Efficient Queue-Balancing Switch for FPGAs

Philippos Papaphilippou[*§], Kentaro Sano[§], Boma A. Adhi[§], and Wayne Luk[*]

[*]Dept. of Computing, Imperial College London, UK {*pp616, w.luk*}@*imperial.ac.uk*

[§]RIKEN Center for Computational Science, Kobe, Japan {*kentaro.sano, boma.adhi*}@*riken.jp*

***Abstract*—This paper presents a novel FPGA-based switch design that achieves high algorithmic performance and an efficient FPGA implementation. Crossbar switches based on virtual output queues (VOQs) and variations have been rather popular for implementing switches on FPGAs, with applications to network-on-chip (NoC) routers and network switches. The efficiency of VOQs is well-documented on ASICs, though we show that their disadvantages can outweigh their advantages on FPGAs. Our proposed design uses an output-queued switch internally for simplifying scheduling, and a queue balancing technique to avoid queue fragmentation and reduce the need for memory-sharing VOQs. Our implementation approaches the scheduling performance of the state-of-the-art, while requiring considerably fewer FPGA resources.**

***Index Terms*—FPGA, switch, virtual output queues, output-queued, crossbar, scheduling algorithms, queue balancing**

I. Introduction

Full-interconnection could be considered as one of the fundamental and most challenging problems in computer science, appearing from low-level circuits to higher-level applications including neural networks. The challenges come from the fact that the possible paths in a fully-interconnected system are $O(P^2)$, where P is the number of inter-connected entities. The scale of this number directly impacts both software and systems engineering, and different techniques exist for attempting to make the best use of silicon and cycles.

Switch designs are an integral part in FPGA designs. They are used inside the programmable logic for accessing different memories and peripherals from multiple workers, as well as for interfacing with input/output devices, such as for implementing network switches or stacks [1], or communicating with other FPGAs. In order to achieve scalability for a higher number of entities, hierarchical approaches try to eliminate hardware complexity from centralised complex switch architectures. Even with hierarchical switch designs, efficient switch architectures of a lower radix (number of ports) are still important as building blocks [2]. Such hierarchical approaches also include network-on-chip (NoC) designs [3], which still rely on switches of a lower radix (NoC routers), with some performance trade-offs [4].

In hardware, such as for network switch implementations, the main principle behind the research in high-performance switching is to try to temporarily rearrange the packets in time, to optimise the efficiency of more primitive interconnects such as crossbars. Generally, the main challenge with the latest FPGA-based switches is that the better the scheduling performance they offer, the more hardware complexity is required, becoming less versatile and scalable.

This paper presents an FPGA-based switch architecture that exhibits high scheduling performance, while having similar FPGA design attributes as simpler designs, at a resource utilisation considerably less that the state-of-the-art. Our target specification is low to medium radix (number-of-port) switches with high scheduling performance, and an efficient full-throughput (output-per-cycle) FPGA implementation. The main contribution is the novel queue balancing technique that achieves an algorithmic performance close to the optimal for the studied traffic pattern, but with less resources than the state-of-the-art on FPGAs [5].

II. Background and Related Work

A. Crossbars and Head-of-Line (HOL) Blocking

The crossbar is a simple and popular interconnect for both FPGAs and ASICs. It consists of wires having crosspoints everywhere there is an input-output port combination, resulting in a crosspoint complexity of $P_I \times P_O$, where P_I is the number of input ports and P_O the number of output ports. On the crosspoints there are smaller "switches" which are logically equivalent to 2-to-1 multiplexers. On FPGAs, crossbars can logically be implemented with a $P_I - to - 1$ multiplexer per output port [6].

Its main functionality is to apply permutations of its input, hence the potential of collisions (head-of-line blocking), that occurs when two entities send a packet to the same entity simultaneously. Thus, more complex switch architectures are required to overcome this limitation, such as by temporarily rearranging the forwarding of the incoming packets.

B. Input-Queued Crossbar

The input-queued switch uses virtual output queues (VOQs) before the crossbar. There are $P_I \times P_O$ queues, where P_I is the number of inputs and P_O the number of outputs, to allow temporarily holding any incoming packets without collisions. Each VOQ corresponds to every input-output combination.

