RingNet: A Memory-Oriented Network-On-Chip Designed for FPGA

Jakub Siast, Adam Łuczak, and Marek Domański, *Member, IEEE*

***Abstract*— In this paper, we identify the general requirements for network-on-chips (NoCs) and the general characteristics of field-programmable gate arrays (FPGAs) from leading producers. Based on the analysis provided, an FPGA-oriented NoC called RingNet is proposed. As a distinctive feature, RingNet uses communication through a centrally placed memory that aims at preventing network congestions and limiting the network buffer requirements. Optimal utilization of FPGA resources is one of the goals of RingNet development. Especially buffers are implemented in distributed RAM available in FPGAs, and the virtual cut-through is used as an efficient switching technique for FPGA. Simulations prove guaranteed throughput, predictable latency, and fair network access provided by RingNet. Synthesis results for sample FPGAs from Xilinx, Intel, and Lattice prove the universality of RingNet. The provided analysis of NoC implementations leads to the conclusion that RingNet needs fewer resources and supports higher clock frequencies than the widely used AXI4 architecture.**

***Index Terms*— Distributed memory, lookup table RAM (LUTRAM), fairness, field-programmable gate array (FPGA), network-on-chip (NoC), virtual cut-through.**

I. Introduction

NETWORK-ON-CHIP (NoC) is a widely adopted solution for the interconnect problem in large systems-on-chip (SoCs). Hitherto, much research on NoCs has been related to application-specified integrated circuits (ASICs) [1] but the rapidly growing field-programmable gate array (FPGA) size also yields growing interest in NoCs implemented in complex FPGAs. Using NoCs as interconnections for FPGAs is not a new idea [2]–[4]. Although FPGAs differ from their ASIC counterparts, most of the known FPGA NoCs were adopted from ASICs without considering FPGA-specific features. Therefore, the potential of NoCs has not been fully exploited for FPGAs. In this situation, older interconnecting techniques like crossbars (e.g., AXI4 Interconnect) are still in use, regardless of their poor scalability [3]. The development of new NoC architectures, better suited to FPGA, is still a challenging problem.

In this paper, we propose a novel NoC architecture called RingNet that is well-suited to the features of contemporary FPGAs. Among other NoC architectures proposed for FPGAs, RingNet stands out with communication through a central memory and traffic load controlled by the recipient. This paper starts with a discussion on the common features of FPGAs that are important for NoC architectures (Section II). In Section III, general requirements for NoCs are discussed. Next, in Section IV, we summarize the state of the art in NoC designs for FPGA and point out NoC design constraints that match the FPGA features considered in Section III. In Section V, we propose the idea of a novel RingNet NoC with its protocol description provided in Section VI. Through Sections VII–IX, we present experimental results, compare RingNet synthesis for different FPGAs, and compare RingNet implementation with the widely used crossbar interconnection called AXI4 Interconnect.

II. Characteristics of FPGAs

In this paper, the basic common features of modern FPGAs are studied in the context of products offered by three of the leading FPGA vendors: Xilinx Inc., FPGA department of Intel (formerly Altera), and Lattice Semiconductor Corporation. We limit our considerations to devices large enough to contain an SoC. The size of an FPGA is measured in logic cells (LCs) that are equivalent to a four-input lookup table (LUT) paired with a flip-flop (FF). A memory controller, a common module of SoC, requires several thousands of LCs [e.g., DDR3 synchronous dynamic RAM (SDRAM) memory controller for Artix7 requires more than 6500 LCs]. The whole SoC is expected to be substantially larger; therefore, we limit the considerations to a series of FPGAs with devices of more than 50 000 LCs.

The FPGA considerations are based on data sheets from the vendors [5]–[18] and summed up in Table I. The above-mentioned three manufacturers offer products similar in various aspects, all based on LUTs and FFs, and with the capacity of up to millions of LCs. One can also see that memory controllers for high-capacity SDRAM are supported as software intellectual properties (IPs) or hardware preengineered blocks. Each FPGA contains memory blocks [block RAM (BRAM) with capacity from 9 kb to 45 Mb] distributed across its array.

Each of the considered devices also includes distributed lookup table RAM (LUTRAM). LUTRAMs utilize LUTs with limited hardware added for supporting on-the-fly LUT reprogramming. In the considered FPGAs, from 11% to 50% of

Manuscript received July 11, 2018; revised November 1, 2018 and January 7, 2019; accepted February 6, 2019. Date of publication March 8, 2019; date of current version May 22, 2019. This work was supported by the Ministry of Science and Higher Education as DS Project at the Chair of Multimedia Telecommunication Microelectronics, Poznań University of Technology. *(Corresponding author: Jakub Siast.)*

J. Siast and M. Domański are with the Chair of Multimedia Telecommunications and Microelectronics, Poznań University of Technology, 60-965 Poznań, Poland (e-mail: jsiast@multimedia.edu.pl; marek.domanski@put.poznan.pl).

A. Łuczak is with the vBionic sp. z o.o., 60-612 Poznań, Poland (e-mail: adam.luczak@vbionic.com).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org. This includes Appendices I–III to Section VII, and Appendix IV to Section VI. This material is 8 MB in size.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2899575

TABLE I
COMPARISON OF FPGAs

Vendor	Device name	Year of announcement	Technology [nm]	Embedded ARM processor	Number of logic cells	FPGA basic cell feature	Supported external memory controller	Size of a block memory (BRAM)	The smallest LUT-based distributed memory (LUTRAM) configurations with separate read and write port	Portion [%] of LUTs usable as RAM
Intel (Altera)	Stratix 10	'13	14	yes	378k — 5.5M	6-input LUT + 2 FF [18]	hard DDR4-2666	20kb & 45Mb	20×LUT as 20b×32-word deep or 10b×64-word deep RAM (32 bits per LUT)	25
	Arria 10	'13	20		160k — 1.15M			20kb		27 — 50
	Cyclone 10	'17	20	no	85k — 220k		hard DDR3-1866			21
	Stratix V	'10	28		236k — 952k		soft DDR3-1600			50
	Arria V	'11	28	yes	75k — 504k		hard DDR3-1066	20kb & 10kb		25 — 50
	Cyclone V	'11	28		25k — 301k		hard DDR3-800	10kb		24 — 35
Xilinx	Zynq UltraScale+	'15	16	yes	83k — 914k	6-input LUT + 2 FF	soft DDR4-2666	36kb or 2×18kb & 288kb	4×LUT as 3b×64-word deep or 6b×32-word deep RAM (48 bits per LUT)	30 — 50
	Virtex UltraScale+	'15		no	690k — 2.9M					
	Kintex UltraScale+	'15			205k — 915k					
	Virtex UltraScale	'14	20		783k — 5.5M		soft DDR4-2400	36kb or 2×18kb		14 — 22
	Kintex UltraScale	'14			318k — 1.5M					14 — 47
	Virtex7	'10	28		326k — 2M		soft DDR3-1866			28 — 43
	Kintex7	'10		yes (in Zynq)	66k — 478k					
	Artix7	'10			13k — 215k		soft DDR3-1066			
	Spartan7	'16		no	6k — 102k		soft DDR3-800			
	Spartan6	'09	45		4k — 147k		hard DDR3-800	18kb or 2×9kb		24 — 50
Lattice	ECP5	'14	40	no	12k — 84k	4 input LUT + 1/0 FF	soft DDR3-800	18kb	6×LUT as 4b×16-word deep RAM (10.7 bits per LUT)	50
	LatticeECP3	'09	65		17k — 149k					11 — 15
	LatticeECP2	'06	90		6k — 95k		soft DDR2-533			

LUTs can be used as RAM. The smallest available LUTRAM configurations have the depth of 16 or 32 words, depending on the producer. Different numbers of LUTRAM bits are available per utilized LUT; on average, 32 bits per an LUT for Intel devices, 48 for Xilinx, and 10.7 for devices from Lattice. As pointed out by a source related to Xilinx [3], RAM-capable LUTs are well spread over an FPGA and their potential should be considered in NoC and SoC designs for Xilinx FPGAs. All the considered FPGAs include LUTRAM, so this conclusion should be extended to all of them.

FPGAs, unlike ASICs, have a predefined network of programmable signal pathways connecting LUTs. Each LUT in FPGA implements a logic function with a limited number of logic inputs. For Intel and Xilinx products, functions of up to six inputs can be implemented in a single LUT. For Lattice products, the number of inputs is limited to four. As the number of inputs for required logic exceeds the number of single LUT inputs, it needs to be realized as multiple layers of LUTs connected with pathways. Additional layers of LUTs reduce the maximum clock frequency. In order to obtain the required design frequency, special care needs to be taken not to exceed the critical number of layers. It is not the case for ASIC designs, where the maximum clock frequency can be balanced with flexible lengths of pathways.

It can be concluded that the considered FPGAs differ from their ASICs counterparts. The key advantages of the considered FPGAs are highly available distributed RAM and support for high-capacity SDRAM, whereas the discussed frequency limitation is the main FPGA constraint. The identified advantages and constraints should be taken into account in the development of NoCs for FPGA.

III. REQUIREMENTS FOR NoC AND RELATED PREVIOUS WORKS

An NoC consists of switches connected with links. Processing elements (PEs) are connected to NoC using network interfaces. Switches can realize two switching techniques: packet switching and circuit switching. This paper considers only packet-switched networks that are characterized by high link utilization and are widely used for FPGA [3], [19]. In packet-switched NoC, a single message is divided into packets and sent across the network. The packets are further divided into flow control units called flits. A flit is a portion of data usually transferred at one clock cycle between connected switches [20].

A number of requirements for NoCs have already been identified [4], [21], [22]. The obvious requirements are as follows.

1) Utilization of resources (power, silicon area, LUTs, and FFs) should be minimized.
2) High data throughput should be offered.
3) Latency should be limited.
   There are some other important requirements that are addressed in a few references only.
4) Fairness of network access should be guaranteed, i.e., all network interfaces should experience throughput proportional to their relative request rates and the same latency [22].
5) Network should be reliable, i.e., it should be deadlock-free, whereas the requirements of the minimum throughput and the maximum latency should be met [21]. Another kind of reliability is fault tolerance required

by some applications [23], but fault tolerance is not considered in this paper.

