Titre: Title:	High-Level Modelling of Optical Integrated Networks-Based Systems with the Provision of a Low Latency Controller
Auteur: Author:	Felipe Gohring de Magalhaes
Date:	2017
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gohring de Magalhaes, F. (2017). High-Level Modelling of Optical Integrated Networks-Based Systems with the Provision of a Low Latency Controller [Ph.D. thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2661/

UNIVERSITÉ DE MONTRÉAL

HIGH-LEVEL MODELLING OF OPTICAL INTEGRATED NETWORKS-BASED SYSTEMS WITH THE PROVISION OF A LOW LATENCY CONTROLLER

FELIPE GOHRING DE MAGALHÃES
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
MAI 2017

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

HIGH-LEVEL MODELLING OF OPTICAL INTEGRATED NETWORKS-BASED SYSTEMS WITH THE PROVISION OF A LOW LATENCY CONTROLLER

présentée par: GOHRING DE MAGALHÃES Felipe
en vue de l'obtention du diplôme de: Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de:

Mme BOUCHENEB Hanifa, Doctorat, présidente
Mme NICOLESCU Gabriela, Doctorat, membre et directrice de recherche
M. HESSEL Fabiano, Ph.D., membre et codirecteur de recherche
Mme LIBOIRON-LADOUCEUR Odile, Ph.D., membre et codirectrice de recherche
M. BELTRAME Giovanni, Ph.D., membre
M. AZEVEDO Rodolfo, Ph.D., membre externe

DEDICATORY

To my family, wife and friends.

ACKNOWLEDGMENTS

First, I would like to thank my wife Val, for the support and friendship. Thanks for staying awake in working late hours and drinking chimarrão with me, so we could stay awake. Thank you for the period abroad in the cold Montreal, your presence made everything easier. Thank you for everything.

A special thanks to my advisors, Professor Fabiano, Professor Gabriela and Professor Odile, for the guidance and patience (specially the patience!). Without your instructions I would never have gone this far.

Thanks to all the friends made along the way, in Brazil and Canada. The discussions brought different views and enriched this project.

Finally, thanks to my family, which was always there, for the good and hard times.

RÉSUMÉ

La tendance du marché dans la conception des architectures multiprocesseurs de la prochaine génération consiste à intégrer de plus en plus de cœurs dans la même puce. Cette concentration des cœurs dans la même puce exige l'amélioration des politiques d'intercommunication. L'une des solutions proposées dans ce contexte consiste à utiliser les réseaux sur puce vu qu'ils présentent une amélioration considérable en termes de la bande passante, l'évolutivité et de l'extensibilité. Néanmoins, vu la croissance exponentielle en nombres de cœurs sur puce, les interconnexions électriques dans les réseaux sur puce peuvent devenir un goulet d'étranglement dans la performance du système. Par conséquent, des nouvelles techniques et technologies doivent être adoptées pour remédier à ces problèmes.

Les réseaux optiques intégrés (OIN venant de l'anglais Optical Integrated Networks) sont actuellement considérés comme l'un des paradigmes les plus prometteurs dans ce contexte. Les OINs offrent une plus grande bande passante, une plus faible consommation d'énergie et moins de latence lors de l'échange des données. Plusieurs travaux récents démontrent la faisabilité des OIN avec les technologies de fabrication disponibles et compatibles avec CMOS. Cependant, les concepteurs des OINs font face à plusieurs défis :

- Actuellement, les contrôleurs représentent le principal goulot d'étranglement de la communication et présentent l'un des facteurs minimisant l'efficacité des OINs. Alors, la proposition des nouvelles solutions de contrôle à faible latence est de plus en plus primordiale pour en tirer profit.

- Le manque d'outils de modélisation et de validation des OINs. La plupart des travaux se concentrent sur la conception des dispositifs et l'amélioration des performances des composants de base, tout en laissant le système sans assistance.

Dans ce contexte, afin de faciliter le déploiement de systèmes basés sur les OINs, cette thèse se focalise sur les trois contributions majeures suivantes: (1) le développement d'un ensemble de méthodes précises de modélisation qui va permettre par la suite de réaliser une plateforme de simulation au niveau du système ; (2) la définition et le développement d'une approche de contrôle efficace pour les systèmes basés sur les OINs; (3) l'évaluation de l'approche de contrôle proposée.

ABSTRACT

Design trends for next-generation Multi-Processor Systems point to the integration of a large number of processing cores, requiring high-performance interconnects. One solution being applied to improve the communication infrastructure in such systems is the usage of Networks-on-Chip as they present considerable improvement in the bandwidth and scaleability. Still as the number of integrated cores continues to increase and the system scales, the metallic interconnects in Networks-on-Chip can become a performance bottleneck. As a result, a new strategy must be adopted in order for those issues to be remedied.

Optical Integrated Networks (OINs) are currently considered to be one of the most promising paradigm in this design context: they present higher bandwidth, lower power consumption and lower latency to broadcast information. Also, the latest work demonstrates the feasibility of OINs with their fabrication technologies being available and CMOS compatible.

However, OINs' designers face several challenges:

- Currently, controllers represent the main communication bottleneck and are one of the factors limiting the usage of OINs. Therefore, new controlling solutions with low latency are required.
- Designers lack tools to model and validate OINs. Most research nowadays is focused on designing devices and improving basic components performance, leaving system unattended.

In this context, in order to ease the deployment of OIN-based systems, this PhD project focuses on three main contributions: (1) the development of accurate system-level modelling study to realize a system-level simulation platform; (2) the definition and development of an efficient control approach for OIN-based systems, and; (3) the system-level evaluation of the proposed control approach using the defined modelling.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

ADL	Architecture Description Language
CMOS	Complementary Metal-Oxide-Semiconductor
CMP	Chip Multiprocessor
CRB	Conflict Resolution Block
DSB	Dynamic Setup Block
eNOC	Electrical Network-on-chip
FIFO	First-in-first-out
FLB	FPGA logical blocks
FSM	Finite-state-machine
Gbps	Gigabits per second
GUI	Graphical User Interface
HDL	Hardware Description Languages
HYCO	Hybrid Controller
I/O	Input-and-Output
IoT	Internet of Things
IP	Intelectual Property
LUCC	LUT-based Centralized Controller
LUT	Look-Up Table
MR	Micro-ring Resonator
MZI	Mach-Zehnder Interferometer
NI	Network-Interface
OINs	Optical Integrated Networks
ONOC	Optical Network-on-Chip
P2P	Point-to-Point
PICS	Photonic Integrated Circuits
RR	Round-Robin
RTL	Register transfer level
SERDES	Serializer/Deserializer
SF-Sim	Straight-forward Simulator
SF	StraightForward
SNR	Signal-to-noise Ratio
SOI	Silicon-on-Insulator
SPF	Shortest Path First

TDM	Time Division Multiplexing
UML	Unified Modelling Language
VHDL	VHSIC Hardware Description Language
WBA	Weight Based Arbiter
WDM	Wavelength-division multiplexing
XUP	Xilinx University Program

LIST OF ANNEXES

CHAPTER 1 INTRODUCTION

This chapter includes an overview of the motivations, objectives, and the proposed contributions in the field of optical integrated networks modelling and controlling. The context of optical networks and the research challenges that inspired this thesis are introduced and deployed solutions to address the challenges are shortly commented. Lastly, the document organization is presented.

1.1 Context and Motivations

Nowadays systems present a rising number of features, leading to a significant growth in the applications' design complexity. Mostly, these systems have their implementation based on multiple processing elements integrated on the same die and running at a lower clock frequency due to energy consumption constraints [1].

Since the introduction of Chip Multiprocessor (CMP), one of the design main concerns lies in how the communication among internal components is performed. Bus-based systems present a well-known solution with a reasonable bandwidth and great ease of implementation. As the number of components rises, the complexity of bus design increases and their application becomes challenging [2]. Moreover, the communication can become a bottleneck in the system performance of traditional bus-based systems, which can compromise its operation [3]. Aiming to solve this issue, Electrical Network-on-Chip (eNoC) is one of the most popular solutions that have been proposed.

Systems based on eNoCs tend to provide better communication performance [4] when compared to traditional bus-based systems. In this case, the communication management is performed by routers that forward packets through the network. Each network node consists of a router and a connected component which could be, for example, a processor or a memory. Besides the gain in the communication capability, eNoCs usually present improved energy reliability and efficiency as well as high re-usability [5]. However, as the number of possible integrated cores on a single chip continues to increase, metallic interconnects in eNoCs will become a bottleneck due to their high power consumption, limited bandwidth, long latency and poor scaleability, leading ITRS [6] to point out the need for a new technology to overcome such restrictions. Another drawback of employing eNoCs is related to their architectural organization. As eNoCs rely on point-to-point communication links, their usage might be limited as the system scales, resulting in higher contention for long-distance

communications among cores and consequently performance degradation through imposing a higher power consumption. In this design context, Optical Integrated Networks (OINs) and the 3D die stacking[1] are currently considered to be the two most promising new paradigms [7–11].

OINs are already a reality for long-distance communications [12] and their usage for short-distance communications, such as inter-chip communications, has already been proven to be applicable [13, 14]. Recently, published work presented photonic architectures with low power consumption, low insertion loss and low power penalty [15, 16]. These work bring forward OINs as attractive candidates for high demanding communicating architectures.

Optical Network-on-chip (ONoC) emerges as a possible solution to overcome the aforementioned eNoCs issues as it presents a higher bandwidth when compared to common electrical eNoCs implementations with low power consumption [17]. Also, another advantage of employing ONoCs lies on their own physical implementation. With eNoCs, communications are usually local (Point-to-Point - P2P), reducing their design complexity but not allowing good scaleability for shared resources. On the other hand, ONoCs present a potential support for broadcasting messages, making them suitable for the new multiprocessors paradigms [18].

The performance and efficiency of such architectures are constrained by their controllers. The control part has an important impact on the OIN overall performance and a better solution is yet to be found [19]. Previous works demonstrated architectures with either long setup time or that have become too complex, thus challenging practical deployment [20, 21]. Consequently, while low latency controllers have been demonstrated [22, 23], further improvement in their response time is still required to realize practical deployment of OINs. Further, most proposed controllers are deeply attached to the network for which they are designed. Although this fact may lead to an increase in the performance of the controller, this goes against the trend for the next generation of communication systems, in which it is believed that each network layer (application, control and physical) will be independent from each other [24, 25].

Designing a system based on OINs is a very complex task that need to be automated using efficient tools. While the device level presents indeed a dense support in terms of automation tools for the development of optical devices, this is not true for the system level. In fact, when compared with high-level electrical design tools, system level support is nearly nonexistent [6]. Acknowledging the importance of this aspect, it is important to highlight that it is not possible to design an entire system with nowadays commercial tools, as it would be a complex

[1] 3D die stacking is an integrated circuit manufacturing technique in which two or more silicon wafers are placed one over the other and interconnected vertically using through silicon via (TSVs) in such a way that they behave as a single device. This technique is key in the OINs dissemination as it allows the integration of a variety of technologies prototyped onto the same chip.

task and would need more advanced tools. So, a research gap is open for the development of methods and tools to aid in the design of OIN-based systems.

Aforementioned issues raise two important questions to be addressed by researchers: (i) how to better explore system-level design of OIN-based systems , and; (ii) how to control OIN-based systems without adding prohibitive overhead to the system.

1.2 Objectives & Contributions

The main objectives of this thesis are:

1. Introducing an efficient solution to help designers better explore the design space for systems integrating optical networks by finding the most promising solutions. This is achieved by providing a modelling strategy that is (i) complete, comprising all important characteristics of such systems; (ii) flexible, so it can describe any type of system; (iii) accurate, so the results extracted using it are close to real system implementations, and; (iv) not complex, so the modelling of systems does not impose a prohibitive an overhead during their deployment.

2. Developing efficient control solutions for OINs. The definition of a general solution for controlling OINs with a low-latency solution, that is flexible, in this way being able to deal with most kinds of optical networks and has a low overhead in time, thus not adding many extra clock cycles to compute requests and set dynamic configurations.

By fulfilling these objectives, we realize three main contributions:

1. The definition of accurate system-level modelling method enabling the development of a system-level simulation platform;

2. The definition and development of efficient control approaches for OIN-based systems, and;

3. The system-level evaluation of the proposed control approaches using the defined modelling methods.

1.3 Document Organization

This document is organized as follows. Next chapter presents basic concepts about photonic devices and networks as well as control techniques usually applied on such networks. Chapter 3 brings a state-of-the-art revision, positioning our approach with existing works. Chapter 4 gives first the global overview of the approach we defined for system-level design of OINs

and presents the proposed accurate system-level modelling methods. Following, Chapter 5 introduces the centralized control unit, employing Look-Up tables (LUT) in order to speed-up the control unit. Chapter 6 presents the hybrid controller developed, which relies on a centralized core and distributed units to reduce the control latency. Chapter 7 brings the simulation platform developed using introduced description models. Following, chapter 8 presents obtained results for both deployed controllers, the simulator accuracy as well as the models integration outcome. Finally, chapter 9 draws the conclusion of this document, along with the possible future work in the field.

CHAPTER 2 BASIC CONCEPTS

This chapter presents general concepts about optical components and systems, their control schemes as well as modelling techniques required to describe them. Section 2.1 covers topics such as waveguides, insertion loss and optical filters, from integration with complementary metal-oxide-semiconductor (CMOS) technology perspective. Section 2.2 discusses controlling solutions for OIN-based systems. Finally, section 2.3 brings an introduction to modelling and description concepts on the scope of multi-processed systems.

2.1 Basic Concepts Related to Optical Interconnects

Optical interconnects are well known for their capacity to transfer data with high transmission rates. On the opposite of their electrical counterpart, optical interconnects use light over an optical via to transmit information instead of wires. In order to take advantage of the maturity of the CMOS manufacturing technology, Silicon is the material of choice in order to deploy integrated optics [26]. One approach used is Silicon-on-Insulator (SOI). The SOI substrate is made of a thin silicon top layer separated from the silicon substrate by a buried oxide layer. The strong refractive index contrast between the core and the cladding material of the waveguide allows the realization of very compact components [27]. Other approaches are also possible, such as doped silica [28], silicon nitride [29] or silicon oxynitride on oxide [30]. Also, it is possible to separate the manufacturing of the electrical and optical circuits. In this case, the devices are made on different wafers, but still relying on the same materials mentioned before and then later bonded. In this thesis, we employ SOI fabrication technology as reference.

All optical devices are composed of some basic structures:

- **A via** is the path where the lights travel on, being the equivalent of wires in electrical devices;
- **An interface** is used to input data on the via or read/receive data to/from the via, and;
- **Active and passive components** are responsible for the control of the data flow on the device working like a filter. Passive components are static, while active components can be dynamic. Both types are detailed in Subsection 2.1.3.

An optical via is called a *waveguide* and is the path on which the light travels, usually with a rectangular structure. Nowadays, designs are projected with a waveguide width greater or equal to 450 nm and, as the size changes, different wavelengths might be transmitted on it.

Different *wavelengths* might be used at the same time on the same waveguide transmitting data in parallel, provided that the waveguide supports these wavelengths. It is important to highlight that simultaneous connections are one of the main advantages of optical interconnections when compared to electrical-based ones.

As previously mentioned, waveguides are the paths on which light travels, and they should be as straight as possible to allow better transmission quality. However, it is nearly impossible to have a system composed exclusively by straight lines, being necessary to *bend* the waveguides in order for the light to change its direction. This is a crucial design step, as light might be completely lost in a very abrupt bend. Also, despite the fact that this may introduce a given signal loss, waveguides might *cross* each other without causing a non-functional state of the system, like it would in electrical crossing.

Figure 2.1 shows the three main types of waveguides: **(a)**: *straight waveguide*, transmitting from one side to the other, exclusively; **(b)**: *crossing waveguides*, where two inputs and two outputs can coexist, and; **(c)**: *waveguide bending*, changing the transmission direction.

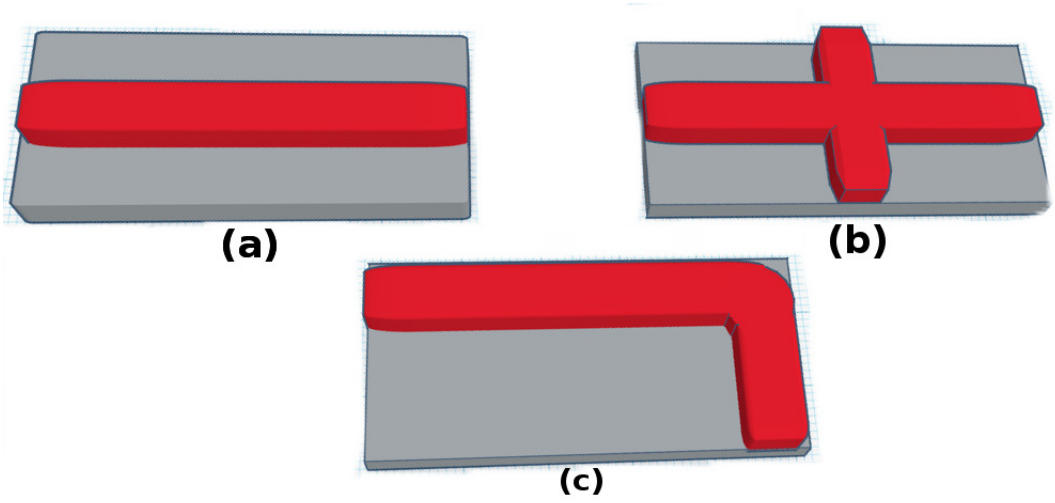


Figure 2.1 Waveguides examples **(a)** *straight waveguide*, **(b)** *crossing waveguides* and **(c)** *waveguide bending*.

When using optical components, one important aspect to be considered is the system *insertion loss*, which directly affects the power required for an input to reach its output. So, the higher the insertion loss, the higher is the power needed on the input. Each component introduces a given loss on the input signal, represented in Figure 2.2.

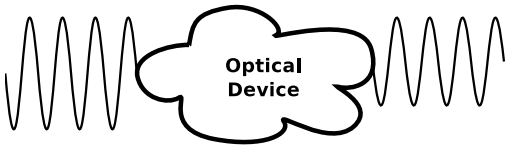


Figure 2.2 Insertion loss overview.

Additionally, *total optical loss* can be expressed as the sum of internal losses, in decibels (dB) [1]

$$Loss_{TOTAL} = Loss_{input} + Loss_{bending} + Loss_{w} + Loss_{output} \tag{2.1}$$

where $Loss_{input}$ is the loss of the input signal when it couples into the waveguide, $Loss_{bending}$ is the bending loss, $Loss_{w}$ is the waveguide loss, and $Loss_{output}$ is the loss of the output [31].

Another important effect to be studied and understood is the *crosstalk*. This happens when part of the signal couples on different waveguides, and it is not desired to happen. Figure 2.3 presents an example of crosstalk in a ring resonator, where it is possible to see the signal being transmitted from left to right, with the undesired fact that part of it is being coupled on the ring and going to the output on the bottom. This kind of issue introduces noise on the communication, which reduces the transmission quality and may jeopardize the reliability of the output signal. This is a risky situation because, sometimes, so much noise is inserted on the channel that is not possible to distinguish if the output signal is real data or only noise.

Most designs focus on reducing both the insertion loss and crosstalk in order to obtain a more efficient architecture, with a better *Signal-to-noise Ratio* (SNR) [32]. SNR is a value that compares the relation of a noise signal to the desired signal over a component. This means that higher the ratio is, higher the transmitted signal quality is.

It is not possible to build an entire system relying solely on waveguides, as it would be necessary to create dedicated links for all possible communication paths, which is not feasible.

[1] Decibel indicates the proportion of a physical quantity in relation to an entry point. In other words, decibels express a power ratio, not an amount. The simple equation which defines a decibel is: $A = 10 * log_{10}\frac{P2}{P1}(dB)$, where P1 is the power being measured, and P1 is the reference to which P2 is being compared. The conversion from decibels measure back to power ratio is given by: $\frac{P2}{P1} = 10^{(}\frac{A}{10})$. For instance, the ratio of 1 kW to 1W, in decibels is: $G_{dB} = 10 * log_{10}(\frac{1000}{1}) = 30$ dB.

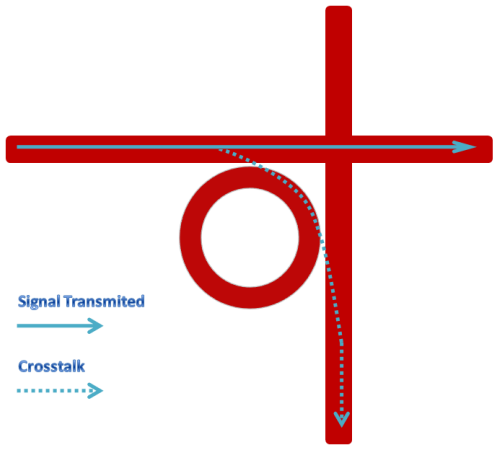


Figure 2.3 Crosstalk example of a ring resonator, showing a portion of the signal being transmitted by the undesired waveguide.

This creates the need for more elaborated components, such as the Mach-Zehnder Interferometer and Micro-Ring Resonator. These components incorporate extra functionalities, enabling the flow control in the network, working as network switches. Both components are presented in detail hereafter.

2.1.1 Mach-Zehnder Interferometer

A *Mach-Zehnder Interferometer* (MZI) is a device used to control the amplitude of an optical wave by dividing it in two, applying a given delay and then merging the two beans of light into one again. The physical implementation of such functionality works as it follows: the inserted wave is split in two, maintaining the same phase and amplitude for both resultant waves. One wave is transmitted over a straight waveguide and keeps its phase and amplitude. The second wave is somehow delayed, which inserts a phase change. Wave phase can be defined as the position of a wave point in one specific time. When a phase change happens, it means that the wave moved in time. Figure 2.4 illustrates this event, where two sinusoidal waves are presented. As it is possible to see, one sinusoidal wave is slightly forward in the X axis, which represents time. This can be defined as a phase change. By the time both waves are merged, the two waves can interfere constructively or destructively at the output. Depending on the interference, the introduced phase change affects the amplitude of the resultant wave [33].

For the shift phase, two approaches might be used: one active and one passive (the concepts of active and passive devices are covered in Subsection 2.1.3). Figure 2.5 presents the basic structure of one MZI, where it is possible to see the presence of two parallel waveguides, one

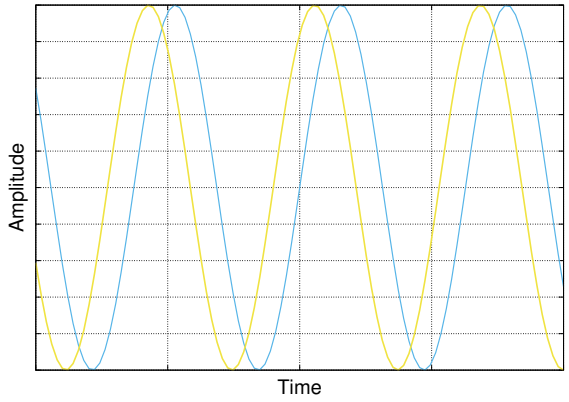


Figure 2.4 Shift phase example, showing a sine wave changing its phase, thus moving in time.

with a delay portion and one as a straight line. In the image, it is possible to see an input signal being split in two, so they can travel each on their own waveguide, being later merged into one signal, on the output.

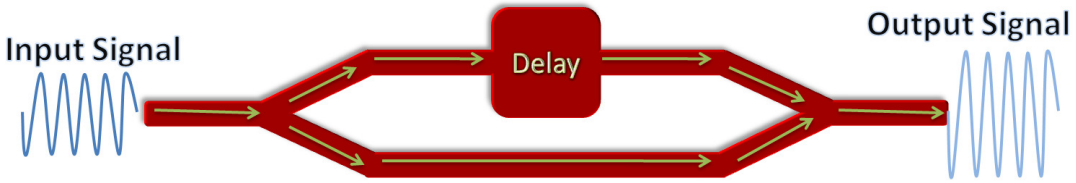


Figure 2.5 Mach-Zehnder Interferometer example structure.

For the delay that introduces the phase shift to be applied to the wave, we can use either an active or a passive device, as previously mentioned. For the active device an electrical arm is attached to the waveguide and a given current is applied to it. Depending on the applied current a different phase shift is induced. For the passive counterpart the phase shift is fixed at design time and may never change, being achieved by adding a series of waveguide bends in order to change the phase.

In order to build a MZI-based switch, two inputs and two outputs are employed. The light is split in the two arms of the input coupler of the interferometer, and they are later recombined in the output coupler of the interferometer. The switching between the ports is achieved by an electro-optic effect within the structure. Voltage, applied to the electrodes deposited on the integrated MZI, alters the electric field distribution within the substrate, which consequently changes its refractive index. By changing the effective refractive indices of one of the arms, it is possible to generate the phase difference between the optical signals in two arms of MZI. Based on the generated phase difference, the light switches from one output port to the other.

The following equations are used in order to determine the normalized power in each output port, where $\Delta\phi$ is the difference of two phase changes [34].

$$P_{out1} = sin^2(\frac{\Delta\phi}{2}); \tag{2.2}$$

$$P_{out2} = cos^2(\frac{\Delta\phi}{2}); \tag{2.3}$$

Figure 2.6 illustrates the 2×2 MZI-based integrated switch, where the input and output ports, the splitters and the integrated electrodes are presented.

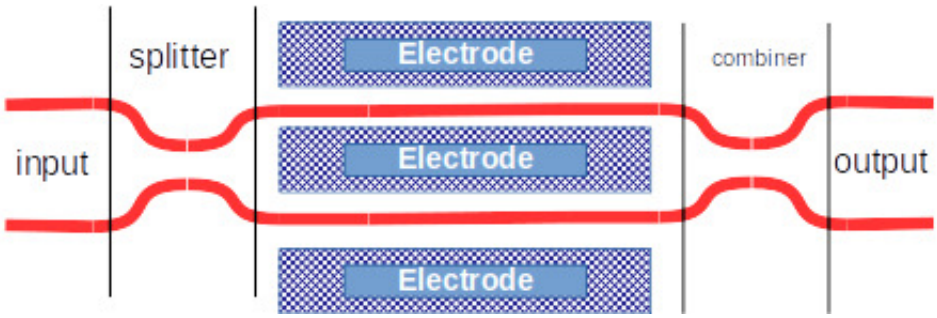


Figure 2.6 2×2 MZI-based integrated switch structure.

2.1.2 Micro-Ring Resonator

A *Micro-ring Resonator* (MR) is an optical filter applied to select a desired *wavelength* from a given input and redirect it to an output [35]. On opposite to the presented MZI, MRs are wavelength selective. The MRs work based on three principles: optical coupling, total internal reflection, and constructive interference.

Total internal reflection is an effect in which the light travelling within a waveguide remains in the waveguide. It happens when the light hits the boundary of the waveguide and does not refract over the boundary. Constructive interference is when two waves interfere on each other and the resultant wave is the sum of the two interfering waves. Optical coupling is the effect of light travelling from one media to another. Particularly in MRs, it is the effect of the light travelling from the waveguide to the ring and from the ring to the waveguide. Three characteristics affect the optical coupling: the distance between the ring and the waveguide, the length of the coupling region and the refractive indexes[2] of the waveguide and the ring. To

[2]The refractive index of a component is a constant that indicates how much the light will bend, or refract, in that medium. It is defined by $n = \frac{c}{v}$, where c is the speed of light, in vacuum and v is the phase velocity. For instance, the refractive index of silicon is 3.42, meaning that light travels 3.42 times faster in a vacuum

optimize the coupling, usually the distance between the ring and the waveguide is narrowed.

Figure 2.7 shows the structure of a simple MR used as a filter. In the figure it is possible to notice the presence of two *waveguides* and one ring between them. On the top *waveguide*, there is one *input* signal that might be transmitted directly, leaving by the *through* port, or it can be coupled into the ring and redirected to the *drop* port, on the lower *waveguide*.

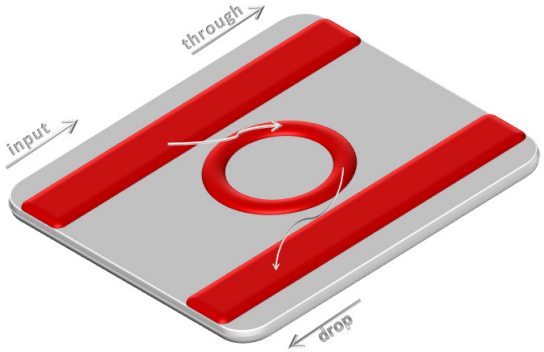


Figure 2.7 Example of a filter using MR.

The decision if a *wavelength* is coupled or not into the ring is defined by some MR's physical characteristics, like the round trip length and the transmitted wavelength. For instance, assuming wavelengths λ_1 and λ_2 are transmitted, if λ_1 satisfies the resonant condition, such as

$$n_{eff}L = m\lambda_1, \tag{2.4}$$

the wavelength λ_1 is coupled. As a result, the wavelength λ_1 will couple to the ring while wavelength λ_2 will not. Here, n_{eff} is the effective index of the input waveguide, L is the length of the optical round trip length and m is the mode number.

Such a capacity to select a given wavelength to be coupled into the ring makes this kind of structure useful on the communications domain, since this behaviour is the same presented by a switch. The implications of this capacity are tremendous considering that MRs are very cheap structures in terms of footprint and power consumption, which makes them a natural solution for integrated optical communications.

2.1.3 Optical Integrated Networks

Optical integrated networks take advantage of the previously presented components to achieve high bandwidth while maintaining low power consumption levels. It is possible to design

than it does in silicon.

basic-switch blocks using MZIs and/or MRs to route optical signals, and group these switches to create a network. Several works were published showing this kind of structure and great improvement on both throughput and bandwidth, with lower latency and power consumption [36–40].

Figure 2.8 shows the structure of a basic MR-based switch used to select if an input (**IN**) will either communicate with one output (**OUT_0**) or the other (**OUT_1**). This selection is performed by using different wavelengths; when **OUT_0** is to be reached, wavelength λ_0 is used. On the other hand, when **OUT_1** is to be reached, wavelength λ_1 is used. It is important to highlight that it is possible to use both wavelengths at the same time, thus transmitting information in parallel.

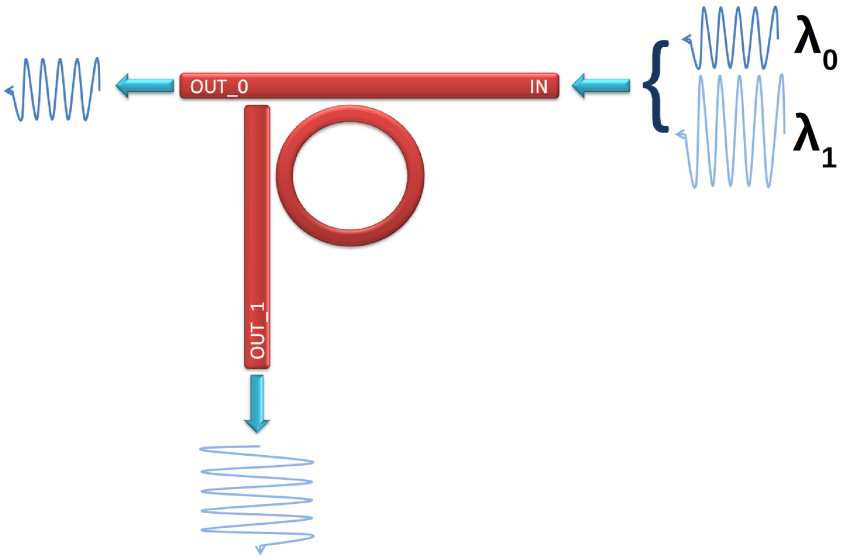


Figure 2.8 1x2 Basic switch using one MR, showing the selection behaviour for two transmitted wavelengths.

By simply replicating this structure, it is possible to design an entire OIN with no contention. In fact, most of proposed architectures [36] rely on similar structures to the one presented in Figure 2.8.

When defining an OIN architecture, the designer might choose between active components and passive components, or decide for a combination of them. Summarizing, these components differ only by the fact that passive components are static while active components are not.

Passive components are reactive devices by definition, which have their behaviours defined during design time. We can use a 1-input $\times$ 2-outputs MR as an example. In this case, during

design time, we define which wavelength(s) will be coupled into the micro-ring and follow a different direction, and which will not, following directly to the default output. This kind of device presents a low design complexity, with low footprint and low power consumption. Still, for scenarios with dynamic characteristics, this type of structure might not be the most suitable, as it would be mandatory to use as many devices as necessary to capture every single configuration needed on the system.

On the other hand, *active components* are designed in such a way that its routing characteristics may change during runtime, by adding an electrical control system to it. One example of the behaviour of active devices might be given by using the same MR as before, but now assuming it as an active component. Instead of a constant coupling in the same wavelength, it is possible to change the effective index by applying a given current on the MR, thus selecting another wavelength. As expected, active components are more area and power demanding, as they have a more complex structure and make usage of power to change their behaviours.

Other components are also usually present when using OINs: *signal converters*, *fast transceivers* and *Serializer(s)/Deserializer(s) (SERDES)*. Signal converters are used, as the name suggests, to convert signals between domains. The Intellectual Property (IP[3]) blocks, which generate network traffic are usually digital. So, digital data must be converted to optical signals to be inserted in the network and in the same way, the optical network output has to be converted to digital signal to be processed by IPs. Still, when using OINs, the designer can define whether the network will send messages bit-by-bit or word-by-word. In cases where the sending is performed bit-by-bit, it is necessary to use a SERDES, a device applied to the serialization and de-serialization of a message. On the sender side, this device receives a word and returns it, bit-by-bit. On the receiver part, the SERDES receives a word, bit-by-bit and returns it as one entire word. Figure 2.9 illustrates an example of a SERDES block. The input message is an 8-bit word array. The figure represents the serialization, where each bit is shown one after the other and lastly the deserialization, where all bits are grouped as an 8-bit word array again. Finally, fast transceivers are used to adjust the data rate injection and reception, as the IP blocks operating frequency might be different than the frequency needed on the network.

[3]An IP block can be defined as any computational unit: a processor, a memory or a DSP, for example. In this context, it represents any type of node connected to the network.

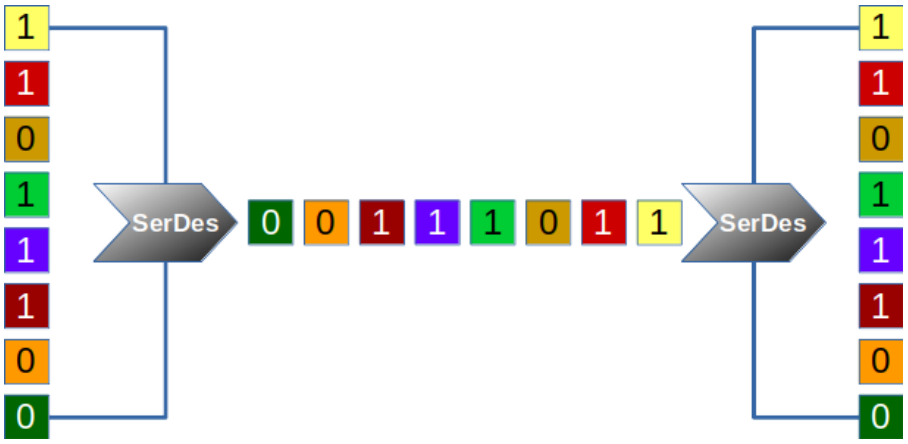


Figure 2.9 SERDES block exemplification showing an 8-bit word being serialized and later deserialized.

Figure 2.10 shows an organizational example of a system using an OIN with the aforementioned components. In the figure, the internal structure of the OIN is not detailed, rather highlighting only the connections with the IPs. The figure also illustrates the connections among components.

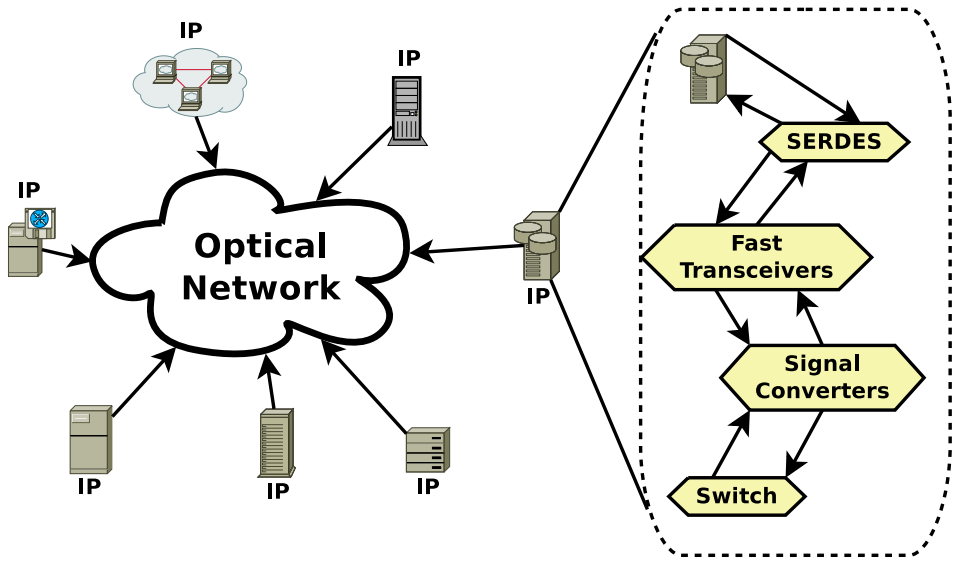


Figure 2.10 Organizational example of OIN-based system illustrating the data flow, passing by the IP, the SERDES, the transceiver, the signal converter and network switch.

