Réseaux d'interconnexion flexible pour architecture reconfigurable dynamiquement

Ludovic Devaux

N° d'ordre : 2011REN1E005 ANNÉE 2011

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de
DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention : Traitement du signal et télécommunications
Ecole doctorale MATISSE
présentée par

Ludovic Devaux

préparée à l'unité de recherche (6074 IRISA UMR)
(Institut de Recherche en Informatique et Systèmes Aléatoires)

Flexible interconnection networks for dynamically reconfigurable architectures

Thèse soutenue à Lannion le 24 novembre 2011

devant le jury composé de :

M. Frederic Pétrot
Professeur, ENSIMAG / président

M. Fernando Moraes
Professeur, PUCRS / rapporteur

M. Jean-philippe Diguet
Directeur de recherche CNRS, UBS / rapporteur

M. Fabrice Muller
Maître de conférence, UNS / examinateur

M. Daniel Chillet
Maître de conférence, ENSSAT / examinateur

M. Didier Demingy
Professeur, IUT Lannion / directeur de thèse

M. Sébastien Pillement
Maître de conférence
IUT Lannion / co-directeur de thèse

Abstract : Dynamic and partial reconfiguration allows to dynamically allocate the tasks constituting an application in the reconfigurable regions of an FPGA. However, dynamic management of the tasks directly impacts the communications since tasks are not always implemented at the same place in the FPGA. So, the communication architecture must support high flexibility and significant qualities of service (guaranteed bandwidth and/or latency).

In this PhD, several interconnection architectures were studied and evaluated regarding their compliance with a dynamically reconfigurable system implemented in FPGA. This study led to the proposal of the DRAFT network that fully supports this concept. This network uses some specificities of the dynamically reconfigurable systems to reduce its hardware resources consumption. Furthermore, if several constraints are verified, the performances are not influenced by the dynamic allocation of the tasks. A network generator, DRAGOON, is also presented in order to implement and simulate the DRAFT network.

Following the realization and characterization of this network which was compared with two very popular networks, its integration inside a system was studied. Consequently, a standard interface was designed in order to ease the interconnection of elements such as microprocessors. Considering the degree of complexity of the hardware parts of a reconfigurable system, an OS is often used to act as an abstraction layer. So, a service allowing to realize communications between the various elements of a system while providing a complete abstraction of the DRAFT network was designed in hardware.

Considering the various constraints on the use of DRAFT, the OCEAN network was proposed. This network allows a simple interconnection of the elements constituting a system with great flexibility. For this purpose, the OCEAN network is based on two sub-networks, one being dedicated to data transfers, while the other ensures its control. OCEAN network lies on dynamically created communication paths following applicative requirements. This network particularly targets ASIC implementations

All these networks were validated and characterized through experiments and implementations in FPGA. Results demonstrate the adequacy between proposed networks and the actual needs, also with the support of complex applications using dynamic reconfiguration. The OCEAN network even proposes an evolution toward future dynamic architectures.

Keywords : network-on-chip - FPGA - dynamic reconfiguration - flexible - ASIC

Résumé : La reconfiguration dynamique partielle permet de placer dynamiquement les tâches d'une application dans des zones reconfigurables d'un FPGA. Cependant, la gestion dynamique des tâches impacte les communications du fait que les tâches ne sont pas toujours allouées au même endroit dans le FPGA. Ainsi, l'architecture d'interconnexions doit supporter une grande flexibilité et un large éventail de qualité de service (bande passante ou latence garantie).

Dans cette thèse, plusieurs architectures d'interconnexion ont été étudiées et évaluées en fonction de leur compatibilité avec un système reconfigurable dynamiquement implémenté sur FPGA. Cette étude a conduit à proposer le réseau DRAFT qui supporte pleinement ce concept. Ce réseau utilise certaines spécificités des systèmes reconfigurables dynamiquement actuels pour réduire sa consommation de ressources. De plus, si certaines contraintes sont vérifiées, les performances ne sont pas affectées par l'allocation dynamique des tâches. Un générateur de réseaux, DRAGOON, est aussi présenté afin d'implémenter et de simuler le réseau DRAFT.

Suivant la réalisation et la caractérisation du réseau DRAFT qui a été comparé à deux réseaux très populaires, son intégration au sein d'un système a été étudiée. C'est ainsi qu'une interface standard a été développée afin de faciliter l'interconnexion d'éléments tels que des processeurs. Etant donné le degré de complexité des parties matérielles d'un système reconfigurable, un OS est souvent utilisé pour en permettre l'abstraction. Ainsi, un service de communication permettant de réaliser des échanges entre les différents éléments d'un système tout en ayant une abstraction totale du réseau DRAFT a été conçu matériellement.

Considérant les différentes contraintes liées à l'utilisation de DRAFT, le réseau OCEAN a été proposé. Ce réseau permet une simplification de l'interconnexion des éléments d'un système avec une très grande souplesse d'utilisation. Ce réseau est pour cela basé sur deux sous- réseaux, l'un étant dédié au transport des données tandis que l'autre en assure le contrôle. Le réseau OCEAN repose sur des chemins de communication créés dynamiquement en fonctions des besoins. Ce réseau dynamique vise plutôt une cible ASIC.

L'ensemble des réseaux proposés ont été validés et caractérisés au travers d'expériences et d'implantations sur FPGA. Les résultats montrent une adéquation avec les besoins actuels, et le support efficace de la dynamicité des applications complexes. Le réseau OCEAN propose même une évolution pour de futures architectures dynamique.

Mots-clés : réseau sur puce - FPGA - reconfiguration dynamique - flexible - ASIC

Summary

Abbreviations and Acronyms

AHB Advanced High-performance Bus

AMBA Advanced Microcontroller Bus Architecture

API Application Programming Interface

ASIC Application-Specific Integrated Circuit

BM Bus Macro

CAD Computer Assisted Design

CLB Configurable Logical Block

CMOS Complementary Metal Oxide Semiconductor

DCM Digital Clock Manager

DMA Direct Memory Access

DPR Dynamic and Partial Reconfiguration

DRAFT Dynamic Reconfiguration Adapted Fat-Tree

DRAGOON Dynamically Reconfigurable Architectures compliant Generator and simulatOr Of Network

DSP Digital Signal Processor

FIFO First In First Out

FOSFOR Flexible Operating System FOr Reconfigurable devices

FPGA Field Programmable Gate Array

FSM Finite State Machine

GALS Globally Asynchronous Locally Synchronous

ICAP Internal Configuration Access Port

IO Input Output

IP Intellectual Property

LUT Look Up Table

NI Network Interface

NoC Network-on-Chip

OCEAN On-Chip Efficiently Adaptive Network

OPB On-chip Peripheral Bus

OS Operating System

PE Processing element

PLB Processor Local Bus

PRR Partially Reconfigurable Region

QoS Quality of Service

R2NoC Reconfigurable Routers based Network-on-Chip

RAM Random Access Memory

RSoC Reconfigurable System-on-Chip

RTL Register Transfer Level

SoC System-on-Chip

UART Universal Asynchronous Receiver Transmitter

VC Virtual Chanel

VHDL VHSIC Hardware Description Language

VHSIC Very-High-Speed Integrated Circuits

VLSI Very Large Scale Integration

Introduction

Since the very beginning of electronics in 1904, (year of ”electronic tube” invention, the ancestor of the transistor), the complexity of electronic systems never ended increasing. Over the years, more and more powerful systems were created. Indeed, this computing power, which can be expressed as the number of operations per time unit, continuously increases along with the needs of our modern society. This research of more computing power led to the miniaturization process. The objective is simple, since the speed of an electron is constant in a metal wire or in a transistor, miniaturization reduces their length so that the electrons reach more rapidly their destinations. So, each operation to be realized takes less and less time in a smaller and smaller electronic system. It is then possible to realize more operations in the same time unit. The fact that electronic systems are now ubiquitous in our life is a direct consequence of this research of computing performances.

A digital electronic system is always composed of the same sets of elements. First are the computing elements that can be processors running user defined software algorithms, or blocks performing all the processing of some incoming data in hardware. If the function computed by latter blocks can be modified, they are called reconfigurable. The second set concerns the storage capacities. Indeed, every application requiring an electronic system embeds computing elements that process data. Consequently these data should be memorized in order for the processing elements to pick them up before processing, and then to store results. Finally, the last set concerns the communication. In every system, a communication medium is required to make the interconnection of the computing element(s) and the storage one(s). Furthermore, interfaces are also necessary in order for the system to take information (data) from their environment. These interfaces are often considered as computing elements providing data by the communication architecture. For convenience purpose, every interconnected element, either processors, computing blocks, memories, and communication interfaces are denominated as Processing Elements (PEs) since they participate to the processing of the overall application. First digital electronic systems were composed of one PE per chip. However, following the miniaturization process, chips have started to embed all the elements required to run the application.

Thus, a single circuit can now embed one or several processors, computing blocks, memories, communication architectures and interfaces. Communications were usually realized through metal wires. This is the definition of a System-on-Chip (SoC).

There are several ways to implement a SoC. The first one consists in designing the system directly in silicon. The resulting chip is then dedicated to the originally aimed application even if an embedded microprocessor can bring a certain flexibility: software codes can evolve in the time. This implementation way, called Application-Specific Integrated Circuit (ASIC) guarantees high performances but no flexibility in the sense that no change can happen to the hardware architecture of the application. The second way to implement a SoC consists in configuring a general purpose architecture. The principle is simple: the architecture embeds logical resources that can be configured to realize a function in hardware. Using these configurable resources and the interconnection wires between them, it is possible to produce the behaviour of all the elements constituting a SoC. At the cost of a loss in performances, the flexibility is thus increased. Some chips like the Programmable Array Logics (PALs) are one time programmable, they are usually used as chips performing a permanent function but at a lower cost than ASICs. Indeed, they do not embed sufficient programmable resources to implement a complete application. Most of SoCs are implemented using reconfigurable chips called Field Programmable Gate Arrays (FPGAs). FPGAs have the particularity that embedded hardware resources are programmed at boot time. So, depending on the configuration that is applied to the FPGA at the power-up, different applications can be implemented on the same chip. This leads to the definition of a Reconfigurable System-on-Chip (RSoC).

Originally FPGAs were mostly used to prototype ASIC implementations. However, their reconfiguration ability leads them to be used inside industrial finite products. Reconfiguration of a chip is a key feature of present and future systems. Indeed, when an application requires a wide range of performances that can only be attained through hardware implementation, the reconfiguration takes all its sense. The reconfiguration is particularly useful at design time in order to reduce the financial and temporal costs to verify the proper behaviour of the system: one or several reconfigurable chips are used. PEs can then be configured inside these chips and reconfigured if an unexpected behaviour is detected. If chips were not reconfigurable, then each time an error is detected a new chip would be designed. With a reconfigurable circuit, the new PE is configured instead of the previous faulty one, thus saving both time and money. If originally reconfigurable circuits were used for design assistance purpose, they have demonstrated their interest in real life applications. Indeed they allow the system to evolve in the time. Thanks to the reconfiguration, PEs can be updated without any human direct intervention on the system, and furthermore without changing any physical element (board, chip, etc.). Completely changing the behaviour of a system in order to re-use its components, providing a better Quality of Service (QoS) reconfiguring a PE for fault tolerance purpose are also key functionalities that are allowed thanks to the reconfiguration ability of the chips.

Originally, the reconfiguration of a chip requires halting the execution of the whole chip in order to configure every one of their logical elements. This is called the static or total reconfiguration. This is the case for most of current and past FPGAs. However, steady technological, scientific, commercial, financial, and finally human evolutions lead desired application to become more and more complex every day. This complexity attains such a level that required hardware resources to support these applications overcome the capacity of present chips. More and more, single chips are not powerful enough to handle present applications. This is why several technological ways are explored. Facing the problem of the resources limitations, three main paths are followed by both researchers and industrials:

- the conception and design of larger and larger chips embedding more and more configurable resources,
- the use of several chips in parallel allowing to multiply the actions that can be realized at a time,
- and the dynamic reconfiguration ability of the circuits.

The conception of larger and larger chips leads to more and more complex designs. Designs become then both hard to debug and to maintain. This leads to the proposal of new Computer Assisted Design (CAD) tools in order to ease both the conception and the maintenance of complex applications. However, the use of several chips is based on the parallelism of the applications. Indeed, following the observation that usually applications are composed of several functions, it is possible to improve performances processing these functions at the same time while using several chips of reduced size. However, new mechanisms are required to guarantee a valid behaviour of the application while exploiting as much as possible the parallelism. Thus, both an Operating System (OS) and CAD tools are required in order to give to the designer an abstraction of the physical components and their operation that steadily becomes more complex.

A new feature appeared recently in the world of FPGAs. This is the Dynamic and Partial Reconfiguration (DPR). Originally motivated by the constant increase of SoCs' complexity, the DPR of an FPGA consists in the possibility to reconfigure not only the whole chip at boot time, but only a part of embedded logical resources during the execution of the application. If previously more and more complex applications required larger and larger SoCs (and thus chips), the DPR offers the possibility to sequentialize the applications. Indeed, every application can be divided into several tasks. These tasks can be processed either in software (running on a hardware processor), or directly in hardware using a computing block. As illustrated in Figure 1, storage elements, communication interfaces and an interconnection architecture may also be required for the whole RSoC to operate properly. However, if the size of the RSoC is too limited to implement the microprocessor and the two computing blocks at a time, the DPR offers the possibility to swap a hardware element by another one. Obviously, the DPR is efficient only if all the tasks do not need to be executed simultaneously. In this example, all logical resources that are used by both the microprocessor and the computing block dedicated to the task 3 can be dynamically allocated at a different time to a different computing block: the one dedicated to the task 2.

