-100G framework. The NetCOPE-100G framework has a linear space complexity of BRAM ($\mathcal{O}(n)$). In case of 256 DMA channels and 512 b width data bus, the NetCOPE-100G framework consumes 2048 BRAMs. The proposed solution consumes just 8 BRAMs – 256 times less. The explanation behind the massive improvement is using a shared buffer

RX DMA channels	LUTs	Regs	BRAMs	Frequency [MHz]
2	12720	8138	16	504
4	12895	8308	16	504
8	13480	8630	16	463
16	14185	9202	16	462
32	15471	10313	16	472
64	18248	12517	16	472
128	23027	16903	16	472
256	32936	25806	16	472

Table 6.2: Synthesis results of `Scheduler` for single RX 1024 b width data path for Xilinx UltraScale+

RX DMA channels	LUTs	Regs	BRAMs	Frequency [MHz]
2	42071	19858	32	364
4	43342	20051	32	406
8	43403	20372	32	432
16	44800	21110	32	369
32	46365	22116	32	432
64	50244	24339	32	406
128	55832	28709	32	390
256	71647	37854	32	428

Table 6.3: Synthesis results of `Scheduler` for single RX 2048 b width data path for Xilinx UltraScale+

RX DMA channels	LUTs	Regs	BRAMs	Frequency [MHz]
2	181765	57121	64	316
4	181188	56999	64	300
8	185022	57722	64	312
16	187099	58372	64	312
32	189315	59633	64	329
64	201961	62005	64	321
128	231755	66662	64	306
256	284016	75953	64	310

Table 6.4: Synthesis results of `Scheduler` for single RX 4096 b width data path for Xilinx UltraScale+

for all DMA channels instead of a separate buffer for each DMA channel (compare block diagrams in the Figures 4.5 and 4.6).

In the third graph (Figure 6.4), is shown frequency dependency on a number of DMA channels of `Scheduler` for single RX data path. The proposed solution has a constant frequency for all data bus widths. In comparison, frequency in the NetCOPE-100G framework decreases by the multiplicative inverse of the logarithm. The proposed solution may be clocked at over 400 MHz even for 2048 b wide data bus. In this configuration, the throughput is over 800 Gbps.

