ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 3, No. 5, 97-104 (2018)

www.astesj.com

Special Issue on Multidisciplinary Sciences and Engineering

ASTESJ

ISSN: 2415-6698

Virtual Output Queues Architecture for High Throughput Data Center Nodes

Angelos Kyriakos[1,2], Ioannis Patronas[1,2], Georgios Tzimas[1], Vasileios Kitsakis[1], Dionysios Reisis[*1,2]

[1]*National and Kapodistrian University of Athens, Electronics Lab, Physics Dpt, GR-15784, Zografos Greece*

[2]*Institute for Communication and Computers (ICCS), National Technical University of Athens, Greece*

ARTICLE INFO

Article history:
Received: 25 July, 2018
Accepted: 12 September, 2018
Online: 22 September, 2018

Keywords:
Data Centers
Virtual Output Queues
FPGAs

ABSTRACT

The latest design approach for Data Centers (DCs) follows the direction of exploiting optical switching to connect Top-of-Rack (ToR) switches that serve thousands of data storing and computing devices. A ToR's usual function is the Virtual Output Queues (VOQs), which is the prevalent solution for the head-of-line blocking problem of the DC switches. An effective VOQs architecture improves the DC's performance by reducing the frames communication latency and it is efficient with respect to the implementation cost. The current paper introduces a VOQs architecture for the ToRs of DCs that function with Time Division Multiple Access (TDMA). The proposed VOQ architecture contains a bounded number of queues at each input port supporting the active destinations and forwarding the input Ethernet frames to a shared memory. An efficient mechanism of low latency grants each queue to an active destination. The VOQs constitutes a module of a ToR development, which is based on a commercially available Ethernet switch and two FPGA Xilinx boards, the Virtex VC707 and the Xilinx NetFPGA. The VOQs architecture's implementation and validation took place on the NetFPGA board.

1. Introduction

Data centers are comprised of a large number of Servers running Virtual Machines (VMs) and storage resources, which are installed in racks and communicate via the local data center network. The data centers performance depends on the available computing and data storing capacity, the architecture and the features as well as the performance of the underlying network and the Top-of-Rack (ToR) switches connecting the servers to the data center. A key factor in improving the performance of the ToR switches is the solution of the head-of-line blocking issue that is most often settled by embedding Virtual Output Queues architectures [1]. The performance of the networks depends on their interconnection scheme, which usually adhered to the multi-layer approach, and they were based on the Fat Tree or the folded Clos architectural schemes [2, 3, 4]. These approaches nevertheless, are not efficiently scalable and also, in the cases of data centers with a large number of nodes, lead to the use of a considerable number of switches, cables and transceivers, which increase power consumption.

In an effort to overcome these deficiencies researchers and engineers have introduced data center interconnections including an optical circuit switching as well as an electrical packet switching networks [5, 6, 7]. A notable design is the all optical data center proposed by the Nephele project [8]. The Nephele design adopts the Time Division Multiple Access (TDMA) mode of operation in the optical data center network. Consequently, the transmissions are completed within fixed time segments, namely the slots; each slot is assigned for sending a TDMA frame on a specific path that connects a transmitter node to a receiver node. The Nephele data center network is a Software Defined Network (SDN) and all the arrangements regarding its operation are dictated by a central data center controller. The controller is responsible for generating the TDMA Schedule, which defines which nodes communicate during each time-slot [9]. The first version of the scalable, high capacity Nephele network is able to accommodate up to 1600 Top-of-Rack (ToR) switches and each ToR uses 20 links to connect to the data center optical network.

The overall system topology of the data center network is depicted in Figure 1. The network includes I ($I \leq 20$) parallel planes, each consisting of I ($I \leq 20$) unidirectional optical rings.