Network reliability can be affected by congestions [20]. Congestions lead to throughput and latency fluctuations, thus the average throughput is below the theoretical maximum value, and the average latency is increased, especially under high network load conditions.

According to [1], [4], [24], [25], and [49], the main aspects influencing the throughput, latency, and probability of congestions are as follows: switching technique, topology together with the routing algorithm, and the size of buffers. For each of those aspects, a number of techniques have been developed to meet the requirements for NoCs.

1) The switching technique—two frequently used techniques are the wormhole and the virtual cut-through [20]. The wormhole is congestion sensitive and may result in low network utilization, but can provide low latency, and requires smaller buffers than the virtual cut-through. The latter requires larger buffers but provides low latency without limiting network utilization due to congestions.
2) The topology and the routing algorithm define paths in the network and influence the loads of individual links and switches. Uneven loads result in bottlenecks in the network that may cause congestions and unfair network access, leading to throughput and latency fluctuations [19].
   Topologies and routing algorithms that prevent congestions have already been investigated in the literature. In [19], the congestions are limited by spreading traffic across NoC evenly by using adaptive routing. Another approach is to use multiple physical (MP) link/network topologies [21], [27]. Redundant physical links increase throughput, reduce bottlenecks, and prevent congestions.
3) The size of buffers has been shown to have a major impact on throughput, latency, and occurrence of congestions [24].

Determining the buffer size is not trivial in the case of *a priori* unknown traffic load generated by PEs and unknown ability of PEs to accept packets from the network buffers. In the references, several mechanisms were proposed to determine the buffer size. The simplest method is to set the size of buffers to hold as many packets as can be generated. In the case of *a priori* unknown traffic load, this worst case approach results in unnecessarily large buffers [28].

More advanced methods of determining the buffer size exploit the statistics of the traffic load. Those statistics need to be explicitly provided [48], or a dedicated traffic load monitoring technique needs to be used [24]–[26], [29].

In [24], the buffer size is adjusted iteratively in consecutive SoC implementations. The monitoring module collects traffic statistics that are used to adjust the buffer size accordingly, and the estimated size is used in the next implementation. Multiple SoC implementations are time-consuming; therefore, in [25], simplified PEs are emulated and traffic statistics are collected faster. In [26], an NoC simulator is proposed to estimate traffic statistics prior to NoC implementation. The above-mentioned mechanisms [24]–[26] need training data and are sensitive to any change in the traffic pattern.

In [29], the total size of memory in a switch is constant; however, based on the measured traffic load at each switch output, the memory is assigned between output buffers adaptively, during runtime. Nevertheless, even in [29], it is pointed out that this mechanism is not dedicated to FPGA due to its complexity.

Commonly, networks distinguish the types of transmitted data and assign separate buffers to these different types. This technique is called virtual channels (VCs) [27], [30], [31], [49]. In NoCs using VCs, it is common to guarantee throughput and latency just for critical types of data, such as control messages. Therefore, the aforementioned buffer size determining techniques are only applied to buffers used by the critical data types. Buffers used by noncritical data can be optimized to reduce the memory cost.

In the NoC proposed in this paper, the buffer size does not depend on the traffic load. It is the opposite, and the traffic load is controlled to utilize the fixed-size network buffers without causing congestions. Details are provided in Section V.

IV. State of the Art in Interconnections for FPGA

A number of NoCs, crossbars, and bus architectures are proposed for ASICs in the references, and some of them are adopted for FPGAs. For the sake of brevity, we focus only on FPGA-oriented crossbars and NoCs.

The AMBA AXI4 [32] is a version of the AMBA crossbar recommended by ARM Ltd. for FPGAs. Originally, AXI4 was used to communicate with ARM cores embedded in many devices (see Table I). AXI4 modules, especially a switch called AXI4 Interconnect, are available as IP cores in the Xilinx Vivado Design Suite and Intel Quartus. Many other IP cores with AXI4 interface are available. Therefore, an FPGA-based SoC using AXI4 Interconnect and AXI4-compatible PEs can be developed rapidly. For this reason, AXI4 is widely used in FPGAs [3], [4]. Despite its popularity, AXI4 Interconnect has drawbacks, for example, in [3], long connections were pointed out as the main drawback, and the application of NoC instead of AXI4 Interconnect was suggested for Xilinx FPGAs.

Two types of NoCs for FPGA have already been proposed in the literature: soft, with infrastructure implemented using general FPGA resources, and hard, using hardware switches embedded into the FPGA fabric as an additional resource [33]–[35]. Hard NoCs may limit the flexibility of FPGA and have not been implemented in chips that are available, and therefore, only soft NoCs will be considered in this paper.

A few soft NoC architectures were proposed recently for FPGAs: Hoplite [36]–[41], RAR-NoC [19], CONNECT [3], [4], [31], [42], and LinkBlaze [43].

The above-mentioned works use different approaches to the requirements for NoC (cf. Section III). All the works consider the need to limit the usage of FPGA resources. Most of them dismiss adaptive routing techniques and exploit static routing, willing for a smaller control logic [3], [4], [36], [42]. Only in RAR-NoC [19] is the usage of adaptive routing considered instead of static, and the increased throughput is reported at the cost of the switch size increase by 11%.

Buffers can consume a significant part of the network resources; therefore, the authors of Hoplite [36] propose

an extreme approach and implement a toroid NoC without buffers. Recently, Hoplite-based NoCs were proposed with the aim of lowering Hoplite average latency by improved routing [37], [38] or changing the topology from toroid to butterfly fat tree [41]. The small size of Hoplite NoC is paid for by no guarantees for packet latency and lack of reliability. Using no buffer may not be justified for FPGAs that provide easily available distributed RAM (LUTRAM).

In [19] and [42], it is proposed that the buffer size is limited by implementing wormhole switching that requires smaller buffers than virtual cut-through. Xilinx-related authors pointed out [3] that LUTRAMs in Xilinx FPGAs have some generally defined minimum depths, so most of the LUTRAM capacity may be wasted by wormhole switching with shallow buffers. Virtual cut-through is proven [3] to be an appropriate switching technique for a system with moderate size packets, i.e., shorter than the depth of LUTRAM-based buffers.

In this paper describing the CONNECT switch [31], the LUTRAM potential is underlined, and the buffers are implemented strictly using LUTRAM. The configurability of the depth and width of the CONNECT switch is an advantage of the implementation, but it is also pointed out that even fixed but appropriate buffer depth can improve the predictability of resource utilization.

Based on [3] and [31], one can conclude that using buffers with a depth equal to the depth of LUTRAM, using packets of a length shorter than the buffer depth and employing virtual cut-through switching can result in resource-efficient NoC. Therefore, we implement those ideas in RingNet NoC.

The maximum operating frequency and the average throughput and latency are reported for all the above-mentioned networks in their source papers. The respective reports are provided for individual FPGA device types [3], [19], [36], [42] or for entire FPGA device lines [4]. The lack of reports for FPGAs from different vendors may be an obstacle in determining the usability of NoCs.

Another NoC designed, especially for FPGA, is LinkBlaze [43] that is dedicated to UltraScale+ devices from Xilinx. In particular, the switching logic was optimized for six-input LUTs, and the switches were placed manually in the array with the aim of connecting them with global pathways. This device-aware network design results in a high-frequency NoC with its throughput higher than obtained for Hoplite and CONNECT when implemented on UltraScale+ FPGA. LinkBlaze is an example of NoC optimized for one line of FPGAs, whereas in this paper, we look for more universal NoC.

A few of the presented NoC proposals focus on the requirement of NoC reliability. RAR-NoC [19] uses a traffic monitor to control the routing algorithm implemented with the aim of reducing congestions. In [3] and [4], switches that support VCs are implemented. However, even though fairness is an essential reliability parameter, it is out of the scope of most propositions. Only Papamichael and Hoe [4] point out the importance of fairness but gave no numerical results for their NoC. In this paper, we will provide a fairness analysis for RingNet, together with resource utilization and the maximum operating frequency for various FPGAs to prove its usability.

Out of the above-mentioned NoCs, only the authors of LinkBlaze [43] consider FPGAs' support for SDRAM. SDRAM is a crucial component of many SoC projects [33] and utilizes a substantial part of NoC throughput. In [20], an example of an ASIC SoC for an AVC decoder is presented. In the system, the total communication traffic to/from SDRAM is much higher than the one required for communication between other PEs. This type of memory-oriented SoCs is the target application for RingNet described in Section V.

V. RingNet Architecture Description

Most NoC proposals focus on switch architecture only. In this paper, we propose a complete NoC architecture and a protocol for a memory-oriented SoC, which we call RingNet. In the development of RingNet, the conclusions from Sections II-IV are taken into account.

A. Primary Ideas of the Proposal

1) As described in Section III, determining the buffer size is often complex due to the *a priori* unknown traffic load. In RingNet, the traffic load is controlled by a destination PE, which guarantees that packets injected into the network can be accepted by the destination PE without congestions. This way, fixed-size network buffers can be utilized.
   With the aim of providing the traffic control mechanism, we disallow direct communication between PEs. In RingNet, all traffic goes through one of the system buffers: system memory (e.g., external SDRAM with a memory controller implemented in FPGA) or the reflector. The system memory is used as a data buffer; on the other hand, the reflector is a dedicated network buffer for control messages. The reflector is introduced because it serves functions that are not supported by an ordinary memory controller. Among its functions, the reflector informs a PE about data waiting to be read from a buffer. For two PEs to communicate, the first PE writes to a system buffer and the second PE reads from it to complete the communication. What is important, in RingNet, a PE requests data from system buffers when it is ready to accept it. However, sending data to the system buffers is not a matter of any restriction and a sending PE can work at its own pace. The reflector and the memory controller are the only devices in RingNet which need to be prepared for traffic with *a priori* unknown pattern.
   RingNet has two distinctive properties among other NoCs proposed for FPGA. These are traffic load controlled by a destination PE and communication through system buffers.
2) Considering the conclusions from Section IV, we propose virtual cut-through switching for efficient utilization of LUTRAM. In Section IV, it was also concluded that packets should be small enough to fit into the available LUTRAM. For the considered FPGAs, it means packets should be shorter than 32 flits (see Table I).
3) The usage of logic functions with a low number of inputs can give a high clock frequency implementation in

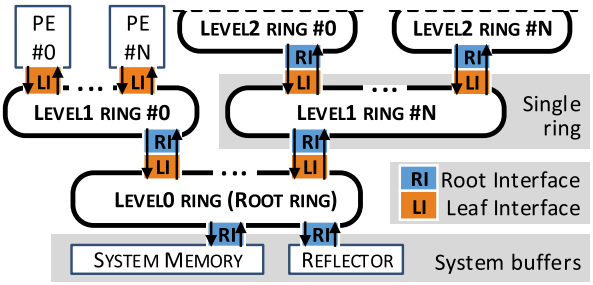

Fig. 1. RingNet topology.

