HELLENIC REPUBLIC
National and Kapodistrian
University of Athens

Efficient Design Techniques of Switches for Optical Networks and Data Centers

Angelos Kyriakos
Physics Department
School of Science
National and Kapodistrian University of Athens

Supervisors: Dionysios Reisis
Hector Nistazakis
Anna Tzanakaki

A thesis presented for the degree of
Doctor of Philosophy

Athens 2023

Efficient Design Techniques of Switches for Optical Networks and Data Centers

Τεχνικές Αποδοτικής Σχεδίασης Μεταγωγών Οπτικών Δικτύων και Κέντρων Δεδομένων

Angelos Kyriakos

Physics Department

School of Science

National and Kapodistrian University of Athens

The research work was supported in part by the Hellenic Foundation for Research and Innovation (HFRI) Fellowship Number: 29

[1]NKUA

[2]NTUA

Acknowledgements

I would like to express my sincere gratitude to my professor and supervisor Dionysios Reisis for his invaluable advice, patience and support during my PhD study. Additionally, this endeavor would not have been possible without the generous support from the Hellenic Foundation of Research & Innovation, who founded in part my research. I would also like to thank all the members of the Digital Systems Team of the Physics Department for the successful collaboration. Lastly, I'd like to express my appreciation to my parents, without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my studies.

Abstract

The latest design approach for Data Centers follows the direction of exploiting optical switching to interconnect Top-of-Rack (ToR) switches that serve thousands of data storing and computing devices. Optical switching provided the means for the development of Data Centers with high throughput interconnection networks. A significant contribution to the advanced optical Data Centers designs is the Nephele architecture that employs optical data planes, optical Points of Delivery (PoD) switches and ToR switches equipped with 10 Gbps connections to the PoDs and the servers. Nephele follows the Software Defined Network (SDN) paradigm based on the OpenFlow protocol and it employs an Agent communicating the protocol commands to the data plane. 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 Data Center switches. An effective VOQs architecture improves the Data Center's performance by reducing the frames communication latency and it is efficient with respect to the implementation cost. The current thesis introduces a VOQs architecture for the Data Center's ToR switches that function with Time Division Multiple Access (TDMA). The proposed VOQs 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 buffer. 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. Moreover, the current thesis presents a management tool for the control plane's Agent of the Data Center. The Graphical User Interface (GUI) of the Agent's management tool is utilized to configure the Agent, create commands, perform step operations and monitor the results and the status. When used as a testing and validation tool, it plays a significant role in the improvement of the Agent's design as well as in the upgrade of the entire Data Center's organization and performance. Furthermore, aiming to improve the Quality of Service (QoS) for diverse applications of the Data Center, recent works utilize advanced Deep Learning techniques. The plethora of Machine and Deep Learning applications involve complex processes that impose the need for hardware accelerators to achieve real-time performance. Among these, notable are the Machine Learning (ML) tasks using Convolutional Neural Networks (CNNs) for classification applications.Aiming at contributing to the CNN accelerator solutions, the current thesis focuses on the design of FPGA Accelerators for CNNs of limited feature space to improve performance, power consumption and resource utilization, merits that ultimately enable the use of CNNs locally at the Data Center's ToR switches. The proposed CNN design approach targets the designs that can utilize the logic and memory resources of a single FPGA device and benefit numerous applications like the Edge, Mobile, Data Center and On-board satellite (OBC) Computing. This work exploits the proposed approach to develop an Example FPGA Accelerator for Vessel Detection, on a Xilinx Virtex 7 XC7VX485T FPGA device. The resulting architecture achieves an operating frequency of 270 MHz, while consuming 5 watts, it validates the approach.

Keywords:FPGAs - Data Centers - Virtual Output Queues -GUI- CNN - Parallel Processing

Περίληψη

Η σύγχρονη σχεδίαση των Κέντρων Δεδομένων εκμεταλλεύεται τις δυνατότητες που προσφέρει η οπτική μεταγωγή με στόχο την διασύνδεση των μεταγωγών ικριώματος μεταξύ τους, οι οποίοι εξυπηρετούν χιλιάδες συσκευές αποθήκευσης και υπολογιστικά συστήματα. Οι καινοτομίες στον τομέα τον οπτικών επικοινωνιών και της οπτικής μεταγωγής συνέβαλλαν σημαντικά στην ανάπτυξη των Κέντρων Δεδομένων με υψηλής διεκπεραιωτικότητας δίκτυα διασύνδεσης. Σημαντική συνεισφορά στα προηγμένα οπτικά Κέντρα Δεδομένων παρουσιάζει η αρχιτεκτονική Nephele, η οποία χρησιμοποιεί οπτικά επίπεδα δεδομένων, οπτικούς μεταγωγούς στα Σημεία Παράδοσης και μεταγωγούς Ικριώματος με δυνατότητα διασύνδεσης της τάξης των 10 Gpbs μεταξύ των Σημείων Παράδοσης και των εξυπηρετητών. Η αρχιτεκτονική Nephele ακολουθεί την Δικτύωση Βασισμένη σε Λογισμικό, χρησιμοποιεί το πρωτόκολλο OpenFlow και στηρίζεται σε έναν Πράκτορα Λογισμικού, ο οποίος υλοποιεί την μεταφορά των εντολών του πρωτοκόλλου στους μεταγωγούς του επιπέδου δεδομένων. Ένας μεταγωγός Ικριώματος καλείται συνήθως να υποστηρίζει την λειτουργία των Εικονικών Ουρών Εξόδου, οι οποίες αποτελούν την επικρατέστερη λύση στο πρόβλημα του αποκλεισμού μετάδοσης πακέτων που προέρχονται από την ίδια είσοδο σε πολλαπλές εξόδους του μεταγωγού. Μία αποτελεσματική αρχιτεκτονική Εικονικών Ουρών Εξόδου βελτιώνει την επίδοση του Κέντρου Δεδομένων μειώνοντας την λανθάνουσα καθυστέρηση της επικοινωνίας πλαισίων δεδομένων και ειναι αποδοτική όσον αφορά το κόστος υλοποίησης. Η συγκεκριμένη διατριβή εισάγει μία αρχιτεκτονική Εικονικών Ουρών Εξόδου για μεταγωγούς Ικριώματος Κέντρων Δεδομένων τα οποία λειτουργούν σύμφωνα με την μέθοδο πολλαπλής πρόσβασης διαίρεσης χρόνου. Η προτεινόμενη αρχιτεκτονική Εικονικών Ουρών Εξόδου περιλαμβάνει έναν περιορισμένο αριθμό ουρών σε κάθε πόρτα εισόδου που υποστηρίζουν τους ενεργούς προορισμούς και αποθηκεύουν προσωρινά τα πακέτα Ethernet σε δυναμική μνήμη τυχαίας προσπέλασης. Ένας αποδοτικός μηχανισμός χαμηλής λανθάνουσας καθυστέρησης αντιστοιχεί κάθε ουρά σε έναν ενεργό προορισμό. Οι Εικονικές Ουρές Εξόδου αποτελούν ένα δομικό στοιχείο του μεταγωγού Ικριώματος, ο οποίος βασίζεται σε ένα εμπορικά διαθέσιμο μεταγωγό Ethernet και σε δύο κάρτες Xilinx FPGA , την Virtex VC707 και την NetFPGA. Η αρχιτεκτονική των Εικονικών Ουρών Εξόδου υλοποιήθηκε και επαληθεύτηκε μέσω δοκιμών στην κάρτα NetFPGA. Επιπλέον, η συγκεκριμένη διατριβή παρουσιάζει ένα εργαλείο διαχείρισης για τον Πράκτορα Λογισμικού του Κέντρου Δεδομένων. Η Γραφική Διεπαφή Χρήστη του εργαλείου διαχείρισης του Πράκτορα Λογισμικού χρησιμοποιείται για την διαμόρφωση του Πράκτορα Λογισμικού, την δημιουργία εντολών, την εκτέλεση λειτουργιών σε βήματα και την παρακολούθηση των αποτελεσμάτων και της κατάστασης των μεταγωγών. Χρησιμοποιούμενο ως εργαλείο δοκιμών και επαλήθευσης, διαδραματίζει ένα σημαντικό ρόλο στην βελτίωση της σχεδίασης του Πράκτορα Λογισμικού καθώς επίσης και στην αναβάθμιση ολόκληρης της οργάνωσης του Κέντρου Δεδομένων και των επιδόσεων του. Επιπρόσθετα, με στόχο την Διασφάλιση της Ποιότητας Υπηρεσιών για τις ποικίλες εφαρμογές των Κέντρων Δεδομένων πρόσφατες έρευνες αξιοποιούν σύγχρονες τεχνικές Βαθιάς Μάθησης. Η πληθώρα από εφαρμογές Μηχανικής και Βαθιάς Μάθησης περιλαμβάνουν πολύπλοκες διεργασίες που επιβάλλουν την ανάγκη των Επιταχυντών Υλικού για την εκτέλεσή τους σε πραγματικό χρόνο. Μεταξύ αυτόν, αξιοσημείωτα είναι τα Συνελικτικά Νευρωνικά Δίκτυα για εφαρμογές κατηγοριοποίησης. Με στόχο την συνεισφορά στον τομέα των Επιταχυντών Υλικού Συνελικτικών Νευρωνικών Δικτύων, η παρούσα διατριβή επικεντρώνεται σε νευρωνικά δίκτυα περιορισμένου αριθμού χαρακτηριστικών για να βελτιώσει τις επιδόσεις, την κατανάλωση ενέργειας και την αξιοποίηση των πόρων, στοιχεία που τελικά θα δώσουν την δυνατότητα για την χρήση τους τοπικά στους μεταγωγούς ενός Κέντρου Δεδομένων. Η προτεινόμενη σχεδιαστική προσέγγιση Συνελικτικών Νευρωνικών Δικτύων στοχεύει στην αξιοποίηση των πόρων λογικής και μνήμης ενός FPGA, και ωφελεί πολυάριθμες εφαρμογές όπως Αποκεντρωμένες και Φορητές εφαρμογές, Κέντρα Δεδομένων και Δορυφορικές εφαρμογές. Η συγκεκριμένη διατριβή εκμεταλλεύεται την προτεινόμενη σχεδιαστική προσέγγιση, ώστε να αναπτύξει ένα Παράδειγμα Επιταχυντή για Αναγνώριση Πλοίων, στην κάρτα Xilinx Virtex 7 XC7VX485T FPGA.Η παραχθείσα αρχιτεκτονική επιτυγχάνει συχνότητα λειτουργίας 270 MHz , καταναλώνοντας 5 watt επαληθεύοντας την σχεδιαστική προσέγγιση.

Λέξεις-κλειδιά:FPGAs - Κέντρα Δεδομένων - Εικονικές Ουρές Εξόδου - Γραφική Διεπαφή Χρήστη - Συνελικτικά Νευρωνικά Δίκτυα - Παράλληλη Επεξεργασία

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AMBA	Advanced Microcontroller Bus Architecture
API	Application Programming Interface
AXI	Advanced eXtensible Interface
BAM	Bit-accurate Model
BRAM	Block Random Access Memory
CMX	Connection Matrix
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Values
CUDA	Compute Unified Device Architecture
DDR	Double Data Rate
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
FF	Flip-Flop
FIFO	First-In First-Out
FPGA	Field-programmable Gate Array
FSM	Finite-State Machine
GFLOPS	Giga Floating Point Operations Per Second

GOPS	Giga Operations Per Second
GPU	Graphics Processing Unit
GTX	Giga Texel Shader
GUI	Graphical User Interface
HLS	High-level Synthesis
IP	Intellectual Property
IT	Information Technology
JVM	Java Virtual Machine
LSTM	Long Short-Term Memory
LUT	Lookup Table
LUTRAM	Lookup Table Random Access Memory
MAC	Media Access Control
MIG	Memory Interface Generator
ML	Machine Learning
OBC	On-board Computing
ONF	Open Networking Foundation
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PCS	Physical Coding Sublayer
PMA	Physical Medium Attachment

PoD	Point of Delivery
QoS	Quality of Service
RAM	Random Access Memory
R-CNN	Region-based CNN
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RIFFA	Reusable Integration Framework for FPGA Accelerators
RNN	Recursive Neural Network
ROM	Read-Only Memory
SDN	Software Define Networking
SHAVE	Streaming Hybrid Architecture Vector Engine
SRAM	Static Random Access Memory
SSD	Single Shot MultiBox Detector
TDMA	Time Division Multiple Access
ToR	Top of Rack
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VLAN	Virtual Local Area Network
VM	Virtual Machine
VOQ	Virtual Output Queue
WDM	Wavelength Division Multiplexing

YOLO	You Only Look Once

1 | Introduction

The ongoing research and innovation in the area of telecommunication networks has enabled the development of data centers, comprising of very large numbers of interconnected servers. The widespread availability of cloud applications to billions of users and the emergence of software-, platform- and infrastructure-as-a-service models led to the reliance on data centers. As traffic within a data center (east-west) is higher than incoming/outgoing traffic (Cisco, 2014-2019), data center's interconnection networks play a crucial role in its performance. State-of-the-art data center networks are based on electronic switches connected in fat-tree topologies using optical fibers, with electro-opto-electrical transformation at each hop (Al-Fares, Loukissas, & Vahdat, 2008). However, fat-trees tend to under-utilize resources, require a large number of cables and switches, suffer from poor scalability and upgradability, and they result in very high energy consumption (Benson, Akella, & Maltz, 2010), (Roy, Zeng, Bagga, Porter, & Snoeren, 2015). The introduction of optical switching in data centers plays a key role in solving these shortcomings. Many recent works proposed hybrid electrical/optical switched data center networks (Farrington et al., 2010), (G. Wang et al., 2010), (Singla, Singh, Ramachandran, Xu, & Zhang, 2010), (Saridis et al., 2016), (Kachris & Tomkos, 2012), (Bakopoulos et al., 2018). Optical switches are mostly used in telecom networks as circuit switches. They passively redirect light from any port to another (full cross-bar), but have high reconfiguration times (tens of ms for high radix and tens of µs for low radix switches), posing barriers to their applicability in data centers.

The significant increase in the available throughput in optical transmission supports the development of data center networks, optical transceivers achieve throughput in the order of 10 Gbps and state of the art solutions aim at 100 Gbps (Zilberman, Audzevich, Covington, & Moore, 2014). The design of hybrid data center nodes that operate in the electrical domain aim to support the high throughput optical interconnection and they have to sustain the same level of throughput in their electrical and computational systems. Moreover, the design has to be efficient regarding the implementation cost and the power consumption. Furthermore, all aforementioned challenges are proportional to the size of the data center network (Vahdat et al., 2010), (Han et al., 2013), thus the provided solutions must present high scalability, in order to be suitable for the constantly growing in size data centers (Cisco, 2014-2019). The current thesis focuses in the study and development of new techniques for the design and implementation of switches in the electrical domain, utilized in state of the art optical interconnection data center networks.

First, the use of optical switches in data center networks introduces various challenges in the design of the data center switches, which has to compromise between their size and the reconfiguration speed and also be able to scale for large data center networks without increased cost. Top-of-Rack (ToR) switches must accommodate the incoming traffic of Ethernet Frames originating at the servers of the data center to packets suitable to be transmitted through the optical data center network, adhering to the scheduling of the central data center controller (H. Liu et al., 2013), (Moor, 2013), (Patronas, Kyriakos, & Reisis, 2016). Moreover, data center nodes have to internally minimize blocking probability, transmission latency and eliminate the head of line blocking, packets originating from the same input port destined to multiple output ports can be held up in a queue by the first packet. The model of the Virtual Output Queues (VOQs) is the most prevalent solution to the head of line blocking (Yébenes, Maglione-Mathey, Escudero-Sahuquillo, García, & Quiles, 2016) however, the implementation is a challenge in large scale data centers because of the need of large shared memory buffers, usually realized in external Dynamic Random Access Memory (DRAM).

Second, hybrid data center networks typically use centralized control following the Software Defined Network (SDN) paradigm (Christodoulopoulos, Lugones, Katrinis, Ruffini, & O'Mahony, 2015), (Saridis et al., 2016), (Bakopoulos et al., 2018). The working time is divided in time slots according to the method of Time Division Multiple Access (TDMA) (Tokas et al., 2016), (Vattikonda, Porter, Vahdat, & Snoeren, 2012). The control plane is based on a Central Controller/Scheduler necessary for the scheduling of the data center network operation with the following steps. First, it gathers the status information of the traffic from every data center switch second, it computes the allocation of the resources/time-slots for the next scheduling period, and third it configures the switches. The distribution of the schedule has to be synchronized and executed as fast as possible, so that the remaining time between the scheduling periods can be allocated to the computation of the schedule, a very challenging task to be solved in real-time. A software Agent of the data center supervises the execution of the schedule and it acts as a bridge between the Central SDN Controller and the data center switches.

The current thesis also considers the design and implementation of Convolutional Neural Networks for classification applications in FPGAs. The Machine and Deep Learning applications include complicated calculations that require large amounts of memory and call for the use of FPGA accelerators (Abdelouahab, Pelcat, Sérot, & Berry, 2018), in order to achieve real-time performance. The goal of this work is to propose a CNN design approach for the limited feature space classification tasks that benefit various applications in the Data Center (Lim et al., 2019), (P. Wang, Ye, Chen, & Qian, 2018), Edge (Choi & Sobelman, 2022), Mobile (Howard et al., 2017) and OBC computing (Rapuano et al., 2021),(Pitonak, Mucha, Dobis, Javorka, & Marusin, 2022), while utilizing the resources of a single FPGA device with only on-chip memories, omitting the use of external DRAM. This fact enables the possible integration of the CNN accelerator in data center switches that use an external DRAM as a buffer for the incoming packets in VOQs.