*Dionysios Reisis, +30 210 727 6708/6720 & dreisis@phys.uoa.gr

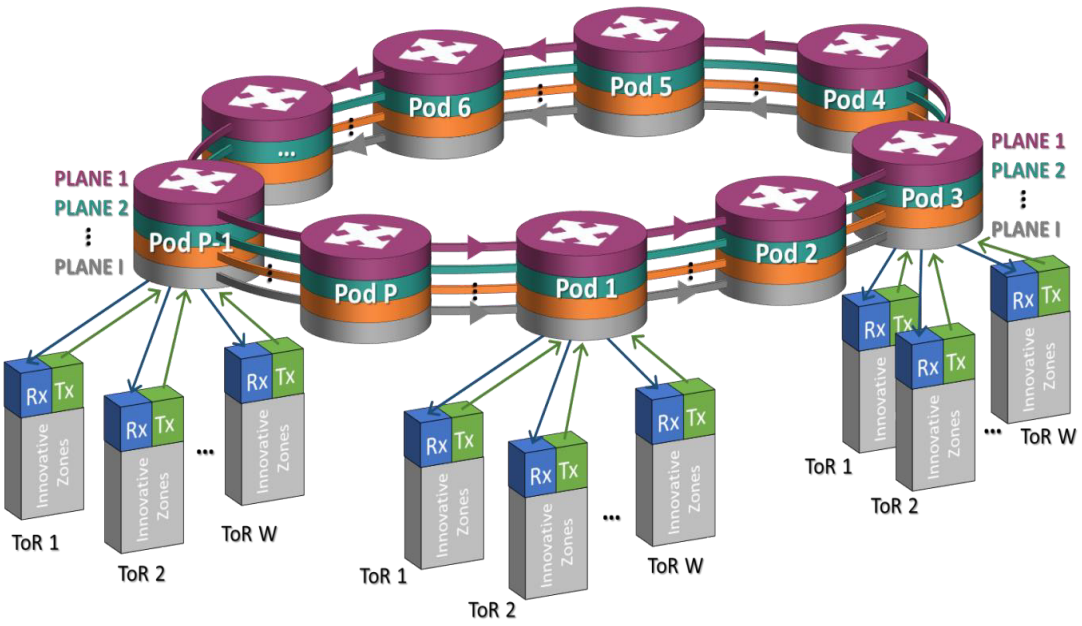


Figure 1: Nephele Data Center Network Architecture

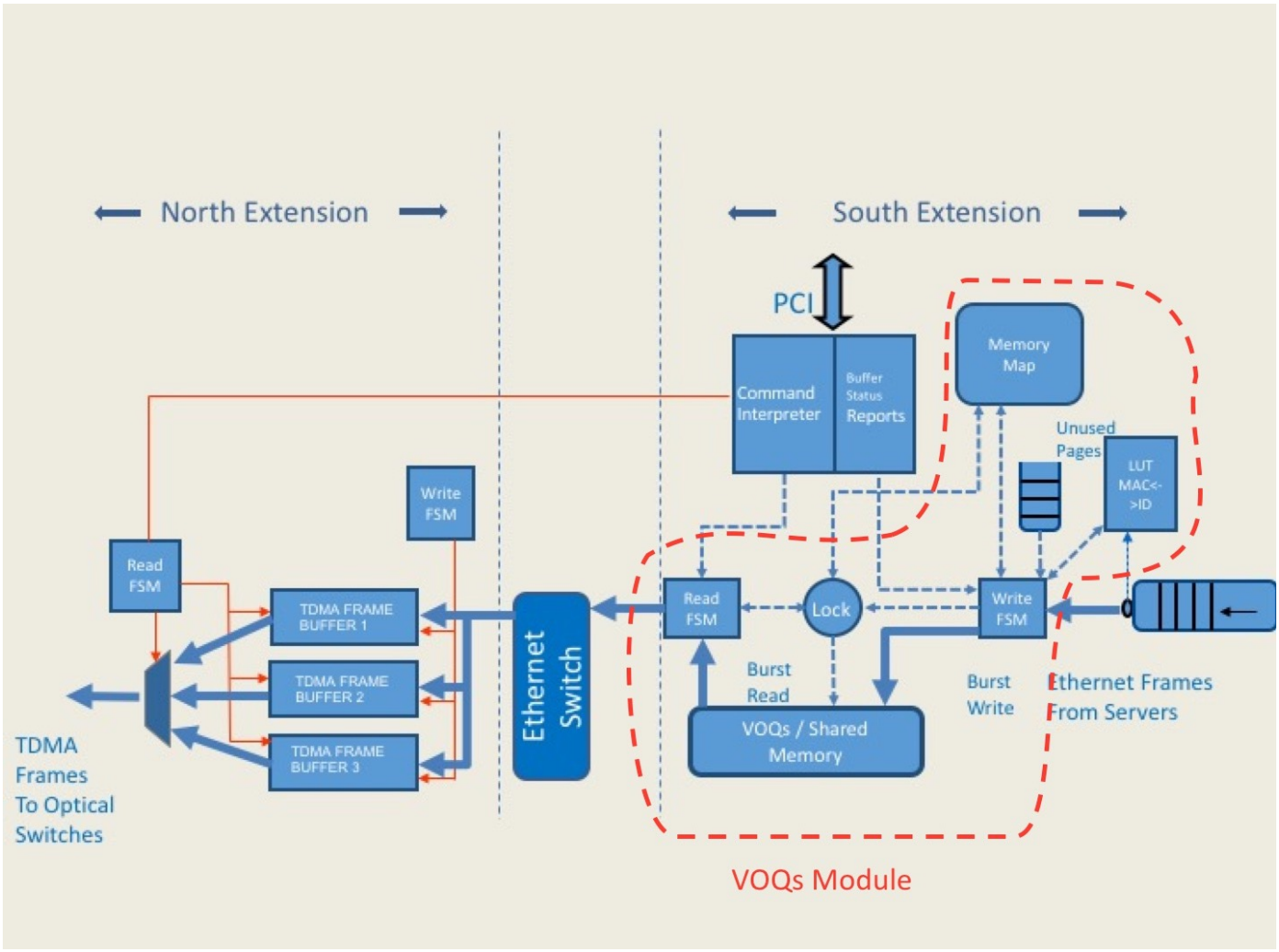


Figure 2: The Nephele Top-of-Rack (ToR) Switch Architecture Overview

The rings interconnect P (P ≤ 20) Points of Delivery (PODs). A POD comprises of I Wavelength Selective Switches (WSS) to connect the I rings, and is connected to W (W ≤ 80) ToRs, through W pod-switches, one for each ToR switch. Each ToR switch has I north ports, such that the *i*th north port is directed to the *i*th POD of each plane (each port is connected to a different POD switch). The south ports of the ToR switch connect the servers, though network interface cards (NICs), with the data center network. The performance of the ToR switch contributes significantly to the operation of the entire data center and it depends on the utilization of its resources as well as on the efficiency of the algorithms and the techniques that it employs. Among the techniques that are critical with respect to the ToR's performance is the handling of the Virtual Output Queues (VOQs). VOQs is an attractive technique for overcoming the head-of-line blocking cases [10].

This paper presents an efficient VOQ organization regarding the resource utilization and the latency needed to assign the incoming Ethernet Frames to the queues of matching destinations. The VOQ architecture introduced in this paper is advantageous due to the following: first, it is efficient with respect to the required implementation area, because it reduces the resources needed to a single shared buffer per output port. This buffer stores all the queues of data that this output port will transmit. Second, the architecture is efficient with respect to the utilization of the shared buffer's bandwidth; this is because it maximizes the throughput utilization of the buffer's interface by utilizing for storing and reading a paging organization, with each page containing a large number of Ethernet frames. Third, the proposed VOQ architecture is scalable, which is an advantage considering the scalability of the entire data center.

The proposed technique achieves the aforementioned goals based on the following ideas. The receiving ethernet Frames with the same destination are collected at the input of the switch into pages of frames. This operation is accomplished by using small sized queues positioned at each input Ethernet port. In the proposed design the number of these small sized queues at the input is bounded by the sum of the connections that are: a) serviced by each input Ethernet port and b) active during a small window of time. The latency is minimized with respect to the time required to associate each input Ethernet Frame to one of the queues. This is accomplished by employing a mechanism that maps each small size queue to one of the active destinations each time an Ethernet Frame arrives at the ToR.