2.2 Control Unit

When talking about OINs, the design of the controller of the system is as important as choosing the right topology. Unlike eNoCs, optical networks do not rely on buffers to temporarily store data at every switch. If that were the case, it would be necessary to convert all data to digital signals and then re-convert back to the optical signals, leading to high and undesired costs. Thus, the controller algorithm is usually based on one of the following techniques:

- **Time sharing**, where time windows are set for each IP to transmit its information. On each time window, a set of IPs is granted to send their data, while the others are stalled until the time window ends;
- **Dynamically paths setting**, which presents a behaviour similar to the one found in circuit-switching eNoCs. The path is defined at the beginning of the transmission. In this context, the controller should compute all the input requests and choose all paths, either by assigning wavelengths on passive-based networks or by tuning the components of active-based networks, and;
- **Wavelength division**, uses the parallel communication capabilities of optical devices to transmit separate signals. This technique divides the total available bandwidth into a series of non-overlapping sub-bands, assigning a different wavelength for each transmitting IP. Usually this technique is employed for passive networks, but not exclusively.

2.2.1 Time Sharing

There are cases where the topologies are built using components that have exclusive access, i.e., allows a single transmission at a time. When this happens, the control unit must manage the requests and granting in a manner that no request waits forever. Figure 2.11 presents a scenario composed of seven communicating blocks that share a single communication medium and a centralized controller. In this case, it is possible to notice conflicting situations that may occur when more than one communicating block tries to access the shared media at the same time.

To prevent this kind of situation, a common approach is to use a time sharing/division multiplexing (TDM) algorithm [41], which divides the time into several recurrent windows (slots), being one for each IP. The time slots might have the same duration depending on the priority attributed to the IPs. Also, the amount of information transmitted over the media on each slot may either be the same for all the IPs or vary according to their priorities.

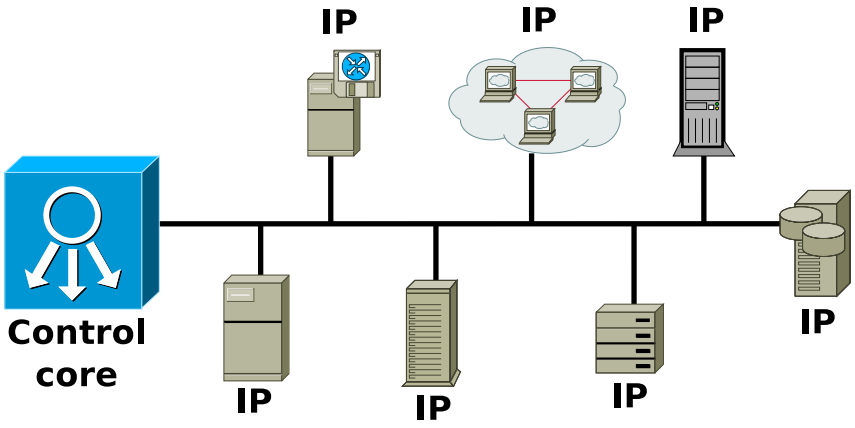


Figure 2.11 Resource sharing system example, where one simple bus is shared between different communicating IPs. The controller (left box), receives requests and grants access to one IP at a time.

The simplest implementation for this kind of situation is based on the Round-Robin Algorithm [42]. It implements a first-in-first-out (FIFO) queue, which stores in each position a single IP ID. For every new time slice, the ID found on the next output of the FIFO has its access granted. This kind of control imposes a fair time overhead on the system, being usually deployed on a single, centralized control unit, being more suitable for a lower design complexity, such as low radix systems.

Figure 2.12 shows an organizational example of using an OIN with a centralized control unit. In the figure both the control and network are integrated as one black box.

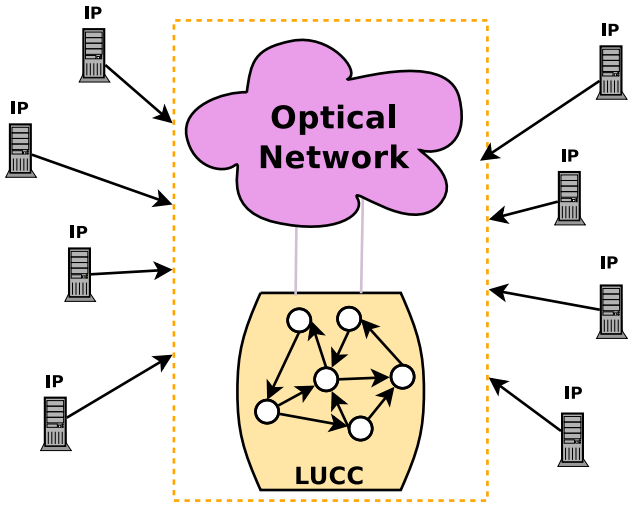


Figure 2.12 System overview using a centralized Time-sharing-based control unit.

2.2.2 Dynamically Paths Setting

Another possibility when controlling OINs concerns a circuit-switching (CS) technique. It is a suitable technique used in dynamic network configurations, where an electrical layer has access to all network nodes and configures those needed for each communication to perform correctly. The main appeal of the circuit switching is its utilization on the 3D stacked integrated-on-chip systems, in which each layer of the chip holds one parcel of the entire architecture [43].

In circuit-switching, each optical network node is directly connected to one electrical network node. The latter is then physically placed above/under the optical node, using intra-connections such as Through-Silicon Vias (TSVs) [44, 45] to share information between each other. The IPs are connected to electrical nodes and requests travel along all the electrical path, thus closing their way from the origin until their destination. By the time the path is closed, the message starts to be sent through the optical path.

Figure 2.13 shows an overview of the stacked networks and their connections. In the figure, the network nodes are illustrated as blocks, where each block is a network router. Figure 2.13(a) presents the top system view. Figure 2.13(b) shows a lateral systems view, where it is possible to see the connections between the electrical and optical nodes. Figure 2.13(c) illustrates the optical network layer. Finally, figure 2.13.(d) indicates the electrical layer showing the IPs connected to the routers.

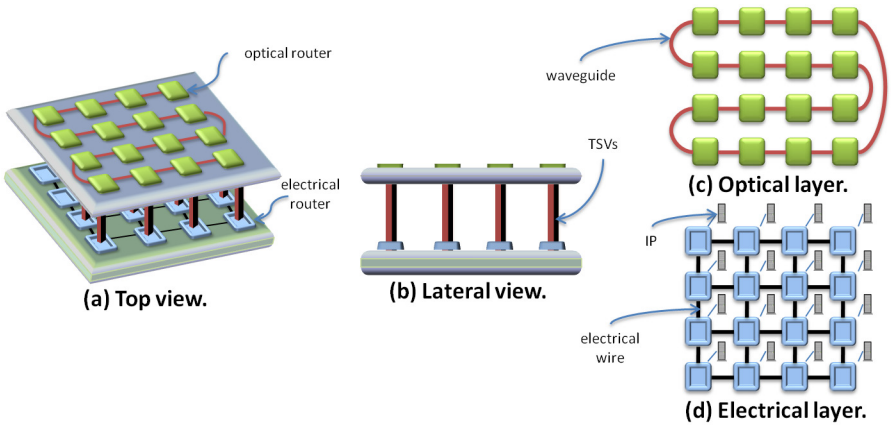


Figure 2.13 System overview using a distributed circuit-switching-based control unit; (a) presents the upper view of the system, where the optical layer is placed above the electrical layer; (b) shows the system view from the side. The layers connections are presented as arrows, and communicating nodes as rectangles; (c) illustrates the optical layer, its switches and connections; (d) presents the electrical layer, its switches, IPs and connections.

In this approach, the tuning time can impose a high overhead. Even though, it remains very

interesting for systems where large messages are exploited. This occurs since the high cost to close the path can then be compensated by the gain to transmit the message.

2.2.3 Wavelength Division

The last technique applied to control units for OINs is the Wavelength Division Multiplexing (WDM) [46]. This technique is very similar to the TDM. However, instead of using different time slices, WDM employs different wavelengths. When using WDM, the available bandwidth of the channel is divided into sub-channels. For each sub-channel a given wavelength is attributed [47].

Figure 2.14 depicts an example of the usage of a WDM-based controller interacting with four requesting IPs which requires access to the network simultaneously. For each one of them, the controller attributes one different wavelength (λ). This way, the four IPs can transmit data in parallel.

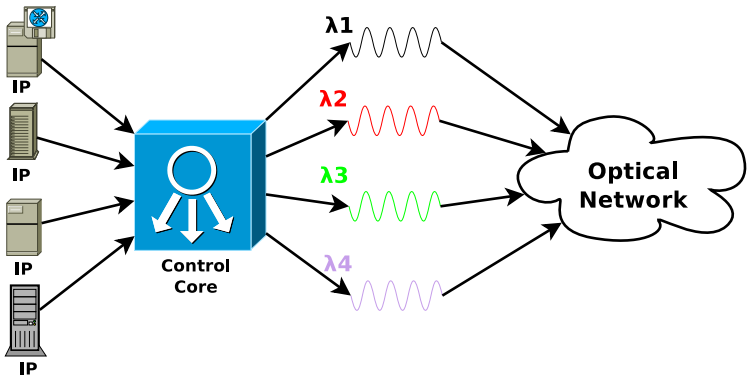


Figure 2.14 System overview using a centralized wavelength-division-based control unit.

This technique allows wavelengths overlapping to be used for two different scenarios: (i) different IPs requesting access simultaneously, as presented in Figure 2.14, and; (ii) for the cases when only one IP is requesting access and all wavelengths are free. In both cases, the controller might attribute more than one wavelength to the same IP, so its transmission might occur in a parallel stream.

2.3 High-level Modelling Strategies and Description Languages

Large system developers usually struggle to precisely identify all aspects that comprise their design, such as the number of components[4] and proper abstraction implementation. In

[4] in this context, a component can vary between a logical port and a processing unit.

current digital systems, such as multiprocessor embedded systems, it is possible to find from hundreds to thousands different components working together, communicating with each other and depending on each other's responses.

Therefore, for a designer to effectively capture and manage all these aspects, higher abstraction levels are mandatory. In this way, a model might be defined as a simplified vision of anything real and, in order to decrease the system design complexity, different models are used. For instance, a computation system can be defined as a set of digital ports, as a series of logical transactions or as a single I/O block.

Diverse challenges arise when modelling current systems. Among those, we highlight model complexity, simulation runtimes and analysis complexity, model abstraction, complex interdependencies, and parallel components.

The Unified Modelling Language (UML) [48, 49] is a general-purpose modelling language designed to be used as a standard description modelling language. It relies on visual blocks to describe all system architectural contents: transactions, components, timing, data flow, etc. Most visual high-level modelling tools use the standardized patterns (UML blocks and their connections) adopted in UML.

In the case of the Petri nets [50], directed graphs are used to model distributed systems, and the system is described as a series of places, transitions and directed arcs. Each place represent a component or logic block. All these places are connected by arcs. For the flow in the net, transitions are used, which represent the movement from one place to the other, passing through transitions. The system flow of a Petri net always follows the same behaviour, where one arc moves towards another passing by one transition.

Another model type is the Kahn Network [51], in which sequential processes communicate through FIFOs. Its main characteristic is the synchronization imposed by the FIFOs, which makes this kind of model suitable for distributed, communicating systems.

Besides specific models, different languages are defined for specifications as well. Although their usage is not restricted to one specific functionality, their focus is usually well defined. Architecture Description Languages (ADL) [52] are designed to describe and represent system architectures. ADLs might be used to describe both software and hardware. For software, ADLs cover features such as processes and threads. For hardware description, ADL describes components, such as processors, devices and buses. Also, the connection between the components is described in ADL. For instance, a processor can be described in ADL as a simple device that receives requests and returns a value.

One example of ADL is the Architecture Analysis and Design Language (AADL) [53] also

known as Avionics Architecture Description Language. It is largely used in the modelling of real time embedded systems, to describe either hardware or software components.

Hardware Description Languages (HDL) focus on describing the characteristics of hardware components. It is possible to describe logical ports, components such as processors and memories and even systems. The main appeal of HDLs lies in the fact that they hold a close relation to the physical layer. Still, it is possible to use HDLs in a way that such physical aspects are abstracted and only the behaviour is captured. This makes HDLs powerful description languages to be used when designing systems. By using these languages, attributes like time, components concurrent execution and connecting signals can be modelled. The most common languages in this group are VHDL [54] and Verilog [55]. Programming languages are a different group of specification languages, where the focus is on software deployment. Usually, they act as an interface between the hardware and the user. Common examples of this group are C++ [56] and Java [57].

Throughout the years, the modelling abstraction level has been upgraded and the introduction of digital designs has led to the need for reducing design complexity by increasing the system-level view. Figure 2.15 presents the evolution of the abstraction level used on the deployment of digital systems. Starting from the transistor - the fundamental design unit - the abstraction level started to rise. First, a digital system was represented as a set of logical ports, then registers, and finally as hardware and software models. In the figure, four rows are presented: the timeline, the device abstraction, the modelling level and the system overview.

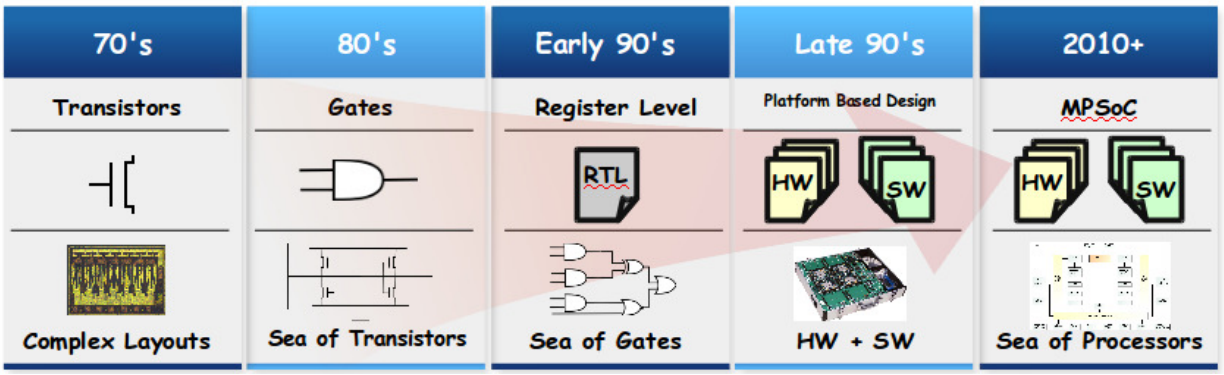


Figure 2.15 Design Abstraction Levels

It is possible to see that the level of modelling method cope with the complexity of the state-of-the-art systems. As new functionalities are added to systems, the modelling strategy has to be updated as well. At first, systems were simpler than nowadays, which made it possible

to describe them one transistor at a time. Later, systems evolved from groups of transistors to big sets of logical ports, which lead to the utilization of Register transfer level (RTL) languages. Nowadays, systems incorporate different elements, as hardware components and software logic. For designers to capture all different aspects involved in the description of such systems, higher-level modelling strategies are adopted. The concepts of high-level modelling, like ADLs and UML are employed for these systems. In the next systems' generation, not only different elements will be present, but also different technologies, where systems will be heterogeneous. Future systems will be composed of digital hardware, such as processors; magnetic components, as magnetic RAMs; logic models, as software; optical components, such as OINs. This makes mandatory the update of current design methods. The update may be achieved by the definition of new modelling languages, where such aspects can be described. Another possibility is the re-utilization of nowadays modelling languages, by employing them in different contexts. While developing new languages for the description of future systems would make them perfectly suited to the context, the re-utilization is appealing, as it takes advantage of mature approaches.

CHAPTER 3 RELATED WORK

This chapter summarizes the most important related works in the field of OINs modelling and simulation as well as controllers for OIN-based systems. Section 3.1 introduces a set of tools and methods to simulate, model and evaluate optical systems. Their description will be divided into two sub-sections: one presenting a discussion about industrial tools and one focusing on academic tools. Moreover, Section 3.2 presents a discussion of state-of-the-art controlling solutions for OIN-based systems, where recent and relevant publications on the field are discussed and their methods are explained.

3.1 Modelling, Simulation and Evaluation

The design of optical devices is a complex task, requiring different variables to be considered. Optical devices are sensitive to dynamic variations, such as temperature and even a small deviation can change the device behaviour, making it mandatory to capture all those small peculiarities. This way, different approaches and tools were made available by either industrial or academic research. These tools can focus on the entire design flow or specific aspects, such as power loss. In this chapter we will focus on tools considering the design from a system-level perspective.

3.1.1 Industrial Tools

Facilitating the employment of optical devices, several companies provide tools and methodologies for engineers to use. Those tools usually cover the entire design flow, from the very beginning, when each device is modelled and the integration of developed devices. These kind of tools depends on numerical methods and solvers to compute all variables that compose the system, which are usually complex and processing demanding.

One example of industrial state-of-the-art company is Lumerical Tools [58], which provides an environment for the deployment of optical devices, from the initial level till the integration level. It has four applications that might be run solo or in an integrated way, following a design plan. Designers can start their device from scratch using MODE, adopted to model the device and make initial verification, like group index. The second application in the design flow is FDTD, which is a 3D Maxwell solver that analyzes the light interactions on different wavelengths scales. Following, Lumerical presents INTERCONNECT, a photonic integrated circuit design environment that enables the design, simulation and analysis of

photonic integrated circuits. INTERCONNECT is able to perform analysis either on the frequency domain or in the time domain. Last of four Lumerical Tools is the DEVICE Tool which is used to integrate photonic and CMOS devices.

VPIPhotonics provides a set of tools for the design of Photonics devices with the difference that most of their tools focus on long-distance communication systems. The provided tool for Photonic Devices design and optimization is the VPIcomponentMaker Photonic Circuits [59], a tool with a focus on photonic integrated circuits (PICs), electric circuits, and optoelectronic devices.

Optiwave [60] is a design suit for the deployment of photonic devices. It offers a great list of applications to ease the project of such systems, making it possible to even integrate their tools with available electrical tools, like Matlab [61]. Optiwave also provides SPICE tools for integration with CMOS devices.

Some other tools might be cited as well, like OptSim [62] and Optilux [63], which also provide support for the development of Optical Devices. These tools present features such as numerical solvers for the analysis of physical aspects of modelled devices.

Aforementioned tools are accurate and provide feedback, relying on precise models to represent the system. By using them, designers can entrust that obtained results are very close to the fabricated devices. However, as implemented models depend on complicated equations and analytical models, their computation time is large, jeopardizing their effectiveness for large systems, limiting their usage to small systems. Although it is important to acknowledge the high complexity to attain the level of accuracy of such tools, the fact that they do not contemplate the entire system design puts at risk the time-to-market.

3.1.2 Academic Tools

In order to support system-level design, initial tools were made available by academy. These tools are meant for academic purposes and are firstly focused on proving new methods and theories. Also, such tools are usually designed for one specific purpose, like analyzing crosstalk [64, 65], thus not allowing an entire system deployment. In this way, with very specific purposes are not covered in this review.

Among system-level tools, it is possible to split them in two groups: extensions on already existing simulations tools and the ones implemented from scratch.

PhoenixSim [66] is an example of a simulator extension. It is built based on the OMNeT++ [67] framework, a well-known network simulation platform that allows users to extend its functionalities. PhoenixSim may be used for designing and analyzing the per-

formance of photonic interconnection networks, generating reports for propagation delay, insertion loss, extinction ratio, spectral resonant profiles, area occupation, and energy dissipation.

GRAPHITE [68] is a distributed simulator for multi-core systems. The focus of this tool is to reduce the simulation time, by splitting the computation overhead among different CPUs in a network, thus giving a fast feedback on different validation scenarios.

An accurate and promising tool was introduced by [69], named DSENT. This tool presents models to simultaneously simulate electrical and optical devices and providing different reports, such as area usage and power consumption. The developed models are precise and rely on information extracted from physical libraries for different technologies, such as 45 nm, 32 nm, 22 nm, 14 nm and 11 nm.

Some other tools used as bases for the design of system simulators are Booksim [70] and NoCSim [71], both projected to validate eNoCs but, thanks to similarity between approaches, can be extended to validate ONoCs as well.

The main issue with these tools is that they are still deeply attached to a component view, making their usage difficult for dynamic scenarios, with applications running and real-time changes occurring. Also, most tools are designed to exploit one specific design constraint, such as power loss, thus leaving the support for all other aspects unattended. While their focus was on solving one specific metric, it is important to highlight that all the design context must be covered.

In this work, the Straight-forward Simulator (SF-Sim) is introduced. The fundamental difference of the aforementioned tools and the SF-Sim lies on the fact that the proposed tool gives engineers an accurate feedback, maintaining obtained results close to the ones obtained in real systems. At the same time, the SF-Sim enables the design flexibility of a system-level perspective. Designers have the opportunity to create systems from scratch by using pre-defined components in a library or by adding new ones using the developed methodology.

Table 3.1 presents the comparison between previous tools and the one proposed in our project, emphasizing the following points:

- **Model Style:** description model provided by the tool. Might be textual, graphical, etc;
- **Level:** abstraction level of the tool;
- **Accuracy:** the given accuracy, such as: cycle, physical, events, etc. For instance, if the tools run at a clock-base, it is defined as cycle accurate and if it runs as series of

events, such as application transactions, it is defined as events;

- **Goal:** focus of the tool. If a tool lets users design an entire system, but is meant for devices only, the design goal will be defined as Device;
- **Status:** if the tool development and user support is either active or inactive;
- **Licence:** the type of licence required for users to have access to the tool, and;
- **Flexibility:** the flexibility to describe models using the tool, from a system-level perspective. In other words, the degree of easiness to describe a system, using available modelling strategy. This comparison is performed based on the functionalities presented by the tools as well as in hands-on experience. For instance, some tools present a deep support to evaluate preloaded optical networks. Still, if needed, the designer cannot add new topologies or systems, thus diminishing the flexibility of the tool.

Table 3.1 Tools Comparison

INDUSTRIAL TOOLS							
Name	**Model Style**	**Level**	**Accuracy**	**Goal**	**Status**	**Licence**	**Flexibility**
MODE	*Graphic & Text*	*Device*	*Physical*	*Component*	*Active*	*Proprietary*	*Low*
FDTD	*Graphic & Text*	*Device*	*Physical*	*Component*	*Active*	*Proprietary*	*Low*
Interconnect	*Graphic & Text*	*Device*	*Physical*	*System*	*Active*	*Proprietary*	*Low*
DEVICE	*Graphic & Text*	*Device*	*Physical*	*System*	*Active*	*Proprietary*	*Low*
VPI	*Graphic*	*Device*	*Physical*	*Component*	*Active*	*Proprietary*	*Low*
OptiWave	*Graphic*	*Device*	*Physical*	*Component*	*Active*	*Proprietary*	*Low*
OptSim	*Text*	*Device*	*Physical*	*System*	*Active*	*Proprietary*	*Low*
OptiLux	*Text*	*Device*	*Physical*	*System*	*Inactive*	*Open Source*	*Low*
ACADEMIC TOOLS							
PhoenixSim	*Graphic & Text*	*Device*	*Physical*	*System*	*Inactive*	*GPL*[1]	*Medium*
Omnet	*Text*	*System*	*Cycle*	*System*	*Active*	*GPL*	*Low*
Graphite	*Graphic & Text*	*Device*	*Physical*	*System*	*Active*	*Open Source*	*Medium*
BookSim	*Text*	*System*	*Events*	*System*	*Inactive*	*Open Source*	*Medium*
NoCSim	*Text*	*System*	*Events*	*System*	*Inactive*	*Open Source*	*Medium*
DSENT	*Text*	*System*	*Device*	*Physical*	*Active*	*Open Source*	*Low*
SF-Sim	***Graphic & Text***	***System***	***Cycle***	***System***	***Active***	***GPL***	***High***

[1] - General Public License (GPL).

3.2 Controlling Schemes

Most state-of-the-art works opt for designing a network controller that is suited for a specific topology or type of architecture, thus optimizing at most its performance. This way, most of proposed controllers are presented along a specific topology or router structure.

In [72], the authors present the design of a 5x5 MR-based router, which uses a mixed active-passive design approach. It is designed to passively route the optical signal travelling in one direction and actively route the optical signal making a turn. The authors claim that this router is designed in such a way that it is scalable to be used as a basic block for large networks. The control unit used for the proposed router is a circuit-switched-based one, where XY algorithm is applied on the electrical layer and closes the paths for the optical layer. The main drawback of this approach lies in the fact that circuit-switching techniques are time demanding and as the network's size grows their response time might get prohibitive.

In [73], a new architecture for photonic networks-on-chip is presented, along with its controlling scheme. The overall power consumption of the system is claimed to be reduced by using a dynamic resources provisioning. Each electrical router, on the electrical layer, is connected to a processor with private L1/L2 caches, and routers and processors are located above the corresponding optical switches. The electrical router and the optical switch communicate through TSVs[1] and optical/electrical signal converters on the control die. Each electrical router contains four I/O interfaces. The controlling layer works along with the Network Interface (NI). The NI role is to activate/deactivate laser sources in order to save power, after the controller triggers the beginning/end of transmissions. Once again, the technique used to define either if an IP will have access to the network or not is by using circuit-switching. Also, to guarantee message delivery, a given time is used before turning the laser source off, where the time is defined according to the number of hops the message will pass by and is dynamically calculated by the controller. The latency of such method was calculated to be around 3.5ns in an 8×8 network where resonators and peripherals run at 5GHz. The architecture was validated on a simulator built over Simics/GEMS framework [74]. The main comment that might be addressed for the solution proposed by the authors is that the usage of circuit-switching for path allocation is very expensive in resource allocation and, even for a fairly small network size (eight IP nodes), the control latency is high (for a 5GHz frequency operation, the period is equal 0.2 ns, meaning that it takes around 17 clock cycles for every request to be computed).

The work presented in [75] brings the design of an ONoC which is an extension of previous works, like the λ-Router [76], but reducing resource utilization. The technique used by the authors to reduce the resources utilization is similar to clustering, where the switches are grouped in small blocks, thus dividing the network in small versions of itself. The number of used resources in which a message may pass is reduced because as it goes in one direction all nodes in previous directions will never be visited. The control unit is based on wavelength

[1] Through-Silicon Vias are vertical structures used to connect different layers of a 3D on-chip system.

routing, where each I/O is assigned to one specific wavelength and might communicate at any time, without arbitration. Although this kind of structure solves any kind of contention, the overhead imposed by this solution is high, as for every I/O port, *'n'* buffers must be used, being $n = number\ of\ IPs$. Still, each I/O requires a specific wavelength, which is difficult to implement from an optical perspective. This is due to physical limitations, which curb the number of possible wavelengths to be used. This way, in a system with a big number of I/Os, not enough wavelengths would be available.

Authors of [47] introduce a routing technique based on wavelength selection integrated with spatial routing. The main contribution authors claim is that classic circuit-switching techniques are too time and resources demanding and based on that they present an extension of such methods. Their technique is based on the unused spectrum that exists between the resonances of a broadband ring switch by interleaving additional wavelength channels in the unused spectral space. That way, instead of having simple wavelength selection a wavelength division multiplexing (WDM) selection is used, where each group is an isolate case. The Circuit-switching technique is used along WDM and each router is composed of a junction of a receiver bank and a modulator bank. The design was tested with the electrical layer running at a frequency of 2.5GHz. The results for timing improvement of the introduced technique shows a reduced latency. Still, even for small messages passing (100-kbit messages, for instance), using a 100Gbps (gigabits per second) transmission rate, the checked latency reported is around 100 ns, which is high, taking into account the fact that this controller is fully tailored for the architecture it is designed for. Consequently, it is not possible to apply it to different network topologies.

An electrical-optical mixed approach is brought by [77], where a butterfly-based topology is introduced and each node is composed of an optical switch and an electrical switch. Optical switches are composed of MRs and are in charge of transmitting data on a circuit-switching fashioned way, while electrical switches are in charge of closing the path by using package-switching techniques. The main contribution of this solution is the fact that, after a set of trials, if the control unit is not able to close the optical path, the message will be transmitted using the electrical layer directly. As for the other cases where circuit-switching is used, the tuning time necessary to close the optical path is the main issue.

An asynchronous and variable-length packet switching is presented in [78]. This technique uses a two layered approach to dynamically set the message path over the network. Every IP is attributed to one exclusive label, which corresponds with each output fiber, for sending out switch fabric reconfiguration. While the message travels over the optical path, when it comes across a new network node, the message gets delayed while its label is computed by

the electrical node. Based on the message location over the network and the labels checking, the switch is reconfigured to direct the message to the correct output. The main drawback of this technique is that, depending on the delay time imposed by the control layer, the gain in using the optical path might be spoiled. Also, the delay necessary for this technique to be applied imposes modifications on the network layer. This diminishes the contribution, as it is not always possible and desirable to modify the network layer.

A multi-cast scheduling control solution is proposed in [79], which focus on input-queued switches based on the Weight Based Arbiter (WBA). The technique used by the authors is based on time sharing and aged-based weight calculations. The goal of this design is in achieving a time slot of around 51 ns for a payload data rate of 40 Gbps. The algorithm works by attributing weights to all the input cells at the beginning of every time slot and, in cases of equality on weights, the controller randomly chooses one of the conflicting choices. Results show a latency of around 20 ns for a 64×64 network. The main issue with this approach lies in the fact that it is suited for only one kind of network, as it was designed for the network the authors presented. This way, this technique would have to be extended, or adapted, in order to be employed in different networks.

Finally, in [80] the authors presented a controlling solution for contention handling based on optical-buffering, by introducing a three-stage buffering method. This method uses electronically controlled wavelength routing switches in combination with optical delay lines to temporarily store data [81–83]. The control unit works as follows. An optical threshold function which is driven by a message packet controls a wavelength routing switch, so that as this packet arrives, the next one is delayed for one packet period. Similarly, the routing of the following packet is determined by the presence of the first two packets. An optical arbiter that drives a wavelength routing switch is employed to decide whether packet contention takes place. The wavelength routing switch is operated by wavelength conversion in combination with a de-multiplexer. The optical-buffer-based scheme was tested on an experimental setup composed only by a laser and a modulator, which injected packets on a data rate of 2.5 Gbps. Although this technique is very promising, as it deals with optical signals directly, it still is at an early stage of deployment, not being suitable for short-mid term designs.

As explained above, most of related works present a network controller that is suited for a specific topology or type of architecture, which is said to optimize at most their performance. However, the topology-constrained controllers diminishes their contribution, as their employment is hard-limited by the very one topology they are built for. Further, this fact goes against the trend for the next generation of communication systems, in which it is believed that each network layer (application, control and physical) will be independent from

each other [24][25]. Also, a good number of works use circuit-switching techniques only, thus adding a control latency that could shatter completely the benefits of using an OIN-based system besides adding extra power consumption for each electrical node added to the control layer.

Table 3.2 Controlling solutions comparison

Reference	Topology-Constrained	Algorithm	Strategy	Latency	Scalable
[72]	*Hard Yes*	*CS*	*Distributed*	*High*	Yes
[73]	*Yes*	*CS*	*Distributed*	*High*	Yes
[75]	*Hard Yes*	*WDM*	*Distributed*	*Low*	Yes
[47]	*No*	*CS & WDM*	*Distributed*	*High*	Yes
[77]	*No*	*CS*	*Distributed*	*High*	Yes
[78]	*Yes*	*WDM*	*Distributed*	*Low*	Yes
[79]	*Hard Yes*	*TDM*	*Centralized*	*Low*	No
[80]	*Hard Yes*	*WDM*	*Distributed*	*Low*	Yes
Proposed 1	*No*	*TDM*[1]	*Centralized*	*Low*	**No**
Proposed 2	***No***	***TDM***[1]	***Hybrid***	***Low***	**Yes**

[1] - The implemented multiplexing control block is TDM-based, not using strict time slots. Still, the multiplexing block might be replaced without affecting the general controller behaviour.

This thesis presents two solutions in order to control OINs. The first solution relies on fast access memories to store pre-calculated routes in order to speed-up the network routing. The second solution, relies on an hybrid approach, where both a centralized control core and distributed configuration units are employed. Compared with the previously presented contributions, this work stands out thanks to the fact that our control units are suitable for a large variety of network topologies: the topologies based on tunable switches or the ones that only need arbitration. Topologies that just need arbitration are the ones where no network configuration is needed, such as the one based on wavelenght routing. For such cases, the controller works as an arbiter, avoiding destination conflicts. Topologies based on tunable switches are the ones where each network node has to be configured so different paths are correctly arranged. Moreover, as opposed to the controllers based solely on circuit-switching, the proposed controllers latency does not affect the overall system performance. Still, for the devices where multiple frequencies are allowed, designers can slightly modify the controller so it comports WDM routing.

Table 3.2 presents the comparison between presented controlling schemes and the ones proposed by this work, emphasizing the following points:

- **Topology-Constrained**: this property specifies if the control unit is somehow con-

strained by the controlled topology or if it has a generic behaviour. Three different categories are going to be used: (1) hard-yes, for the cases deeply constrained; (2) yes, for cases directly related to the topology, but somehow being applicable to others, and; (3) no, for the cases where the control might be used with none or small modifications for any topology;

- **Routing Algorithm**: the algorithm used by the controller in the network;
- **Strategy**: specify if the control is centralized, distributed or abstract;
- **Latency**: the time overhead due to the controller. An estimation based on the presented data and routing techniques is made. This information is normalized and simply presented as LOW or HIGH. The threshold from HIGH to LOW is a composition of different aspects of presented results for each work. For instance, a controller that shows a routing capability of 5 ns, but executes with a clock frequency of 10 GHz, thus taking 100 clock cycles to execute is considered HIGH. At the same time, a controller that takes the same 5 ns to perform routing, but executes with a clock frequency of 1 GHz, thus taking five clock cycles is considered LOW;
- **Scalable**: indicating the capacity to employ the approach in systems with a rising number of network nodes without risking its physical implementation.

CHAPTER 4 METHODOLOGY

This Chapter illustrates the defined methodology as well as the proposed approach for OIN-based systems modelling. Section 4.1 presents an overview of the defined methodology followed by Section 4.2, which illustrates the considered modelling strategy. In Section 4.3, the approach used to model the controlling solution is shown and, finally, Section 4.4 introduces the validation platform.

4.1 Methodology Overview

The methodology follows a step-by-step approach. First, basic characteristics, such as transmission delay, are modelled and validated, and then, new characteristics, such as power dissipation, are added and validated. The final goal is a simulation platform composed of two types of components: (1) OIN's devices models parameterized using fabrication results and (2) OIN controllers.

Figure 4.1 presents the development methodology as a series of iterative steps. The main steps of the defined methodology are described as it follows.

Constraints Analyses and Proposed Design Solutions

In the first step, optical integrated networks are analyzed while looking for the main constraints involved in their design and usage. Constraints such as the controlling aspects of OINs and the latency they impose are evaluated. Considering design constraints, aspects like OINs physical charachteristcs and their impact on the system behaviour are analyzed. Available approaches to surpass these constraints are studied and possible solutions evaluated. Subsequently, the controlling solution (algorithm, target platform, etc.) and modelling strategy for OIN-based systems are defined.

Models Definition and Refinement

Following, the controller as well as the components that compose an OIN-based system are modelled. The model definition and refinement step are divided in two main stages.

- **Controller modelling:** the model of the control unit is generated and later has its performance evaluated. To do so, firstly, we define the controller main blocks, defining aspects such as routing algorithms. Later, the design is simulated under different traffic

patterns and its latency is verified. Following, it is prototyped on an FPGA and later integrated with optical switches, lasers and photo-detectors in an optical lab in order to verify its behaviour under realistic (i.e., not simulated) scenarios.

- **Systems components modelling:** different devices are modelled. These devices include switches, routers, and links which are used to build a complete network. Also, besides networks and their components, blocks such as traffic injectors and traffic analyzers are defined in order to evaluate the network. Most of the existing works available in literature employ analytical methods for devices modelling [84], as analytical models for basic components are already available and are well proven [69]. However, these models are used to describe simple components and devices, thus leaving the system level unattended. It is a very complex task to model an entire system relying on available models, as it is hard to capture every aspect of the system. Models to enable the definition of entire systems, such as for optical switches and routers and ultimately at the network level are developed.

Models Integration

Considering the second step, both models for the controller and the system, are integrated in order to enable the deployment of an entire OIN-based system, composed of network nodes, traffic injectors and a control unit. At this stage, it is possible to execute realistic scenarios and obtain initial feedback, like the network latency.

Models Extension

Next, we extend the developed models, where both models for the controller and OIN components are improved. Once again, this step is done in parallel:

- **Controller:** the scaleability and frequency execution of the controller are mainly addressed in this step. Different aspects of the controller design are taken into account, so the controller blocks could be improved. To do so, the reports of synthesis tools for both ASIC and FPGA technologies are used.

- **System component models:** addition of different aspects to modelled components, presented in Section 4.2, such as insertion and transmission losses. To do so, extracted values from lab measurements are employed to create estimation constants, which are used in the models to obtain accurate results.

Final Integration - Simulation Platform

Finally, aforementioned models are deployed using VHDL (VHSIC Hardware Description Language) hardware description language and grouped in a simulation platform. This platform gives designers the flexibility to use, add or modify components and then validate them.

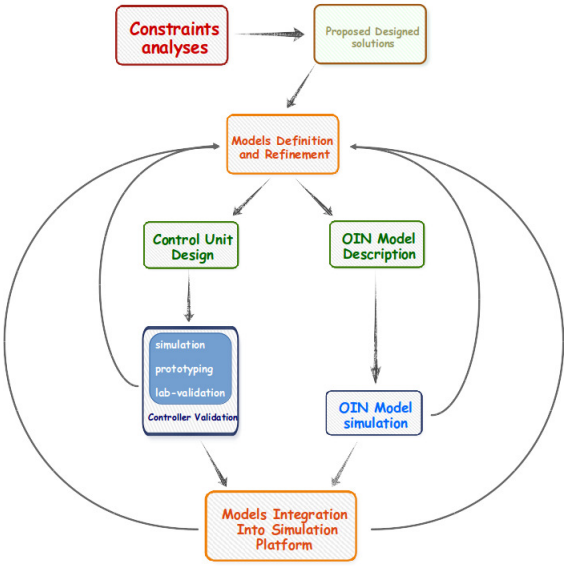


Figure 4.1 OIN-based system modelling iterative methodology.