Figure 1(a) shows how VOQs are used in high-level for a switch of 4 input ports. A scheduling algorithm is responsible for the dequeuing decisions. One advantage of the input-queued crossbar is that each group of VOQs can be represented by a single memory, because there is up to one write and up to one read from each group per cycle.

The input-queued switch is one of the most popular switch architectures on FPGA designs. There are works that focus on the implementation efficiency of its scheduling algorithm [7], [8], as well as its memory-sharing potential [2]. The research on scheduling algorithms for VOQs has been rather deep

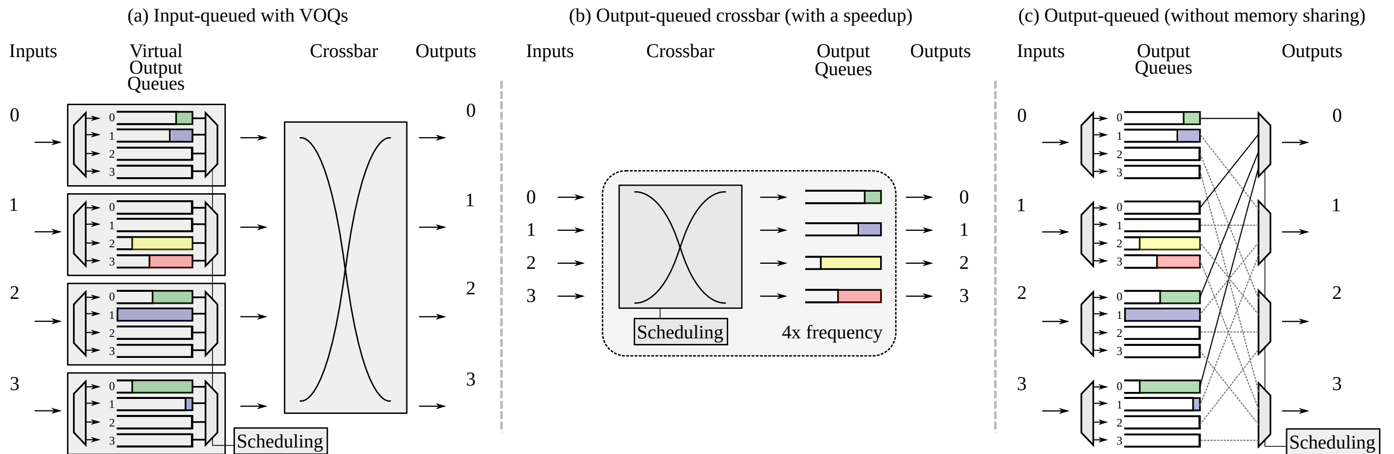

Fig. 1. Switches using different traditional queuing techniques

outside the FPGA domain, due to the importance of scheduling performance and complexity, as well as the performance requirements under different traffic types.

One fundamental limitation of input-queued switches is that only one packet per VOQ group can be extracted per cycle. Another disadvantage is the queue fragmentation, as the distribution of free space depends on the nature of the traffic.

Finally the most important drawback for FPGA implementation could be considered the scheduling algorithm, which can be computationally expensive. The challenge is that for performing well there still need to be a multiple-cycle scheduling step to adequately approximate maximal matching [9]. On hardened network switches this is not a concern, such as with an ASIC implementation that reconfigures the crossbar once every 9 cycles [10]. This is because of the much higher operating frequency and the wider packets, which can be sent progressively in smaller flits. On an equivalent FPGA implementation, although a high operating frequency is achieved from splitting scheduling across multiple cycles, the throughput is reduced considerably [5].

C. Output-Queued Crossbar

The output-queued crossbar is considered the best performing switch algorithmically [11]. As illustrated in the example of figure 1(b), there is a crossbar and only one queue per input port, however, these operate at P_I times the base operating frequency. This speedup (or equivalent workarounds) is necessary to be able to serve all requests one-by-one, and essentially remove HOL blocking from the crossbar. This is widely accepted as expensive and non-scalable [5], [12].