The motivation for designing the proposed VOQs architecture came by the requirements of the ToR included in the Nephele project but it can serve any network, that receives an input of Ethernet frames and particularly those networks which operate under TDMA scheme, are software defined and their nodes may have to overcome the head-of-line blocking. The prototype Nephele ToR switch includes a commercially available Ethernet switch (Mellanox SX1024 [11]) and two Xilinx boards: one Virtex VC707 and one NetFPGA SUME [12]. The implementation and the validation of the VOQs architecture took place on the NetFPGA board.

The paper is organized in five sections. Section II briefly highlights the architecture of the Nephele ToR switch. Section III introduces the architecture and the organization of the VOQs. Section IV presents the details of the FPGA implementation and finally, Section V concludes the paper.

2. The Architecture of the Top-of-Rack (ToR) Switch

The ToR design is a switch and its ports are divided in two sets: a) the south ports, which are 16 10GEthernet ports connecting the ToR with the servers b) the corresponding 16 10Gbps north ports that are connected to the optical data center network. The ToR switch consists of three fundamental blocks. The first is an Ethernet 16×16 switch having all ports as 10GEthernet [11]. The second is the North Extension. It is implemented on an FPGA and its role is: a) the formation of TDMA frames that consist of Ethernet frames and b) to implement the interface of the ToR to the network's optical (POD) switches by using its north ports. The third block is the South Extension. This FPGA based block connects the servers to the ToR. It has increased complexity and its functionality includes: a) the execution of the scheduling commands, b) to be responsible for the communication of the ToR to the data center's control plane, c) to implement the VOQs design and d) to control all the functions of the ToR.

Figure 2 presents the ToR switch's architecture as well as the functional blocks dedicated to the upstream traffic. In the part of the South Extension the figure shows the *LUT MAC-ID* that it assigns a tag to each incoming Ethernet frame. These 11 bits tags will be used within the ToR for addressing the Ethernet frames and saving on the required resources for address bits with respect to the bits required for the MAC addresses of the destinations of the incoming frames. The next action is to forward the Ethernet frames to the *VOQs/Shared Memory* block. This block stores the Ethernet frames in pages. Each page includes a large number of Ethernet frames and its length matches the length of a TDMA frame (also called Nephele frame). All the pages that belong to a destination are arranged in a linked list. The pointers required for keeping the information of each destination's linked are managed by the *Memory Map* block.