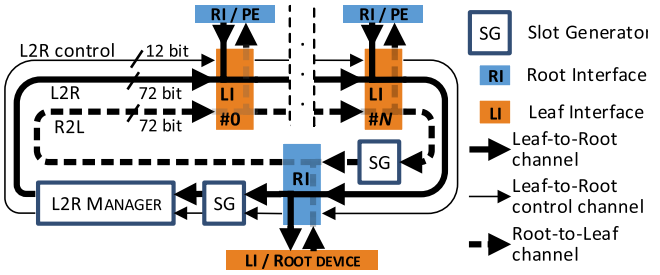

Fig. 2. Single RingNet ring.

FPGA (cf. Section II). In [4], it is shown that a low number of ports results in a high-frequency switch. In fact, a three-port switch is the smallest switch usable in a network (see also [33] and [43]). In order to maximize the frequency of the network, we use three-port switches, called the leaf interface (LI) and root interface (RI).

B. Secondary Ideas of the Proposal

1) A tree-of-rings topology is used (see Fig. 1) with the system memory and the reflector connected to the ring at the root of the network tree, whereas PEs are connected to the rings at higher levels of the tree. The ring topology is recommended for FPGAs by several authors [31], [33], and it is one of just few topologies that can be constructed using three-port switches. Moreover, the ring topology allows the network to spread over the whole FPGA area.
   On the other hand, in NoCs with ring topology, latency increases proportionally to the number of PEs, and for a high number of PEs, this latency may be unacceptably high. Such latency can be reduced with the use of a mixed topology of smaller rings connected to a tree.
   Both the tree and ring topologies are easy to scale in FPGAs without reducing the maximum clock frequency [31]. For the tree-of-rings topology, there is one path between the root and a leaf device, so static routing may be efficiently used, which simplifies the RingNet logic.
2) The choice of topology affects the parameters of RingNet, especially it limits network throughput. The throughput of a ring is a function of the flit width and the clock frequency. The width of a flit in RingNet is constant, so the maximum ring throughput is determined by the maximum clock frequency for a given FPGA. To overcome this limitation, the MP network technique is used. A single ring at any level of the tree can be replaced with multiple rings connected in parallel. The traffic is spread evenly between the parallel rings, and the throughput is multiplied. The traffic in RingNet aggregates in a root ring and this level can also be multiplied to meet the throughput of the attached SDRAM memory.
   The usage of the MP technique makes the throughput of RingNet controllable. In this situation, the system memory throughput becomes an obvious limitation of our approach. Nevertheless, the system memory load can be reduced by connecting additional memory buffers at any ring. PEs attached to a common ring can exchange data using the additional buffers. Each such buffer is connected to a ring through its own RI and can utilize the block RAMs available in all considered FPGAs.
3) Flits in RingNet have 8 data bytes with additional 8 bits of byte-enable (BEN). Similarly, in AXI4, BEN bits indicate which data bytes from a flit are valid. One of the goals of RingNet development is to provide throughput equal to or higher than the throughput of SDRAMs supported by FPGAs. While using flits with 8 bytes of data, the throughput of RingNet is compared with the throughput of the supported SDRAMs.
   a) The slowest SDRAM from Table I is DDR2-533, which provides the throughput of 4.3 Gb/s for an 8-bit interface, whereas a single RingNet ring running at a moderate frequency of 75 MHz already exhibits the throughput of 7 Gb/s.
   b) The most demanding DDR4-2666, supported in Xilinx UltraScale+ series and Intel 10 series, offers 192 Gb/s for a 72-bit interface. For comparison, five parallel RingNet rings can transfer 233 Gb/s when running at the clock frequency of 500 MHz that is easily achievable for FPGAs.
4) Considering the requirements for fair network access (Section III), we propose a flow control mechanism for RingNet with the access controlled locally, at the level of each ring (cf. Section VI).

VI. Elements and Protocol of the RingNet Network

This section provides an overview of the implementation of example RingNet components and protocol.

In this NoC, there are two physical channels. The first one transports packets from a PE to a system buffer (system memory or reflector). The second one transports packets from a system buffer to a PE. PEs are connected at the leaves of the network tree, whereas the system buffers are connected at the root of the tree; therefore, the first channel is called leaf-to-root (L2R) and the second one is called root-to-leaf (R2L).

The main elements of RingNet are two types of network interfaces: LI and RI. The LI is used to connect a PE to a network. The RI is used to connect a system buffer. As depicted in Fig. 1, a combination of RI and LI is used to connect rings at different levels of a network tree. Both RI and LI are three-port switches and they insert packets to a ring using a 2×1 multiplexer and accept packets from a ring using

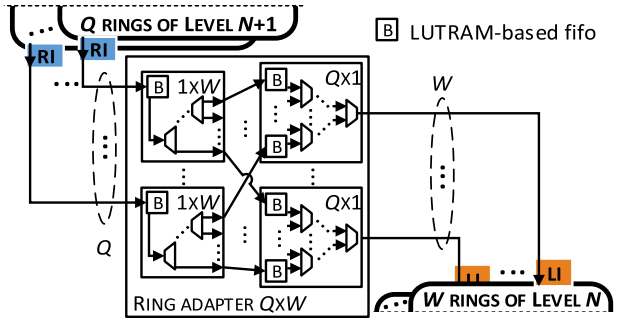

Fig. 3. Ring adapter for L2R channel.

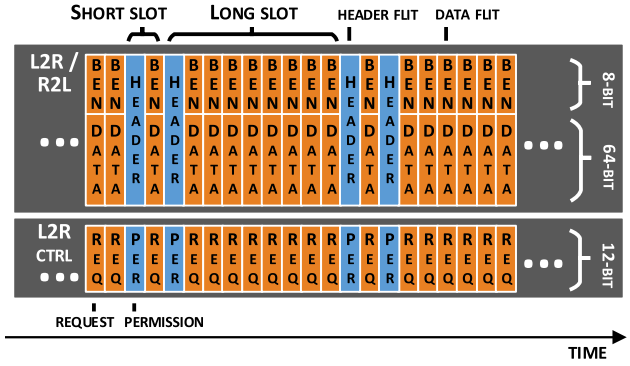

Fig. 4. Time slots at R2L, L2R, and L2R control channels, passing through an interface.

1×2 demultiplexers. Detailed descriptions of the LI and RI are provided in Supplementary Materials as Appendix IV.

Each ring in RingNet has one L2R channel and one R2L channel (see Fig. 2). A flow control mechanism for the L2R channel uses an additional L2R control channel.

As already mentioned, the rings can be connected in parallel in order to increase the overall network throughput, e.g., at certain critical levels of the tree. The multiplied rings can be connected to other levels of the tree using a dedicated adapter. Fig. 3 depicts an adapter for the L2R channel. The adapter is placed between LIs and RIs of the rings when at least one ring is multiplied. An individual adapter is used for L2R and R2L channels. It is built of 2×1 multiplexers and 1×2 demultiplexers and LUTRAM-based 32-flit deep buffers.

For RingNet, a protocol is developed, which provides communication between PEs, limits congestions, provides fair network access, and supports packet priorities. The protocol covers four layers of the open systems interconnection (OSI) model, from the data link layer to the session layer. Higher OSI layers are out of the scope of the RingNet protocol.

A. Data Link Layer

The data link layer specifies the structure of flits and packets and defines the ring access protocol.

As recommended in Section V, packets should be shorter than the depth of the buffers. For the FPGAs in Table I, the shallowest LUTRAM is 32-word deep. Therefore, 32 and 64-flit deep buffers are used in RingNet. We choose packets with two possible lengths of 2 and 9 flits. The purpose of the two lengths is described in Section VI-C. The first flit of a packet is the header that encapsulates control information for each protocol layer (especially the routing information). The following 1 or 8 flits transport data. In RingNet flits, 64 bits are transmitted with additional 8 bits of BEN.

Fig. 4 depicts a pattern of time slots for header flits and data flits circulating through interfaces around a ring. Those flit slots are organized into long and short slots for long and short packets, respectively. Slots are produced by the slot generator (SG) depicted in Fig. 2. The LI and the RI can populate those slots with packets.

The access to the L2R channel slots is controlled by the L2R manager. In order to access this channel, LI sends a request to the L2R manager and obtains permission. The requests and permissions are sent using the L2R control channel. The L2R manager keeps the requests in LUTRAM-based buffers and grants the permissions based on their order of arrival. This strategy is sufficient to guarantee fair access to the L2R channel for all PEs in terms of the average latency and granted throughput (see simulation results in Section VII).

The flow control mechanism used for the L2R channel also guarantees fair access to the R2L channel. Therefore, no flow control for the R2L channel is needed, and no additional resources are used. Details will be given in Section VI-C.

RingNet supports four packet priorities, ordered from the highest priority 3 to the lowest priority 0. Each packet and the associated request have an assigned priority. The L2R manager has individual buffers for requests of different priorities.

The LIs buffer packets, send requests, and insert buffered packets onto the L2R ring after acquiring permission. Sending a request and obtaining permission requires some additional time, so a constant flow of packets from one LI may be impossible. To overcome this problem, multiple packets can be buffered in LI and multiple requests can be sent without receiving permission. This overlap lets a single LI exploit the full ring throughput. The buffer used in LIs utilizes LUTRAM with a depth of 64 words. It provides enough capacity for six short packets and five long packets.