The use of High Level Synthesis (HLS) is prevalent in modern CNN FPGA Accelerators mainly due to the short development time and the provided hardware abstraction (Kim, Grady, Lian, Brothers, & Anderson, 2017), (Solovyev, Kalinin, Kustov, Telpukhov, & Ruhlov, 2018), (Zhang et al., 2015), (Sankaradas et al., 2009), (Peemen, Setio, Mesman, & Corporaal, 2013), (B. Liu et al., 2019). Although, the HLS design approach impede the FPGA designers/engineers to develop an efficient FPGA architecture with respect to resource utilization, energy consumption and the achievable performance (Pelcat, Bourrasset, Maggiani, & Berry, 2016). Few authors present solutions for CNN application with FPGA accelerators developed with VHSIC Hardware Description Language (VHDL). (Rapuano et al., 2021) present an On-board satellite FPGA accelerator for CNN inference, implemented with VHDL which utilizes a single processing unit with external DRAM memory. The current thesis focuses on streamline architectures, developed with VHDL, that implement the contiguous CNN layers in a pipeline fashion that differs to the implementation of a systolic array. The advantages of the proposed design is first the avoidance of idle computing time by extensive pipelining and second the reduced memory resources (Y. Zhao et al., 2019), leading to the use of only on-chip (FPGA) memory, features that combined improve resources utilization, reduced latency and power consumption (Sze, Chen, Yang, & Emer, 2017), (Lamoureux & Luk, 2008).

The proposed CNN design approach is divided into three phases. The first phase introduces guidelines for the design stage of the CNN with a tool like TensorFlow (Abadi et al., 2015), adhering to them produces a CNN model that is friendlier to a hardware implementation, because the resulting CNN model has reduced memory requirements but keeps the same levels of classification accuracy. The second phase transforms the CNN model into a fixed-point Bit-accurate Model (BAM) performing the same calculations but in an arithmetic representation that can be efficiently implemented in hardware that does not decrease classification accuracy. The third phase utilizes an already developed library of VHDL Blocks, in order to save developing time, and along with the proposed approach's mapping methodology results in a streamline architecture FPGA accelerator with improved performance, reduced power consumption and resource utilization.

1.1 Thesis Outline

The remainder of this thesis is organised as follows:

Chapter 2 — introduces the study and implementation of the Virtual Output Queues architecture and its integration in the Top of Rack switch of the Nephele data center.

Chapter 3 — demonstrates the management tool for the optical interconnected data center Nephele, how is developed, the Graphical User Interface, its usecases as well as its advantages.

Chapter 4 — presents the CNN design approach for real-time classification FPGA accelerators that can be realised in a single FPGA device and it proves its merits by developing an Example FPGA Accelerator for Vessel Detection.

Chapter 5 — concludes this thesis and presents the major paths for future work.

2 | Virtual Output Queues

2.1 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 center's performance depends on the available computing and data storing capacity, the architecture, 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 (Yébenes et al., 2016), (Kyriakos, Patronas, Tzimas, Kitsakis, & Reisis, 2017).

The motivation for designing the proposed VOQs architecture came by the requirements of the ToR switch 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 following section briefly highlights the Nephele data center architecture then section 2.3 introduces the architecture of the Nephele ToR switch. Section 2.4 presents the organization of the VOQs. Finally, Section 2.5 presents the details of the VOQs Controller FPGA implementation.

2.2 Background on the Nephele Data Center

The performance of the data center 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 (Al-Fares et al., 2008), (Greenberg et al., 2009), (Farrington, Rubow, & Vahdat, 2009). These approaches nevertheless, are not efficiently scalable and also, in the case 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 (Bazzaz et al., 2011), (Farrington et al., 2010), (Tokas et al., 2016). A notable design is the all optical data center proposed by the Nephele project (Bakopoulos et al., 2018).

The Nephele data center involves a slotted hybrid electrical/optical interconnection network that is advantageous with respect to the dynamic allocation of resources. The network includes PoDs of racks that communicate with the so-called innovation zones, which are the devices dedicated for the disaggregated computing, storage and memory resources. The innovation zones are connected to ToR switches (Patronas et al., 2016). Each innovation zone can communicate to other innovation zones through an all optical or an electro-optical channel.

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 (Christodoulopoulos, Kontodimas, Yiannopoulos, & Varvarigos, 2016). 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.

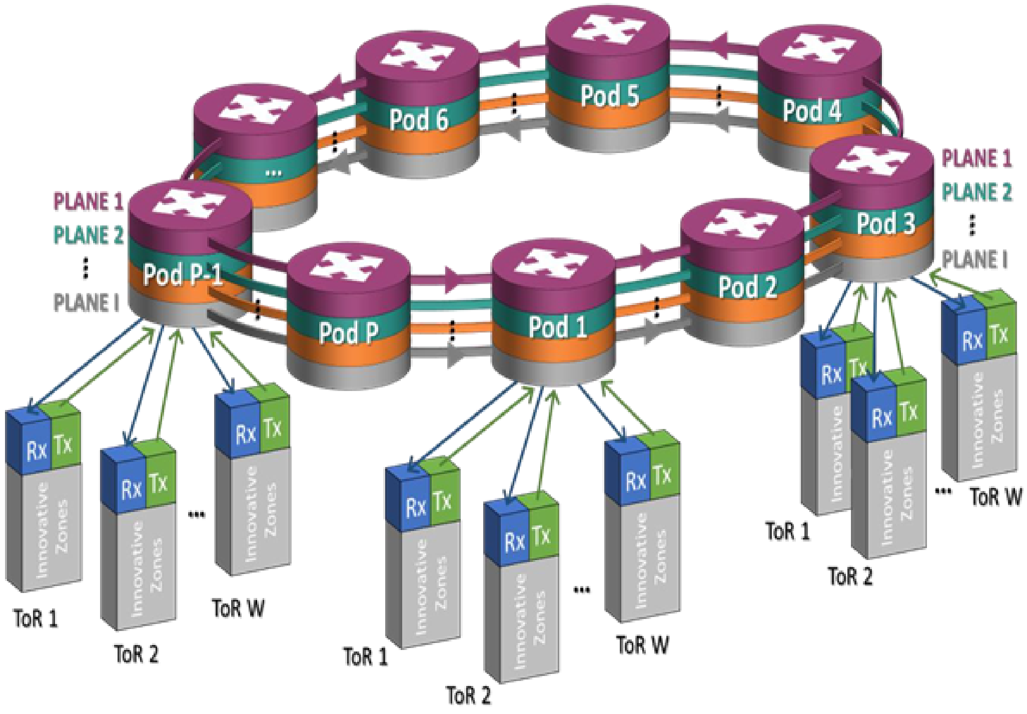


Figure 2.1: The Nephele Data Center Network Architecture

The overall system topology of the data center network is depicted in Figure 2.1. The network includes I ($I \leq 20$) parallel planes, each consisting of I ($I \leq 20$) unidirectional optical rings. The rings interconnect P ($P \leq 20$) Points of Delivery (PoDs). A PoD comprises of I Wavelength Selective Switches (WSS) to connect the I rings, and is connected to W ($W \leq 80$) ToRs, through W pod-switches, one for each ToR switch. Each ToR switch has I north ports, such that the ith north port is directed to the ith PoD of each plane (each port is connected to a different PoD switch). The south ports of the ToR switch connect the servers, through 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 (Yébenes et al., 2016).

2.3 The Architecture of the Top-of-Rack 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 (Mellanox-Technologies, May 2013). 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. The prototype Nephele ToR switch includes a commercially available Ethernet switch (Mellanox SX1024 (Mellanox-Technologies, May 2013)) and two Xilinx boards: one Virtex VC707 and one NetFPGA SUME (Zilberman et al., 2014). The implementation and validation of the VOQs architecture took place on the NetFPGA board. This section briefly highlights the architecture of the Nephele ToR switch

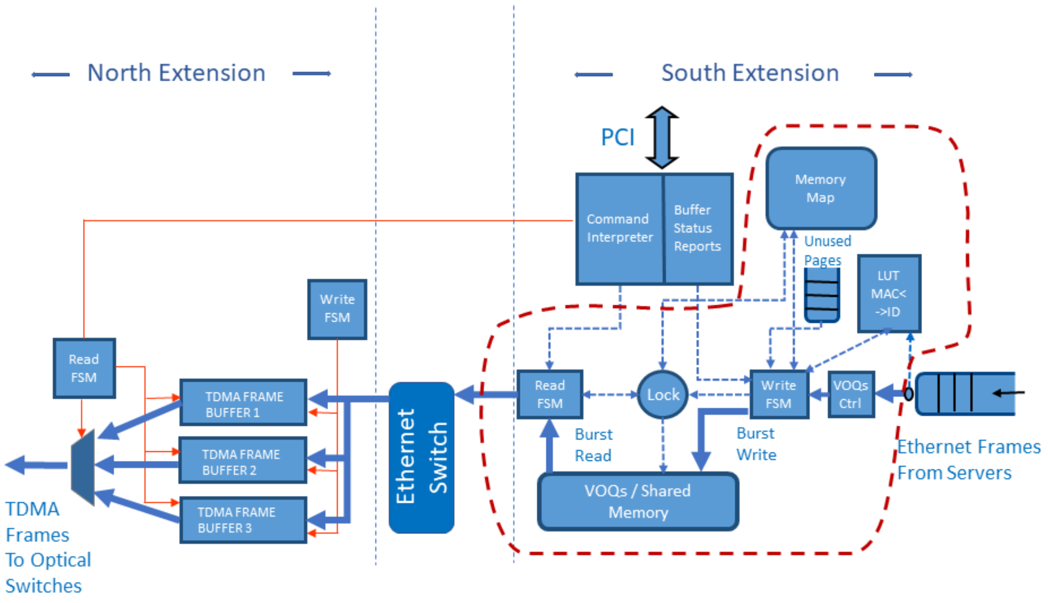


Figure 2.2: The Nephele Top-of-Rack (ToR) Switch Architecture Overview

Figure 2.2 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 assigns a tag to each incoming Ethernet frame. These 11 bit 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 list are managed by the Memory Map block.

The Command Interpreter block (Figure 2.2) 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 2.2) 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 2.2) 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) (Jacobsen, Richmond, Hogains, & Kastner, 2015). The RIFFA framework consists of an API (Application Programming Interface), a driver/kernel module for the host PC and an 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 IP core for PCIe, which handles the physical layer of the PCIe interface. The control of the ToR switch is presented in the next Chapter, more specifically in subsection 3.3.3, in which the implementation of the PCIe interface is presented in more detail.

2.4 Virtual Output Queues

This Section 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 chapter 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 proposed VOQ design improves the required hardware resources based on the following concept. During a narrow time window TB, 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 active destinations, during a narrow time window TB, has an upper limit, which can be an outcome of statistical measurements of the network traffic patterns across the data center. 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 TB 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 a 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 2.2 and third, the VOQs controller. The detailed architecture of the VOQs controller is shown in Figure 2.3: 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 2.3 there is one queue to support each active destination. In order to define the length of the time window TB we consider the following facts. 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.

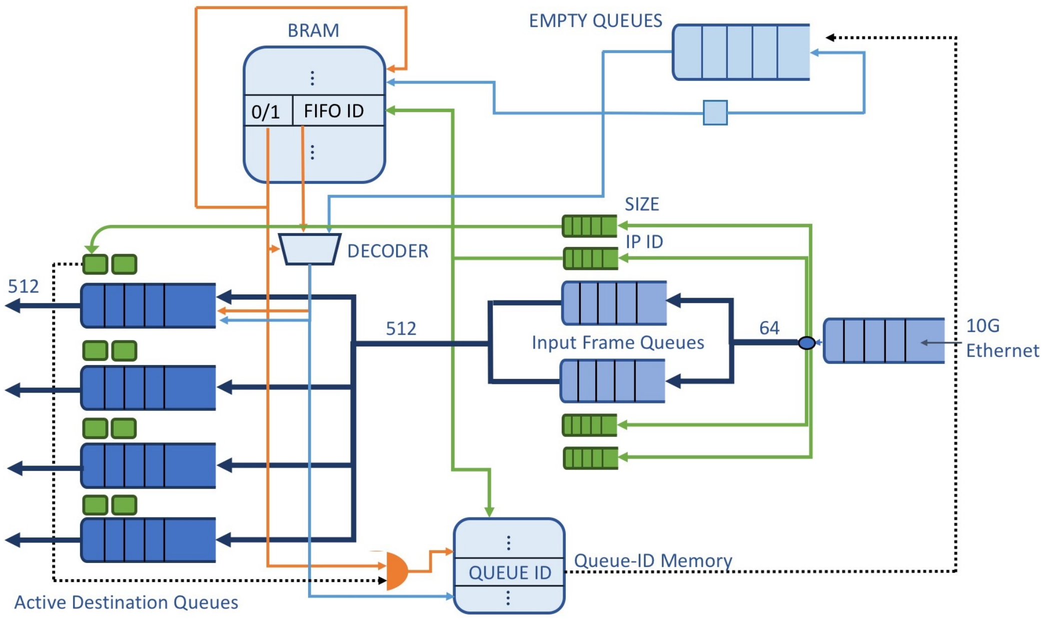


Figure 2.3: VOQs Controller Architecture Overview

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 IPs, 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 $k \times t_m b$.

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's linked list of pages; b) minimize latency and c) minimize the number of the k queues along with their size. The succeeding 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 2.3) 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 2.3) 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 2.3.

Each frame's IP stored in the *Input Frame Queues* is passed as input (address) to a LUT, named BRAM in Figure 2.3. The LUT will specify (will give as output) the id of an *Active Destination Queue* (on Figure 2.3 we show 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 in the linked list of pages of that destination in the DRAM buffer. Apart from 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 2.3. 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 2.3 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 2.3 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 2.4, 2.5 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"

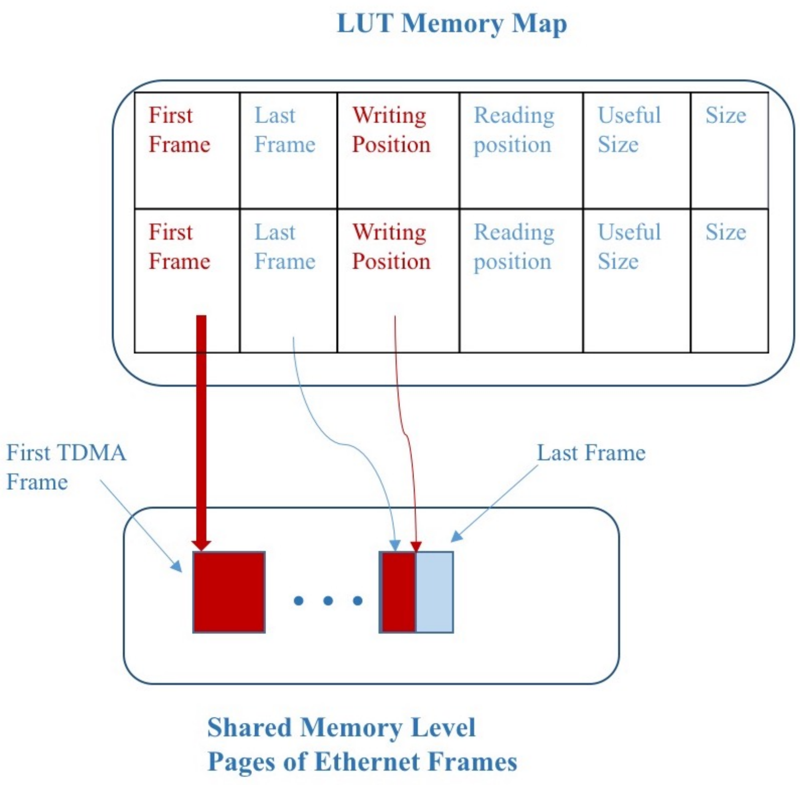


Figure 2.4: Memory Map Organization while Buffering

address of the last page (writing position in Figure 2.4), one for the "next to read" address of the first page (reading position in Figure 2.5). 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 the padding.

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

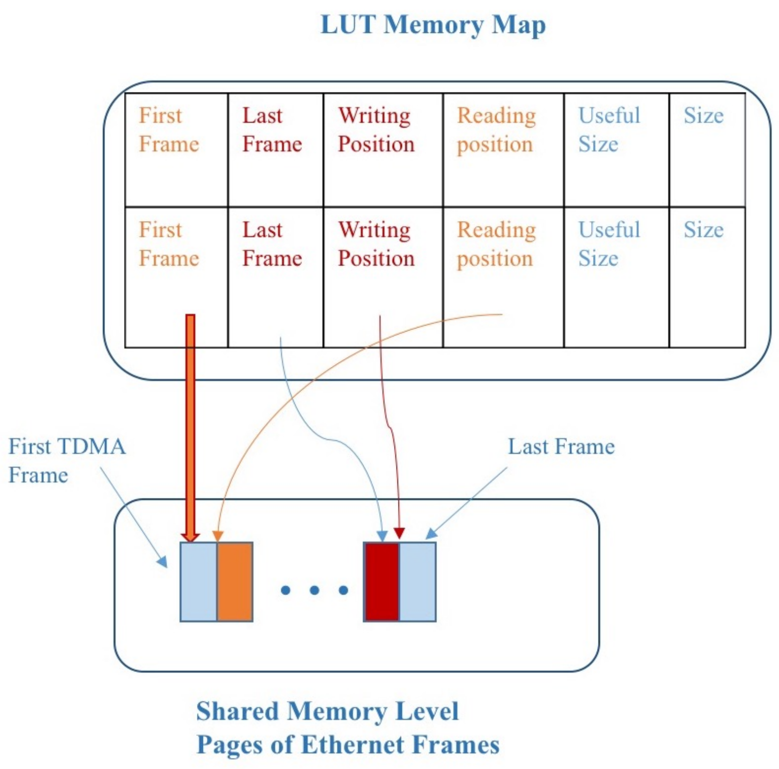


Figure 2.5: Memory Map Organization Concurrent Write & Read Operations

write and read from the same linked list of pages as shown in Figure 2.5.

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 case the size of the mapping information is relatively large, hence constraining the implementation of the Memory Map block with internal BRAM memory, the proposed Memory Map can be stored on external Static Random Access Memory (SRAM).

2.5 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.

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 to match the bus width of the Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface 4 (AXI4) of the Double Data Rate 3 (DDR3) DRAM controller.

Table 2.1: Resource Utilization of VOQs Controller

Resource	**Utilization**
LUT	2639
LUTRAM	1285
FF	4848
BRAM	62
DSP	50743

3 | Tools for Data Center Control

3.1 Introduction

Currently, the integration of Information Technology (IT) activities and applications takes place in data centers, which also include the necessary devices for communication, high performance computing and data storage. Data centers play an important role in organizations based on IT services, as they provide the means for fast responses to business demands, they facilitate the IT operations and their utilization leads to the reduction of the capital expenditures and the operating costs. Targeting the improvement of data centers, researchers and engineers focus on the use of optical switching due to the bandwidth capabilities that it provides. A significant contribution to this design effort features optical links connected through optical PoD switches to the ToR switches, SDN with OpenFlow organization, an Agent connecting the SDN controller and the data plane and an enhanced agent management tool (Kyriakos, Tsavalos, & Reisis, 2017) , which all integrate in the Nephele data center (Bakopoulos et al., 2018).