The *Command Interpreter* block (Figure 2) is responsible for the translation of the SDN controller commands: it provides to this ToR the destination ToR, which has to receive data in the upcoming TDMA slot. The ToR complies to this command and it retrieves the first page with Ethernet frames that belongs to the linked list associated to the commanded destination and sends this page to the *Ethernet switch*. There is a *Lock* mechanism (Figure 2) that grants either the storing operation of the input Ethernet frames to the shared memory or the reading operation from that memory of the TDMA frames. In more detail, the *Lock* mechanism divides the time into small time windows T_L. Each T_L is dedicated for either writing to the shared buffer or reading from

it. Hence, when the ToR reads from the shared buffer it will continue buffering in the small size queues the incoming traffic from the servers. The length of the T_L is computed at design time to balance: first, the throughput of the shared buffer, which requires long burst transactions for improved performance and second, the need of the ToR operation for writing/reading to/from the shared buffer at close time instances.

The role of the *Ethernet switch* in the upstream direction, is to forward the Ethernet frames to the North Extension and particularly to the buffer of the corresponding destination's north port. In that buffer the Ethernet frames formulate the final TDMA/Nephele frame, to which are also added first, the preamble and second, a word required for each device synchronization. The *Command Interpreter* follows the schedule received from the control plane servers and it specifies (the red control signal of Figure 2) the slot that the ToR will transmit that TDMA frame. For the downstream direction, the Nephele design mandates the Ethernet switch to just forward the frame from the north input port to the corresponding south port. That is, the design complexity of the ToR is mostly related to the upstream path.

The communication of the ToR switch with the control plane is accomplished through the PCI Express interconnection. The PCI Express interface in the proposed architecture is implemented by the use of the Xilinx IP Core for PCIe and RIFFA (Reusable Integration Framework for FPGA Accelerators) [13]. The RIFFA framework consists of an API (Application Programming Interface) and a driver/kernel module for the host PC and IP core for the FPGA, all of which are open-source. The module provided by RIFFA for the FPGA is designed as an extension to the Xilinx core, which handles the physical layer of the PCIe interface.

3. Virtual Output Queues

The proposed VOQ design improves the required hardware resources based on the following concept. During a narrow time window T_B, the ToR switch receives Ethernet frames at its south ports for various destinations in the data center network, which we define as *active destinations*. We consider that for all practical purposes, the number of the active destinations, during T_B, has an upper limit, which can be an outcome of statistical measurements of the network traffic patterns. The active destinations' upper limit is significantly smaller compared to the number of all the possible destinations in the data center. Hence, letting a queue to keep all the incoming Ethernet frames during T_B that have the same active destination and prepare in this queue a burst to be written to the shared buffer, leads to an architecture that includes a set of queues with cardinal number equal to that of the active destinations, while it still keeps the high throughput at the shared buffer.

Considering the above, the VOQs architecture is comprised of: first, the Shared Memory (buffer), second the Memory Map depicted in Figure 2 and third, the VOQs controller. The detailed architecture of the VOQs controller is shown in Figure 3: it is a design of the VOQs controller that includes four (4) active destinations and the corresponding queues, based on a hypothesis that the application asks for four active destination and as shown in Figure 3 there is one queue to support each active destination. In order to define the length of the time window T_B we consider the following facts..