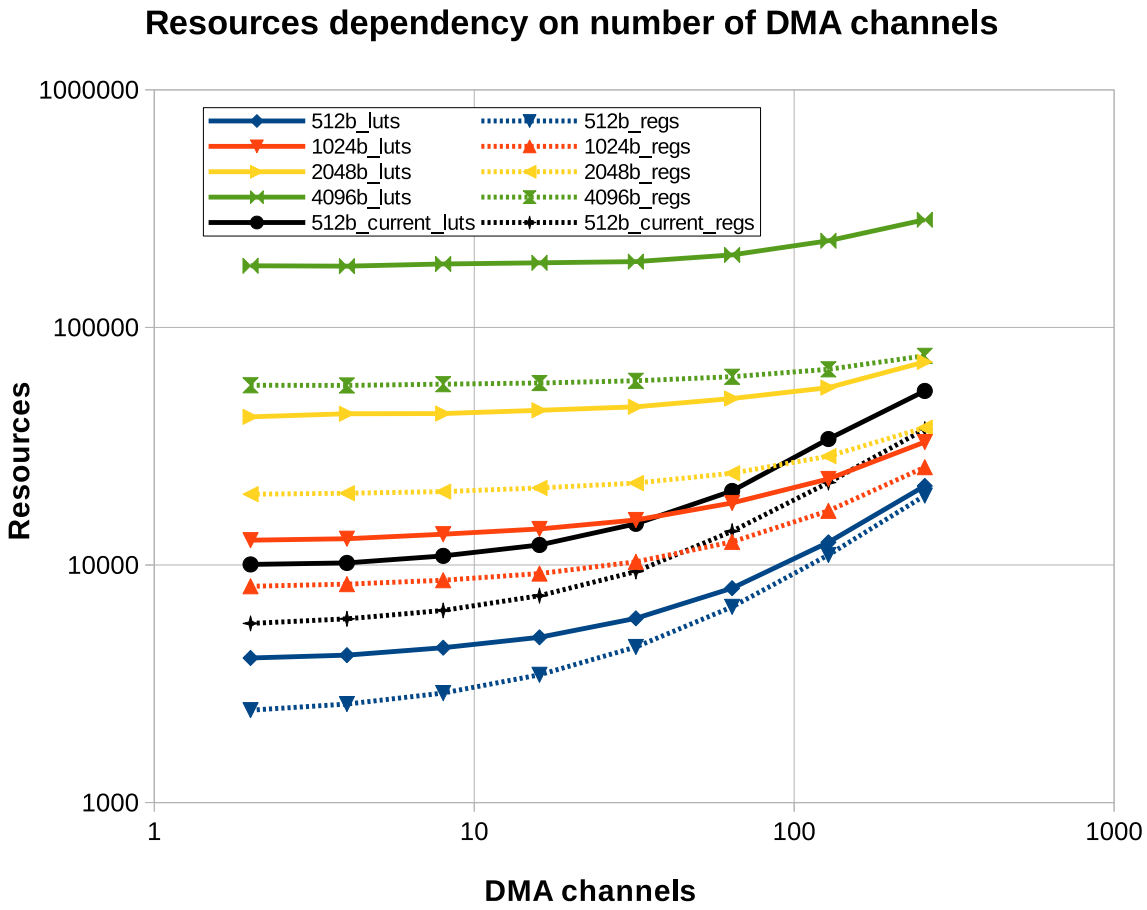

Figure 6.2: Resources dependency on number of DMA channels of `Scheduler` for single RX data path

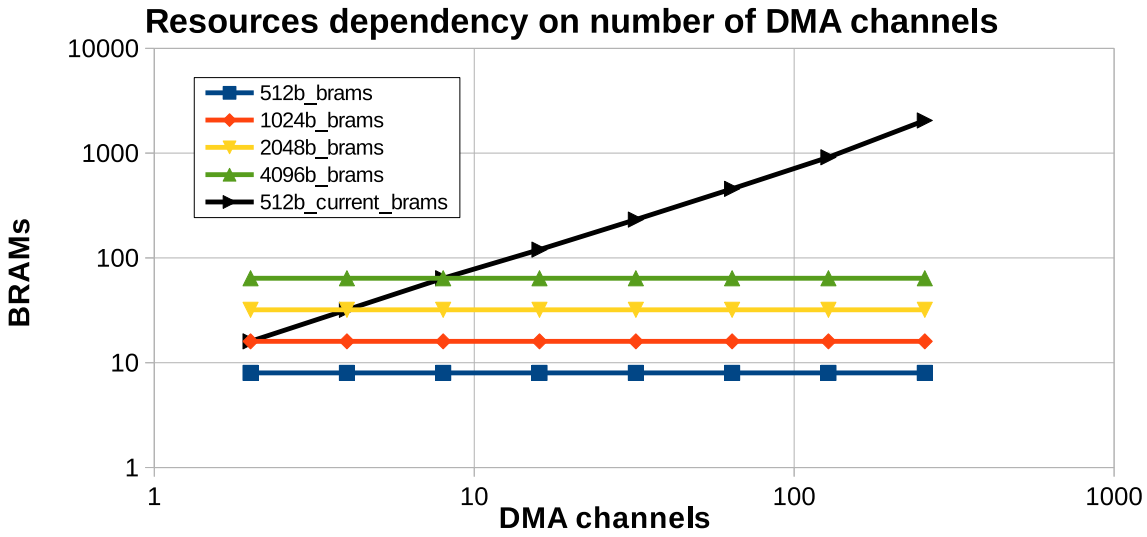

Figure 6.3: BRAMs dependency on number of DMA channels of `Scheduler` for single RX data path

