HiPerNoC: A High-Performance Network-on-Chip for Flexible and Scalable FPGA-Based SmartNICs

Klajd Zyla, Marco Liess, Thomas Wild, Andreas Herkersdorf
TUM School of Computation, Information and Technology, Technical University of Munich
Munich, Germany
{klajd.zyla, marco.liess, thomas.wild, herkersdorf}@tum.de

***Abstract*—A recent approach that the research community has proposed to address the steep growth of network traffic and the attendant rise in computing demands is in-network computing. This paradigm shift is bringing about an increase in the types of computations performed by network devices. Consequently, processing demands are becoming more varied, requiring flexible packet-processing architectures. State-of-the-art switch-based smart network interface cards (SmartNICs) provide high versatility without sacrificing performance but do not scale well concerning resource usage. In this paper, we introduce HiPerNoC—a flexible and scalable field-programmable gate array (FPGA)-based SmartNIC architecture deploying a 2D-mesh network-on-chip (NoC) with a novel router design to manage network traffic with diverse processing demands. The NoC can forward incoming network packets to the available processing engines in the required sequence at a traffic load of up to 91.1 Gbit/s (0.89 flit/node/cycle). Each router applies distributed switch allocation and avoids head-of-line blocking by deploying queues at the switch crosspoints of input-output connections used by the routing algorithm. It also prevents deadlocks by employing non-blocking virtual cut-through switching. We implemented a prototype of HiPerNoC as a 4x4 2D-mesh NoC in SystemVerilog and evaluated it with synthetic network traffic via cycle-accurate register-transfer level simulations in Vivado. The evaluation results show that HiPerNoC achieves up to 53 % higher saturation throughput, occupies 53 % fewer lookup tables and block RAMs, and consumes 16 % less power on an Alveo U55C than ProNoC—a state-of-the-art FPGA-based NoC.**

***Index Terms*—On-chip interconnect, NoC, FPGA, SmartNIC, In-network computing**

I. Introduction

The expansion of the interconnected world is being accompanied by a growing array of online services (e.g., video conferencing, remote control, extended reality, and mobile applications). This phenomenon is bringing about a significant surge in the volume of network traffic, placing higher processing demands on data centers and posing challenges in meeting throughput requirements and round-trip delay bounds. However, CPU performance is limited, mainly due to the high power consumption and memory-access latency [1]. A recent approach that addresses this issue is in-network computing [2]—a paradigm shift where computational tasks are performed within the network infrastructure, transcending traditional server-based computing models. In-network computing minimizes data movement, reduces latency, and boosts system performance by offloading computational tasks to network devices. For this purpose, network devices (e.g., smart network interface cards (SmartNICs), switches, and routers) are equipped with specialized hardware to accelerate networking, storage, and security functions [3], enhancing their conventional data-forwarding role. Field-programmable gate arrays (FPGAs) play a critical role as a target technology by enabling quick and low-cost implementation of new applications in hardware.

Since network devices can execute an increasing number of tasks, the processing requirements of incoming traffic are becoming more diverse, demanding flexible packet-processing architectures. Some approaches in the literature reduce the effort that developers have to put into implementing network applications by providing frameworks that allow them to program at higher abstraction layers (e.g., match-action tables (MATs) [4], extended finite state machines [5], and Linux's eXpress Data Path [6]). These approaches provide more freedom than specialized SmartNICs. However, they usually deliver low performance for complex computations. Some researchers address this issue by proposing SmartNIC architectures that implement traditional protocol stacks in hardware (e.g., transport protocols [7], TCP/IP stack [8], and RDMA [9]). Such architectures deploy programmable hardware accelerators to reduce the processing time for resource-intensive tasks (e.g., congestion control, I/O virtualization, encryption, and compression [3]) while providing limited flexibility. These accelerators are typically arranged in the order in which their tasks are usually performed. Although this approach is suitable when all packets demand the same task sequence, it does not support network traffic with varied processing requirements.

In order to tackle these challenges, we propose HiPerNoC—a flexible and scalable FPGA-based SmartNIC architecture that deploys a 2D-mesh network-on-chip (NoC) with a novel router design to handle network traffic with diverse processing requirements. Our design contains several processing engines attached to the NoC routers as local nodes. The NoC forwards network packets to the required processing engines in the specified sequence, enabling the execution of various applications. Each router contains per-input demultiplexers, which forward incoming packets to the queues located between the respective input ports and the output ports connected to the possible next hops, and per-output multiplexers, which resolve output port contentions. HiPerNoC avoids head-of-line (HOL) blocking by deploying a separate queue for each input-output connection

We acknowledge the financial support from the Bavarian Ministry of Economic Affairs, Regional Development and Energy in the context of the project "6G Future Lab Bavaria".

978-3-9826741-0-0/DATE25/© 2025 EDAA

used by the routing algorithm in every router. It also prevents deadlocks by applying non-blocking packet-level flow control (at the cost of potential packet loss in high-load conditions).