Section 4.4 introduces the devices and topologies to be used to validate the controller presented in Section 4.3 as well as the model illustrated in Section 4.2.

4.2 High-level Modelling

In order to implement the required devices and components, a high-level modelling approach is performed to capture their different aspects. Models are divided into four main categories: **vias**, **devices**, **networks** and **controllers**. Each category can be used as a standalone model or along the others, as they have category affinities with each other.

A waveguide $\mathcal{W}$ is defined as a function of $\langle \mathcal{T}_\mathcal{W}, \mathcal{L}_\mathcal{W}, \mathcal{P}_\mathcal{W} \rangle$ with each parameter standing for:

- $\mathcal{T}_\mathcal{W}$: transmission delay in pico-seconds. In other words, it is the time it takes for the light to travel from the input port to the output port;
- $\mathcal{L}_\mathcal{W}$: insertion loss, and;
- $\mathcal{P}_\mathcal{W}$: stores the power penalty for each travelling wavelength on the waveguide.

Each modelled device, such as an optical switch, is expressed as a function with five parameters. This way, an optical device $\mathcal{C}$ is defined as an n-uple $\langle \mathcal{N}_\mathcal{C}, \mathcal{T}_\mathcal{C}, \mathcal{L}_\mathcal{C}, \mathcal{P}_\mathcal{C}, \mathcal{R}_\mathcal{C} \rangle$, where:

- $\mathcal{N}_\mathcal{C}$: number of inputs and outputs. For instance, a 4×4 switch has $\mathcal{N}_\mathcal{C} = 4$;
- $\mathcal{T}_\mathcal{C}$: is the transmission delay or the time it takes for an optical signal to travel from an input to an output port. As there are more than one input and/or output, and the variation for each pair of I/O is small at a system-level perspective, the attributed value is an average of all I/Os;
- $\mathcal{L}_\mathcal{C}$: stands for the insertion loss. For the same reason as above, the value used here is an average;
- $\mathcal{P}_\mathcal{C}$: holds the average power penalty of the component, and;
- $\mathcal{R}_\mathcal{C}$: is used for the components that behave as a switch or router. It represents the routing information of the component, which maps an input to an output.

This way, an OIN Υ can be defined as an n-uple $\langle \mathcal{N}, \mathcal{C}_o, \omega_j, \mathcal{J}, \mathcal{O} \rangle$, where:

- $\mathcal{N}$: is the number of inputs and outputs in the network. For instance, an 8×8 network has $\mathcal{N} = 8$;
- $\mathcal{O}$: number of components in the system;
- $\mathcal{C}_o$: is a list which comprises the optical components used to route light over the network;
- $\mathcal{J}$: is the number of waveguides in the system;
- ω_j: is a list of waveguides used to connect the internal network components.

Finally, in order to model a control unit, a function of four parameters is used, in which a controller κ can be defined as an n-uple $\langle \mathcal{N}_\kappa, \mathcal{S}, \mathcal{D}, \mathcal{A}_i \rangle$, where:

- $\mathcal{N}_\kappa$: is the number of communicating nodes using the network. For instance, in a system composed of four communicating processors, one DSP and one shared memory, $\mathcal{N}_\kappa = 6$;
- $\mathcal{S}$: defines the routing algorithm used to compute the messages path;
- $\mathcal{D}$: holds the delay, in clock cycles, imposed by the processing time of the routing algorithm and network tuning, and;
- $\mathcal{A}_i$: is a memory array composed of i elements used to store information related to the control like allocation tables for the dynamic network tuning.

4.3 Control Unit Design

The performance and efficiency of OIN-based architectures can be constrained by their controllers. Long setup time of circuit-switching techniques makes them not efficient for OINs, while the physical design and implementation of TDM and WDM techniques for OINs can become complex in practice. At the same time, controllers have been successfully demonstrated [22], thus showing the approach to be potentially used.

The design of the controller is based on independent blocks, which are graphically illustrated in Figure 4.2, in which it is possible to see four main blocks:

- the *matrix definition:* defines the network connections;
- the *paths creation:* holds the routing information of the controlled network. This block is used mainly to reduce computation time, thus reducing control latency;
- the *conflicts resolution block:* is responsible for detecting destination conflicts and solving them by using a given algorithm, and;
- the *real-time calculations:* is a block responsible for on-line calculations, like path attribution and memory addresses reading.

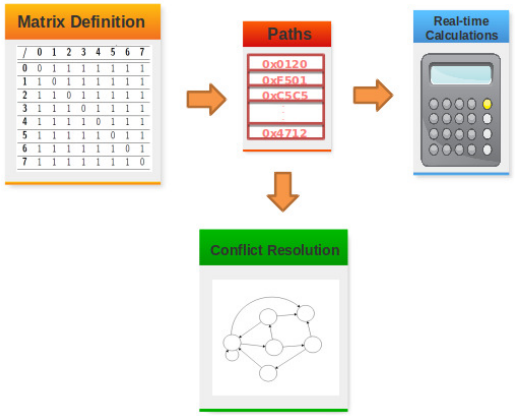


Figure 4.2 Controller design overview presenting the four main blocks of the modelled controller.

We start by defining the adjacency matrix for all the I/O ports in the network. It is also possible to consider methods to reduce/simplify the matrix [85], thus reducing the controller overhead. In particular, such reduction methods can be used to remove the redundant and/or unused network nodes in the matrix when not all the network nodes are connected to each

other (depending on the topology). Figure 4.3 presents two 8×8 network topologies, the SF and the Spidergon. While the SF topology has a direct connection path for all the I/Os, the Spidergon does not, as presented in the figure.

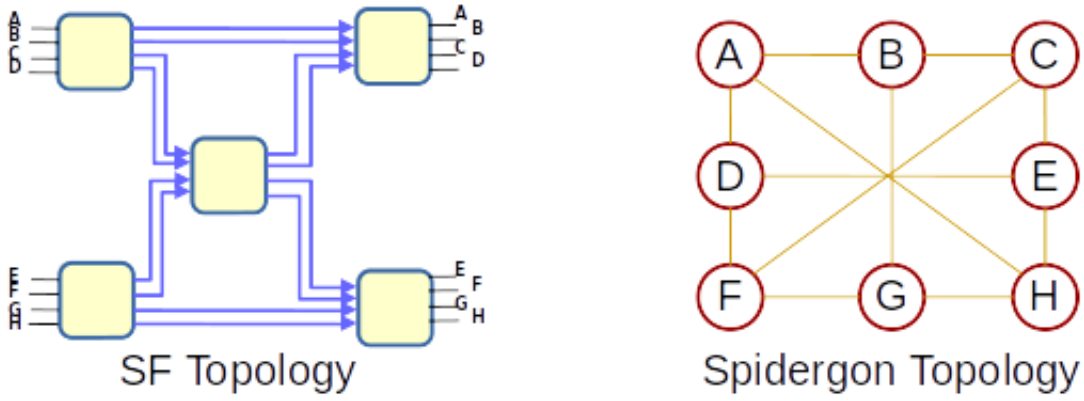


Figure 4.3 Controller design overview presenting the four main blocks of the modelled controller.

Table 4.1 indicates the adjacency matrices $\mathcal{M}$ for the two 8×8 networks presented in Figure 4.3. In the matrices, each paired index i,j represents a communication from input i to output j, such that:

$$\forall_{ij} \in \mathcal{M}, if \mathcal{M}_i \iff \mathcal{M}_j \Rightarrow \mathcal{M}_{ij} = 1, \tag{4.1}$$

resulting in the adjacency matrix $\mathcal{M}$.

Table 4.1 Adjacency matrix examples. The left matrix represents the SF 8×8 network, while the right matrix represents the Spidergon 8×8 network.

Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
A	1	1	1	1	1	1	1	1
B	1	1	1	1	1	1	1	1
C	1	1	1	1	1	1	1	1
D	1	1	1	1	1	1	1	1
E	1	1	1	1	1	1	1	1
F	1	1	1	1	1	1	1	1
G	1	1	1	1	1	1	1	1
H	1	1	1	1	1	1	1	1

Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
A	0	1	0	1	0	0	0	1
B	1	0	1	0	0	0	1	0
C	0	1	0	1	0	1	0	0
D	1	0	0	0	1	1	0	0
E	0	0	1	1	0	0	0	1
F	0	0	1	1	0	0	1	0
G	0	1	0	0	0	1	0	1
H	1	0	0	0	1	0	1	0

After defining the adjacency matrix, the second step is to compute all the input requests as fast as possible, as a high-latency controller impacts the efficiency of the entire network.

In order to minimize the controller overhead, we target one clock cycle latency, resulting in a latency within nanoseconds. Algorithm 1 describes the logic for the conflicts resolution and dynamic setup, determining the best available path for each input request. In the algorithm, one can notice different possible scenarios, such as when several input requests are received and conflicts are found. Summarizing, the algorithm works by checking all the input requests (*while* $i < numberOfInputs$) and looking for conflicts for each one of them ($checkConflicts(i)$). When no conflict is found and the target is not busy ($noOpenComm(i)$), the requesting node is granted access. If a conflict is found ($conflict()$), the controller checks to see if the next node in the Round-Robin queue[1] is the requesting node ($roundRobinControl()$), and if so, it grants the access. For the cases where the round robin queue is used, a new requesting node is set as the next one in the Round-Robin queue ($roundRobinSpin()$). Lastly, the controller defines the network configuration ($pathDefinition()$) by accessing the LUT. The concept of LUT will be covered later in this thesis.

Algorithm 1 Conflicts Resolution and Granting Control

```
while i < numberOfInputs do
    checkConflicts(i);
    if requestReceived(i) and noOpenComm(i) then
        if destIsAvail() and noDestConf() then
            acknowledge(i) ← 1;
            writeEnable(i) ← 1;
        else if destIsAvail() and conflict() then
            if roundRobinControl() then
                acknowledge(i) ← 1;
                writeEnable(i) ← 1;
            end if
            roundRobinSpin();
        end if
    else if endOfCommunication(i) then
        acknowledge(i) ← 0;
        writeEnable(i) ← 0;
        endOfComm(i) ← 1;
    else
        endOfComm(i) ← 0;
    end if
end while
pathDefinition();
```

Finally, the last step for the controller is to configure the path that the message should take

[1]The Round-Robin queue size is equal to the number of inputs of the network, avoiding queue overflow. The RR control works as classic Round-Robin implementation, where the queue works as a First-in-first-out queue. This way, all requesting inputs are guaranteed to be granted at some point in time, avoiding starvation.

from the source to the destination node. For every input request, more than one optical path might be selected, generating a huge number of path possibilities for the controller to compute. Figure 4.4 illustrates this situation. In the 2×2 switch, six different message patterns may be found: **(i) A** → **D**; **(ii) B** → **C**, and; **(iii) A** → **D** and **B** → **C**; **(iv) A** → **C**; **(v) B** → **D**, and; **(vi) A** → **C** and **B** → **D**. In the second scenario for the 4×4 switch, not only different message patterns are found, but also each pattern may follow different paths to reach its destination. For instance, for the message path **A** → **H**, the message may pass through nodes *1, 2, 5, 8, 21 and 24*. Alternatively, it may use nodes *1, 4, 17, 20, 23 and 24* to reach the same destination.

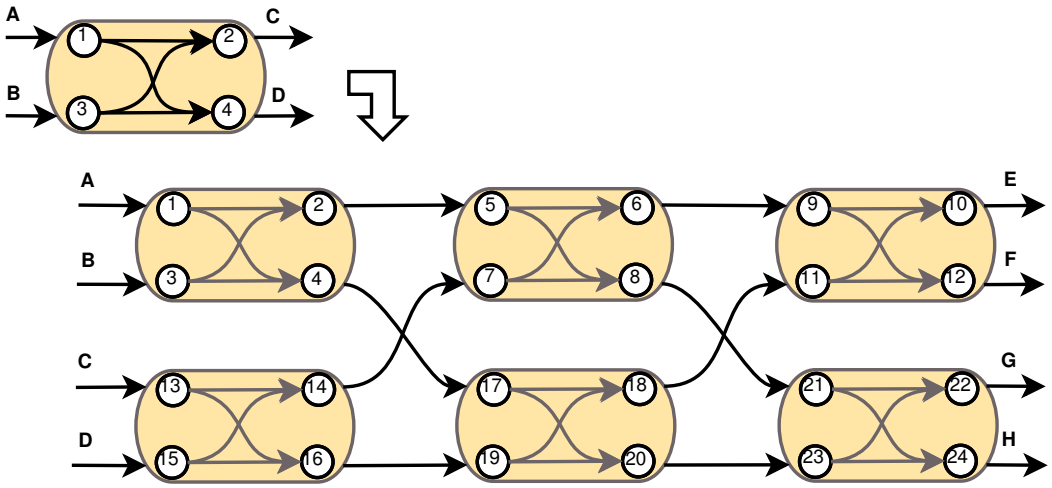


Figure 4.4 LUT growing size exemplification.

One possible approach to address this issue is to use a shortest path first (SPF) algorithm, like the Dijkstra algorithm [86]. The SPF algorithm looks for the minimal distance between the source and the destination in a network. The distance between the source and destination is considered as the number of network nodes the message should pass by. It works by considering the value of each edge that connects any two nodes, and then looking for the minimal sum of those edge values between the source and the destination. The edge weight can be defined as any measure inside the network context. For example, the distance between nodes or the power consumption. It depends on the routing technique objective and is defined by the designer. If power consumption is to be considered, each edge might express the power dissipated between each nodes pair, for instance. As it would be very complex to calculate a new path for every new input request, a pre-calculated path allocation lookup table is generated and stored in memory (i.e., static memory) by implementing the SPF algorithm and running it during the design time.

Another possibility, besides the usage of an SPF algorithm, is the employment of distributed

configuration units. Learning from circuit-switching-based techniques, it is possible to deploy a similar approach, which is scalable and maintains low latency. By keeping all other blocks centralized and distributing only the configuration units, it is possible to reduce the block complexity, accelerating its execution time and thus not compromising the system latency. This is achieved by reducing the distributed blocks complexity, thus speeding up its execution time.

For the routing block(s), we represent the network as a graph. The OIN architecture in our work consists of several optical switches integrated to form the network. As a result, the graph representation starts from MZI-based optical switches, for example. Figure 4.5 presents the graph abstraction of a 2×2 MZI-based optical switch, where inputs and outputs are represented as a graph node and connections are represented as edges.

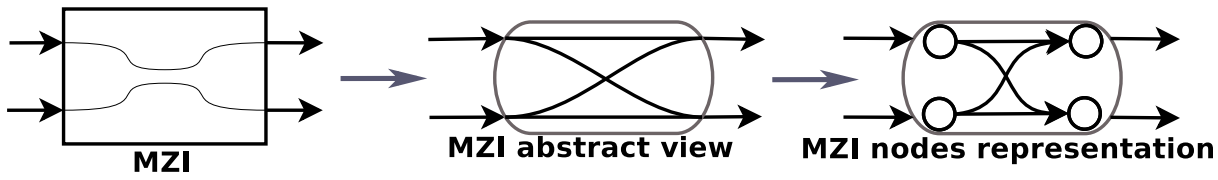


Figure 4.5 Graph representation of a 2×2 MZI-based optical switch.

Employing several 2×2 switching blocks, it is possible to form a 4×4 optical switch, shown in Figure 4.6. As can be seen, six 2×2 optical switches are interconnected, forming an optical switch with four inputs and four outputs. The presented topology is the non-blocking 4×4 Beneš topology. Non-blocking topologies are designed in such a way that for any combination of input outputs, a route is possible.

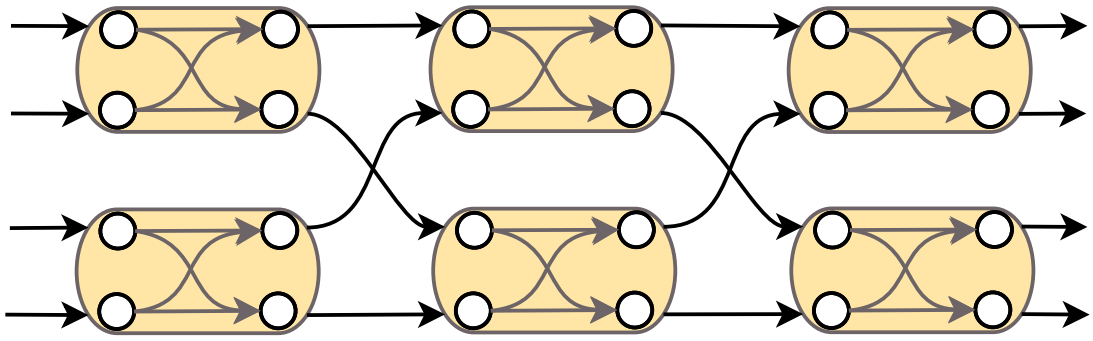


Figure 4.6 Graph representation of a 4×4 MZI-based Beneš optical switch.

Another possible 4×4 optical switch based on the 2×2 switching blocks is the 4×4 Spanke-Beneš topology. On opposite to the presented 4×4 Beneš switch, the Spanke-Beneš is a

blocking structure, which means that not all input-output combinations are possible simultaneously. Figure 4.7 illustrates the internal organization of the 4×4 Spanke-Beneš switch, where the five internal 2×2 switching blocks and their connections are shown.

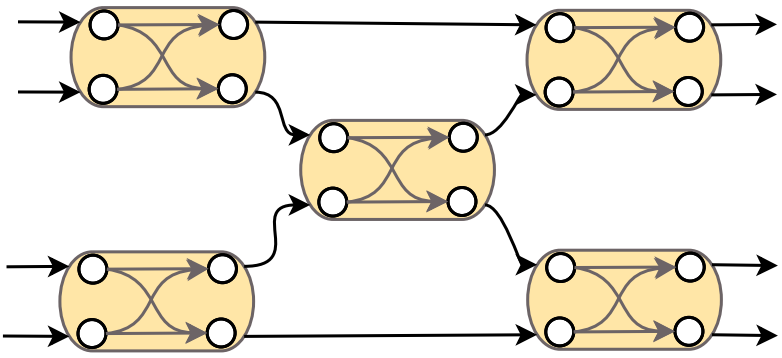


Figure 4.7 Internal architecture of a 4×4 MZI-based Spanke-Beneš optical switch.

Besides non-blocking and blocking topologies, a third possibility is the strictly non-blocking topology. While non-blocking topologies have possible routes for any inputs-outputs combinations, they sometimes require the re-routing of already open paths. Strictly non-blocking topologies present routes for any input-output combination without the need for re-routing. Figure 4.8 presents a 4×4 strictly non-blocking topology, 16 2×2 switching blocks and their connections.

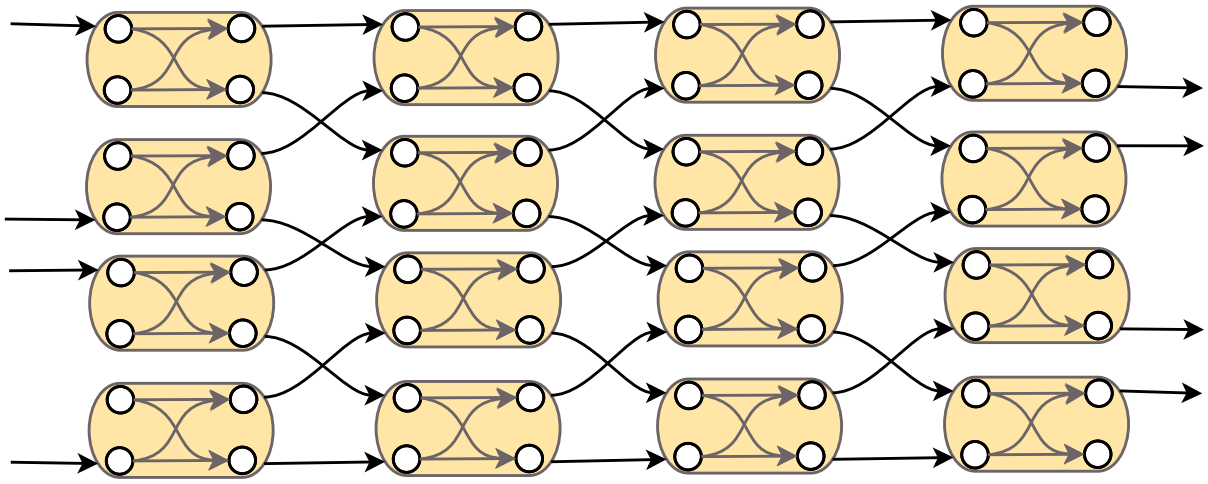


Figure 4.8 Internal architecture of a 4×4 MZI-based strictly non-blocking optical switch.

At this point, the resulting graph is already complex to compute during runtime, as not only the number of possible source-target path combinations is already big but the internal switch organization might change. Therefore, the possible hardware implementation,

although feasible, would be costly.

Employing the 2×2 and 4×4 optical switches and connecting them accordingly, one can construct different OINs. Figure 4.9 shows the graph representation of one possible OIN topology, the 8×8 Beneš network, in which eight 2×2 and two 4×4 optical switches are employed. As can be seen, the complexity increases with the addition of extra nodes and I/O ports, justifying the choice for a pre-computed path lookup table. The path allocation lookup tables have to be generated for every new topology. This is due to the fact that for different topologies various possible paths can be found. It is worth mentioning that this computation can be performed once and it is off-line, reducing the final design complexity.

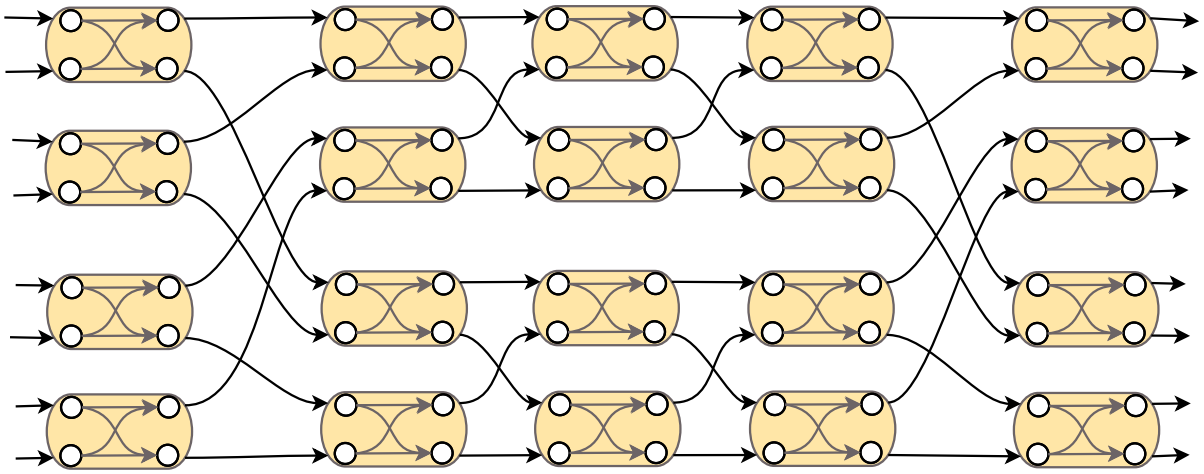


Figure 4.9 Internal architecture of an 8×8 Beneš network based on employing several 2×2 and 4×4 optical switches.

4.4 Validation Architectures

To validate the developed models and controllers, we described the basic network components, like MR-based and MZI-based switches and entire networks, in VHDL. The results are evaluated by comparing results extracted from fabricated devices with simulation results.

4.4.1 Switches

Firstly, optical-based switches are described based on the modelling strategy introduced in Section 4.2, with internal behaviours based on MZIs and MRs blocks. The timing and functional characteristics of such devices are considered, where values such as the network transmission rate and dissipated power are obtained from simulations.

To do so, basic building blocks are defined and then replicated to build bigger systems. Figure 4.10 presents the architectural structure of a 2×2 switch. This device routes two inputs to two different outputs and might be designed as active or passive. This is the building block and different topologies are built using this block. Also, this is the starting validation point, where all models are employed for the first time.

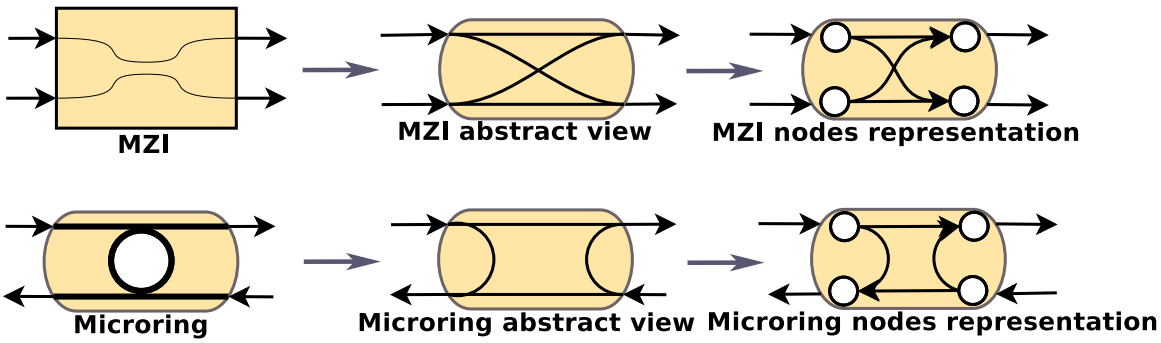
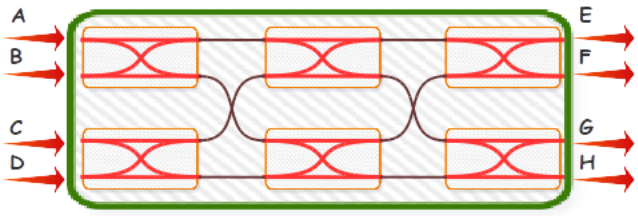
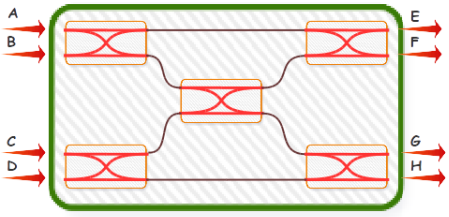


Figure 4.10 2×2 validation switch block, based on MZIs and MRs.

Figure 4.11 shows the second validation system, based on 4×4 switches. Figure 4.11.(a) presents a 4x4 Beneš switch, while Figure 4.11.(b) illustrates a 4x4 Spanke-Beneš switch. We selected these blocks thanks to the fact that they are already widely employed and have a rich gamma of results available in literature. Further, prototyped devices such as the ones presented are available for in-lab validations, enhancing obtained results.



(a) 4x4 MZI-Based Beneš Switch



(b) 4x4 MZI-Based Spanke-Beneš Switch

Figure 4.11 4×4 validation topologies.

Figure 4.12 shows a 5×5 MR-based strictly non-blocking switch [87]. The figure illustrates the router internal organization where it is possible to see the 16 micro-ring resonators (MRs), six waveguides, and two waveguide terminators. The MRs in the switching fabric are identical, and have the same on-state and off-state resonance wavelengths.

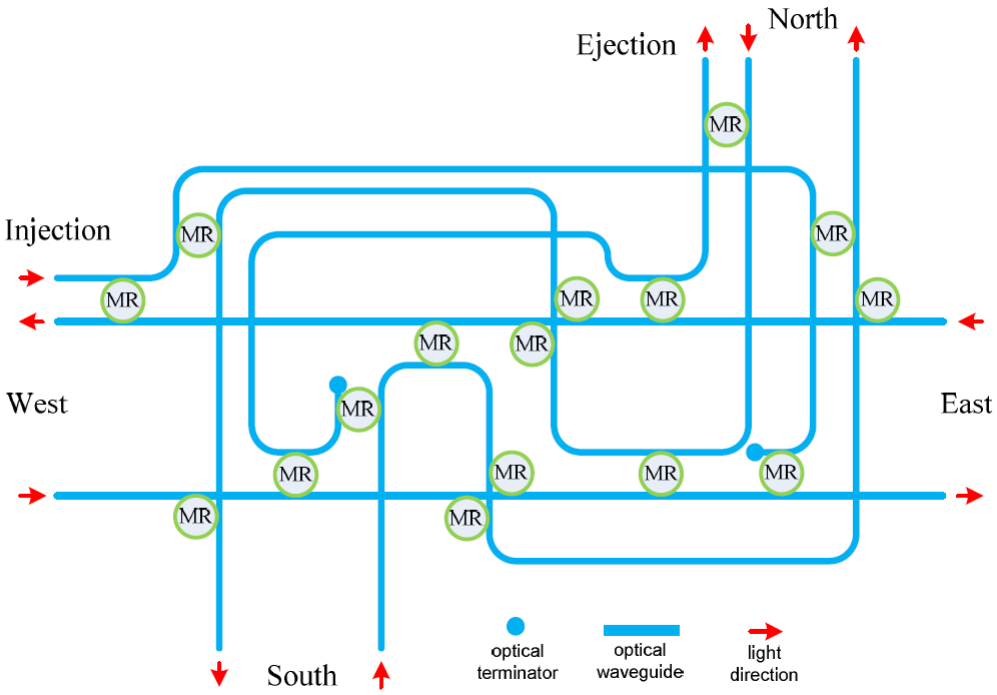


Figure 4.12 5×5 MR-based strictly non-blocking switch [87].

All network topologies used in this thesis are based on the introduced switching blocks. The topologies are illustrated using the switches abstracted view or nodes representation view.

4.4.2 Topologies

Different network topologies are described. Network topologies such as the 8×8 Beneš (eight I/O ports and 20 switches) network [88], a strictly non-blocking 32×32 topology (32 I/O ports and 1024 switches) [89] and a MR-based mesh topology [90] are used in order to measure the models accuracy in different scenarios.

In addition to describing already known topologies, we propose a new topology, called StraightForward (SF), is used for comparison. The SF topology is presented in Figure 4.13. This network is defined to be simple, with no waveguide crossings between blocks. Still in the same Figure, it is possible to see, once again, that only basic 4×4 switch blocks are used. When compared to the 8×8 Beneš, the 8×8 SF presents a better path availability, as presented in Section 5.1, which means that more parallel communications can be performed. Also, for any configure path in the network, the number of nodes will be the same.

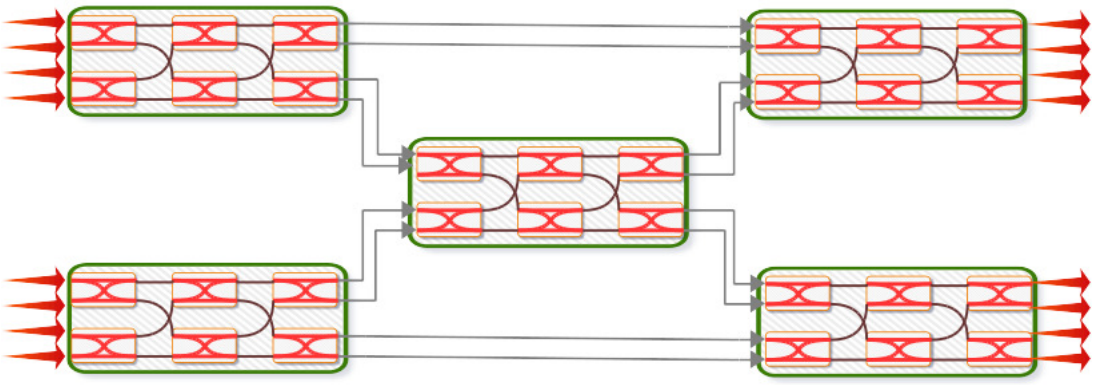


Figure 4.13 8x8 SF Network

At this point, it is important to highlight that the majority of the validation architectures are based on MZIs, but are not limited to them. This is due the fact that prototyped MZI-based switches are available, thus making them more suitable for models validation. Nevertheless, topologies based on MRs are also used and have their published results used [87][65].

CHAPTER 5 CONTROL UNIT - THE LUCC

In OIN based systems, the employment of optical technology provides a fast communication path. From a network perspective, where the network is composed of a control layer (responsible for the access control and network configuration) and a transmission layer, the controller is the slowest component. Due to this fact, its latency determines the latency of the entire system. So, further speed-up in OINs can be achieved through the improvement of the controller. Therefore, to take advantages of the OIN characteristics, it is important to find new solutions enabling the acceleration of the control unit in OIN based systems.

Centralized controllers have been successfully demonstrated [22], thus making this solution newsworthy to be exploited. We propose a new control unit which relies in pre-calculated routes stored on fast access memories, such as Look-up Tables (LUTs). The routes are generated by the Shortest Path First algorithm, so the smallest number of hops can be used. As the control unit uses Look-up Tables (LUT), the developed controller is named LUCC (LUT-based Centralized Controller).

The design of LUCC consists of four different parts: the adjacency matrix definition, the conflict resolution, the path configuration, and the runtime calculation (i.e., dynamic setup). The matrix definition is a technique used to describe all the network connections, facilitating applying different methods for possible simplifications (e.g., the matrix reduction method [85]). The conflict resolution is responsible for detecting same-destination conflicts (i.e., when several IPs simultaneously request to communicate with the same IP core), and solving them using a given algorithm. The path configurations is a memory block used to store static data accessed by the controller during the runtime, which can be built using the output from the matrix definition. This memory is used mainly to reduce computation time, thus reducing the overall control latency. The runtime calculations is a block responsible for real-time calculations (e.g., computations related to path assignments and readings of memory addresses). Figure 5.1 illustrates the designed controller overview. The matrix definition and the path configuration are performed at design time and result in the LUT which holds the network paths, while the conflict resolution and real-time calculations are performed at runtime.

To minimize the time overhead, the iSLIP algorithm [91] is used as an inspiration for the LUCC. This algorithm has been proven to work efficiently for small to medium size networks and it works with *'n'* iterations, where each iteration can be divided into three steps:

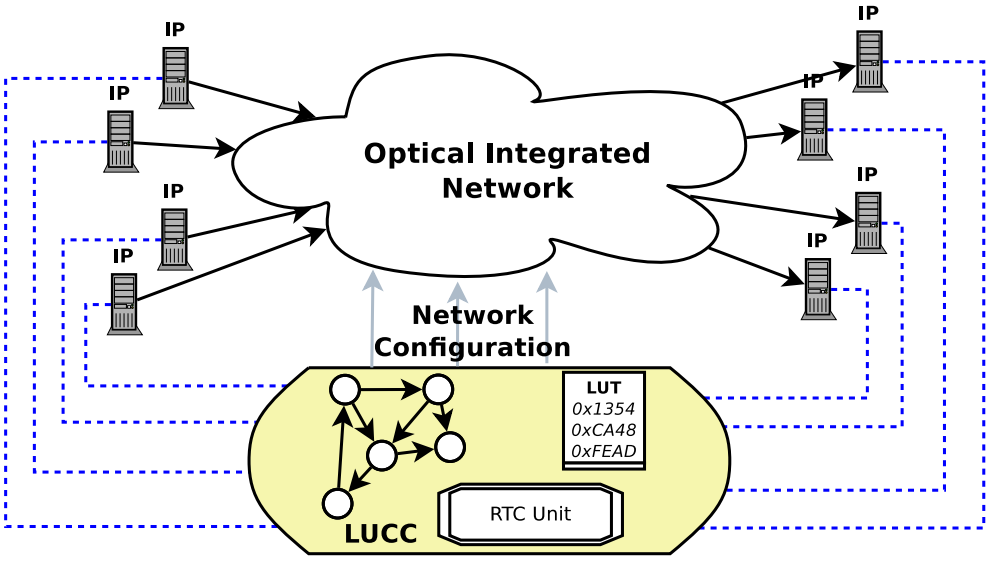


Figure 5.1 LUCC design overview.

- **Request**: all requesting input ports signal their intention to have access to the network, which are stored in virtual queues, one for each port;
- **Grant**: each output port that has received a request must determine which input ports may be granted to send. For the situations where only one input port is requesting access, this step is straightforward. For the cases where two or more requests are targeting the same output, a Round-Robin (RR) algorithm is used to decide which will have access granted, and;
- **Accept**: the input ports consider all grant messages and choose from them in a Round-Robin fashion. When selecting, the input notifies the targeted output that the access is granted and only after receiving the accept signal it can move to the next request.

The iSLIP runs through multiple iterations. As a result, the matches number of input-to-output is improved and studies showed that running the iSLIP beyond four iterations does not yield significantly greater matches [91]. Figure 5.2 illustrates the relation between the number of iterations and the obtained network throughput under different traffic patterns. The network throughput does not increase after two iterations [92].

Similarly, the implemented control core relies on three steps with the difference that the LUCC does not perform recursive iterations nor utilizes virtual queues on the input ports[1].

[1]As the LUCC works in a request-based approach, there is no need for virtual queues in the input. This way, each requesting input port signals its desire to access the network using one bit (e.g., *request* signal) and the LUCC signals the input port may start transmitting using an acknowledge signal (e.g., *ack* signal).

