BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

High-speed DMA packet transfer in system DPDK

Jan Kubálek

Abstract
Devices in computer networks, which are used for network management, require a high-speed processing of large amounts of data for analysis. For a device to enable the monitoring of a network with high data traffic, its network interface card needs to be capable of transferring received data to RAM at sufficient speed. My project deals with the design, implementation, and testing of a new module for an FPGA chip on a network interface card, which will carry out these transfers. The design aims to achieve a high throughput of up to 200 Gb/s for the transfer of packets from the FPGA chip to a computer memory via a PCI-Express bus. For faster packet processing, in software system DPDK is used for data transfer control. This paper contains a short introduction to technologies used in the project and the summary of the resulting module design. Performance testing has shown that the module can achieve the target throughput of 200 Gb/s, but also revealed possible ways for further improvements.

Keywords: DMA — DPDK — FPGA — PCI-Express

Supplementary Material: *N/A*

*xkubal11@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The Internet requires the maintenance of large number of servers and hubs that allow the connection of computers throughout the world. For correct network load control and failure detection, special devices are needed, that constantly monitor the subsections of the Internet network. Such devices are required to calculate statistics about transferring hundreds of gigabits of data per second. Network interface cards (NICs) used in common personal computers are not capable of receiving such huge amounts of data.

One of possible existing solutions is CESNET's system SZE2 (*Straight ZEro copy*) for NIC-to-RAM data transfers. This system involves support in software provided by NIC drivers and support in hardware provided by CESNET's NetCOPE platform. Together, the system enables receiving network packets at high speed to be processed by a software application. In SZE2, data packets are delivered one after another into a single block of memory space. This, however, makes reading the received packets slow from the side of the software. Open source system DPDK (*Data Plane Developement Kit*) takes a different approach to these transfers. Each packet is stored in separated space making the much easier to process afterwards. The NetCOPE platform does not yet support system DPDK, as this system requires specialized module in hardware design.

The project aims to design a module as part of an architecture for an FPGA (*Field Programmable Gateway Array*) chip placed on a NIC, that will control DMA (*Direct Memory Access*) transfers of data in system DPDK from the FPGA to the computer's RAM. This DMA module will communicate with a software

application, which will allow programs running on the computer to receive data from the network. The implemented DMA module has to be able to manage all of the required functions at the speed of 100 Gb/s and more while being limited by resources availble on the FPGA chip.

The the target of the implementation is the Virtex UltraScale+ VU7P FPGA from Xilinx. Located on the NFB-200G2QL NIC by Netcope Technologies, this FPGA controls all major operations done by the card. The card itself is fitted with interfaces connecting it to the host computer and to the network. These interfaces allow transferring data at the theoretical speed of up to 200 Gb/s, making it the target throughput for the DMA module.

The implemented DMA module is able to manage DMA transfers of packets with data throughput reaching to 200 Gb/s. Thanks to the support of system DPDK, a software application working with the transferred data can carry out complex operations over the data without the risk of lowering the throughput.

2. DMA

For network data inside a network interface card to be received by a software application running on a computer's CPU, it has to be stored in the computer's RAM. This is done by transferring the data from the NIC, which is connected as a peripheral device of the computer, to the RAM using DMA. With the DMA technology large amounts of data can be moved between any peripheral device and a memory without the need for intervention from a CPU. An application that seeks to receive the data only has to reserve a sufficient area in a memory space and tell the NIC to transfer the data to this area. When the transmission is completed, the application is informed and can use the received data.

In this project, the NIC, RAM and CPU are connected via a data bus PCI-Express Gen3 x16 (PCIe, *Peripheral Component Interconnect Express*), which is capable of transmitting data at the speed of 100 Gb/s. The NFB-200G2QL NIC is fitted with two PCIe ports of this kind, permitting the throughput of up to 200 Gb/s.

3. DPDK

Even though the technology of DMA transfers allows the NIC to write the received data to the RAM autonomously, the transfer still needs to be in the control of a software application. The DMA module needs to know where in the memory it can write the received packet data and, in return, the application needs to be informed when new packets have been received. The system of packet transfer control chosen for this project is system DPDK. While other possible approaches exist — most notably SZE2 — DPDK allows to partially accelerate packet processing in hardware by storing each packet into separate space in a memory, which makes them easy (and fast) to read by software. Also, as opposed to the SZE2, system DPDK is widely used open source technology.