We implemented a prototype of HiPerNoC as a 4x4 2D-mesh NoC in synthesizable hardware description language (HDL) code. We evaluated it with uniform random, exponential, and bit-complement traffic via register-transfer level (RTL) simulations in Vivado. The evaluation results demonstrate that HiPerNoC can forward IP packets of variable size to the deployed processing engines in the specified sequence at a traffic load of up to $91.1\,\mathrm{Gbit/s}$ ($0.89\,\mathrm{flit/node/cycle}$). Furthermore, it achieves up to $53\,\%$ higher peak throughput, uses $53\,\%$ fewer lookup tables (LUTs) and block RAMs (BRAMs), and consumes $16\,\%$ less power on an Alveo U55C High Performance Compute Card [10] than ProNoC [11]—a state-of-the-art, open-source FPGA-optimized NoC that supports advanced NoC features.

Section II briefly describes two state-of-the-art flexible SmartNIC designs, introduces some basic NoC concepts, and describes the main contributions of ProNoC. Section III describes the main components of the proposed design and compares it with existing approaches. Section IV describes the simulation setup used to evaluate the design and shows and discusses the evaluation results, including resource usage and power consumption. Finally, the main contributions of this paper are summarized in section V.

II. Background and Related Work

In this section, we briefly describe two state-of-the-art switch-based SmartNIC architectures and point out their limitations. Moreover, we explain some basic NoC concepts and summarize the router architecture optimizations employed in ProNoC [11]—a state-of-the-art FPGA-based NoC.

A. Switch-Based SmartNIC Architectures

PANIC [12] is a versatile SmartNIC architecture that supports packet streams (flows) with diverse processing demands. It uses a combined input- and output-queued crossbar switch [13] to forward network packets to the available processing units in an arbitrary order. A MAT-based parser determines to which computing units the switch must forward incoming packets and in which sequence based on the associated flows. A centralized scheduler privileges high-priority packets in high-load conditions and provides load-aware packet steering. The main bottleneck of PANIC is the interconnect. The deployed switch has no virtual output queues and operates without speedup, leading to HOL blocking, which causes a low saturation throughput of $48.3\,\%$ for uniform random traffic [14]. It also exhibits quadratic scalability concerning the switch allocation complexity and the number of wires.

FlexCross [14] is another flexible packet-processing architecture tailored for high-performance SmartNICs and network switches. It employs a MAT-based parser to derive the task sequence allocated to each packet based on the associated flow and a switch to forward incoming traffic to the required processing engines. In contrast to PANIC, FlexCross has no centralized scheduler, applies distributed switch allocation, and contains queues only at the switch crosspoints. It avoids HOL blocking by deploying a separate queue for each input-output link, thus achieving a maximum throughput of nearly $100\,\%$. The main limitation is the quadratic scalability of the interconnect in terms of the number of wires and queues.

B. NoC Concepts

NoCs are already well established in the research community as a scalable alternative to crossbar switches. We introduce some NoC features to help readers unfamiliar with this research topic understand the rest of this paper.

1) Topology: Topology is the way routers in a NoC are arranged and connected. The most commonly used topology is the 2D mesh, which utilizes low-radix routers connected to their neighbors via bidirectional links. The benefits are short wires and low switch allocation complexity, but the downside is the high mean hop count. Popular alternative topologies that reduce the hop count are the concentrated mesh [15], the flattened butterfly [16], and Multidrop Express Channels [17].

2) Virtual Channels: Virtual channels (VCs) provide the abstraction of multiple logical channels over an underlying physical channel. NoC routers typically realize VCs by placing multiple queues at each input port to avoid HOL blocking and deadlocks [18]. HOL blocking arises when packets whose output port is busy stall following packets whose output port is free. Deadlocks in NoCs can generally be classified into routing-dependent and message-dependent. Routing-dependent deadlocks occur due to routing cycles formed by packets belonging to the same pair of source-destination nodes. By contrast, message-dependent deadlocks happen due to routing cycles created by packets associated with different pairs of source-destination nodes. VCs increase performance but require additional logic (VC allocation) and buffer space.

3) Flow Control: Flow control denotes the protocol that manages the transmission of flits between network nodes, dealing with channel bandwidth and buffer space allocation. The most commonly used protocols are wormhole and virtual cut-through (VCT) switching. In wormhole switching, routers make transfer decisions at the level of flits. The transmission is stopped if the allocated buffer in the downstream router is occupied, which can cause congestion. By contrast, in VCT switching, routers make transfer decisions at the level of packets. The packet is sent downstream only if the allocated buffer has enough space, which increases the buffer space requirements but generally leads to higher link utilization. Since there are usually one or multiple pipeline stages between the buffers of adjacent routers when using any protocol, credit counters must generate an ”early full” signal so that there is enough space to store all flits that are underway, which leads to underutilization of buffers [19], [20].