According to the virtual cut-through switching, the packets from the L2R channel leave a ring through RI only if the RI has buffer space available for a whole packet. Otherwise, the packet is rejected by the RI and starts to cycle around the ring until enough space is available. A circulating packet is recognized by the L2R manager, and no new permission is granted until all packets that circulate on L2R leave the ring. This way, all packets should finally leave the ring with limited differences in latency. A detailed description of the L2R manager is provided in Supplementary Materials as Appendix IV.

B. Network Link Layer

The network layer protocol defines RingNet addressing. A different addressing is used for the L2R and R2L channels. For the L2R channel, 37-bit memory addressing is used; therefore, up to 128 GB can be addressed in the network; both the reflector and system memory have assigned memory address spaces recognized by RIs. For the R2L channel, the packet address contains five (each 4-bit wide) numbers. Each number identifies the LI that should accept the packet at

a certain RingNet tree level. The addressing used in the R2L channel limits the number of network tree levels to 5 and the number of network interfaces connected to a single ring to 15. The applied addressing scheme limits the number of PEs that can be connected to the RingNet network to 759 375.

C. Transport Layer

The transport layer defines logical channels and describes the communication between PE and system buffers.

In RingNet, packets are transported to and from memory-mapped system buffers; therefore, the RingNet protocol supports basic memory operations. The memory read or write operation is encoded in a packet header, thus creating a read or write packet. If a packet is sent from PE via the L2R channel to a system buffer, a response packet has to be sent back to the PE by the system buffer through the R2L channel. For the read operation, the response packet contains the data read from the memory; on the other hand, for the write operation, the response packet is a write operation acknowledgment.

Logical channels are defined for RingNet. In each logical channel, a certain memory operation and packets of a certain length are used. Most transfers are realized by two logical channels as follows.

1) *Logical Write Channel:* PE sends a long write packet through the L2R physical channel, with data to be stored to the memory. A short response packet is sent back by a system buffer through R2L as a write acknowledgment.
2) *Read Logical Channel:* PE sends a short read packet through the L2R physical channel, and a long packet with data read from the memory is sent back from a system buffer through the R2L physical channel.

Other types of logical channels are used by the session layer. What is very important is that for read and write logical channels, a single response packet is sent via R2L in response to the L2R packet. Therefore, the same number of packets is transferred through L2R and R2L channels. This way, fair access to the L2R channel guarantees fair access to the R2L channel as well, despite the fact that for R2L, no requests–permissions mechanism is used explicitly. The proposed flow control prevents congestions in both physical channels, still utilizing resources in the L2R channel only.

Theoretical throughput $T_{\mathrm{RW\_MAX}}$ of the logical read and write channels, expressed in bits per clock tick (bpt), can be calculated according to the following formula:

$$T_{\mathrm{RW\_MAX}} = R \cdot 64 \cdot 8/11[\mathrm{bpt}] \tag{1}$$

where 64 is the number of data bits in a flit, the 8/11 factor is the share of flits carrying write data at the L2R physical channel, and the share of flits carrying read data at the R2L physical channel, and R is the number of parallel rings used at the root ring.

D. Session Layer

The transport layer describes the communication between PEs and system buffers but not between individual PEs. The communication between PEs is finally possible at the session layer. An example of communication between two PEs using the system memory and the reflector is depicted in Fig. 5. The example illustrates two transactions of sending data from PE1 to PE2.

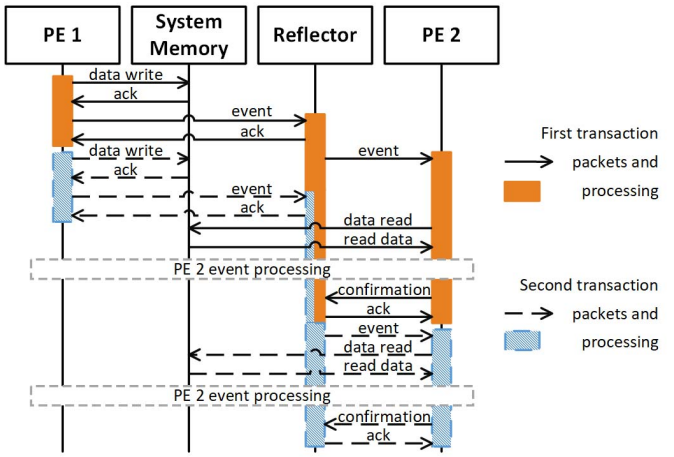

Fig. 5. Sequence diagram for two transactions between PEs.

PE1 starts the transaction by sending data to the system memory. Then, PE1 informs PE2 that data in the system memory are ready to be processed by sending an event message through the reflector. An event is a short packet with an address in a range reserved for the reflector. The events are buffered in the reflector and sent to the destination PE one at a time. After processing data, PE2 sends the event confirmation to the reflector, which ends the transaction.

The example demonstrates two advantages of the proposed indirect PE communication. First, congestions arising due to PE overload are prevented, because the destination PE (PE2 in the example) controls its load. Second, the source PE (PE1 in the example) does not need to monitor the state of the destination PE before starting a new transaction.

The reflector has short dedicated first-in–first-out buffers (FIFOs) for each PE and a general buffer for events that do not fit into the dedicated FIFOs. When space in the dedicated FIFO is released, an event from the general buffer is transferred to the dedicated FIFO.

The events transfer control information and use packets with the highest priority (3). Moreover, a dedicated VC is created by reserving part of the LI buffer just for the event packets.

The reflector provides additional system functions, such as informing about PEs connected to NoC, registering new PEs, informing about events buffered in the reflector, alarming about the fullness of event FIFOs, resetting a PE, and so on.

VII. Simulation Results

The proposed RingNet is simulated with the aim of testing its throughput and latency. Moreover, the following reliability aspects are tested: interchannel dependencies, network access fairness, and the priority mechanism.

The simulations use a model of the system memory that supports unlimited throughput and introduces negligible latency, thanks to which the system memory does not affect network performance results. There is no need to simulate various traffic patterns with various packet destinations [4], [49] because the system memory is the destination of all the data in RingNet.

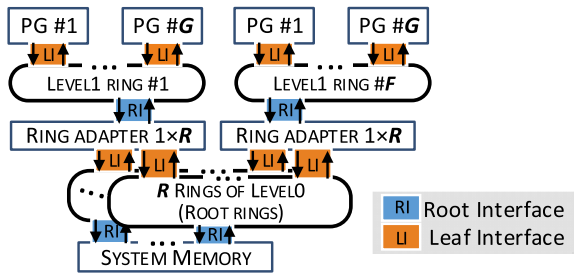

Fig. 6. Simulated RingNet topology.

PEs are simulated with the use of packet generators (PGs). Each PG independently generates packets for the logical read and write channels with the configurable average delay D_{aver} between two consecutive packets

$$D_{\text{aver}} = \frac{N \cdot 64 \cdot 8}{T_{\text{RW\_MAX}} \cdot L \cdot 11} \tag{2}$$

where $T_{\text{RW\_MAX}}$ is the theoretical throughput of a network, and L is the requested aggregated load generated by all PGs expressed as a percentage of the throughput $T_{\text{RW\_MAX}}$. Equation (2) is explained because 64 is the number of data bits in a flit, the 8/11 factor is the share of flits carrying write data at the L2R physical channel and the share of flits carrying read data at the R2L physical channel. The actual delay D is defined as the time interval that a PG waits before it sends another packet. D is an output of a random number generator with discrete uniform distribution $\mathcal{U}\{0.8D_{\text{aver}}, 1.2D_{\text{aver}}\}$ used for making the traffic more realistic.

In the experiments, RingNet with two levels of rings is simulated as depicted in Fig. 6. The size of the network is controlled using the following parameters.

1) R: Multiplication degree of the root level, i.e., the number of parallel rings used at the root level (level 0).
2) F: The number of first-level rings.
3) G: The number of PGs connected to a single first-level ring.

Latency is measured in terms of the clock cycles that elapse between the emission of a new packet from a PG and the acceptance of the corresponding response packet (send by the system memory) by the PG.

The Verilator compiler [44] is used to convert Verilog code into C++ code that is further compiled by the GCC compiler and executed to perform functional simulations. The presented results summarize the simulations of over 3000 different configurations of network size and load.

A. Performance Test

In this test, the achievable throughput of RingNet is checked together with the average latency that can be expected under different conditions. The theoretical value of throughput for read and write logical channels can be calculated according to (1).

In the experiment, all packets are sent between PGs and the system memory. Only packets with the lowest priority (priority 0) are generated. The parameters of the test are: the multiplication degree R of a root level is restricted to the intervals 1–4, the number F of the first-level rings is from the intervals 1–5, while the number G of PGs connected to a single first-level ring is set in the range of 1–15 (up to 75 PEs are connected for $F = 5$ and $G = 15$). The aggregated load generated by all PGs is set in the range of $0\% \div 100\%$ of the theoretical throughput $T_{\text{RW\_MAX}}$ (1). The logical channel load is separately set for the read and write channels.

TABLE II
AVERAGE LATENCY (EXPRESSED IN CLOCK CYCLES) FOR READ CHANNEL FOR VARIOUS NETWORK CONFIGURATIONS

Number of 1st level rings (F)	Number of PGs connected to a 1st level ring (G)						
	1	2	3	4	7	15
5	141	163	169	175	201	258
4	130	151	157	163	189	245
3	129	148	155	161	185	241
2	113	134	142	145	182	225
1	95	118	120	122	147	195

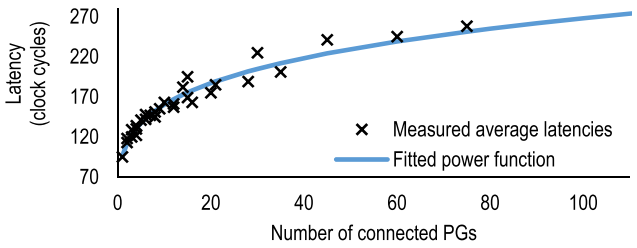

Fig. 7. Average latency (expressed in clock cycles) for read channel for various network sizes.