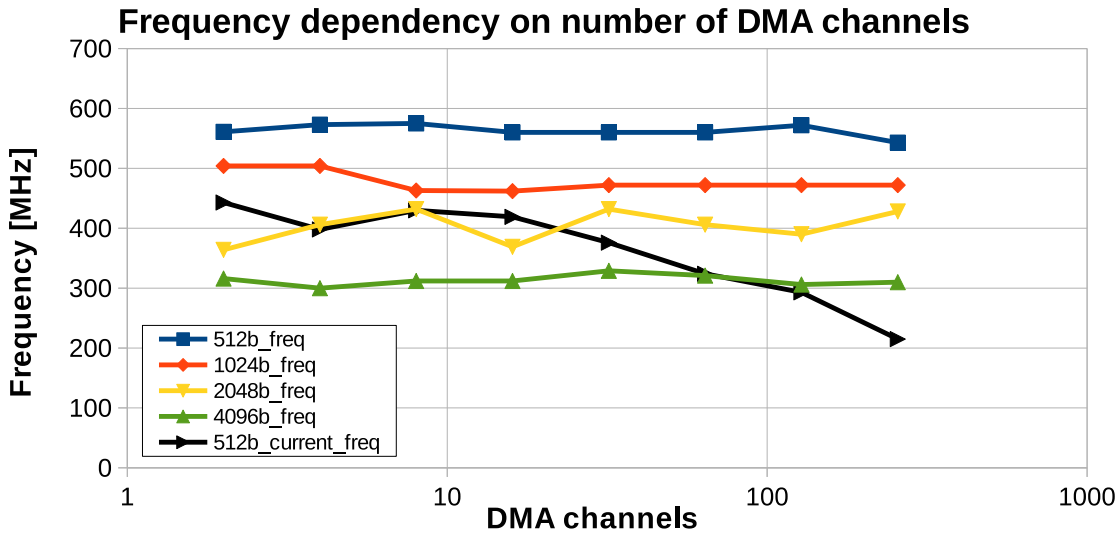

Figure 6.4: Frequency dependency on number of DMA channels of `Scheduler` for single RX data path – Xilinx Ultrascale+

Chapter 7

Conclusion

The goal of this thesis is to create a framework for hardware acceleration of 400 Gb networks. At the beginning, theoretical essentials about high-speed networks, Ethernet, PCI Express expansion bus, and Data Plane Development Kit (DPDK) were studied. Results of these studies, including technology consideration with respect to 400 Gb networks, are summed up in the chapter Theoretical background 2. Despite high demands, the specification of 400 Gigabit Ethernet is only a draft waiting for ratification.

Time-critical operations in computer networks were studied together with hardware architecture to speed up network traffic processing. These time critical operations and system DPDK is described in chapter 2. The analysis is also provided how to support multiple independent DMA channels in DPDK. Moreover, the NetCOPE framework for rapid development of hardware accelerated network applications for 100 Gigabit Ethernet is described in detail. All these studies are summed up in the chapter Related Work 3.

System analysis was performed of the current framework for hardware acceleration of 100 Gb networks in the chapter 4. There were found three major obstacles to deploying the current framework for 400 Gb networks and beyond. The first obstacle is missing support to process multiple packets per clock cycle. Therefore, the current NetCOPE-100G framework with 200MHz frequency can not exceed 100 Gbps throughput. The second obstacle is the limited support of multiple independent DMA channels. It has become a problem in recent years because modern-day CPUs provide many cores. More completely independent DMA channels are needed to fully utilize these CPUs. The third obstacle is resource demanding processing pipeline.

The new architecture of the framework for hardware acceleration of 400 Gb networks and beyond was introduced to address all three mentioned obstacles of the current framework. The architecture contains paths for packet reception and transmit. Multiple Ethernet ports were also considered. Emphasis was put on high-speed transfers between acceleration card and the host computer via PCI Express bus. The proposed architecture is capable of processing multiple network packets per clock cycle. Therefore, it enables to process 400 Gb networks and beyond. Moreover, the architecture has a constant space complexity $\mathcal{O}(1)$ of on-chip FPGA memory with respect to the number of DMA channels. A lot of effort was made to lower the resource consumption as much as possible.

Critical parts of the framework were implemented and its function verified in simulations. The design is capable running at very high frequency exceeding 400 MHz for FPGA Xilinx UltraScale+. The frequency is not decreased with the number of DMA channels. Therefore, the proposed architecture is capable of processing 400 Gb networks using just 1024 b wide data bus and 800 Gb networks using 2048 b.

The future work will be focused on functional verification in SystemVerilog of the proposed framework for hardware acceleration of 400 Gb networks and its subsequent deployment on an acceleration card. Furthermore, possibilities will be analyzed how to simplify design and implementation of application processing with P4 language.

Bibliography

[1] Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model. International Standard ISO/IEC 7498-1. 1996.

[2] *PCI Express® Base Specification*. PCI–SIG. revision 3.0 edition. 2010.

[3] CESNET and INVEA-TECH demonstrate 100 Gbps transfers over PCIe with a single FPGA. CESNET. 2014. [Online; visited 5.1.2017].
Retrieved from: https://www.cesnet.cz/cesnet/reports/press-releases/100-gbps-over-pcie-with-single-fpga/?lang=en

[4] Cisco Visual Networking Index: Forecast and Methodology, 2015-2020. Cisco Systems, Inc. 2015. [Online; visited 19.12.2016].
Retrieved from:
http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

[5] Cisco Global Cloud Index Supplement: Cloud Readiness regional Details. Cisco Systems, Inc. 2016. [Online; visited 19.12.2016].
Retrieved from: https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738089.pdf

[6] DPDK Data Plane Development Kit: Programmer's Guide. Release 16.11.0. November 2016.
Retrieved from: http://fast.dpdk.org/doc/pdf-guides/prog_guide-16.11.pdf