4) Routing algorithm: The routing algorithm is the protocol that computes the next hop in each router based on the destination of each packet. Routing algorithms are usually classified into deterministic and adaptive algorithms. The most prominent deterministic algorithm is XY routing, where each packet is first routed along the X dimension until it reaches the same column as its destination and then along the Y dimension

until it reaches the destination. This minimal routing algorithm causes no cycles in the channel dependency graph for packets belonging to the same pair of source-destination nodes, thus avoiding routing-dependent deadlocks without using additional VCs. The most common partially adaptive routing algorithms follow the turn model [21] (e.g., west-first, north-last, and negative-first routing). These algorithms prohibit certain turns to avoid routing-dependent deadlocks. However, they provide more freedom than deterministic algorithms, allowing routers to consider the network state when computing the next hop. Routers commonly use look-ahead routing to perform route computation for the next hop in parallel with VC allocation to reduce the pipeline delay [22].

5) Switch Allocation: Switch allocation is the procedure that establishes connections between the input and output ports of the router. It is achieved by matching requests from active VCs at the input ports with switch connections to the output ports based on the buffer space availability at the downstream routers. The matching quality has a significant impact on the performance of the switch. A prominent scheduling algorithm that can achieve $100\,\%$ throughput for uniform traffic is iSLIP [23]. In addition to look-ahead routing, routers often use techniques like speculative switch allocation and combined VC and switch allocation to reduce the pipeline delay [13].

C. ProNoC

ProNoC [11] is an open-source FPGA-optimized, VC-based NoC and an automated system integration tool for prototyping NoC-based many-core system-on-chips targeting FPGAs. All VCs located at an input port share the same BRAM, in contrast to CONNECT [24]—another state-of-the-art FPGA-optimized NoC—which uses lookup table RAMs (LUTRAMs). This approach reduces the router's LUT utilization. In order to remove the multiplexing time of the "full" signals generated by the credit counters located at the router's output side from the critical path, ProNoC sends 2-bit status data for each output VC credit counter to the input side. This information is used to predict if the assigned output VC will be full in the next clock cycle. ProNoC reduces the router's cost by removing switch connections between inputs and outputs of the same port. It employs non-speculative combined VC and switch allocation by performing the VC allocation only for requests that have received the grant from the switch allocation. This method reduces the matching complexity and enables the usage of the result from the first stage of the switch allocation in the VC allocation. ProNoC uses a static straight allocator to forward flits requiring no direction change in one clock cycle when there are no output port contentions. In contrast to predictive mechanisms, this method does not require misprediction correction because it performs VC and switch allocation at flit arrival time. ProNoC employs an optimized deadlock-free flow control scheme for fully adaptive routing that allows packets to move from escape VCs to adaptive VCs and non-atomic VC reallocation for $80\,\%$ of all VCs. ProNoC achieves $86\,\%$ higher peak throughput and $28\,\%$ lower logic cell utilization than CONNECT. However, it saturates at a throughput of about $28\,\%$ for uniform random traffic and $20\,\%$ for bit-reversal traffic.

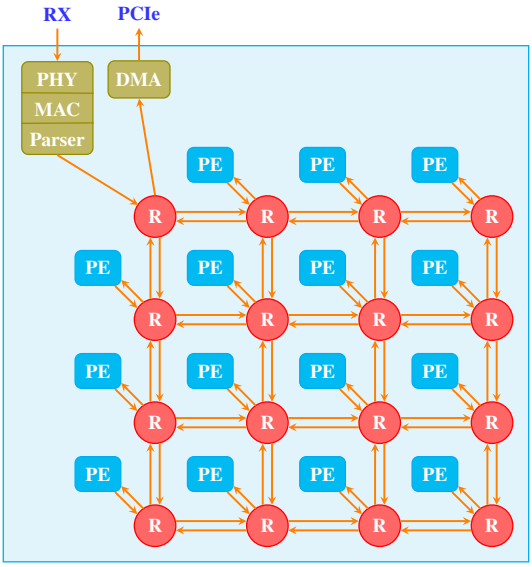


Fig. 1: Block diagram of the architecture of HiPerNoC. RX = receiver, PCIe = peripheral component interconnect express, PHY = physical layer, MAC = media access control, DMA = direct memory access, R = router, PE = processing engine.

III. Proposed Architecture

In order to address the shortcomings of state-of-the-art switch-based SmartNIC designs and FPGA-based NoCs, we propose HiPerNoC—an FPGA-based SmartNIC architecture deploying a high-performance NoC for flexible and scalable network packet processing.

A. NoC-Based SmartNIC Architecture

Fig. 1 depicts the architecture of HiPerNoC. All modules exchange data via the AXI4-Stream protocol [25]. The total flit/link width is $707\,\mathrm{bit}$, including packet data, metadata, and protocol signaling. HiPerNoC has a data width of $512\,\mathrm{bit}$ and operates at a frequency of $200\,\mathrm{MHz}$, thus achieving a bandwidth of $102.4\,\mathrm{Gbit/s}$. We chose a wide data path to exploit the FPGA's abundance of wires [24]. We arranged the routers in a 2D mesh because this topology requires simple switch allocation logic and few queues, which is convenient for FPGA prototypes. The size of the NoC is reconfigurable. We opted for a 4x4 network due to the limited number of available BRAMs, which we use to implement the queues.