There are optimisations that use a lower speedup [11]. Still, requiring a logic speedup is less desirable on FPGAs [5], due to the restricted operating frequency when compared to ASICs overall, as well as the more homogeneous timing behaviour across the different FPGA resources [13], [14]. Hipernetch [5] is a relatively recent FPGA-based solution that uses a fully-pipelined structure that emulates an output-queued crossbar, with its high resource utilisation as its main drawback.

D. Output-Queued Switch (Without Memory Sharing)

A simpler variation of the latter design is the output-queued switch, which achieves the same performance as the output-queued crossbar in terms of average packet latency, without any speedup. There are output queues in the same organisation as virtual output queues, i.e. each port de-multiplexes the packet to P_O queues according to its destination. The difference to VOQs is that the output arbiters multiplex between queues destined directly for the output port, rather than between queues of the same input port. This is visualised in figure 1(c).

One consideration has to do with the memory organisation of the switch, which becomes a concern for scalability, as some configurations require $P_I \times P_O$ queues. In literature, an output-queued switch usually refers to also having a memory sharing technique, sometimes similar to VOQs. When compared to the input-queued switch, one notable advantage of this approach is that it simplifies the scheduling complexity, as the arbiter decisions are independent of one another. This is because no crossbar is involved for limiting to passable permutations once the packets are stored in the queues.

On FPGAs, this is not a common network switch architecture currently [2]. However, an equivalent generalised approach without memory sharing is used in router designs on FPGA-based NoCs [15], also known as the split-merge switch [16].

III. Proposed Solution

Our solution uses an output-queued switch to simplify scheduling decisions and improve scheduling performance, in combination with a queue balancing mechanism for overcoming memory fragmentation with little hardware overhead.

One shortcoming of the output-queued switch is the reduced efficiency of the queues, which can cause fragmentation for certain traffic. For uniform Bernoulli arrivals this is not a problem, because the probability of each queue receiving a packet is equal among all queues, resulting in uniform queue occupancy. However, under uneven traffic such as with bursts, it is helpful to have a mechanism to balance the queues.

The idea is to add a rotator near the input ports, to create a round-robin effect for queue-balancing, such as from bursts. Note that there is no balancing guarantee, as it rotates all inputs based on a cycle counter, and alternative designs for randomising the input packets would also be appropriate. Figure 2 presents this approach in high-level.

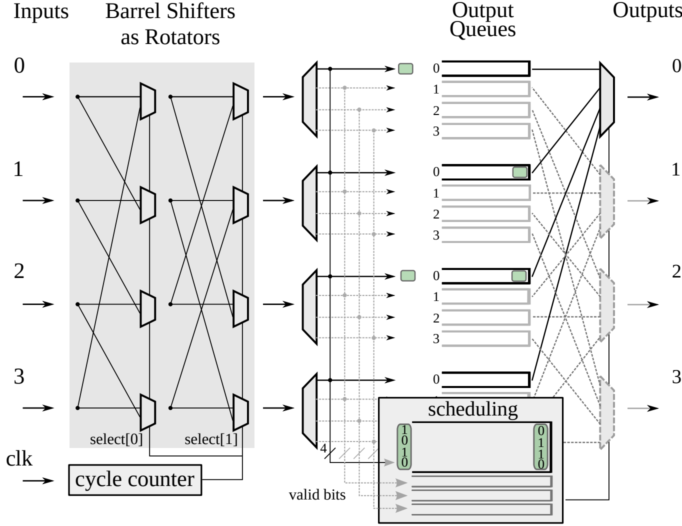

Fig. 2. Input rotation for load balancing, and valid bit queues for reproducing the correct packet order

If plain round-robin arbiters are controlling the queue multiplexers, one issue would be that the order between packets arriving from a source to a destination is partially lost. This is because the rotation effect can land a packet in multiple queues (as the number of inputs (P_I)). Thus, a workaround is required, that allows the arbiters to prioritise based on the arrival time.