The current Chapter presents a management tool for the Agent of the Nephele's data center. The advantage of creating and using the proposed management tool is that the data center designers and engineers can create their own schedule as the tool's GUI users and then transfer that schedule to each data plane ToR switch. The user can control graphically in real time the transmission of Nephele frames originating at the ToR switch to the other Nephele ToRs in the data center network. Moreover, the management tool can be of even further use if it will be extended to create the scheduling tables of a PoD switch in the Nephele network. The first Section of the Chapter highlights the Control of the Nephele data center and the Agent. The second Section presents the Agent's management tool.

3.2 Background on Nephele's Data Center Control

The Nephele is based on a dynamic optical network infrastructure for scale-out, disaggregated datacenters that leverages optical switching with SDN control and orchestration to overcome current datacenter limitations. The Nephele design follows vertical end-to-end development approach extending from the data center architecture to the overlaying control plane and its interface to the application, in order to deliver a fully-functional networking solution, extending network virtualization to the optical layer. The Nephele design achieves dynamic reconfiguration by utilizing the slotted operation of the network based on the Time-Division Multiple Access (TDMA). Moreover, the SDN control can effectively manage the data plane elements. The OpenFlow protocol communicates the SDN control's messages to the data plane (McKeown et al., 2008). Nephele uses an Agent to realize the communication between the SDN controller and the data plane. The Agent includes functions filtering the control plane (SDN controller and the Agent) instructions that are transmitted through the OpenFlow messages; the Agent translates these messages and forwards them to the corresponding ToR switch. Although, the Agent can be classified as a back-end process, there is a need for an interactive management tool that allows the interaction of the designers and the future users with the Agent. The need for the above tool appeared in the course of the data center's design and implementation phase, it became more emphatic during the integration and finally the validation and testing phases. Similar interactive tools are reported in the literature as important tools for the management, testing and evaluation of networks (Lin & Geigel, 1997), (Turon, 2005), (Corazza & Reale, 1992).

Focusing on providing an effective tool mainly for advancing, testing and monitoring the Agent's functionality and performance (Landi et al., 2017), the current Chapter presents a management tool for the Nephele Agent. The proposed Agent's tool is able to access all the information that it is directed to the data plane. Moreover, it can be used to create the commands for the data plane, monitor the commands transmission to the devices and also, the corresponding responses of the devices to the Agent. Furthermore, it provides the ability to request all the information with respect to the status of the devices. The use of the proposed management tool contributed significantly to the development of the entire Nephele data center and consequently the testing phase. Additionally, it benefits the entire system because it will still be most suitable for effectively monitoring the Agent's performance during normal operation and also it provides the means for realizing scenarios in the cases of demonstrations and presentations (Landi et al., 2017).

The architecture of the Nephele data center is presented in Chapter 2, in this chapter we will elaborate on the control plane of the data center. The Nephele data center is designed for an operation that includes dynamic and efficient sharing of the optical resources and a collision free network operation by using Time Division Multiple Access (TDMA). The control plane is based on a Software Defined Network (SDN). The SDN controller is divided in two distinct interfaces, namely the Northbound Interface and the Southbound Interface. A high-level view of the Nephele control plane architecture is presented on Figure 3.1.

The Application to Controller Plane Interface defined by ONF (Open Networking Foundation) in the SDN architecture is realized by the Northbound Interface of the Nephele SDN controller. This interface allows the interaction between the core services of the Nephele SDN controller and the upper layer network applications, which implement the logic of the network resource allocation in the data center. The Nephele's design follows the approach of an overall centralized architecture. For this purpose, all the scheduling plans are carried out according to the algorithms that are performed by the central controller's Traffic Offline Schedul-

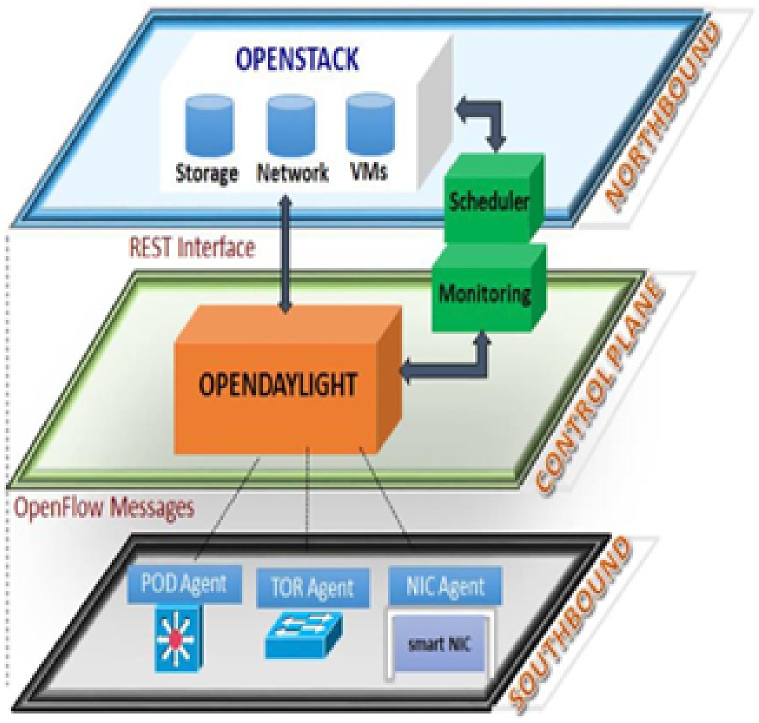
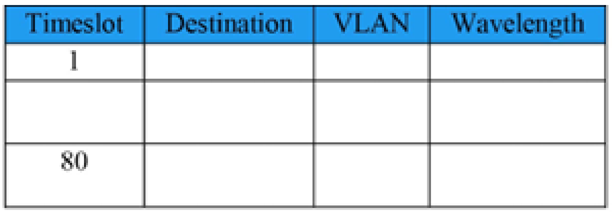


Figure 3.1: Nephele SDN Control Plane

ing Engine (Christodoulopoulos et al., 2016). Considering the optimization of the utilization of the entire network the Offline Scheduling Engine is equipped with mechanisms able to allocate resources of the data center network in the long term.

The data-controller plane interface defined by ONF in the SDN architecture is realized by the Southbound Interface of the Nephele SDN controller. The commonly used in these cases OpenFlow has been chosen as a standardized communication channel for this interface. It executes two main tasks: to command and configure the data plane devices via the device specific Agents. A device specific Agent performs as a proxy for the data plane switching devices. Consequently, the Agent should have two communication interfaces the Agent-Controller interface and the Agent-FPGA interface. The Nephele Agent's is mainly devoted to filter the control plane instructions, that are included in the OpenFlow messages. Additionally, the Agent translates these instructions and then, it forwards them to the corresponding FPGA via a PCI Express interconnection. The Agent is a back-end process. It is activated at the beginning of each Nephele scheduling period and it will communicate the new schedule instructions in order to configure the data plane switches. The instructions come in the form of scheduling tables; the format of these scheduling tables is presented by Figure 3.2.



Timeslot	Destination	VLAN	Wavelength
1			
80			

Figure 3.2: The Format of the Scheduling Table

3.3 The Management Tool of the Agent

The present section describes first the graphical user interface (GUI) architecture of the management tool of the Agent; second, the tool's usability and third, the back-end of the management tool.

3.3.1 The GUI Architecture

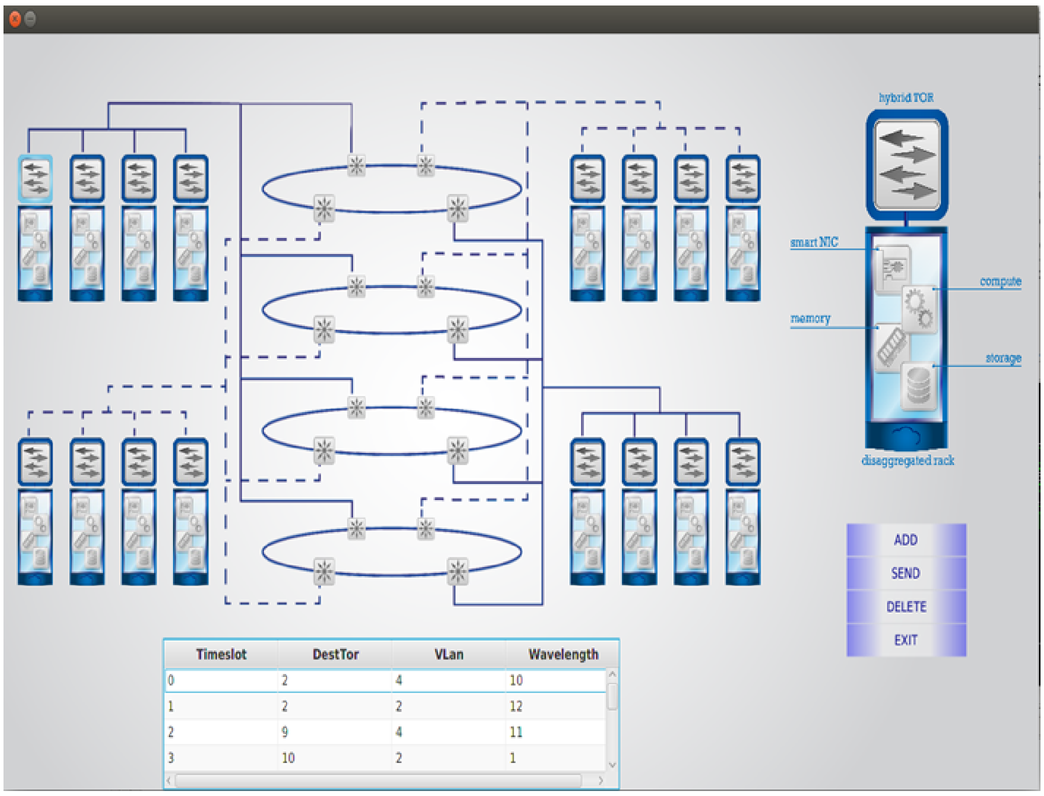
The Agent's management tool is implemented by using the JavaFX software platform of the Java programming language; JavaFX consists of a set of graphics and media packages, which provide the means to the developers for the design, creation, testing, debugging, and deployment of rich client applications that operate consistently across diverse platforms. The management tool includes a GUI that presents to the user a Nephele network of smaller size as an image-map. This image-map includes clickable areas, which are illustrated graphics created on a raster graphics editor and enhanced with interactive attributes. This design has led to the implementation of a graphic environment, which, considering the interaction of the user with the management tool, ensures both, optimized usability and user experience, compared to an environment using the standard widgets, provided by JavaFX.

The user of the management tool sees the data center network, the scheduling table, an explanatory image and a menu, which are brought to her/him as the main scene of the GUI. This main scene is shown in Figure 3.3. The smaller scale network includes four PoDs residing in the network and connected via four WDM (Wavelength Division Multiplexing) rings. Each of the PoDs includes four PoD elements; these are divided into the disaggregated rack and the ToR switch.

The GUI includes an explanatory image, that is located over the menu in the right top corner. The image presents an enlargement of a PoD element in higher resolution and it is augmented with annotations, so that the user is able to understand what the image portrays.

In order to present the graphic display of the PoD elements three objects of the ImageView class were stacked in a StackPane object (Johan Vos, 2014). This design has been implemented as follows: three image layers have been aligned one over another (depicted by Fig. 3.4), so that they appear as a single solid object and at the same time the developer can handle each one independently. The ImageView object is a type of Node object in the JavaFX Scene Graph that is used for painting a view; the painting is carried out by using data contained in an Image object. The StackPane is also a type of Node object acting as the layout container and it contains the ImageView objects. The three ImageView objects include the images that represent the ToR switch, the disaggregated rack and a visual effect.

In the GUI, the ToR switches are the interactive parts of the management tool: the



Timeslot	DestTor	VLan	Wavelength
0	2	4	10
1	2	2	12
2	9	4	11
3	10	2	1

Figure 3.3: The GUI Main Scene

user can select by clicking on them and she/he can create the scheduling table of the data center. Each ToR is a clickable area and it can be used by the user as the source and/or the destination in the scheduling table entry. In our case, the upper left ToR is chosen by default as the host Agent PC scheduling engine. This is the source ToR and the remaining ToRs are the destinations. The interactive feature is accomplished by registering an event handler on the ImageView object that includes the ToR image. An event handler is an implementation of the EventHandler interface. The handle() method of this interface will let the code filling the entries in the scheduling table to perform if the ToR image is clicked. Upon the cursor click event, all the necessary code is executed to fill in the required fields of a scheduling table's entry. The management tool fills the Destination field with the identity (id) of the ToR switch where the event occurred. The Timeslot field takes the value of the time sequence of the event, which is calculated based on a counter. The Wavelength field is filled with a value selected from a closed interval of integer values. Finally, the VLAN (Virtual LAN) field entry represents the identification number that is assigned to the WDM ring, through which the data transmission will occur. Furthermore, when the ToR is clicked, as depicted in Figure 3.4, it will trigger the effect displaying that it is the selected ToR. The effect is represented by a brighter image enclosing the ToR switch. The effect is set not to be visible at first, it will be set to full opacity if the ToR is selected and it will return to zero opacity with a two seconds lasting fade transition. The fade transition is an instance of the FadeTransition class, which is a subclass of the JavaFX Animation class and it changes the opacity of a node over a given time. The same effect has been implemented similarly to the WDM rings and it indicates graphically what WDM ring is chosen based on the VLAN field in the scheduling table entry.

All the aforementioned elements of the tool's design let the user to construct the scheduling table and provide the option of editing it; this operation can be carried out by the use of the menu. The menu consists of four buttons and inherits its attributes from the Vbox class, which is a container that sorts its contents into a single vertical column. The menu buttons were created as a separate class. It is distinct from the Button class of JavaFX and is created by stacking a TextField object over a filled Rectangle object. This object's filling is colored by an instance of the LinearGradient class, in order to apply effects that are suitable to the entire design of the GUI and preserve the uniformity to the user eye. These effects are triggered by the events originating from the mouse cursor and their implementation is based on switching the order of the colors in the gradient fill. Each time the user clicks the Add menu button she/he will start a new session of constructing a scheduling table and the source ToR will be automatically selected

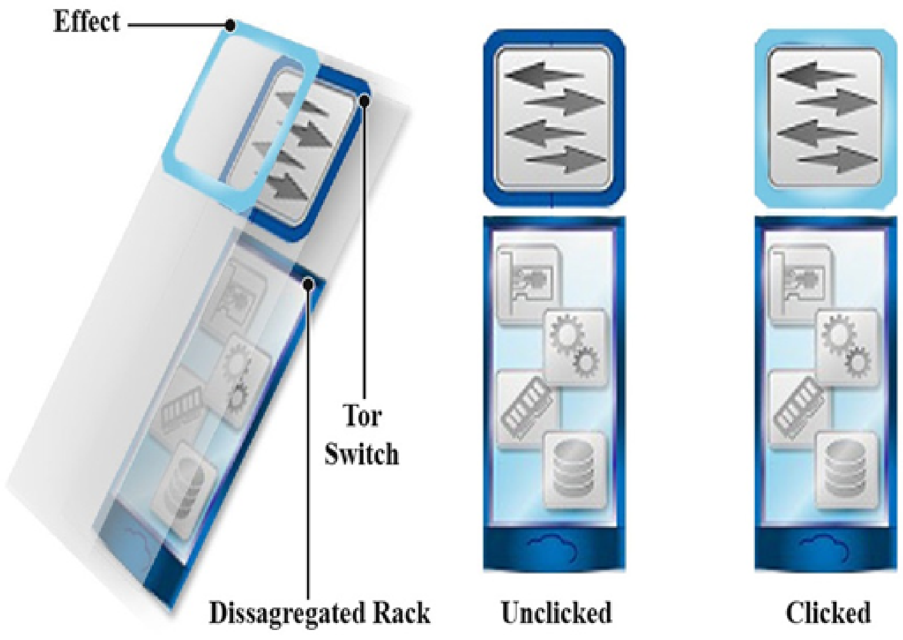
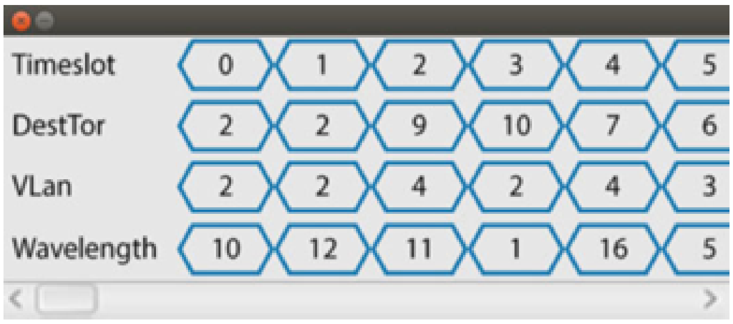


Figure 3.4: Effect of clicking the ToR

and indicated. A scheduling period of the Nephele network can accommodate up to eighty entries, as the corresponding allowed time slots. If the user exceeds that ceiling, a pop-up dialog box will emerge with the corresponding message, prompting her/him to stop importing entries. The dialog box prevents the user from interacting with the main application window but it keeps the window visible in the background. When the user has completed the creation of the scheduling table, she/he is able to review it and delete any misplaced entries by using the delete button from the menu. If the key is pressed and no entry is selected or the scheduling table is empty, a pop-up window will be called informing the user of the corresponding case. As a final step the user presses the send button, an action which transmits the scheduling table to the FPGA data plane devices.