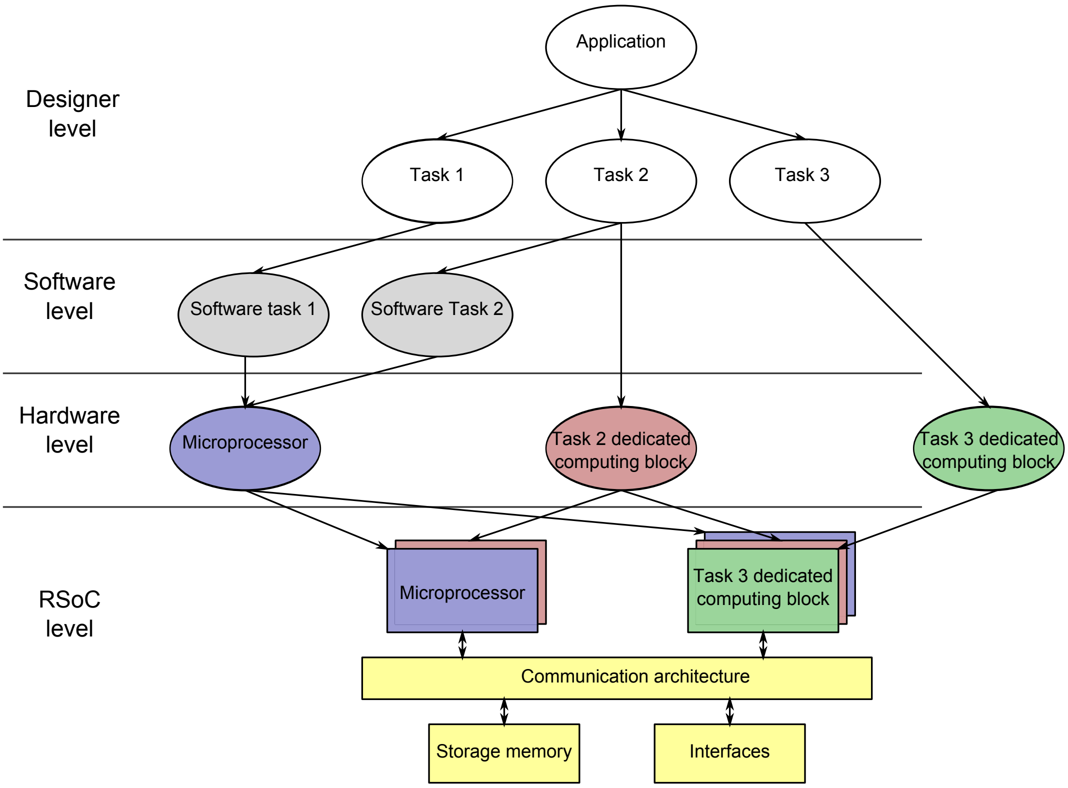


Figure 1 : Hierarchization of an application. An application can be divided into several tasks at the designer level. Some tasks can then be implemented in software, or directly in hardware. Software implemented tasks are processed on a hardware processor while hardware implemented ones lead to dedicated computing blocks. All these hardware elements are allocated inside a RSoC along with storage and communication elements. The resources are shared in the time.

The DPR allows reducing the size of the RSoCs, and thus their complexity. However, this concept of dynamic hardware implementation of the tasks has a significant impact over both the design and the functioning of a SoC. Indeed, the allocation of hardware components of an application needs now to be scheduled, and their placement must be controlled. Furthermore, it also impacts the communication architecture. From the communication architecture point of view, every interconnected element has specific communication requirements in terms of amount of data that must be transferred, and of maximum time allowed to transfer these data. In SoCs, it was possible to design a communication architecture optimally considering the communication requirements inherent to each PE. However, due to the DPR, PEs can be exchanged by others with different communication needs. Furthermore the same PE is not guaranteed to be always located at the same place in the RSoC. Indeed there is no obstacle for a PE to be relocated at a new place depending on the applicative context. All of this avoids any possibility for the resulting data traffic or the communication infrastructure to be predicted at design time.

If computing elements perform all the operations and make all the decisions, most of present digital systems are limited by their interconnection [33]. Until the late 1990s, the traditional way of interconnecting PEs such as computing blocks, microprocessors and memories was to use dedicated links to minimize latencies or shared buses for simplicity. With subsequent increases in number of interconnected PEs on a single chip along with the length and delay of wires to cross a chip, it has become important to share on-Chip interconnect bandwidth in a more structured way. This leads to the notion of Networks-on-Chip (NoCs). Since then, numerous NoCs have been proposed but the introduction of the DPR in RSoCs reveals a lack of flexibility for lot of them. Communication architectures thus become a major problem when designing RSoCs. This is why the research work presented in this report proposes new flexible interconnection architectures. This work took place into the scope of a PhD financed by the *Agence Nationale de la Recherche* (ANR) through the Flexible Operating System FOr Reconfigurable devices (FOSFOR) project.

Applicative context

DPR compliant FPGAs

The three main FPGA manufacturers are Xilinx, Altera and Atmel. There are also small societies that take advantage of niche markets like Achronix. Achronix provides asynchronous FPGAs where resources are distributed in small tiles working in pipelined fashion [1]. However, only a small range of FPGAs are currently available and the support of the DPR is not foreseen in their business plan. Atmel devices propose large varieties of FPGAs with a regular intrinsic structure [14]. However, when compared with Altera and Xilinx devices, the number of reconfigurable resources is too limited to handle the implementation of large and complex applications. At present time, Xilinx is the market leader concerning FPGAs. Xilinx is at the origin of the first FPGAs supporting DPR: the XC6200. However, DPR is only supported by Xilinx software tools since the ISE 9.1 and ISE 9.2 design suits [73] targeting the Virtex II Pro FPGA. Even though, DPR could only be realized through early access add-ons to these software tools. Much more series of dynamically reconfigurable devices were proposed by Xilinx through the Virtex IV, V, VI, and the most recent one VII [116], [117], [125], [123]. The first versions of ISE have evolved and now the versions 12 and 13 offer the DPR ability to every designer without any add-on (at the cost of the appropriate license) [122], [124]. Furthermore, the PlanAhead software that is part of the ISE/EDK suit [121] offers interesting possibilities to designers to control which PE should be dynamically reconfigurable.

From the beginning Xilinx has felt the interest of the DPR and invested a lot of time, money, and manpower to offer viable devices to industrials and academics. Since the DPR becomes a real market with growing interests and earnings, Altera recently announced their first FPGAs supporting DPR [36]. In Stratix IV series, only the input/outputs were already dynamically reconfigurable [8]. The programmable resources inside the FPGA were not dynamically reconfigurable [9]. This is not the case for the Stratix V series that is announced by Altera. Every programmable resource is dynamically reconfigurable, just like Xilinx FPGAs [10]. The use of the DPR is proposed to designers through the well known Quartus II design suit [10] on the same model as for Xilinx ISE/EDK recent versions. Altera recently published an article presenting the interest of systems using DPR in order to reconfigure the hardware architecture without rebooting the OS, even if it runs on an on-chip processor [62]. It is particularly interesting to notice that Altera's vision of applications using DPR is very similar to Xilinx's.

At the beginning of this PhD, only Xilinx devices were compliant with DPR. This is why in this report, only Xilinx architectures are considered.

The FOSFOR project

The FOSFOR project aims to reconsiderate the structure of an OS which is usually software, centralized, and static. The OS proposed by the FOSFOR project should be flexible and distributed even if proposing an homogeneous interface from the application point of view. For this purpose, the DPR ability of modern FPGA should be applied to complex RSoCs and completely supported by the FOSFOR OS. In these complex RSoCs, tasks can be either statically or dynamically implemented, and even allocated in software or hardware. The FOSFOR project proposes mechanisms of virtualization of the various services constituting an OS. This way, through these services, applicative tasks can be processed and can communicate without any knowledge about their implementation (software or hardware) nor about their location. More precisely, the FOSFOR project considers three fundamental services of an OS: the scheduling and placement of the tasks, the communication and the memory management. More than expected theoretical results, the FOSFOR project aims to demonstrate the viability of presented concepts through a demonstrator. This demonstrator is a hardware platform representative from future embedded applications [38].

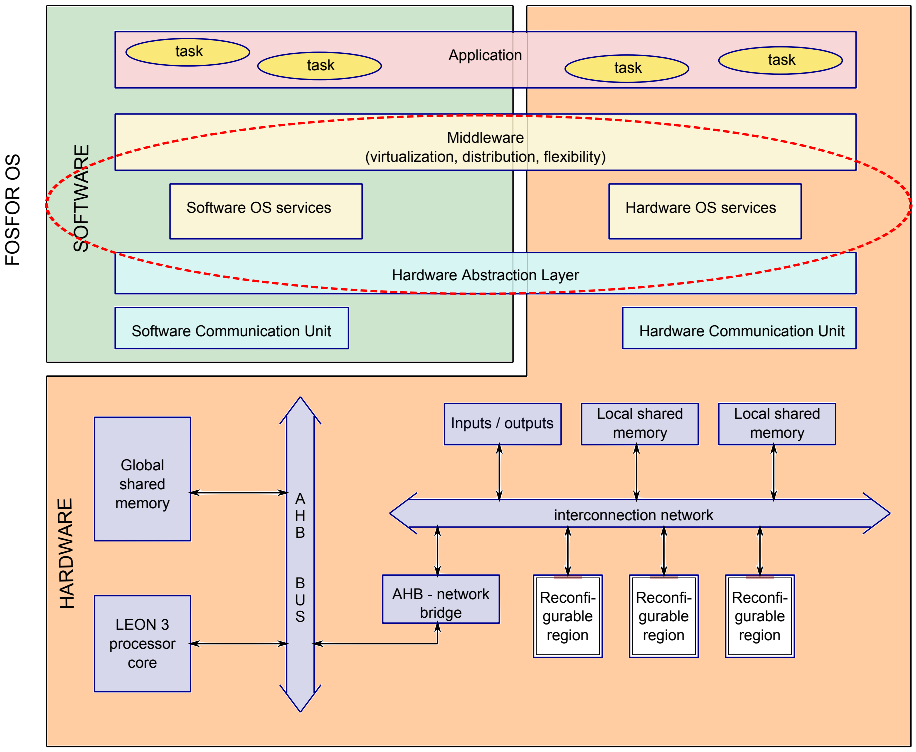


Figure 2 : Presentation of the FOSFOR environment. Constituted of two parts (software and hardware), the FOSFOR OS provides an abstraction of these two parts to the application. The services of the OS are distributed both in software and in hardware. Through an hardware abstraction layer and control interfaces, the OS manages the hardware platform. This platform consists in a LEON 3 processor core, a shared memory connected to the processor through an AHB bus, and reconfigurable regions interconnected by a network. PEs corresponding to hardware tasks are dynamically implemented in the reconfigurable regions.

The FOSFOR environment presented in Figure 2 is constituted of the FOSFOR OS whose services are distributed both in hardware and software. Through the specific block called middleware, the OS is able to make the relation between its various services and to provide a uniformized interface to the application. Through the middleware, the application is executed without any knowledge of where the tasks are implemented (software or hardware). Through the hardware abstraction layer, the OS manages the execution of the tasks allocating them on the LEON 3 processor (if software) or inside the reconfigurable regions. Allocated tasks always have the possibility to communicate together through the AHB bus and the interconnection network. They can also communicate with the FOSFOR OS through software command units (if allocated on the LEON 3 processor), or the hardware one (if allocated in a reconfigurable region).

Objectives and contributions of this work

The contribution of this PhD, lies in the proposal of an innovative interconnection architecture. The proposed network should be compliant with the DPR paradigm while being flexible supporting heterogeneous environments (software and hardware implemented tasks). Furthermore, since it could be integrated in the FOSFOR platform, this network should be able to efficiently interconnect the various PEs of the FOSFOR demonstrator. The demonstrator is only representative from future applications, this is why the proposed network should be as scalable as possible. Finally, the integration of proposed interconnection architecture should be eased providing a network with standardized interfaces.

However, the research works that took place during this PhD do not aim only in providing a communication architecture to the FOSFOR project. Much more contributions are expected. So, the second contribution should be to ease the use of the proposed interconnection network through a dedicated generation and characterization environment. This environment should provide to the designers an easy way to parameterize and generate the network. Furthermore, using this environment, the designer should easily receive estimations of the performances that are expected to be obtained from chosen configuration.

In order to ease the use of proposed interconnection network, an efficient communication manager is proposed. This manager should give to the tasks that wish to communicate an abstraction of the hardware components. Through this manager that acts as an OS communication service, tasks should be able to communicate in an heterogeneous platform without taking care about if they are statically or dynamically allocated, and even software or hardware implemented. Their location inside the platform should not even be known by the tasks. the challenge with this communication service is then to make communicate the same way hardware and software tasks. Furthermore, the communication service should provide efficient mechanisms in order to take into account communication cases that can be induced by the DPR ability of the RSoC. For example, the communication service should provide answers to improve the communications when a task requests to send data to a non-allocated one. Even if proposed outside of the scope of the FOSFOR project, it would be interesting for this communication service to be compliant with the FOSFOR OS.

Finally, considering the lack of flexibility of present interconnection networks, along with their limited scalability, the other main contribution of this PhD is to propose a NoC with very high network performances. This network should present a reduced cost in order to be scalable enough to fit future large scale applications. Furthermore, along with high bandwidth and low latency, this network should be flexible enough to support the DPR of PEs with a significant increase in QoS. Finally, since the need of performances is not reserved only to RSoCs, this network must be fully compliant with future flexible ASIC implementations.