For a software application that works with network data, DPDK comes with a set of open source libraries. These libraries allow the DPDK application to receive and send data by communicating with the NIC's driver. For more information on integration of DPDK in software applications see [1].

The sequence of actions needed for receiving new data in system DPDK is as follows:

1. The application reserves several separate areas within memory, to which packets will be received. Each of these areas is described by a "descriptor" containing its address and size in RAM.
2. The application writes the descriptors to a reserved place in the memory, which is refered to as a "descriptor buffer".
3. The application starts the DMA module and informs it where is the descriptor buffer and how many descriptors are currently in it.
4. The DMA module reads multiple avalible descriptors from the buffer and waits for incoming data.
5. Upon receiving new packets on its data input, the DMA module starts sending them one by one to the memory space described by the descriptors.
6. After a certain timeout, the DMA module informs the application about the number of descriptors that have been used.
7. The application reads the received packets and allows the DMA module to reuse the corresponding descriptors.

In DPDK, descriptors must be used in the same order in which they are written in the descriptor buffer. Each area of a descriptor can contain the data of only one packet (hence packet separation). If a packet does not fit in one descriptor's area, it can use multiple consecutive descriptors. The DPDK application can determine how many descriptors have been used for a packet by reading its length, which is stored at its beginning. The descriptor buffer in DPDK works as a ring buffer and the usage of descriptors stored in it is controlled by a pair of pointers: a write pointer from the software and a read pointer from the hardware.

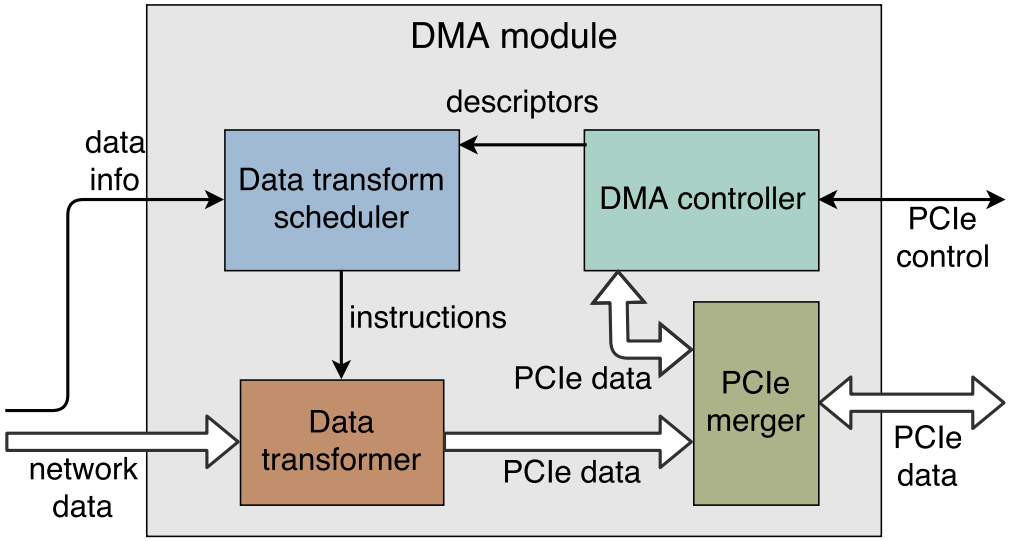

Figure 1. The diagram of the designed DMA module.

4. DMA channels

The DPDK transfer can be divided into multiple parallel transfers, which are controlled from parallel processes of an application to ease the workload per CPU core. The separation of packets to individual channels is defined by a distribution module placed in the receiving part of the FPGA architecture before the DMA module. Because of this division, the DMA module is required to keep track of multiple DPDK transfers. Each of them has its own descriptor buffer, SW pointer and HW pointer and each channel can be started and stopped separately by the application. Packets belonging to a stopped channel are discarded by the DMA module and not sent to the RAM.

For the DMA module design, this separation can possibly mean difficulties, since some parts of the architecture grow in complexity with higher number of supported DMA channels. The DMA module should support at least 128 DMA channels.