Each router is connected to a processing engine (PE), except for the one located at the top-left corner of the NoC, which has an input port connected to the parser and an output port connected to the DMA engine. HiPerNoC receives IP packets from the network via Ethernet and sends them to the host via PCIe after executing the required tasks. The parser extracts the flow type and packet size from each packet header, looks up the PE sequence mapped to the flow type, and encodes all this information as metadata. The metadata are transferred

along with the data stream as user-defined sideband information via the TUSER signal of the AXI4-Stream protocol. The NoC receives packets from the parser and forwards them to the required PEs in the specified sequence.

We deploy five PE types (tasks) that process the packet header in our prototype: checksum verification (CRC) [26], firewall, network address translation (NAT), packet forwarding, and load balancing. We instantiate each PE type three times to connect each router to a local node. Each PE performs its task in the first pipeline stage in one clock cycle, like a MAT [27]. In the second pipeline stage, it shifts the metadata field "required PE sequence" such that the bits that encode the current PE are removed, and the bits that represent the next required PE are pushed to the front. The total processing delay is two clock cycles.

B. NoC Router Architecture

Fig. 2 represents the router architecture. We take a different approach from state-of-the-art NoCs (e.g., ProNoC [11] and CONNECT [24]) regarding VCs, flow control, and switch allocation. Instead of using VCs, we deploy first-in, first-out (FIFO) queues in each router/switch only at the crosspoints of input-output connections used by the routing algorithm. This design choice avoids HOL blocking and decouples the switch's input side from the output side, simplifying the switch allocation. Additionally, we employ non-blocking VCT switching to prevent deadlocks without using additional queues.

When a header flit (along with the metadata) arrives at an input port, the route computation (RC) is performed in the first pipeline stage based on its destination (next required PE or DMA engine) and the routing algorithm. In the second pipeline stage, the packet size is compared against the free buffer space of the queue connecting the respective input port to the output port linked to the selected next hop. If there is enough space to store the whole packet, the respective demultiplexer (DEMUX) forwards all flits to the queue in subsequent clock cycles. Otherwise, all flits are discarded to avoid eventual backpressure, which can lead to congestion propagation and deadlocks at high traffic loads [28], [29]. The consequences of packet loss can be mitigated using reliable end-to-end network protocols that resend lost packets (e.g., TCP). Each FIFO queue is mapped to 10 BRAMs with a size of 36 kbit each to cover the total flit width and has two pipeline stages at the output [30]. We set the queue depth to 512 flit—the minimum configurable BRAM depth for the UltraScale+ family [31]—to fully utilize the available BRAM capacity and minimize packet loss. At least 21 packets with a size of 1500 B / 24 flit (Ethernet's maximum transmission unit) fit in a queue.

Each multiplexer (MUX) fetches flits from the queues attached to its inputs and forwards them to the input port of the router/PE to which it is connected. The round-robin arbiter (RRA) resolves contentions for the same output port by selecting the non-empty queue from which the next packet must be read. It finishes the computation in one clock cycle while the MUX forwards the last flit of the recently scheduled packet and writes the result in a register, which the MUX reads in the following cycle. We opted for a RRA because it is simple,

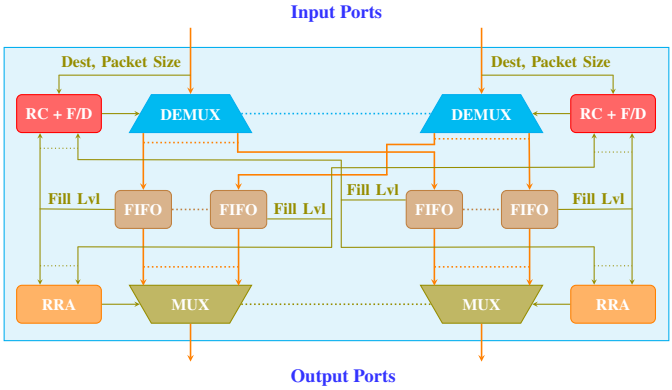


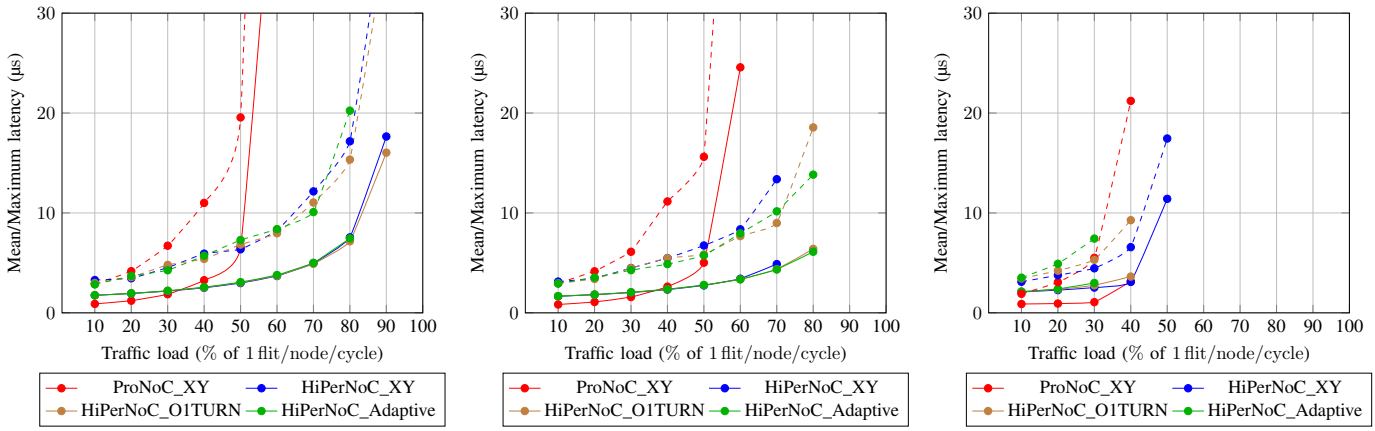
Fig. 2: Block diagram of the router architecture. DEMUX = demultiplexer, MUX = multiplexer, FIFO = first-in, first-out queue, RC = route computation, F/D = forward or discard, RRA = round-robin arbiter.