Organization of the report

This report is organized in four chapters. Chapter 1 provides an overview of current dynamically reconfigurable architectures. FPGAs supporting DPR are presented along with the available design flows. Furthermore, the advantages and drawbacks of the DPR are also explained in this chapter. Then, before presenting some of the existing interconnection architectures, definitions are stated concerning every aspect of the conception and characterization of an interconnection architecture. Using these definitions also with the constraints induced by the DPR, existing interconnection architecture are detailed and discussed.

In Chapter 2, the DRAFT Network-on-Chip designed for the FOSFOR project is presented. Using both the constraints of the DPR and previously presented NoCs, the particular topology of the DRAFT network is investigated. Next, the various tools to make easier the use of DRAFT are detailed. Indeed, first is presented the DRAGOON environment that allows to parameterize, generate, and simulate networks. Then, a bridge making the relation between DRAFT and an AHB bus is detailed. This bridge is particularly useful providing a standardized and largely used interface. This allows for example the integration of DRAFT inside the FOSFOR demonstrator. Finally, a communication service acting as an overlayer to the DRAFT network is proposed. This communication service provides efficient mechanisms to manage communications in heterogeneous applications (hardware and software) using DPR.

In Chapter 3, the R2NoC and OCEAN networks are presented. R2NoC is an innovative NoC that uses the DPR ability of the FPGAs to provide high level performances with a reduced cost. Constraints and technological drawbacks of R2NoC are presented in this chapter. This leads to the definition of the OCEAN network which is a high performances, scalable, flexible, and easy to use NoC. This network auto-reconfigures its structure to provide maximum bandwidth and minimum latencies to the communications. As presented in this chapter, OCEAN aims both DPR compliant FPGAs and future flexible ASIC implementations. In order to evaluate the performances of OCEAN, a simulation environment is detailed.

In Chapter 4, characterizations of the proposed networks are presented. Experimental conditions are first detailed. Next, DRAFT and OCEAN characterizations are provided. All these networks are compared with existing and popular networks that we characterized in exactly the same conditions. Finally, ASIC implementation results of the OCEAN network are presented.

Finally, the last chapter concludes this work and proposes further researches.

Chapter 1

State of the art - Definitions

This chapter aims to give an overview of modern dynamically reconfigurable architectures. Both reconfigurable devices and reconfiguration techniques are presented. Following this presentation, a terminology is proposed in order to define every parameter influencing the conception of an interconnection architecture. Characterization metrics are also defined. Following these definitions, most popular interconnection architectures are presented and confronted to the dynamic reconfiguration paradigm.

1.1 Dynamic Reconfiguration

1.1.1 General definition

The dynamic reconfiguration consists in the possibility to change only one part of the chip while the other parts continue their execution. This is also called the DPR. This reconfiguration is based on the observation that in most applications, several functions do not need to be executed at the same time. This leads to a temporal partition of the application. Following this temporal partition, the idea is then to configure in the chip only the functions that need to be executed. At the end of their execution, they can be removed, freeing resources to dynamically configure another function. DPR significantly reduces the number of required hardware resources, and thus the size of the chips. The difference between the various reconfiguration types are presented in Figure 1.1.

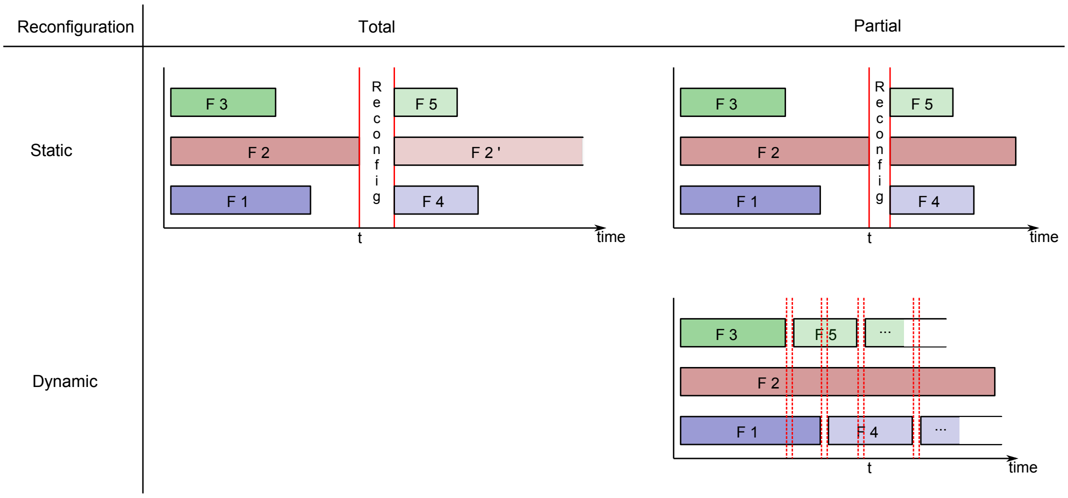


Figure 1.1 : Presentation of the various reconfiguration types. Two short functions F1 and F3 are reconfigured respectively by functions F4 and F5. The reconfiguration occurs at last at the instant ”t”. A longer function F2 is configured and needs more than ”t” to complete its execution.

In the static and total reconfiguration, at reconfiguration times all functions are halted and removed. The reconfiguration delay is quite long because the entire chip is reconfigured. The function F2 is un-allocated and reconfigured (equivalent to a reset) in instance F2'. If specific mechanisms were not foreseen, the execution of F2' starts from the beginning and all actions performed by F2 are lost. The static partial reconfiguration provides significant improvements compared with the total one. Since it is a static reconfiguration all functions are halted at the reconfiguration time, but only the functions that need to be reconfigured are un-allocated. This way, the reconfiguration time is shorter: only the needed resources are reconfigured. At the end of the reconfiguration process, F2 can resume its execution. The dynamic total reconfiguration does not make sense since every function is reconfigured. It is then similar to the static total reconfiguration. Finally, the dynamic partial reconfiguration lead for the functions that finished their execution to be reconfigured at once without disturbing the execution of the others. The reconfiguration process is shorter than for other reconfiguration types since only the needed functions are reconfigured at a time and longer functions are not halted. Thus DPR allow to best use the temporal partition of the application limiting the delays for reconfiguration purpose while ensuring that other functions are not impacted at all.

First industrial and academic applications using DPR were presented since 2001 [20], [49]. However, only the proposal of first chips proposing a DPR ability could demonstrate the viability of these applications since 2005 [82]. In latter article, issues were presented in order for the DPR to be more and more interesting for both industrials and academics. Issues were notably the creation of design flows supporting the DPR, larger scale FPGAs, and more FPGA vendors providing this functionality. Furthermore complex applications requiring DPR already exist like the real time video processing [28] and also automotive applications that are now part of our daily life [19].

1.1.2 Xilinx partial reconfiguration

Xilinx DPR occurs in specific regions called Partially Reconfigurable regions (PRRs) [26]. These regions are statically defined before place and route phase of the implementation. Partial reconfiguration can only occur inside these regions, thus distinguishing the dynamic parts from the static ones of the FPGA. Inside a PRR, every Configurable Logical Block (CLB) is dynamically reconfigurable, thus forming the smallest reconfigurable zones. As presented in Figure 1.2, these blocks are constituted of two regions so called slices. CLBs are arranged in column, allowing embedded slices to communicate with directly upper neighbours or with other parts of the chip through the switch matrices.

In Xilinx Virtex V series, each slice is constituted of four 6-inputs Look Up Tables

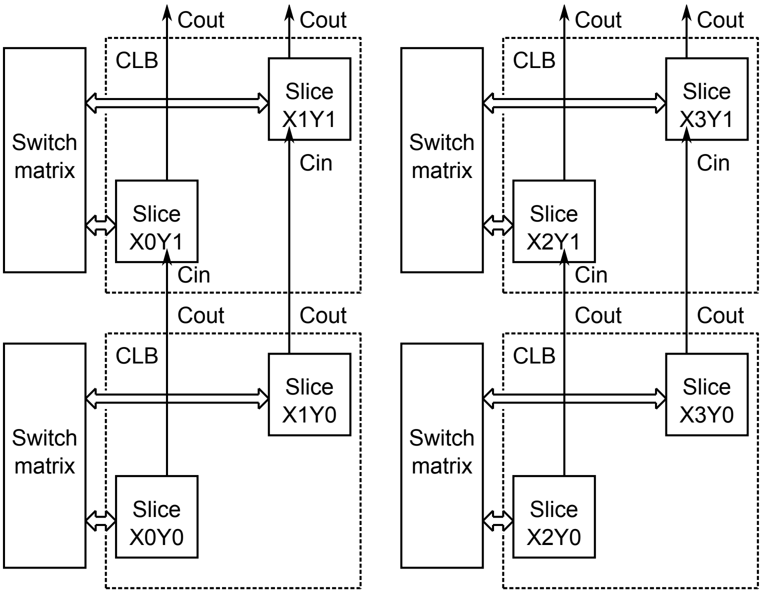


Figure 1.2 : Diagram of the resources embedded in Xilinx FPGAs. CLBs, composed of two slices, are the smallest reconfigurable entities. CLBs are arranged in column: each slice can communicate with upper neighbour one. Switch matrices allow the communication between a CLB and other resources (CLBs, input/outputs, etc.) located in different regions of the chip.

(LUTs) that are generators of logic-functions, four registers (called FF), multiplexers, and combinatorial logics. The diagram from a single slice is presented in Figure 1.3. Some LUTs support extra functions: storing data using distributed Random Access Memory (RAM), or shifting data with 32 bits registers [118].

During the DPR process, the function implemented inside the LUTs, the data in registers, the connexion of the multiplexers, and the switch matrix of a CLB are all reconfigured. A PRR can embed specific resources like RAM blocks (BRAMs) allowing 36 Kbits data storage each, and DSP blocks that are specific signal processing oriented units. This allows users to implement dynamically a complete PE using all required resources. The configuration of a PRR is realized through a bit file called partial bitstream. In order to reconfigure a PRR, a partial bitstream is loaded inside a specific interface called Internal Configuration Access Port (ICAP) [117]. Partial bitstreams are generated at compile time for every PRR regarding every possible configuration they can take. This way, each partial bitstream configures a whole PRR. This means that it is not possible to dynamically reconfigure only a part of a PRR, the whole PRR is reconfigured. However, if a resource has the same configuration in the novel configuration as in the previous one, then it is not affected by the reconfiguration process. This means that no glitch can occur, thus disrupting its behaviour [73].

The communication between resources embedded inside a PRR and those from

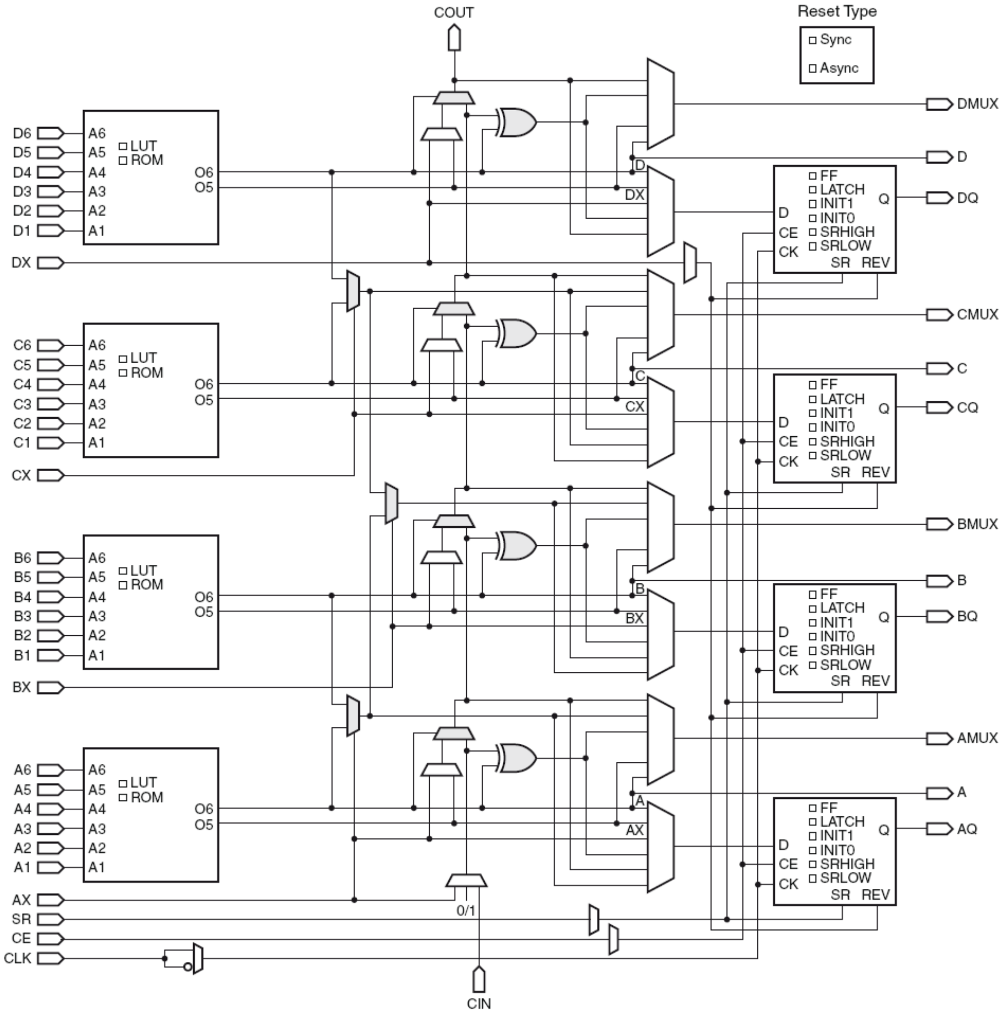


Figure 1.3 : Diagram of a single slice composed of four LUTs (LUT/ROM boxes of the left side), four registers (FF boxes of the right side), multiplexers and combinatorial logics. This diagram is issued from [118].

the static defined part of the FPGA are realized through interfaces called Bus Macros (BMs). BMs are statically defined inside the PRRs. Each BM acts as a bridge between dynamic and static parts of the FPGA. A single BM is implemented using a full slice [115].