5. DMA module design

A high bit width of data processed in the FPGA architecture implies high resource requirements even for operations as simple as moving the data from one place to another. Therefore, the key idea behind the DMA module's design is to reduce the operations performed on the large data to the minimum and to make all complex decisions based on the provided meta-data only. While the data itself is being stored in one place in the module.

Figure 1 shows a simplified diagram of the designed DMA module. When receiving network data from the previous component in the FPGA architecture, the DMA module also acquires information describing the individual data packets. For the DMA module, the most important information are the length of the packet and the assigned DMA channel. These are used to determine whether the packet can be sent to the RAM and what descriptors will be used to store it. All of this is done in the "Data transform scheduler", which prepares a set of instructions for the "Data transformer". Based on the instructions, the Data transformer converts the input data to a form, in which it will be sent to PCIe as individual data transactions. After that, the data are organized into PCIe write requests and sent towards the PCIe bus by the "PCIe merger".

Apart from sending data, the DMA module also needs to be capable of communicating with the CPU and reading descriptors from the RAM. These functions are implemented by the "DMA controller". From the perspective of the DMA module, the PCIe bus interface is divided into two parts. One is devoted to communication with the RAM and the other ("PCIe control" in Figure 1) connects the module to the CPU. When being controlled from the software, the DMA controller receives write and read requests from the PCIe control interface. For reading new descriptors from the memory, the DMA controller uses the PCIe data interface and sends read descriptors directly to the Data transform scheduler. As the DMA module contains one PCIe data interface only, read transactions from the DMA controller have to be merged with data transactions in the PCIe merger.

The most complicated operations are implemented in the Data transform scheduler and the DMA controller. The other two components consist mostly of data buffers and queues.

6. DMA module implementation

The DMA module is primarily aimed to be used in the FPGA placed on the NIC model NFB-200G2QL vended by Netcope Technologies (Figure 2). The specific FPGA utilized on this NIC is Virtex UltraScale+ VU7P from Xlilinx. The resources and advanced technology of the chip enable the implementation of large and complex designs running at high clock frequency. For more information on Virtex UltraScale+ design see the FPGA documentation [2, 3, 4].

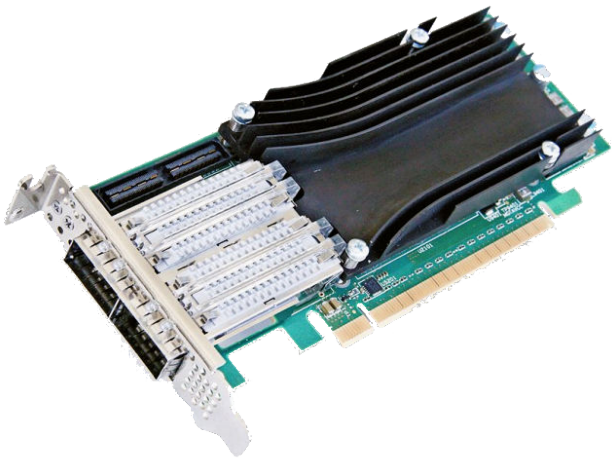

Figure 2. Network interface card NFB-200G2QL Source: [5]

DMA channels	LUT ([%])	BRAM ([%])	Frequency [MHz]
8	10 524 (1)	57.5 (4)	329
16	12 029 (2)	73.5 (5)	329
32	15 188 (2)	105.5 (7)	336
64	20 644 (3)	169.5 (12)	329
128	33 463 (4)	297.5 (21)	333
256	59 930 (8)	553.5 (38)	326

Table 1. Resources utilized by the DMA module for a various number of channels. The numbers are reported after synthesis for FPGA Virtex UltraScale+ VU7P

As the target platform is an FPGA chip, the implementation language chosen for this project is VHLD. VHDL is a standardized [6] language for hardware design description, which is supported by all hardware simulation and synthesis tools.

Target frequency for the DMA module is 200 MHz and the width of the data input is 512 b. This way the DMA module is able to receive 102.4 Gb of data every second. To enable transmissions at the speed of up to 200 Gb/s, as allowed by the NIC's interfaces, the FPGA architecture can be fitted with two DMA modules, each operating independently and utilizing one PCIe bus.

Another possible way of reaching 200 Gb/s would be increasing the input data width to 1024 b. However, this would greatly increase the module complexity, making it harder to reach the desired frequency. This option may be considered later, when designing the module for 400 Gb/s speed.