As shown in figure 2, the required order information can be obtained by an additional set of P_O queues, each holding packets of P_I bits. Each packet is holding bits corresponding to the enqueue signals of each output queues, when grouped per output port rather than per input port (as per output-queuing), that have arrived on the same cycle. Each output arbiter holds the head packet of its corresponding queue until it extracts all packets that arrived together on that cycle, represented by this head packet. On each cycle, when no packet goes to a port, then no packet is stored in the valid bit queue of that port. In this way, the arbiters can expect at least one valid bit per valid bit packet, and no idle cycles or valid bit queue space is wasted for low input rates. The queue depth requirement for each valid bit queue is $P_I * depth_{FIFO}$, to be able to handle the worst case of full utilisation with packets arriving on different cycles.

IV. Evaluation

In order to evaluate our solution and the queue balancing approach, we compare it against a selection of alternative monolithic (non-hierarchical) switch architectures. First, the scheduling performance is evaluated using simulations. The simulation experiments (sections IV-A and IV-B), low-level implementation details are abstracted, meaning that there is 0 processing latency, and the packet-size, operating frequency and other details are not used. Then we proceed with FPGA-based implementation (section IV-C) to investigate their resource requirements and timing characteristics.

The list of compared switches goes as follows: (1) input-queued that uses VOQs with dual round-robin matching, which is a typical scheduling algorithm (DRRM [9]), (2) “output-queued”, (3) “output-queued with rotator” for load balancing and (4) Hipernetch [5], which is a highly-optimised FPGA-based network switch implementation, and is functionally equivalent to the ”output-queued crossbar” of section II-C. Note that two variations of (1) are studied, First, (1a) the input-queued ”DRRM (1-iter)” switch only performs a single iteration of DRRM, while (1b) “DRRM” uses a $log_2(P_I)$-iteration version of the scheduling algorithm for better scheduling performance [17].

For all experiments, we consider a memory organisation of $P_I \times P_O$ independent queues for all switches, including the optimal ”output-queued crossbar” (represented by Hipernetch) for consistency and simplicity. Any memory sharing techniques are purposely omitted to also approach the best case in terms of any related implementation overhead on performance. For FPGA implementation, (1b) is not used, since such iterative algorithms are more challenging to implement within a single FPGA cycle for yielding full-throughput. In this case, (1a) can still give insights on (1b) with a hypothetical optimal scheduler implementation, as the FPGA-based evaluation is not used here for assessing algorithmic performance.

A. Scheduling Performance (Simulation): Latency

First, we would like to measure the algorithmic performance of the approaches, independently of the implementation details and packet size. We adopt the open-source simulation framework of Hipernetch [5]. Two additional switch models are developed, the output-queued and the output-queued with the rotator. Each data point represents an average of multiple runs, each with 25000 cycles of simulation time. The number of ports is indicatively set to 16, and each input port produces a burst of 32 packets on average, with the same average rate across the ports (uniform bursty traffic).

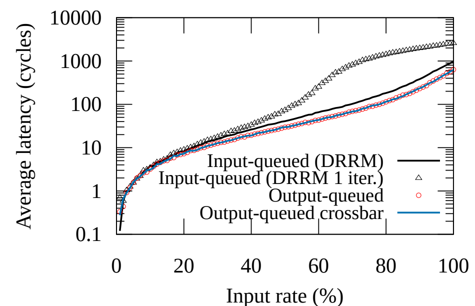

Fig. 3. Output-queued switches yield the lowest packet latencies

The first experiment studies the average packet latency, i.e. the average time a packet stays in the queues in cycles. For

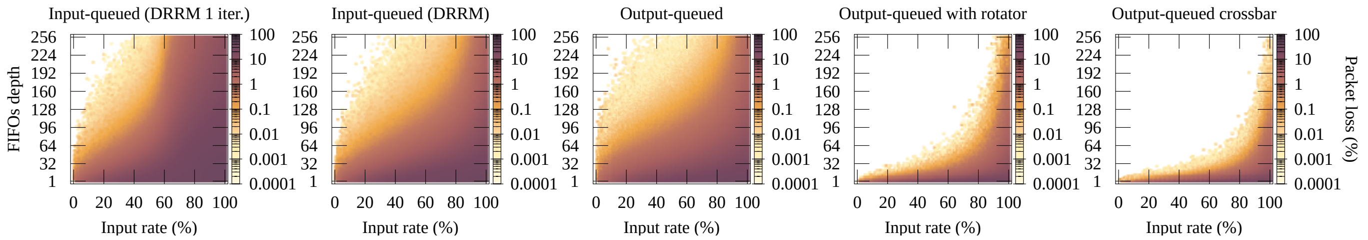

Fig. 4. Packet loss while varying the input rate and FIFO depth under uniform bursty traffic

this experiment, the queues are conventionally considered of infinite size, to focus on the weakness of the switches as scheduling approaches.