The conclusion of the transaction is marked by the appearance of a pop-up window that it will be shown to the user. The window includes all the values that were sent

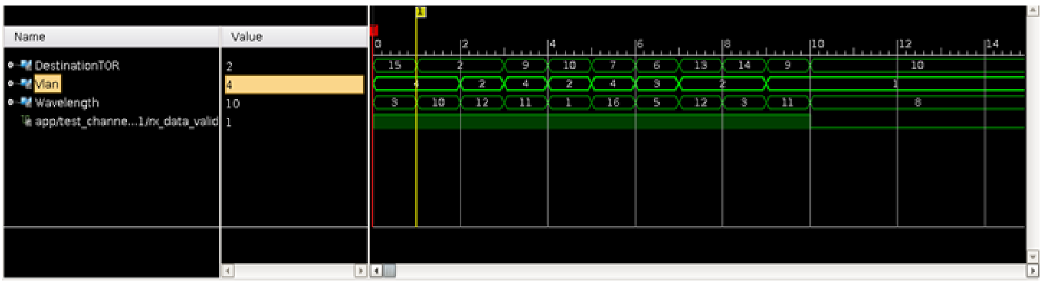


Timeslot	0	1	2	3	4	5
DestTor	2	2	9	10	7	6
VLan	2	2	4	2	4	3
Wavelength	10	12	11	1	16	5

Figure 3.5: Pop-up Window with the Values sent to FPGA

to the FPGA in a format that resembles that of a logic analyzer. The pop-up window is shown in Figure 3.5 and the corresponding output of the logic analyzer is depicted in Figure 3.6. The logic analyzer exports the output as a CSV file (Comma-Separated Values); this file can be processed by the management tool and in this case, the file's values will be forwarded to the pop-up window. The pop-up window incorporates the graphical theme of the management tool and is designed to model the layout of the logic analyzer.

3.3.2 Usability of the Agent's Management Tool



Name	Value
DestinationTOR	2
Vlan	4
Wavelength	10
app/test_channe...1/rx_data_valid	1

Figure 3.6: Output of the Logic Analyzer

The use of the management tool is of great importance to the development and operation of the data center, since the users can create their own traffic schedule and then transfer that schedule to the data plane ToR switch. The engineers are able to control the data plane switches, during the development and testing phase of the physical layer of the data center network as shown in Fig 3.7. The tool's GUI allows to construct the commands directly in the format of the scheduling tables of the FPGAs (instead of using the OpenFlow protocol). Additionally, it is straightforward to extend the management tool for creating the scheduling tables of a PoD switch in the Nephele network. Given the fact that there is a ToR Agent PC for each ToR switch in the network, the tool is executed on the Agent computer and it provides the user with the means for the scheduling of the network from the view point of the specific ToR switch. The user can control graphically and more importantly in real-time the transmission of Nephele frames originating at the ToR switch (that is controlled by the Agent computer) to the other Nephele ToRs in the data center network (Chen JW, 2007).

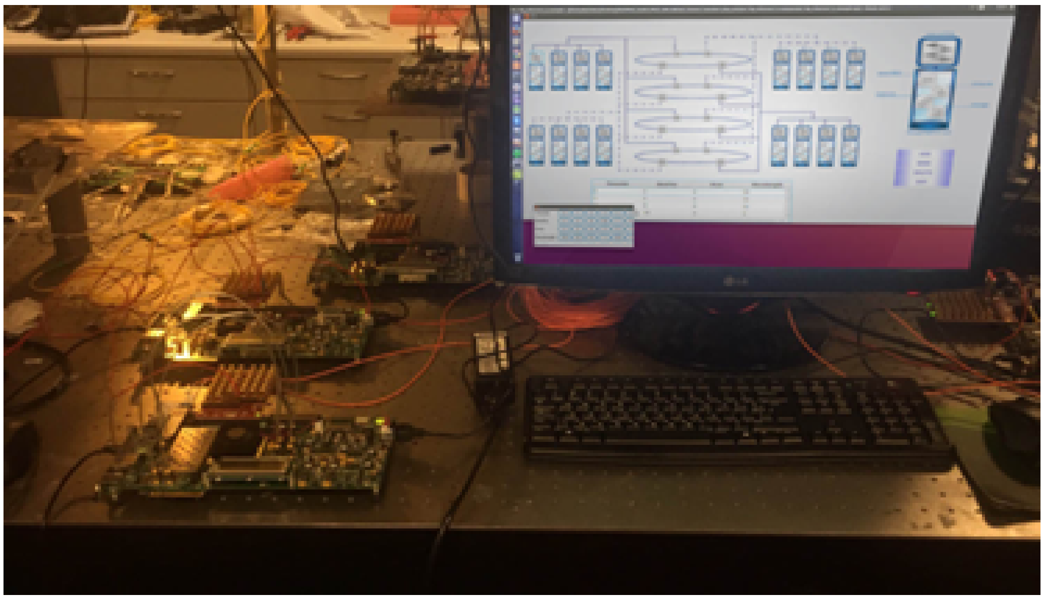


Figure 3.7: Nephele Data Plane Development

The benefit of designing, developing and effectively using the proposed management tool has already been proven during test procedures and demonstrations. An illustrious example is the application, which has been shown during a presentation of the control plane of the Nephele data center. The scenario for this demonstration has as follows: the control plane includes parts of the FPGA's implementations of the data plane, the Agent, and the SDN controller. Given that a functional data center Agent was not available, we presented the control plane by dividing it into two experiments. The first experiment demonstrates the SDN controller and the second the FPGA's operation controlled by the management tool. The management tool has successfully imitated the functions of the Agent; the majority of the people that interacted with the management tool understood the concepts behind the architecture of the Nephele network and the function of the Agent in the Nephele data center. The demonstration as shown in Figure 3.8 consists of the SDN controller software presentation, the FPGA that represents the ToR switch, and the Desktop PC that executes the management tool, which is connected to the FPGA board via PCI Express. The user can interact with the management tool and give his/her own commands to the demonstration system.

3.3.3 Back-End of the Agent's Management Tool

In the Nephele data center the ToR switch design includes multiple FPGAs; all the FPGAs that belong to a single ToR implementation use PCI Express to communicate with the host ToR Agent computer. The management tool divides the scheduling information into distinct parts, so that each part corresponds to the scheduling information concerning the corresponding FPGA; then it creates distinct threads to complete the entire operation. We use a single thread to communicate with a single FPGA and transfer the respective part of the ToR switch traffic schedule. Note here that, the communication is performed in parallel for all the FPGAs belonging to the same ToR switch.

In order to develop the PCI Express interface of the FPGAs we used the Xilinx IP Core for PCI Express and the RIFFA (Reusable Integration Framework for FPGA Accelerators) framework (Jacobsen et al., 2015). The framework consists of an

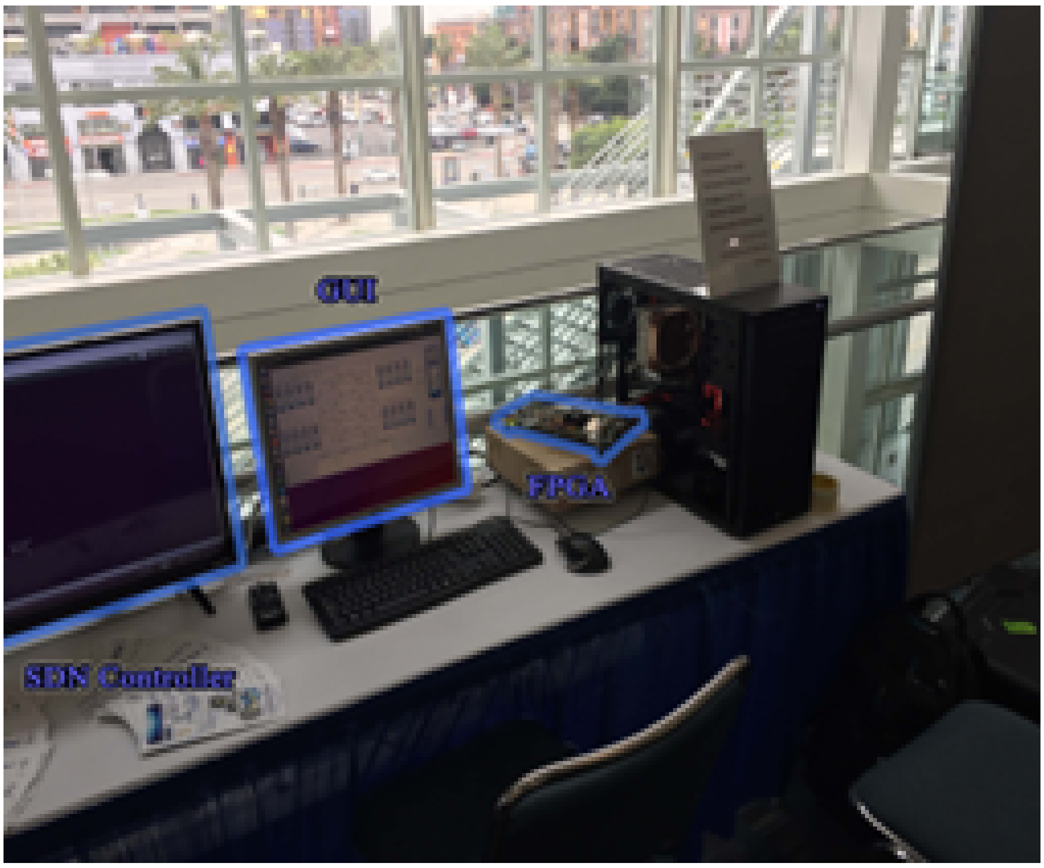


Figure 3.8: Live Demo of the Management Tool

API (Application Programming Interface), a driver/kernel module and an IP core for the FPGAs. All the above parts are open-source. It is designed to perform with the Xilinx IP core that handles the physical layer of the PCI Express interface. The API is designed to support multiple languages like C/C++, Java and Python. Moreover, it includes the necessary function/methods that the management tool needs to invoke, in order to communicate with the FPGA. The entire API is designed to be executed by threads and the design of the management tool takes full advantage of this capability.

The communication that is directed from the Agent PC to a FPGA operates according to the following steps. In the first, the application initiates the transaction by calling the fpga_send method. Then, the thread invokes the operation of the kernel driver, which writes to the FPGA configuration registers the necessary information to begin the transaction. The FPGA uses DMA (Direct Memory Access) to read the scatter gather elements (Jacobsen et al., 2015) that the driver instructed. At the time that the transaction will be completed the driver will read the final count of the data passed, the amount of the data is then returned to the management tool as the return value of the fpga_send method.

In the design of the tool special attention was paid to the operation of the RIFFA API, because the RIFFA's driver requires all the data in contiguous memory locations (in an array). Note here that, the Java's Array object can't be used in this case. An attractive solution to this problem is the employment of the ByteBuffer Class of Java, which is a class that is created to handle a stream of raw data. The operations on the buffer can be carried out byte by byte, but casting is also supported for the user to be able to write a whole Java data type, like an integer.

Finally, the endian of the data has been tackled as follows. The JVM (Java Virtual Machine) stores class files in big endian byte order, where the high byte comes first. Multibyte data items are always stored in big-endian order. Given that the Xilinx FPGAs operate in little-endian byte order, the change of the endianness could be arranged either during the construction of the ByteBuffer or at the receiving buffer in the FPGA. The latter choice has been proven more efficient and gave us the advantage of the ByteBuffer casting, which would not be useful in the case of changing the order of the byte inside the ByteBuffer in the Java application.

4 | Neural Networks on FPGA

4.1 Introduction

The evolution of FPGAs with respect to the increased hardware resources and the efficiency of their programming tools has affected significantly the applications with real-time specifications. Deep Learning techniques (Mordvintsev, Olah, & Tyka, 2015) and CNNs, benefit by the utilization of FPGAs as accelerators to accomplish real-time performance (Abdelouahab et al., 2018). FPGAs are advantageous for these tasks because of their ability to reconfigure and/or reprogram the architectures and consequently, the designer can follow the continuous improvement of the CNN algorithms and techniques. Among the aforementioned processes, those that are destined for edge, mobile and on-board satellite (OBC) computing have to use accelerator designs that are performance, power and resource efficient. Aiming at improving the performance of these tasks, the current chapter presents a design approach for real-time classification FPGA accelerators that can be implemented with the logic and memory resources of a single FPGA device and it shows its advantages by developing a Vessel Detection FPGA Example Accelerator.

The proposed approach is effective for CNN applications with relatively low feature space (Lei, Liu, Dai, & Ling, 2020), (Kyriakos, Kitsakis, Louropoulos, Papatheofanous, & Patronas, 2019), (Li, Lin, Shen, & Brandt, 2015) such as the classification problems that share similar characteristics between classes (Sermanet & LeCun, 2011), (*Airbus Ship Detection Challenge*, 2019), (L. Wang et al., 2018) and CNNs requiring few convolution layers such as SAT-4/SAT-6 (Gorokhovatskyi & Peredrii, 2018). The proposed FPGA design approach includes three phases with each phase targeting distinct design and performance gains. The first phase introduces guidelines that lead the CNN design process with TensorFlow to a model of reduced computational and memory requirements but of high classification accuracy. In the second phase, the model is transformed into a fixed-point bit-accurate model (BAM) simulating the hardware calculations and allowing the designer to decide on the arithmetic representation of the model's parameters that provide the optimal trade-off between bit-width reduction and accuracy losses. For the third phase, we developed a library of algorithm specific blocks in VHDL implementing the CNN functions with fixed-point arithmetic. These blocks, along with our proposed methodology for mapping the CNN to the FPGA, provide the means to the FPGA designer to initiate the third phase and implement a distinct module for each CNN layer. The completion of the third phase places these modules in a pipeline fashion forming a streamline architecture, to result in an efficient FPGA accelerator with respect to power consumption and resource utilization while saving significantly on the development time.

The chapter is organized with the following section reporting the background work in the area of the CNN accelerators in the literature. Section 4.3 introduces the approach for designing the CNN and mapping them on the FPGA. Section 4.4 describes the example FPGA accelerator. Subsection 4.4.1 presents the necessary background for the target example application that is the vessel detection. In subsection 4.4.5 the corresponding FPGA and performance results are presented in detail.

4.2 Background on CNN FPGA Accelerators

Researchers have studied and provided FPGA accelerator solutions for CNNs based mainly on the automated software development tools like the HLS (Kim et al., 2017), (Solovyev et al., 2018), (Zhang et al., 2015), (Sankaradas et al., 2009), (Peemen et al., 2013), (B. Liu et al., 2019), due to short development time and hardware abstraction. The approach followed in (Kim et al., 2017) improves the time of the entire design process by parallelizing the CNN C code with Pthreads and optimize the FPGA accelerator through software/constraint changes only. The authors of (Solovyev et al., 2018) target feasibility at low cost by choosing inexpensive FPGA devices and cores for their accelerator. The authors of (Zhang et al., 2015) focus on optimizing the accelerator's performance by considering the architecture's throughput combined with the external memory's throughput. The FPGA accelerator of (Sankaradas et al., 2009) interfaces with a host PC and it utilizes off-chip memories with the loading/storing of the intermediate results optimized for higher bandwidth. The (Peemen et al., 2013) reports an FPGA accelerator template with an HLS FPGA architecture consisting of a cluster of Multiply Accumulate Processing Elements for convolutions acceleration; this work focuses on a design flow selecting processing schedules that minimize external memory accesses and buffer size by means of data reuse. The authors in (B. Liu et al., 2019) present an accelerator based on a single-processing engine that targets standard and depthwise separable convolution. In this work the authors aim to reduce the delay added by the off-chip memory data exchange by using a data stream interface and ping-pong on-chip cache. All the HLS design approaches though prevent experienced designers from optimizing the HDL code towards a more efficient FPGA architecture with respect to resource utilization, throughput and energy consumption (Pelcat et al., 2016). The authors in (Rapuano et al., 2021) present an on board satellite FPGA accelerator for CNN inference, which utilizes a single processing unit with external DRAM memory, developed with VHDL code. Note here that, the current work focuses on streamline architectures that implement the contiguous CNN layers in a pipeline fashion and differs to the implementation of a systolic array that is reconfigured each time it computes a CNN layer (Zhang et al., 2015). Hence, the advantages of the proposed designs are to avoid idle computing and memory (Y. Zhao et al., 2019) resources, use only the on-chip (FPGA) memory and extensive pipeline, features that lead to improved resources utilization, reduced latency and power consumption (Sze et al., 2017), (Lamoureux & Luk, 2008).

4.3 CNN Design Approach

The current section introduces the three distinct phases of the proposed FPGA accelerator design approach. It begins by presenting the first phase with the guidelines for the CNN model design. Then, for the second phase, it describes the development of the fixed-point BAM representation of the CNN floating-point model based on the factors, that play a key role in the design of the entire FPGA accelerator. Finally, the third phase introduces the configurable VHDL blocks and the mapping methodology of the CNN layers to the FPGA by utilizing these blocks. The result is to map the CNN layers on a pipeline of modules, where each module is optimized to the corresponding layer computations. The proposed streamline architecture designs save significantly on the FPGA resources compared to the architectures that implement all the CNN layers on a systolic array (Zhang et al., 2015) and leave idle resources as the layers progress.

4.3.1 CNN Design Space Exploration

This work focuses on single FPGA device solutions for classification applications and more specifically, binary and limited feature space classification tasks. Consequently, the design process has to consider all the factors reducing the resources' requirements. For this purpose, in the first phase the designer will use the TensorFlow estimator API to design the CNN's model targeting to fit within a single

FPGA's resources. Focusing on all the key factors of the data under consideration the designer can develop the model by keeping to the following guidelines for the:

- *Number of Layers:* the neural networks for the low feature space classification applications can achieve a high accuracy rating even with a relatively small number of Convolution Layers (Gorokhovatskyi & Peredrii, 2018).
- *Size of convolution kernels:* considering the input is relatively small, the recognized objects tend to occupy a large portion of the input data and hence, large and medium size convolution kernels suffice.
- *Choosing the size of the Pooling Layers windows:* the feature space is relatively limited and hence, the use of 4×4 pooling layers will not affect the accuracy meanwhile it will improve significantly the resources' requirements of the succeeding layers.
- *Avoid padding:* this should be implemented throughout the CNN because-most of the time it does not affect the accuracy at all.
- *Divisibility:* it refers to the divisibility of each convolution layer's output size by the kernel size of the succeeding pooling layer. If it is applied, it will: a) allow the omission of padding with no accuracy loss and b) lead to efficiently pipeline these contiguous layers.