Using Xilinx ICAP interface through the genuine controller provided by Xilinx usually leads to long reconfiguration delays. Indeed reconfiguration times from the decade to several hundreds of milliseconds are commonly encountered depending on the size of the PRRs. As an example, the dynamic reconfiguration of a PRR requiring a bitstream of 22KB takes 11.3ms. However, for many applications with hard real time constraints, such delays are not acceptable. This is why several works proposed optimized ICAP controllers leading to reconfiguration delays from the decade to several hundreds of microseconds [72], [35]. With such rapid ICAP controllers, expected reconfiguration times are between 50-60μs for a 22KB partial bitstream. However, such rapid ICAP controllers can only be implemented at the cost of a massive BRAM usage. Obviously, dynamically reconfiguring a PRR using Xilinx genuine ICAP controller is much faster than reconfiguring statically the whole FPGA.

Along with researches leaded in order to improve the dynamic reconfiguration process, there are many studies aiming to better control this process. Indeed, usually the DPR of a PRR is initiated by a software code running on an on-chip implemented processor like a MicroBlaze or a powerPC [92]. However, some researches aim to improve this DPR initiation through an object oriented control running along with an OS like linux [45] or OS4RS [75], [88].

Many works like [18] take advantage of the new possibilities that are, or should be, offered by the DPR. This way, placement of the PEs and of the PRRs is a key research topic. Virtex II pro FPGAs were only compliant with 1D placement of the PRRs [57]. This means that when reconfiguring a PRR, the whole column was impacted. Consequently it was not possible to place two PRRs in the same columns of resources. Presently, the 2D placement of the PRRs is fully supported. This means that PRRs are defined as rectangular arrays that can be placed everywhere in the FPGA matrix. Several PRRs can thus be placed in the same column of resources. Concerning PEs, like PRRs they can be placed in 2D in the FPGA. When a PE is dynamically implemented in a PRR, concerning the overall application, it leads to the 3D placement where the third coordinate is the time [17]. Due to their column based repartition of the resources, Xilinx FPGAs are not homogeneous in 2D, thus constraining the placement of the PEs. For example, the organization of a Xilinx FPGA also with placement of a PRR are depicted in Figure 1.4. Indeed, PEs are then implemented as arrays that can be swapped depending on the moment they are scheduled to be executed [105]. The online placement of the PEs in the PRRs was investigated through several studies like [4] [5]. As a result of these works, a high flexibility is provided to the applications because the initial placement of the PEs can evolve depending on the applicative context. However, the flexibility concerning the placement of the PEs directly impacts the communication architecture since it must support this dynamic adaptation of the overall architecture.

The DPR proposed by Xilinx presents several advantages and constraints. The first constraint lies in the number of PRRs that can be reconfigured at a time. Starting from the Virtex V series, Xilinx DPR compliant FPGAs embed two ICAP ports. However they can not operate at the same time. Partial bitstreams can be forwarded to the ICAP controllers in parallel, but the effective reconfigurations will

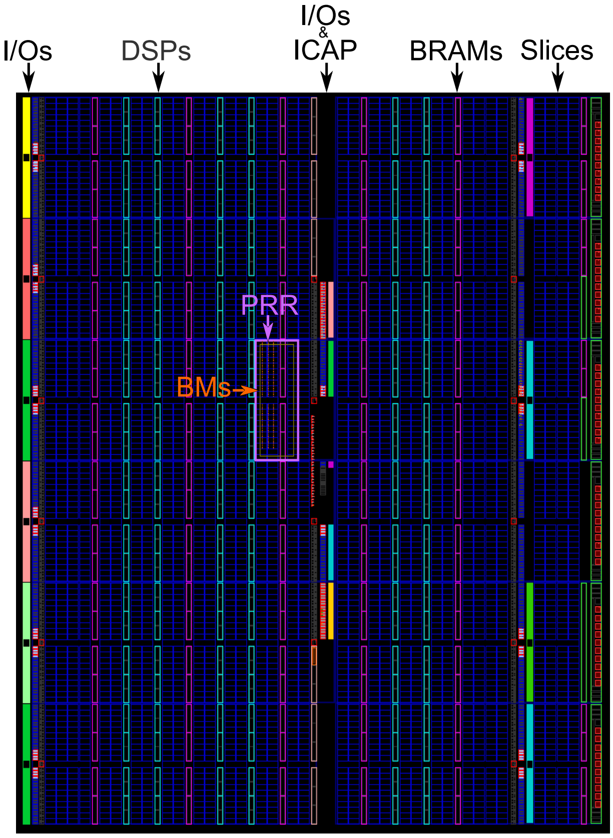


Figure 1.4 : Internal structure of a Xilinx Virtex V FPGA. Resources are distributed in columns. Resources (slices and BRAMs) included inside the defined PRR can be dynamically and partially reconfigured.

happen sequentially. So, two PRRs can not be reconfigured in parallel, but the presence of a second ICAP interface allows to save time considering the transfers of the partial bitstreams [117].

In every Xilinx Virtex FPGAs, partial bitstream are encrypted. Indeed, analyzing the bitstreams would give information on involved resources starting from their configuration bits as well as the implemented functions. However, this indispensable industrial protection is also a major constraint for researchers because it is not possible to generate online a partial bitstream. Since bitstreams are encrypted, and since FPGAs are not homogeneous in 2D, a small change between two configurations can not be realized in situ: both configurations are to be generated at compile time and stored in memory. Thus, this industrial protection has a dramatic effect regarding the number of partial bitstreams to generate and then to store in complex applications. Until the Virtex IV series, the addresses of the PRRs could be read at the beginning of the partial bitstreams. This way, the addresses of a given partial bitstream could be modified, and then the partial bitstream could be applied on another PRR. This leads to the relocation of a PE. This is why basic PEs relocation and defragmentation could be realized [30]. Relocation were also realized using Xilinx more mature devices (XCV200) [42]. However, since the Virtex V series, Xilinx encrypted not only the payload of the partial bitstreams but also the addresses. This way addresses can not be modified and any relocation/defragmentation approach is thus impossible.

1.2 Definitions

Sometimes in literature, researchers do not employ the same vocabulary. When characterizing a PE or a communication architecture, the definition that lies behind the performance metrics vary also quite often. This is why in this Section, all the needed parameters in the conception of a communication architecture and all metrics required for its complete characterization are defined. Definitions are issued from both [33] and [46] that are reference publications in the domain of the communication architectures. Obviously, all stated definitions are carefully respected in the remaining of this report.

1.2.1 Communication architecture

A digital system is composed of three basic building blocks: logic, memory, and communication. Logic transforms and combines data, for example performing an arithmetic operation or making decisions. Memory stores data for later retrieval. Communication moves data from one location to another. The communication medium linking the various logic and memory blocks, along with its dedicated control, is called communication architecture. For a SoC, there are two main types of communication architectures: the buses and the Networks-on-Chip (NoCs). In this report, Processing Elements (PEs) are composed of both the logic and the memory blocks since they both participate to the processing of the data.

Bus

Communication architecture where interconnected elements (PEs) share a unique medium of communication. Buses were historically composed of a set of parallel wires with multiple connections. A bus is usually composed of data and address wires. Every PE requiring to communicate must reserve the bus. When no other PE uses the bus, data can be transmitted on the data wires while asserting the proper destination address on the corresponding wires. Every connected PE accesses both data and address wires so that only the one seeing its address on dedicated wires can read the data. In order to improve performances in complex systems, communication architectures were created using several buses in parallel: they are called multiple buses. This way several communications can occur at a time, one on each unitary bus. Furthermore, segmented buses were also created. Segmented buses are composed of several unitary or multiple buses serially linked by elements called bridges. Every bridge makes the relation between one bus and another one. Examples of multiple and segmented buses are provided in Figure 1.5. Further details are provided in Section 1.3.

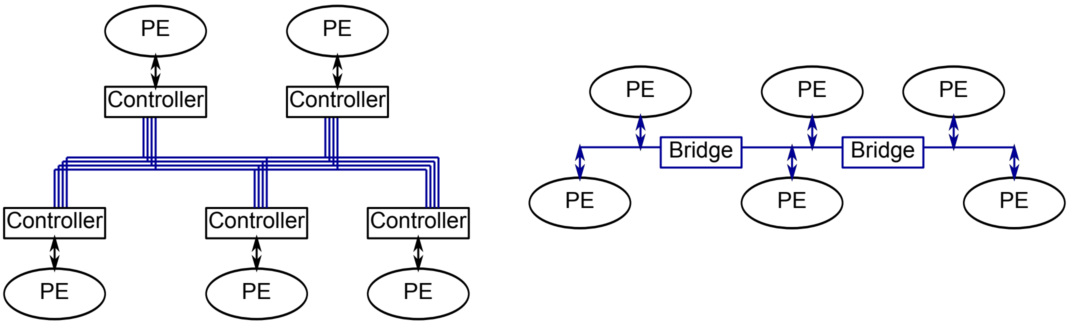


Figure 1.5 : Examples of a multiple bus (left) accessed through controllers, and of a segmented bus whose segments are interconnected by bridges.

Network-on-Chip (NoC)

Networks-on-Chip are defined as a set of shared routing nodes and communication links. There are two types of networks: direct and indirect. In a direct network, each PE is connected to a dedicated routing node. In an indirect network, some of the routing nodes are linked with one or several PEs while remaining ones are only linked with neighbour routing nodes. Direct and indirect NoCs are presented in Figure 1.6. A network is defined by its topology, its routing techniques, its switching policy and its flow control.

1.2.2 NoC parameters

Topology

The topology refers to the static arrangement of links and routing nodes in an interconnection network. The topology can be viewed as a road map. Data (like cars) are transmitted on links (like roads) from a routing node to another one (like intersections). Finally, the topology is not only the arrangement of the routing nodes and links, it is also the size of these links (width of the roads). The topology is the first choice when designing a network because the routing strategy and the flow control method are highly correlated with it. A topology is chosen regarding its cost and

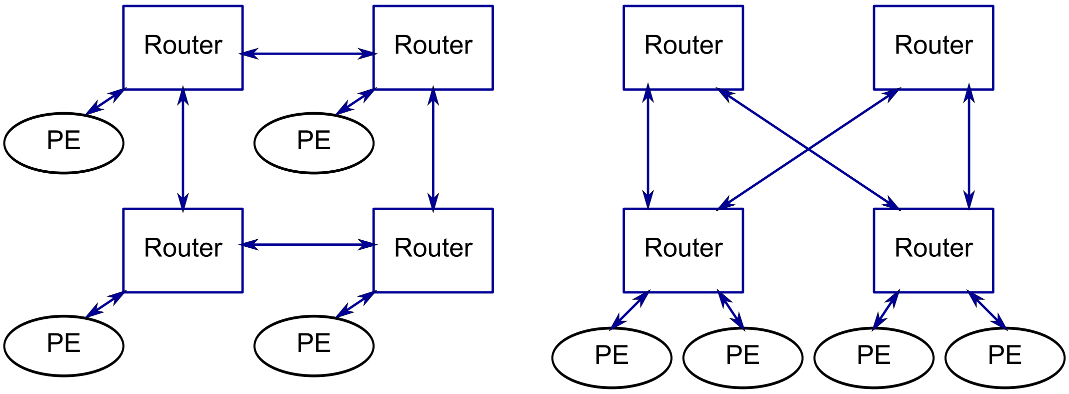


Figure 1.6 : Direct (left) and indirect (right) networks. In a direct network every router connects a PE while in an indirect one several routers do not connect directly any PE.

performance. The cost is defined as the number and the complexity of the routing nodes also with the density and length of involved links. Performance has two components: the bandwidth and the latency. A good topology is a trade off between the cost that should be minimized and the performance that should fit applicative requirements at minimum. Further details concerning topologies are provided in Section 1.3.

Routing

The routing method employed in a network determines the path taken by a data to reach its destination. If the topology is the road map of the network, routing is the next logical step: choosing one road to reach the destination. A good routing algorithm balances the load across the network links even in the presence of a non uniformly distributed data traffic. The more balanced is the load, the more performances will be close from ideal. A well designed routing algorithm also keeps the path length as short as possible in order to reduce the overall latencies of the data. There are three types of routing algorithms:

- Deterministic: the routing algorithm always chooses the same route for a specific couple source and destination PEs. This is for example the case of the XY algorithm: in an array of routers, like the direct network presented in Figure 1.6, data always travel on the X axis until they reach the column the destination PE is connected to. When in the proper column, they follow the Y axis reaching the line where the destination PE is located.

- Oblivious: this routing algorithm includes the deterministic routing as a subset, the choice of the path is made without considering any information about the network's present state. As an example, an oblivious but not deterministic algorithm consists, considering an indirect network like in Figure 1.6, in randomly choosing a path through upper level routers, then climbing down the structure of the network to reach the destination PE.

- Adaptive: the routing algorithm adapts itself considering the state of the network, for example avoiding congested areas. An example of adaptive routing is the turn model: data travel using a deterministic algorithm, but if they encounter a router that is already used for a communication, then they turn on another direction, thus reaching the destination through a different path (different lines and columns are used).