As observed from figure 3, the output-queued switch and the output-queued crossbar perform almost identically, and achieve the lowest average latency. The input-queued DRRM switch is a close second, while its single-iteration version is significantly worse. As a numerical example for 100% input rate of bursty traffic, the output-queued switches yield an average latency of 633 cycles, which is 1.2 times lower than the 738 cycles of DRRM, which is in turn 3.3 times lower than the 2404 cycles of DRRM with 1 iteration.

Since this assumes infinite queues, the proposed switch with the rotator is omitted here, as its average-latency performance is identical to the other output-queued switches.

B. Scheduling Performance (Simulation): Packet Loss

There are applications where packet loss is allowed, such as in network switches. For instance, the Transmission Control Protocol (TCP [18]), which is the prominent protocol in today's internet, assumes and mitigates packet losses, such as through acknowledgements and retransmissions. In such cases, minimising the packet loss rate is crucial for service and application performance.

On FPGAs it is important to study the impact of packet loss, as it relates directly to the queue size and organisation. For example, the main argument for input-queued switches (with VOQs) is the ability to share memories, reducing the queue complexity from $P_I \times P_O$ (quadratic for $P_I = P_O$) to P_I [2]. However, according to the theoretical switching performance, we show that it can still be worth using $P_I \times P_O$ queues, but of a fraction of the size.

Figure 4 illustrates the impact of the queue depth on the packet loss rate, for five different switching approaches. From left to right, the switches are sorted according to their ascending overall performance, starting from a DRRM-based switch with 1 iteration. The main observation is the rotator effect on the output-queued switch, which considerably improves the queue utilisation efficiency. The output-queued switch with the rotator yields a similar packet loss profile to the output-queued crossbar (equivalent [5]), which is considered optimal. The packet loss profile of the output-queued switch without rotator is very similar to the input-queued switch.

As a numerical example, an input rate of 80% and a FIFO depth of 32, the output-queued switch improves its packet loss from 11.7% to 1.3% by including a rotator (9x reduction in packet loss). An 11.7% packet loss can also be achieved with an output-queued switch with a rotator with a FIFO depth of only 7 (4.6x reduction in queue space). Under the same traffic, a FIFO depth of 32 yields 26% packet loss for DRRM with 1 iteration (20x more than the rotator with the same FIFOs), and it is closer to our proposed switch with only a single register per FIFO (32x space reduction) at 26.1% packet loss. Lowering the memory requirements can be a key to avoiding BRAM-based implementation using VOQs, which can be a limiting factor for scalability.

C. FPGA Performance (Implementation)

In order to assess the implementation efficiency of the compared switches as FPGA designs, they are implemented from scratch, except with Hipernetch [5], which uses a similar setup. These implementations work as peripherals on a real FPGA for validation purposes, but are out-of-context in terms of any I/O devices, system memories etc. Switches (1), (2) and (3) feature LUTRAM-based FIFOs with depth equal to 8. The open source Hipernetch (4) implementation [5] only has a register per FIFO by default.

The plots (a), (b) and (c) in figure 5 present the implementation results as reported by Vivado for the Xilinx Alveo U280 board. As can be observed, the maximal operating frequency (f_{max}) is similar between switch implementations of the same port configuration, except with the input-queued switch which has a significant overhead in both 8x8 and 16x16 configurations. With respect to the FPGA resources, the look-up table (LUT) and flip-flop register (FF) utilisation varies less between different switch sizes, and the difference is more consistent across different approaches.