starvation-free, and widely used in state-of-the-art NoCs. The zero-load routing (+ link traversal) delay is five clock cycles. The pipeline stages are added to help the design meet the timing constraints for a frequency of 200 MHz.

We implemented three routing algorithms for our prototype: XY routing, O1TURN [18], and minimal adaptive routing. O1TURN stands for orthogonal one-turn routing, allowing packets to change direction only once and ensuring they take minimal paths. If two possible paths exist, the router connected to the sending PE chooses the first dimension of traversal based on the fill level of the queues connecting the PE to the two possible next hops. The packet is forwarded to the queue with the lowest fill level. In minimal adaptive routing, the route is computed such that each packet takes a minimal path without limiting the number of direction changes. When two possible next hops exist, the packet is forwarded to the queue with the lowest fill level. We removed switch connections joining inputs and outputs of the same port to reduce the number of wires and queues because minimal routing algorithms do not use these connections [11]. We additionally removed switch connections leading to a change of direction from the Y dimension to the X dimension when using XY routing since this algorithm does not use them [19].

C. Comparison to State-of-the-Art Architectures

HiPerNoC addresses the scalability limitation of switch-based SmartNIC architectures because as the number of PEs increases, the switch allocation complexity remains constant, whereas the number of wires and queues scales linearly (at the cost of higher latency). ProNoC [11] relies on using many VCs, optimal VC allocation, and switch allocation with a high matching quality to prevent HOL blocking and deadlocks. By contrast, HiPerNoC avoids HOL blocking by deploying queues in each router/switch only at the crosspoints of input-output links used by the routing algorithm. It also prevents deadlocks by employing non-blocking VCT switching. Hence, our architecture does not need VCs or VC allocation, and the switch allocation is less complex, which leads to simpler router



(a) Uniform random traffic (b) Exponential traffic (c) Bit-complement traffic

Fig. 3: Load-latency curves of ProNoC and HiPerNoC using three routing algorithms for three typical traffic patterns. The solid lines indicate the respective mean latency, while the dashed lines represent the maximum latency.

logic and lower resource usage. Moreover, HiPerNoC achieves higher saturation throughput than ProNoC (discussed in section IV). The main drawback of HiPerNoC is that it needs large queues to minimize packet loss. However, instead of cascading multiple BRAMs, we fully utilize the available BRAM depth on the FPGA to avoid additional costs.

IV. Experimental Evaluation

We implemented a prototype of HiPerNoC in SystemVerilog and evaluated it with synthetic network traffic via cycle-accurate RTL simulations in Vivado using ProNoC [11] as a baseline. We obtained load-latency curves for three typical traffic patterns: uniform random, exponential, and bit complement. Furthermore, we synthesized and implemented both architectures on an Alveo U55C High Performance Compute Card [10], which features a Virtex XCU55 UltraScale+ FPGA [32]. For a fair evaluation, we configured ProNoC with the same bandwidth ($102.4\,\mathrm{Gbit/s}$) and queue depth ($512\,\mathrm{flit}$) as HiPerNoC. We employed 4 VCs in ProNoC so that it connects at least as many queues as HiPerNoC to each router input port.

A. Simulation Setup

We developed a traffic generator and a traffic sink in SystemVerilog, which interface with the architecture via the AXI4-Stream protocol [25]. The traffic generator injects packets of variable size into the NoC via the input port of the parser. It decides when to generate a packet based on a weighted distribution where the probability of injecting a packet is equal to the specified rate. The size of the generated packet or the number of idle clock cycles (if no packet is injected) is also determined based on a weighted distribution. This method causes the injection rate to fluctuate unpredictably around the specified value. The traffic sink receives packets from the router placed at the top-left corner of the NoC. It measures the throughput, the mean, minimum, and maximum per-packet latency. The throughput is the rate at which the NoC forwards data to the sink. The per-packet latency is the elapsed time from when the header flit enters the parser until it arrives at the sink—including routing, link traversal, and processing delay.