Circuit and packet switching

A network can be circuit switched or packet switched. In circuit switch, resources (links, buffers, etc.) are first allocated for a communication forming a reserved communication channel between the source and the destination PEs. When this channel is created, data can be transmitted (one or several packets of data). In circuit switch the routing node of the network are called switches while they are called routers in packet switch. In packet switch, data are injected inside the network with the destination address. The data path is allocated to this communication as packets advance inside the network. This is in packet switch that the flow control protocols presented in next section take all their sense. So, in packet switch, data are divided in one or several packets. Each packet can follow a different data path. For this purpose, usually each packet is composed of first sets of bits called flits that contain for example source and destination addresses, as well as other control informations (Cyclic Redundancy Check (CRC), etc.). These first flits are called header. Usually the header is followed by a flit called count since it contains the number of flits of data that are part of the packet. Data flits are called the payload.

Flow Control

The flow control protocol determines how resources of the network (links, buffers, etc.) are allocated. A good flow control method allocates these resources in an efficient manner so the network achieves a high fraction of its ideal bandwidth while delivering data with low predictable latencies. When two data arrive at a routing node for the same destination, the flow control method allocates resources to a communication while dealing with the other one. This way contentions are resolved. There are three main flow control methods: the store and forward, the virtual cut through, and the wormhole. In store and forward, every routing node stores the whole packet in its buffer before sending it to the following neighbour or the destination PE. In virtual cut through, the whole packet is also stored in a buffer of the routing node until the whole packet is successfully transmitted, but flits are forwarded as soon as possible. In wormhole, only a small amount of flits are stored in a buffer. Flits are forwarded as soon as possible freeing a memory space in the buffer. This way, when a flit is forwarded, another can arrive. So, the packet is distributed over several routing nodes along its route like a worm.

A flow control protocol also manages the functioning of the buffers. One management protocol is based on credits. Each buffer has a counter indicating if further data can be received. Each time a flit is transmitted to next node (router or PE), the counter of considered buffer is incremented. Similarly, each time a flit arrives in the buffer, the counter is decremented. If the counter value is zero, the buffer is full and a signal indicates to previous node (router or PE) that no further data can be received. Another flow control protocol is the On/Off. A signal indicates to a previous node if it is allowed to transmit data. If the buffer storing incoming data contains more than a certain number of flits (called threshold), the "Off" signal is transmitted halting the communication in previous nodes. When the number of flits in the buffer becomes less than the threshold value, the "On" signal is asserted so that further flits can be received. Finally, another flow control protocol reducing the load of the buffers is the Ack/Nack (also called handshake). Each time a flit arrives in a router, if it can be stored in a buffer an acknowledgment (Ack) is sent. If no buffer is available, the flit is drop and a non-acknowledgment (Nack) is sent. When receiving a "Nack" signal, previous node sends the same flit again until it receives an "Ack".

Traffic Patterns

The Traffic pattern is the main concern when performances of a network are to be evaluated. Indeed, the topology of a network, the routing algorithm and the flow control methods react differently depending on the traffic pattern. A random traffic in which each source PE sends data with equal probability to each destination is the most commonly used traffic. A completely random traffic do not stress a network because, by making the traffic uniformly distributed, it balances data load inside the structure of the NoC even for topologies and routing algorithms that normally have very poor balance. Some very bad topologies and routing algorithms look good if only evaluated with a random traffic. To stress a topology or routing algorithm, permutation based traffic is typically used. In this pattern, each source PE sends all of its traffic to a single destination. Because load are then concentrated on individual source/destination pairs, permutations stress the load balance of topology and routing algorithm. In such a traffic, destination addresses are calculated from source addresses with a permutation or rotation of the bits.

In addition with the choice of destination addresses that can notably be random or permutation based, the temporal distribution of the communication is also important. So, a PE sending all its data at a constant data rates forms a Uniform data traffic. The data rate can also vary in the time, reaching some peaks. Traffics patterns with data rates following a Normal temporal distribution are commonly encountered in applications. Another temporal distribution characterize the functioning of most memories. This is the burst transfers: data are grouped and transmitted sequentially without interruption. When all data are transmitted, periods of silence occur.

Deadlock, livelock and contention

Deadlock occurs in a network when a group of packets are unable to progress because they are waiting on another one to release resources (buffer, link, etc.). If a sequence of packets forms a cycle in the network, then the network is deadlocked. A deadlock is catastrophic because when few resources are occupied by deadlocked packets, other packets will block on these resources thus completely paralyzing the network. Furthermore the network would remain in this state until external intervention.

A closely related network pathology is livelock. In livelocks, packets continue on moving inside the network but without making progress toward their destinations. This becomes a concern for example when packets are allowed to take non minimal paths through the network. Livelocks are as dramatic as deadlocks for a network.

Contentions are defined as delays imposed to a packet in order to wait for a resource to be available. Contentions are not problematic like deadlocks and livelocks because the network recovers from contention without any external intervention. It recovers as soon as previous communication finishes or as soon as another path is found.

1.2.3 Network performances

Latency

Latency is defined as the time elapsed between the moment the source PE sends the first bit of a data and the moment the destination PE receives the last bit of data. There are different types of latencies: the sender overhead, the time of flight, the transmission time, and the receiver overhead. These latencies are presented in Figure 1.7. The sender overhead corresponds to the time for the source PE to prepare the packets that are to be injected in the network. The time of flight is defined as the time for the first flit to cross the network, and thus to reach the destination PE. Time of flight usually depends on the traffic inside the NoC, so it may vary accordingly with the injected traffic. The transmission time corresponds to the time taken by the whole packet of data to pass through the network. The transmission time do not include the time of flight. However, this time is highly linked with the size of the packet in terms of number of flits. The receiver overhead is defined as the time required by the destination PE to process the incoming data. The sum of the time of flight and the transmission time is denominated as transport latency. Transport latency thus corresponds to the time the packet spends inside the network. Finally, the total latency is defined as the sum of the transport latency and the sender/receiver overheads. Total latency represent from the PE point of view the total time required for communication purpose. Usually, in network characterization, only the transport latency is considered. This allows to extract the performances of the network alone, or interconnecting ideal PEs.

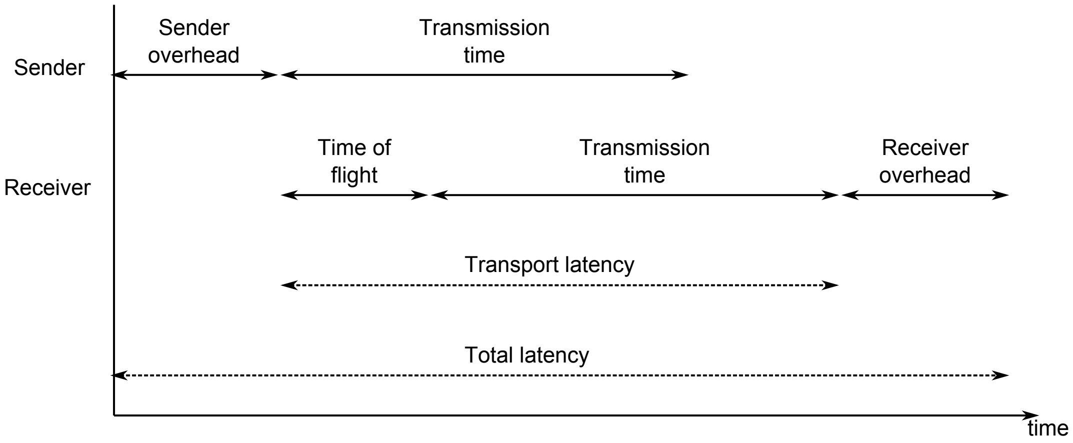


Figure 1.7 : Presentation of the various delays constituting the total latency.

In the remaining of this report, the transport latency will always be considered under the appellation ”latency”.

Injection rate

The injection rate, also called offered traffic, is defined as the number of packets that are offered by every PE to the network. Offered traffic corresponds indeed to the number of packets each PE would like to inject in the network per unit of time. As in next section, offered traffic is often confronted to the accepted traffic, i.e. the number of data the network can accept. When the accepted traffic is less than the offered one, the network is not able to transfer all the data provided to it (this is the saturation). Injection rates are usually expressed in percentage, as a fraction of capacity which corresponds to the theoretical bandwidth of the network. If C is the capacity of a network, D_w the width in bits of the data lines, and F the operating frequency of the NoC, then C is

$$C = D_w * F \tag{1.1}$$

Then, if $D_{offered}$ corresponds to the number of data bits per second offered to the network, then the injection rate I_r, in percentage, is

$$I_r = \frac{D_{offered} * 100}{C} \tag{1.2}$$

From this formula, PEs injecting packets using all the theoretical bandwidth of the network define an injection rate of 100%. Similarly PEs injecting data using only half of this theoretical bandwidth offer an injection rate of 50%.

Throughput

Throughput corresponds to the rate at which packets are delivered by the network. It is measured counting the number of packets that arrive at destination over a time interval. Usually presented in percentage as a fraction of network's total capacity, it is also sometime expressed as a data rate. Throughput represents the traffic that is accepted by the network. This is why throughput is usually contrasted with the injection rate (offered traffic). At traffic level less than saturation, accepted traffic is proportional to offered traffic. However, when the network saturates the accepted traffic either reaches a constant value or decreases. When a constant value is reached, the network is called stable because accepted traffic is ensured to fit the offered traffic until a maximum fraction of capacity. However, an accepted traffic decreasing when the network saturates reveals an instability of this network: accepted traffic feats the offered one until reaching a pick after which performances fall down. Stable and instable networks are presented in Figure 1.8

1.3 Interconnection architectures

In previous sections, the impact of the DPR over the conception of Systems-on-Chip was presented. Keeping in view the large range of applications that have to be implemented, an interconnection architecture should support several constraints induced by the DPR. Current applications are very complex and their task graphs exhibit a large degree of parallelism. Thus, from the interconnection point of view, the architecture must provide the possibility to realize several communications in parallel. Furthermore, dynamic placement and scheduling of PEs in an FPGA require a high level of flexibility. So, neither the location of PEs nor the data traffic (uniform, permutation, etc.) can be predicted at compile time. These requirements of flexibility should be considered by the network topology and the routing algorithm, and thus

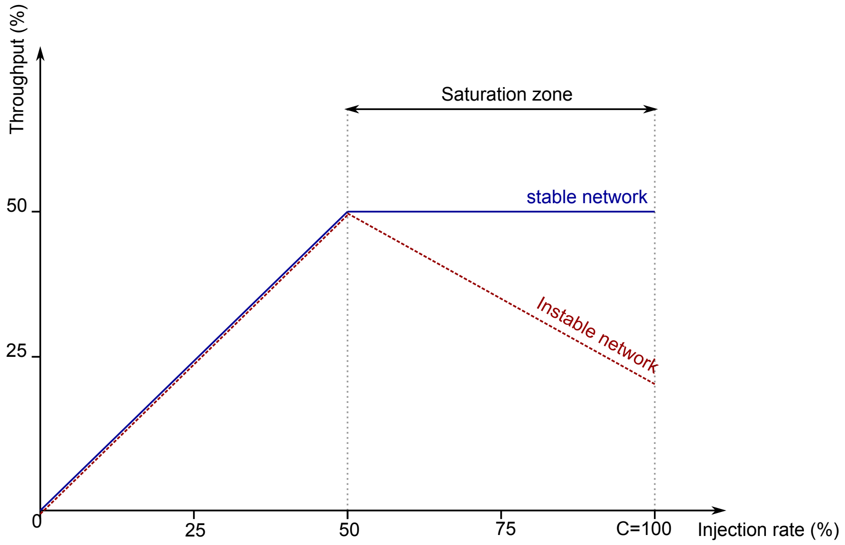


Figure 1.8 : Presentation of the throughput compared with the injection rate for a stable (blue) and an instable (red) network.

the available network performances. An application is typically split into tasks, and there is no reason for every task to be implemented in homogeneously sized hardware PEs. Heterogeneously sized PEs are considered in this work. Furthermore the matrices of current FPGAs are highly heterogeneous when considering the 2D repartition of the resources. So, both heterogeneously sized PEs and heterogeneous FPGAs should be supported by the interconnection architecture. Finally, a communication architecture should present an interesting trade off between the implementation cost (used resources) and performances. Since a large range of applications is considered, network performances can be sparingly privileged.

Some existing interconnection architectures are now presented. Considering the very large amount of different architectures that were proposed in research, this presentation can not be comprehensive. However, most popular interconnection are presented and confronted to the constraints induced by the DPR.

1.3.1 Bus based interconnections

Buses are very popular in present on-Chip circuits. Since the sizes of most affordable FPGAs are still limited in terms of CLBs number, the number of PEs constituting an application is also limited. This is why Xilinx proposes bus based interconnection architecture directly accessible from the ISE/EDK design environments. Thus, both the Processor Local Bus (PLB) and the On-chip Peripheral Bus (OPB) are available from these tools. These two buses provide high level performances with a reduced cost. They are quite simple to implement into the scope of an application thanks to the ISE/EDK environments and to their well documented interfaces [120], [119]. Furthermore, thanks to the ISE/EDK environments, many IP cores are available to be connected to these buses: MicroBlaze or PowerPC processors, memory controllers, communication interfaces like PCI bridges, etc.

Others largely used bus based interconnections are the Advanced High-performance Bus (AHB) that is inherited from Advanced Microcontroller Bus Architecture (AMBA) [40], and the Wishbone bus [89]. Both AHB and Wishbone buses are open source. Wishbone is provided by OpenCores while AHB is accessible through the GRLIB library also with many IP cores that can be directly interconnected: LEON 3 processor, memory controllers, etc. AHB provides high level performances as demonstrated in [12]. Considering these bus based interconnections along with their design environment and existing compliant IP cores (PEs), designing complete applications is substantially eased.