The overhead of adding the rotator to the output-queued switch is rather small, as the f_{max} drops from 380 to 343 MHz for 8x8, but increases from 188.5 to 202 MHz for 16x16. Small variations are expected from heuristic-based place-and-route tools, but the increase in frequency can also be explained by the addition of pipeline stages from the rotator. In terms of LUTs and FFs, there is generally a 10% to 19% increase when adding the (pipelined) rotator. Though, it is still considerably more efficient than Hipernetch that uses, for example, 59% more FFs and 44% more LUTs for 16 ports.

Figure 5(d) summarises how the obtained operating frequency translates into the aggregate port bandwidth in billion bits per second (Gbps). Since all studied approaches produce an output per cycle [5], the frequency is multiplied by the studied packet size (256-bit) and number of ports to provide

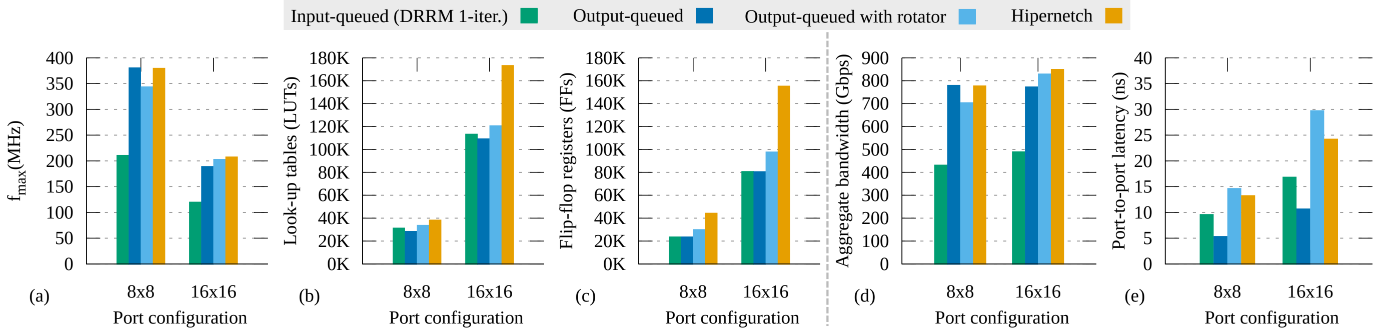

Fig. 5. Vivado-reported (a) f_{max}, (b) LUT and (c) FF utilisation, and obtained (d) bandwidth and (e) latency performance metrics

the aggregate throughput. As an observation, at around 800 Gbps for both 8 and 16 ports, the aggregate bandwidth is similar between the switches excluding the input-queued. The jump to 16 ports approximately halves the operating frequency, hence the smaller variation in the aggregate throughput. This means that hierarchical switches using centralised designs as building blocks are still relevant for scalability [2].

Finally, figure 5(e) presents the port-to-port latencies in nanoseconds by multiplying the clock period by the latency of each switch in FPGA cycles. The latency of (1) and (2) is 1 cycle (plus one more for enqueuing a single packet), where (3) uses a pipelined barrel shifter implementation, resulting in additional $log_2(P_I)$ cycles, though further optimisation and retiming is left as future work. (4) comes with an optimisation parameter (S). We select two well-performing [5] configurations with $S = 3$ and 4 for the 8x8 and 16x16 configurations respectively, that provide a latency of 4 FPGA cycles in both cases. (2) has the lowest latency, which may be also useful in certain applications where the single-cycle latency might be desirable, such as for easier backpressure support, or in systems where exhaustive scheduling is a requirement, including for bursts in some systems interconnects. All switches here provide relatively low port-to-port latencies, including our proposal (3), especially when compared with hierarchical and iterative alternatives which have already been shown to be an order of magnitude higher than (4) Hipernetch [5].

V. Conclusions

A novel switch architecture is proposed that approaches the algorithmic and FPGA-based performance of the state-of-the-art, but with a considerable reduction in resource utilisation. It is also demonstrated that the input-queued switches can be inappropriate for high-throughput FPGA applications, due to their costly scheduling algorithms and the low theoretical switching performance. Additionally, queue fragmentation can reduce the utilisation by tens of times for demanding traffic. Our rotator-based switch solves the fragmentation issue for the output-queued switch, while having a small hardware overhead. Our solution targets monolithic high-performance designs with a low to medium radix in mind, and for FPGA use, but can be applied in existing hierarchical designs such as for implementing routers for network-on-chips (NoCs) and other switches aiming at scalability.