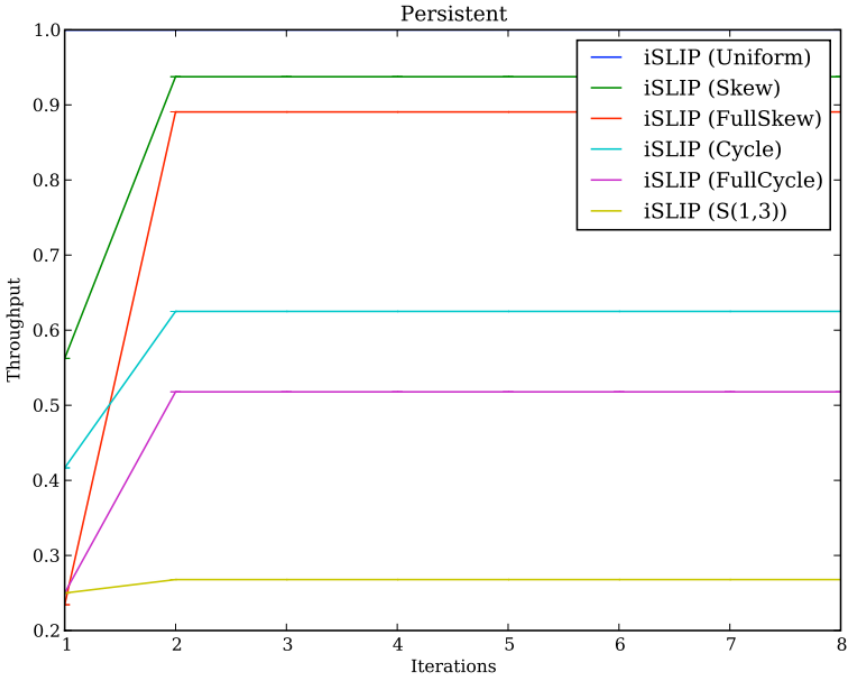


Figure 5.2 iSLIP throughput X iterations number [92].

The reason for that is to reduce the complexity of the implemented core focusing on a fast and efficient conflict solution. A possible drawback in removing such features from the original iSLIP implementation lies in the fact that the recursive iterations are designed to improve throughput. This way, for some specific scenarios, the LUCC throughput might be inferior when being compared with the original iSLIP implementation.

Figure 5.3 presents the decision chart of the controller, where the LUCC execution flow is depicted into single decision steps. Mostly, the overall behaviour of the conflicts resolution block and the path attribution are illustrated. The flow chart represents all steps taken by the LUCC within one clock cycle. The one clock cycle latency is obtained by the combination of the fast access LUT and the optimized conflicts resolution blocks, presented hereafter. The execution flow works as follows. The controller expects a request, and by the time it receives one it checks if the targeted destination is available (not busy, in the middle of another transmission or somehow not able to receive a message). Next, the LUCC checks if there is no destination conflicts: if a conflict is found, the round robin algorithm determines which requesting node should have its access granted. Following, the LUT is accessed in order for the network to be configured and the granting signal is sent to the requesting node. Finally, the LUCC waits for the signal to point the end of communication, sent by the transmitting

port and by the time it receives the signal, the connection is closed.

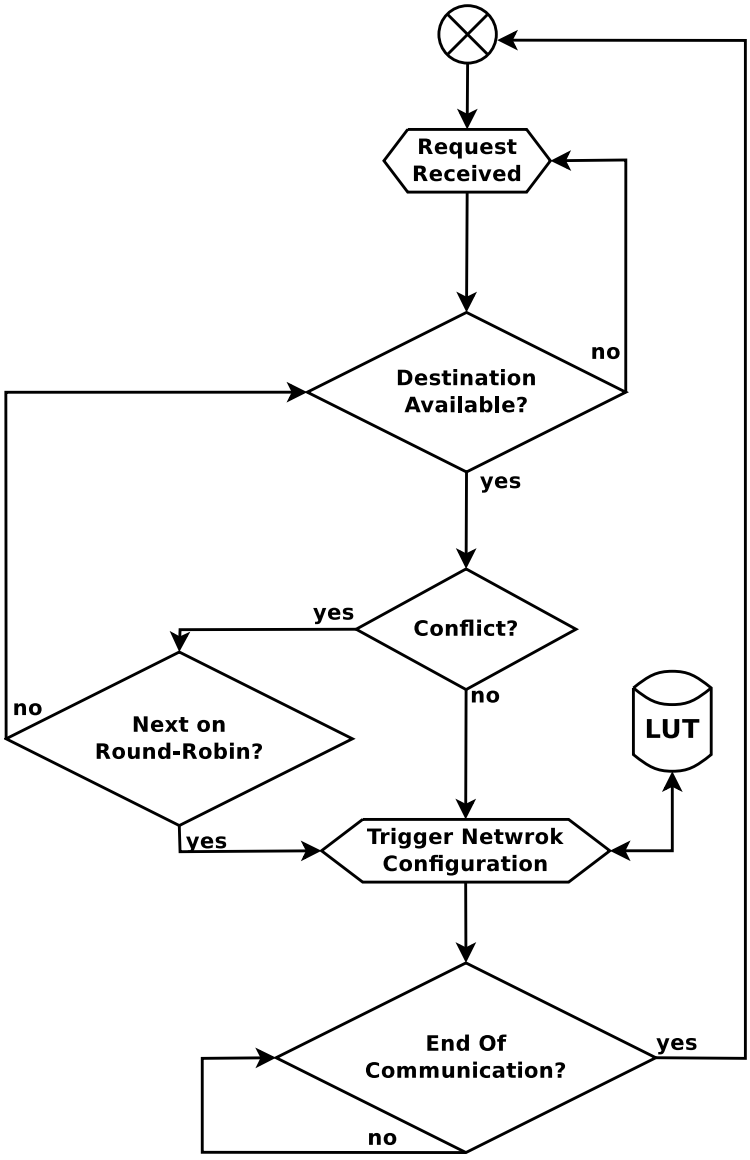


Figure 5.3 LUCC decision flow chart.

5.1 Path Analyzer and LUT Creation - The PALC

Besides developing the LUCC, a path analyzer and a table generator framework were implemented. This tool was created to ease the early stage evaluation of topologies, even before their modelling. To use the tool, the designer must describe the topology as a graph, i.e. Figure 5.4, and use this graph as the entry point. Starting from the topology graph, all possible communication combinations and all possible paths are generated. The worst, best

and average cases are also analyzed. The best cases stand for those cases where all requests of an input set are satisfied. Average cases are for those cases where a portion of the request combinations are satisfied, while the worst cases are for the worst obtained routing for one input set. Finally, the tool generates the memory array used by the controller, which stores the pre-calculated paths.

Listing 5.1 presents the textual entry for the tool, where the Graph view of the 8×8 Beneš network, illustrated in Figure 5.4, is described and three distinct groups are found:

1. The number of connections in the network are the number of connections, related to the internal graph nodes and their connections and not the system nodes, like IPs;

2. The network nodes connections, where the internal nodes and their weighted edges are shown. The connections are presented in trios - origin, destination and weight of each connection. As it is possible to see in Figure 5.4, each node is assigned with an ID. These IDs are used as index to represent the origin node and destination node, for each connected pair. The weight of each connection can represent different aspects of the connection: the distance between nodes or the consumed power to communicate between connected nodes. For example, the power consumption for the BAR and CROSS-BAR states of the MZI have different values. The weight can represent these consumption in such a way as the tool takes into account these variations when performing its computation, and;

3. The I/O mapping, where the input and output network points, i.e. the IPs, are mapped to one specific node in the topology. For instance, in Figure 5.4, one IP is mapped to input network node 57 and output network node 74. This means that, this specific IP will insert its data into the network through node 57, and any other IP which targets it will address their messages to node 74.

Although this file was generated manually, some third-party tool might be used, like [93], in order to accelerate the description.

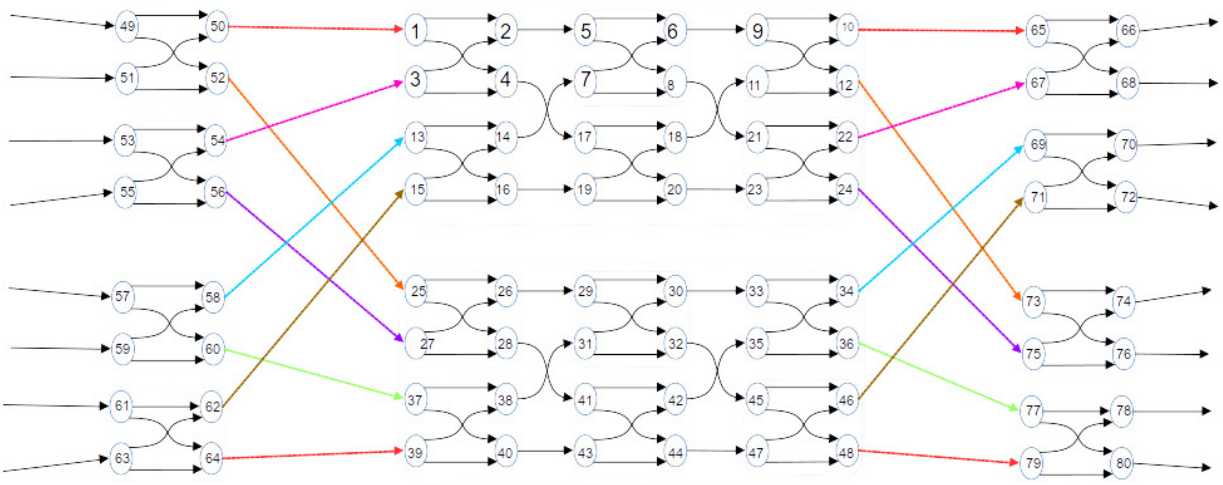


Figure 5.4 8×8 Beneš topology numbered graph representation.

Listing 5.1 8×8 Beneš Network description as a graph

```
1  --Number of Connections
2  112
3  --Nodes Connections
4  1 2 1    1 4 1    3 2 1    3 4 1    5 6 1    5 8 1    7 6 1    7 8 1    9 10 1   9 12 1
5  11 10 1  11 12 1  13 14 1  13 16 1  15 14 1  15 16 1  17 18 1  17 20 1  19 18 1  19 20 1
6  21 22 1  21 24 1  23 22 1  23 24 1  2 5 1    4 17 1   6 9 1    8 21 1   14 7 1   16 19 1
7  18 11 1  20 23 1  10 65 1  12 73 1  22 67 1  24 75 1  25 26 1  25 28 1  27 26 1  27 28 1
8  29 30 1  29 32 1  31 30 1  31 32 1  33 34 1  33 36 1  35 34 1  35 36 1  37 38 1  37 40 1
9  39 38 1  39 40 1  41 42 1  41 44 1  43 42 1  43 44 1  45 46 1  45 48 1  47 46 1  47 48 1
10 26 29 1  28 41 1  30 33 1  32 45 1  38 31 1  40 43 1  42 35 1  44 47 1  34 69 1  36 77 1
11 46 71 1  48 79 1  49 50 1  49 52 1  51 50 1  51 52 1  50 1 1   52 25 1  53 54 1  53 56 1
12 55 54 1  55 56 1  54 3 1   56 27 1  57 58 1  57 60 1  59 58 1  59 60 1  58 13 1  60 37 1
13 61 62 1  61 64 1  63 62 1  63 64 1  62 15 1  64 39 1  65 66 1  65 68 1  67 66 1  67 68 1
14 69 70 1  69 72 1  71 70 1  71 72 1  73 74 1  73 76 1  75 74 1  75 76 1  77 78 1  77 80 1
15 79 78 1  79 80 1
16 --I/O Mapping
17 49 66
18 51 68
19 53 70
20 55 72
21 57 74
22 59 76
23 61 78
24 63 80
```

The implemented SPF algorithm is Dijkstra [86] and the tool output, after processing the given topology and analyzing the data patterns is presented in Listing 5.2. The performed analysis is analytical, and does not consider traffic patterns and wavelength division. It is important to highlight that provided information is an early stage evaluation of topologies, obtained from the analytical analysis of possible paths in the network, considering only the

network routes and contention. In summary, the following information is provided:

- **# RUNS**: presents the number of different combinations used to feed the network. For example, the connection between the first network input and the second network output is one combination. The connection between the first network input and the second network output at the same time as the connection between the third network input and the fourth network output is another combination;
- **# DIFFERENT INPUTS USED**: shows the number of I/O ports in the analyzed network;
- **# ROUTING CASES**: presents the percentage of successful routing for a given number of inputs. So, for example, line 14 shows that, on average, when two distinct inputs are trying to reach two different outputs, the topology is able to provide a path at 90.39% of the time. For the remaining 9.61% of cases, it is not possible to have both connections in parallel. Another example is in line 20, where eight inputs are trying to reach eight outputs (all-to-all). In this case, in average the Beneš topology is able to provide a path to satisfy the eight requesting inputs at the same time at 55.39% of the time. This category is subdivided in three extra categories:
  - **BEST CASES**: as the name suggests, shows the best cases for all groups of inputs number, one-by-one;
  - **AVERAGES**: presents the mean of the success rate for closing connections in parallel, for all inputs number, and;
  - **WORST CASES**: presents the worst cases obtained by the given topology for all the inputs.
- **LAST WORST CASE**: is the last worst case mapping configuration checked, and;
- **LAST RUN**: is the last tried mapping configuration.

Listing 5.2 8×8 Beneš summary

```
693838 RUNS - 8 DIFFERENT INPUTS USED
        * BEST CASES NODES ROUTING:
                ** 1 INPUT(S) - 1 ROUTED (100.00 %)
                ** 2 INPUT(S) - 2 ROUTED (100.00 %)
                ** 3 INPUT(S) - 3 ROUTED (100.00 %)
                ** 4 INPUT(S) - 4 ROUTED (100.00 %)
                ** 5 INPUT(S) - 5 ROUTED (100.00 %)
                ** 6 INPUT(S) - 6 ROUTED (100.00 %)
                ** 7 INPUT(S) - 7 ROUTED (100.00 %)
                ** 8 INPUT(S) - 8 ROUTED (100.00 %)
        * AVERAGES NODES ROUTING:
                ** 1 INPUT(S) - 1.00 ROUTED (100.00 %)
                ** 2 INPUT(S) - 1.81 ROUTED (90.39 %)
                ** 3 INPUT(S) - 2.70 ROUTED (89.94 %)
                ** 4 INPUT(S) - 2.39 ROUTED (59.69 %)
                ** 5 INPUT(S) - 4.80 ROUTED (95.92 %)
                ** 6 INPUT(S) - 3.44 ROUTED (57.26 %)
                ** 7 INPUT(S) - 6.89 ROUTED (98.44 %)
                ** 8 INPUT(S) - 4.48 ROUTED (55.96 %)
        * WORST CASES NODES ROUTING:
                ** 1 INPUT(S) - 1 ROUTED (100.00 %)
                ** 2 INPUT(S) - 1 ROUTED (50.00 %)
                ** 3 INPUT(S) - 2 ROUTED (66.67 %)
                ** 4 INPUT(S) - 2 ROUTED (50.00 %)
                ** 5 INPUT(S) - 3 ROUTED (60.00 %)
                ** 6 INPUT(S) - 3 ROUTED (50.00 %)
                ** 7 INPUT(S) - 4 ROUTED (57.14 %)
                ** 8 INPUT(S) - 4 ROUTED (50.00 %)
        * LAST WORST CASE RUN NODES:
          || 0 -> 7 || 1 -> 6 || 2 -> 5 || 3 -> 4 || 4 -> 3 || 5 -> 2 || 6 -> 1 || 7 -> 0 ||
        * LAST RUN TRIAL NODES:
          || 0 -> 7 || 1 -> 6 || 2 -> 5 || 3 -> 4 || 4 -> 3 || 5 -> 2 || 6 -> 1 || 7 -> 0 ||
```

As it is possible to see, the provided summary helps designers obtain an initial feedback on a given network topology, even before its deployment, without any need for describing, simulating or testing it, but simply by analytically evaluating it. For comparison matters of practical employment of the tool, the same procedure used to analyze the 8×8 Beneš Network was performed for the 8×8 SF Network, which was modelled as a graph, its textual description was inserted into the analyzing tool and its summary was collected. The Listing 5.3 illustrates the obtained information, which shows that the 8×8 SF Topology presents a better path availability than the 8×8 Beneš Topology, as both average and worst-case scenarios presented a higher parallel path availability. This is thanks to the fact that the SF topology presents more internal connections, when compared to Beneš. As presented in figures 5.4 and 5.5, the Beneš is composed of two 4×4 switches and eight 2×2 switches, while the SF is composed of five 4×4 switches.

Listing 5.3 8×8 SF Summary

```
--SUMMARY--
693838 RUNS - 8 DIFFERENT INPUTS USED
        * BEST CASES NODES ROUTING:
                ** 1 INPUT(S) - 1 ROUTED (100.00 %)
                ** 2 INPUT(S) - 2 ROUTED (100.00 %)
                ** 3 INPUT(S) - 3 ROUTED (100.00 %)
                ** 4 INPUT(S) - 4 ROUTED (100.00 %)
                ** 5 INPUT(S) - 5 ROUTED (100.00 %)
                ** 6 INPUT(S) - 6 ROUTED (100.00 %)
                ** 7 INPUT(S) - 7 ROUTED (100.00 %)
                ** 8 INPUT(S) - 8 ROUTED (100.00 %)
        * AVERAGES NODES ROUTING:
                ** 1 INPUT(S) - 1.00 ROUTED (100.00 %)
                ** 2 INPUT(S) - 2.00 ROUTED (100.00 %)
                ** 3 INPUT(S) - 2.24 ROUTED (74.61 %)
                ** 4 INPUT(S) - 2.48 ROUTED (62.11 %)
                ** 5 INPUT(S) - 4.03 ROUTED (80.58 %)
                ** 6 INPUT(S) - 6.00 ROUTED (100.00 %)
                ** 7 INPUT(S) - 7.00 ROUTED (100.00 %)
                ** 8 INPUT(S) - 8.00 ROUTED (100.00 %)
        * WORST CASES NODES ROUTING:
                ** 1 INPUT(S) - 1 ROUTED (100.00 %)
                ** 2 INPUT(S) - 2 ROUTED (100.00 %)
                ** 3 INPUT(S) - 2 ROUTED (66.67 %)
                ** 4 INPUT(S) - 2 ROUTED (50.00 %)
                ** 5 INPUT(S) - 3 ROUTED (60.00 %)
                ** 6 INPUT(S) - 4 ROUTED (66.67 %)
                ** 7 INPUT(S) - 4 ROUTED (57.14 %)
                ** 8 INPUT(S) - 4 ROUTED (50.00 %)
        * LAST WORST CASE RUN NODES:
          || 0 -> 3 || 1 -> 2 || 2 -> 1 || 3 -> 0 || 4 -> 7 || 5 -> 6 || 6 -> 5 || 7 -> 4 ||
        * LAST RUN TRIAL NODES:
          || 0 -> 7 || 1 -> 6 || 2 -> 5 || 3 -> 4 || 4 -> 3 || 5 -> 2 || 6 -> 1 || 7 -> 0 ||
```

Besides analyzing the topologies, the proposed tool also generates the memory arrays to be used by the LUCC. The created memory array is a static table structure and, after its creation, it will never be changed by the LUCC. This way, even though its generation processing time and complexity is costly (it is a NP-complete problem [94]) the benefits of having this structure during the runtime pays off the effort. This is thanks to the fact that the time it takes to access the memory array is smaller than the time it would take to calculate the entire route for every request.

5.2 Conflict Resolution Block - The CRB

The conflict resolution block (CRB) is a piece of hardware responsible for detecting and solving conflicting points in the targeted IPs, where a conflicting point is defined as any situation in which two or more inputs are requesting the same output. It works by analyzing

all LUCC's received requests and comparing them among each other to check for disputes.

The CRB works in two steps: first, it analyzes all requests, looking for a conflict, and; second, if a conflict is found, a Round-Robin algorithm is applied to define which IP will have its accessed granted. Algorithm 2 presents the logical behaviour of the CRB block, showing the pseudo-code extracted from the implemented design.

Algorithm 2 CRB Logic Implementation

```
i ← N_κ                                   ▷ number of I/Os in the network
j ← N_κ                                   ▷ number of I/Os in the network
for all i do
    for all j do
        if conflict(i) = 0 then
            if req(j) ≠ req(i) then
                conflict(i) ← 0
            else if req(j) = req(i) then
                conflict(i) ← 1
            end if
        end if
    end for
end for
for all i do
    if conflict(i) = 1 then
        for all j do
            while roundDone ≠ 1 do
                if RoundRobinChooser() = j then
                    roundDone ← 1
                    granted(j) ← i
                else
                    roundDone ← 0
                end if
            end while
        end for
    end if
end for
```

The conflict resolution block works by analyzing all input and output pairs and then finding the ones where two or more inputs are requesting the same output. This block might work either for the cases where the receiver accepts only one incoming message at a time or when all possible wavelengths are being used, thus not leaving any wavelength available for the new request.

Following, the Round-Robin (RR) algorithm implements a first-in-first-out (FIFO) queue to

decide which port will have its access granted when a conflicting situation is found. As the RR uses a FIFO for each input, each request is treated individually as one particular process, which is triggered by the conflict bit. Lastly, the control algorithm implements the LUCC, by processing the received requests, where each request port has its own running process.

5.3 Dynamic Setup Block

The Dynamic Setup Block (DSB) is implemented to alleviate the design physical implementation complexity, as only the usage of the CRB and the LUT would imply a prohibitive resource usage to efficiently apply the LUCC. If only the CRB and LUT blocks were used, a big (physically unbearable) chip area would be needed, as the memory array would get bigger as the number of paths combinations rise. Table 5.1 presents the LUT size for an I/O number up to 10, where it is possible to see that, even for a small number of I/Os, the LUT size gets big. The number of combinations is the same no matter what is the network topology. This is thanks to the fact that the number of combinations is only related to the input-output combinations. What changes accordingly to the topology is the LUT information, which holds the specific paths for each input-output combination. Also, the reduction method introduced later is also topology dependent, as the network organization reflects in the degree of possible LUT optimization.

Table 5.1 LUT Growing Size.

I/O	Table Entries
2	**6**
3	**16**
4	**106**
5	**778**
6	**6598**
7	**63838**
8	**693838**
9	**8361358**
10	**110557438**

Consequently, a practical implementation relying only on the LUT is not feasible. The DSB works as a real-time calculation unit on the LUCC. Acknowledging the fast access time of LUTs and the ease of their usage, a reduced version of the final LUT is created, as a LITE version. The LITE-LUT stores only small portions of the network info, such as the most critical paths or the paths of one of the basic 4×4 switches used as construction blocks and the DSB performs the remaining calculations necessary to choose the ideal path. The path

allocation generation process is presented in the Algorithm 3.

Algorithm 3 LUT Generation and Reduction

```
inputsCombinations();
for all inputs do
    spf_algorithm();          ▷ executes the SPF algorithm for each combination set
end for
allocOptmization();           ▷ exclude duplicated table entries / realloc values
while allocSize ≥ threshold do[2]
    allocReduction();
end while
```

Considering Figure 5.5 as an example, which presents the SF topology, it is possible to see one of the LITE-LUT's reduction approaches. In the original LUT size for this network, all possible combinations and paths were calculated and stored, resulting in a LUT's size to be equal to 693838 table entries, as eight I/Os are used. The network is composed of five 4×4 identical switches. This makes it possible to generate the LUT for only one of the switches and then replicate the information for all remaining 4×4 switches. Thus, on the LITE-LUT version only the paths for one 4×4 switch is stored, and the complete path is calculated on the fly, as a composition of the information for one 4×4 switch by the DSB. In this case, the LUT size dropped from 693838 to 106 table entries.

Different topologies lead to different reduction effectiveness. For instance, aforementioned 8×8 SF topology, the LUT size dropped from 693838 to 106 table entries. This is thanks to the fact that the SF topology is composed of similar 4×4, making it possible to reduce at most the LUT, resulting in a small LITE-LUT. The 8×8 Beneš network has a different internal organization, being composed of two 4×4 switches and eight 2×2, leading to a bigger LITE-LUT. This way, for the 8×8 Beneš network, not only the 106 entries for the 4×4 switch is needed, but also the information of the 2×2 switches. This results in a LITE-LUT size of 154 entries.

The DSB works using the information stored in the LITE-LUT. For example, using the 8×8 SF network from Figure 5.5 and assuming that the LITE-LUT stores the path for one 4×4 switch, used in the 8×8 construction. The DSB receives the request for a given path

[2]In this context, a threshold size might be defined as any limitation, or constraint, on the final memory size. For example, memory utilization on FPGAs can be defined as the upper threshold size. Using the 4×4 Beneš switch and the 2×2 switching block it is possible to make a comparison. While the LUT synthesis for a given FPGA for the 2×2 switching block results in nine flip-flops, the same process for the 4×4 Beneš results in 27 flip-flops. It is important to highlight that the synthesis process performs optimizations, thus the original LUT size is not directly related to the number of used flip-flops.

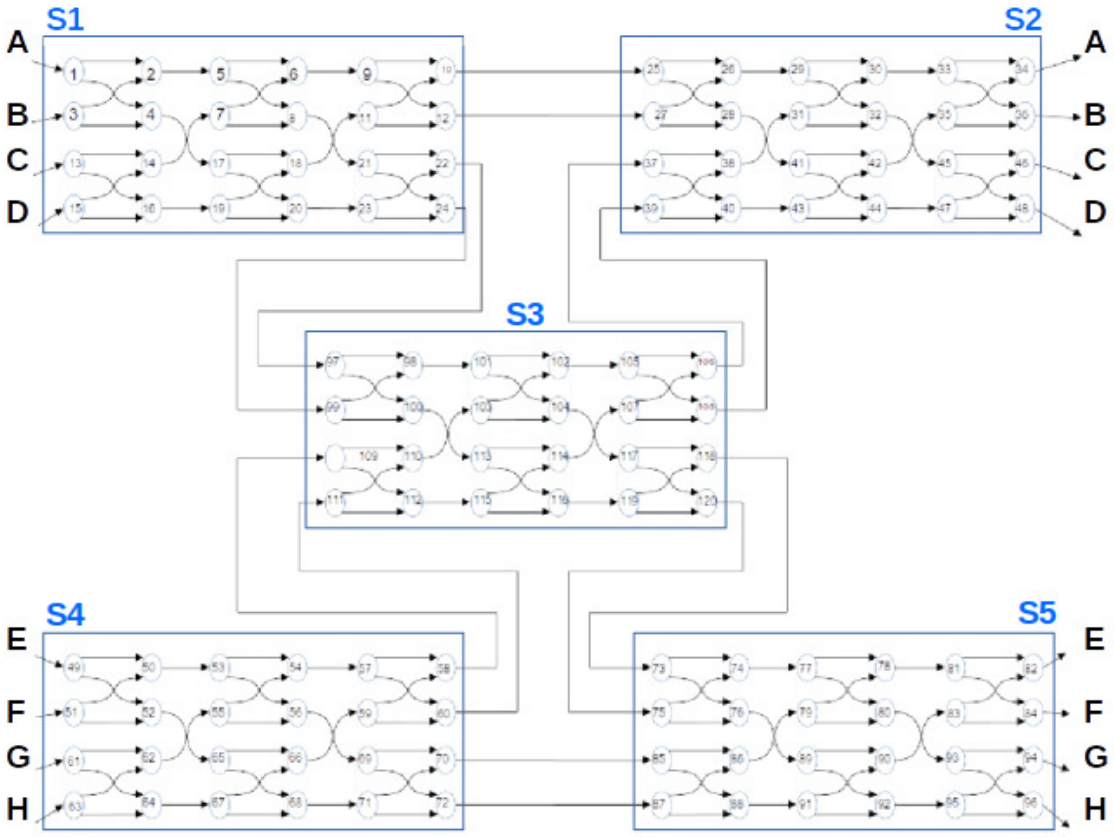


Figure 5.5 Graph view of 8×8 SF Network.

and analyzes the information regarding the 4×4 switch. Following, it checks the next 4×4 switch in the path, until the destination is reached. In the figure, it is possible to notice the procedure using any input as an example. Assuming input A wants to communicate with output A. The DSB checks that input A is connected to the 4×4 switch S1. Using the LITE-LUT information a given route will be processed. Analyzing the route, the DSB checks that the next 4×4 switch is the S2. Once again, the LITE-LUT is accessed and the route for this portion is processed. By the time the DSB realizes the end of the path, the entire route is attributed to the input A to output A communication.

5.4 Discussion

The LUCC alleviates the control latency impact in OIN-based systems. The use of lookup tables containing pre-calculated routes reduces the time spent for network routing. Also, the Round Robin and conflicts detection blocks are implemented in a way to avoid adding extra processing overhead. On the other hand, the dynamic setup block adds extra resource utiliza-

tion overhead, as extra hardware blocks are used. Due to the extra hardware overhead, the power consumption is increased. Also, as a centralized core is used, the controller scaleability can be an issue. Further, as the number of network I/Os rises, the LUT size grows as well. As a result, the LUCC is suited for small-sized networks. Even so, as presented in Chapter 8, it has unmatched latency time.

The LUCC main advantages are:

- Small overhead to detect and solve conflicts;
- Low latency to receive requests and determine network route, and;
- Using a SPF algorithm ensures that, for any requests combination, the best possible route will be used.

The LUCC main drawbacks are:

- Low scaleability as a result of using a centralized core, and;
- Larger memory utilization, used to store the LUT with the pre-calculated routes. Even though reduction methods were introduced to lower the memory utilization, the LITE-LUT is still translated into memory blocks, which requires a large area to be deployed.

CHAPTER 6 CONTROL UNIT - THE HYCO

This chapter introduces the Hybrid Controller (HyCo), a new controller defined to cope with the scaleability issue presented by the LUCC controller. Leveraging the scaleability of circuit-switched techniques and the full control provided by centralized cores, the HyCo employs both approaches to fully exploit OINs capabilities. Moreover, the HyCo takes advantage of the introduced pre-calculated routes to expedite the network configuration and lower the latency. A Bloom filter [95] is employed for the controller to self-optimize itself by learning and storing critical information about the network during the system execution.

Figure 6.1 presents an overview of the HyCo organization. The execution flow followed by requests are represented in the figure with arrows, which show the interaction between blocks.

The main blocks of the HyCo architecture are presented hereafter and each one of the blocks will be explained in the following subsections.

- **Bloom filter**: used to check if the desired route is available;
- **Round-Robin**: employed for the cases where a route is not available or a conflict is found. It is placed in the conflict resolution unit (CRU);
- **Control unit**: receives access requests and triggers the network configuration. It is the access control unit (ACU), and;
- **Configuration units:** are the distributed nodes that configure the optical switches, named distributed configuration units (DCU).

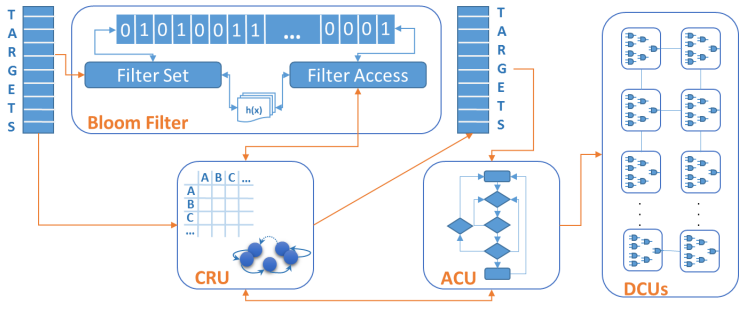


Figure 6.1 HyCo overview. The conflict resolution unit (CRU), the access control unit (ACU) and the distributed configuration units (DCU) are illustrated. The bloom filter functions and array are represented in the top.

Figure 6.2 presents the HyCo execution flow, where it is possible to see the steps executed by the controller, from the moment it receives a request until the request is granted and then the end of the communication. The flowchart represents all steps taken by the HyCo when processing requests. The HyCo expects a request, and by the time it receives one it checks if the targeted destination is available. Next, the HyCo checks if there is no destination conflicts: if a conflict is found, the round robin algorithm determines which requesting node should have its access granted. Following, the HyCo verifies if the requested route was already tried, by checking the Bloom filter. Later, the DCUs are triggered in order for the network to be configured and by the time the network is configured the granting signal is sent to the requesting node. Finally, the HyCo waits for the signal to point the end of the communication, and by the time it receives it, the connection is closed.

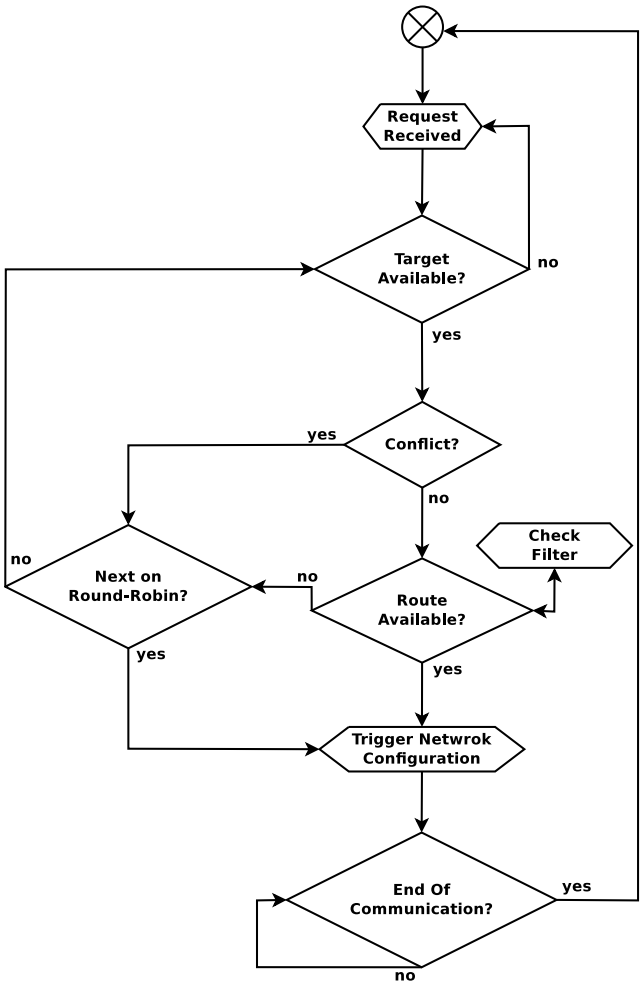


Figure 6.2 HyCo execution flow.

6.1 Conflict Resolution Unit

The Conflict Resolution Unit (CRU) is a hardware block responsible for detecting conflicts in targeted IPs. A conflict is defined as any situation in which two or more source IPs are targeting the same destination IP at the same time.

The CRU works as follows: firstly, it analyzes all the requests, looking for a conflict. Secondly, if a conflict is found, a Round-Robin (RR) algorithm is applied to define which IP will have its accessed granted. To detect a conflict, a matrix method is used: for every new request, all the source-target pairs are mapped to a matrix $\mathcal{R}$ of requests, and then each column j is checked for any possible conflicts. For instance, the matrix for a 3×3 optical switch, where all the inputs are requesting to communicate with the output two ($\langle 0 \to 2, 1 \to 2, 2 \to 2 \rangle$) is:

$$\mathcal{R} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Considering the same switch, but in a different scenario such that $\langle 0 \to 2, 1 \to 0, 2 \to 1 \rangle$, the request matrix changes to:

$$\mathcal{R} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

The matrices are created based on the IDs of the requesting input port and the requested output port. For example, using the same switch as discussed above, $\mathcal{R}(i, j)$ =1 if the input port i requests to access output port j in the switch, such that:

$$\forall_{ij}, if\ request(i) = j \Rightarrow \mathcal{R}_{ij} = 1. \tag{6.1}$$

As the matrix can be accessed directly (i.e., the hardware implementation is a register), no extra processing is needed, thus accelerating the conflict detection.

Once the matrix is generated, all the columns of the matrix (each column is associated with an output port) are verified to find any possible conflicts, where a conflict is defined as any situation in which two or more inputs are requesting the same output. This can be defined as:

$$\forall j \in \mathcal{R}, \quad \neg XOR(j) \wedge OR(j) \implies conflict(j) = 1. \tag{6.2}$$

Taking matrix $\mathcal{R}$ as an example, such that:

$$\mathcal{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix},$$

a conflict is configured for inputs one and two, as both of them are targeting output two. Using equation 6.2 for column two we have:

$$\neg(0\ XOR\ 1\ XOR\ 1) \wedge (0\ OR\ 1\ OR\ 1) \implies \neg(0) \wedge (1) \implies (1) \wedge (1) \implies conflict(2) = 1.$$

When a conflict is found, FIFO queue is implemented based on the RR algorithm to decide which port will be granted access. Since the RR algorithm uses a FIFO for each input, each request is treated individually as one particular process that is triggered by the conflict flag controlled by the matrix method.

Figure 6.3 illustrates the conflict detection block execution flow on a waveform graph. In the figure, it is possible to see two different input ports requesting (*req(0)* and *req(1)*) access to the same output port (*target(0)* and *target(1)*): output 2. This situation sets the conflict bit on the output 2 (*conflict(2)*) which triggers the Round-Robin algorithm. Further, it is possible to see that for each new conflict, the RR picks a different port to be granted, avoiding starvation. When a port requests access to the network it is expected that at some point it will have its access granted, even when conflicts are found. When a port is never granted access, it is called starvation.

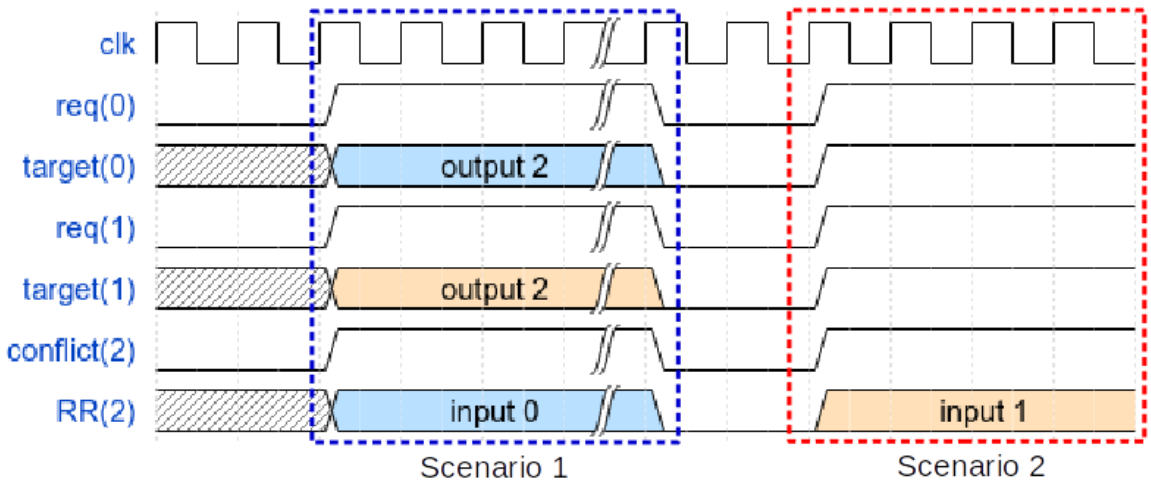


Figure 6.3 Conflict detection and Round-Robin execution flow.