Several others bus based interconnections were proposed as NECoBus that is very close from AMBA [13]. Some of them are used on-Chip like in 8032 or ARM processors [112]. However, as every bus based interconnections, they suffer of a poor scalability resulting in a fall of performances when the number of interconnected cores increases. Furthermore, even if currently DPR is realized through statically implemented interfaces, all these buses were foreseen to operate only with statically allocated PEs.

Some buses have been designed in order to present both improved scalability and higher performances. Segmented buses with different clocks per segment were presented in 1999 [84]. This is the Globally Asynchronous Locally Synchronous (GALS) approach. Each segment has a dedicated clock reference allowing synchronous data transfers. The clock adaptation between the various segments is realized through the use of bridges. This way, the overall interconnection architecture seems asynchronous. As demonstrated through an ASIC implementation, the proposed bus presents interesting performances. Following the same idea, another segmented bus was proposed for high performance compliance purpose in 2002 [102]. Finally, the HIBI interconnection network was designed. Indeed, HIBI is based on segmented buses and uses also the GALS approach [99]. Following the GALS approach, a fully asynchronous bus was proposed in 2003 [56]. Latter bus is expected to provide better performances than synchronous based ones because it does not take care about propagation delays nor the parasitic effects that affect the physical layers of the chips. Unfortunately, no comparison with synchronous buses was presented.

More researches were lead in order to design efficient buses. This is the case of the bus presented in 2009 in [87], or of FLEXBUS [103]. However the introduction of the DPR paradigm in the on-Chip design methodology has lead to DPR oriented buses. This is the case for example for ReCoBus [64] which is defined as a shared bus interconnecting PRRs. The force of ReCoBus lies in its design framework that allows it to be statically generated accordingly with aimed application [65].

However, even with segmented and GALS approaches, buses always suffer of a low scalability. Performances fall down significantly when the number of interconnected PEs grows up. This is due to the share of the communication mediums and their controllers that scale badly and dramatically increase the delays for control purpose. This way, as specified in [127], NoCs provide better performances than simple buses starting from 16-25 interconnected PEs. Currently, applications present only a dozen of simultaneously interconnected PEs, and rarely more than 15. Buses are then particularly interesting despite of their drawbacks. However, applications with more interconnected PEs must be foreseen to become a reality in a close future. This is why NoCs are so popular in the research community. DPR significantly reduces the numbers of PEs that must be interconnected, but 32 or even 64 simultaneously connected PEs seems reasonable to be anticipated respectively at medium and long term.

1.3.2 Static NoCs

There exist a very large variety of NoCs, such that even most comprehensive surveys like [98] or [23] can not cover every proposed work. So, in this section we do not aim to present all the existing NoCs, but only some representative ones. Motivations for using NoCs instead of buses are not new. Starting in 2001, several works pointed out the interest of using NoCs [32], [21], [52]. Following these interests, NoCs switched from being an emerging research topic to an active community. This way, many publications were realized in order to explore all the various parameters of a NoC (topology, routing, flow control, quality of service, etc.) and their impact over performances [53], [22], [41], or even [68].

If there exists a large variety of NoCs, they can be classified in only few basic families depending on their topology. Latter families are described in next sections.

Mesh and torus

Mesh and torus, presented in Figure 1.9, are two very common topologies for NoCs [33]. Such topologies are commonly encountered in commercial high-performance machines notably from IBM [46]. Due to the direct connection of the PEs, mesh based networks appeared to be interesting solutions for improved scalability. Indeed, if the number of connected PEs is N, then the number R_{mesh} of routers needed to build a square mesh is

$$R_{mesh} = \left\lceil \sqrt{N} \right\rceil^2 = B^2 \tag{1.3}$$

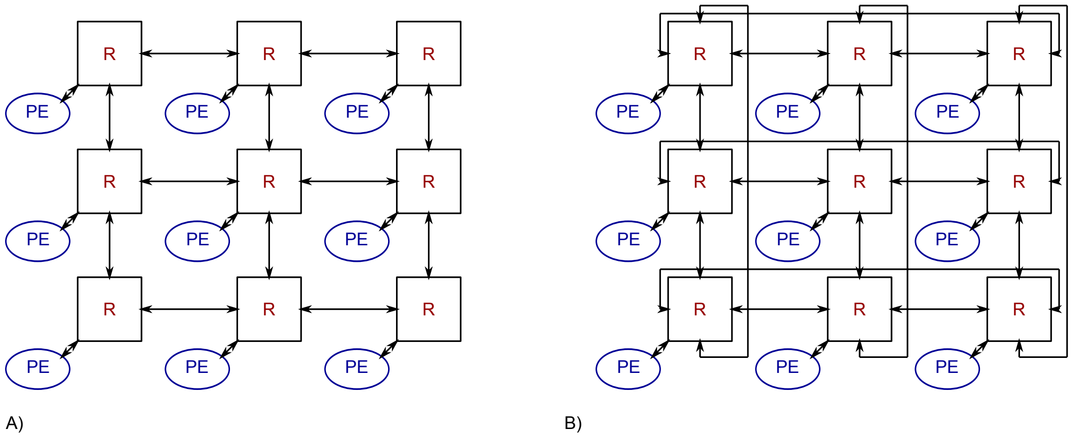


Figure 1.9 : A) Mesh topology: routers "R" are interconnected forming a regular matrix. Every router is linked with a dedicated PE. B) Torus topology: each column and each line of router is interconnected as a ring. For both topologies, connection links are bidirectional.

Where B is the base: number of router per line of a square mesh. Whereas the number L_{mesh} of needed communication links is

$$L_{mesh} = N + 2(B^2 - B) \tag{1.4}$$

Concerning on-Chip implementations, HERMES is undoubtedly the most popular NoC in research community [81]. Based on a mesh topology, HERMES is both simple to use and efficient considering produced network performances. Studies were lead in order to formalize the performances so that they could be predicted into the scope of as much applications as possible [34]. Following this formalization, several mesh based networks were proposed [69], [110], but one is particularly interesting: Ætheral [44]. This network has the particularity to guarantee levels of service and performances. This way, applications can be designed with guaranteed performances and reliability. When compared with an AHB bus, Ætheral reveals itself to be cost efficient and to significantly reduce contentions [16]. A variant from Ætheral with different flow control was proposed in [101].

If mesh based networks are foreseen with a regular structure interconnecting homogeneously sized PEs, some studies deal with the problem of how interconnecting heterogeneously sized PEs in heterogeneous FPGAs. An answer is DyNoC [25]. DyNoC interconnects homogeneously sized sets of logical resources. However, PEs are implemented using one or several of these sets. This way heterogeneously sized PEs can be interconnected. However this solution implies a significant waste of hardware resources due to the unused routers, the granularity of the sets of resources, and furthermore the communications between the various resources included inside the sets constituting a single PE. the CuNoC network was recently proposed improving

DyNoC with a routing algorithm taking into account the heterogeneity of the PEs [55]. However, a significant hardware overhead is still induced by the connection of these heterogeneous PEs. This is one of the main reasons why mesh based networks are not fully compliant with applications using DPR in heterogeneous environments. Another reason lies in the location of the shared element inside mesh structure. Indeed overall performances of a mesh are very sensitive to their location. A shared element located in the centre of a mesh induces a lot of contentions themselves impacting the communications in the whole network.

However, the quality of service in a mesh can be significantly improved with simple mechanisms such as adaptive routing algorithms or flow control strategies [66], [104], [93]. Furthermore, some mechanisms were also proposed in order to support multicast (sending of data to multiple destinations) [100]. This is particularly useful for data flow oriented applications.

Improved performances and flexibility can also be attained through modification of the topology. This leads notably to the torus topology [74]. Every line and column of a mesh is replaced by a ring. These way additional data paths are created with a small hardware overhead. This is the case for the DMESH network that is based on the xmesh topology [109]. A xmesh is a standard mesh where diagonal links are implemented in every set of four routers. However, these topologies face the same problems as the regular mesh when facing the DPR paradigm. They are still sensitive to the location of the shared PEs and to the heterogeneity of both the size of the PEs and the FPGAs themselves. This is why they are more adapted for statically implemented on-Chip applications. In this latter case they are particularly interesting even considering the power consumption. As demonstrated in [113], mesh based networks consume less power than buses when related to the number of transmitted bits per second. Meshes are most power efficient. This study also indicates that circuit switched NoCs reduce again the power consumption because less buffers and simpler control mechanisms are required.

If mesh are usually designed for packet switching, several evolutions were proposed to handle circuit switching [114]. Like PNoC [47] which is made of a set of crossbar based switches, NoCs such as SoCBUS are based on the mesh distribution of the switches [111]. These networks only supporting circuit switching as well as hybrid ones merging circuit and packet switching [80], [79] allow to significantly improve network performances and power consumption. The complementarity of circuit and packet switching in latter network allows to improve both the scalability and the flexibility facing various data traffics that become the weakness of others circuit switching networks. Since the support of circuit switching can improve network performances, it could be interesting to consider carefully this switching policy even with other network topologies.

Ring and spidergone

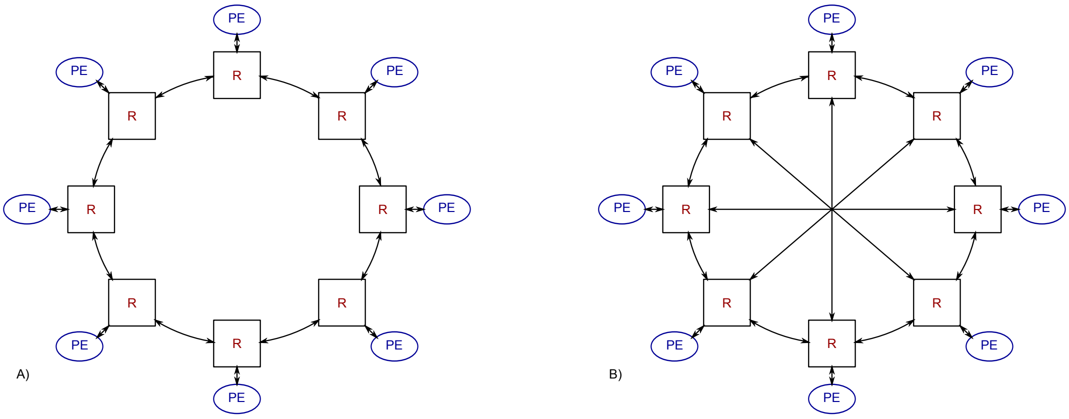


Figure 1.10 : A) Ring topology: every PE is interconneted through a dedicated router, itself linked with neighbor routers forming a cycle. B) inherited from the ring topology, spidergone pocesses additional diagonal links allowing to directly access the opposite side of the network.

Unlike bus based interconnection architectures, rings (Figure 1.10.A) can allow "many" transfers simultaneously. Indeed the first PE can send data to the second while the second sends data to the third and so on. However, as dedicated links do not exist between nonadjacent PEs, data often travel through several routers until they can reach their destinations. Thus significantly increasing the transport latency [46]. Consequently, ring based networks are very sensitive to the localization of the interconnected PEs. One way to achieve better performances is to use multiple rings. Multiple rings like RMBoC [3], similarly to multiple buses, have the particularity that several communication ways are available between every router. A comparison was made in [24] between RMBoC and DyNoC. If no network performances comparisons are presented, it appears that RMBoC presents a significant overhead considering the use of hardware resources. Multiple rings constitute very efficient communication architectures and are used in high performances machines like the CELL broadband engine from IBM [51].

Another way to improve performances is to use hierarchical rings [29]. Using several ring interconnected together allows to create sub-networks that significantly reduce the number of routers involved in every communication. However, if PEs can be optimally placed in a static environment, such interconnection network is quite impractical in a DPR compliant architecture. Indeed, the dynamic placement of some PEs that share a large amount of data on distinct sub-networks would result in a significant fall of performances. This is why rings should not be used in the context of an application using DPR.

Spidergones (presented in Figure 1.10.B) are very close from the ring based networks. The difference lies in the additional links that make the relation between every diametrically opposed routers. Spidergones are for example part of the Star-Wheels network [43] or used in the QUARC NoC [78], [77]. All these networks present an acceptable hardware cost while offering high network performances. However, they both suffer from the same drawback as for ring based networks. They are not flexible considering the dynamic location of the PEs that significantly impacts the performances.

Tree and multi-stages network

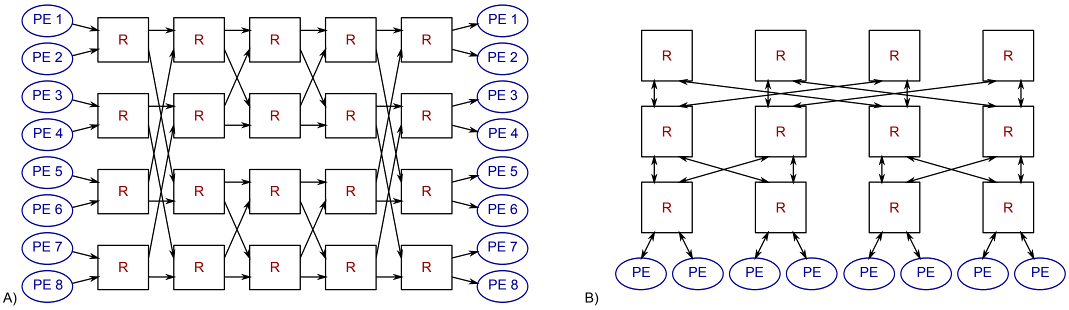


Figure 1.11 : A) Beneŝ network: multi-stages network in which output of the connected PEs are all located at the left side while inputs are at the right side. Thus communications are uni-directional. B) Fat-tree topology: PEs are connected to routers located at the base level of a tree. Since the number of links is the same between each hierarchical level, the available bandwidth is constant all along the network.