[7] Network Functions Virtualization. ETSI. 2016. [Online; visited 7.1.2017].
Retrieved from: http://www.etsi.org/images/files/ETSITechnologyLeaflets/NetworkFunctionsVirtualization.pdf

[8] Open vSwitch. A Linux Foundation Collaborative Project. 2016. [Online; visited 19.12.2016].
Retrieved from: http://openvswitch.org/

[9] Cards | Liberouter / Cesnet TMC group. 2017. [Online; visited 5.1.2017].
Retrieved from: https://www.liberouter.org/technologies/cards/

[10] DPDK Data Plane Development Kit. 2017. [Online; visited 8.1.2017].
Retrieved from: http://dpdk.org/

[11] Liberouter / Cesnet TMC group | Programmable hardware | Liberouter / Cesnet TMC group. 2017. [Online; visited 5.1.2017].
Retrieved from: https://www.liberouter.org/

[12] NetFPGA. 2017. [Online; visited 28.4.2017].
Retrieved from: http://netfpga.org/site/#/

[13] Technology - Accelerated Network Technologies. Accelerated Network Technologies at Faculty of Information Technology, Brno University of Technology. 2017. [Online; visited 24.4.2017].
Retrieved from: http://merlin.fit.vutbr.cz/ant/technology/index.html

[14] Bosshart, P.; Daly, D.; Gibb, G.; et al.: P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*. vol. 44, no. 3. July 2014: pp. 87–95. ISSN 0146-4833. doi:10.1145/2656877.2656890.
Retrieved from: http://doi.acm.org/10.1145/2656877.2656890

[15] Braden, R.: Requirements for Internet Hosts – Communication Layers. RFC 1122. IETF. October 1989.
Retrieved from: https://tools.ietf.org/html/rfc1122

[16] Brebner, G.: The Role of the FPGA in 400GbE Technology Development. Ethernet Alliance. 2013. [Online; visited 6.1.2017].
Retrieved from: http://www.ethernetalliance.org/wp-content/uploads/2012/12/OFC-Ethernet-Alliance-talk-Brebner.pdf

[17] D'Ambrosia, J.: IEEE P802.3bs Baseline Summary. IEEE. July 2015.
Retrieved from: http://www.ieee802.org/3/bs/baseline_3bs_0715.pdf

[18] Gupta, P.; McKeown, N.: Algorithms for Packet Classification. Computer Systems Laboratory, Stanford University. [Online; visited 8.1.2017].
Retrieved from:
http://yuba.stanford.edu/~nickm/papers/classification_tutorial_01.pdf

[19] Lawley, J.: Understanding Performance of PCI Express Systems. WP 350. Xilinx. October 2014.
Retrieved from:
https://www.xilinx.com/support/documentation/white_papers/wp350.pdf

[20] Martínek, T.; Košek, M.: NetCOPE:Platform for Rapid Development of Network Applications. In *11 th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. DDECS 2008. 2008. pp. 1–6.

[21] Martínek, T.; Žádník, M.: Precise Timestamp Generation Module and its Applications in Flow Monitoring. Technical Report 13. CESNET. 2009.

[22] McKeown, N.: The iSLIP Scheduling Algorithm for Input-Queued Switches. In *IEEE/ACM TRANSACTIONS ON NETWORKING*, vol. 7. IEEE. April 1999.

[23] Puš, V.; Kekely, L.; Špinler, M.; et al.: HANIC 100G: Hardware accelerator for 100 Gbps network traffic monitoring. Technical Report 2. CESNET. 2014.

[24] Scheidt, P.: Flexible 400 Gigabit Ethernet PCS. Altera. 2014. [Online; visited 6.1.2017].
Retrieved from: http://www.ethernetsummit.com/English/Collaterals/Proceedings/2014/20140430_1C_Scheidt.pdf

[25] Society, I. C.: *IEEE Standard for Ethernet*. IEEE. std 802.3-2012 edition. 2012.

[26] Stauffer, D. R.; Anderson, S. D.; Sanders, A.; et al.: Comparison of PAM-4 and NRZ Signaling. IBM. March 2014.
Retrieved from:
http://www.ieee802.org/3/bladesg/public/mar04/anderson_01_0304.pdf

[27] Zilberman, N.; Audzevich, Y.; Covington, G. A.; et al.: NetFPGA SUME: Toward Research Commodity 100Gb/s. University of Cambridge and Stanford University. 2014.
Retrieved from:
http://www.cl.cam.ac.uk/~awm22/publications/zilberman2014sume.pdf

Appendix A

Contents of the enclosed memory media

- /doc – electronic version of this document
- /src – source code of the critical parts of the proposed framework
- /tex – source code of this document