6.2 Bloom Filter

A Bloom filter [95] essentially consists of a bit vector of length m. In order to insert a new entry in the Bloom filter, different hash functions[1] are used. Each function returns a given value, which is used as index. The returned indexes are then used to access the Bloom bit vector and set these positions to '1'. To test if an item is in the filter, again the filter is fed with the same hash functions. This time, the filter is checked to see if any of the bits of these positions are not '1'. If any bit is not '1', it means that the item is definitely not in the set. Otherwise, it is probably in the set.

For example, assuming a Bloom filter with $m = 16$ and two hash functions, such that: h1(x) takes the lower bits of the input data and h2(x) takes the higher bits of the input data. Two scenarios show the process to update the bloom filter bit array. First, the Bloom filter bit array should be initialized and all the positions should be marked to '0', as presented next.

Table 6.1 Bloom filter bit array initial state.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In the first scenario, the input set is equal to 108. This way:

$$input = 108d \rightarrow 01101100b$$

$$h1(x) = 1100b \rightarrow 12d$$

$$h2(x) = 0110b \rightarrow 6d$$

By applying the hash functions, it is possible to use the resultant values as indexes to update the Bloom filter bit array:

Table 6.2 Updated Bloom filter bit array with positions 12 and 6 marked to '1'.

0	0	0	**1**	0	0	0	0	0	**1**	0	0	0	0	0	0

[1] A hash function h(x) is a mathematical expression that returns one number based on an implemented equation. In the current context, the hash functions receive as input the targets set and then apply different mixing mechanisms on it, in such a way that each set results in different values. For example, assuming a hash function h(x) that shifts all input bits to the left and $targets = 32$, such that $\langle 0 \rightarrow 2 \; and \; 1 \rightarrow 3 \rangle$. First, the targets array is converted from integer to binary, such that $targets = 1110$. Next, the hash function h(x) is used, so all the bits are shifted left: h(x) = 1101. Finally, the resulting bit array is converted back to integer: $1101 \rightarrow 13$. This way, the Bloom filter array will have its bit in position 13 set to '1'.

In the second scenario, the input set is equal to 161, so:

$$input = 161d \rightarrow 10100001b$$

$$h1(x) = 0001b \rightarrow 1d$$

$$h2(x) = 1010b \rightarrow 10d$$

using the result of the hash functions, the Bloom filter bit array is updated again:

Table 6.3 Updated Bloom filter bit array with positions 1 and 10 marked to '1'.

0	0	0	1	0	**1**	0	0	0	1	0	0	0	0	**1**	0

To check if a value is in the Bloom filter array, the same hash functions should be used. Applying obtained indexes, the Bloom filter bit array is tested: if any position has a zero, then the value is not in the array. If all checked positions have ones, then probably the value is in the bit array.

For example, an input set that equals to 225 gives:

$$input = 225d \rightarrow 11100001b$$

$$h1(x) = 0001b \rightarrow 1d$$

$$h2(x) = 1110b \rightarrow 14d$$

Table 6.4 Bloom filter bit array testing example for input = 225.

0	**0**	0	1	0	1	0	0	0	1	0	0	0	0	**1**	0

In the position 14 the bit is '0', which indicates that this input was not fed to the Bloom filter. Now, testing an input which equals to 161 gives:

$$input = 161d \rightarrow 10100001b$$

$$h1(x) = 0001b \rightarrow 1d$$

$$h2(x) = 1010b \rightarrow 10d$$

Table 6.5 Bloom filter bit array testing example for input = 161.

0	0	0	1	0	**1**	0	0	0	1	0	0	0	0	**1**	0

In this case, both tested positions have '1' which indicates that the input set was probably fed to the Bloom filter.

The Bloom filter is employed in the context of the HyCo in order to avoid unnecessary path searching. By storing in the Bloom filter the information of tried paths which were unsuccessful to route all requesting inputs, the HyCo is able to avoid trying the same path again in the future. For example, using the 4×4 MZI-based Spanke-Beneš optical switch: if the requesting inputs are targeting destinations 0123, such that $\langle 0 \rightarrow 3, 1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 0 \rangle$, not all the paths would be available, as would cause contention. In this case, the input set 0123 would be used to update the Bloom filter, using the same procedure as introduced above. In the future, when the same 0123 set is tried again, the HyCo does not try to route it, as regarding in the information stored in the Bloom filter, it will know that this route is not possible. In this way, it is possible to speed up the routing decisions, as unsuccessfull routes will never be tried again. As the information of which routes are not possible can be added during run-time, the HyCo self-optimizes itself, through learning from previous tries.

The HyCo Bloom filter block works as follows:

1. The block receives the destinations (targets) array and applies different hash functions on them;

2. Using the obtained indexes from the hash functions, the Bloom filter is tested on the resulting positions. Two possible cases can happen:

   - All tested positions have '1'. This means that desired routes were already tried and were not available. In this case, the RR algorithm is used and one of the requesting ports is chosen to be stalled. The new set, without the stalled port is used as input of the Bloom filter block, starting the process from the beginning, and;
   - At least one of the tested bits in the Bloom filter is equal to '0'. This means that this input set either was not tried before or it was tried and successfully routed. In both cases, this indicates that the controller should try to route this input set, which triggers the network distributed configuration units.

3. If all the nodes are successfully routed, the Bloom filter takes no action. For the cases where any input is not satisfied within a specific time, configured by the user, the Bloom

filter is updated. This is achieved by using the hash functions and using the obtained indexes to mark the corresponding bits in the Bloom filter array to '1'. This way, next time the same set is tried again, the Bloom filter will indicate that the desired routes are not possible[2].

The Bloom filter block allows three configurations. STATIC configuration calculates all the possible routes at design time (off-line) and store only the filter. In this case, the Bloom Filter would only be accessed, as there would be no need for further additions. For the cases where the calculation of all possible routes is not possible (due to a large number of combinations, for instance), the Bloom filter can be fed during runtime. In this case, LIMITED configuration is established where a stopping point can be defined, indicating that no extra additions will be admitted to the filter, for the cases where it is known that after some point no extra combinations will be found. Finally, DYNAMIC configuration is used in cases where there is no limit to add new combinations to the filter.

The number of hash functions and the size of the Bloom filter bit array are determined based on the size of the network. As the number of I/O nodes rises, the number of hash functions and the size of the bit array have to be enlarged as well. The number of hash functions varies from four functions up to 16 hash functions, while the bit array size can be configured between a size of 16 and 256 bits. One important aspect to be taken into account when deciding the number of hash functions and the bit array size is their impact on the number of false positives. A small bit vector enlarge the possibility of false positives being found. On the other hand, too big bit array would lead to unnecessary usage of memory. This way, the choice of the bit array size is a trade off between false positives and memory utilization. This decision can be performed relying on analytical methods, which usually leads to the minimization of false positives at the cost of high memory utilization. In order to avoid unnecessary memory utilization, it was chosen to perform this tunning based on the knowledge of the network, where each network has its Bloom filter parameters configured accordingly.

In order to illustrate the basic functionality of the Bloom filter, Figure 6.4 illustrates an execution example. Three scenarios are presented:

1. A request (*req*) is made targeting a given set of output ports (*targets* = 3210, such that

[2]Bloom filters can produce false positives, which are situations where two or more different inputs result in the same indexes, after applying the hash functions. This might lead to a situation where one input set is not routed and the other set, which generates the same indexes is routed. Due to this fact, it was chosen to feed the filter with unsatisfied routes. This way, for the cases where a false positive is found, this will not compromise the HyCo functionalities, as the only thing that will happen is for the HyCo to try to close the route, as if it was not tried unsuccessfully before.

$\langle 0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3 \rangle$). Firstly, the Bloom filter is tested (bloom check = '1') to check if the input set was already tried. If it was not, the HyCo tries to configure the network in order to satisfy all the requests. If this scenario is sucessfull, then by the time the network is already configured, the ACU sends the confirmation signal (*ack* = '1'). As all the requests are satisfied, the Bloom filter is not updated (bloom set = '0');

2. Different ports are targeted (*targets* = 1203, such that $\langle 0 \rightarrow 3, 1 \rightarrow 0, 2 \rightarrow 2, 3 \rightarrow 1 \rangle$). Once again, the first performed action is to check the Bloom filter (bloom check = '1') and as this input set was not already tried the HyCo tries to configure the network in order to satisfy all requests. In this case, not all the inputs could be satisfied, as the network did not provide paths to route all inputs. In this case, hash functions are used in the input set (1203) and the resulting bit(s) in the Bloom filter marked to '1', and;

3. A pre-tried set is used as a request group (*targets* = 1203). In this case, checking the Bloom filter (bloom check = '1') returns positive, pointing out that this set was already tried and the network was not able to route all the requests. Thus, one of the requesting ports is stalled (*targets* = 120X, where X indicates that this input is not requesting access), to be granted later. The network is then configured for the three remaining ports.

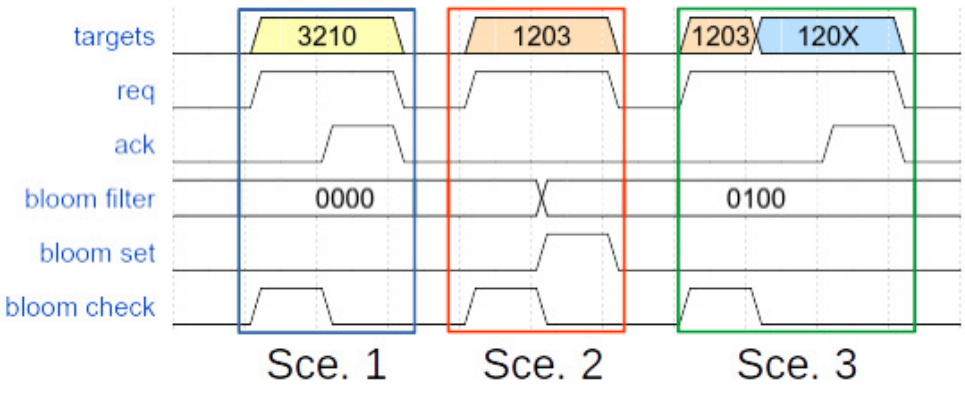


Figure 6.4 Bloom filter block execution example for three scenarios.

The main gain on using a Bloom filter lies in the fact that, as the system runs, previously received requests can be treated faster as the information regarding whether a route is possible or not is stored in the filter. This enables the system to be self-optimized, as its knowledge over the controlled network will improve over time, thus lowering its control latency.

6.3 Access Control Unit (ACU)

The Access Control Unit (ACU) is responsible for the access control to the network. It holds the states of the connected IPs, such that the availability and busy status. When a request arrives, the ACU checks the conflicts, the status of both destination IP and filter to make a decision. If all states are found to be as expected, the ACU triggers the configuration of the network switches. By the time it receives the confirmation pointing that the network is configured, an acknowledge signal is sent to the requesting IP.

Furthermore, the request computation runs in parallel, and each possible request port is considered as one running process. Thus, it is possible for the HyCo to receive requests, solve conflicts and grant access to the network within a short time.

Figure 6.5 presents an example of the ACU execution. For a better visualization, not all the involved signals in the process are presented. Also, each point mentioned in the text is highlighted in the figure with an ID, such as A.1, for instance. In the figure, it is possible to check the ACU block behaviour when a request arrives. Different scenarios are presented:

1. Output port three is targeted. The ACU checks for a conflict on the targeted port (*conflict(3)*), represented by the point 1.A in the figure. As a conflict is found, the ACU expects the RR (*nextOnRR*, point 1.B in the figure) to signal that it is the turn of the requesting port to have its access granted. After the RR confirmation, the network distributed configuration units are triggered, in order for the paths to be configured. After receiving the confirmation that the route is closed (*routeClosed*, point 1.C in the figure), the acknowledge signal is sent (point 1.D in the figure);

2. Port one is targeted. As no conflict is found (point 2.A in the figure) the network distributed configuration units can be triggered. As soon as the path is closed (point 2.B in the figure), the acknowledge signal can be easily granted (point 2.C), and;

3. Port three is once again targeted, and again a conflict is found (point 3.A in the figure). This time, as the port is not the next one in the RR queue (point 3.B in the figure), it is stalled until the granted port targeting the output three ends its communication.

6.4 Distributed Configuration Unit (DCU)

Using the LUTs decreases the computation time, as the routing information can be stored in fast access memories, replacing computation time for storage. Nevertheless, their usage

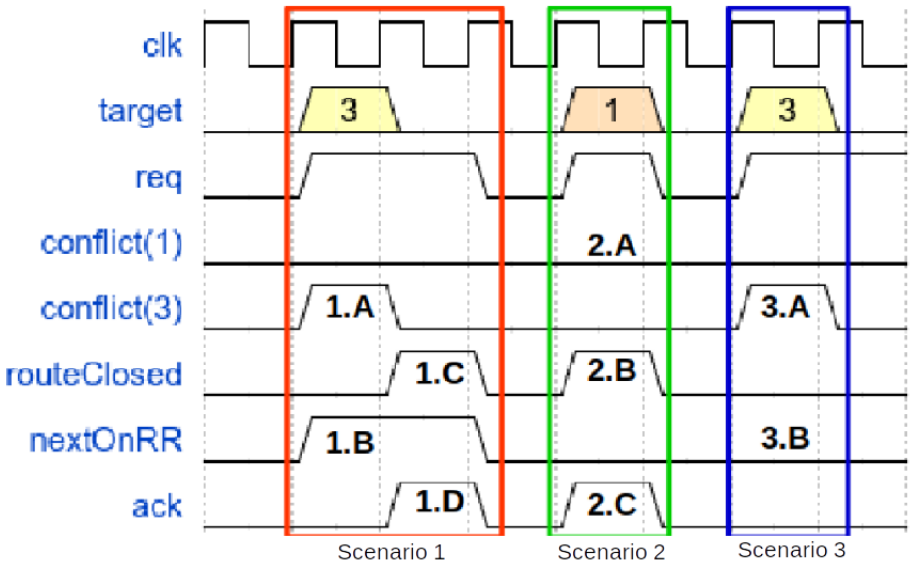


Figure 6.5 Access control unit simplified execution exemplification for one input with three scenarios.

increases the physical memory utilization as the number of path combinations rises to a point at which it becomes prohibitive. The DCU was conceived to reduce the memory utilization.

Since practical implementation relying only on LUTs is hard to achieve for large scale networks, the DCU works as a small, replicable, configuration unit. Similar to the technique introduced in Section 5.3, a reduced version of the final LUT is created, the same LITE version. The LITE-LUT stores only small portions of the network info, such as the most used paths or the paths for one portion of the network. This small portion is then distributed. For example, Figure 6.6.(a) presents a hypothetical network, where a read-only database is connected with four nodes (A, B, C and D). In this case, the paths for nodes A, B, C and D to read from the database can be stored in the LITE-LUT, as they would happen more frequently. Any other communication combination, such as A to B or C to A, for example, would have to be defined during the system execution. In Figure 6.6.(b), if the LUT stores the information of all the possible paths for a 4×4 switch, the LITE-LUT could store only the info of one 2×2. Using the LITE-LUT information, it is replicated in order to have the entire path for the 4×4 switch.

The main drawback when using DCU lies in the fact that not all the possible routes are precalculated and stored. This fact leads to an overhead in time for the network configuration, as the routes need to be calculated during the system execution. Indeed, opposed to the LUCC, the HyCo does not take only one clock cycle. A more detailed discussion related to this aspect will be presented in Chapter Chapter 8. The latency associated with the

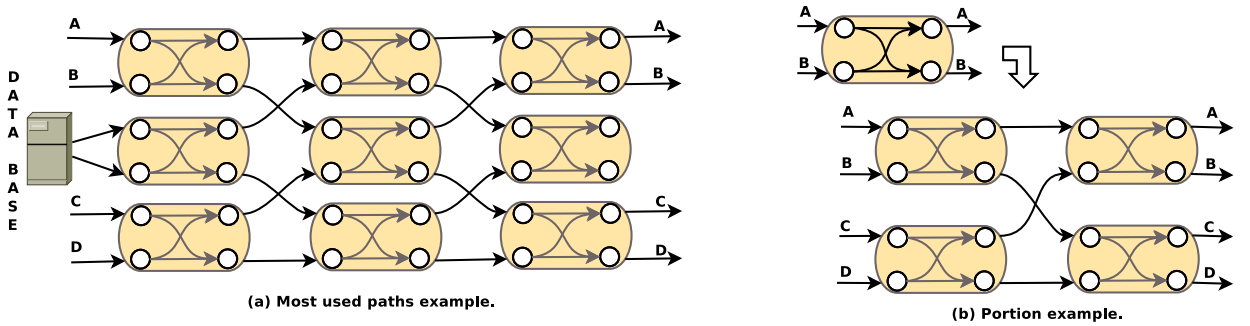


Figure 6.6 DCU Distribution approach.

network configuration is defined by the size of the network, where the larger the number of the network nodes, the higher the latency is. For instance, while the LUCC takes one clock cycle to configure the 4×4 Beneš, the HyCo can take up to three clock cycles. Still, as it will be discussed in Chapter 8, the HyCo presents one of the lowest latency among state-of-the-art works, being overpassed only by the LUCC, with the advantage of a better scaleability.

It is important to highlight that, even-though MZI-based switches were used for example architectures, the HyCo is not anyhow limited to them. HyCo blocks would remain the same, no matter what kind of topology or technology it is controlling. The only thing that changes accordingly with the network is the information stored in the LITE-LUTs and DCUs. As a result, the HyCo can be effectively employed in the networks using MZI-based switches and in those employing Microresonator (MR)-based switches.

6.5 Discussion

The HyCo incorporates the benefits of both centralized and distributed cores to control the network. The main gain in using the distribution approach allied to a centralized core lies in the fact that most of the processing is performed in the central core, allowing the distributed units to be as simple as possible, speeding up its execution and reducing the time required to configure the entire network. Indeed, as presented in Section 8.5, the DCU time overhead is significantly small (nanoseconds), so it does not impose excessive latency on the OIN execution. The deployed matrix method accelerates the conflict detection and the Bloom filter enables self-optimizing feature. Nevertheless, using a Bloom filter adds an overhead in memory, as it uses a bit array. Different bit array sizes lead to different impacts on memory. The matrix method also increases the memory utilization, as hardware registers are used to store the input-output pairs. As the network I/Os number gets larger, the matrix sizes in the matrix method grows accordingly. Further, despites the fact that not only a centralized

core is used, still its usage will limit the scaleability of the approach at some point.

The HyCo main advantages are:

- Acceleration of conflicts detection using a matrix method;
- Improved scaleability for a system based in a central controller;
- Self-optimization characteristics using a Bloom filter, reducing the controller latency over time, and;
- Compared with the classic circuit-switching distributed approach, the introduced HyCo distributed configuration units have reduced latency, thanks to their simplification, leading to a scalable low latency solution.

The HyCo main drawback is:

- Larger memory utilization is required, due to using the Bloom filter bit array and for the matrix method registers.

6.5.1 HyCo and LUCC Comparison

Thanks to the fact that the HyCo was conceived based on the LUCC cons and pros, both controllers have similarities and differences. The two controllers rely on centralized cores for the conflicts solving and Round Robin implementation. Nevertheless, while the LUCC implements a point-to-point conflicts checking, the HyCo utilizes a matrix method. Further, the LUCC have all the network routes stored in fast access memories, accessed by a centralized network configuration block during run time. Likewise, the HyCo uses pre-calculated routes, but they are stored in distributed units. Lastly, the HyCo implements a Bloom filter, which enables the self-optimization of the controller.

The HyCo is able to extend the LUCC scaleability as the need for the controller to have access to all the network nodes is droped. Instead, the HyCo central core is connected to IP nodes and input network nodes only, where the network input nodes are the ones connected directly to IPs. The remaining network nodes are configured by the distributed units. This hybrid approach enabled the HyCo to be used in larger network topologies, when compared to the LUCC. Indeed, while the LUCC is suited for low radix networks, such as 8×8 and 16×16 topologies, the HyCo can be employed in bigger topologies, like 64×64 and 128×128 topologies.

CHAPTER 7 SIMULATION PLATFORM - THE SF-SIM

Based on the defined components, presented in Section 4.4, and using the description models, presented in Section 4.2, a simulation platform was developed. This simulation platform is built over a components library, which is composed of switches and network topologies. The simulation platform is depicted in Figure 7.1, where it is possible to see the steps required for a simulation. The simulation platform is named Straight-forward Simulator (SF-Sim).

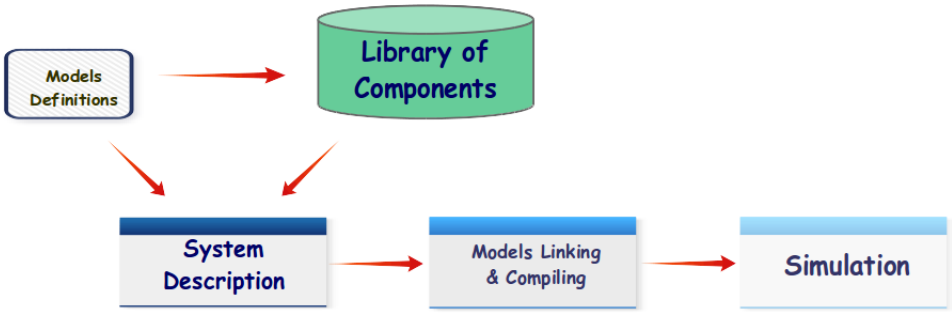


Figure 7.1 Simulation platform.

The SF-Sim consists of a library of components described in hardware description language (VHDL) [54], integrated within a user interface and runs using ModelSim simulation engine [96]. VHDL is used as it has the closest relationship to the hardware layer, which makes it a powerful language to capture desired devices characteristics. Also, the use of VHDL allows to prototype developed designs in FPGA technology, so the models can be further verified. Nevertheless, the models can be employed to describe the components in different programming languages, thus the ModelSim engine will not be required. The specific aspects regarding the implementation can be found in Annex A.

The development of the SF-Sim was achieved in two main steps: first, the devices were modelled using the modelling methodology presented in Chapter 4 and, in the second stage, they were described using VHDL. We explain the main concepts of SF-Sim using simple and illustrative networks/switches.

The first example is a 2×2 MZI-based switch, illustrated in Figure 7.2. This device is used to build all other system components. This block was simulated to verify the expected behaviour, using the ModelSim Simulation Tool.

Figure 7.3 presents the simulation output showing the internal signals and the correct behaviour and timing. In this example, the internal delay is configured to be $\mathcal{T}_{\mathcal{C}} = 4$, so while

Figure 7.2 2×2 modelled switch block.

the signal *tb_we* is high, the correspondent output will receive the inserted data from the input, after 4 ps. Each delay value is configurable, thus any positive time can be used. This delay may be modeled using equations specific to the devices (e.g. MZIs and MRs). The imposed delay is a composition of different physical aspects of fabricated switches, such as waveguide lengths and ring radius. For the validation of the models and results comparison, the delay values were obtained from a collaboration project with McGill University, which helped us to exploit prototyped designs. As presented in Section 8.2, different experiments were performed, which allowed the extraction of precise reference values.

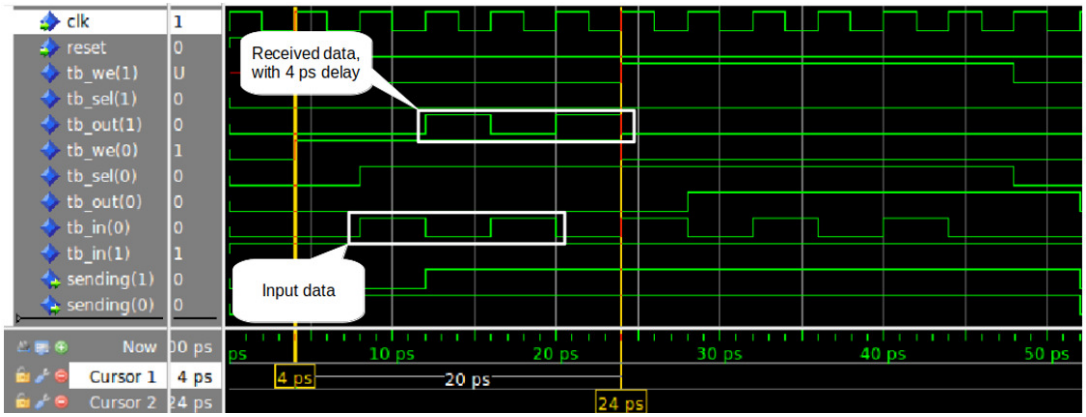


Figure 7.3 2×2 MZI-based switch simulation.

Following, the same steps were applied to the description of a 4×4 MZI-based Beneš switch. The 4×4 Beneš switch is a structure which simply replicates basic 2×2 switches with the addition of channels to interconnect them. Figure 7.4 shows an overview of the designed 4×4 MZI-based Beneš switch, where it is possible to see the 2×2 switches and the I/O pins.

Subsequently, the 4×4 Beneš switch was simulated to verify its correctness. In this example, the transmission delays of connecting waveguides, which connect the internal 2×2 switches are neglected. In order for the waveguides delay to be taken into account, the waveguide models introduced in Section 4.2 should be used. Figure 7.5 presents the simulation output showing the expected timing of the internal signals. Once again, the internal delay is configured to be $\mathcal{T}_C = 4$ ps, so while the signal *we* is high, the correspondent output will receive the

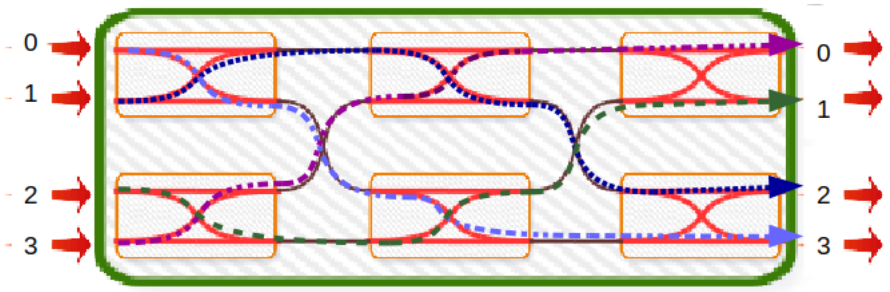


Figure 7.4 4×4 MZI-based Beneš switch block.

correspondent data after 12 ps[1]. Still, the following messages addressing is configured: In0 → Out3; In1 → Out2; In2 → Out1, and; In3 → Out0. In the figure, four scenarios highlighting the serial transmission. Taking the top scenario as an example, it is possible to see four signals: *we(0), input(0), output(3) and sending(3).* In the input side, while we(0) is '1', data is inserted in the switch, represented by signal input(0). In the receiver side, the inserted data is received after the given delay, and is represented by signal output(3). As can be seen, the same inserted data is received in the output, after the configured transmission delay.

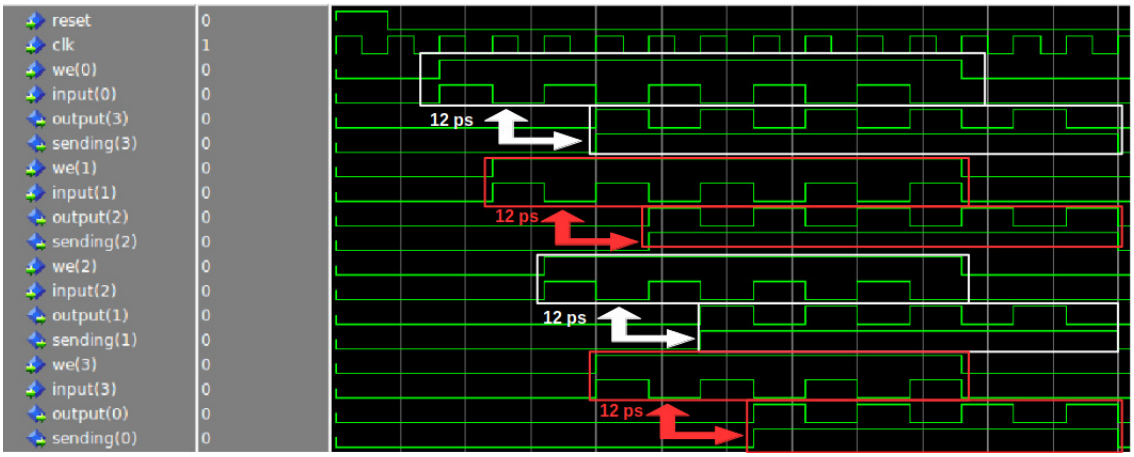


Figure 7.5 4×4 MZI-based switch simulation.

Lastly, two different 8×8 topologies are presented. The first network topology described is an 8×8 Beneš Network, as introduced in Figure 4.9. Finally, the 8×8 SF Network, introduced in Figure 4.13, is described. Both network illustrations were omitted as their final logical designs are too complex.

[1]It takes 12 ps as the message should travel through three 2×2 switches before reaching the output. Given $\mathcal{T}_C$ = 4 ps, 4 ps + 4 ps + 4 ps = 12 ps.

As both topologies rely on the same basic blocks and on the same internal logic to connect those blocks, only the simulation results of one of them will be indicated. This way, Figure 7.6 shows the simulation output for an 8×8 SF Network, where different arriving times can be seen[2]. The unequal arriving times are due to the different paths messages have to travel from the input until they reach the output. As a different number of switches might be crossed by each message and each one of the switches imposes a given delay, the total delay for each message might change. In the figure, two arriving times are highlighted, output(0) receiving the inserted data at input(0) and output(1) receiving the inserted data at input(1).

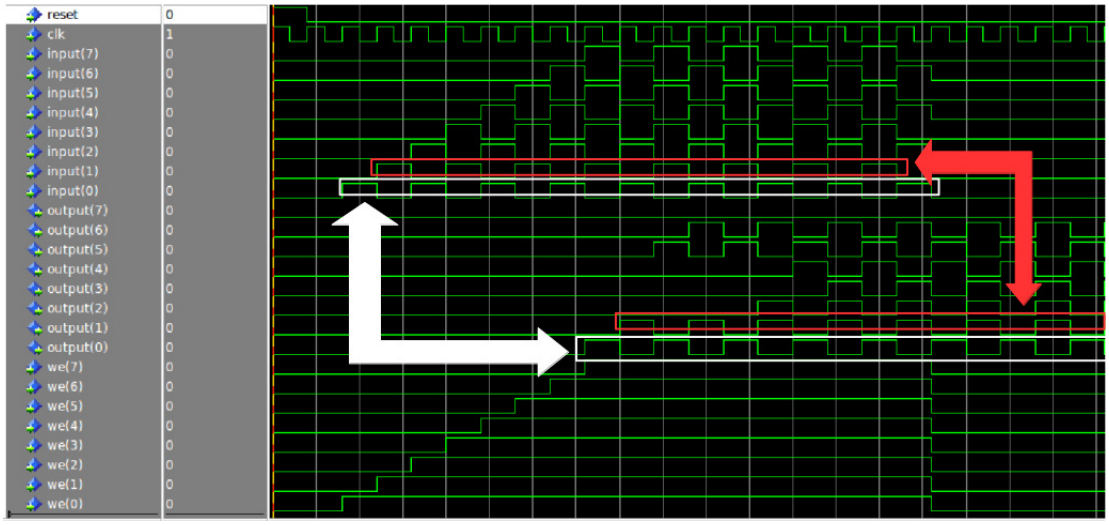
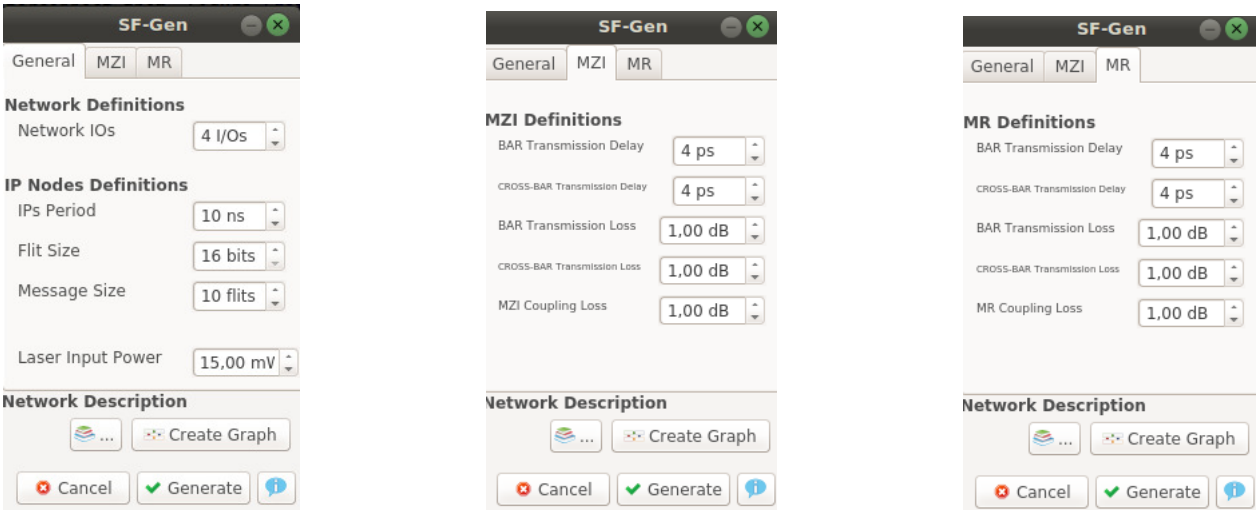


Figure 7.6 8×8 SF network simulation.

Besides the VHDL implementation of the models, a Graphical User Interface (GUI) was also developed. The main goal of this GUI is to ease the deployment of larger systems, as the textual description of such systems would turn into a difficult task. Through the GUI, the user can configure different aspects that comprise the simulation platform, as presented in Figure 7.7: blocks delay (execution time), laser input power, switches (2×2 blocks) transmission and insertion losses and delay, as well as the option to open a third party tool to graphically describe the system. In this case, the DIA tool [97] is used, as it is an open source diagramming tool, suitable for the description task.

After configuring all the aspects of the system, the DIA tool can be employed to visually describe the system. Figure 7.8 illustrates the SF-Sim component library already made available on the DIA tool. These components were modelled and later added to the DIA

[2]Arriving time corresponds to the clock cycle the first sent bit arrives at the targeted output



(a) SF-Sim Traffic Configuration Window.

(b) SF-Sim MZIs Configuration Window.

(c) SF-Sim MRs Configuration Window.

Figure 7.7 SF-Sim Configuration Window.

library. As an example, a small system described in the tool is presented, a hypothetical blocking 4×4 MR-based switch. On the left-hand-side of the Figure, the available components to be used to build the network are presented, and on the right-hand-side the example topology is illustrated: four I/O nodes (each coloured one is one I/O node, where each colour corresponds to one index. This way, two nodes have the same colour, one representing the node input and the other indicates the node output), 12 MRs and their connections. The available components in the tool are the ones presented as building blocks in this thesis: 2×2 blocks (MR and MZI), 4×4 blocks (Beneš and Spanke-Beneš) as well as a strictly non-blocking 8×8 topology.

7.1 Discussion

The SF-Sim was conceived in order to ease the deployment of OIN-based systems and to simplify the generation of introduced controlling solutions. Its usage enables early evaluation of OINs as well generates the VHDL files which are necessary to use the low latency controllers. From the simulations, it is possible to obtain results in terms of dissipated power and network traffic. Also, by using the DIA tool, the description of the topologies is facilitated, as it is possible to describe different topologies graphically. Further, designers can add new optical components using the models introduced in Section 4.2 and enrich the available library of components. Besides that, all files needed for the controllers VHDL files to be created are automatically generated by the tool. This way, all topology related controllers configurations

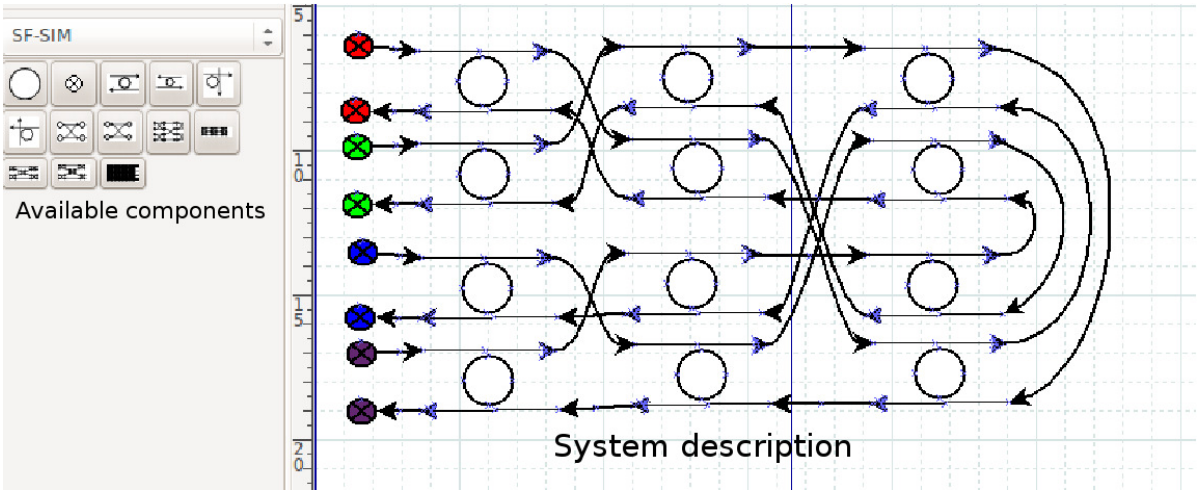


Figure 7.8 4×4 Example MR-based topology on DIA tool.

are made available: the LUT/LITE-LUT; the matrix for the conflicts resolution units; the distributed configuration units information, among others.