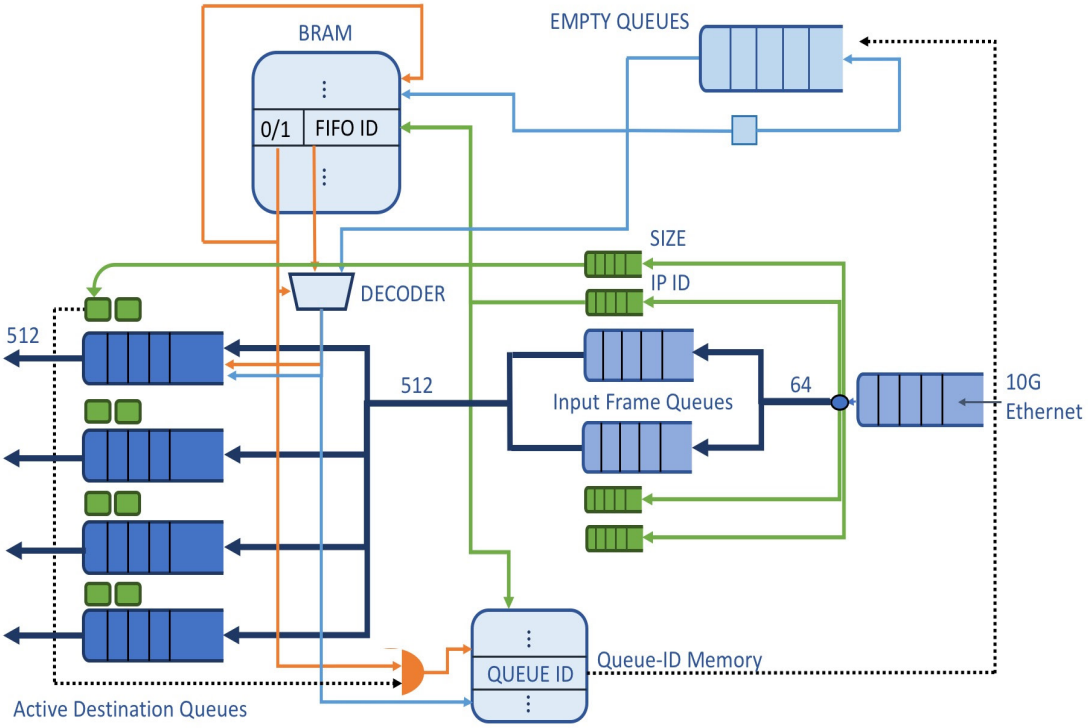


Figure 3: VOQs Controller Architecture Overview

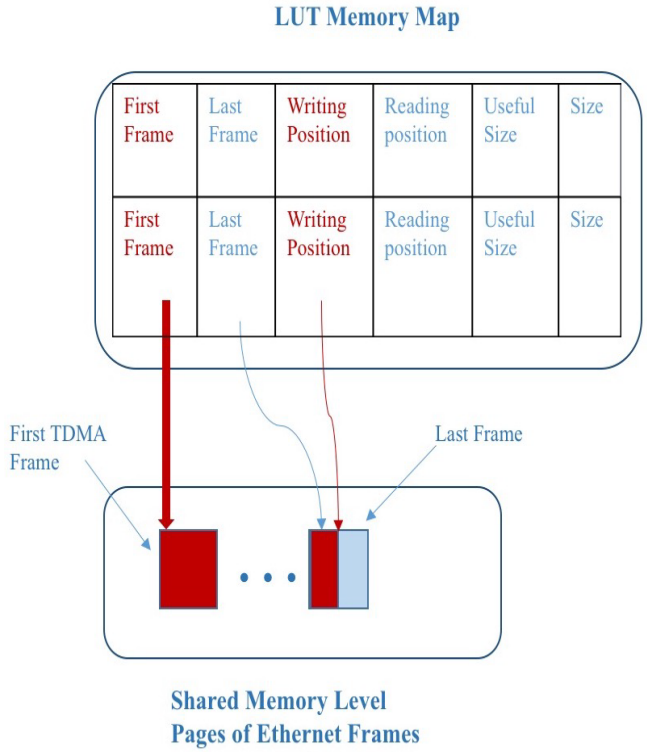
The design of the shared buffer employs a Dynamic Random Access Memory (DRAM) that can reach the considerable throughput of 80 Gbps at its interface; this performance will be feasible if the entire VOQs architecture can operate with burst transactions for reading and writing from/to the shared buffer, thus exploiting the DRAM interface, which requires a minimum burst time t_{mb} depending on the DRAM specifications. The performance of the DRAM organization degrades significantly when the size of the burst size decreases. We note here that, this performance degradation cannot be expressed (defined) as a function of the burst size, e.g. proportional. Therefore, reading and writing from/to a page in the shared buffer (in the linked list assigned to a destination) must be performed in bursts and each burst has to consist of multiple Ethernet frames, in the order of Kbytes. Therefore, we need an architecture of queues able to gather into a single queue all the incoming Ethernet frames that have the same destination; in that queue, the controller will formulate a burst of these Ethernet frames. Finally, it will operate in burst mode to store these frames into the page of the linked list of that destination, which is kept in

We consider the time window T_B and the number of queues for active destinations k to be calculated by the following reasoning. At a clock cycle T_0, given that are available k queues storing Ethernet frames of k different IP, there will be Ethernet frames arriving to at most all of these queues and at the clock cycle T_b that at least one of these queues has completed a burst, and this queue can write the burst to the buffer. Therefore, this queue can formulate another burst either for the IP that it was supporting up to T_b or the queue can be reassigned by the controller to serve another IP. Thus, in this scenario, the worst case is that we have to keep the k queues serving their IPs for as long as no queue has completed a burst: assuming that each queue receives an Ethernet frame in a round robin fashion T_B is at most equal to $k \times t_{mb}$.

According to the above, the efficiency of the VOQs architecture is defined as the maximization of the utilization of the available resources and the DRAM buffer throughput. For this purpose, the design has to: a) include k queues for preparing the bursts, so that each queue prepares a burst that will be stored in an active destination linked list of pages; b) minimize latency and c) minimize the number of the k queues along with their size. The su

cceeding paragraphs describe how we achieve the above goals and they describe in detail the operations of the VOQs Controller as well as its functional blocks and components.