Acknowledgements

This research was sponsored by dunnhumby. The partial support of EPSRC (grant numbers EP/L016796/1, EP/L00058X/1 and EP/N031768/1) and the Japan Society for the Promotion of Science (JSPS) KAKENHI (grant numbers 20H00593 and 21H04869) is gratefully acknowledged.

References

[1] M. Ruiz, D. Sidler, G. Sutter, G. Alonso, and S. López-Buedo, “Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack,” in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, 2019, pp. 286–292.

[2] Z. Dai and J. Zhu, “Saturating the transceiver bandwidth: Switch fabric design on FPGAs,” in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, ACM, 2012, pp. 67–76.

[3] T. Van Chu and K. Kise, “LEF: An Effective Routing Algorithm for Two-Dimensional Meshes,” *IEICE TRANSACTIONS on Information and Systems*, vol. 102, no. 10, pp. 1925–1941, 2019.

[4] N. Kapre and J. Gray, “Hoplite: Building austere overlay NoCs for FPGAs,” in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, 2015, pp. 1–8.

[5] P. Papaphilippou, J. Meng, N. Gebara, and W. Luk, “Hipernetch: High-Performance FPGA Network Switch,” *ACM Transactions on Reconfigurable Technology and Systems (in-press)*, 2021. DOI: 10.1145/3477054.

[6] S. Shreejith, P. Mundhenk, A. Ettner, S. A. Fahmy, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, “VEGa: A high performance vehicular Ethernet gateway on hybrid FPGA,” *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1790–1803, 2017.

[7] J. C. Borromeo, I. Cerutti, P. Castoldi, R. Reyes, and N. Andriolli, “FPGA-based implementation of two-step schedulers for modular optical interconnection networks,” *Journal of Optical Communications and Networking*, vol. 13, no. 5, pp. 116–125, 2021.

[8] I. Cerutti, J. A. Corvera, S. M. Dumlao, R. Reyes, P. Castoldi, and N. Andriolli, “Simulation and FPGA-based implementation of iterative parallel schedulers for optical interconnection networks,” *Journal of Optical Communications and Networking*, vol. 9, no. 4, pp. C76–C87, 2017.

[9] Y. Li, S. Panwar, and H. J. Chao, “On the performance of a dual round-robin switch,” in *INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society*, vol. 3, 2001, pp. 1688–1697.

[10] P. Gupta and N. McKeown, “Designing and implementing a fast crossbar scheduler,” *IEEE micro*, vol. 19, no. 1, pp. 20–28, 1999.

[11] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, “Matching output queueing with a combined input/output-queued switch,” *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1030–1039, 1999.

[12] R. B. Magill, C. E. Rohrs, and R. L. Stevenson, “Output-queued switch emulation by fabrics with limited memory,” *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 606–615, 2003.

[13] Xilinx, *Block memory generator, v8. 4. logicore ip product guide*, 2017.

[14] A. Interconnect, “V2. 1 logicore ip product guide,” *PG059, Xilinx, December*, vol. 20, 2017.

[15] Y. Huan and A. DeHon, “FPGA optimized packet-switched NoC using split and merge primitives,” in *2012 International Conference on Field-Programmable Technology*, IEEE, 2012, pp. 47–52.

[16] N. Kapre, N. Mehta, M. Delorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon, “Packet switched vs. time multiplexed FPGA overlay networks,” in *14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006, pp. 205–216.

[17] F. J. Gonzalez-Castano, C. Lopez-Bravo, R. Asorey-Cacheda, P. S. Rodriguez-Hernandez, and J. Pousada-Carballo, “Analytical evaluation of PHM convergence,” *IEEE transactions on communications*, vol. 54, no. 9, pp. 1547–1553, 2006.

[18] J. Postel, *RFC 793: Transmission control protocol, September 1981.*