The SF-Sim main advantages are:

- Acceleration of the description of topologies, by using a GUI;
- Including a library of fast and accurate components to be used in the OINs description;
- Automated generation of controllers VHDL files, and;
- Simplified way to obtain results regarding the OIN from a system level perspective.

The SF-Sim main drawback is:

- Larger simulation time due the utilization of VHDL when compared to higher level description languages.

CHAPTER 8 RESULTS

This chapter presents the evaluation of the proposed approaches. Section 8.1 shows the LUCC evaluation, where its latency is measured within the context of its deployment: low radix network systems. Later, in Section 8.2 the opto-electrical co-design of the controller model in a prototyped system is presented. Section 8.3 presents the evaluation of the simulation models, by comparing the simulation output with the measured values from real devices, such as the 4×4 MZI-based switch introduced in Section 8.2, or using values extracted from referred publications. Also, the average simulation time for illustrated cases is presented. Next, Section 8.4 presents results obtained from applying HyCo, where its self-optimization feature is shown along with the HyCo latency and in Section 8.5 synthesis reports are presented. Finally, in Section 8.6, deployed controllers are compared with state-of-the-art controllers to verify their low latency.

8.1 LUCC Execution Results

To validate LUCC, different traffic patterns, such as complement and all-to-all are considered to assess LUCC's performance under various request conditions. The complement traffic pattern is used to verify the largest paths in the network. Largest paths refer to the paths in which the transmitted message passes through the largest number of network nodes. All-to-all traffic pattern comprises all possible communication combinations in the network, as the IPs present in the system request access to all nodes, one by one. For these validations, the traffic load is not taken into account, as the aspects involved in the conflict resolution and network configuration are not directly affected by that. The focus is on determining the LUCC response time for different request configurations and patterns. Figure 8.1 depicts employed traffic patterns. In the figure, the IPs zero and seven (IP 0 and IP 7) are used as an example, where it is possible to see the targets each pattern employs. For the all-to-all traffic pattern, IP 0 requests access to all the nodes in the network, one at a time, while in the complement, IP 0 requests access to the other extreme of the network (IP 7), and IP 7 request access to its other extreme, the IP 0. Both simulations and FPGA-based prototyping were performed.

An effort is made to verify the one clock cycle latency of the LUCC under different scenarios, as well as exploring the correct network configuration. We first perform simulations and then the same scenarios were applied on FPGA prototypes. For both cases (simulation and prototyping), a list of signals is employed to illustrate the results, presented as arrays of

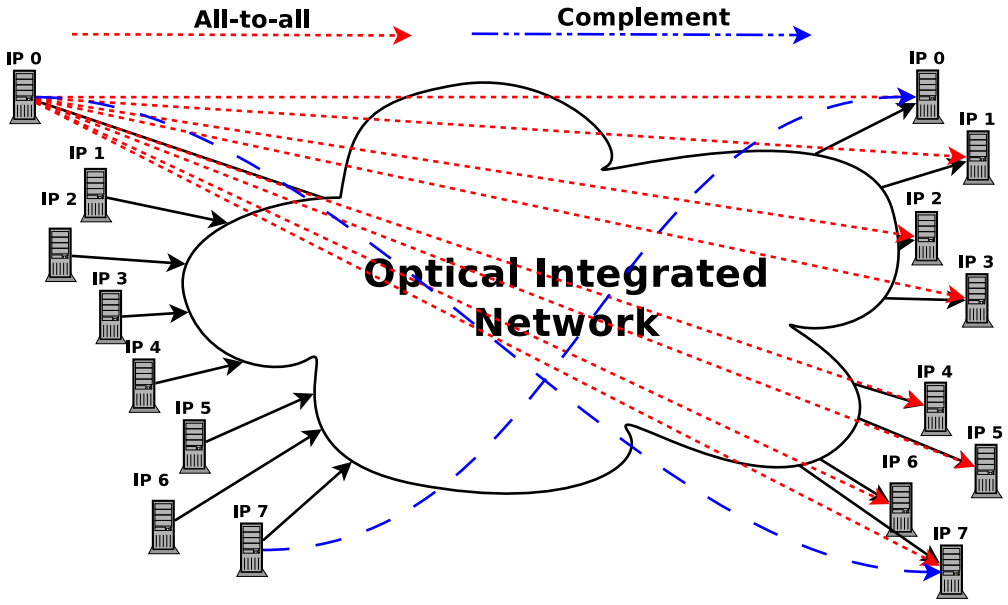


Figure 8.1 Traffic patterns exemplification.

values organized in a little-endian fashion. As a result, the rightmost and leftmost values of the array hold the information of *IP 0* and *IP 7*, respectively. The most important used signals are:

- **target:** it holds the target for each requesting port. If the value for a position equals *-1*, it means that the IP at that position is not requesting access to the network. Otherwise, it shows the desired destination. For instance, the array *target = 0 -1 -1 -1 -1 -1 3 -1* indicates that the *IP 7*, whose position in the array is on the leftmost, is targeting *IP 0* and *IP 1* is targeting *IP 3*. Also, the other IPs are not requesting any access;

- **request:** it is used by the IPs to signal their intention to access the network, where *1* shows a request for access and *0* means idle (i.e., no request). For instance, the array *request = 00000001* indicates that the *IP 0*, whose position in the array is at the rightmost, is requesting access to the network, while all the other IPs are idle, and;

- **ack:** it is used by the LUCC to signal IPs when the access is granted, where *1* means that the access is permitted and *0* means that it is not. For instance, the array *ack = 10010000* shows that the *IP 7* and *IP 4* are permitted to access the network, while all the others are not.

8.1.1 LUCC Simulation

The simulation results show the fast response time of LUCC. The latency is given by the frequency execution of the platform, as the control unit is able to solve requests in one clock cycle. Figure 8.2 illustrates one cycle response time of LUCC. In this example, two scenarios are presented.

In the first scenario, marked with yellow boxes, four simultaneous input requests (*request = 1111*) are generated (*target* = 0123, such that $\langle 0 \rightarrow 3, 1 \rightarrow 2, 2 \rightarrow 1, 3 \rightarrow 0 \rangle$) and their access is granted after one clock cycle (in the figure, *request* signals are set to *1* and one clock cycle later *ack* signals are set to **1**). The yellow arrow shows the period it takes for the LUCC to acknowledge the access.

In the second communication scenario, marked with red boxes, four simultaneous input requests are again generated (*target* = 2022, such that $\langle 0 \rightarrow 2, 1 \rightarrow 2, 2 \rightarrow 0, 3 \rightarrow 2 \rangle$). Note that more than one input is targeting the output 2, and hence a conflict occurs. It is possible to see the resolution of the conflicted request at a time, and even for the cases in which a conflict is found, the controller latency is not affected for computing the requests. First, among the conflicting request ports, *IP 0* is granted access (i.e., *ack = ***1*). Next, *IP 1* has its access granted (i.e., *ack = **1**), and finally *IP 3* is granted access (i.e., *ack = 1****). In this situation, the receiver side takes two clock cycles to set its connection, thus blocking its port. This fact delays the granting response time of LUCC, as the destination is blocked until the receiver side computes the end of the transmission. This can be noticed in the figure by considering **trans_end** and **end_ack** signals. The simulation results indicate the fast response time of LUCC whose latency is constrained by the frequency of the platform, in which the control unit is able to solve requests in one clock cycle.

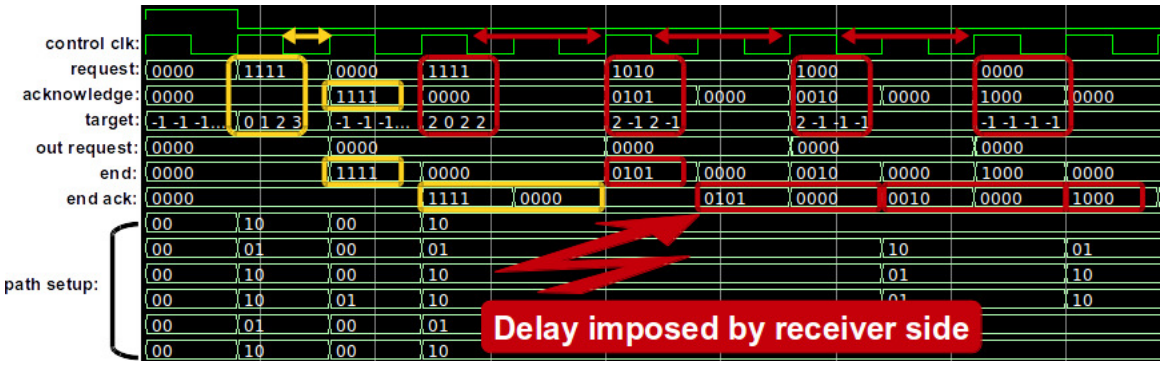


Figure 8.2 LUCC simulation diagram running at 500 MHz for a 4×4 Beneš topology, which shows: input requests (*request*), LUCC grants (*acknowledge*), destination outputs (*target*), LUCC requests for destination outputs (*out request*), end of transmissions (*end*), acknowledges for transmissions ending (*end ack*) and the bits to set up the MZI switches (*path setup*). In each array of bits, one bit corresponds to an I/O. Thus, the size of the array is given by the number of I/Os.

Figure 8.3 illustrates one cycle response time of LUCC when controlling the 8×8 SF topology. Once again two scenarios are presented.

In the first scenario, marked with red boxes, eight simultaneous input requests (*request* = *11111111*) are generated (*target* = 76543210, such that $\langle 0 \to 0, 1 \to 1, 2 \to 2, 3 \to 3, 4 \to 4, 5 \to 5, 6 \to 6, 7 \to 7 \rangle$) and their access is granted after one clock cycle, as there is no conflict. The red arrow shows the period it takes for the LUCC to acknowledge the access.

In the second communication scenario, marked with yellow boxes, four simultaneous input requests are again generated (*target* = 76543217, such that $\langle 0 \to 7, 1 \to 1, 2 \to 2, 3 \to 3, 4 \to 4, 5 \to 5, 6 \to 6, 7 \to 7 \rangle$). Note that more than one input is targeting the output 7, and hence a conflict occurs. It is possible to see the resolution of the conflicted request at a time, and even for the cases in which a conflict is found, the controller latency is not affected for computing the requests.

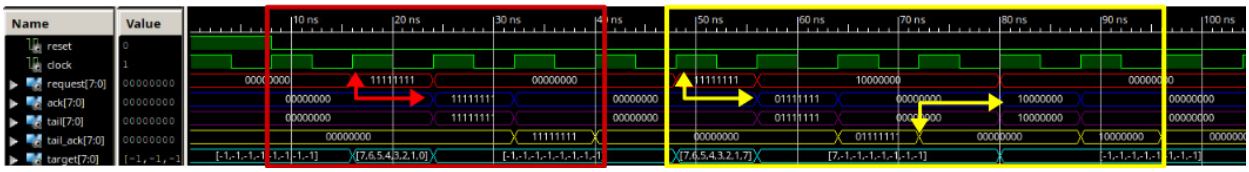


Figure 8.3 LUCC Simulation for 8×8 SF topology.

8.1.2 LUCC Prototyping in Xilinx FPGA

LUCC was synthesized for a Xilinx FPGA [98], the Xilinx II-Pro, from the Xilinx University Program (XUP). The signal readings were performed using the Chipscope Pro Analyzer, also from Xilinx. The 4×4 Beneš topology is employed and the input-output configuration, for the presented example is 0123, such that $\langle 0 \to 3, 1 \to 2, 2 \to 1, 3 \to 0 \rangle$. Figure 8.4 shows four ports requesting (**RX_***) access and receiving a granting (**ACK_***) one clock cycle after. Also, in the figure, the end of communication signals (**TAIL_*** and **TAIL_ACK_***) are presented, illustrating the protocol used at the end of a transmission. The LUCC latency is based on the deployed technology, as it is able to compute the requests in one clock cycle. Thus, for this specific scenario, the FPGA execution frequency was configured to 50 MHz, which gives a period of 20 ns. As a result, for this FPGA the LUCC latency is 20 ns.

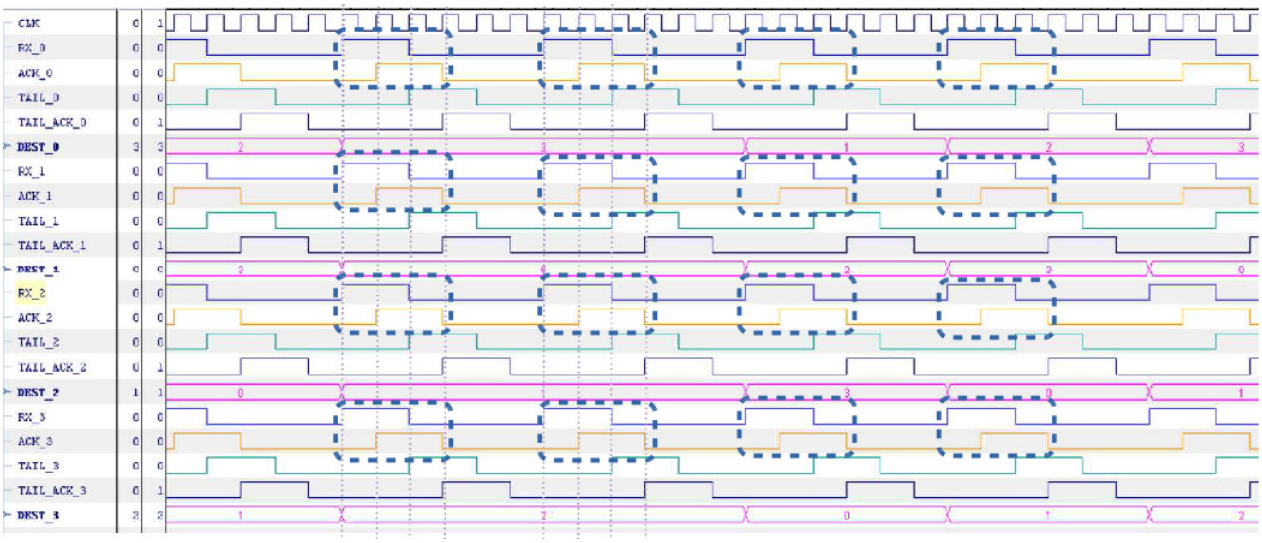


Figure 8.4 LUCC Xilinx FPGA execution with an operation frequency set to 50 MHz. Highlighted boxes show the request and granting moments, within one clock cycle.

8.1.3 LUCC Prototyping in Altera FPGA

After validating the LUCC by simulating and prototyping on Xilinx FPGA, the design was once again prototyped, but this time on an Altera's FPGA [99], with the signal readings being performed with Altera's SignalTap Logic Analyzer Tool [100]. For this step, not only the LUCC was prototyped, but fast transceivers as well, configured for an injection rate of 8 Gbps. In this scenario, the LUCC receives requests from input blocks. Once the access is granted, the input blocks insert traffic into the transceivers. The transceivers output is

connected to I/O pins in the FPGA. In order to emulate the transmitted traffic, virtual links were used to connect these I/O pins. The FPGA used in this case was the Stratix IV 330T, configured to an execution frequency of 100 MHz, with a period equals 10 ns. Figure 8.5 presents the readings, where it is possible to see the request, ack, tail and tail_ack signals, with the same functionalities as explained in the previous section. Moreover, three more signal groups are presented: *Gen_DATA_**, *TX_DATA_** and *RX_DATA_**. These signals correspond to the traffic generated in the inputs (*Gen_DATA*), the fast transceivers input (*RX_DATA*) and the fast transceivers output (*TX_DATA*). The main goal of this step is to show the LUCC interacting with IPs, in this case the traffic injectors.

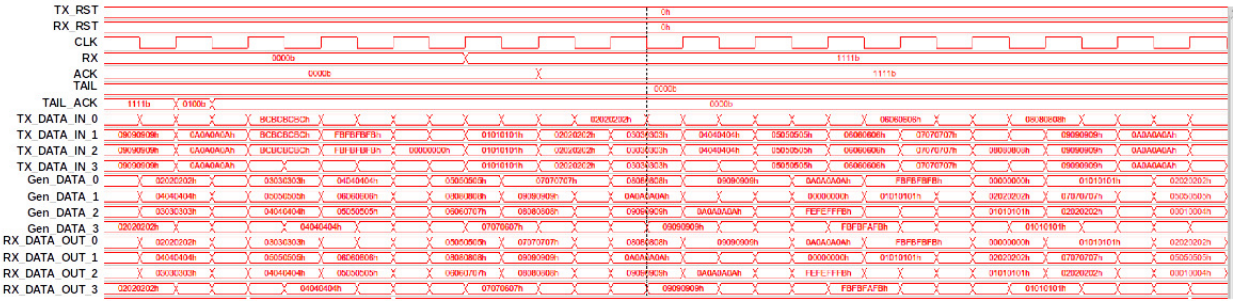


Figure 8.5 LUCC on Altera FPGA execution with an operation frequency set to 100 MHz.

8.2 Co-Design of the Control Unit and the Optical Switch

This section discusses the joint design (also known as co-design) of the LUCC and a fabricated optical switch. Different aspects are involved when co-designing OIN-based systems. The multiprocessor systems are mainly composed of digital components, such as processors and memories, while OINs work in the optical domain, mostly running at a light-based basis. Therefore, the opto-electrical co-design must consider the integration of different domains while maintaining the desired performance and full functionality of the system.

The LUCC was integrated with a fabricated switch to verify its behaviour in a realistic and dynamic scenario. The integrated Silicon Photonic (SiP) switch used is a 4×4 optical switch distributed on a Spanke-Beneš topology with five integrated 2×2 MZIs directly controlled by LUCC. Carrier injection tuning method was employed to bias one arm of the MZI for high-speed and efficient switching [23]. The SiP chip was fabricated by the IME foundry and the measured $V_\pi L$ and switching time are $0.18\,V mm$ and 6 ns, respectively. This integration took place in the Photonics System Group at McGill University. Figure 8.6 presents a microscope picture of the aforementioned switch.

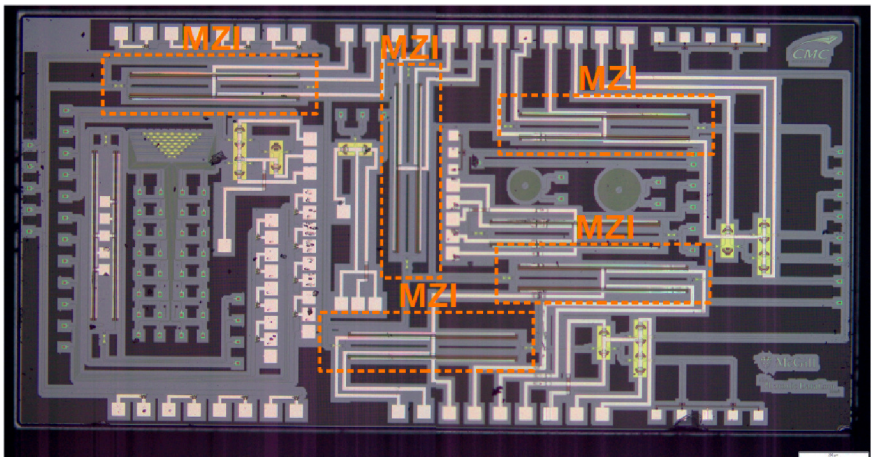


Figure 8.6 Microscopic picture of the 4×4 MZI-based switch.

The lab setup is illustrated in Figure 8.7. The figure presents all the blocks used during the opto-electrical co-design, such as: the driving circuit used as an interface between FPGA and the optical switch; the conversion equipment employed to convert digital to optical and optical to digital; signal amplifiers; optical switch, and; digital blocks (controller, traffic generators, etc.).

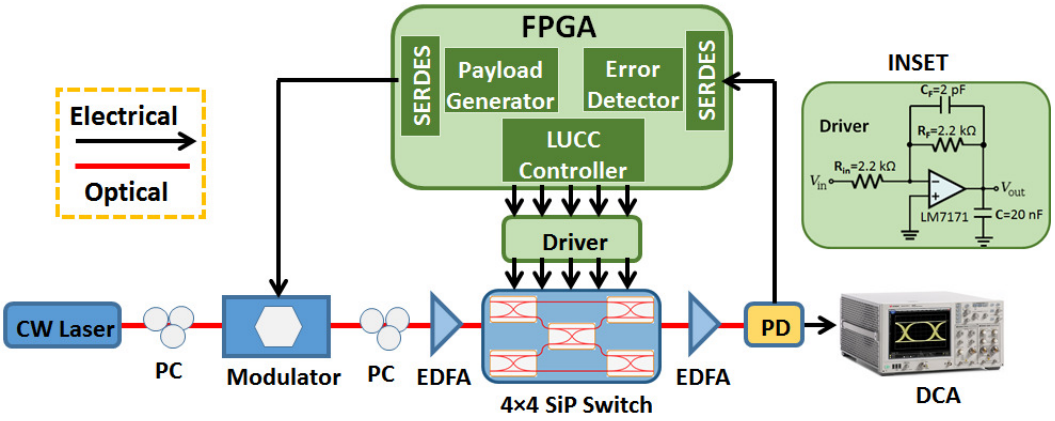


Figure 8.7 Schematic of the lab setup for the opto-electrical co-design. INSET: amplifier circuit used as a drive between FPGA and the optical switch. CW: continuous wave; PC: polarization controller; EDFA: erbium doped fibre amplifier; PD: Photodetector; DCA: digital communication analyzer [101]

.

Before measuring the integration outcome, the switch was solely appraised. The reason to do so is to have reference values to be used as comparison after the integration process. The switch values are used in the simulation models introduced in Chapter 7. Firstly, the MZI-

based 2×2 switch was measured. For this purpose, PRBS31 traffic pattern[1] was injected into one of the inputs of the switch. Following, the switch was analyzed for both bar and cross states, and different aspects measured, such as its switching time and power loss. Figure 8.8 illustrates the PRBS31 packages travelling through the switch. On the top of the figure we show the BAR state configuration and on the bottom we show the CROSS state configuration. In the Figure 8.8.A, it is possible to see the global picture of the execution, where the packages are travelling one after the other. Next, in Figure 8.8.B, the rising and fall times of each package are indicated. Finally, in Figure 8.8.C, the eye diagram of the measured switch. From the experiments, it was possible to extract the information needed by the simulation models as well as to define the comparison scenario to be used later, when all the components are integrated.

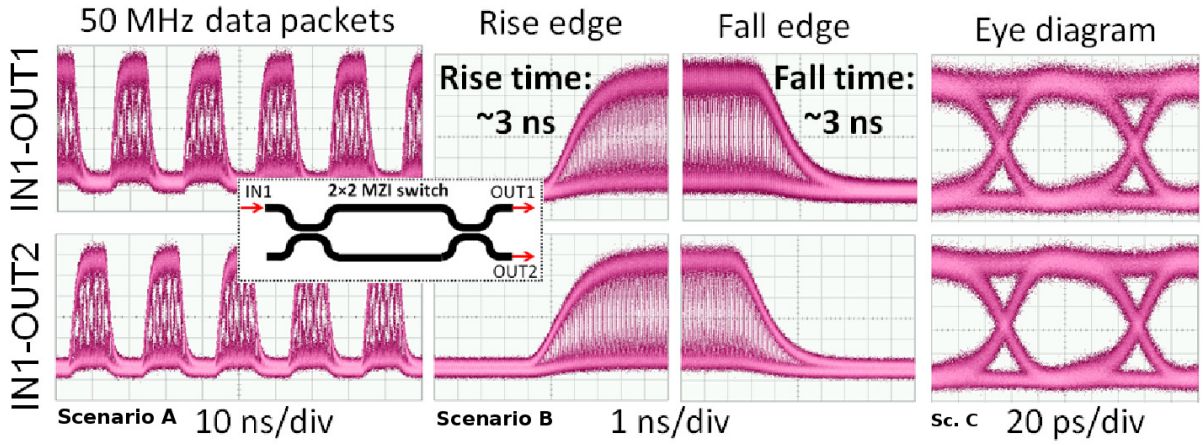


Figure 8.8 Measured 10 Gb/s PRBS31 signal switched by the 2×2 MZI switch. The top row shows the signal for the bar state and the bottom row for the cross state. The inset picture shows the schematic of the 2×2 balanced MZI switch element. The bar and cross states are IN1-OUT1 and IN1-OUT2, respectively [101].

For the opto-electrical co-design, the 4×4 switch was controlled by the LUCC running on the Altera Stratix IV FPGA. Besides the LUCC design, the FPGA was added with a requests generation block. The FPGA was configured to execute at 100 MHz, with a period of 10 ns. Due to the experimental driving circuit, used to configure the MZIs, the switching time was set to 5 s. The lab-setup overview is illustrated in Figure 8.9, where the main blocks are presented. In the figure, it is possible to see that for this experiment only inputs **A** and **B** and the outputs **F** and **G** are used.

Figure 8.10 presents the extracted results, where the FPGA readings are shown as well. The

[1]PRBS31 is a pseudo-random binary sequence that is difficult to predict and exhibits statistical behaviour similar to a truly random sequence.

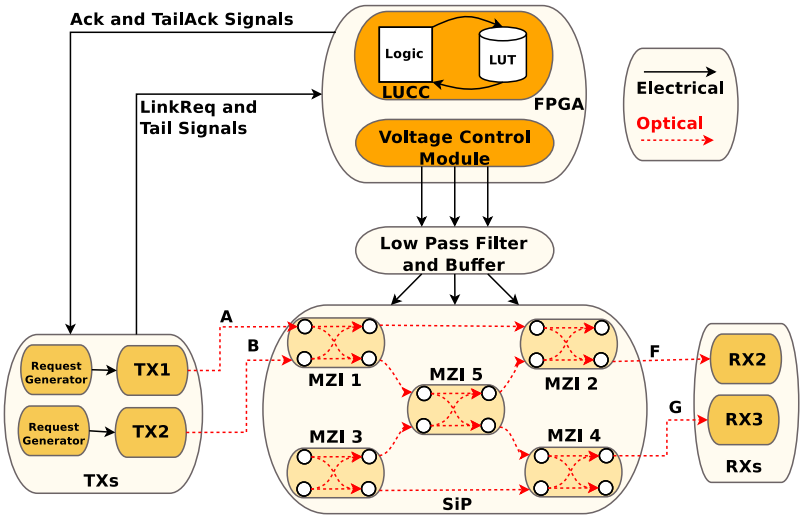
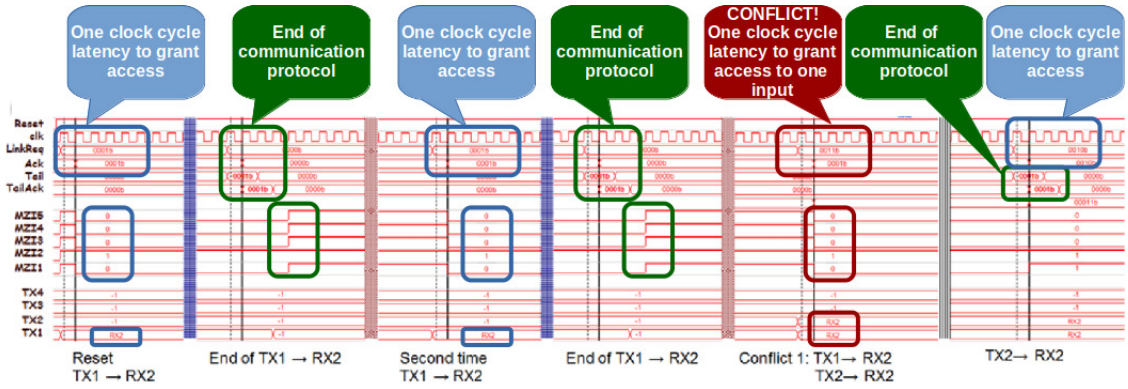
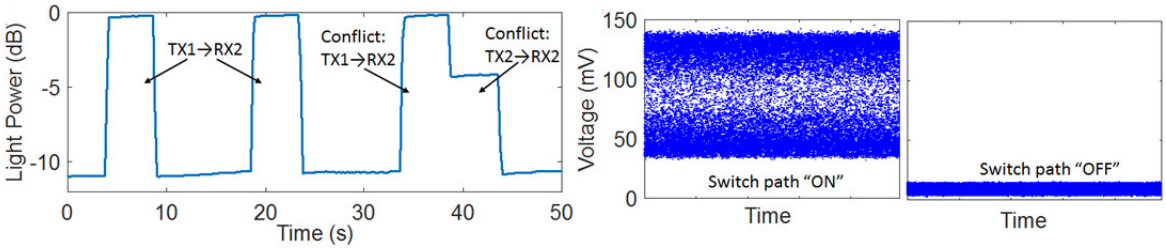


Figure 8.9 Lab-setup overview.

execution of the following configurations are presented: **(i)** input A targeting output F twice, so it is possible to see the switching ON and OFF, and **(ii)** inputs A and B targeting output F, which results in a conflict. The figure illustrates the FPGA readings for each scenario, where it is possible to see the requests, acks, tail, tail acks, MZIs configuration and target signals. Tail and tail acks signals are used in the protocol to close the connection. The protocol works in a handshake fashion, where the requesting port signals it will finish its connection (setting tail signal to '1') and then receiving a confirmation from the LUCC (tail ack signal equals '1'). As the figure illustrates, the LUCC computes requests within one clock cycle, even when a conflict is configured, as in the last scenario. In order to identify if a conflict is configured, the signals *LinkReq*, *TX1* and *TX2* should be verified. The conflict is illustrated when both *TX1* and *TX2* are requesting access simultaneously (LinkReq=0011) while targeting the same destination, *RX2*. For this co-design step, the request generator send requests to the controller, which configures the network and grant access. One external equipment then injects traffic into the network with a 10 Gbps injection rate. As can be seen from the figure, the traffic payload (illustrated as light power) dynamically changes when different requesting nodes are granted access. From these experiments, it was possible to verify the interfaces needs and to observe the correct switch behaviour when dynamically controlled by the FPGA.



(a) FPGA timing diagram for all scenarios



(b) Switch measurements for all scenarios

Figure 8.10 FPGA and Optical switch readings.

In order to further explore the opto-electrical co-design, we realized a second set of experiments. Figure 8.11 gives an overview of the system setup, where it is possible to see the considered connections. Three possible communication scenarios could be configured for this setup: **(i)** input A targeting output F, **(ii)** input B targeting output F and **(iii)** input D targeting output F. Also, this setup contains all the components needed in a system, not relying on any extra device. Traffic injectors and traffic analyzers were deployed on the FPGA. The execution flow is illustrated in the figure: the traffic injector requests access and by the time it is granted, it starts to send data. On the receiver side, the data is collected and then analyzed for potential transmission errors.

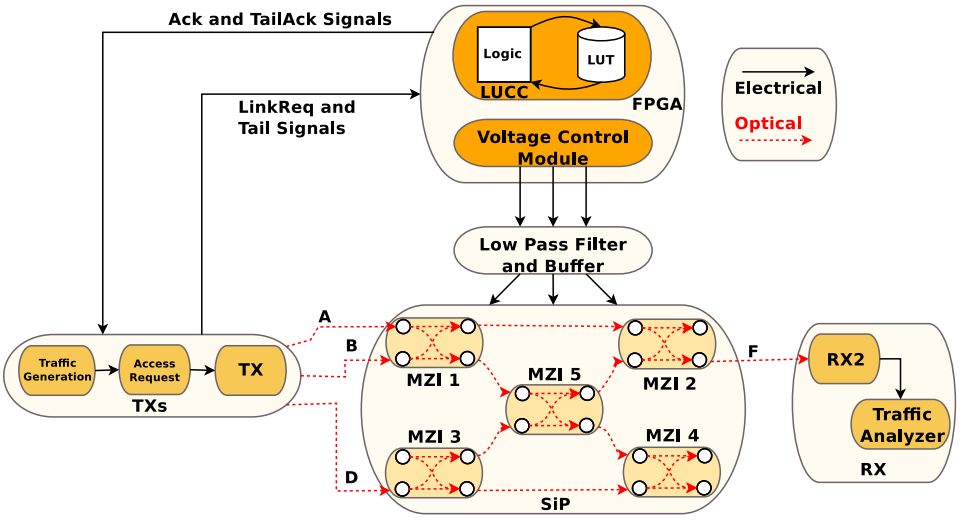


Figure 8.11 Complete system lab setup overview.

In the first set of measurements, only inputs A and B were used. Figure 8.12 presents obtained results for this configuration. Figures 8.12.(a) and 8.12.(b) illustrate the traffic generated on the FPGA travelling through the switches. Also, the request signal and the exact moment of the beginning of the transmission are presented, showing one clock cycle latency. Figure 8.12.(c) brings the traffic injector and analyzer, where a bit error count is used to show the number of miss transmitted bits. From the figure, it is possible to see that 37402251 bits were inserted in the network and properly read on the receiver and seven bit were not successfully read.

Finally, Figure 8.13 presents a setup with three inputs enabled. Three different scenarios are illustrated: **(a)** only input A is requesting access, targeting output F; **(b)** inputs A and B are both targeting output F, and; **(c)** all three inputs request access simultaneously, all targeting output F. For each scenario, it is possible to see the payload readings in the output F. For the conflicting cases, the figure shows the sequential transmission, where each input waits the grant signal to be transmitted.

The initial measurements and the co-design of the controller with the switch enabled the models to be verified and validated. Also, extracted data was integrated into the SF-Sim to enable its high accuracy, as presented in Section 8.3. Further, the performed opto-electrical co-design shows a promising advance to the silicon photonic technology on system-level design, pushing the boundaries for the technology integration one step ahead.

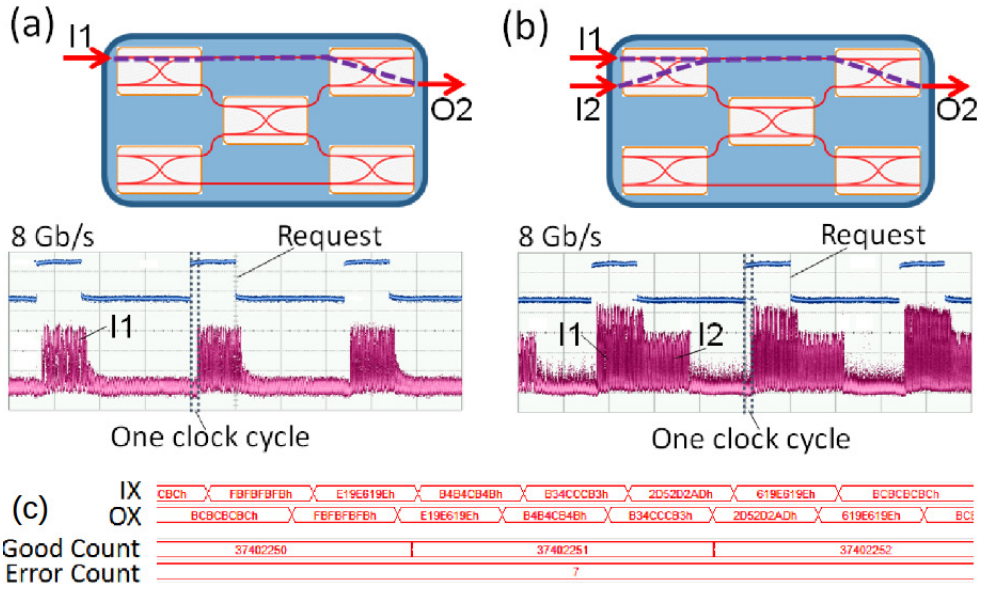


Figure 8.12 Online readings of prototyped optical switch and FPGA execution.

8.3 Models Integration

The models introduced in Section 4.2 are used to describe optical switches and OINs. Also,the controllers developed in the context of this thesis, the HyCo and the LUCC are used to control the described networks. In order to do so, the integration of the control models and optical models is performed. This integration allows the verification of accuracy and correctness of all the models. Data extracted from real devices is available, as presented in Section 8.2, which makes it possible to improve the models in order to reduce the error rate of obtained results when using them. The models integration lead to the SF-Sim, presented in Chapter 7.

The simulation models accuracy was validated in four steps:

1. MZI and MR based switches were modelled with focus on their behaviour;
2. These models had their results compared with the analytical values and the models were updated accordingly;
3. Aspects such as insertion loss were added to the models. This enabled us to measure the power budget of the designed system, under different traffics, and;
4. The models were expanded to the network level so entire systems could be evaluated.