The ToR switch is connected with 10G Ethernet to the servers through its south ports. First, the Ethernet Frames that arrive from the servers at the rate of 10G are buffered in the port queue of the 10G Ethernet module and then are forwarded and buffered to the two *Input Frame Queues* (Figure 3) in the following way: we start counting the incoming frames and depending on the arrival sequence the odd numbered incoming Ethernet Frames are stored in the first *Input Frame Queue* (the upper queue on Figure 3) and the even Ethernet Frames to the second queue. This dual queue architecture gives us the necessary time in order to perform in real-time the two following operations on the Ethernet Frames: while we store a frame in one of the *Input Frame Queues*, we calculate its size and extract its destination's IP, which then are stored to two queues of significantly lesser size, the *IP ID* queue and the *SIZE* queue, which are positioned close to each *Input Frame Queue* in the design of Figure 3.



Each frame's IP stored in the *Input Frame Queues* is passed as input (address) to a LUT, named *BRAM* in Figure 3. The LUT will specify (will give as output) the id of an *Active Destination Queue* (on Figure 3 we shown an example design with four queues): in the specified *Active Destination Queue* we will buffer all the Ethernet Frames with the current active destination IP, in order to form a burst that it will be stored into the linked list of pages of that destination in the DRAM buffer. Apart the id of the *Active Destination Queue* in that *BRAM* location is also stored a flag (0/1). When the flag is equal to "1", it specifies the case in which the *Active Destination Queue* id (stored in the LUT) is granted to the active destination IP. Alternatively, the case when the flag equals to "0" indicates that the frame's destination IP is not yet served by any of the *Active Destination Queues* and hence, the controller has to assign an *Active Destination Queue* to this IP. Now, we consider the case of an Ethernet frame arriving at the ToR and its IP address does not correspond to any of the *Active Destination Queues.* If we have correctly calculated (during the design of the ToR) the minimum required number of the *Active Destination Queues* that it is sufficient to serve the application demands, the VOQs controller will have an empty *Active Destination Queue* available for assignment to a newly arrived

Ethernet frame that requests an *Active Destination Queue* to buffer the following frames with the same IP destination. All the id (numbers) of the unused *Active Destination Queues* are buffered in the queue named *Empty Queues* in Figure 3. At the same clock cycle that we read from the *BRAM* the id of the *Active Destination Queue* that serves the frame's IP along with the "1/0" (assigned to a queue or not) flag, we also read the first empty queue id from the *Empty Queues*. The multiplexer shown at Figure 3 bellow the BRAM is controlled by the flag in order to select: a) the *BRAM* output when the flag equals "1" and b) the *Empty Queues* output if the flag is "0". In the first case where we will use the *BRAM* output, the empty queue id that was just extracted from *Empty Queues* will be returned back in the *Empty Queues,* since it was not used. The above design minimizes the latency for the assignment of an active queue to the new destination.

We have to mention that in order to exploit the high throughput of the DRAM interface, we have to write the Ethernet Frames in the shared buffer as a burst of contiguous words of a significant length (512 bits in the example implementation of the proposed architecture). We note here that, in a writing burst of Ethernet frames the last 512-bit word might not be completely filled with Ethernet frames payload and for completing the burst we add 0xFF as padding. The simple padding provides the advantage of simplifying the control and it reduces the latency at the cost of the dummy data overhead in many pages in the shared buffer. This padding overhead becomes larger for small Ethernet frames and it is reduced significantly in the case of full Ethernet frames. Note here that, when it's time to transmit a TDMA frame the shared memory will provide us with a page: we must be informed regarding the exact number of the useful data in this page in order to remove the padding. For this purpose, we store in the header of each page the useful size along with the actual page size, which is the overall sum of the useful size and the size of the padding stored in the shared buffer.

A small size dual port memory shown in Figure 3. as *Queue-ID Memory*, stores the IP that it is currently served by each *Active Destination Queue*. Each address *X* of the *Queue-ID Memory* corresponds to the *Active Destination Queue* with id *X*. The data at that address *X* of the *Queue-ID Memory* is the destination's IP that is accommodated by this *Active Destination Queue*. When it is the first time that an Ethernet Frame is stored in an empty *Active Destination Queue* the id of this queue is used as the address to the *Queue-ID Memory*, and in that address, we store the frame's IP. During the whole time that this *Active Destination Queue* serves the IP, the *Queue-ID Memory* keeps the IP in that address. Only when an *Active Destination Queue* is left with all its data forwarded to the shared buffer, we will: first, erase the contents of the served destination in the *BRAM* by acquiring the address (IP) from the *Queue-ID Memory* and second, write the queue id to the *Empty Queues* to refresh the *Active Destination Queues* that are vacant and they can be granted to another destination IP. Consequently, the location in the *Queue-ID Memory* will be overwritten by the new IP, which will be served by the corresponding *Active Destination Queue*.