Table 1 presents resources utilization by the DMA module after synthesis for Virtex UltraScale+ VU7P, as well as achieved frequency, for a various number of supported DMA channels. The resources are represented by the number of used LUTs (*Look-Up Tables*) and the number of used BRAMs (*Block RAMs*). The table also contains relative resource utilization of the selected FPGA (in percents) and lastly the maximum possible operating frequency for each of the configurations.

As can be seen from the table, the module can operate on the target frequency of 200 MHz on any of the tested configurations. BRAM usage can, however, be critical, when using two DMA modules combined together with the rest of the NIC architecture, if supporting 256 DMA channels.

7. Throughput measurement

To acquire relevant information on the throughput of the DMA module, a complete architecture for NIC's FPGA have been implemented and the NIC has been connected to receive artificially generated packets. Only one DMA module, one Ethernet port, and one PCIe bus port were used for testing, making 100 Gb/s the maximum achievable throughput.

A DPDK application, that provided the DMA module enough descriptors to write data to was running on the host computer. For the transfer not to be slowed down by the software, the application was running on 16 DMA channels and the data load was evenly distributed among them.

To calculate actual throughput, there were counters of arriving packets, received packets and discarded packets placed on the very input to the FPGA. Once the DMA module would not be able to accept incoming data every clock cycle, the input interface would start discarding. Another set of counters for discarded packets was placed in the DMA module, since the DMA module can discard packets as well. All of these counters could be read from software to acquire values needed for determining achieved throughput.

$$T = S * \frac{(P_{Iarr} - P_{Idis} - P_{DMAdis})}{P_{Iarr}} \qquad (1)$$

Throughput T over a specific time period was com-

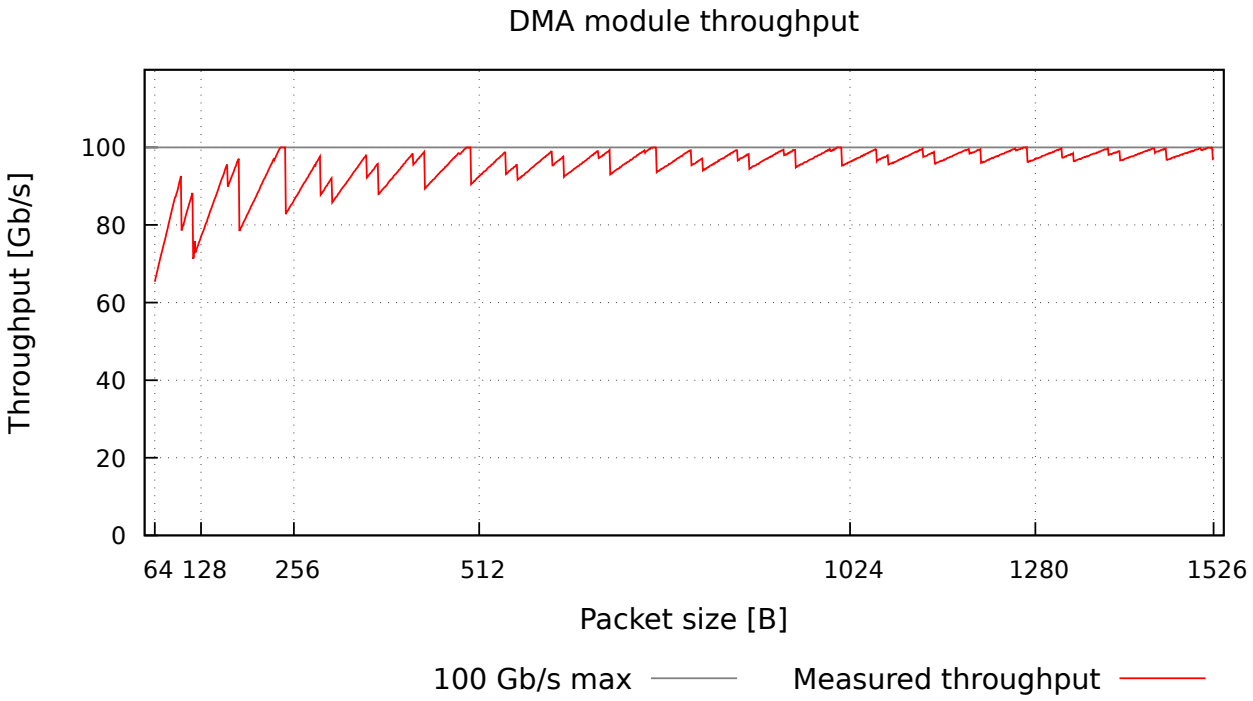

Figure 3. Graph of measured throughput of DMA module on packet lengths 64 to 1526 B