In order to present an overview of the simulation, two different scenarios are presented, both

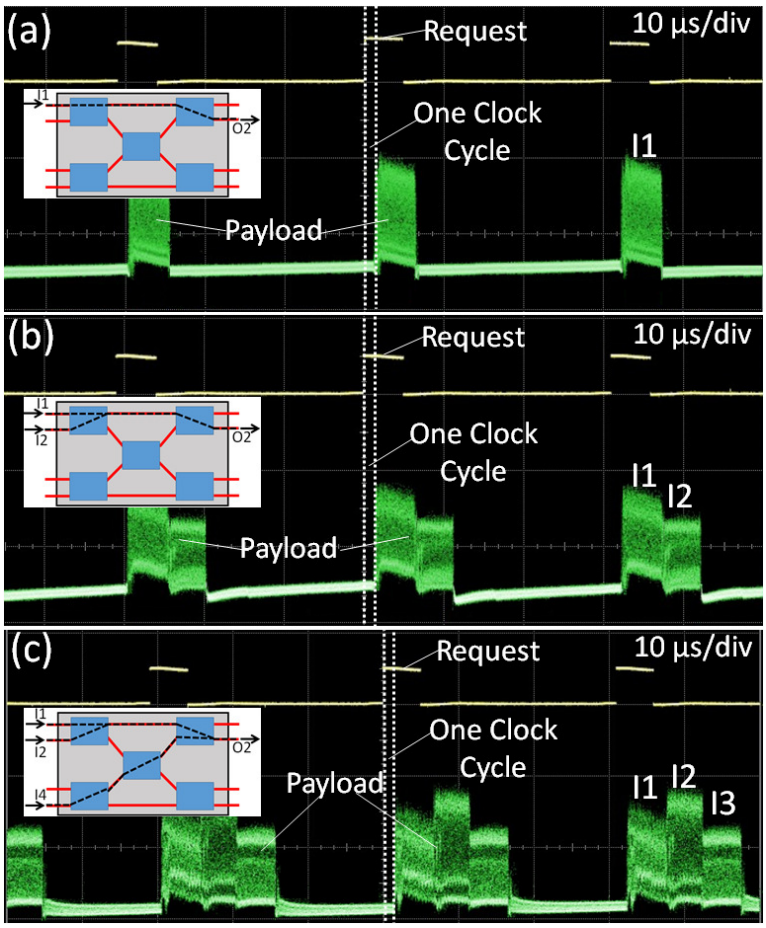
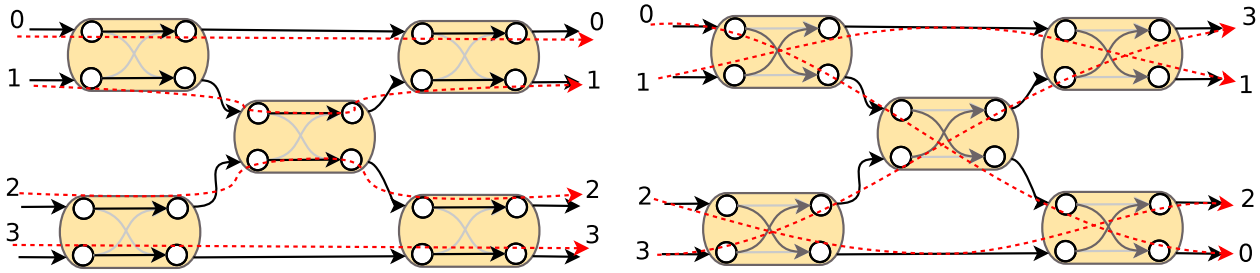


Figure 8.13 Extracted data from lab experiments in McGill Laboratories.

of them using a 4×4 Spanke-Beneš topology. Figure 8.14 presents the schematic of the scenarios considered as an example, where it is possible to see the 4×4 switch organization and the message path for each scenario. In Figure 8.14.(a) the first communication scenario is illustrated: all switches are configured as bar state. This way, input 0 heads towards output 0, input 1 heads towards output 1, input 2 heads towards output 2 and input 3 heads towards output 3. In the second scenario, presented in Figure 8.14.(b) all switches are configured to cross-state, so: input 0 heads towards output 3, input 1 heads towards output 1, input 2 heads towards output 2 and input 3 heads towards output 0. These two scenarios are used as they present an opposite behaviour, when compared with each other. While in one scenario, all switches are configured to bar state, thus not having any communication crossing the switch, in the other scenario all switches are configured to crossbar state, which leads to messages crossing the switch.

An optical network is composed of a large number of components, which are translated as



(a) All switches configured to bar state. (b) All switches configured to cross-state.

Figure 8.14 4×4 Spanke-Beneš Simulation Scenarios.

simulation signals. To show all involved signals in the simulation of an OIN would require an unbearable big figure. This way, for the sake of a better explanation, we use a small system, using a small number of signals. Table 8.1 shows the configuration parameters used for the 2×2 switch. The transmission losses for cross and bar states are the same, normalized as *Transmission loss*.

Table 8.1 Configuration parameters for the 2×2 switch.

Parameter	**Value**
Transmission delay	*100 ps*
Transmission loss	*2 dB*
Coupling loss	*10 dB*
Laser input power	*1 mW*

Figure 8.15 presents the I/O signals of the 4×4 switch, where it is possible to see the following signals: sim_clk, ip_in, ip_out, switch_configuration and power_out. The sim_clk signal is used to ease the results visualization, where each period is configured to 100 ps. In the figure, the left side presents the first scenario, where it is possible to see the delay for the messages to go from one input to the targeted output as well as the output optical power for each output. Taking the communication from input 0 to output 0 as example, the time it takes for the data to go from the input to the output is highlighted in the figure, 200 ps. Also, by checking the power_out(0) signal, the estimated optical power in the output is presented: 0.0398107 mW. This output considers the power losses for all the switches in the message path and the coupling loss. Given the value of 0.04 mW obtained by prototyping in Section 8.2, we conclude that the error rate of the tool is less than 1%. The right side of

the figure presents the second scenario, with all the switches configured as cross-states. The same evaluation might be performed for this case. In the figure, it is possible to check the output power for all the communication inputs, as well as the transmission delay for each case. For instance, the output power of input 1 can be seen in output 1. In this case, the estimated output power is 0.0398107 mW.

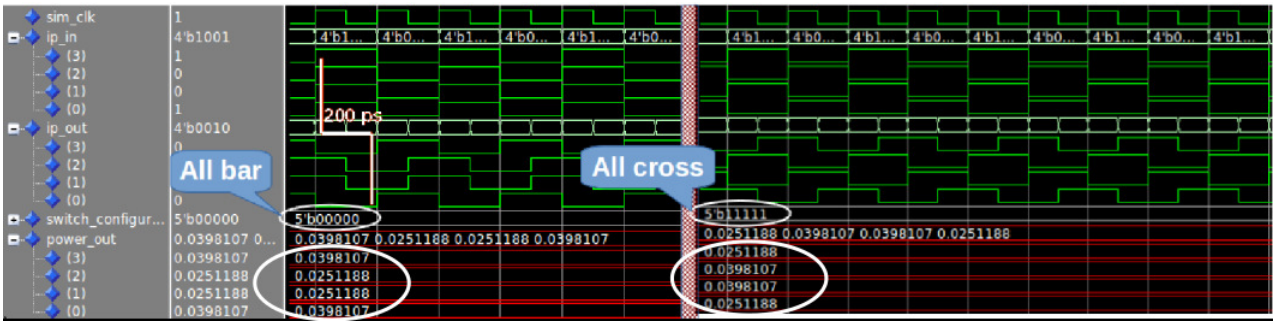


Figure 8.15 4×4 Spanke-Beneš simulation scenarios. On the left, all switches are configured to bar state. On the right, all switches are configured to the cross-state. The clk_in signal can be used as a reference point, with a period configured to 100 ps.

After simulating and validating the network models, all presented models were integrated into one single simulation platform. Figure 8.16 shows one example of a validation platform setup, where it is possible to see the presence of I/O nodes interacting with the control node and with the optical path. The 4×4 Spanke-Beneš topology is used, but this time integrated with traffic injectors and traffic analyzers. In this system, input nodes send requests to the control core, that process the requisitions and configure the network. By the time the network is already configured, and acknowledge signal is sent to requesting nodes, which triggers the communications start.

Figure 8.17 presents the simulation output of the controlled 4×4 switch. In the figure, the switches I/Os, the controller request and ack signals, the switch configuration signals, the inputs targets and the power outcome are presented. The four inputs are requesting access simultaneously ($tb\_req = 1111$). After one clock cycle, two of them have their access granted (inputs one and three - $tb\_ack = 1010$), as it would not be possible to route all the inputs at the same time. After the messages are sent (ip_in), and the end of communication is treated by the controller, the two remaining inputs are granted access (inputs zero and two - $tb\_ack = 0101$).

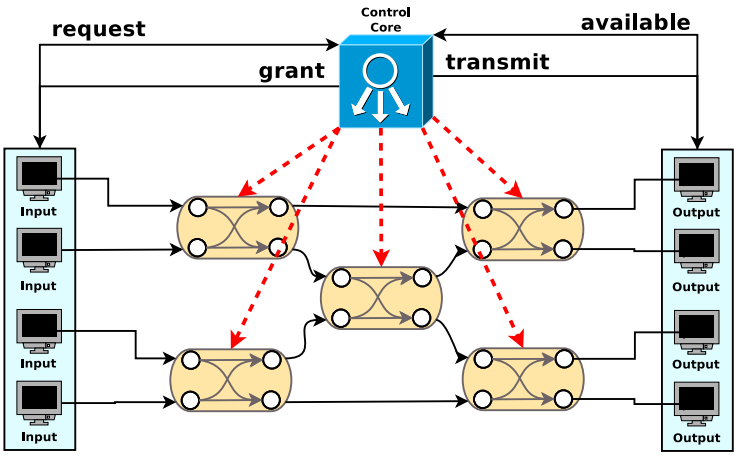


Figure 8.16 System Simulation Setup.

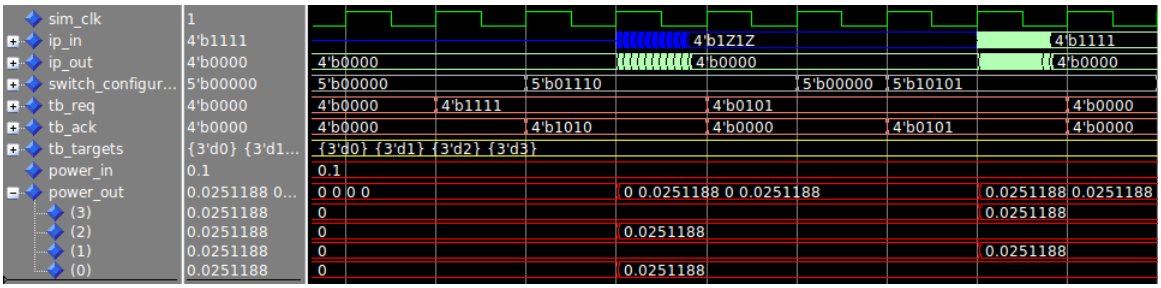


Figure 8.17 System simulation output.

The time required for each network topology is presented in Table 8.2. The considered topologies are: 4×4 Spanke-Beneš topology, 8×8 PILOSS topology [16], 16×16 topology, 32×32 strictly non-blocking topology [89] and 64×64 topology. The number of atomic units (AUs), or switching blocks, is an important factor for the simulator speed. This is explained by the fact that each AU counts as one network node and one control node/table entry. Another important factor in the simulator speed depends on the controller behaviour. The control unit initializes all the network nodes before enabling the network to be accessed, meaning that as the number of network nodes rises, the time it takes for the controller to initialize all nodes rises along. These facts explain why the simulation of the 32×32 SNB Topology is the one that takes the longest time. This topology counts with the largest number of switching blocks, which results in the same number of distributed units, for the controller. This way, when the simulation is eerformed, more simulated components should be taken into account. The executed simulations were configured with a word size of 16 bits and each package composed of 10 messages. IPs frequencies were normalized for all thee scenarios and were executed with a frequency of 100 MHz.

Table 8.2 Simulation times for different topologies.

Network Topology	#AU	Co-Simulation Time (s)
4×4 Spanke-Beneš	*5*	≈ *0.190*
8×8 Ring Topology	*8*	≈ *0.220*
8×8 SF	*20*	≈ *0.317*
8×8 PILOSS Topology	*64*	≈ *0.514*
16×16 Topology	*48*	≈ *0.498*
64×64 Topology	*384*	≈ *1.103*
32×32 SNB Topology	*1024*	≈ *21*

Table 8.3 illustrates the simulator accuracy by comparing the results obtained with the proposed simulator and the results obtained in our measurements and in the literature [16, 89]. The first two scenarios present the measurements of the in-house MZI switches, presented in Section 8.2. The next two scenarios present the results for two different networks, found in published literature. For these cases, the results are approximated. The simulator accuracy is high, with a top error rate of 12 %. It is important to highlight that the reference values are also passive to errors. For instance, the results obtained in [16] have 1 dB error rate (13 %) in their published values. This way, for the 8×8 PILOSS topology for example, the reference value may vary from 6.5 db to 8.5 dB. For our measurements, the error involved in the reference values are due to the equipment employed in the experiments. The obtained values were compared with the state-of-the-art tools and analytical methods and the reading errors could be neglected.

Table 8.3 Simulation accuracy comparison.

Network Topology	Reference Values	Obtained Results	Error Rate
2×2 MZI Switch[1]	≈ *10.96 dB*	≈ *10.97 dB*	≈ *1%*
4×4 Spanke-Beneš[1]	≈ *13.97 dB*	≈ *14 dB*	≈ *1%*
8×8 PILOSS Topology[2]	≈ *7.5 dB*	≈ *8.5 dB*	≈ *12%*
32×32 SNB Topology[2]	≈ *15.8 dB*	≈ *16.18 dB*	≈ *2.5%*

[1] - Measurements performed in the frame of our project, at McGill Photonics Laboratory.

[2] - Results extracted from the state-of-the-art publications. Both cases have approximated results (+- 1 dB).

8.4 HyCo Execution Results

As presented in Chapter 6, the HyCo is powered by a Bloom filter, used to dynamically reduce the controller latency as it executes. The real impact of the filter and the controller on a system was verified. To do so, the devices presented in Chapter 7 were employed and different

network topologies were deployed. Different traffic patterns, such as complement and all-to-all, were used. Figure 8.18 presents the latency for the following network configurations: 4×4 Spanke-Beneš topology, 4×4 Beneš topology, 6×6 Fat-Tree topology, 8×8 Beneš topology [88], 8×8 Ring topology, 8×8 PILOSS topology [16], 16×16 Beneš topology, 32×32 Beneš topology, 32×32 strictly non-blocking topology [89] and 64×64 Beneš topology. In most cases as the system runs, the HyCo starts to exhibit lower latency. This is thanks to the HyCo's self-optimizing, powered by the Bloom filter. As presented in the figure, after running for a given time, which changes depending on the topology, the filter reaches it threshold value and the latency remains the same. For the topologies that are strictly non-blocking, the Bloom filter has no impact. This happens as there are no impossible routes, therefore the Bloom filter is never used. In the figure, the results are presented as a curve that relates latency to time. The latency illustrates the number of clock cycles needed by the HyCo to compute each request. The time axis is presented as a normalized time. This is thanks to the fact that each topology is simulated for a given amount of time, needed to reach the threshold value for the Bloom filter. For example, the 4×4 Spanke-Beneš is simulated for 2 seconds, while the 6×6 fat-tree topology is simulated for 5 seconds. The 4×4 Spanke-Beneš reaches its threshold value at around 1,35 seconds of execution, which represents ≈ 80% of its execution time. The 8×8 Beneš reaches its threshold value at around 4.8 seconds, which is nearly all of its execution time. In order to ease the visualization of the results, the curves are drawn following this normalization approach.

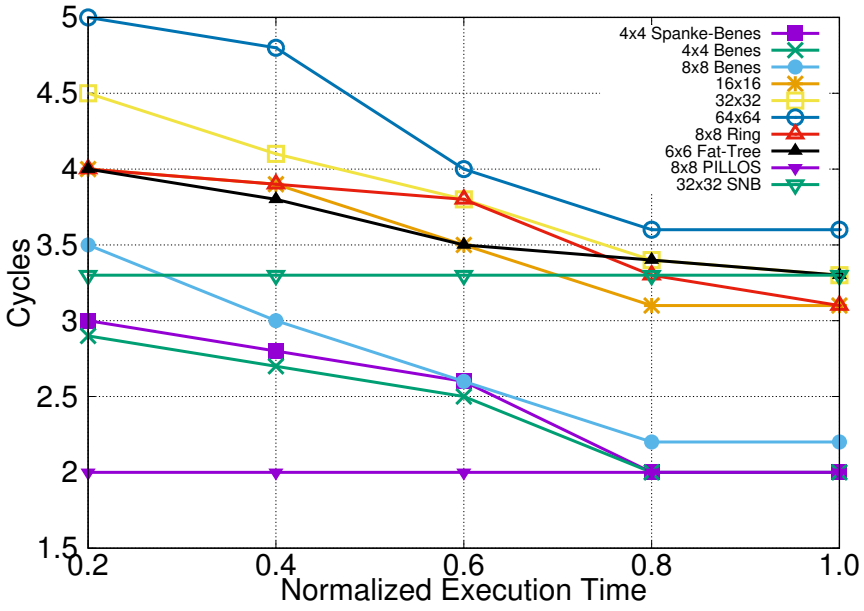


Figure 8.18 Obtained latency of the Hybrid Controller for different topologies.

8.5 HyCo Synthesis Reports

The HyCo was synthesized for different topologies: 4×4 Spanke-Beneš topology, 4×4 Beneš topology, 6×6 Fat-Tree topology, 8×8 Beneš topology, 8×8 Ring topology, 8×8 PILOSS topology, 16×16 topology, 32×32 topology, 32×32 strictly non-blocking topology and 64×64 topology. The chosen topologies were picked to show the HyCo latency for different scenarios, with various radix, switches count, and topological properties. It is important to highlight that the HyCo is not limited to the presented topologies. The only requirement for using HyCO in other topologies is the generation of the LITE LUT.

Table 8.4 presents the obtained results after the synthesis of the Virtex V 330T Xilinx FPGA. We give the number of FPGA logical blocks (FLB[2]) for each block, the minimum delay, as well as the number of distributed configuration units (DCUs). For the 64×64 topology, composed of almost 400 network nodes, the HyCo is able to operate with a delay of 8.57 ns. As presented in subsection 8.4, it takes five clock cycles on average for all the requests to be granted. More specifically, in the worst case, the HyCo can receive, process and grant requests in up to 50 ns.

Table 8.4 Synthesis values for the Virtex V 330T Xilinx FPGA

Network Topology	#FLBs	Block Delay (ns)	#DCUs
4x4 Spanke-Beneš	*1443*	*3.57*	*5*
4x4 Beneš	*1615*	*3.391*	*6*
6x6 Fat-Tree Topology	*3037*	*3.38*	*54*
8x8 Ring Topology	*2370*	*4.15*	*8*
8x8 Beneš	*2556*	*4.12*	*20*
8x8 PILOSS Topology	*3625*	*4.29*	*64*
16x16 Topology	*5792*	*6.25*	*48*
32x32 Topology	*18199*	*7.57*	*120*
32x32 SNB Topology	*32948*	*7.54*	*1024*
64x64 Topology	*67918*	*8.57*	*384*

Table 8.5 illustrates the obtained results for the Stratix IV FPGA from Altera. Altera and Xilinx rely on different technologies and synthesis techniques, and hence different values are found considering the same scenarios. Compared to the Virtex V, the results for the Stratix IV FPGA indicate a slighter bigger area and an increased block delay.

The last executed synthesis is targeting ASIC technology. For that, the proposed flow for the 65 nm STMicro technology was employed. Table 8.6 illustrates the obtained results, organized in three columns presenting the results: *Cell Area:* occupied chip area for the synthesized

[2]In literature, these blocks are called LUTs. As in this work the term LUT is already employed for a different context, the FPGA LUT blocks are named FLB.

Table 8.5 Synthesis values for the Stratix IV Altera FPGA

Network Topology	FLBs	Block Delay (ns)	DCUs
4x4 Spanke-Beneš	*2325*	*3.91*	*5*
4x4 Beneš	*2573*	*3.7*	*6*
6x6 Fat-Tree Topology	*3945*	*4.01*	*54*
8x8 Ring Topology	*2931*	*4.76*	*8*
8x8 Beneš	*3556*	*4.75*	*20*
8x8 PILOSS Topology	*3981*	*5.01*	*64*
16x16 Topology	*6587*	*6.75*	*48*
32x32 Topology	*19273*	*7.97*	*120*
32x32 SNB Topology	*41958*	*8.63*	*1024*
64x64 Topology	*85837*	*9.77*	*384*

block, in millimetres; *Block Delay:* the block signals propagation delay, which shows the minimum clock period for the block, and; *DCU:* the number of distributed configuration units for each topology. Obtained results indicate that, in the worst case, the maximum delay is 3.3 ns. Also, it is possible to see that for a 64×64 topology, composed of almost 400 distributed elements, the chip area would not exceed 34×34 mm, including wiring.

Table 8.6 Synthesis values for the 65nm STMicro Library

Network Topology	Chip Area (mm^2)	Block Delay (ns)	DCUs
4x4 Spanke-Beneš	2.59×2.59	*1.05*	*5*
4x4 Beneš	2.75×2.75	*1.04*	*6*
6x6 Fat-Tree Topology	7.03×7.03	*1.7*	*54*
8x8 Ring Topology	4.29×4.29	*1.5*	*8*
8x8 Beneš	5.2×5.2	*1.1*	*20*
8x8 PILOSS Topology	6.71×6.71	*1.2*	*64*
16x16 Topology	9.3×9.3	*1.5*	*48*
32x32 Topology	18×18	*2*	*120*
64x64 Topology	34×34	*3.3*	*388*

8.6 Controllers Comparison

The LUCC and the HyCo were compared with the state-of-the-art controllers. For a fair comparison, the well-known 8×8 Beneš [88] topology was used. The 8×8 Beneš was selected as it is a well-established topology which is composed of a fair number of switches and is not strictly non-blocking, thus requiring the controllers to perform routing and deal with conflicts. Also, based on the fabricated optical switch, the latency for each optical bit to pass through the network was rounded to 210 ps. The comparison was performed by analyzing the total time required for a message to be arbitrated and passes through the network, such that: $TotalTime = CL + Nob * TD$, where CL stands for control latency, Nob stands for

the number of bits transmitted and *TD* stands for transmission delay. Still, four different message sizes (128 B, 256 B, 512 B, and 1 Kb) were used. Figure 8.19 presents the latency comparison between the HyCo and LUCC and the three state-of-the-art solutions [73, 78, 79].

Figure 8.19 shows that the LUCC and HyCo latency are smaller than the fastest state-of-art solutions. The provided solution [78] was validated through FPGA prototyping, using the Xilinx 1I000E FPGA, with similar latency to our solutions. Nevertheless, its usage imposes modifications on the application network layer, which is not always possible, thus reducing its applicability. The approach used in [79] uses the same time division technique as this work, and obtained fairly similar results. However, the solution is suited for one specific topology, jeopardizing its application to other cases. Finally, the solution presented in [73] claims to use an operation frequency of 5 GHz, which is not realistic, and hence the validations were under simulations only.

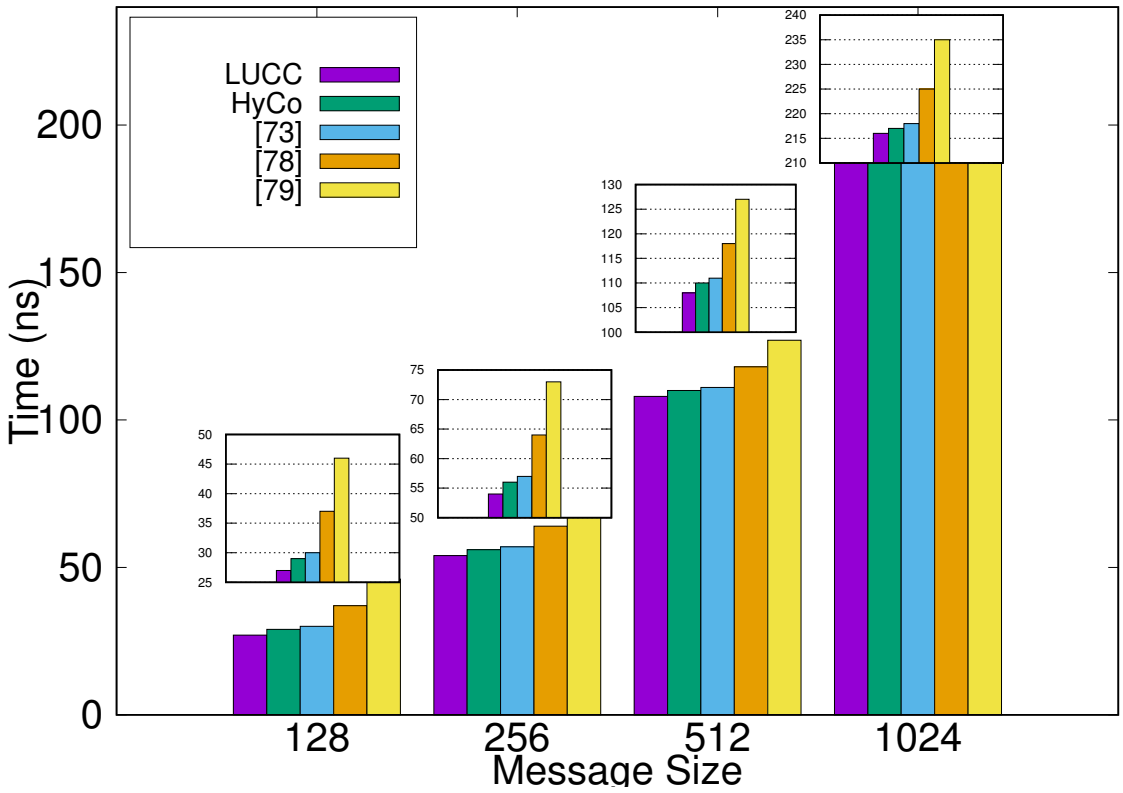


Figure 8.19 Latency comparison between controllers and state-of-the-art.

Based on the presented results, it is possible to see that both the LUCC and the HyCo have good performances when processing requests and dealing with conflict situations for OIN-based systems. Due to implementation characteristics, the LUCC is suited for small

networks, while the HyCo might be applied to bigger topologies. The HyCo's Bloom filter shown to reduce the controller latency until it reaches a threshold value. Also, the Bloom filter exploitation enabled to improve scaleability for HyCo. Besides showing the lowest latency, neither controller demands any changes to the network layer, coping with the tendency for the next generation of high-bandwidth networks.

CHAPTER 9 CONCLUSION

This chapter summarizes the main contributions of this thesis and presents the future work.

9.1 Final Remarks

Optical Integrated Networks (OINs) are currently considered to be one of the most promising paradigm in the multiprocessor systems design context: they present high bandwidth, low power consumption and low latency to broadcast information. Still, their application may be restricted by poor controlling solutions and lack of proper design tools to aid OINs design.

In this context, we proposed the following contributions:

1. The definition of accurate system-level modelling method enabling the development of a system-level simulation platform.

2. The definition and development of efficient control approaches for OIN-based systems.

   The second contribution of this work is the development of low latency controlling solutions, as it follows:

   - *Low latency centralized controller* - a low latency controller based on pre-calculated routes technique for OIN-based systems. This technique enables the exploitation of full capacity of the OIN. Also, this controller is not specific to a given network opology, and;
   - *Hybrid Controller* - based on both a centralized and distributed units, it may be used in high radix count systems and yet present low latency. Further, as the LUCC, the HyCo is not limited by the network it is controlling, being able to employ it in a variety of systems.

3. The system-level evaluation of the proposed control approaches using the defined modelling methods.

   Introduced models were integrated into a simulation platform, where it is possible to design and test OIN-based systems from a system-level perspective. The developed controllers could be deeply evaluated under different scenarios. Their impact could be measured for different technologies, such as MZIs and MRs. Lastly, in the context

of system-level evaluation, the opto-electrical co-design of OIN-based systems was performed. Optical integrated networks are still in the initial development stage, and their application is still limited.

9.2 Future Work

This project opens several new research directions:

1. Expand the developed models in order to incorporate new features. Current models perform power and timing analysis. Expanded models can evaluate new aspects, such as heat dissipation.

2. Extend the controller by adding spatial routing support. Proposing a new approach for exploring routing in both space and time. Our approach considered only the time domain.

3. Further evaluation of the HyCo distributed units. The DCUs are able to perform routing based on pre-calculated routes, stored as LUTs. Still, the relationship between DCU size and obtained latency is not explored yet. Synthesis techniques could be employed to discover the best relationship, in order for the best configuration to be found.

4. Expand the co-design based on a commercial system. Even though the performed co-design proved the OIN-based system feasibility, a design deployed after real systems would enrich the contribution. For instance, a system targeting the Internet of Things (IoT) servers could be deployed, which would serve as a perfect test case for OIN-based systems on one of the mainstream future systems.

REFERENCES

[1] W. Wolf, A.A. Jerraya and G. Martin. Multiprocessor System-on-Chip (MPSoC) Technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, Oct 2008.

[2] T. Le and M. Khalid. NoC prototyping on FPGAs: A case study using an image processing benchmark. In *2009 IEEE International Conference on Electro/Information Technology*, pages 441–445, June 2009.

[3] C. Hilton and B. Nelson. PNoC: a flexible circuit-switched NoC for FPGA-based systems. *IEEE Proceedings in Computers and Digital Techniques*, 153(3):181 – 188, May 2006.

[4] S. Tota, M.R. Casu, M.R. Roch and M. Zamboni. A multiprocessor based packet-switch: performance analysis of the communication infrastructure. In *IEEE Workshop on Signal Processing Systems Design and Implementation, 2005.*, pages 172 – 177, nov 2005.

[5] L. Benini and G. De Micheli. Powering networks on chips: energy-efficient and reliable interconnect design for SoCs. In *Proceedings of the 14th International Symposium on Systems Synthesis*, ISSS '01, pages 33–38, New York, NY, USA, 2001. ACM.

[6] ITRS. International technology roadmap for semiconductors, http://www.itrs.net/ - last access on 12/2016.

[7] W. C. Lo, Y. H. Chen, C. T. Ko and M. G. Kao. TSV and 3D wafer bonding technologies for advanced stacking system and application at ITRI. In *2009 Symposium on VLSI Technology*, pages 70–71, June 2009.

[8] S. Le Beux, G. Nicolescu, G. Bois and P. Paulin. A system-level exploration flow for optical network on chip (ONoC) in 3D MPSoC. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3613–3616, May 2010.

[9] I. P. Kaminow. Optical Integrated Circuits: A Personal Perspective. *Journal of Lightwave Technology*, 26(9):994–1004, May 2008.

[10] W. J. Dally. Future Directions for On-Chip Interconnection Networks. *http://www.ece.ucdavis.edu/ ocin06/talks/dally.pdf*.

[11] Euronymous: 3D integration: a revolution in design. Real World Technologies. May 2007 .

[12] A. F. Benner, M. Ignatowski, J. A. Kash, D. M. Kuchta and M. B. Ritter. Exploitation of optical interconnects in future server architectures. *IBM Journal of Research and Development*, 49(4.5):755–775, July 2005.

[13] D.A.B. Miller. Rationale and challenges for optical interconnects to electronic chips. *Proceedings of the IEEE*, 88(6):728–749, June 2000.

[14] B. G. Lee, A. V. Rylyakov, W. M. J. Green, S. Assefa, C. W. Baks, R. Rimolo-Donadio, D. M. Kuchta, M. H. Khater, T. Barwicz, C. Reinholm, E. Kiewra, S. M. Shank, C. L. Schow and Y. A. Vlasov. Monolithic Silicon Integration of Scaled Photonic Switch Fabrics, CMOS Logic, and Device Driver Circuits. *Journal of Lightwave Technology*, 32(4):743–751, Feb 2014.

[15] M. S. Hai, P. Liao, M. M. Shafiei, O. Liboiron-Ladouceur. MZI-based Non-blocking SOI Switches. In *Asia Communications and Photonics Conference 2014*, page ATh3A.147. Optical Society of America, 2014.

[16] K. Suzuki, K. Tanizawa, T. Matsukawa, G. Cong, S. Kim, S. Suda, M. Ohno, T. Chiba, H. Tadokoro, M. Yanagihara, Y. Igarashi, M. Masahara, S. Namiki and H. Kawashima. Ultra-compact 8x8 strictly-non-blocking Si-wire PILOSS switch. *Opt. Express*, 22(4):3887–3894, Feb 2014.

[17] D.A.B. Miller. Why use optics for interconnects. In *LEOS '97 10th Annual Meeting. Conference Proceedings., IEEE*, Nov 1997.

[18] A.R. Mickelson. Silicon photonics for on-chip interconnections. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pages 1–8, Sept 2011.

[19] A. Shacham, K. Bergman, and L.P. Carloni. Photonic networks-on-chip for future generations of chip multiprocessors. *Computers, IEEE Transactions on*, 57(9):1246–1260, Sept 2008.

[20] A. Biberman, B. G. Lee, N. Sherwood-Droz, M. Lipson and K. Bergman. Broadband Operation of Nanophotonic Router for Silicon Photonic Networks-on-Chip. *IEEE Photonics Technology Letters*, 22(12):926–928, June 2010.

[21] Z. Li, M. Mohamed, X. Chen, H. Zhou, A. Mickelson, L. Shang and M. Vachharajani. Iris: A Hybrid Nanophotonic Network Design for High-performance and Low-power On-chip Communication. *J. Emerg. Technol. Comput. Syst.*, 7(2):8:1–8:22, July 2011.

[22] F. Lou, M. Moayedi Pour Fard, P. Liao, M. S. Hai, R. Priti, Y. Huangfu, C. Qiu, Q. Hao, Z. Wei and O. Liboiron-Ladouceur. Towards a centralized controller for silicon photonic MZI-based interconnects. In *2015 IEEE Optical Interconnects Conference (OI)*, pages 146–147, April 2015.

[23] Y. Xiong, F. G. de Magalhães, B. Radi, G. Nicolescu, F. Hessel and O. Liboiron-Ladouceur. Towards a Fast Centralized Controller for Integrated Silicon Photonic Multistage MZI-based Switches. In *Optical Fiber Communication Conference*, page W1J.2. Optical Society of America, 2016.

[24] Huawei Technologies. White Paper - Huawei Observation to NFV. Technical Report 399662, 2014.

[25] Ericsson AB. Network Functions Virtualization and Software Management. Technical Report Uen 284 23-3248, 2014.

[26] R. Orobtchouk. *On Chip Optical Waveguide Interconnect: the Problem of the In/Out Coupling*, pages 263–290. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[27] W. Bogaerts, P. Dumon, D. Taillaert, V. Wiaux, S. Beckx, B. Luyssaert, J. Van Campenhout, D. Van Thourhout and R. Baets. SOI nanophotonic waveguide structures fabricated with deep UV lithography. *Photonics and Nanostructures - Fundamentals and Applications*, 2(2):81 – 86, 2004.

[28] M. Kawachi. Recent progress in silica-based planar lightwave circuits on silicon. *IEE Proceedings - Optoelectronics*, 143(5):257–262, Oct 1996.

[29] X. Meng, V. Depauw, G. Gomard, O. El Daif, C. Trompoukis, E. Drouard, C. Jamois, A. Fave, F. Dross, I. Gordon and C. Seassal. Design, fabrication and optical characterization of photonic crystal assisted thin film monocrystalline-silicon solar cells. *Opt. Express*, 20(S4):A465–A475, Jul 2012.

[30] F. Ay and A. Aydinli. Comparative investigation of hydrogen bonding in silicon based PECVD grown dielectrics for optical waveguides . *Optical Materials*, 26(1):33 – 46, 2004.

[31] I. O'Connor. Optical solutions for system-level interconnect. In *Proceedings of the 2004 International Workshop on System Level Interconnect Prediction*, SLIP '04, pages 79–88, New York, NY, USA, 2004. ACM.

[32] R. Paschotta. Signal to noise ratio. *Encyclopedia of Laser Physics and Technology*, 1, Oct 2008.

[33] P Hariharan. *Basics of interferometry*. Elsevier Academic Press, Amsterdam Boston, 2007.

[34] S. Kumar, S. Raghuwanshi and A. Kumar. Implementation of optical switches using Mach–Zehnder interferometer. *Optical Engineering*, 52(9):097106–097106, 2013.

[35] M.K. Chin and S.T. Ho. Design and modeling of waveguide-coupled single-mode microring resonators. *Lightwave Technology, Journal of*, 16(8):1433–1446, Aug 1998.

[36] S. Le Beux, J. Trajkovic, I. O'Connor, G. Nicolescu, G. Bois and P. Paulin. Optical Ring Network-on-Chip (ORNoC): Architecture and design methodology. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011.

[37] A. Shacham, B.G. Lee, A. Biberman, K. Bergman and L.P. Carloni. Photonic NoC for DMA Communications in Chip Multiprocessors. In *15th Annual IEEE Symposium on High-Performance Interconnects, 2007. HOTI 2007.*, pages 29–38, Aug 2007.

[38] S. Pasricha and S. Bahirat. OPAL: A multi-layer hybrid photonic NoC for 3D ICs. In *2011 16th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 345–350, Jan 2011.

[39] S. Koohi, A. Shafaei and S. Hessabi. An Optical Wavelength Switching Architecture for a High-Performance Low-Power Photonic NoC. In *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA)*, pages 1–6, March 2011.

[40] Z. Li, A. Qouneh, M. Joshi, W. Zhang, X. Fu and T. Li. Aurora: A Cross-Layer Solution for Thermally Resilient Photonic Network-on-Chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, PP(99):1–1, 2014.

[41] W.P. Boothroyd and E.M. Creamer. A time division multiplexing system. *Transactions of the American Institute of Electrical Engineers*, 68(1):92–97, July 1949.

[42] A.C. Lantz and B. Mukherjee. Efficient and modified round-robin protocols for fiber optic networks. In *Communications, 1990. ICC '90, Including Supercomm Technical Sessions. SUPERCOMM/ICC '90. Conference Record., IEEE International Conference on*, pages 1687–1691 vol.4, Apr 1990.

[43] J. Carson. The emergence of stacked 3D silicon and its impact on microelectronics systems integration. In *Proceedings of the Eighth Annual IEEE International Conference on Innovative Systems in Silicon, 1996*, pages 1–8, Oct 1996.