The proposed design minimizes the time required to perform all the previously mentioned operations with respect to clock cycles. The architecture can achieve the time minimization due to the parallelization of the operations and as a result, the VOQ architecture diminishes the latency of each stage. Consequently, the *Active Destination Queues* can be as many as the application dictates as upper bound. Moreover, the length of each queue doesn't need to grow beyond the size of the burst that it is specified by the DRAM controller for reaching its maximum throughput.

The block called *Memory Map* stores all the information related to each linked list in the shared buffer associated to each destination IP. The memory map entries are shown in Figure 4, 5 in two working examples. Each entry of the *Memory Map* block has the following pointers: one at the address of the first page of the list noting from what page we are currently reading data to transmit; one to the last page, required to inform the VOQs that this is the page, which currently stores all the Ethernet frames for the associated destination; one for the "next to write" address of the last page (writing position in Figure 4), one for the "next to read" address of the first page (reading position in Figure 5). Moreover, the *Memory Map* entry provides the exact number of useful data in the page: this information is used to compute the total volume of data of the Ethernet frames with or without out the padding.

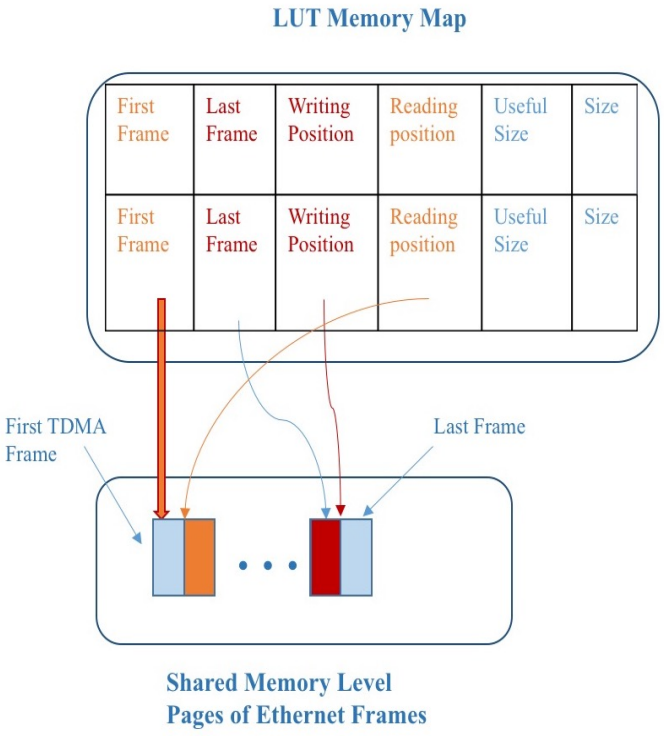


Figure 5: Memory Map Organization Concurrent Write & Read Operations

The pointers at each *Memory Map* block location are refreshed during each burst write/read transaction. Thus, at the beginning of a write/read operation to/from the DRAM buffer we know the exact number of the data (bytes) that will be written/read. We operate the linked list of pages as a queue since we always transmit the head page of the list. The memory map architecture is able to concurrently write and read from the same linked list of pages as shown in Figure 5.

A noteworthy advantage of the novel VOQ technique is the scalability of the architecture, which can be easily configured to accommodate various numbers of Active Destination Queues, the size of the DRAM shared buffer and the size of the *Memory Map* block. The pointers and the size of the linked list of pages for each destination are stored in block rams (BRAM) in the FPGA. The required size of the BRAMs is proportional to: first, the DRAM memory size, and second the number of destinations in the data center network. In the case that the size of the mapping information is relatively large and it constrains the designer of implementing the Memory Map on the FPGA internal BRAM memory, the proposed architecture of the Memory Map block can be implemented on an external Static Random Access Memory (SRAM).

4. FPGA Implementation Details

We have realized an example VOQs design with four (4) active destinations (k = 4 is adequate for most applications in accordance with our T_B and k calculations). The development of the example implementation was made on the NetFPGA SUME board using the Xilinx Vivado development tool. The design includes 3 Intellectual Property (IP) hardware Cores from Xilinx: a) 10GbE Subsystem, which includes the MAC and the 10GbE PCS/PMA b) Integrated Block for PCI Express c) Memory Interface Generator (MIG) for the shared DRAM buffer. The NetFPGA board receives the scheduling commands from the host desktop PC, which is running Linux and communicates with the data center's controller, which runs on a different PC in the same local network.