4.3.2 Bit-Accurate Model Development

During the second phase the designer develops the BAM of the designed and trained CNN. The BAM emulates the exact same fixed-point calculations that the hardware accelerator will perform. For the BAM, we perform quantization of the CNN model trainable parameters, starting from the 32-bit floating point representation provided by TensorFlow to a desired $Qm.n$ fixed-point representation. The number of bits for the integer part m and fractional part n are accepted as input parameters to the BAM. This allows the designer to perform a trade-off study between saving on FPGA resources due to the reduced bit-width of the CNN parameters and maintaining high classification accuracy as a result of the reduced arithmetic precision.

4.3.3 VHDL Blocks

The proposed approach combines the VHDL advantages with an efficient, with respect to the developing time, design methodology for CNN accelerators. Multiple instances of configurable and reusable VHDL blocks, each with different configuration, are used for the development of each layer. The following subsections present these reusable VHDL blocks developed in this work.

4.3.3.1 Input Block

This block consists of a *Block RAM* that stores one input data channel and a *Window Generator* as shown in Fig. 4.1. The *Window Generator* formulates the input to the following Convolution Layer as windows (matrices) of size equal to the Convolution Layer's $n \times n$ kernel (e.g. 3x3, 5x5, etc). It uses n shift registers with each register containing one image row, in order to avoid the indexing of pixels and thus, lead to improved performance. The *Kernel Window Controller FSM* of the *Window Generator* reads n rows from the *Block RAM* and copies them into the first set of n *Shifting Registers*. The *DSP Decoder* formulates the $n \times n$ window: the first n pixels (memory words) of each of the n *Shifting Register*, are routed in parallel to the input of the following Convolution Layer. To create the next window we shift the n registers by one pixel. There are two sets of *Shifting Registers* forming a double input buffer. If the following Convolution Layer uses $n \times n$ kernels, the n shift registers forward an input $n \times n$ window per cycle to fully pipeline the two layers. Configurable are the: a) input data sizeb) the n registers, c) the kernel $n \times n$ and d) pixel bit-depth.

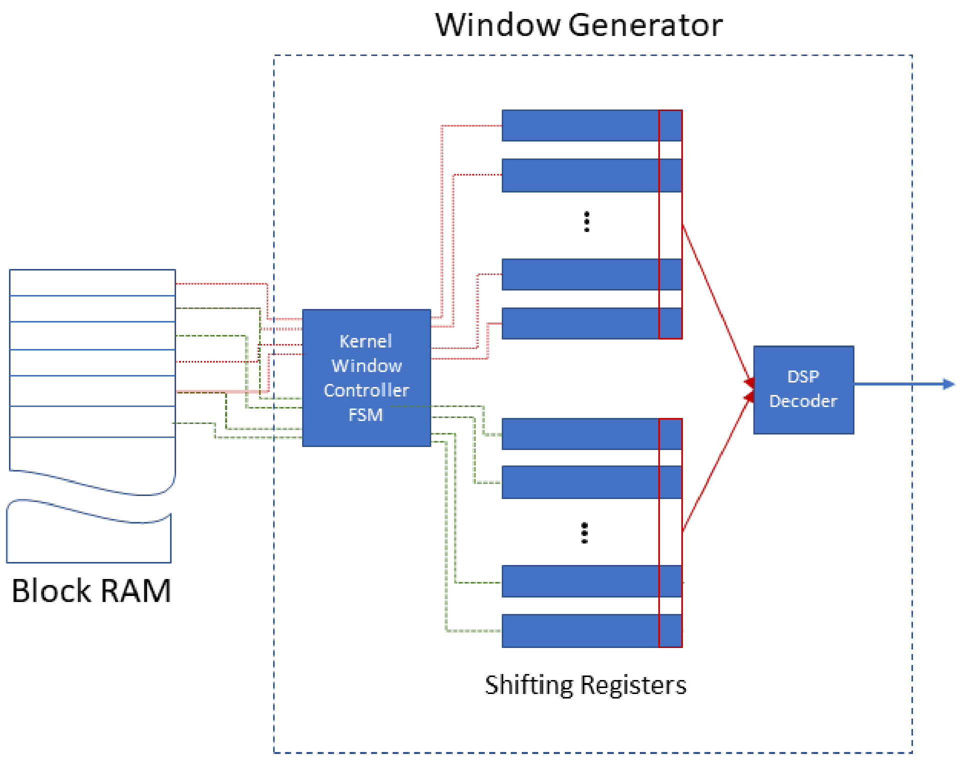


Figure 4.1: Input Block Architecture

4.3.3.2 Convolution Block

The *Convolution Block* (Fig. 4.2) receives a single channel of the input data (or a single feature map) in the format of kernel sized matrices ($n \times n$) and it calculates the convolution of a single filter's kernel with the input. The *Convolution Block* includes $n \times n$ multipliers; each multiplier has input one element of the $n \times n$ matrix and the corresponding kernel weight. Different filter kernels are stored at the on-chip ROM of the *Convolution Block*. To calculate the output of the *Convolution Block* a tree of adders (of height $\lceil log_2(n \times n) \rceil$) completes the addition of all the products of the multipliers in a pipeline fashion.

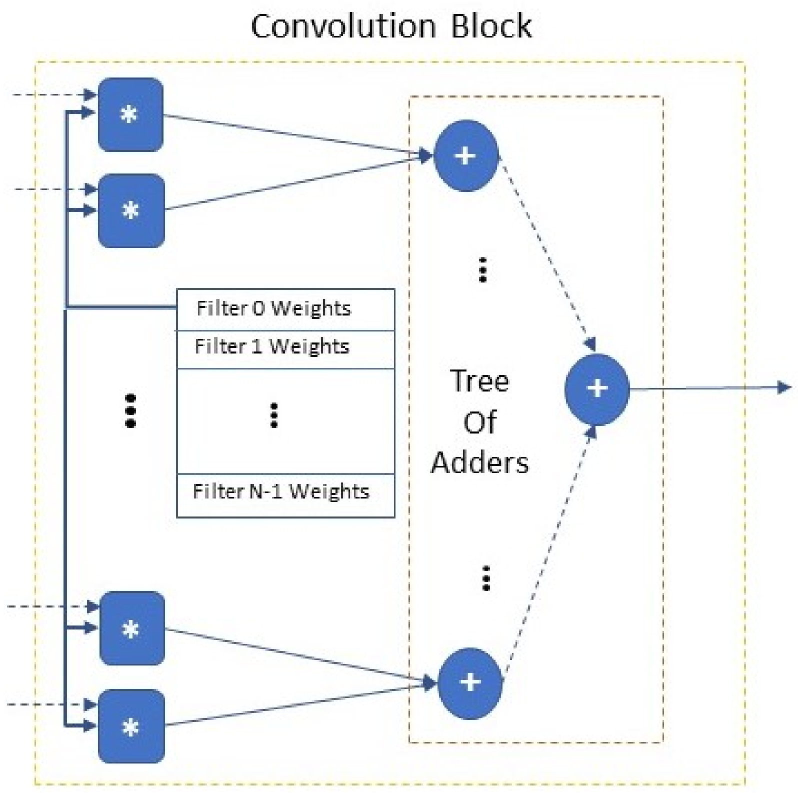


Figure 4.2: Convolution Block Architecture

4.3.3.3 Pooling Block

The *Pooling Block* (Fig. 4.3) receives the feature map produced by a preceding Convolution Layer: a $k \times k$ array forwarded one value at each cycle. The *Pooling Block* selects the maximum value of each $l \times l$ matrix, for all the matrices in the feature map with stride l (e.g. 2×2 or 4×4 max pooling) and outputs the $k/l \times k/l$ array of the above maximum values. In detail: first, from the $k \times k$ matrix the sub-block *Row Max Pooling FSM* gets the maximum of each l-tuple of values of each row to provide a $k \times k/l$ array; l registers are written in l consecutive cycles and we choose the max of the l registers. There are l *Pooling FIFOs*: the *Row Max Pooling FSM* stores the result in the next available FIFO and marks it as the active *Pooling FIFO*, i.e. the k/l results of the rows $0l$, $1l$, $2l$, etc. will be stored in the first *Pooling FIFO*, those of the rows $0l + 1$, $1l + 1$, $2l + 1$, etc. in the second and so on. When l rows of the output feature map ($l \times k/l$ values) are stored at the *Pooling FIFOs* the *Column Max Pooling FSM* starts the vertical max pooling; it chooses the maximum of l data (one from each *Pooling FIFO*) to produce the $k/l \times k/l$ matrix.

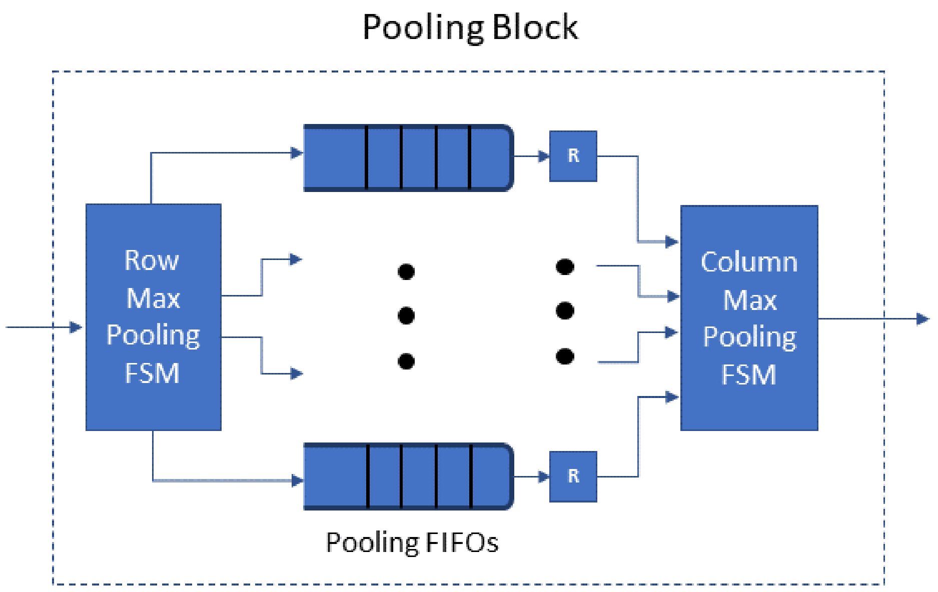


Figure 4.3: Pooling Block Architecture

4.3.3.4 Vector Multiplier

The *Vector Multiplier* realizes a Fully Connected Layer neuron; it computes the dot product of the 1-D input vector I (the flattened result of the preceding layer), received one point at a time, with the corresponding row of the Fully Connected Layer's weight matrix. The weight matrix W is stored in a ROM, where each memory word contains the weights of every neuron for each input. At each cycle, the input value of I and the corresponding row of W are multiplied and the block accumulates the result, which will be forwarded to the following blocks. This block is implemented with clock gating to limit the dynamic power consumption because it is operational for a short time compared to the rest of the modules. Disabling the clock for the design blocks that perform no computations at a given time prevents signal transitions limiting power consumption (Osborne, Luk, Coutinho, & Mencer, 2008).

4.3.3.5 ReLU and Output Block

The *ReLU Block* is a 2-to-1 multiplexer. The select bit of the multiplexer is the Most Significant Bit (MSB) of the input value. If the MSB/select is “1”, the input is a negative number and the multiplexer outputs zero, otherwise it forwards the input to the output.

The *Output Block* is the CNN’s final Fully Connected Layer. Its architecture is shown in Fig. 4.4. It executes the matrix multiplication of the flattened input array I with the weights W of the output neurons and then adds the Bias. In a pipeline fashion, it is executed once for each output neuron/class.

4.3.4 Methodology for Mapping the CNN on the FPGA

The current section describes the major considerations and recommendations for mapping the CNN functionality on a VHDL architecture by utilizing the above blocks. The proposed approach uses the mapping to result in a streamline architecture that implements all the layers of the CNN as a pipeline of modules: each module implements a CNN layer’s computations. This allows flexibility in the parallelization strategy of the computations of each layer (implemented as a

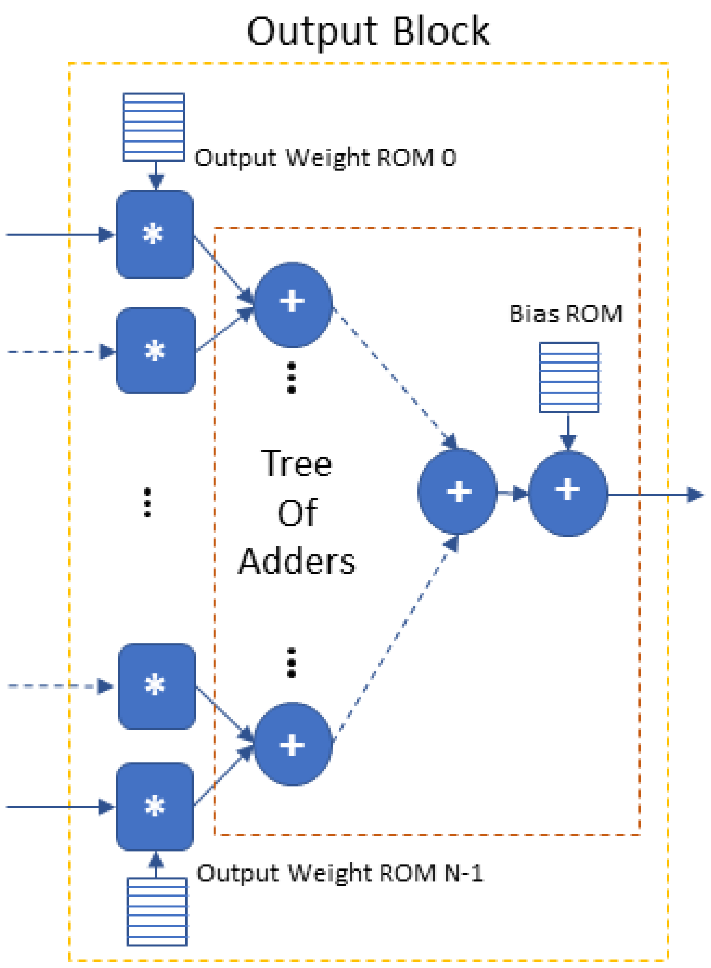


Figure 4.4: Output Block Architecture

module) and our proposed approach aims at parallelizing the layers in a way that enables extensive pipelining between them and minimizes the use of intermediate buffers. In more detail, for the acceleration of binary and limited feature space classification tasks with shallow CNNs that this approach targets, the streamline architectures and the proposed design approach have the following benefits:

a) High efficiency in resources utilization and computing since all hardware is generated specifically for each CNN layer (module) and the layers are pipelined.

b) Significantly reduced memory requirements for the intermediate results and

use of buffers only on the on-chip memory. The extensive pipeline of the proposed approach allows for succeeding layers (modules) to directly consume the data generated by the preceding ones and thus minimize the buffering of the intermediate results.
c) Reduced latency for shallow CNNs designed for the target limited feature space classification tasks. This is achieved by the parallelization strategy, the pipelining between the VHDL implemented layers (modules) and the use of only low-latency on-chip RAM. Moreover, pipelining a design can reduce the amount of energy used per operation compared to the non pipelined version at the same clock frequency (Wilton, Ang, & Luk, 2004).

The resource utilization, and power efficient design approach has to focus on the following characteristics. The key issue is to keep the memory and DSP requirements of the CNN accelerator design within the limits of the target FPGA device. Consequently, the objectives of CNN accelerator's design are first, the minimization of the buffering between consecutive layers second, the required memory of each layer and third, the real-time performance of the accelerator. The methods for improving the key issues of the FPGA accelerator are:

- *Buffers between layers and Speed-up:* The effort is given to parallelize the N filters in each Convolution Layer (except the first). Assuming that a Convolution Layer is designed with N filters, then the accelerator can have K parallel *Convolution Blocks* to complete the N convolution filters in N/K steps. The accelerator design with $K = N$ is preferable because first, it maximizes the speed-up second, it allows the pipelining of the input to every Convolution Block and avoids the buffer between this and its preceding layer.

- *Reduce the memory of each layer:* each Convolution Layer produces N

feature maps and apart the first accumulates these in N memories. The size of each of these N memories depends on the size and the number of the preceding Pooling Layers. We denote by $(sp_i)^2$ the dimensions of the $(i+1)^{th}$ pooling layer. If the input data are sized $Q \times Q$ and there are p Pooling Layers of sizes $sp_0 \times sp_0,\ sp_1 \times sp_1,\ \ldots,\ sp_{p-1} \times sp_{p-1}$ each memory (of the N memories of the current layer) has size $[Q \times Q] / \prod_{i=0}^{i=p-1}(sp_i \times sp_i)$. Hence, higher dimension pooling layers reduce the memory size and allow to implement N parallel filters with their individual memories.

- *The First Convolution Layer.* The proposed parallelization technique for this layer leads to the balance of the speed-up against the available number of DSP Blocks and Block RAMs of the target FPGA device. The key computational role is realized by a parallel *Structure* consisting of one Convolution Block per channel; these blocks compute the convolution of all the input data channels (e.g. 3 channels and 3 corresponding blocks in the case of an RGB image). Each block completes the convolution in real time and it forwards each result to the following Pooling Layer without a buffer, a design feature that significantly improves the memory requirements since the First Convolution Layer operates on the full size input data (without any downsampling). The use of one (1) *Structure* to complete all the filters of the First Convolution Layer is resource efficient. Depending on the target FPGA's resources, we can use k instances ($k \leq (sp_0)^2$, where $(sp_0)^2$ the dimensions of the first pooling layer) of this *Structure* in parallel to improve the speed-up by k. We note here that each additional parallel *Structure* first, adds a set of Convolution Blocks (one for each input data channel) increasing the use of the FPGA DSP Blocks, second it adds another memory buffer at the interface between the First Pooling Layer and the Second Input Layer. However, the size of each additional input buffer is considerably reduced due to the high dimensions of the First Pooling Layer. Using k such *Structures* and the k buffers is limited by the available DSP Blocks.

- *Scalability*. The aforementioned techniques lead to a scalable FPGA accelerator design. The architecture of the First Convolution Layer enables the engineer to opt for more performance or optimize the design for FPGA devices with limited resources. Moreover, the Fully Connected Layers can use a *Vector Multiplier* per neuron: parallelizing the neurons is advantageous leading to a layer design irrespective of the size and the number of the feature maps produced by the preceding layer; and more importantly it is scalable.