Multi-stages networks were initially proposed for telephone exchanges in the 1950s and have since been used to build the communication backbone for parallel supercomputers, symmetric multiprocessors, and multi-computer clusters [46]. Multi-stages networks like Omega or the Beneŝ network that is presented in Figure 1.11.A are based on the same principle. The network is composed of routers (or switches) forming several hierarchical levels. The connection of the various routers allows data to reach any destination from any source. Usually, the source PEs (their outputs) are located at the left side of the NoC while destinations (PEs' inputs) are at the right side. So, both links and router inside the network are unidirectional [33]. However some symmetrical networks like Beneŝ' can be easily folded. This way links and routers become bidirectional. Finally, a folded beneŝ network is equivalent to a fat-tree as presented is Figure 1.11.B.

Just like mesh, the fat-tree topology and the Omega network have been used in some high performances machines from both IBM and Intel [46]. However, the main drawback of these networks when aiming an on-Chip implementation lies in their scalability. Indeed their hardware resource consumption significantly increases with the number of connected PEs [15]. Considering only complete fat-trees, if the number of connected PEs is N ($N \geq 2$) and the number of communication ports for each router is r, then the total number p of communication ports is

$$p = 2^{\lceil log_2(N) \rceil} \tag{1.5}$$

This leads to the number $R_{fat-tree}$ of routers in a fat-tree [46] which is

$$R_{fat-tree} = \frac{2p}{r}(\log_{\frac{r}{2}} p) \tag{1.6}$$

Furthermore, the number $L_{fat-tree}$ of connection links needed by the fat-tree topology [46] is

$$L_{fat-tree} = N(\log_{\frac{r}{2}} N) \tag{1.7}$$

Comparisons between the numbers of routers and links involved in both fat-tree and mesh based networks are presented in following chapter. However, since 1985 fat-trees have been characterized for supercomputing purpose and it appeared that this topology allowed to build near optimal networks [71]. Following this observation, some researches were lead in order to characterize and implement fat-tree based networks aiming on-Chip applications [94]. This is the origin for example of the SPIN network [2]. However, when comparing on-Chip implemented fat-trees to meshes, despite of a higher hardware cost, fat-trees provide much better performances [86]. Higher bandwidths with lower latencies were measured for the fat-trees. Furthermore, since PEs are not distributed inside the structure of the network, fat-trees are fully compliant with the interconnection of heterogeneous PEs in a heterogeneous FPGA. This notably led to the on-Chip implementation of the SPIN network [11].

Considering the advantages of fat-tree topologies, many investigations took place in order to further improve this topology. This is how some researchers modified the fat-tree topology in order to improve the throughput [27] even if it implies an hardware overhead (more wires and more buffers). Consequently this network does not seem to be realistic for on-Chip implementations. However other researchers aimed to reduce the hardware cost of fat-trees. Resulting network is the XGFT [59], [60]. Instead of directly connecting the PEs to the fat-tree, they are grouped in several sets themselves connected to the NoC. This way the number of communication ports needed for the fat-tree is reduced and so is the overall hardware cost [58]. Furthermore, XGFT was improved with fault tolerance mechanisms [61]. This is a very powerful interconnection network that is compliant with heterogeneously sized PEs. However, when considering the DPR paradigm, XGFT is not so good because of the shared interconnection mediums that are embedded inside the sets of PEs. Indeed since the placement of the PEs can not be predicted, the bandwidth that is offered for intra-fat-tree communications is reduced compared with the number of interconnected PEs.

Finally, several researches were also lead in order to create a dynamic routing of the data inside a fat-tree based network [7], [126]. This research aims to balance the data traffic inside the structure of the NoC, thus allowing better performances, and to improve its fault tolerance. In this latter case, some faulty routers can be avoided by the data, thus allowing the communications to happen at the cost of only a little higher latency (due to the probability of contentions that increases for a constant offered traffic).

Considering all these studies, fat-tree based networks seems to be viable solutions for on-Chip implemented applications using DPR if the hardware cost can be limited.

Crossbar

A single switch suffices to interconnect every simultaneously implemented PE of an application if it possesses a sufficient number of communication ports: at least one per PE. This simple network is usually referred to as a crossbar. Crossbars allows high speed interconnections without any risk for a conflict, and then a contention, to happen [33]. Indeed, as it can be seen from Figure 1.12, a direct and dedicated data path exists between each couple of source/destination PEs. However, the complexity of a crossbar increases quadratically with the number of interconnected PEs [46]. This is why, despite of their very high network performances, crossbars present a very poor scalability. Indeed, if a crossbar has a number p of ports, then the number $Nb_{combination}$ of possible routing combinations is equal to $factorial(p)$: $Nb_{combination} = p!$

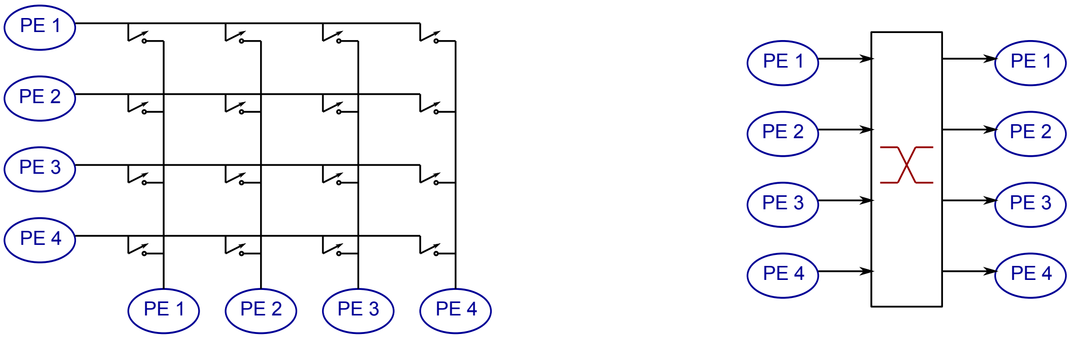


Figure 1.12 : Functional (left) and symbolic (right) view of a crossbar interconnecting 4 PEs. A crossbar is non-blocking meaning that no contention can occur between several communications if destination PEs are different.

However, several networks were proposed for on-Chip implementations using crossbars. This is for example the case of SCORE [54]. Even if not referenced as a crossbar, through the architecture of the routers and the topology, this network is no more than a crossbar. As expected, very high performances are provided by this network, but the hardware cost is too consequent to foresee a realistic on-Chip implementation. Unfortunately, both the crossbars based NoC presented in [70] and the MCNoC [39] lead to the same conclusion: large crossbars based interconnections present a cost that is prohibitive for on-Chip implementations. This is why crossbars should not be used as the unique interconnection mean but combined with an efficient network topology (embedded in the architecture of the routers). This way, a trade off should be found between the size of the crossbars (that should be minimized) and the resulting network performances (that should be maximized).

Hierarchical network

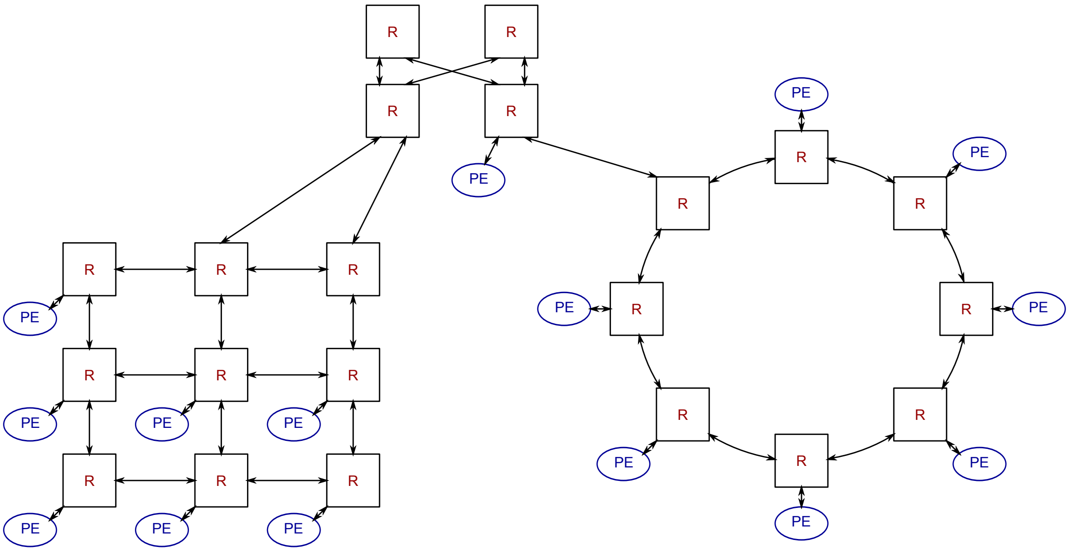


Figure 1.13 : Hierarchical network: a fat-tree based network interconnects both a PE and two sub-networks. Each sub-network can have a dedicated topology that is chosen accordingly with interconnected PEs.

Hierarchical networks are usually composed of several sub-networks linked directly together or interconnected through an over-layer network. Such a network is presented in Figure 1.13, where mesh and ring based networks are interconnected through a fat-tree based over-layer network. Hierarchical networks are particularly suited for large scale applications with a large number of involved PEs. Assuming that every PE do not share exactly the same amount of data with every others, they can be grouped in several sets depending on their communication preferences. This way, sub-networks can be chosen accordingly with the communication graph of the PEs they interconnect. This is how optimal or at least near optimal interconnection sub-networks can be chosen. The advantage of an hierarchical network is also to improve the scalability of the overall interconnection architecture. Indeed, very high-performances can be extracted from sub-networks with a very poor scalability. Interconnecting only a part of the PEs with this sub-network allows to reduce its size, and thus to improve the global scalability. This is the case for example with the XGFT network where the over-layer network is a fat-tree while the sub-networks are shared buses [58].

If hierarchical networks can be implemented in a single chip, they are also well suited for multi-chips implementations. Indeed, each sub-network can be implemented in a dedicated FPGA. Connections between the various FPGAs could be provided through direct links or via a specific chip acting as the over-layer network. So, hierarchical networks allow to foresee future large-scale applications. Furthermore, several researchers already proposed methodologies to guarantee deadlock free routing algorithms in such networks [48]. However, in the case of an application supporting the DPR of the various PEs, sub-networks can no longer be designed optimally. Since placement of the PEs is dynamic, data traffic is the same and finally some inefficiencies can occur with a large amount of communications between several sub-networks. Furthermore a routing algorithm supporting DPR can be quite difficult to embed in a hierarchical NoC. Hence, hierarchical networks should be avoided as the unique interconnection architecture in a DPR system. However, there is no obstacle for networks adapted to the DPR to be part of a hierarchical network (in a multi-chips system).

Custom network

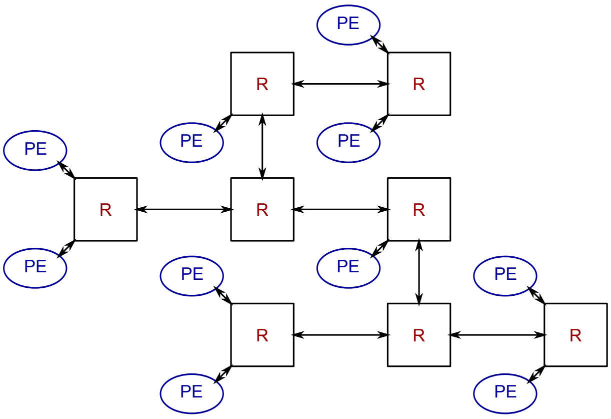


Figure 1.14 : Custom network: the irregular topology is chosen accordingly with the communication graph of the interconnected PEs.

In custom networks, the topology is defined depending on the communication graph that characterizes the PEs. For example, two PEs that share a large amount of data will have a direct link allowing fast communications with maximum bandwidth. This is the case for example for the iNoC network where the overall topology is statically built considering the communication needs of the PEs [85]. However, designing a reliable custom network is not so trivial because of the many inefficiencies that can occur. However key features to make reliable a custom network are provided in [83]. Furthermore, some tools provide an automated framework to design a custom NoC fitting the applicative requirements. This is the case of the μSpider NoC proposed in [37] where the custom network is generated through a dedicated environment using fully parameterizable routers.

Since the structure of the network is irregular, it can be not so trivial to propose a compliant routing algorithm. In the case of custom NoCs, an adaptive routing algorithm can thus be quite interesting. Indeed, such a routing algorithm would adapt the path taken by the data to the structure of the network without for the designer to statically foresee which path must be chosen. Adaptive routing algorithms were already proposed for custom NoCs in [76], or even in [107] which targets the HoneyComb network. This latter NoC is particular due to its octagonal routers in which PEs are included.

In a statically implemented application, it is possible to create a dedicated custom NoC that best fits the communication requirements. However, when aiming a dynamic application, things become even more difficult. Indeed, if the placement of the tasks can not be foreseen at design time, then every possible placement should be checked in order to define the most compliant custom topology. Considering the number of PEs that keeps on growing and thus the number of possibilities, defining a static custom topology of network seem to be quite impractical. This is the reason why a ”general purpose” topology should be privileged.