[44] M.G. Smith and S. Emanuel. Methods of making thru-connections in semiconductor wafers, September 26 1967. US Patent 3,343,256.

[45] S.J. Ben Yoo, B. Guan and R.P. Scott. Heterogeneous 2D/3D photonic integrated microsystems. In *Microsystems & Nanoengineering*, volume 2, page 16030, 2016.

[46] C. White. *Data communications and computer networks : a business user's approach*. Thomson Course Technology, Boston, Mass, 2007.

[47] J. Chan and K. Bergman. Photonic interconnection network architectures using wavelength-selective spatial routing for chip-scale communications. *IEEE/OSA Journal of Optical Communications and Networking*, 4(3):189–201, March 2012.

[48] G. Booch. *The unified modeling language user guide*. Addison-Wesley, Upper Saddle River, NJ, 2005.

[49] M. Elhaji, P. Boulet, A. Zitouni, S. Meftali, J. Dekeyser and R. Tourki. System Level Modeling Methodology of NoC Design from UML-MARTE to VHDL. *Des. Autom. Embedded Syst.*, 16(4):161–187, November 2012.

[50] M. Zhou. *Modeling, simulation, and control of flexible manufacturing systems : a Petri net approach*. World Scientific, Singapore River Edge, NJ, 1999.

[51] J. L. Rosenfeld. *Information processing 74 : proceedings of IFIP Congress 74*. North-Holland American Elsevier, Amsterdam New York, 1974.

[52] Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, July 1997.

[53] P.H. Feiler, D.P. Gluch, and J.J. Hudak. The Architecture Analysis & Design Language (AADL): An Introduction. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, 2006.

[54] Peter J. Ashenden. *The Designer's Guide to VHDL, Volume 3, Third Edition (Systems on Silicon) (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3 edition, 2008.

[55] S. Palnitkar. *Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition*. Prentice Hall Press, Upper Saddle River, NJ, USA, second edition, 2003.

[56] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 2000.

[57] J. Gosling, B. Joy, G.L. Steele Jr., G. Bracha and A. Buckley. *The Java Language Specification, Java SE 7 Edition*. Addison-Wesley Professional, 1st edition, 2013.

[58] Lumerical tools - https://www.lumerical.com/tcad-products/ - last access on 03/2017.

[59] Vpicomponentmaker photonic circuits - http://www.vpiphotonics.com/applications/photoniccircuits/ - last access on 03/2017.

[60] Optiwave: Design software for photonics - http://optiwave.com/ - last access on 03/2017.

[61] Matlab - http://www.mathworks.com/products/matlab/ - last access on 03/2017.

[62] Synopsys optical solutions: Optsim - http://optics.synopsys.com/rsoft/rsoft-system-network-optsim.html - last access on 03/2017.

[63] Optilux: and open source of light - http://optilux.sourceforge.net/ - last access on 03/2017.

[64] Y. Xie, M. Nikdast, J. Xu, W. Zhang, Q. Li, X. Wu, Y. Ye, X. Wang and W. Liu. Crosstalk noise and bit error rate analysis for optical network-on-chip. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 657–660, June 2010.

[65] Q. Shixiong, W. Kun, G. Huaxi, W. Kang and W. Xiaolu. Crosstalk analysis for closed ring-based optical network-on-chip. In *2015 IEEE International Conference on Communication Problem-Solving (ICCP)*, pages 331–333, Oct 2015.

[66] J. Chan, G. Hendry, A. Biberman, K. Bergman and L.P. Carloni. PhoenixSim: A simulator for physical-layer analysis of chip-scale photonic interconnection networks. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 691–696, March 2010.

[67] György Pongor. OMNeT: Objective Modular Network Testbed. In *MASCOTS '93: Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, pages 323–326, San Diego, CA, USA, 1993. The Society for Computer Simulation, International.

[68] J.E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. GRAPHITE: A distributed parallel simulator for multicores. In *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12, Jan 2010.

[69] C. Sun, C.H.O. Chen, G. Kurian, W. Lan, J. Miller, A. Agarwal, P. Li-Shiuan and V. Stojanovic. DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling. In *2012 Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pages 201–210, May 2012.

[70] N. Jiang, D.U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D.E. Shaw, J. Kim and W.J. Dally. A detailed and flexible cycle-accurate Network-on-Chip simulator. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 86–96, April 2013.

[71] P. Bhojwani, R. Mahapatra, J.K. Eun and T. Chen. A heuristic for peak power constrained design of network-on-chip (NoC) based multimode systems. In *VLSI Design, 2005. 18th International Conference on*, pages 124–129, Jan 2005.

[72] H. Jia, Y. Zhao, L. Zhang, Q. Chen, J. Ding, X. Fu and L. Yang. Five-Port Optical Router Based on Microring Switches for Photonic Networks-on-Chip. *IEEE Photonics Technology Letters*, 25(5):492–495, March 2013.

[73] Z. Li and T. Li. ESPN: A case for energy-star photonic on-chip network. In *2013 IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 377–382, Sept 2013.

[74] M. M. K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill and D.A. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, November 2005.

[75] H.A. Khouzani, S. Koohi and S. Hessabi. Fully contention-free optical NoC based on wavelenght routing. In *2012 16th CSI International Symposium on Computer Architecture and Digital Systems (CADS)*, pages 81–86, May 2012.

[76] M. Briere, B. Girodias, Y. Bouchebaba, G. Nicolescu, F. Mieyeville, F. Gaffiot and I. O'Connor. System Level Assessment of an Optical NoC in an MPSoC Platform. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, April 2007.

[77] J. Wang, B. Li, Q. Feng and W. Dou. A Highly Scalable Butterfly-Based Photonic Network-on-Chip. In *2012 IEEE 12th International Conference on Computer and Information Technology (CIT)*, pages 33–37, Oct 2012.

[78] H. Yang, V. Akella, C. N. Chuah and S. J. B. Yoo. Design of Novel Optical Router Controller and Arbiter Capable of Asynchronous, Variable length Packet Switching. In *International Conference on Photonics in Switching, 2006. PS '06.*, pages 1–3, Oct 2006.

[79] M. Shoaib. Selectively Weighted Multicast Scheduling Designs For Input-Queued Switches. In *2007 IEEE International Symposium on Signal Processing and Information Technology*, pages 92–97, Dec 2007.

[80] Y. Liu, M. T. Hill, H. de Waardt, G. D. Khoe and H. J. S. Dorren. All-optical buffering using laser neural networks. *IEEE Photonics Technology Letters*, 15(4):596–598, April 2003.

[81] D.K. Hunter, M.C. Chia and I. Andonovic. Buffering in optical packet switches. *Journal of Lightwave Technology*, 16(12):2081–2094, Dec 1998.

[82] M. Renaud, C. Janz, P. Gambini and C. Guillemot. Transparent optical packet switching: The European ACTS KEOPS project approach. In *IEEE Lasers and Electro-Optics Society 1999 12th Annual Meeting*, 1999.

[83] T. Sakamoto, K. Noguchi, R. Sato, A. Okada, Y. Sakai and M. Matsuoka. Variable optical delay circuit using wavelength converters. *Electronics Letters*, 37(7):454–455, Mar 2001.

[84] Mode from lumerical tools - https://www.lumerical.com/tcad-products/mode/ - last access on 03/2017.

[85] S. Le Beux, I. O'Connor, G. Nicolescu, G. Bois and P. Paulin. Reduction Methods for Adapting Optical Network on Chip Topologies to 3D Architectures. *Microprocess. Microsyst.*, 37(1):87–98, February 2013.

[86] N. Jasika, N. Alispahic, A. Elma, K. Ilvana, L. Elma and N. Nosovic. Dijkstra's shortest path algorithm serial and parallel execution performance analysis. In *Proceedings of the 35th International Convention MIPRO*, May 2012.

[87] H. Gu, K. H. Mo, J. Xu and W. Zhang. A Low-power Low-cost Optical Router for Optical Networks-on-Chip in Multiprocessor Systems-on-Chip. In *2009 IEEE Computer Society Annual Symposium on VLSI*, pages 19–24, May 2009.

[88] V.E. Beneš. On rearrangeable threestage connecting networks. pages 1481–1492. Bell Syst. Tech. J., 1962.

[89] K. Tanizawa, K. Suzuki, S. Suda, H. Matsuura, K. Ikeda, S. Namiki and H. Kawashima. Silicon photonic 32x32 strictly-non-blocking blade switch and its full path characterization. In *2016 21st OptoElectronics and Communications Conference (OECC) held jointly with 2016 International Conference on Photonics in Switching (PS)*, pages 1–3, July 2016.

[90] F. G. de Magalhães, F. Hessel, O. Liboiron-Ladouceur and G. Nicolescu. Cluster-based architecture relying on Optical Integrated Networks with the provision of a low-latency arbiter]. In *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, Aug 2016.

[91] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 7(2):188–201, Apr 1999.

[92] Petros Mol, Todor Ristov, and Nikolaos Trogkanis. Throughput/Fairness Trade offs for the iSLIP Scheduling Algorithm. 2009.

[93] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North and G. Woodhull. Graphviz and dynagraph – static and dynamic graph drawing tools. In *GRAPH DRAWING SOFTWARE*, pages 127–148. Springer-Verlag, 2003.

[94] J. Leeuwen. *Handbook of theoretical computer science*. Elsevier MIT Press, Amsterdam New York Cambridge, Mass, 1994.

[95] B.H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, July 1970.

[96] *Handbook of Digital Techniques for High-Speed Design*. Pearson Education, 2007.

[97] DIA Diagram Editor - http://dia-installer.de/index.html.en - Last access on 12/2016.

[98] XILINX. http://www.xilinx.com - Last access on 12/2016, 2007.

[99] Altera Stratix IV - http://tinyurl.com/neqmmj3 - last access on 07/2016.

[100] Signaltap II Embedded Logic Analyzer - http://tinyurl.com/nhx69nj - last access on 07/2016.

[101] Y. Xiong, F. G. de Magalhães, G. Nicolescu, F. Hessel and O. Liboiron-Ladouceur. Co-design of a Low-latency Centralized Controller for Silicon Photonic Multistage MZI-based Switches. In *Optical Fiber Communication Conference*, page Th2A.37. Optical Society of America, 2017.

ANNEXE A IMPLEMENTATION DETAILS

The details regarding the implementation optical models as well as the SF-Sim usage are presented in this annex. Focusing in the 2×2 switching block, which is used to build all topologies illustrated in this thesis, the aspects of its behaviour are illustrated. Further, the steps taken for the SF-Sim to be exploited are presented, illustrating the entire description and configuration flow.

2x2 Switch

The 2×2 switching blocks, introduced in Figure 7.2 are used to build the network topologies in this thesis. Simplifying, the physical realization of optical switches involves waveguides and transmitted beans of light. The switching aspects of such optical blocks take into account their physical characteristics, as introduced in Section 2.1. Nevertheless, it is required a lot of computational power for a simulator to handle all those aspects, as they are mostly ruled by differential equations. In this context, the modelled optical devices use abstractions in order to emulate the optical behaviour, while maintaining simplicity. VHDL is used as description language, as it holds the closest relation to the hardware layer. Still, the same approach introduced in this annex can be employed to describe the optical blocks using different programming languages.

The implemented models take advantage of VHDL features in order to capture the emulated optical behaviours. For example, in order to emulate the transmission delay of the optical blocks, the synchronization characteristics provided by clocking signals in VHDL can be used. This way, it is possible to define a clock configured with the desired transmission delay and use it as transmission windows.

The described switches work by logically choosing the direction that inserted data must follow by using the *config* and *we* signals and based on their combination, writing the *input* port on the *output* port after a given delay. Table A.1 presents the logical used to select which input will be directed to each output, where X's mean the output is not being used. The following signals are employed:

- **config:** holds the switching block configuration, where '0' represents BAR state and '1' represents CROSS-BAR state;
- **we:** enables the transmission of the designed port. It is a two-bit signal, where the bit

zero is used as write enable of input 0 and bit one is used as write enable of input 1;

- **output:** is the switch output, which receives the inserted data. It is a two-bit signal, one bit for each output, and;
- **input:** the switch input. It is a two-bit signal: bit zero is the input 0 and bit one is the input 1.

Table A.1 2x2 Switch Logic Table

config	we(0)	we(1)	output(0)	output(1)
0	*0*	*0*	***X***	***X***
0	*0*	*1*	***X***	***input(1)***
0	*1*	*0*	***input(0)***	***X***
0	*1*	*1*	***input(0)***	***input(1)***
1	*0*	*0*	***X***	***X***
1	*0*	*1*	***X***	***input(0)***
1	*1*	*0*	***input(1)***	***X***
1	*1*	*1*	***input(1)***	***input(0)***

Figure A.1 illustrates the overview of the 2×2 switching block and its control and I/O signals. In the figure, it is possible to see the two inputs, the control bit (*config*), the write enable bits (*we(0)* and *we(1)*) and the switch outputs.

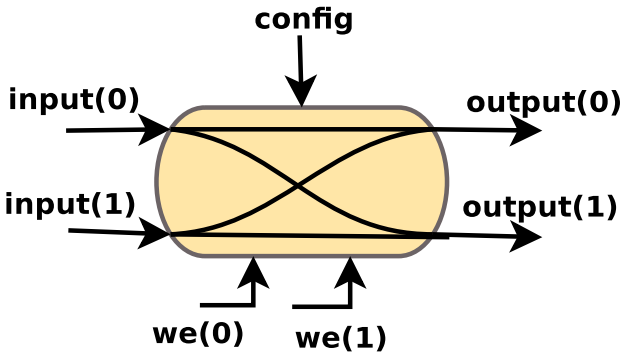


Figure A.1 2×2 switch block overview.

Figure A.2 illustrates a finite state machine diagram (FSM) that shows the required steps the switch selection mechanism should perform: **(i)** while *we* signal is low (two bits), the switch is idle; **(ii)** if *we* gets high for any bit, the FSM moves for the next stage: pick the correct output, and; **(iii)** while the *we* signal stays high, the switch keeps transmitting, where each output receives its related input[1].

[1]the following nomenclature is used: **(wr01)** means that output zero receives input one at the same time

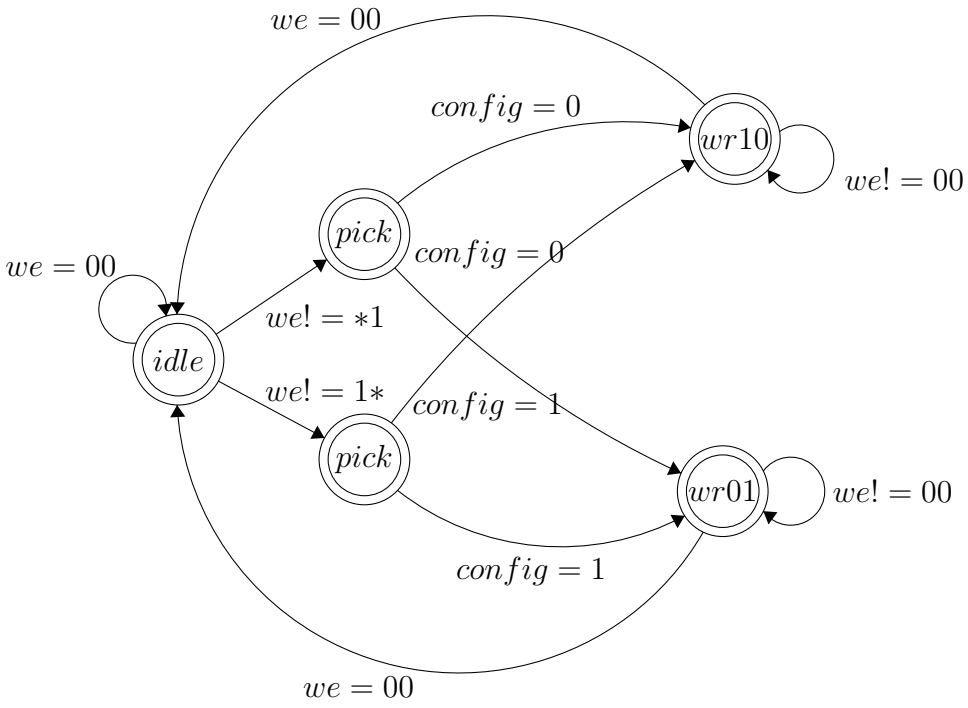


Figure A.2 2×2 switch output selection logic

The Listing A.1 presents the 2×2 MZI-based switch instantiation (in VHDL), where the signals used to interact with the MZI are shown. In this model, the following signals are employed:

- *reset:* as this model is described as a digital block, this signal resets all involved signals of the block;
- *clk:* used as the synchronization signal. Is configured based on the desired transmission delay of the optical block;
- *input:* receives the incoming data;
- *output:* receives the inserted data from its paired input port;
- *we:* enables the transmission of the designed port;
- *config:* holds the switching block configuration, where '0' represents BAR state and '1' represents CROSS-BAR state;

that output one received inputs zero, and; **(wr10)** means that output one receives input one at the same time that output zero receives input zero.

- *sending:* signalizes that the switch has data coming out. It is used when connecting several switches, so they can synchronize the transmission;
- *i_pwr:* input optical power, and;
- *o_pwr:* output optical power, after the involved losses in the transmission.

Listing A.1 2×2 MZI-based Switch instantiation in VHDL

```
1  entity switch_2x2 is
2          generic(
3                  power_count: string      :=       "ON"
4          );
5          port (
6                  reset             : in std_logic;
7
8                  i_we              : in std_logic_vector(1 downto 0);
9                  config            : in std_logic;
10                 input             : in std_logic_vector(1 downto 0);
11
12                 o_we              : out std_logic_vector(1 downto 0);
13                 output            : out std_logic_vector(1 downto 0);
14
15                 i_pwr             : in SWITCH_POWER;
16                 o_pwr             : out SWITCH_POWER
17         );
18 end switch_2x2;
```

Hereafter, having the internal switch logic defined, the design is translated into logical ports (NORs, ANDs, ORs, Muxes, Flip-Flops, etc) so it can be simulated. Figure A.3 presents the final 2×2 Switch internal organization and connections. All other components are built over this basic block.

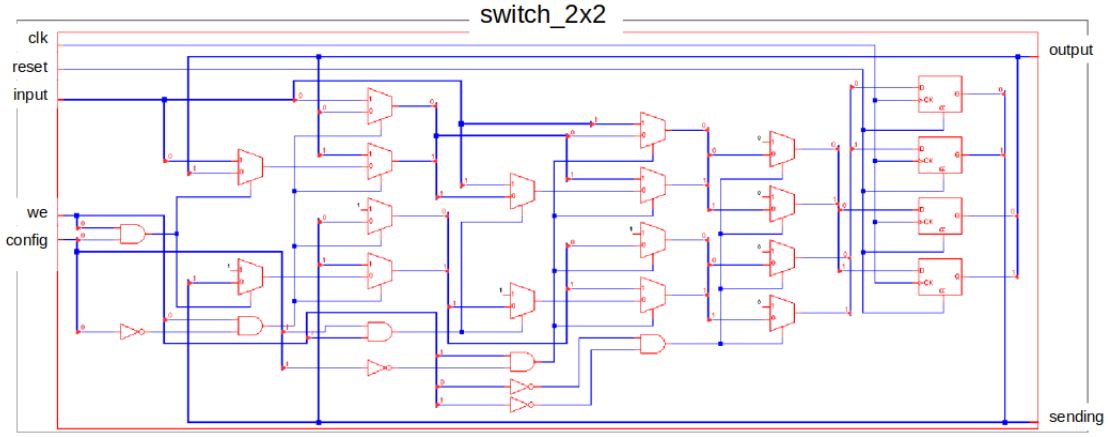


Figure A.3 Logical Block of Described 2×2 MZI-based Switch

VHDL Package

In order to ease the usage of the described models, a VHDL package file was created so basic simulation configurations could be written directly to it, facilitating future deployments. Listing A.2 presents the created package with types, constants and function definitions, which are going to be used from this point on.

Listing A.2 Package with definitions in VHDL

```
package sim_pack is
constant ATOMIC_UNITS: integer := 5;            --number of 2x2 blocks
constant MZI_BAR_DELAY  : time := 4 ps;         --component transmission delay (SIM)
constant MZI_CROSS_DELAY        : time := 4 ps;    --component transmission delay (SIM)
constant MR_BAR_DELAY   : time := 4 ps;         --component transmission delay (SIM)
constant MR_CROSS_DELAY : time := 4 ps;         --component transmission delay (SIM)
constant MZI_PERIOD: time := MZI_CROSS_DELAY/2; --defined period for internal switch delay (Emu)
constant MR_PERIOD: time := MR_CROSS_DELAY/2;   --defined period for internal switch delay (Emu)
constant IP_PERIOD: time := 10 ns;              --defined period for IP simulation

--FLIT DEFINITION
constant MESSAGE_SIZE: integer := 16;           --flit size, in bits
subtype MESSAGE is std_logic_vector(MESSAGE_SIZE-1 downto 0); --one flit

constant MESSAGE_PACKAGE: integer := 10;        --package size, in flits
type MESSAGE_ARRAY is array(MESSAGE_PACKAGE-1 downto 0) of MESSAGE; --package array
type MESSAGES_INPUT     is array (NETWORK_IOS-1 downto 0) of MESSAGE;

--SERDES IN/OUT RATE
constant SERDES_RATE: time := (IP_PERIOD/MESSAGE_SIZE) -  MZI_BAR_DELAY;

--POWER DEFINITIONS (mW)
type POWER_MEASURE is range 0.0 to 99999999.0;  --type definition
type NET_POWER is array (NETWORK_IOS-1 downto 0) of POWER_MEASURE; --network power
constant MZI_INS_LOSS       :   POWER_MEASURE := 0.794328; --MZI insertion loss, in %
constant MR_INS_LOSS        :   POWER_MEASURE := 0.794328; --MR insertion loss, in %
constant MZI_BAR_LOSS   :       POWER_MEASURE := 0.794328; --MZI BAR transmission insertion loss, in %
constant MZI_CROSS_LOSS :       POWER_MEASURE := 0.794328; --MZI CROSS transmission insertion loss, in %
constant MR_BAR_LOSS    :       POWER_MEASURE := 0.794328; --MR BAR transmission insertion loss, in %
constant MR_CROSS_LOSS  :       POWER_MEASURE := 0.794328; --MR CROSS transmission insertion loss, in %
constant LASER_INPUT_POWER      :       POWER_MEASURE := 15.0; --power of input laser
end sim_pack;
```

SF-Sim Design Flow

This section presents the design flow when using the SF-Sim to deploys a system. A 4×4 topology will be described.

First, the general simulation parameters should be configured, as presented in Figure A.4.

- *Number of IOs:* defines the number of IP nodes in the system. In the figure, four IPs will be connected to the configured network.
- *IPs Period:* the IPs execution period, consequently frequency. In this case, 10 ns or 100 MHz.
- *Flit Size:* is the number of bits of each transmitted package. In the figure, each flit is configured to have 16 bits.

- *Message Size:* is the number of transmitted packages. In this case, 10 flits are transmitted. As each flit is configured with 16 bit, in total 160 bits are to be transmitted.
- *Laser Input Power:* defines the available laser power on each network input.

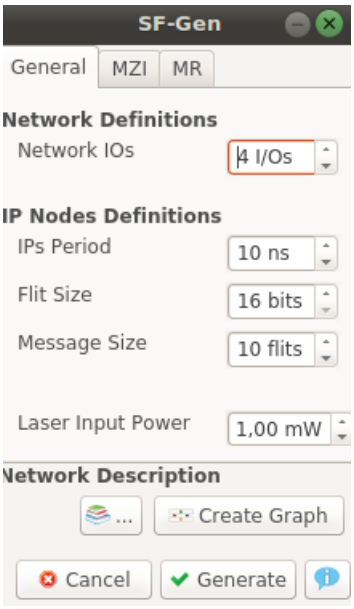


Figure A.4 SF-Sim GUI Configuration Window.

Next, the MZI and MR configuration is performed. Dynamic parameters should be configured, as presented in Figure A.5.

- *BAR Transmission Delay:* defines the transmission delay, when in BAR state. Parameter in pico-seconds.
- *CROSS Transmission Delay:* defines the transmission delay, when in CROSS state. Parameter also in pico-seconds.
- *BAR Transmission Loss:* is the transmission loss, when the switch is configured to BAR state. The loss is defined in dB.
- *CROSS Transmission Loss:* is the transmission loss, when the switch is configured to CROSS state. Again, the loss is defined in dB.
- *Coupling Loss:* defines the coupling loss for I/O switches. As the last two cases, the loss is defined in dB.

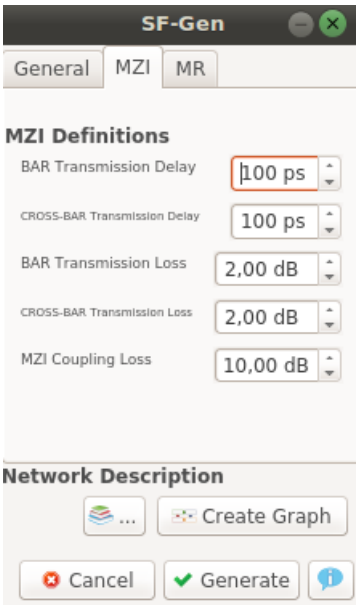


Figure A.5 SF-Sim GUI MZI configuration window.

After configuring the simulation parameters, the user should describe the system. To do so, the DIA tool is used. When the Create Graph button is triggered, the DIA interface is called. The Figure A.6 presents the DIA tool main window. In the initial screen, it is possible to see the available components in the left and the workbench in the right. To build the system the user can use the drag-and-drop feature and simply connect desired network nodes. The available components in the tool are the ones presented as building blocks in this thesis: 2×2 blocks (MR and MZI), 4×4 blocks (Beneš and Spanke-Beneš) as well as a strictly non-blocking 8×8 topology.

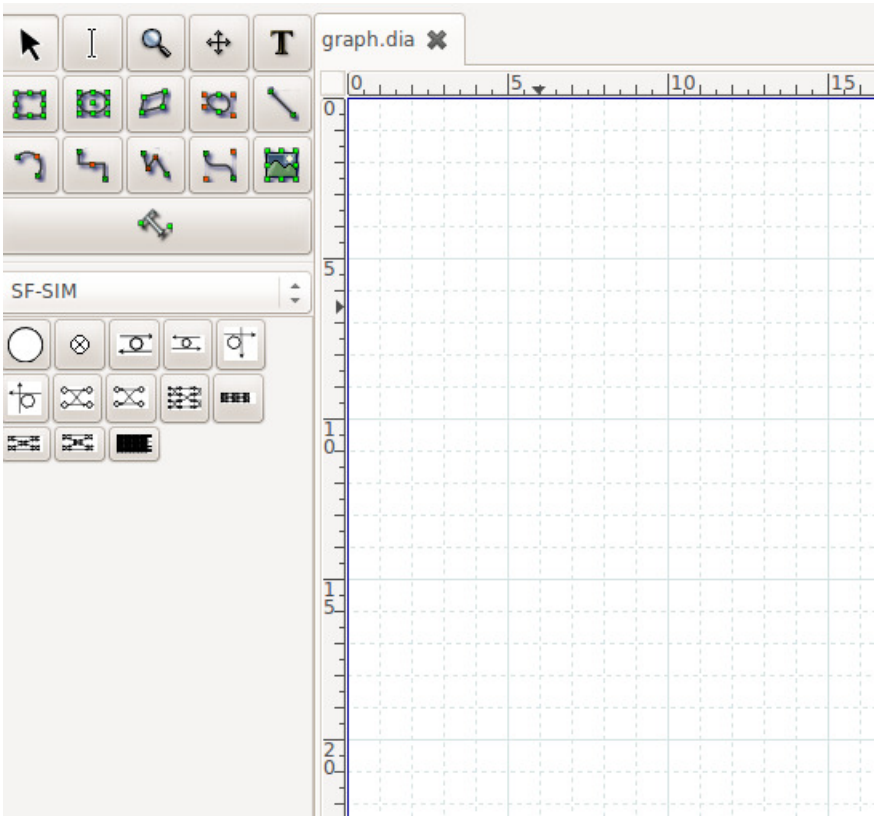


Figure A.6 SF-Sim GUI DIA configuration window.

Figure A.7 shows the workbench filled with 12 2×2 switches used to build the strictly non-blocking 4×4 Beneš topology. At this point no connection neither IPs are placed, only the network nodes.

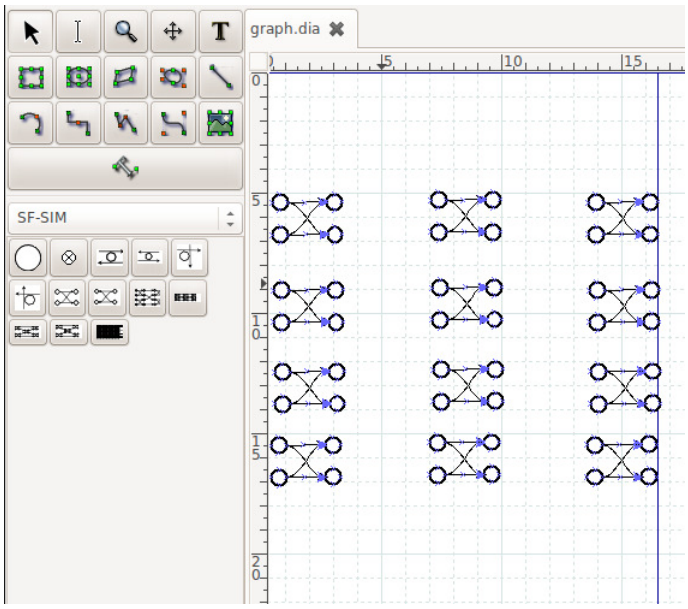


Figure A.7 SF-Sim GUI DIA Workbench.

Next, it is possible to connect the placed nodes in the workbench. To do so, the available arrows should be used and their direction respected. Figure A.8 illustrates the 12 2×2 switches connected to each other.

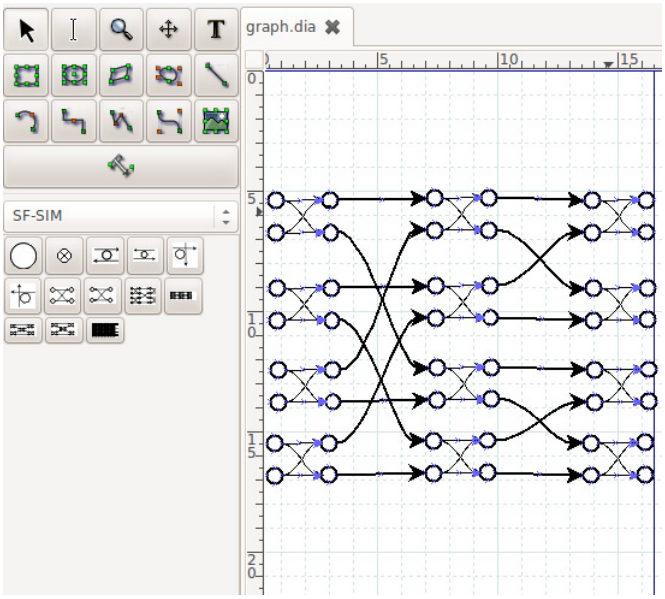


Figure A.8 SF-Sim GUI DIA Workbench Presenting Connected Nodes.

After, the I/O nodes should be placed and connected to their input and output switches. Figure A.9 shows the I/O nodes with their respective switches.

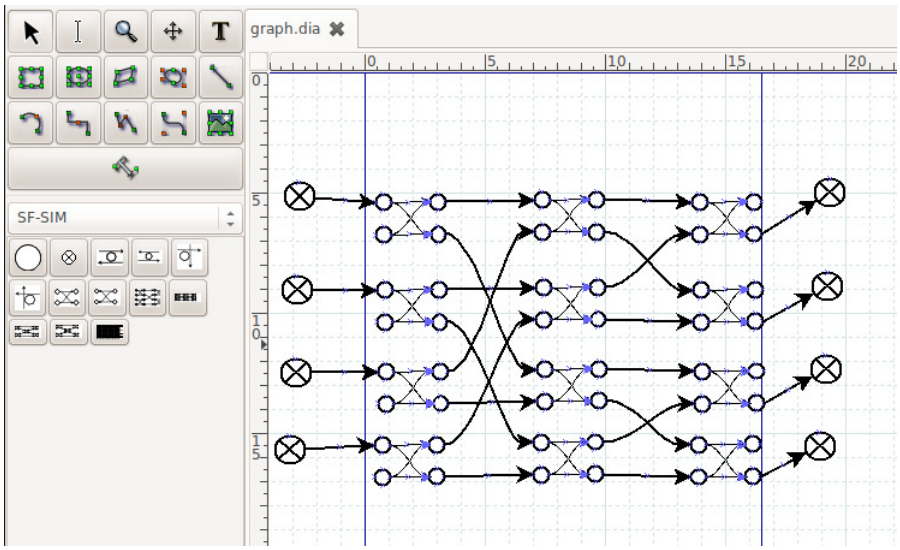


Figure A.9 SF-Sim GUI DIA Workbench I/O Nodes Placement.

Finally, the I/O nodes should be coloured in order to the proper I/O mapping to be set. Figure A.10 shows the coloured I/O nodes with their respective switches. In the figure, coloured nodes in the left are input nodes and coloured nodes in the right are output nodes.

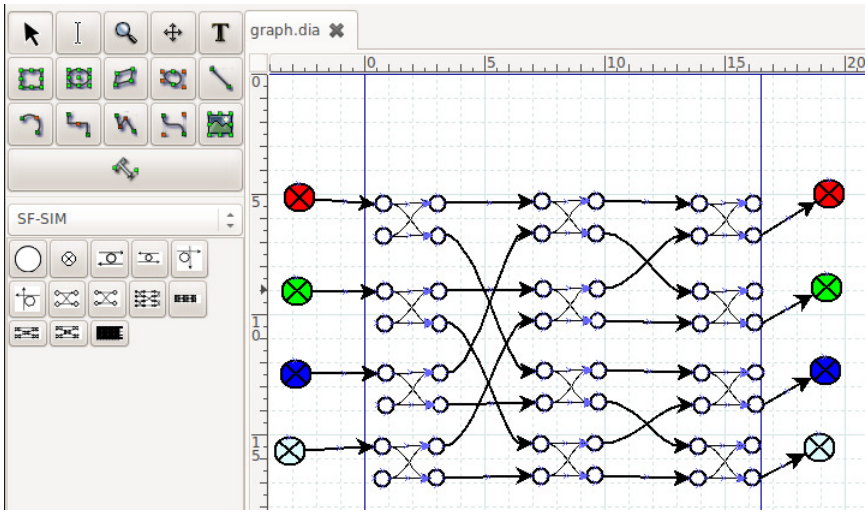


Figure A.10 SF-Sim GUI DIA Workbench I/O Nodes Colouring.

After all the system is described and configured, the user can trigger the Generate button. This will generate all needed files to simulate the created system, besides the HyCo files to control the same system. Figure A.11 presents the generated files. The following files are presented: the HyCo VHDL files, used to control the described network. The simulation files,

such as the MZI/MR switching blocks and the Modelsim script, to trigger the simulation. Further, the VHDL package, which holds all the configurations needed to simulate the system is created as well. The traffic generator, used to inject data in simulated networks and finally, the test bench files are created.

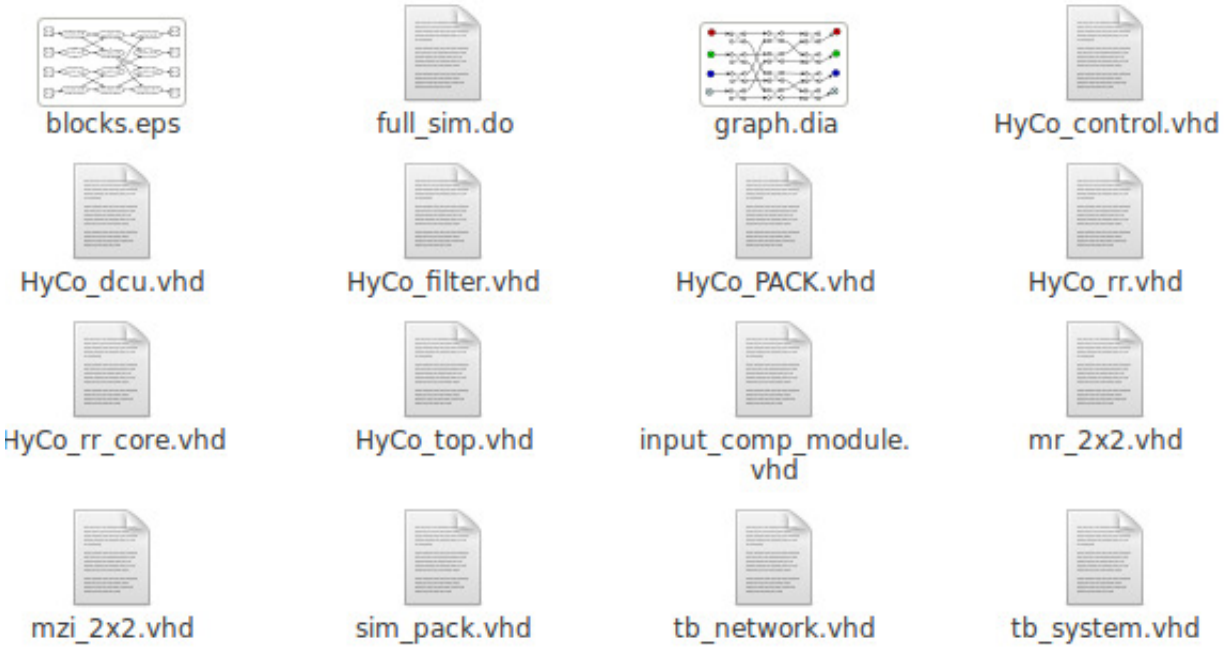


Figure A.11 SF-Sim Generated Files.