4.4 Vessel Detection FPGA Accelerator Example

Considering as target application the Vessel Detection, we exploit the proposed approach to design an example accelerator within a single FPGA device, which decides whether there is a vessel (*Airbus Ship Detection Challenge*, 2019), in the input image. This image classification task utilizes a CNN trained for the Planet's "Ships in Satellite Imagery" dataset (*Planet: Ships-in-Satellite-Imagery*, 2019) and the resulting FPGA accelerator using the resources of only the Xilinx Virtex 7XC7VX485T device (operating on a Xilinx VC707 board) achieves almost 98% prediction accuracy and high throughput by classifying a 80×80 RGB 24 bits/pixel image in 0.68 msec. Moreover, the accelerator can be used in a sliding window application for scenes up to 4K. To compare the FPGA's performance we execute our code on the low power Intel's Myriad2 processor (Barry et al., 2015) used for cameras and OBC (España Navarro et al., 2021), (Rapuano et al., 2021) and the edge-computing NVIDIA's Jetson Nano (*Nvidia Jetson Nano*, 2020) either on the Jetson's ARM processor or the GPU.

The following sections employ the proposed design approach, presented in Section 4.3 to develop a Vessel Detection Example FPGA accelerator that can also be used in sliding window applications of large images. The use of the example Vessel Detection FPGA accelerator can be realised in the context of an FPGA system, in which the accelerator interfaced with a host processor/FPGA-engine, is receiving windows of 80x80 for classification of an larger image stored in central Mass Memory (e.g. 3081x1597 in Planet's dataset used) obtained from the camera sensor.

4.4.1 Background on Vessel Detection

The vessel detection is among the most important tasks of the Maritime Domain Awareness (Dekker et al., 2013), (Kanjir, Greidanus, & Oštir, 2018) including all the activities associated with the maritime activities that could impact upon the security, safety, economy, or the environment and which are related to any navigable gateway and the associated infrastructure, people, cargo and vessels. For the corresponding applications the vessel detection is keystone because it has a very extended scope of applications in the areas of maritime safety and rescue missions, marine traffic control, sea pollution, maritime spatial planning, management of remote fisheries, area fishing control, illegal migration, customs control, observation of naval borders, etc. Calling as vessels the ships and all the floating manufactured objects and given that it is rather straightforward to distinguish an object in optical images produced either by space or drones or harbor cameras, the processes that identify the vessel in the image frames play a key role in the above applications.

Moreover, note that for the ships greater than 300 tons is mandatory to use shipborne transponders to report their position. Smaller ships though, do not have to own and use these devices and also, the ships that contact illegal operations either turn them off or they try to deceive the authorities with false reports of their position. Hence, vessel detection comes to support effectively the maritime domain awareness. Consequently, the exploitation of images and especially satellite ones plays a key role for locating vessels on the sea surface. Notable example is the satellite-based radar images most often as Synthetic Aperture Radar (SAR) that are of common use for maritime surveillance because they provide the ability to detect the vessels either in the case of cloudy skies or clear ones. The interest though in using optical images in the applications of maritime surveillance escalated significantly due to the availability of optical imaging satellites.

Considering the problem definition, the vessel detection can be envisaged as a task of detecting an object given that the background in most cases has the characteristics of the surface of the water. Following the latter model the researchers and the engineers focused on providing solutions in terms of automated analytical methods for remote sensors. These efforts are the consequence of the existence of the large number of Earth-orbiting sensors and their ability to generate and transmit big volumes of data. Hence, the detection systems have to process large volumes of sensor data and in many cases to conform to near-real or real time requirements. Accordingly, the limitations in the execution time as well as the restrictions in the power consumption and the resource utilization, call for power and resource efficient hardware accelerators (España Navarro et al., 2021). A generic approach for the vessel detection is to receive an input image of size $k \times k$ pixels, on which it will perform the calculations of the trained CNN model, the convolution with the filters kernels, the max-pooling and finally the classification with a fully connected neural network. This operation is repeated on overlapping image patches extracted from an large image of size $x \times x$ pixels, where $x >> k$, gathering the patches that contain vessels and discarding the remaining.

Regarding the results related to the Vessel Detection (Dekker et al., 2013), (Kanjir et al., 2018), that the current work has as target application, most of the published results exploit algorithmic techniques to improve the execution time. Widely known are the R-CNNs (Girshick, Donahue, Darrell, & Malik, 2013), Faster-RCNN (Ren, He, Girshick, & Sun, 2015), You Only Look Once (YOLO) (Redmon, Divvala, Girshick, & Farhadi, 2015), and Single Shot MultiBox Detector (SSD) (W. Liu et al., 2015). Another approach in (H. Zhao, Zhang, Sun, & Xue, 2019) recognizes the key parts of the vessel and classifies the ship's identity by using these key parts. These classification results are then voted for the decision of the ship's identity with achieved highest accuracy 92.63%. Hardware accelerators developed solely for Vessel Detection are in (Ji-yang, Dan, Lu-yuan, Jian, & Yan-hua, 2016) but without CNNs: they propose a technique based on statistical analysis, of the inspected and neighboring areas to distinguish the “possible ship" to other objects and by the geometric features of the target they decide whether the target is a ship achieving 90% success rate. The large number of approaches, algorithmic techniques and results related to the vessel detection is due to the attention that vessel detection as a task has gained the last two decades.

4.4.2 Model Architecture and Training

The model was trained with the “Ships in Satellite Imagery" Kaggle dataset. It contains 4000 80x80 RGB images in total, labeled with either “ship” or “no-ship” binary classification: 3K images were selected for the training process and the remaining for model validation.

A variety of training processes was performed with the TensorFlow Estimator API in Python to create a CNN model close to optimal with respect to prediction accuracy, number of operations and resources requirements. The CNN design space exploration (described in 4.3.1) resulted in the final model architecture shown in Fig. 4.5. The CNN model consists of 84K weights optimized using the Adam optimizer with the cross-entropy loss function; it achieves 97.6% accuracy after 50 epochs. It compares favorably to similar trained models due to the following results of the design study:

- *Number of Convolution layers:* The proposed CNN with only 2 convolution layers achieves accuracy 97.6% that is close to CNNs with more, e.g. a CNN with 3 convolution layers before any of the proposed optimizations achieved 98.5% accuracy.
- *Ship Orientation:* is limited and along with the proportion of the 80x80 image that the ship occupies, it leads to use 32 filters per convolution layer for achieving the best accuracy-computational cost trade-off.
- *Max Pooling layer:* size 4×4 achieved accuracy similar to that of size 2×2.
- *The kernel's size for each Convolution Layer:* the first achieved improved accuracy with a 5×5 kernel, while the choice for the Second Convolution Layer is a 4×4 kernel because its output has to be divisible by the following Max Pooling layer. As a result we don't use padding in the convolutions since this doesn't induce accuracy loss.
- *Fully Connected Layer's neurons:* 128 neurons of the fully connected layer is the minimum number to use in order to avoid prediction accuracy loss.

4.4.3 Bit-Accurate Model (BAM)

The design flow, following the realization of the TensorFlow model for the Vessel Detection, develops a bit-accurate model (BAM) that represents the exact operations and calculations in integer arithmetic that will take place in the FPGA. We note here that, the input image is represented in RGB with 8-bits per pixel and the

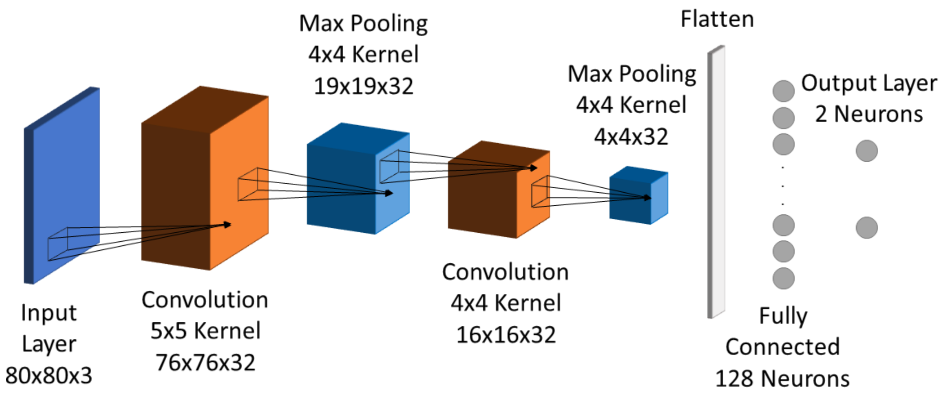


Figure 4.5: Model Architecture

BAM keeps (does not reduce) for each pixel the input bit-depth. Each parameter of the CNN model (weights, biases) is represented as a fixed-point number with 1-bit for the sign, 1-bit integer part and 6-bit fractional part (Q2.6). Throughout the BAM we preserve the 6-bit fractional part by truncating the result of each multiplication. In order to avoid accuracy losses due to overflow after consecutive additions the integer part is increased and the final results are represented in Q11.6.

4.4.4 The Example CNN FPGA Accelerator

The example accelerator's architecture consists of eight structural blocks on which we map the functionality of the Software model blocks: a) the Input Layer, b) the First Convolution Layer, c) the First Pooling Layer, d) the Second Input Layer, e) the Second Convolution Layer, f) the Second Pooling Layer, g) the Fully Connected Layer and h) the Output Layer. The overall architecture is illustrated in two figures, Fig. 4.6 and Fig. 4.7. The example accelerator exploits the parallelization of the CNN model in order to increase performance, minimize buffering and improve the throughput via pipelining of its operations. The following paragraphs present the significant details of the structure and operation of the accelerator's

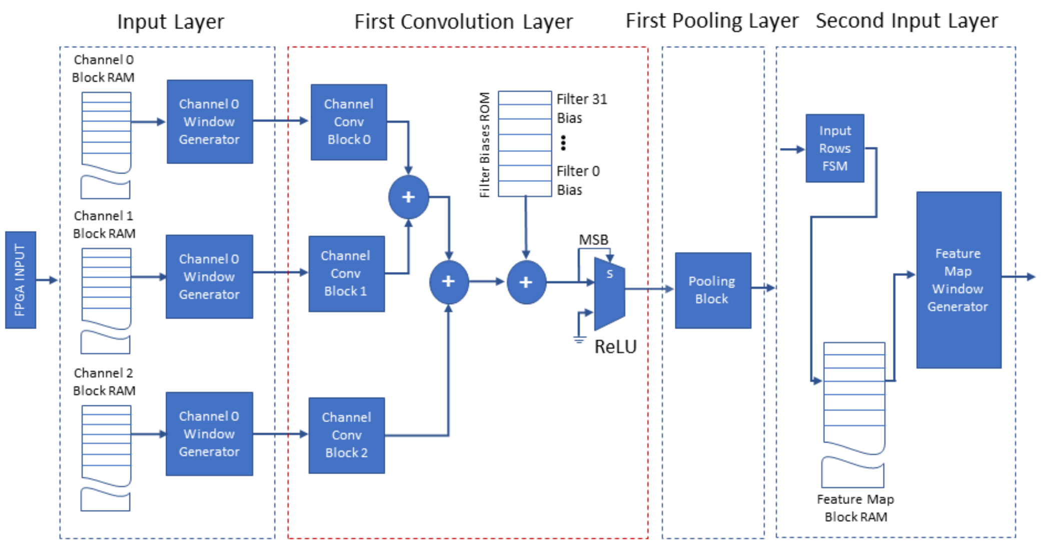


Figure 4.6: FPGA Architecture of the Input Layer, First Convolution & Pooling Layers and the Second Input Layer.

blocks and their advantages.

Figure 4.6 depicts the four leading blocks of the architecture (Input Layer, First Convolution Layer, First Pooling Layer and Second Input Layer). The architecture uses the blocks described in Section 4.3.3: the Input Layer with the *Window Generators* and the First Convolution Layer including three *Convolution Blocks*. Their output is forwarded to the ReLU and the First Pooling Layer consisting of one Pooling Block configured for 4×4 max pooling. The Second Input Layer includes a single input block. This design minimizes the memory required by the proposed accelerator in two ways. The First Convolution Layer calculates and adds in parallel the convolution of each input image channel with the corresponding kernel producing one complete output feature map, pipelining each value to the First Pooling layer without buffer use. The calculations are repeated for the remaining 31 feature maps, with the corresponding filter kernels. The Second Convolution Layer calculates the 32 filter convolutions on each received feature map in parallel and buffers the 32 results for accumulation. The required buffering at the output of this layer is reduced to 32 arrays of 16×16 13-bit values, because at this stage we have already executed the First Pooling Layer (4×4 max pooling). The latter shows the advantage of the proposed approach when it is used for shallow CNNs, because considering a systolic array accelerator for the same task, it would require a total of 2.03 Mbit to store the intermediate result of the output of the First Convolution Layer. In contrast, the proposed streamline architecture uses buffering of intermediate results only at the end of the Second Convolution Layer, following the downscale of the data by previous pooling operations: this is only 106.50 Kbit and hence, it achieves a 19.1x reduction in the required memory.

Another key element of the example accelerator's architecture is the Input Layer also depicted on Fig. 4.6; its design is based on the FPGA's features. The FPGA can support a variety of interfacing methods with the host such as PCIe, Ethernet and USB to receive the image. The Input Layer stores each channel (RGB) of the input image row by row in the corresponding *Channel Block RAM* (on-chip memory), so that we can read a whole row in a single clock cycle. These blocks along with the 3 *Window Generators* of the 3 *Input Blocks* constitute the Input Layer. The *Window Generators* are configured to accept one 80×80 image (1 image channel each) and generate all the windows of size 5×5 of that image channel; they operate as described in section 4.3.3.1. When the image is stored in each *Channel Block RAM*, the 3 *Window Generator* blocks operating in parallel, load the 3 distinct channel windows of size 5×5 in parallel to the 3 corresponding *Channel Convolution Blocks* of the First Convolution Layer, as shown in Fig. 4.6. Three distinct RGB windows of size 5×5 forwarded in parallel at each clock cycle to the *Convolution Blocks* in a fully pipelined operation.

Figure 4.7 depicts the second half of the example accelerator's architecture (the Second Convolution & Pooling Layers, the Fully Connected Layer and the final

Output Layer). The Second Convolution Layer includes 32 *Filter Convolution Blocks*, each block configured for 4×4 convolution kernels. The Second Convolution Layer receives one by one the feature maps of the previous Layers and performs the 32 filter convolutions of this layer in parallel with 32 *Filter Convolution Blocks*, each of which accumulates the result in a dedicated *Accumulator RAM* of size 16×16 words. Each *Filter Convolution Block* stores the kernel weights associated with each input feature map in an internal Block RAM. The results of this Layer are complete when every feature map of the previous Layer is received and processed. At the final accumulation step, each filter's bias is added and the *Accumulator RAM* contents of each *Filter Convolution Block* are forwarded, in a continuous stream (in filter order) to the Second Pooling Layer. The Second Pooling Layer is similar to the First Pooling Layer, also configured for 4×4 pooling, where memories act as a buffer in order to provide an uninterrupted flow of data to the succeeding Fully Connected Layer. Finally, the Fully Connected Layer uses 128 parallel *Vector Multipliers* one for each neuron. When all the multiply-accumulate steps are complete, the 128 parallel multipliers and a tree of adders calculate the inference result.

Although the CNN Vessel Accelerator improves the performance of CPU, GPU and edge processors, as will be shown in section 4.4.5.2, it is worth noting that the entire CNN Vessel Accelerator architecture can be configured to operate on two distinct input frames in a pipeline fashion. In that configuration while the first frame processing will occupy the fully connected layer, the two convolutional layers will be dedicated to the process of the second (following the first) frame.

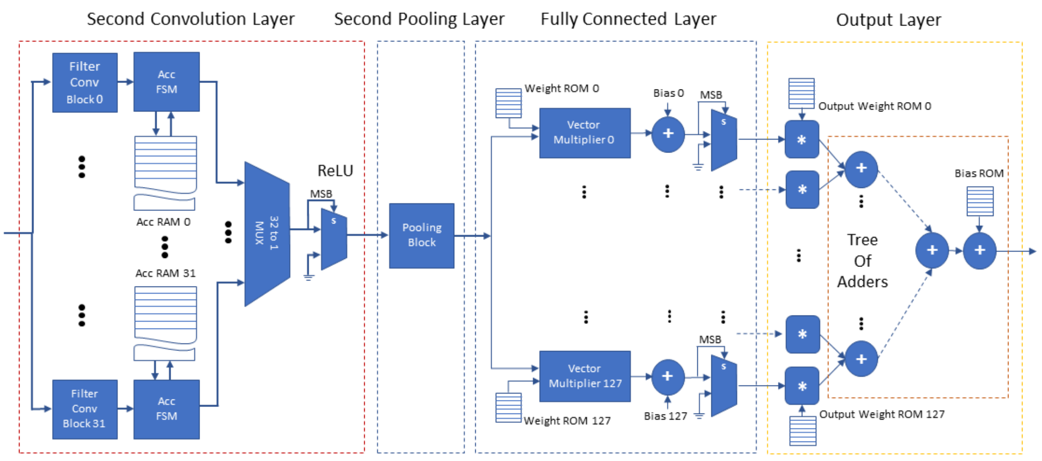


Figure 4.7: FPGA Architecture of the Second Convolution & Pooling Layers, Fully Connected Layer and Output Layer.

4.4.5 Vessel Detection CNN FPGA Accelerator Results & Comparison

This section presents first, the results of the example accelerator's implementation on the Xilinx VC707 board and second, the comparison with the corresponding performance of our code executed on: a) the low power Intel's Myriad2 processor, b) the edge-computing NVIDIA's Jetson Nano Jetson's ARM processor, and c) the Jetson Nano GPU.

4.4.5.1 FPGA Implementation Results

The development and validation of the example accelerator targeted the Xilinx Virtex 7 Development board (XC7VX485T) with the use of the Vivado development tool. The resource utilization of the FPGA on the Virtex 7 board is presented in the Table 4.1. More specifically, the example accelerator uses 9.37% of the FPGA's BRAMs and 30.11% of the available DSP blocks of the FPGA device. The example accelerator's power requirements are 5.001 W reported by the Vivado power estimator. Figure 4.8 presents the on-chip power utilization per resource type.