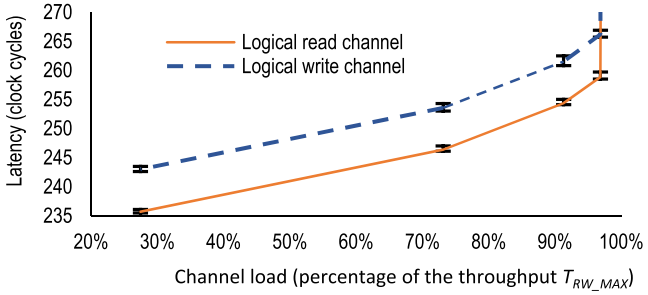

Fig. 8. Load-latency curves. These curves represent the average latency of each logical channel as a function of the channel load. Error bars represent the minimum and maximum average latencies of the channel at a given load when the second channel load changes in the range of 0% to 100%.

In Table II, the average latency for the read channel is presented for a high load (92% to 97%) and various network sizes ($R = 1$, $F = 1 \div 5 \times G = 1 \div 15$). One can see that latency increases with the increased number of connected first-level rings and PGs. For the write channel, the measured latencies are, on average, seven clock cycles longer than those reported in Table II. The results for the write channel and for the increased number of parallel rings used at the root level ($R = 2 \div 4$) are provided in the Supplementary Material as Appendix I. It is tested that increasing the root ring multiplication degree by one increases the average latency reported in Table II by only six clock cycles.

In Fig. 7, the latencies in Table II are presented as a function of the number of connected PGs. The fitted latency function flattens for a growing number of PG, i.e., for large

networks, the latency increases slowly with the growth of the network.

NoCs are often characterized by providing load-latency curves that represent packet latency as a function of network load [1], [4], [20], [22], [27], [28], [30], [36], [45]. Fig. 8 depicts load-latency curves for the logical channels of the RingNet network with five first-level rings and 15 PGs connected at each ring ($F = 5 \times G = 15$, 75 PEs connected) and four parallel rings used at the network root ($R = 4$). The results for other network configurations are provided in the Supplementary Material as Appendix II.

The average latency for both logical channels increases with the channel load. For a 100% channel load, all network buffers are filled, and the average latency increases drastically, which seems to be obvious. On the other hand, an increase in channel load from 27% to 97% results in the average latency increase by only 10%. Moreover, both logical channels are independent, i.e., the load of one logical channel has a negligible impact on the average latency in other logical channels and does not influence its throughput.

For the presented network configuration (and for all other tested configurations), throughput $T_{\text{RW\_MAX}}$ calculated according to (1) is obtained for both the read and write channels, regardless of the multiplication degree of the root ring (R), the number of first-level rings (F), and the number of connected PGs (G).

The performance of RingNet is compared with the performance of the state-of-the-art networks. For RingNet, an average latency of about 90 clock cycles is observed for a network with just one PE and increases to about 250 clock cycles when 75 PEs are connected. Negligible changes in latency are observed for traffic loads set in the range of 27%–98%. For a mesh NoC with 64 PEs described in [3], an average latency of 60 clock cycles is reported. Next, a latency of about 10 clock cycles is reported for CONNECT [4] mesh with 16 PEs. The relatively high latency observed in the RingNet network is a result of two factors. First, the RingNet ring topology features relatively long routes when compared with a mesh. Next, latency is caused by the applied flow control mechanism that exchanges flow control messages before a packet can be sent. Therefore, RingNet is not recommended for designs requiring very low latency. Nevertheless, the applied flow control mechanism is demonstrated to provide a fair network access not reported for the state-of-the-art NoCs [3], [4].

In order to compare the throughputs of RingNet and the NoCs proposed in [3] and [4], we consider a traffic scenario where all PEs send packets to just one destination, i.e., a single memory device connected to the network. In such a simplified case, the throughput of the network is at most equal to the throughput of a network interface through which the memory is connected. The throughputs of a network interface for NoCs from [3], [4] and for RingNet are proportional to the flit width and frequency of a clock. In Section VIII, it will be demonstrated that RingNet features substantially higher maximum frequency and uses less resource for the same flit width. It means that in the traffic scenario where all PEs send packets to just one destination, RingNet provides higher throughput than the state-of-the-art NoCs from [3] and [4].

TABLE III
TRAFFIC STATISTICS OVER 75 PGS

Load (percentage of the throughput T_{RW_MAX} (1))	Average latency (clock cycles)				Average throughput (bits per cycle)			
	Read		Write		Read		Write	
	$\overline{L_{PG}}$	$\sigma_{\overline{L_{PG}}}$	$\overline{L_{PG}}$	$\sigma_{\overline{L_{PG}}}$	$\overline{T_{PG}}$	$\sigma_{\overline{T_{PG}}}$	$\overline{T_{PG}}$	$\sigma_{\overline{T_{PG}}}$
27%	236	6	243	6	0.7	0.01	0.68	0.01
97%	259	5	267	5	2.4	0.01	2.4	0.01
100%	1157	7	1175	9	2.5	0.00	2.48	0.00

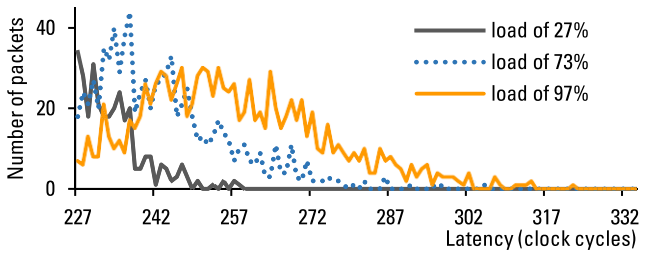

Fig. 9. Histogram of packet latency for read channel for an example of PE.

The most important conclusion drawn from the test is that the theoretical throughput $T_{\text{RW\_MAX}}$ (1) is guaranteed for RingNet, and the latency is correlated with network size, and it can be estimated during network configuration.

B. Network Access Fairness Test

In test *A*, the average parameters over all PEs are estimated. In real network applications, fair access to NoC for each PE can be an even more important aspect than the average parameters. In order to check the fairness, the statistics need to be gathered for each individual PE.

In the test, a RingNet network with 75 PEs is simulated ($F = 5 \times G = 15$), and four parallel rings are used at the root level ($R = 4$). PEs are simulated using PGs requesting the aggregated load of 27%–100% of the network throughput $T_{\text{RW\_MAX}}$ (1). In this experiment, all PGs send packets to the system memory with priority 0. For each individual PG, we collect the values of the average latency L_{PG} expressed in clock cycles, and the values of throughput T_{PG} expressed in bits per clock cycle, for both logical channels.

In Table III, the average values of L_{PG} and T_{PG}, calculated over all PGs ($\overline{L_{\text{PG}}}$ and $\overline{T_{\text{PG}}}$, respectively), are presented together with a standard deviation for those variables ($\sigma_{\overline{L_{\text{PG}}}}$ and $\sigma_{\overline{T_{\text{PG}}}}$, respectively). Low values of the standard deviations mean that all PGs are expected to exhibit the same performance expressed in terms of the average latency and throughput. These conclusions are valid for a wide range of loads. The results prove that RingNet provides fair access for each connected PE. More results that prove the fair access for other network sizes are available in the Supplementary Material as Appendix III.

C. Latency Distribution Test

Now, the distribution of packet latency in RingNet is researched.

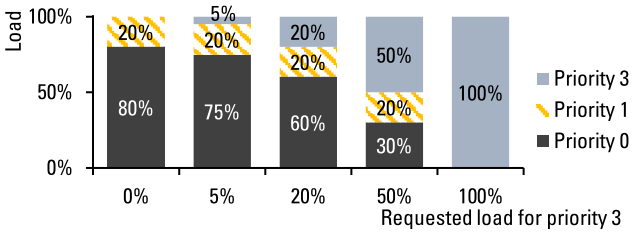

Fig. 10. Load, expressed as percentage of the theoretical throughput T_{RW_MAX} (1), granted to each priority under constant requested load for priorities 0 and 1 and increasing requested load for priority 3.

A network with 5 first-level rings and 15 PGs connected at each ring ($F = 5 \times G = 15$) is simulated under the loads of 27%–97%. Four parallel rings are used at the root level ($R = 4$).

Fig. 9 depicts a histogram of packet latency for the read channel. The data are collected for a sample PE (PG connected to the first-level ring as the fourth PE). For the write channel and for other PGs, the histograms look similar, which is reasonable in view of the fact that fair network access was proven in test B.

The values of 227 and 322 are the minimum and maximum latencies observed, respectively. For low loads, more packets experience latency close to the minimum. An increase of the load results in shifting the histogram toward higher latencies. However, the difference between the extreme values is just 95 clock periods, which is 42% of the minimum latency.

D. Packet Prioritization Test

The proposed RingNet architecture supports packet prioritization. Higher priority packets obtain access to the L2R channels first. Under high-load conditions, the high-priority packets should experience lower latency; also, the requested throughput should be granted starting from the highest priority requests. To check the prioritization mechanism, a test is conducted under high-load conditions.

1) Network with 28 PGs is tested ($F = 4 \times G = 7$).
2) Two parallel rings are used at the root level ($R = 2$).
3) Each PG has three internal sources of packets with priorities 0, 1, and 3. The source of packets with priority 0 constantly tries to send a packet, and on its own, it would generate the load of 100% of the theoretical throughput T_{RW_MAX} (1). The sources of packets with priority 1 and 3 are set to request various loads.

In Fig. 10, examples of load shares are presented for sources of packets with priority 1 requesting 20% of the throughput, and sources of packets with priority 3 requesting from 0% to 100% of the throughput. For the highest priority packets, the share of throughput is always granted as requested. 20% of the throughput, requested for priority 1 packets is granted when the packets with the highest priority use less than 80% of the RingNet throughput. The example demonstrates that the lower priority packets do not influence the share of throughput granted to the higher priority packets. It needs to be emphasized that using the prioritization mechanism does not limit the aggregated throughput below the theoretical value T_{RW_MAX} (1) in any tested case.

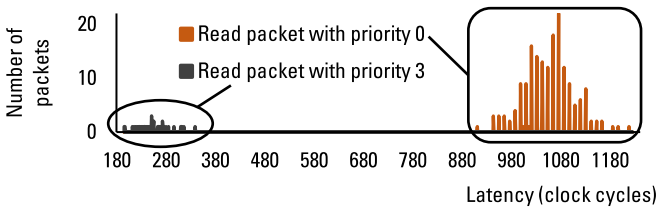

Fig. 11. Latency histogram for read channel.