puted using Equation 1. Here P_{Iarr} is the number of packets arriving to the input, P_{Idis} is the number of packets discarded on the input and P_{DMAdis} is the number of packets discarded in the DMA module. Constant S is a data load incoming from the Ethernet interface and as such it is the reference maximum speed (100 Gb/s in this case).

The throughput of the DMA module was measured for each packet size from 64 B to 1526 B. The results of this measurement are shown in Figure 3 in comparison to the maximum throughput of 100 Gb/s.

You can see that the DMA module's throughput approximates the maximum value with periodical drops every 64 B, the first of them being at the length 117 B. After arriving from the Ethernet port, every packet is stripped of 4 bytes of CRC and extended by 16 bytes of a header added by the distribution architecture. This way, the packet is 12 B larger on the DMA module's input, turning a 117-byte packet to 129-byte one. Because the DMA module sends data to the RAM over PCIe in one or more write transactions and each transaction has the size of 64 B, the last transaction of every packet contains less than 64 B (unless the packet's size is aligned to this size). This causes the reduction of throughput, as the DMA module is currently able to send only one transaction per clock cycle. Considering 117-byte packets, every third data transaction sent on the PCIe bus contains only 1 B of data ($117+12=129=64+64+1$). This lowers the actual throughput to mere 68.8 Gb/s. The best way to eliminate these drops would be to modify the DMA module to produce more smaller transactions in one clock cycle.

With these results, it can be said that the designed DMA module is able to achieve the maximum throughput of 100 Gb/s on some specific packet lengths. After some modifications for improving its effectiveness, it should be able to reach the throughput of 200 Gb/s by using two DMA modules at once.

8. Conclusions

The subject of this paper are the technologies used for managing DMA transfers of received network data in system DPDK, the description of the design of a new DMA module, which can carry out these transfers, the summary of resources used by the implemented DMA module, and, the presentation of data throughput results achieved by this DMA module. Thanks to DPDK support, a software application can achieve much higher speed when processing the reveived packets as opposed to previously used system SZE2.

This new DMA module was designed to operate on a high number of parallel DMA channels, allowing the division of the data load among up to 256 CPU cores while utilizing less then 10% of resources on the used FPGA Virtex UltraScale+ VU7P.

The implemented module was able to reach the desired throughput on only some of the selected packet

lengths. However, after implementing already prepared adjustments its performance should improve, which will allow it to transfer all received network packets to a DPDK application at the maximum speed of 200 Gb/s. The DMA module was designed with regards to flexibility and will be further developed to reach data transfers on 400 Gb/s.

Acknowledgements

I would like to thank my supervisors Ing. Jan Kořenek, Ph.D., and Ing. Jiří Matoušek for their help with working on this project and preparing this paper. I would also like to thank my colleagues from CESNET Ing. Martin Špinler, Ing. Václav Hummel, and Ing. Viktor Puš, Ph.D., for their assistance with the DMA module design and implementation. This research has been supported by the Technology Agency of the Czech Republic from project TH02010214.

References

[1] Data plane development kit documentation, 2017. https://dpdk.org/doc.

[2] Ultrascale architecture memory resources, 2017. https://www.xilinx.com/support/documentation/user_guides/ug573-ultrascale-memory-resources.pdf.

[3] Ultrascale architecture configurable logic block, 2017. https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf.

[4] Ultrascale plus fpga product tables and product selection guide, 2017. https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf.

[5] Meet nfb-200g2ql, new 200g programmable smart nic, 2017. www.netcope.com/en/company/press-center/press-releases/meet-nfb-200g2ql,-new-200g-programmable-smart-nic.

[6] Ieee standard vhdl language reference manual. *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pages c1–626, Jan 2009.