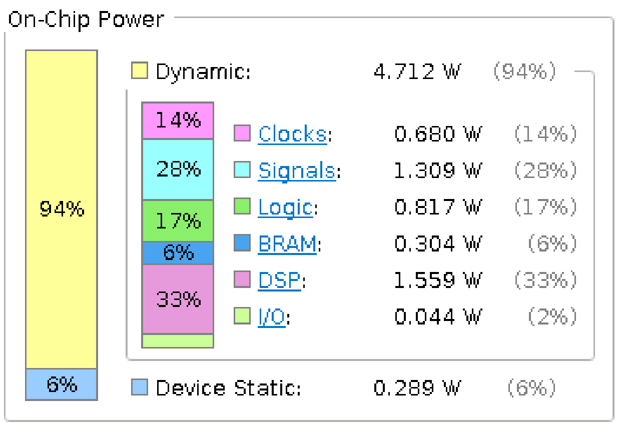
The FPGA implementation of the example accelerator has achieved a maximum operating frequency (f_{max}) of 270 MHz. The number of operations per second of the accelerator is 52.8 GOP/s and the processing time for a single input image (or a 80×80 sliding window) is 0.687 ms. In order to showcase an indicative baseline evaluation result, Table 4.2 presents the execution time comparison of the example accelerator to the CPU and GPU software implementations. The CPU and GPU software implementations are based on the TensorFlow implementation of the model executed with a single image as input and targeting the Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz and the NVIDIA GeForce RTX 3080 correspondingly. The CPU processes a single input image in 4.696ms while the GPU processes the same input image in 2.202ms. The example accelerator achieves a speed-up of 6.836 and 3.205 when compared to the CPU and GPU correspondingly.

Table 4.1: Resource Utilization

Resource	Utilization	Utilization %
LUT	50743	16.71
LUTRAM	4228	3.23
FF	70786	11.66
BRAM	96.5	9.37
DSP	843	30.11

Table 4.2: Performance Comparison to CPU & GPU

	Execution Time (ms)	FPGA Speed-up
FPGA	0.687	-
CPU	4.696	6.836
GPU	2.202	3.205



Dynamic:	4.712 W	(94%)
Clocks:	0.680 W	(14%)
Signals:	1.309 W	(28%)
Logic:	0.817 W	(17%)
BRAM:	0.304 W	(6%)
DSP:	1.559 W	(33%)
I/O:	0.044 W	(2%)
Device Static:	0.289 W	(6%)

Figure 4.8: Power Utilization

4.4.5.2 Comparison to Edge Devices and Low Power Processors

In order to evaluate the proposed approach we compared the performance of the Vessel Detection CNN FPGA Accelerator to the other edge devices, which have high performance at low power consumption according to their specs. Notable representatives are the NVIDIA's Jetson Nano and the Intel's Myriad2 processor. The Jetson Nano of 472 GFLOPS (FP16) at 10W includes an ARM processor and an 128-core Maxwell GPU targeting computer vision and deep learning applications. The Myriad2 processor is being utilized for on-board satellite computing applications in missions (Giuffrida et al., 2022) and research projects (España Navarro et al., 2021) due to the fact that it has undergone extensive radiation characterization (Furano et al., 2020) in order to be deemed suitable for space applications. It has two Leon and 12 SHAVE processors, it is optimized for machine learning applications, that can aggregate 1000 GFLOPS (FP16) with at most 1W consumption. Moreover, it includes a multicore on-chip memory subsystem (2MB), called Connection Matrix (CMX) memory and low-power DDR3 DRAM (512MB).

The comparison is based on a sequential C code for the vessel detection. This is executed on a single core of the Jetson's Nano ARM CPU and measured at 440 ms. From this point, we developed a custom CUDA accelerated application taking advantage of the 128 CUDA cores. The mapping of calculations to grids of thread blocks optimize the scheduling of warps on the 128 CUDA cores. The shared memory is used to store global data in a thread block and the intermediate results. The execution time of the CUDA application is 20.3 ms.

The development on the Myriad2 starts with the optimization of the sequential C code, using efficiently the CMX, DDR and cache memories; this single core application took 56.27 ms with less than 0.5 W. The parallel Myriad2 code uses the 12 SHAVES, by dividing the CMX memory between them, minimizing the required memory of each SHAVE by pipelining the operations of each processor, the parallel code takes 14.6 ms at 1 W.

The detailed results are presented in Table 4.3. The example FPGA accelerator achieves the highest performance, regarding execution time, median power consumption but the highest performance per watt among the other two devices. The Myriad2 is the most power efficient by consuming 1 W, while its performance is one order of magnitude lower than the FPGA accelerator. The Jetson Nano falls short in either metric with a consumption of 10 W and execution time in the same order to Myriad2, but it provides the most developer friendly platform, which is an advantage leading to short development time and effort. The example FPGA accelerator has the highest performance per watt, followed by the Myriad2 and at the last place is the Jetson Nano.

Finally, the benefit of introducing the approach for the image classification on a single FPGA device, whenever this is feasible, can be shown by the Vessel Detection Accelerator performance and compared to optimized FPGA CNN accelerators (Zhang et al., 2015) and also to low cost ones (Solovyev et al., 2018).

Table 4.3: Performance & Power Comparison to Edge Devices

	Execution Time (ms)	Speed-Up	Power (W)
Jetson Nano CPU	440	-	10
Jetson Nano GPU	20.3	21.7	10
Myriad2 1 SHAVE	56.27	7.8	0.5
Myriad2 12 SHAVE	14.59	30.1	1
FPGA Accelerator	0.687	640.5	5

4.4.5.3 Comparison to Other FPGA Accelerators

This subsection aims to provide more context to the proposed approach by showcasing where the proposed accelerator stands in the field of FPGA accelerators in the literature. A straightforward comparison though of the resulting accelerator to FPGA-based CNN accelerators is a challenging task (Zhang et al., 2015) because:
a) The same metrics between different FPGA accelerators may not be suitable for direct comparisons due to different FPGA platforms, benchmarking methodologies, etc.
b) While the majority of related works focus on accelerators for well-known CNN models, this work proposes a design approach that includes guidelines for designing CNN models from scratch, resulting in a custom model for the Vessel Detection application.
c) This work focuses on accelerator designs for shallow CNNs suitable for binary and low feature space classification tasks while most works in the literature study complex and larger CNN models and result in substantially different architectures. Regarding these architectural differences, the proposed streamline architectures in this work use contiguous modules for each layer of the CNN in a pipeline fashion. These architectures have particular benefits for our target applications (described

in section 4.3.4) and are further highlighted in the comparisons with other FPGA-based accelerators and the corresponding analysis below.

Table 4.4: Reporting the Features of Related Results

	Zhang et al., 2015	B. Liu et al., 2019	Rapuano et al., 2021	Example Accelerator
Precision	fl. point 32 bits	fl. point 32 bits	fixed-point 16 bits	fixed-point 17 bits
Frequency (MHz)	100	100	156	270
FPGA	Xilinx Virtex VC707	Xilinx Zynq 7100	Xilinx Zynq ZCU106	Xilinx Virtex VC707
CNN Size	1.33 GFLOP	N/A	N/A	18.122 MMAC
Performance (GOP/s)	61.62	17.11	N/A	52.80
Power (Watt)	18.61	4.083	3.4	5.001
Perf./Watt (GOP/s/Watt)	3.31	4.19	N/A	10.56
DSPs	2240	1926	1175	843
DSP Efficiency (GOP/s/DSP)	0.027	0.008	N/A	0.062

Taking into account the aforementioned considerations, Table 4.4 presents notable works on FPGA-based CNN accelerators, their most important features and the corresponding results metrics. Note that the example accelerator achieves the highest operating frequency of 270 MHz and this advantage is due to the custom VHDL design of the proposed approach especially when compared to the 100 MHz of the HLS generated designs of (Zhang et al., 2015) and (B. Liu et al., 2019). Moreover, the advantage of the streamline architecture as well as the utilization of only the on-chip memory is observed when compared to the 156 MHz of the Single Processing Unit VHDL design of (Rapuano et al., 2021).

Regarding the performance, the accelerator of (Zhang et al., 2015), targeting a much larger CNN model, exhibits a slightly larger performance of 61.62 GOP/s compared to the 52.80 GOP/s of the example accelerator. However, considering that both use the same FPGA device, the current work achieves this performance by utilizing only 843 DSPs, compared to the 2240 DSPs of (Zhang et al., 2015) and hence, it results in a significantly higher DSP efficiency of 0.062 GOP/s/DSP. The reason for this improvement in hardware efficiency is the proposed mapping methodology that produces a streamline architecture with multiple layers operating at the same time with extensive pipelining, in contrast to the systolic array architecture implementing a single layer at a time (Zhang et al., 2015).

Considering power consumption, the authors of (Rapuano et al., 2021) report 3.4 Watt while our example accelerator consumes 5.001 Watt. However, in that work there is no report of several features of the design that play a role in power consumption such as CNN size and performance. The accelerator in (B. Liu et al., 2019) reports power consumption of 4.083 Watt but achieves lower performance per Watt compared to the example accelerator. Finally, the power measurements in (Zhang et al., 2015) follow a different methodology by measuring the power consumption of the entire FPGA board rather than on-chip power consumption that we report and thus their measurement is not suitable for direct comparisons.

5 | Conclusions and Future Work

The current thesis presented a VOQs architecture, which is efficient with respect to latency and 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 lead 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. The control of the Active Destination Queues is efficient due to the minimum latency that it induces to the operation of the ToR switch. 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).

Moreover, in this thesis we presented a management tool for the Agent of the Nephele data center. The advantage of creating and using the proposed management tool is that the data center designers and engineers can create their own schedule as the tool's GUI users and then transfer that schedule to each data plane ToR switch. The user can control graphically in real time the transmission of Nephele frames originating at the ToR switch to the other Nephele ToRs in the data center network. The management tool can be of even further use if it will be extended to create the scheduling tables of a PoD switch in the Nephele network.

Furthermore, in this thesis is presented a design approach for FPGA accelerators for image classification CNNs with limited feature space targeting the data center, edge, mobile and on-board satellite computing applications. The objective of this work is to achieve real-time performance by placing all the inference task computations and memory within a single FPGA device. The benefits of the resulting architecture are the low power consumption, the higher operating frequency and the improved resources utilization. These advantages are shown by an Example FPGA accelerator for the Vessel Detection that compares favorably to the performance of notable edge and low power processors. The benefit of introducing the approach for the image classification on a single FPGA device, whenever this is feasible, can be shown by the Example Accelerator's performance and compared to optimized FPGA CNN accelerators and also to low cost ones.

Beyond any doubt, the results of the current thesis provide strong foundation for future work. First, the area of future work considers the use of machine learning techniques in data center switches that utilize VOQs, to provide Quality of Service (QoS) (L. Wang et al., 2018). The classification of the flow of data at switch level between mice flows, which are data flows small in size with high occurrence frequency and elephant flows, which appear sparsely but contain large amount of data, will enable the dynamic management of the VOQs of the data center switch in real-time. Second, the state of the art network traffic classification techniques are based on deep learning to extract the features of the data traffic (Abbasi, Shahraki, & Taherkordi, 2021). More specifically, supervised machine learning like the CNNs (Lim et al., 2019), (P. Wang et al., 2018), the recursive neural networks (RNNs) (Lopez-Martin, Carro, Sanchez-Esguevillas, & Lloret, 2017) and the recursive Long Short-Term Memory (LSTM) neural networks are used for the classification of traffic depending on its characteristics/features (Lee, Xie, Ngoduy, & Keyvan-Ekbatani, 2019). Thus, the final objective of the future work is to utilize the CNN Design Approach to enable on-chip traffic/flow classification for the efficient real-time management of the VOQs in data center network switches to support the QoS.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from https://www.tensorflow.org/ (Software available from tensorflow.org)

Abbasi, M., Shahraki, A., & Taherkordi, A. (2021). Deep learning for network traffic monitoring and analysis (ntma): A survey. *Computer Communications*, *170*, 19-41. Retrieved from https://www.sciencedirect.com/science/article/pii/S0140366421000426 doi: https://doi.org/10.1016/j.comcom.2021.01.021

Abdelouahab, K., Pelcat, M., Sérot, J., & Berry, F. (2018). Accelerating CNN inference on fpgas: A survey. *CoRR*, *abs/1806.01683*. Retrieved from http://arxiv.org/abs/1806.01683

Airbus ship detection challenge. (2019). https://www.kaggle.com/c/airbus-ship-detection.

Al-Fares, M., Loukissas, A., & Vahdat, A. (2008, August). A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, *38*(4), 63–74. Retrieved from http://doi.acm.org/10.1145/1402946.1402967 doi: 10.1145/1402946.1402967

Bakopoulos, P., Christodoulopoulos, K., Landi, G., Aziz, M., Zahavi, E., Gallico, D., … Avramopoulos, H. (2018). Nephele: An end-to-end scalable and dynamically reconfigurable optical architecture for application-aware sdn cloud data centers. *IEEE Communications Magazine*, *56*(2), 178-188. doi: 10.1109/MCOM.2018.1600804

Barry, B., Brick, C., Connor, F., Donohoe, D., Moloney, D., Richmond, R., … Toma, V. (2015, Mar). Always-on vision processing unit for mobile applications. *IEEE Micro*, *35*(2), 56-66. doi: 10.1109/MM.2015.10

Bazzaz, H. H., Tewari, M., Wang, G., Porter, G., Ng, T. S. E., Andersen, D. G., … Vahdat, A. (2011). Switching the optical divide: Fundamental challenges for hybrid electrical/optical datacenter networks. In *Proceedings of the 2nd acm symposium on cloud computing* (pp. 30:1–30:8). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2038916.2038946 doi: 10.1145/2038916.2038946

Benson, T., Akella, A., & Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th acm sigcomm conference on internet measurement* (p. 267–280). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1879141.1879175 doi: 10.1145/1879141.1879175

Chen JW, Z. J. (2007). Comparing text-based and graphic user interfaces for novice and expert users. In *Amia annual symposium proceedings.* doi: PMID:18693811;PMCID:PMC2655855

Choi, K., & Sobelman, G. E. (2022, aug). An efficient cnn accelerator for low-cost edge systems. *ACM Trans. Embed. Comput. Syst.*, *21*(4). Retrieved from https://doi.org/10.1145/3539224 doi: 10.1145/3539224

Christodoulopoulos, K., Kontodimas, K., Yiannopoulos, K., & Varvarigos, E. (2016, July). Bandwidth allocation in the nephele hybrid optical interconnect. In *2016 18th international conference on transparent optical networks (icton)* (p. 1-4). doi: 10.1109/ICTON.2016.7550704

Christodoulopoulos, K., Lugones, D., Katrinis, K., Ruffini, M., & O'Mahony, D. (2015, Mar). Performance evaluation of a hybrid optical/electrical interconnect. *J. Opt. Commun. Netw.*, *7*(3), 193–204. Retrieved from https://opg.optica.org/jocn/abstract.cfm?URI=jocn-7-3-193 doi: 10.1364/JOCN.7.000193

Cisco. (2014-2019). *Cisco global cloud index: Forecast and methodology.* Retrieved from https://virtualization.network/Resources/Whitepapers/0b75cf2e-0c53-4891-918e-b542a5d364c5_white-paper-c11-738085.pdf

Corazza, S., & Reale, S. (1992). Network management system graphical interface. In *Eighth international conference on software engineering for telecommunication systems and services, 1992.* (p. 135-138).

Dekker, R., Bouma, H., den Breejen, E., van den Broek, B., Hanckmann, P., Hogervorst, M., ... others (2013). Maritime situation awareness capabilities from satellite and terrestrial sensor systems. *Proc. Maritime Systems and Technologies MAST Europe*.

España Navarro, J., Samuelsson, A., Gingsjö, H., Barendt, J., Dunne, A., Buckley, L., ... Steenari, D. (2021, jun). High-performance compute board - a fault-tolerant module for on-boards vision processing. Zenodo. Retrieved from https://doi.org/10.5281/zenodo.5521624 doi: 10.5281/zenodo.5521624

Farrington, N., Porter, G., Radhakrishnan, S., Bazzaz, H. H., Subramanya, V., Fainman, Y., ... Vahdat, A. (2010). Helios: A hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the acm sigcomm 2010 conference* (pp. 339–350). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1851182.1851223 doi: 10.1145/1851182.1851223

Farrington, N., Rubow, E., & Vahdat, A. (2009). Data center switch architecture in the age of merchant silicon. In *2009 17th ieee symposium on high performance interconnects* (p. 93-102). doi: 10.1109/HOTI.2009.11

Furano, G., Meoni, G., Dunne, A., Moloney, D., Ferlet-Cavrois, V., Tavoularis, A., ... Fanucci, L. (2020). Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities. *IEEE Aerospace and Electronic Systems Magazine*, *35*(12), 44-56. doi: 10.1109/MAES.2020.3008468

Girshick, R. B., Donahue, J., Darrell, T., & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, *abs/1311.2524*. Retrieved from http://arxiv.org/abs/1311.2524

Giuffrida, G., Fanucci, L., Meoni, G., Batič, M., Buckley, L., Dunne, A., ... Aschbacher, J. (2022). The -sat-1 mission: The first on-board deep neural network demonstrator for satellite earth observation. *IEEE Transactions on Geoscience and Remote Sensing*, *60*, 1-14. doi: 10.1109/TGRS.2021.3125567

Gorokhovatskyi, O., & Peredrii, O. (2018, 08). Shallow convolutional neural networks for pattern recognition problems.. doi: 10.1109/DSMP.2018.8478540

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., ... Sengupta, S. (2009). Vl2: A scalable and flexible data center network. In *Proceedings of the acm sigcomm 2009 conference on data communication* (pp. 51–62). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1592568.1592576 doi: 10.1145/1592568.1592576

Han, S., Egi, N., Panda, A., Ratnasamy, S., Shi, G., & Shenker, S. (2013). Network support for resource disaggregation in next-generation datacenters. New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2535771.2535778 doi: 10.1145/2535771.2535778

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T.,

… Adam, H. (2017). *Mobilenets: Efficient convolutional neural networks for mobile vision applications.* arXiv. Retrieved from https://arxiv.org/abs/1704.04861 doi: 10.48550/ARXIV.1704.04861

Jacobsen, M., Richmond, D., Hogains, M., & Kastner, R. (2015, sep). Riffa 2.1: A reusable integration framework for fpga accelerators. , *8*(4). Retrieved from https://doi.org/10.1145/2815631 doi: 10.1145/2815631

Ji-yang, Y., Dan, H., Lu-yuan, W., Jian, G., & Yan-hua, W. (2016, Nov). A real-time on-board ship targets detection method for optical remote sensing satellite. In *2016 ieee 13th international conference on signal processing (icsp)* (p. 204-208). doi: 10.1109/ICSP.2016.7877824

Johan Vos, S. C. D. I. a. J. W., Weiqi Gao. (2014). *Pro javafx 8: A definitive guide to building desktop, mobile, and embedded java clients.* Apress, 1st edition.