In Fig. 11, latency histograms are depicted for packets with different priorities. The presented statistics are collected for the case when packets with priorities 3 and 0 utilize 5% and 95% of the network throughput, respectively. The histograms for both priorities are concentrated. The high-priority packets experience moderate latencies even under the maximum network loads.

The proposed communication protocol with prioritization and flow control mechanisms was tested in a number of simulations. The results prove that the RingNet network throughput can be calculated according to (1) and the average latency can be estimated based on NoC size. Moreover, the proven fair access to RingNet network is a distinctive property among other NoCs proposed for FPGA.

In addition to the simulations, RingNet was tested in a hardware application. It was successfully used as a communication backbone for an FPGA-based depth map estimation device [46] demonstrating the above-mentioned features.

VIII. RingNet Synthesis

This section summarizes RingNet synthesis for various FPGAs and tests RingNet scalability.

RingNet is synthesized for sample FPGAs from Xilinx (Artix7, Kintex UltraScale, Virtex7), Intel (Stratix V, Arria V), and Lattice (ECP5). We synthesize individual RingNet modules and rings of different sizes that are useful for networks with a tree topology, i.e., rings with one RI and from 2 to 15 LIs. For a fair comparison, the same synthesis software—Synplify Premier 2017.03-SP1—is used for all devices. The synthesis tools available in Synplify, such as the shift register inferring and the register and logic replication [47], can balance the resource cost and clock frequency. The tools perform differently for different devices. With the aim of obtaining fair results for all devices, we disable the shift register inferring tool. Also, the register and logic replication tools are effectively disabled by setting the requested frequency to a low value of 1 MHz.

Our comparison is summarized in Table IV and reports the maximum clock frequency and the utilized resources.

The obtained frequency values are compared with the maximum frequencies specified by vendors for hardware blocks of each FPGA [5]–[15]. The frequencies for the common DSP and BRAM blocks are presented. The maximum frequencies are given for the fastest, fully pipelined configuration of each hardware block. For BRAM, two versions are given: one with simultaneous read and write to the same address handled in additional logic (RW check), and another one without this extension (no RW check). The provided frequencies are a raw estimation of what the maximum frequency can be

TABLE IV
RESOURCES UTILIZATION AND ESTIMATED MAXIMUM FREQUENCY AFTER SYNTHESIS FOR RINGNET MODULES AND RINGS

RingNet modules, RingNet rings, and FPGA hardware blocks	FFs utilized / LUTs utilized as logic + LUTs utilized as RAM / Maximum clock frequency in MHz					
	Xilinx			Intel		Lattice
	Artix7 xc7a100tcsg324-1	Kintex UltraScale xcku060-ffva1156-3-e	Virtex7 xc7vx550tffg1158-1	Stratix5 5SGXMABK2H40C3	Arria5 5AGXBA7D6F31C6	ECP5 lfe5u_85f-8
Ring adapter 1×2	383 / 93 + 48 / 485	383 / 94 + 48 / 837	383 / 94 + 48 / 469	383 / 105 + 144 / 551	383 / 102 + 144 / 377	383 / 189 + 216 / 312
Ring adapter 2×1	462 / 184 + 96 / 459	462 / 184 + 96 / 837	462 / 184 + 96 / 486	462 / 123 + 288 / 529	462 / 127 + 288 / 345	462 / 351 + 432 / 312
L2R Manager	499 / 271 + 96 / 337	499 / 264 + 96 / 875	499 / 286 + 96 / 461	499 / 317 + 128 / 527	507 / 349 + 128 / 334	499 / 373 + 384 / 374
Slot Generator	106 / 11 + 56 / 487	106 / 11 + 56 / 837	106 / 11 + 56 / 487	106 / 14 + 168 / 983	106 / 12 + 168 / 468	106 / 16 + 126 / 372
Leaf Interface	759 / 227 + 104 / 405	759 / 218 + 104 / 807	759 / 227 + 104 / 509	759 / 145 + 166 / 524	761 / 160 + 166 / 293	761 / 391 + 450 / 235
Root Interface	648 / 125 + 48 / 469	648 / 125 + 48 / 837	648 / 125 + 48 / 483	648 / 56 + 144 / 779	648 / 57 + 144 / 381	648 / 204 + 216 / 314
Ring 2×1	2k8 / 805 + 424 / 354	2k8 / 801 + 424 / 753	2k8 / 805 + 424 / 445	2k8 / 631 + 916 / 449	2k8 / 648 + 914 / 273	2k8 / 1k3 + 1k4 / 235
Ring 4×1	4k3 / 1k2 + 632 / 363	4k3 / 1k2 + 632 / 700	4k3 / 1k2 + 632 / 460	4k3 / 934 + 1k2 / 430	4k3 / 966 + 1k2 / 250	4k3 / 2k0 + 2k3 / 235
Ring 6×1	5k8 / 1k7 + 840 / 333	5k8 / 1k7 + 840 / 763	5k8 / 1k7 + 840 / 440	5k8 / 1k3 + 1k6 / 409	5k8 / 1k3 + 1k6 / 262	5k9 / 2k8 + 3k3 / 235
Ring 8×1	7k3 / 2k1 + 1k0 / 328	7k3 / 2k1 + 1k0 / 726	7k3 / 2k1 + 1k0 / 440	7k3 / 1k6 + 1k9 / 397	7k3 / 1k7 + 1k9 / 272	7k4 / 3k6 + 4k2 / 235
Ring 11×1	9k5 / 2k8 + 1k4 / 333	9k5 / 2k8 + 1k4 / 753	9k5 / 2k8 + 1k4 / 446	9k5 / 2k1 + 2k4 / 434	9k6 / 2k2 + 2k4 / 241	9k7 / 4k9 + 5k8 / 235
Ring 15×1	12k5/ 3k7 + 1k8 / 325	12k5/ 3k7 + 1k8 / 763	12k5/ 3k7 + 1k8 / 443	12k5/ 2k7 + 3k1 / 403	12k5/ 2k8 + 3k0 / 222	12k8/ 6k5 + 7k6 / 235
Switch from [3]		--- / 1678 / 470				
Switch from [4]	--- / 430 / 241 (results for Virtex-6 LX760 speed grade -2)					
DSP	/ 392	/ 687	/ 463	/ 400	/ 200	/ 185
BRAM no RW check	/ 388	/ 660	/ 458	/ 650	/ 285	/ 272
BRAM RW check	/ 339	/ 575	/ 400	/ 455	/ 240	/ 214

expected for a typical, high-performance project implementation for each device. In Table IV, it can be seen that the maximum frequencies obtained for RingNet modules are comparable with or higher than those given for DSPs and BRAMs. However, the results are generated for the required clock frequency set to 1 MHz. If the required frequency is tuned, the Synplify synthesizer can optimize modules using techniques like register replication, and higher maximum frequencies can be obtained, gaining several dozens of megahertz on an average. The obtained maximum clock frequencies are estimations, and actual frequency values should be generated by implementation tools provided by a device vendor such as Vivado from Xilinx (see Section IX).

Resource utilization for basic RingNet modules and rings is reported in Table IV. The utilization of FFs and LUTs is presented.

The utilization of FFs in different devices is almost the same with the maximum deviation from the average module size of 2%.

In Table IV, two types of utilized LUTs are distinguished: LUTs utilized for logic implementation and LUTs utilized as RAM. The number of LUTs utilized as logic is 1.8 times higher in the Lattice device than in the devices from Xilinx and Intel, on average. The reason for the disproportion is because Xilinx and Intel use six-input LUTs in their products and Lattice uses smaller, four-input LUTs. Among Xilinx and Intel devices, the maximum deviation in the number of LUTs utilized as logic is 19% among all synthesized RingNet rings. Differences in LUT utilization between Xilinx and Intel FPGAs are the result of minor differences in the architectures used by each vendor, such as different multiplexing hardware at inputs and outputs of the LUTs.

The number of LUTs utilized as RAM differs significantly between vendors. It is a result of different numbers of LUTRAM bits available per utilized LUT; on average, 32 bits per LUT for Intel devices, 48 for Xilinx, and 10.7 for devices from Lattice (see Section I). Hence, the expected ratio between the LUTs utilized as RAM in Xilinx, Intel, and Lattice FPGAs is 1:1.5:4.5. The obtained average ratio for RingNet rings is 1:1.8:4, so it is close to the expected ratio.

The average ratio between the number of utilized FFs and the number of utilized LUTs is equal to 2.2:1 for RingNet rings synthesized for Intel and Xilinx devices and 1:1 for Lattice. Those ratios are close to the ratios of FFs and LUTs available in the devices (see Table I) that we find desirable for FPGA design.

For most of the NoCs, a significant part of FPGA resources is utilized for buffers. It is also true for RingNet, where buffers are implemented using LUTRAM, whereas RAM-capable LUTs are a limited resource (see Table I). As for any limited resource, using less LUTRAM can make routing of synthesized design easier and provide shorter links and higher clock frequency. We checked what share of the LUTs utilized for RingNet rings is utilized for RAM. This share happens to be constant for a given FPGA vendor: about 33% for rings synthesized for FPGAs from Xilinx, 55% for Intel, and 54% for ECP5 from Lattice. The share of utilized RAM-capable LUTs exceeds the share of available RAM-capable LUTs, but only by 4% and 5% in the case of Lattice and Intel, respectively. This shows reasonable LUTRAM utilization, suitable for FPGA implementation.

Through the presented synthesis results, it is shown that RingNet modules can be efficiently implemented in devices of various types from different vendors, with comparable resource utilization and a high value of the maximum clock frequency.

In Table IV, the results for the state-of-the-art 64-bit flits switches are presented in [3] and [4]. The five-port switch proposed in [3] is reported to utilize 1678 LUTs per PE when implemented in the Xilinx Kintex UltraScale device.