In each simulation run, we injected 52 000 synthetic IP packets into each NoC with a trimodal packet size distribution around $40\,\mathrm{B}$, $1500\,\mathrm{B}$, and $1300\,\mathrm{B}$ at $40\,\%$, $20\,\%$, and $10\,\%$ of packets, respectively [33]. The PE sequence allocated to each packet includes all 15 PEs (1x each)—leading to a traffic load of $1\,\mathrm{flit/node/cycle}$ for an injection rate of $100\,\%$ of the bandwidth ($102.4\,\mathrm{Gbit/s}$)—and is determined by the traffic pattern: uniform random, exponential, or bit complement. Uniform random traffic means an equal distribution of PE sequences, whereas exponential traffic represents an unequal distribution. In bit-complement traffic, each packet visits the PEs in the same sequence while traversing the network bisection when moving between local nodes. We start the measurement when the traffic sink has received 1000 packets and stop it when it has received 51 000 packets so that the NoC is in a steady state. The mean latency did not vary significantly during each simulation run (excluding loads beyond the saturation throughput), justifying the chosen number of injected packets.

B. Simulation Results

Fig. 3 illustrates the mean (solid lines) and the maximum (dashed lines) per-packet latency achieved by ProNoC [11] using XY routing and HiPerNoC using XY routing, O1TURN [18], and minimal adaptive routing for various traffic loads under three typical traffic patterns. We assume in this subsection that both architectures meet the timing requirements for a target frequency of $200\,\mathrm{MHz}$ (discussed in subsection IV-C). We plot only the latency values for traffic loads where the throughput of the respective NoC matches the load. HiPerNoC achieves $17\,\%$–$53\,\%$ higher peak throughput than ProNoC. It performs worse only when using minimal adaptive routing for bit-complement traffic. Moreover, HiPerNoC attains lower mean latency for traffic loads of $40\,\%$ or higher and lower maximum latency for loads of $20\,\%$ or higher for uniform random and exponential traffic. The main reason for the superior performance of

TABLE I: Packet loss rate of HiPerNoC using three routing algorithms (XY, O1TURN, and minimal adaptive) for different traffic loads under three typical traffic patterns

Load (%)	Packet loss rate (%)								
	Uniform random			Exponential			Bit complement		
	XY	O1	Ad	XY	O1	Ad	XY	O1	Ad
10	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	1.9
50	0	0	0	0	0	0	0.5	2.1	13.4
60	0	0	0	0	0	0	8.4	10.6	22.4
70	0	0	0	0	0	0	14.5	18	30.2
80	0	0	0	4.3	0	0	–	–	–
90	0.6	1.1	2.3	10.4	2.4	1.8	–	–	–
100	5.9	6.3	9.6	14.3	7.4	6.9	–	–	–

HiPerNoC is that ProNoC uses additional VCs for deadlock-free routing, which reduces the number of VCs available to reduce HOL blocking. Additionally, outputs of queues located at a router input port are multiplexed to the same switch input in ProNoC. This design choice prevents queued packets arriving via the same port with different next hops from being forwarded simultaneously. However, ProNoC achieves lower mean latency than HiPerNoC for traffic loads of 30 % or lower due to the lower routing delay.

The routing algorithm affects HiPerNoC's performance. XY routing achieves the highest peak throughput for uniform random (91.1 Gbit/s) and bit-complement traffic (51.1 Gbit/s). O1TURN and minimal adaptive routing reach maximum throughputs of 90 Gbit/s and 87.3 Gbit/s, respectively, for uniform random traffic, and 49 Gbit/s and 39.3 Gbit/s for bit-complement traffic. Minimal adaptive routing delivers the best performance for exponential traffic (88.6 Gbit/s). XY routing and O1TURN attain saturation throughputs of 75 Gbit/s and 87.8 Gbit/s, respectively, for the same traffic pattern. Table I shows that the algorithm delivering the highest peak throughput for each traffic pattern leads to the lowest packet loss rate. O1TURN and minimal adaptive routing achieve lower performance than XY routing for uniform random and bit-complement traffic because adaptive algorithms tend to concentrate the traffic in the center of the NoC [34]. The saturation throughput of XY routing for bit-complement traffic approaches the theoretical maximum of 0.5 flit/node/cycle because every packet has to cross the network bisection [18].

C. Resource Usage and Power Consumption

We integrated ProNoC [11] and HiPerNoC into the AMD OpenNIC Shell [35]—an open-source FPGA-based NIC platform. We ran Synthesis and Implementation in Vivado 2022.2, setting the Alveo U55C High Performance Compute Card [10] as the target device. HiPerNoC meets the timing requirements for a bandwidth of 102.4 Gbit/s (512 bit x 200 MHz), while ProNoC achieves a worst negative slack of −3.235 ns. Hence, HiPerNoC can operate at a higher frequency than ProNoC, resulting in more considerable throughput and latency improvements than reported in subsection IV-B. Fig. 4 depicts the share of the available FPGA resources consumed by each