Table 1: FPGA Resources

Slice LUTs	4194
Slice Registers	2415
Slices	1888
Logic LUTs	2639
Memory LUTs	1285
Flip flop pairs LUTs	4848
Block RAMs	62

The resources occupied in the NetFPGA SUME for the VOQs Controller are presented in the Table 1, reported by the Vivado tool. The input small sized queues are all performing at 156 Mhz clock and use 64 bits word length, in order to comply with the 10G Ethernet physical layer standard. The Active Destination Queues and memories alongside of them in our implementation are performing at 200 MHz with 512-word length.

5. Conclusion

The current paper presented a VOQs architecture, which is efficient with respect to latency and the hardware resources and it supports a ToR switch that is adaptable to any data center network operating under the TDMA scheme. The most noteworthy novelty of the proposed VOQs architecture is the efficient use of a single large shared buffer, the performance of which is fully exploited. The VOQ organization is based on the notion of *Active Destination Queues* that leads to maximize the utilization of the shared buffer and reduces significantly the required number of the *Active Destination Queues* to the number of the connections that are active during a narrow time window. Moreover, the management/control of the *Active Destination Queues* is efficient due to the minimum latency that induces to the operation of the ToR switch. Furthermore, the proposed architecture is scalable with respect to the number (k) of the *Active Destination Queues,* the scale of the data center network (number of destinations), the shared buffer size and the Ethernet protocol (Ethernet type/Frame size).

Conflict of Interest

The authors declare no conflict of interest.

References

[1] A. Kyriakos, I. Patronas, G. Tzimas, V. Kitsakis and D. Reisis, "Realizing virtual output queues in high throughput data center nodes," 2017 Panhellenic Conference on Electronics and Telecommunications (PACET), Xanthi, 2017, pp. 1-4. doi: 10.1109/PACET.2017.8259971

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," SIGCOMM Comput. Commun. Rev., vol. 38, no. 4, pp. 63–74, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402967

[3] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 51–62. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592576

[4] N. Farrington, E. Rubow, and A. Vahdat, "Data center switch architecture in the age of merchant silicon", in Proceedings of the 2009 17th IEEE Symposium on High Performance Interconnects, ser. HOTI '09, 2009, pp. 93–102.

[5] H. H. Bazzaz, M. Tewari, G. Wang, G. Porter, T. S. E. Ng, D. G. Andersen, M. Kaminsky, M. A. Kozuch, and A. Vahdat, "Switching the optical divide: Fundamental challenges for hybrid electrical/optical datacenter networks," in Proceedings of the 2Nd ACM Symposium on Cloud Computing, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 30:1–30:8. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038946

[6] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in Proceedings of the ACM SIGCOMM 2010 Conference, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 339–350. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851223

[7] K. Tokas, C. Spatharakis, I. Kanakis, N. Iliadis, P. Bakopoulos, H. Avramopoulos, I. Patronas, N. Liakopoulos, and D. Reisis, "A scalable optically-switched datacenter network with multicasting," in 2016 European Conference on Networks and Communications (EuCNC), June 2016, pp. 265–270.

[8] P. Bakopoulos, K. Christodoulopoulos, G. Landi, M. Aziz, E. Zahavi, D. Gallico, R. Pitwon, K. Tokas, I. Patronas, M. Capitani, C. Spatharakis, K. Yiannopoulos, K. Wang, K. Kontodimas, I. Lazarou, P. Wieder, D. Reisis, E.

Varvarigos, M. Biancani, H. Avramopoulos, “NEPHELE: an end-to-end scalable and dynamically reconfigurable optical architecture for application-aware SDN cloud datacenters”, IEEE Communications Magazine, 2018

[9] K. Christodoulopoulos, K. Kontodimas, K. Yiannopoulos, E. Varvarigos, "Bandwidth Allocation in the NEPHELE Hybrid Optical Interconnect", 2016 18th International Conference on Transparent Optical Networks (ICTON), July 2016.

[10] Pedro Yébenes, German Maglione-Mathey, Jesus Escudero Sahuquillo, Pedro J. García, Francisco J. Quiles, “Modeling a switch architecture with virtual output queues and virtual channels in HPC-systems simulators”, 2016 International Conference on High Performance Computing & Simulation (HPCS), Innsbruck, Austria, July 2016.

[11] Mellanox Technologies, “SX1024: The Ideal Multi-Purpose Top-of-Rack Switch”, White Paper, May 2013. Available: https://www.mellanox.com/pdf/whitepapers/SX1024-The-Ideal-Multipurpose-TOR-Switch.pdf

[12] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, “Netfpga sume: Toward 100 gbps as research commodity,” IEEE Micro, vol. 34, no. 5, pp. 32–41, Sept 2014.

[13] Matthew Jacobsen, Dustin Richmond, Matthew Hogains, and Ryan Kastner. 2015 RIFFA 2.1: A Reusable Integration Framework for FPGA Accelerators. ACM Trans. Reconfigurable Technol. Syst. 8, 4, Article 22 (September 2015), 23 pages. Available: http://dx.doi.org/10.1145/2815631