The CONNECT network using three-port switches and configured in ring topology [4] is reported to utilize 430 LUTs per PE when implemented in Xilinx Virtex-6 LX760. On the other hand, RingNet utilizes only 367 LUTs per PE if 15×1 ring is used. It is clear that three-port switches used in RingNet and CONNECT utilize fewer LUTs than the five-port switch in [3]. In RingNet ring, unlike in [3] and [4], some network elements, i.e., SGs and L2R manager, are common to the number of connected PEs. The cost of the elements is shared among the number of the connected PEs, which decreases the resources utilized per PE. It is the reason for fewer LUTs utilized per PE in RingNet when compared with CONNECT. It needs to be pointed out that the shared L2R manager used in RingNet not only lowers resource utilization but also increases latency, which is discussed in Section VII. On the other hand, the L2R manager has knowledge about the required load; therefore, it can provide a fair access for all its associated PEs.

When implemented in the devices of the same speed class, the RingNet rings can be clocked with higher clock frequency than switches proposed in [3], as well as the CONNECT network. This feature is achieved by the use of small and optimized three-port switches in RingNet. For example, for the Kintex UltraScale Xilinx devices, the maximum clock frequency is above 700 and about 470 MHz for RingNet and the network of switches [3], respectively (cf. Table IV). Another example shows that for a Virtex6 device, the maximum frequency for CONNECT network is about 31%–59% of the maximum DSP frequency, whereas it is 84% to 107% of the maximum DSP frequency for a RingNet ring.

The RingNet network utilizes less resource than the NoCs from [3] and [4], but it comes at the expense of increased latency (see Section VII). On the other hand, the high latency can be mitigated with a high maximum clock frequency of RingNet.

IX. Comparison between RingNet Ring and AXI4 Interconnect

We compare RingNet with AXI4, a widely used communication infrastructure for FPGAs. Both have common features. According to [32], AXI4 is designed for high-performance memory-mapped requirements, just like RingNet. Both use packets with a single address flit and separate write and read channels. In contrast to RingNet, AXI4 supports packets of different sizes up to 256 flits and various data widths from 32 up to 1024 bits.

AXI4 connects memory-mapped devices using AXI Interconnect. It is built of a crossbar, a data FIFO used for buffering, pipeline FFs used to break a critical timing path, and an address range decoder. AXI Interconnect is available as a standalone IP in the Xilinx IP Catalog.

We configured a RingNet ring and AXI4 Interconnect to have similar features and compared the implementations in terms of resource utilization and the maximum clock frequency. The configurations are described in Table V.

The implementation results were obtained just for a sample FPGA device, namely, Artix7 (xc7a100tcsg324-1). The synthesis discussed in Section VIII already showed comparable results of RingNet for different types of FPGA. Therefore, the conclusions from the implementation can also be extended to the other FPGAs.

TABLE V
RingNet Ring and AXI4 Interconnect Configuration

Parameter	RingNet ring	AXI4 Interconnect
Address width	37 bits	37 bits
Data width	64 bits (single ring)	64 bits
Number and types of available interfaces	*N*×1: - 1 RI for connecting another ring or a memory device - *N* (2 ÷ 15) of NIs for connecting PEs	*N*×1: - 1 slave interface for connecting memory device, - *N* (2 ÷ 15) of master interfaces for connecting PEs
Arbiter	L2R Manager	Round-Robin arbiter
Buffering	LUTRAM-based fifos	
Performance optimization	---	Crossbar in performance optimized version named Shared-Address, Multiple-Data (SAMD); Use of pipelining FFs called AXI Register Slice
Data enable	Enable bit for each transmitted data byte.	

TABLE VI
Utilized Resources and the Maximum Frequency of RingNet Ring and AXI4 Interconnect Implementations

		LUTs utilized	FFs utilized	Max freq. [MHz]
AXI4 Interc.	2×1	1370 (22% as RAM)	2801	268
	4×1	2205 (23% as RAM)	4480	226
	6×1	3175 (22% as RAM)	6151	192
	15×1	7181 (23% as RAM)	13650	151
RingNet ring	2×1	1185 (44% as RAM)	2047	396
	4×1	1846 (43% as RAM)	3188	392
	6×1	2557 (44% as RAM)	4343	382
	15×1	5650 (43% as RAM)	9595	365

AXI4 Interconnect was configured using the AXI Interconnect RTL 1.7 generator from Vivado 2016.4. The implementations were performed using Vivado 2016.4.

The goal of the implementations was to estimate the maximum clock frequency. In order to find it, multiple implementations were performed for the requested clock frequencies changed with the resolution of 1 MHz. The results obtained for the maximum obtained frequencies are presented in Table VI.

One can compare the results obtained for the implementation and synthesis of RingNet rings (see Table IV for Artix7 and Table VI). The differences in the number of utilized LUTs reported in both tables are minor. The maximum clock frequency and the number of utilized FFs are higher for the implementation by 12% and 34% on average, respectively. Those increases are expected and are the result of register replication enabled during the implementation.

From Table VI, one can conclude that AXI4 Interconnect does not scale well and the maximum clock frequency decreases rapidly for a growing size of the AXI4 crossbar. On the other hand, RingNet, due to its optimized three-port switches and ring topology, provides high maximum clock frequency across a wide range of network sizes. For the corresponding configurations, RingNet supports higher frequencies

TABLE VII
RESOURCES UTILIZED FOR RINGNET RING AND AXI4 INTERCONNECT IMPLEMENTATIONS

	Resources added per PE (R_{PE})		Resources utilized for core elements (R_{core})	
	LUTs	FFs	LUTs	FFs
AXI4 Interc.	449	834	473	1133
RingNet ring	344	581	497	884

than AXI4 Interconnect. The increase in the maximum clock frequency is from 48% (for the 2×1 configuration) up to 142% (for the 15×1 configuration).

For all the cases RingNet ring requires less resource than AXI4 Interconnect. The FF utilization is about 40% lower for all configurations. The reduction in the number of used LUTs ranges from 16% (for the 2×1 configuration) to 27% (for the 15×1 configuration). Despite utilizing less LUTs, a RingNet ring utilizes more LUTs as RAM than a corresponding AXI4 Interconnect, providing more buffer space.

Both RingNet rings and AXI4 Interconnect utilize a part of the resources for core elements that are shared between a number of attached PEs. Those core elements for RingNet ring are: L2R manager, SGs, and RI, whereas the core elements of AXI4 Interconnect are the round-robin arbiter and a slave interface. For both architectures, the numbers of utilized LUTs and FFs follow the equation:

$$R_{\text{Total}} = p \cdot R_{\text{PE}} + R_{\text{core}} \tag{3}$$

where p is the number of connected PEs, R_{PE} is the number of LUTs or FFs utilized per PE, and R_{core} is a constant number of LUTs or FFs utilized for the core elements. The parameters R_{PE} and R_{core} can be calculated for LUTs and FFs based on the results presented in Table VI by using linear regression. Results of the regression are presented in Table VII.

For other state-of-the-art NoCs in [3] and [4], one switch is added per PE; therefore, the utilization of resources is proportional to the number of PEs. For interconnects like RingNet or AXI4, core elements are used, and their cost is shared between all the connected PEs. As already stated in Section VIII, this approach can reduce the overall resource utilization of a network.

X. CONCLUSION

In this paper, the problems related to the design of NoCs on FPGAs are considered in the context of the typical features of FPGAs manufactured by leading vendors. As a result of these considerations, the RingNet architecture and communication protocol are proposed with the aim to exploit the specific potential of FPGA devices as much as possible. Although few FPGA-oriented NoCs have been proposed so far [3], [4], [19], [31], [36]–[43], the RingNet design is likely the most determinedly adopted to the typical FPGA resources and their architectures. The specific features of RingNet include: communication exclusively through system memory (large SDRAM or block RAM), control over the traffic load executed by the PEs, FPGA-optimized three-port switches organized into the tree-of-rings topology, distributed memory (LUTRAM) used as small buffers in the switches, and virtual cut-through switching. In this paper, it is demonstrated that RingNet features guaranteed throughput, predictable latency, and fair network access. In this paper, the simulation results demonstrate that the RingNet implementations are efficient in terms of the maximum clock frequency and resource consumption for flagship FPGA devices from major manufacturers. As compared to the widely accepted state-of-the-art interconnection architecture AXI4, the implementations demonstrate higher maximum clock frequency and lower resource consumption in RingNet. Therefore, the authors believe that the RingNet NoC architecture and protocol may be widely adopted in FPGA-based SoC designs, especially in high-volume data processing applications, e.g., in video processing.

In complex SoCs, it may be desired to establish clock domains with individual clock frequencies, e.g., with the aim of optimizing power dissipation. We think that the RingNet ring is a natural candidate for grouping PEs with the similar maximum clock frequency, or more generally, in sections with common power management. Nevertheless, the usage of rings in separate clock domains requires methods for interdomain transitions. Considerations on such methods for RingNet would be an interesting direction for future research, in particular, when searching for techniques that would be efficient for several families of FPGA devices.

ACKNOWLEDGMENT

The authors would like to thank Synopsys Inc., for providing Synplify synthesizer for experimentation.