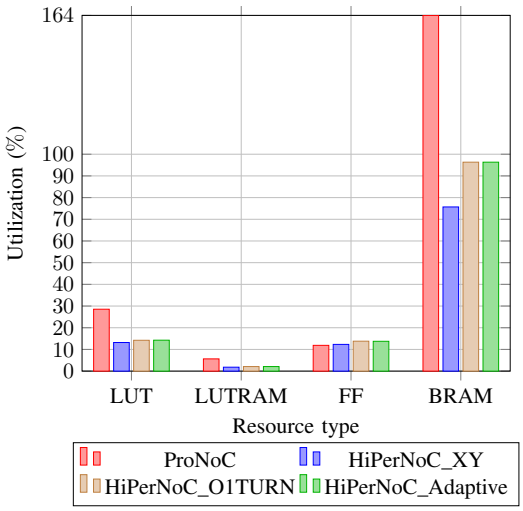


Fig. 4: Resource usage of ProNoC and HiPerNoC on an Alveo U55C. LUT = lookup table, LUTRAM = lookup table RAM, FF = flip-flop, BRAM = block RAM

architecture. An Alveo U55C contains 1 303 680 LUTs, 600 960 LUTRAMs (36.7 Mbit), 2 607 360 flip-flops (FFs), and 2016 BRAMs (70.9 Mbit). Since ProNoC needs more BRAMs than the available quantity on the FPGA, Vivado uses UltraRAMs to implement the queues/VCs. Hence, we convert the portion of utilized UltraRAMs into BRAMs to draw a direct comparison with HiPerNoC. HiPerNoC using XY routing consumes 53 % fewer LUTs and BRAMs, and only 3 % more FFs than ProNoC. HiPerNoC uses significantly fewer LUTs due to the simpler router logic. ProNoC uses more BRAMs because it contains four queues at every router input port regardless of its connectivity. The HiPerNoC routers using XY routing constitute 34 % of the consumed LUTs, 44 % of the consumed FFs, and 90 % of the consumed BRAMs. According to the Vivado power analysis, the estimated power consumption of HiPerNoC using XY routing is 29.485 W—16 % lower than ProNoC (35.357 W). When applying O1TURN and minimal adaptive routing, the estimated power consumption of HiPerNoC is 32.877 W and 32.3 W, respectively. HiPerNoC requires less static and dynamic power to operate logic and RAM than ProNoC due to the lower resource usage.

V. Conclusion

In this paper, we proposed HiPerNoC—an FPGA-based SmartNIC architecture deploying a 2D-mesh NoC with a novel router design to handle network traffic with varied processing demands. HiPerNoC employs distributed switch allocation and does not use VCs. It avoids HOL blocking by deploying a separate queue for each switch connection used by the routing algorithm and prevents deadlocks by applying non-blocking VCT switching. We implemented a prototype of HiPerNoC as a 4x4 2D-mesh NoC in SystemVerilog and evaluated it via RTL simulations. The evaluation results demonstrate that HiPerNoC outperforms ProNoC [11] for different traffic patterns, requires fewer FPGA resources, and consumes less power.

REFERENCES

[1] T. Hoefler, S. Di Girolamo, K. Taranov, R. E. Grant, and R. Brightwell, "sPIN: High-performance streaming processing in the network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126970

[2] S. Di Girolamo, A. Kurth, A. Calotoiu, T. Benz, T. Schneider, J. Beránek, L. Benini, and T. Hoefler, "A RISC-V in-network accelerator for flexible high-performance low-power packet processing," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 958–971. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00079

[3] NVIDIA, "NVIDIA BlueField-3 Networking Platform," https://resources.nvidia.com/en-us-accelerated-networking-resource-library/datasheet-nvidia-bluefield?lx=LbHvpR&topic=networking-cloud, accessed: 2024-11-20.

[4] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy, "High performance packet processing with FlexNIC," *SIGARCH Comput. Archit. News*, vol. 44, no. 2, p. 67–81, Mar. 2016. [Online]. Available: https://doi.org/10.1145/2980024.2872367

[5] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, F. Huici, and G. Siracusano, "FlowBlaze: Stateful packet processing in hardware," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 531–548. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/pontarelli

[6] M. S. Brunella, G. Belocchi, M. Bonola, S. Pontarelli, G. Siracusano, G. Bianchi, A. Cammarano, A. Palumbo, L. Petrucci, and R. Bifulco, "hXDP: Efficient software packet processing on FPGA NICs," *Commun. ACM*, vol. 65, no. 8, p. 92–100, jul 2022. [Online]. Available: https://doi.org/10.1145/3543668

[7] M. T. Arashloo, A. Lavrov, M. Ghobadi, J. Rexford, D. Walker, and D. Wentzlaff, "Enabling programmable transport protocols in High-Speed NICs," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 93–109. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/arashloo

[8] D. Sidler, Z. István, and G. Alonso, "Low-latency TCP/IP stack for data center applications," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–4. [Online]. Available: https://doi.org/10.1109/FPL.2016.7577319

[9] D. Sidler, Z. Wang, M. Chiosa, A. Kulkarni, and G. Alonso, "StRoM: smart remote memory," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3342195.3387519

[10] AMD Xilinx, "Alveo U55C High Performance Compute Card," https://www.amd.com/en/products/accelerators/alveo/u55c/a-u55c-p00g-pq-g.html.html, accessed: 2024-11-20.