Kachris, C., & Tomkos, I. (2012). A survey on optical interconnects for data centers. *IEEE Communications Surveys Tutorials*, *14*(4), 1021-1036. doi: 10.1109/SURV.2011.122111.00069

Kanjir, U., Greidanus, H., & Oštir, K. (2018). Vessel detection and classification from spaceborne optical images: A literature survey. *Remote sensing of environment*, *207*, 1–26.

Kim, J. H., Grady, B., Lian, R., Brothers, J., & Anderson, J. H. (2017, Sep.). Fpga-based cnn inference accelerator synthesized from multi-threaded c software. In *2017 30th ieee international system-on-chip conference (socc)* (p. 268-273). doi: 10.1109/SOCC.2017.8226056

Kyriakos, A., Kitsakis, V., Louropoulos, A., Papatheofanous, E.-A., & Patronas, G. (2019, 07). High performance accelerator for cnn applications. In (p. 135-140). doi: 10.1109/PATMOS.2019.8862166

Kyriakos, A., Patronas, I., Tzimas, G., Kitsakis, V., & Reisis, D. (2017). Realizing virtual output queues in high throughput data center nodes. In *2017 panhellenic conference on electronics and telecommunications (pacet)* (p. 1-4). doi: 10.1109/PACET.2017.8259971

Kyriakos, A., Tsavalos, T., & Reisis, D. (2017). Gui for the communication agent of the "nephele" data center. In *2017 south eastern european design automation, computer engineering, computer networks and social media conference (seeda-cecnsm)* (p. 1-5). doi: 10.23919/SEEDA-CECNSM.2017.8088237

Lamoureux, J., & Luk, W. (2008, June). An overview of low-power techniques for field-programmable gate arrays. In *2008 nasa/esa conference on adaptive hardware and systems* (p. 338-345). doi: 10.1109/AHS.2008.71

Landi, G., Patronas, I., Kontodimas, K., Aziz, M., Christodoulopoulos, K., Kyriakos, A., ... Avramopoulos, H. (2017). Sdn control framework with dynamic resource assignment for slotted optical datacenter networks. In *Optical fiber communication conference* (p. Tu3L.1). Optica Publishing Group. Retrieved from http://opg.optica.org/abstract.cfm?URI=OFC-2017-Tu3L.1 doi: 10.1364/OFC.2017.Tu3L.1

Lee, S., Xie, K., Ngoduy, D., & Keyvan-Ekbatani, M. (2019). An advanced deep learning approach to real-time estimation of lane-based queue lengths at a signalized junction. *Transportation Research Part C: Emerging Technologies*, *109*, 117-136. Retrieved from https://www.sciencedirect.com/science/article/pii/S0968090X1830812X doi: https://doi.org/10.1016/j.trc.2019.10.011

Lei, F., Liu, X., Dai, Q., & Ling, B. (2020, 01). Shallow convolutional neural network for image classification. *SN Applied Sciences*, *2*. doi: 10.1007/s42452-019-1903-4

Li, H., Lin, Z., Shen, X., & Brandt, J. (2015, 06). A convolutional neural network cascade for face detection. In (p. 5325-5334). doi: 10.1109/CVPR.2015.7299170

Lim, H.-K., Kim, J.-B., Heo, J.-S., Kim, K., Hong, Y.-G., & Han, Y.-H. (2019). Packet-based network traffic classification using deep learning. In *2019 international conference on artificial intelligence in information and communication (icaiic)* (p. 046-051). doi: 10.1109/ICAIIC.2019.8669045

Lin, Y.-B., & Geigel, J. (1997). A graphical user interface design for network simulation. *Journal of Systems and Software*, *36*(2), 181-190. Retrieved from https://www.sciencedirect.com/science/article/pii/0164121295002014 doi: https://doi.org/10.1016/0164-1212(95)00201-4

Liu, B., Zou, D., Feng, L., Feng, S., Fu, P., & Li, J. (2019). An fpga-based cnn accelerator integrating depthwise separable convolution. *Electronics*, *8*(3). Retrieved from https://www.mdpi.com/2079-9292/8/3/281 doi: 10.3390/electronics8030281

Liu, H., Lu, F., Kapoor, R., Forencich, A., Voelker, G. M., Papen, G., ... Porter, G. (2013). Reactor: A reconfigurable packet and circuit tor switch. In *2013 ieee photonics society summer topical meeting series* (p. 235-236). doi: 10.1109/PHOSST.2013.6614526

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., & Berg, A. C. (2015). SSD: single shot multibox detector. *CoRR*, *abs/1512.02325*.

Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., & Lloret, J. (2017). Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, *5*, 18042-18050. doi: 10.1109/ACCESS.2017.2747560

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., . . . Turner, J. (2008, mar). Openflow: Enabling innovation in campus networks. , *38*(2), 69–74. Retrieved from https://doi.org/10.1145/1355734.1355746 doi: 10.1145/1355734.1355746

Mellanox-Technologies. (May 2013). *Sx1024: The ideal multi-purpose top-of-rack switch.* White Paper. Retrieved from https://network.nvidia.com/related-docs/whitepapers/SX1024-The-Ideal-Multipurpose-Top-of-Rack-Switch.pdf

Moor, . S. I. (2013). *Intel's disaggregated server rackv.* Retrieved from https://moorinsightsstrategy.com/wp-content/uploads/2013/08/Intels-Disagggregated-Server-Rack-by-Moor-Insights-Strategy.pdf

Mordvintsev, A., Olah, C., & Tyka, M. (2015). *Inceptionism: Going deeper into neural networks.* https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

Nvidia jetson nano. (2020). https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

Osborne, W. G., Luk, W., Coutinho, J. G. F., & Mencer, O. (2008, July). Reconfigurable design with clock gating. In *2008 international conference on embedded computer systems: Architectures, modeling, and simulation* (p. 187-194). doi: 10.1109/ICSAMOS.2008.4664863

Patronas, I., Kyriakos, A., & Reisis, D. (2016). Switching functions of a data center top-of-rack (tor). In *2016 ieee international conference on electronics, circuits and systems (icecs)* (p. 364-367). doi: 10.1109/ICECS.2016.7841208

Peemen, M., Setio, A. A. A., Mesman, B., & Corporaal, H. (2013, Oct). Memory-centric accelerator design for convolutional neural networks. In *2013 ieee 31st international conference on computer design (iccd)* (p. 13-19). doi:

10.1109/ICCD.2013.6657019

Pelcat, M., Bourrasset, C., Maggiani, L., & Berry, F. (2016, July). Design productivity of a high level synthesis compiler versus hdl. In *2016 international conference on embedded computer systems: Architectures, modeling and simulation (samos)* (p. 140-147). doi: 10.1109/SAMOS.2016.7818341

Pitonak, R., Mucha, J., Dobis, L., Javorka, M., & Marusin, M. (2022). Cloudsatnet-1: Fpga-based hardware-accelerated quantized cnn for satellite on-board cloud coverage classification. *Remote Sensing*, *14*(13). Retrieved from https://www.mdpi.com/2072-4292/14/13/3180 doi: 10.3390/rs14133180

Planet: Ships-in-satellite-imagery. (2019). https://www.kaggle.com/rhammell/ships-in-satellite-imagery.

Rapuano, E., Meoni, G., Pacini, T., Dinelli, G., Furano, G., Giuffrida, G., & Fanucci, L. (2021). An fpga-based hardware accelerator for cnns inference on board satellites: Benchmarking with myriad 2-based solution for the cloudscout case study. *Remote Sensing*, *13*(8). Retrieved from https://www.mdpi.com/2072-4292/13/8/1518 doi: 10.3390/rs13081518

Redmon, J., Divvala, S. K., Girshick, R. B., & Farhadi, A. (2015). You only look once: Unified, real-time object detection. *CoRR*, *abs/1506.02640*. Retrieved from http://arxiv.org/abs/1506.02640

Ren, S., He, K., Girshick, R. B., & Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, *abs/1506.01497*.

Roy, A., Zeng, H., Bagga, J., Porter, G., & Snoeren, A. C. (2015, aug). Inside the social network's (datacenter) network. , *45*(4), 123–137. Retrieved from https://doi.org/10.1145/2829988.2787472 doi: 10.1145/

2829988.2787472

Sankaradas, M., Jakkula, V., Cadambi, S., Chakradhar, S., Durdanovic, I., Cosatto, E., & Graf, H. P. (2009, July). A massively parallel coprocessor for convolutional neural networks. In *2009 20th ieee international conference on application-specific systems, architectures and processors* (p. 53-60). doi: 10.1109/ASAP.2009.25

Saridis, G. M., Peng, S., Yan, Y., Aguado, A., Guo, B., Arslan, M., ... Simeonidou, D. (2016). Lightness: A function-virtualizable software defined data center network with all-optical circuit/packet switching. *Journal of Lightwave Technology*, *34*(7), 1618-1627. doi: 10.1109/JLT.2015.2509476

Sermanet, P., & LeCun, Y. (2011, July). Traffic sign recognition with multi-scale convolutional networks. In *The 2011 international joint conference on neural networks* (p. 2809-2813). doi: 10.1109/IJCNN.2011.6033589

Singla, A., Singh, A., Ramachandran, K., Xu, L., & Zhang, Y. (2010). Proteus: A topology malleable data center network. In *Proceedings of the 9th acm sigcomm workshop on hot topics in networks.* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1868447.1868455 doi: 10.1145/1868447.1868455

Solovyev, R. A., Kalinin, A. A., Kustov, A. G., Telpukhov, D. V., & Ruhlov, V. S. (2018). Fpga implementation of convolutional neural networks with fixed-point calculations. *CoRR*, *abs/1808.09945*.

Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, *105*(12), 2295-2329. doi: 10.1109/JPROC.2017.2761740

Tokas, K., Spatharakis, C., Kanakis, I., Iliadis, N., Bakopoulos, P., Avramopoulos, H., ... Reisis, D. (2016, June). A scalable optically-switched datacenter network with multicasting. In *2016 european conference on net-*

works and communications (eucnc) (p. 265-270). doi: 10.1109/EuCNC.2016.7561045

Turon, M. (2005). Mote-view: a sensor network monitoring and management tool. In *The second ieee workshop on embedded networked sensors, 2005. emnets-ii.* (p. 11-17). doi: 10.1109/EMNETS.2005.1469094

Vahdat, A., Al-Fares, M., Farrington, N., Mysore, R. N., Porter, G., & Radhakrishnan, S. (2010). Scale-out networking in the data center. *IEEE Micro*, *30*(4), 29-41. doi: 10.1109/MM.2010.72

Vattikonda, B. C., Porter, G., Vahdat, A., & Snoeren, A. C. (2012). Practical tdma for datacenter ethernet. In *Proceedings of the 7th acm european conference on computer systems* (p. 225–238). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2168836.2168859 doi: 10.1145/2168836.2168859

Wang, G., Andersen, D. G., Kaminsky, M., Papagiannaki, K., Ng, T. E., Kozuch, M., & Ryan, M. (2010). C-through: Part-time optics in data centers. In *Proceedings of the acm sigcomm 2010 conference* (p. 327–338). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1851182.1851222 doi: 10.1145/1851182.1851222

Wang, L., Wang, X., Tornatore, M., Kim, K. J., Kim, S. M., Kim, D.-U., ... Mukherjee, B. (2018). Scheduling with machine-learning-based flow detection for packet-switched optical data center networks. *Journal of Optical Communications and Networking*, *10*(4), 365-375. doi: 10.1364/JOCN.10.000365

Wang, P., Ye, F., Chen, X., & Qian, Y. (2018). Datanet: Deep learning based encrypted network traffic classification in sdn home gateway. *IEEE Access*, *6*, 55380-55391. doi: 10.1109/ACCESS.2018.2872430

Wilton, S. J. E., Ang, S.-S., & Luk, W. (2004). The impact of pipelining on energy per operation in field-programmable gate arrays. In J. Becker, M. Platzner, & S. Vernalde (Eds.), *Field programmable logic and application* (pp. 719–728). Berlin, Heidelberg: Springer Berlin Heidelberg.

Yébenes, P., Maglione-Mathey, G., Escudero-Sahuquillo, J., García, P. J., & Quiles, F. J. (2016). Modeling a switch architecture with virtual output queues and virtual channels in hpc-systems simulators. In *2016 international conference on high performance computing simulation (hpcs)* (p. 380-386). doi: 10.1109/HPCSim.2016.7568360

Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., & Cong, J. (2015). Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 acm/sigda international symposium on field-programmable gate arrays* (pp. 161–170). New York, NY, USA: ACM. doi: 10.1145/2684746.2689060

Zhao, H., Zhang, W., Sun, H., & Xue, B. (2019, 02). Embedded deep learning for ship detection and recognition. *Future Internet*, *11*, 53. doi: 10.3390/fi11020053

Zhao, Y., Gao, X., Guo, X., Liu, J., Wang, E., Mullins, R., ... Xu, C.-Z. (2019). Automatic generation of multi-precision multi-arithmetic cnn accelerators for fpgas. In *2019 international conference on field-programmable technology (icfpt)* (p. 45-53). doi: 10.1109/ICFPT47387.2019.00014

Zilberman, N., Audzevich, Y., Covington, G. A., & Moore, A. W. (2014). Netfpga sume: Toward 100 gbps as research commodity. *IEEE Micro*, *34*(5), 32-41. doi: 10.1109/MM.2014.61

A | Publications

1. Angelos Kyriakos, Elissaios-Alexios Papatheofanous, Charalampos Bezaitis, Dionysios Reisis, ”Resources and Power Efficient FPGA Accelerators for Real-Time Image Classification”, J. Imaging 2022, 8, 114 April 2022.

2. Vasileios Leon, Charalampos Bezaitis, George Lentaris, Dimitrios Soudris, Dionysios Reisis, Elissaios-Alexios Papatheofanous, Angelos Kyriakos, Aubrey Dunne, Arne Samuelsson, David Steenari, ”FPGA VPU Co-Processing in Space Applications: Development and Testing with DSP/AI Benchmarks”, 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), 2021, pp. 1-5 November 2021.

3. Joaquín España Navarro, Arne Samuelsson, Henrik Gingsjö, Julius Barendt, Aubrey Dunne, Léonie Buckley, Dionysios Reisis, Angelos Kyriakos, Elissaios Alexios Papatheofanous, Charalampos Bezaitis, Peter Matthijs, Juan Pablo Ramos, David Steenari, ”High-Performance Compute Board - A Fault-Tolerant Module For On-Board Vision Processing”, European Workshop on On-Board Data Processing (OBDP), June, 2021.

4. Tokas, K.; Patronas, G.; Spatharakis, C.; Bakopoulos, P.; Kyriakos, A.; Landi, G.; Zahavi, E.; Christodoulopoulos, K.; Aziz, M.; Pitwon, R.; Gallico, D.; Reisis, D.; Varvarigos, E.; Avramopoulos, H., ”End-to-End Real-Time Demonstration of the Slotted, SDN-Controlled NEPHELE Optical Datacenter Network”, Photonics 2020, 7, 44, June 2020.

5. Angelos Kyriakos, Elissaios-Alexios Papatheofanous, Bezaitis Charalampos, Evangelos Petrongonas, Dimitrios Soudris and Dionysios Reisis, ”Design and Performance Comparison of CNN Accelerators based on the Intel Movidius Myriad2 SoC and FPGA embedded prototype”, 3rd International

Conference on Control, Artificial Intelligence, Robotics and Optimization (ICCAIRO), Athens, Greece, December 2019.

6. Angelos Kyriakos, Vasileios Kitsakis, Alexandros Louropoulos, Elissaios-Alexios Papatheofanous, Ioannis Patronas, Dionysios Reisis, ”High Performance Accelerator for CNN Applications”, 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Rhodes, Greece, July 2019.

7. A. Kyriakos, T. Tsavalos, D. Reisis, ”Management Tool for the “Nephele” Data Center Communication Agent”, Advances in Science, Technology and Engineering Systems Journal, vol. 3, no. 6, pp. 144-150, November 2018.

8. A. Kyriakos, I. Patronas, G. Tzimas, V. Kitsakis, D. Reisis, ”Virtual Output Queues Architecture for High Throughput Data Center Nodes”, Advances in Science, Technology and Engineering Systems Journal, vol. 3, no. 5, pp. 97-104, September 2018.

9. K. Tokas, C. Spatharakis, I. Patronas, P. Bakopoulos, G. Landi, K. Christodoulopoulos, M. Capitani, A. Kyriakos, M. Aziz, R. Pitwon, D. Gallico, D. Reisis, E. Varvarigos, E. Zahavi, H. Avramopoulos, “Real time demonstration of an end-to-end optical datacenter network with dynamic bandwidth allocation”, 44th European Conference on Optical Communication-ECOC, Rome, Italy, September 2018.

10. P. Bakopoulos, K. Tokas, C. Spatharakis, I. Patronas, G. Landi, K. Christodoulopoulos, M. Capitani, A. Kyriakos, M. Aziz, D. Reisis, E. Varvarigos, E. Zahavi, H. Avramopoulos, “Optical datacenter network employing slotted (TDMA) operation for dynamic resource allocation”, SPIE OPTO, 2018, San Francisco, California, United States, February 2018.

11. G. Landi, I. Patronas, K. Kontodimas, M. Aziz, K. Christodoulopoulos, A. Kyriakos, M. Capitani, A. Hamedani, D. Reisis, E. Varvarigos, P. Bakopoulos, H. Avramopoulos, ”SDN Control Framework with Dynamic Resource Assignment for Slotted Optical Datacenter Networks”, 2017 Optical Fiber Communication Conference(OFC), Los Angeles, California, USA, March 2017.

12. Ioannis Patronas, Angelos Kyriakos, Dionysios Reisis, “Switching Functions of a Data Center Top-of-Rack(ToR)”, 23rd IEEE International Conference on Electronics Circuits and Systems on, Monte Carlo, Monaco, December 2016.