REFERENCES

[1] K. A. Helal, S. Attia, T. Ismail, and H. Mostafa, "Comparative review of NoCs in the context of ASICs and FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Lisbon, Portugal, May 2015, pp. 1866–1869.
[2] A. Łuczak *et al.*, "Network-on-multi-chip (NoMC) with monitoring and debugging support," *J. Telecommun. Inf. Technol.*, no. 3, p. 81–86, 2011.
[3] P. Maidee and A. Kaviani, "Improving FPGA NoC performance using virtual cut-through switching technique," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Mexico City, Mexico, 2015, pp. 1–6.
[4] M. K. Papamichael and J. C. Hoe, "The CONNECT network-on-chip generator," *Computer*, vol. 48, no. 12, pp. 72–79, Dec. 2015.
[5] *Stratix 10 GX/SX Device Overview, S10 Overview*, Intel, Santa Clara, CA, USA, Oct. 2016.
[6] *Arria 10 Device Overview, A10 Overwiev*, Altera, San Jose, CA, USA, Oct. 2016.
[7] *Intel Cyclone 10 GX Device Overview, C10GX51001*, Intel, Santa Clara, CA, USA, Nov. 2017.
[8] *Stratix V Device Overview, SV51001*, Altera, Santa Clara, CA, USA, Oct. 2015.
[9] *Devices: 28nm Device Portfolio*, Altera Product Catalog, Altera, Santa Clara, CA, USA, 2015.
[10] *Zynq UltraScale + MPSoC, XMP104, Version 2.1*, Xilinx, Santa Clara, CA, USA, 2016.
[11] *UltraScale + FPGAs, XMP103, Version 1.10*, Xilinx, Santa Clara, CA, USA, 2017.
[12] *UltraScale FPGA, XMP102, Version 1.7*, Xilinx, Santa Clara, CA, USA, 2016.
[13] *All Programmable 7 Series, XMP101, Version 1.2*, Xilinx, Santa Clara, CA, USA, 2016.
[14] *Cost-Optimized Portfolio, XMP100, Version 1.6*, Xilinx, Santa Clara, CA, USA, 2016.
[15] *ECP5 and ECP5-5G Family, DS1044, Version 1.6*, Lattice Semiconductor, Hillsboro, OR, USA, Feb. 2016.
[16] *LatticeECP3 Family Data Sheet, DS1021, Version 02.8EA*, Lattice Semiconductor, Hillsboro, OR, USA, Mar. 2015.
[17] *LatticeECP2/M Family Data Sheet, DS1006, Version 04.1*, Lattice Semiconductor, Hillsboro, OR, USA, Sep. 2013.

[18] *Stratix V Device Handbook, SV51002, Version 1.3*, vol. 2, Altera, Santa Clara, CA, USA, Nov. 2011.
[19] J. Rettkowski and D. Göhringer, "RAR-NoC: A reconfigurable and adaptive routable Network-on-Chip for FPGA-based multiprocessor systems," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Cancun, Mexico, 2014, pp. 1–6.
[20] D. Bertozzi and L. Benini, "Xpipes: A network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits Syst. Mag.*, vol. 4, no. 2, pp. 18–31, Sep. 2004.
[21] S. Murali *et al.*, "Synthesis of predictable networks-on-chip-based interconnect architectures for chip multiprocessors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 8, pp. 869–880, Aug. 2007.
[22] S. Liu, A. Jantsch, and Z. Lu, "A fair and maximal allocator for single-cycle on-chip homogeneous resource allocation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 10, pp. 2230–2234, Oct. 2014.
[23] T. Wehbe and X. Wang, "Secure and dependable NoC-connected systems on an FPGA chip," *IEEE Trans. Rel.*, vol. 65, no. 4, pp. 1852–1863, Dec. 2016.
[24] E. Todorovich, M. Leonetti, and R. Brinks, "An advanced NoC with debug services on FPGA," in *Proc. Southern Conf. Program. Logic (SPL)*, Buenos Aires, Argentina, 2014, pp. 1–6.
[25] M. Zhu, Y. Jiang, M. Yang, and L. De Luna, "A scalable parameterized NoC emulator built upon Xilinx Virtex-7 FPGA," in *Proc. Int. Conf. Syst. Eng. (ICSEng)*, Las Vegas, NV, USA, 2017, pp. 287–290.
[26] H. M. Kamali, K. Z. Azar, and S. Hessabi, "DuCNoC: A high-throughput FPGA-based NoC simulator using dual-clock light-weight router micro-architecture," *IEEE Trans. Comput.*, vol. 67, no. 2, pp. 208–221, Feb. 2018.
[27] Y. J. Yoon, N. Concer, M. Petracca, and L. P. Carloni, "Virtual channels and multiple physical networks: Two alternatives to improve NoC performance," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 32, no. 12, pp. 1906–1919, Dec. 2013.
[28] Y. H. Song and T. M. Pinkston, "A progressive approach to handling message-dependent deadlock in parallel computer systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 3, pp. 259–275, Mar. 2003.
[29] M. A. Al Faruque, T. Ebi, and J. Henkel, "AdNoC: Runtime adaptive network-on-chip architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 2, pp. 257–269, Feb. 2012.
[30] J. Postman, T. Krishna, C. Edmonds, L.-S. Peh, and P. Chiang, "SWIFT: A low-power network-on-chip implementing the token flow control router architecture with swing-reduced interconnects," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 8, pp. 1432–1446, Aug. 2013.
[31] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-examining conventional wisdom for designing NoCs in the context of FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, New York, NY, USA, 2012, pp. 37–46.
[32] *Vivado AXI Reference Guide, UG1037, Version 3.0*, Xilinx, Santa Clara, CA, USA, Jun. 2015.
[33] M. S. Abdelfattah and V. Betz, "Power analysis of embedded NoCs on FPGAs and comparison with custom buses," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 1, pp. 165–177, Jan. 2016.
[34] M. S. Abdelfattah and V. Betz, "The case for embedded networks on chip on field-programmable gate arrays," *IEEE Micro*, vol. 34, no. 1, pp. 80–89, Jan./Feb. 2014.
[35] M. S. Abdelfattah, A. Bitar, and V. Betz, "Design and applications for embedded networks-on-chip on FPGAs," *IEEE Trans. Comput.*, vol. 66, no. 6, pp. 1008–1021, Jun. 2017.
[36] N. Kapre and J. Gray, "Hoplite: Building austere overlay NoCs for FPGAs," in *Proc. Int. Conf. Field Program. Logic Appl. (FPL)*, London, U.K., 2015, pp. 1–8.
[37] S. Wasly, R. Pellizzoni, and N. Kapre, "HopliteRT: An efficient FPGA NoC for real-time applications," in *Proc. Int. Conf. Field Program. Technol. (ICFPT)*, Melbourne, VIC, Australia, 2017, pp. 64–71.
[38] K. Vipin, J. Gray, and N. Kapre, "Enabling partial reconfiguration and low latency routing using segmented FPGA NoCs," in *Proc. Int. Conf. Field Program. Logic Appl. (FPL)*, Ghent, Belgium, 2017, pp. 1–8.
[39] N. Kapre, "On bit-serial NoCs for FPGAs," in *Proc. Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Napa, CA, USA, 2017, pp. 32–39.
[40] N. Kapre, "Implementing FPGA overlay NoCs using the Xilinx UltraScale memory cascades," in *Proc. IEEE Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Napa, CA, USA, Apr./May 2017, pp. 40–47.
[41] N. Kapre, "Deflection-routed butterfly fat trees on FPGAs," in *Proc. Int. Conf. Field Program. Logic Appl. (FPL)*, Ghent, Belgium, 2017, pp. 1–8.
[42] S. N. Shelke and P. B. Patil, "Low-latency, low-area overhead and high throughput NoC architecture for FPGA based computing system," in *Proc. Int. Conf. Electron. Syst., Signal Process. Comput. Technol.*, Nagpur, India, 2014, pp. 53–57.
[43] P. Maidee, A. Kaviani, and K. Zeng, "LinkBlaze: Efficient global data movement for FPGAs," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Cancun, Mexico, 2017, pp. 1–8.
[44] W. Snyder. *Introduction to Verilator*. Accessed: Dec. 19, 2017. [Online]. Available:. [Online]. Available: https://www.veripool.org/wiki/verilator
[45] R. Kumar and A. Gordon-Ross, "MACS: A highly customizable low-latency communication architecture," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 237–249, Jan. 2016.
[46] M. Domański *et al.*, "Fast depth estimation on mobile platforms and FPGA devices," in *Proc. 3DTV-Conf., True Vis.-Capture, Transmiss. Display 3D Video (DTV-CON)*, Lisbon, Portugal, 2015, pp. 1–4.
[47] *Synopsys FPGA Synthesis User Guide*, Synopsys, Mountain View, CA, USA, Apr. 2017.
[48] M. S. Abdelfattah and V. Betz, "LYNX: CAD for FPGA-based networks-on-chip," in *Proc. 26th Int. Conf. Field Program. Logic Appl. (FPL)*, Lausanne, Switzerland, 2016, pp. 1–10.
[49] S. Abba and J.-A. Lee, "A parametric-based performance evaluation and design trade-offs for interconnect architectures using FPGAs for networks-on-chip," *Microprocessors Microsyst.*, vol. 38, no. 5, pp. 375–398, Jul. 2014.

Jakub Siast received the M.Sc. degree from the Faculty of Electronics and Telecommunications, Poznań University of Technology, Poznań, Poland, in 2009, where he is currently working toward the Ph.D. degree.

He has coauthored several papers related to hardware implementation in field-programmable gate array (FPGA), 3-D video coding, and depth estimation. His current research interests include FPGA and microprocessor architecture design.

Adam Łuczak received the M.Sc. and Ph.D. degrees from the Poznań University of Technology, Poznań, Poland, in 1997 and 2001, respectively.

His current research interests include video coders control and hardware implementations (FPGA) of digital signal processing algorithms.

Dr. Łuczak is a member of the Polish Society Theoretical and Applied Electrical Engineering. He was a recipient of the Annual Fellowship for Young Scientist from the Foundation for Polish Science and a Group Award of the Minister of the National Education for research in the National Education for research in image compression.

Marek Domański (M'91) received the M.S., Ph.D., and Habilitation degrees from the Poznań University of Technology, Poznań, Poland, in 1978, 1983, and 1990, respectively.

In 2004, he led the team that developed one of the very first AVC decoders for TV set-top boxes and various AVC, HEVC, and AAC HE codec implementations and improvements. Since 1993, he has been a Professor and leads the Chair (Department) of Multimedia Telecommunications and Microelectronics, Poznań University of Technology. He has coauthored highly ranked technology proposals submitted in response to MPEG calls for scalable video compression in 2004 and 3-D video coding in 2011. He has authored three books. He holds 14 patents and authored/coauthored over 300 research papers, mostly on image, video and audio compression, virtual navigation, free-viewpoint television, image processing, multimedia systems, 3-D video and color image technology, digital filters, and multidimensional signal processing.

Dr. Domański served as a member for various steering, program, and editorial committees of international journals and international conferences. He was the General Chairman/Co-Chairman and the Host of several international conferences: Picture Coding Symposium, PCS 2012; IEEE International Conference on Advanced and Signal-Based Surveillance, AVSS 2013, European Signal Processing Conference, EUSIPCO 2007; 73rd and 112th Meetings of MPEG, 69th JPEG Meeting and others.