[11] A. Monemi, J. W. Tang, M. Palesi, and M. N. Marsono, "ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors and Microsystems*, vol. 54, pp. 60–74, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933117302417

[12] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "PANIC: A High-Performance programmable NIC for multi-tenant networks," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 243–259. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/lin

[13] D. U. Becker, *Efficient microarchitecture for network-on-chip routers*. Stanford University, 2012. [Online]. Available: http://cva.stanford.edu/publications/2012/dub-thesis.pdf

[14] K. Zyla, M. Liess, T. Wild, and A. Herkersdorf, "FlexCross: High-speed and flexible packet processing via a crosspoint-queued crossbar," in *2024 27th Euromicro Conference on Digital System Design (DSD)*, 2024, pp. 98–105. [Online]. Available: https://doi.org/10.1109/DSD64264.2024.00022

[15] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *ACM International Conference on Supercomputing 25th Anniversary Volume*. New York, NY, USA: Association for Computing Machinery, 2006, p. 390–401. [Online]. Available: https://doi.org/10.1145/2591635.2667187

[16] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 172–182. [Online]. Available: https://doi.org/10.1109/MICRO.2007.29

[17] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Express cube topologies for on-chip interconnects," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, 2009, pp. 163–174. [Online]. Available: https://doi.org/10.1109/HPCA.2009.4798251

[18] D. Seo, A. Ali, W.-T. Lim, and N. Rafique, "Near-optimal worst-case throughput routing for two-dimensional mesh networks," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005, pp. 432–443. [Online]. Available: https://doi.org/10.1109/ISCA.2005.37

[19] Y. Huan and A. DeHon, "FPGA optimized packet-switched NoC using split and merge primitives," in *2012 International Conference on Field-Programmable Technology*, 2012, pp. 47–52. [Online]. Available: https://doi.org/10.1109/FPT.2012.6412110

[20] Q. Chen and Q. Liu, "Pipelined NoC router architecture design with buffer configuration exploration on FPGA," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–4. [Online]. Available: https://doi.org/10.1109/FPL.2015.7293981

[21] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *SIGARCH Comput. Archit. News*, vol. 20, no. 2, p. 278–287, Apr. 1992. [Online]. Available: https://doi.org/10.1145/146628.140384

[22] M. Galles, "Spider: A high-speed network interconnect," *IEEE Micro*, vol. 17, no. 1, pp. 34–39, 1997. [Online]. Available: https://doi.org/10.1109/40.566196

[23] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, 1999. [Online]. Available: https://doi.org/10.1109/90.769767

[24] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-examining conventional wisdom for designing NoCs in the context of FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 37–46. [Online]. Available: https://doi.org/10.1145/2145694.2145703

[25] Arm Developer, "AMBA 4 AXI4-Stream Protocol Specification," https://developer.arm.com/documentation/ihi0051/a, accessed: 2024-11-20.

[26] "Ultimate CRC," https://opencores.org/projects/ultimate_crc, accessed: 2024-11-20.

[27] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, "Packet transactions: High-level programming for line-rate switches," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 15–28. [Online]. Available: https://doi.org/10.1145/2934872.2934900

[28] J. Gregoire, X. Qian, E. Frazzoli, A. de La Fortelle, and T. Wongpiromsarn, "Capacity-aware backpressure traffic signal control," *IEEE Transactions on Control of Network Systems*, vol. 2, no. 2, pp. 164–173, 2015. [Online]. Available: https://doi.org/10.1109/TCNS.2014.2378871

[29] A. A. Zaidi, B. Kulcsár, and H. Wymeersch, "Back-pressure traffic signal control with fixed and adaptive routing for urban vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 8, pp. 2134–2143, 2016. [Online]. Available: https://doi.org/10.1109/TITS.2016.2521424

[30] A. Forencich, "Verilog AXI Stream Components," https://github.com/alexforencich/verilog-axis, accessed: 2024-11-20.

[31] AMD Xilinx, "UltraScale Architecture Libraries Guide (UG974)," https://docs.amd.com/r/2021.1-English/ug974-vivado-ultrascale-libraries/RAMB36E2, accessed: 2024-11-20.

[32] ——, "Virtex UltraScale+," https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale-plus.html, accessed: 2024-11-20.

[33] R. Sinha, C. Papadopoulos, and J. Heidemann, "Internet packet size distributions: Some observations," *USC/Information Sciences Institute, Tech. Rep. ISI-TR-2007-643*, pp. 1536–1276, 2007. [Online]. Available: https://www.cs.colostate.edu/~christos/papers/Sinha07a.pdf

[34] A. V. de Mello, L. C. Ost, F. G. Moraes, and N. L. V. Calazans, "Evaluation of routing algorithms on mesh based NoCs," *PUCRS, Av. Ipiranga*, vol. 22, 2004. [Online]. Available: https://www.pucrs.br/facin-prov/wp-content/uploads/sites/19/2016/03/tr040.pdf

[35] "AMD OpenNIC Shell," https://github.com/Xilinx/open-nic-shell, accessed: 2024-11-20.