Conclusion over static NoCs

As presented in this section, the choice of a NoC is always a trade off between network performances and hardware costs (inducing the scalability). These trades off were studied in [91] where the conclusion is that there is no ultimate NoC with both maximal performances and minimal costs. In the case of applications using the DPR paradigm, there is one more parameter that should be taken into account in the global trade off: the flexibility of the network when facing dynamically reconfigurable PEs. This latter parameter modified substantially the conclusion over many powerful topologies that are a good trade off in a static context. At this point, there is no NoC that seems to be perfectly adapted for an on-Chip application using DPR. However, despite of its limited scalability, the fat-tree topology appeared to be interesting due to both its high performances and its flexibility. However, its use in a DPR compliant SoC would require significant improvements concerning its hardware cost.

1.3.3 Flexible NoCs

If some statically implemented networks can lack of flexibility when facing dynamically reconfigurable PEs, applying the DPR directly to these NoCs could improve their flexibility. The DPR could be used to adapt the routing algorithm depending on communication requirements, or even to the network topology itself. Several researchers investigated this dynamic reconfiguration of the networks. This is notably the case for the architecture presented in [31]. This architecture is fractioned into several homogeneous tiles that can be dynamically reconfigured as a PE or a router. This approach is very close from CoNoChi [96] which is presented in Figure 1.15. This latter NoC lies on the same principle: dynamically reconfigurable areas are configured as PEs, routers, or links accordingly with the applicative needs. Both networks provide high performances and CoNoChi is demonstrated in [95] to be more adapted to the DPR paradigm than RMBoC or DyNoC. However, they both suffer from a major drawback: they both use homogeneously sized reconfigurable regions to implement PEs, routers, and even communication links. Since PEs are supposed to heterogeneous, there is no reason for them to require as many hardware resources as routers and moreover links. So, a lot of hardware resources are wasted by both NoCs. Furthermore both of them are not compliant with present technology since the connection of direct links between BMs is not feasible. Moreover the implementation of a PE using several PRRs is impossible: since BMs have to be used, such implementation implies to define two sub-PEs communicating through the BMs.

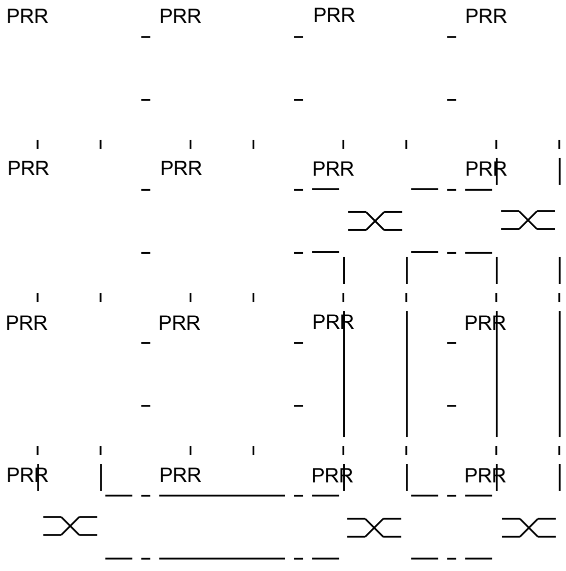


Figure 1.15 : CoNoChi network based on PRRs used to dynamically implement PEs, routers and links.

A different approach is the dynamically reconfigurable NoC which is implemented inside a single PRR [97]. Depending on the applicative needs, the whole network can be reconfigured. This way, topology, routing algorithms, or even flow control techniques can be adapted. Since the network is always implemented in the same PRR, there is no obstacle for PEs to be heterogeneous. However as the DPR can not be applied only to a part of a PRR, this implies that every communication must be halted each time the network is adapted. Or at least each communication whose data path is adapted from the older configuration to the new one. Otherwise, if hardware resources all along the data path are not modified, the communication is guaranteed to happen without any glitch.

The DRNoC network is based on a mesh topology with diagonal links (xmesh) [67]. This network interconnects homogeneously sized PRRs. Large PEs can be implemented using one or several of these PRRs like in DyNoC. More than the xmesh topology, the particularity of DRNoC lies in the reconfigurable routers and network interfaces. So, DRNoC is highly flexible despite a significant hardware cost. Another mesh based NoC is the one proposed in [80]. This network is based on two sub-networks: the packet switched HERMES NoC for control purpose, and a circuit switched mesh network for data transfers purpose. The main particularity comes from the circuit switched network that is dynamically reconfigured by HERMES in order to create direct and dedicated data paths. This way, fast data transfers are achieved with an improved quality of service. This network is particularly interesting even if it suffers from technological limitations such as the reconfiguration times that are of 17.3ms per PRR. However, more reduced reconfiguration delays are expected from improved ICAP interfaces. This way reconfiguration time from the order of 10 μs are expected. Another drawback is the number of partial bitstreams to be stored since they can not be generated online nor relocated. However, this network remains interesting in the use of a circuit switch sub-network that can be adapted on demand.

1.4 Synthesis of the chapter

In this chapter a state of the art was proposed concerning the dynamically reconfigurable architectures. Techniques to use the DPR ability of these architectures were presented along with their constraints. So, if Altera is currently starting to propose DPR compliant FPGAs, at the beginning of this PhD only Xilinx FPGAs were dynamically reconfigurable. DPR occurs in statically defined regions called PRRs through a reconfiguration interface called ICAP.

After the state of the art concerning dynamically reconfigurable devices, some of present interconnection architectures were presented and contrasted with the constraints induced by the DPR. Due to the very large number of communication architectures that were proposed in the world, this state of the art could not be comprehensive. However, most popular architectures were presented. So, if buses are undoubtedly the most popular interconnection means, they face many problems when considering both large scale applications and DPR. This is notably true considering present Xilinx FPGAs that can be reconfigured in 2D. This is why NoCs emerged as being viable interconnection architectures with improved scalability.

Most popular topologies like mesh, torus, ring, tree, or even custom were detailed. Many of them, such as the mesh one are very efficient in a static context but reveal themselves to be quite sensitive to the dynamic location of the PEs. Indeed, depending on the placement of the PEs, data traffic can be concentrated in some areas of the network, resulting in a drop of performances. Fat-tree networks are less sensitive to these placement problems, and are thus more flexible, but they suffer from a significant hardware cost. The design of an interconnection architecture is always a trade off between the hardware cost, the network performances, and the offered flexibility. So, a fat-tree can be interesting for DPR compliant on-chip implementations and even more if the hardware cost can be reduced. Flexible interconnection networks, in the sense that they can be dynamically reconfigured, were also investigated. If some of them seem to be a bit far from present technology, some are very realistic and were already implemented in FPGA. However, every one of these suffers from the technological constraints that are notably important reconfiguration latencies, and statically generated bitstreams that have thus to be stored in the architecture.

Chapter 2

The DRAFT network

In this chapter, a new interconnection network is presented: the DRAFT network. This network aims to be an interesting trade off between hardware resources consumption, network performances, and flexibility. The utilization of this network is also detailed into the scope of applications using dynamic and partial reconfiguration. A dedicated bridge is presented in order for DRAFT to be directly connected to a standard bus. Furthermore, a communication service that will be part of an operating system is also presented in detail. This service provides an abstraction of the communication and of the flexibility induced by the dynamic and partial reconfiguration. This abstraction can improve the designers' acceptance of the DRAFT paradigm.

2.1 Objectives and motivations

The various blocks constituting an application may have different behaviours and natures. For example, blocks can be either hardware or software implemented. An hardware block can be a storage memory, an Intellectual Property (IP) block performing in hardware a specific function, a microprocessor running software instructions, a communication interface allowing to send/receive informations to/from an external component, etc. That is what we define as Processing Elements (PEs). In this report, a software block can be composed of one or several software tasks, a complete OS or subsets of its services, etc. Usually, software blocks are executed on one or several hardware microprocessors.

Communications between software blocks are realized thanks to a storage memory in which data are stored until the destination block reads it. The storage memory is often directly connected to the hardware processor. However, at least one shared memory is required when several processors run in parallel, since every one of them should be able to access it. In SoC, software blocks may also communicate with IP blocks or communication interfaces. Furthermore, also with these software to hardware and hardware to software communications, hardware to hardware ones must be foreseen. Indeed, several IP blocks should be able to interact together, but also with other hardware elements like communication interfaces. Since the cost of a memory architecture dramatically increases with its number of input/output ports, a shared memory with one port dedicated to every PE is not a realistic solution.

This is why hardware communication architectures are needed in real life applications. An appropriate communication architecture should interconnect every PE. This way, hardware to hardware communications can take place without requiring any shared memory, directly sending the data to the receivers. Furthermore, the communication architecture eases software to hardware and hardware to software communications allowing several PEs to access successively a unique memory port or a single communication interface. Since the communication ports associated to interconnected PEs are shared, their hardware cost can be significantly reduced (in terms of used hardware resources). Since PEs are both rapid and efficient, very high performances are required from the communication architecture while its hardware cost should be reduced as much as possible.

However, steady technological evolutions, increasingly complex applications are to be implemented. This led to the DPR of the PEs. DPR highly impacts the communication architecture since dynamic PEs can be instantiated in various areas of a DPR compliant architecture depending on available reconfigurable hardware resources. So, nor the placement of the PEs nor the produced data traffic can be foreseen at design time. Thus the fact that dynamic shared PEs can be placed everywhere depending on the applicative context should not affect the overall performances of the communication architecture.

In this chapter, a new communication architecture is presented in order to be implemented in real life applications using the DPR ability of present FPGAs. Using definitions and networks presented in Chapter 1, an innovative NoC is proposed: Dynamic Reconfiguration Adapted Fat-Tree (DRAFT). The objectives concerning DRAFT are multiple. Indeed, DRAFT should provide very high network performances with a reduced hardware overhead in the context of an application using DPR. The compliance with the dynamic placement of the PEs is called flexibility. A flexible enough NoC should not present network performances varying a lot depending on the placement of the shared element. More than the definition of a flexible NoC, its integration inside real life applications using DPR is also expected in this chapter. For this purpose, communication interfaces are presented as well as a communication service allowing to make a reality the management of software to hardware, hardware to software, and hardware to hardware communications. Furthermore this service supports communications where some of involved PEs are not active/allocated in the RSoC.

2.2 Topology of the DRAFT network

As presented in Section 1.3, there are a lot of existing topologies and furthermore of NoCs. Every topology has its own advantages and drawbacks. This is particularly true when facing the DPR ability of some current FPGAs. However, the fat-tree topology appeared to be flexible enough to support dynamic applications. This topology is very popular in the static context of very large systems [46] but less when aiming on-chip applications even if several fat-tree based networks were proposed [98]. This is due to the scalability which is quite limited in the fat-tree topology. Indeed, when the number of interconnected PEs scales up, the number of required routers dramatically grows up. However, using a fat-tree in an application using DPR can make sense. The main goal of the DPR is to limit the number of simultaneously implemented PEs. Consequently, the number of PEs to be interconnected is limited and we can assume that a NoC interconnecting 64 to 128 PEs is more than enough regarding present applications. Due to the indirect nature of the fat-tree networks, a large amount of communication paths are provided to the PEs. Since these PEs are not distributed inside the structure of the fat-tree, their placement do not impact at all the overall performances of the network. The intrinsic nature of the fat-tree topology ensures that used data paths are always minimal and that no deadlock could occur. So, if its hardware cost could be significantly reduced (and thus its scalability improved), fat-tree based networks would be an attractive interconnection architecture for applications using DPR.

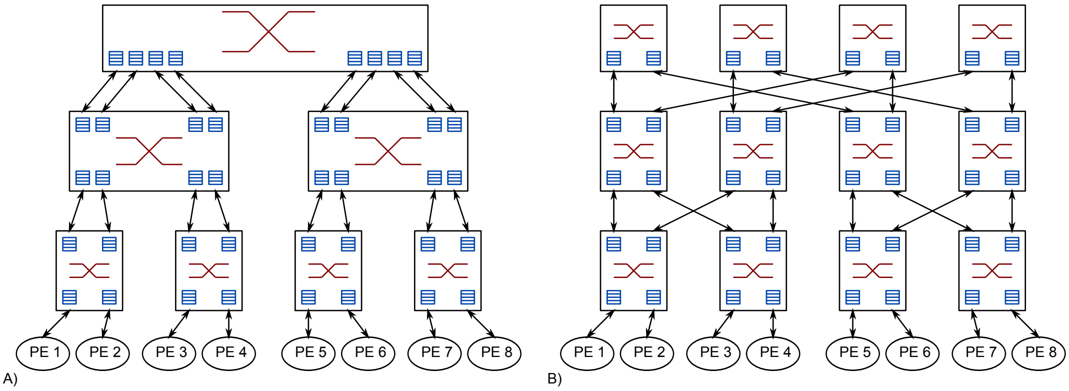


Figure 2.1 : Presentation of two equivalent fat-tree topologies. If A) and B) offer the same number of data paths, and thus the same bandwidth all along the networks, B) has generically defined routers significantly reducing the hardware cost of involved crossbars.

This is the main objective of the DRAFT network: preserving the high level network performances of a fat-tree based network while significantly improving its hardware consumption. The fat-tree topology can be simplified in order to reduce its hardware consumption. Indeed, the fat-tree presented in Figure 2.1.A is equivalent to the one presented in Figure 2.1.B, but the second one has a much reduced hardware overhead. If the number of links and thus of buffers is the same, the crossbars that are embedded inside the routers scale badly with the number of ports to interconnect. Consequently, each time a port is added to a router, its complexity and thus its hardware cost dramatically increases (see Section 1.3.2). This is why the fat-tree presented in Figure 2.1.B has a much reduced hardware cost due to its generic routers with a limited number of ports (4 ports). In Figure 2.1.B, the network is called a $4-radix3-stages$ fat-tree where the $radix$ represents the number of communication ports per router (input/output), and where the $stages$ are the number of hierarchical levels of routers. As of now, only $4 - radixn - stages$ structures are considered for the fat-tree topologies.

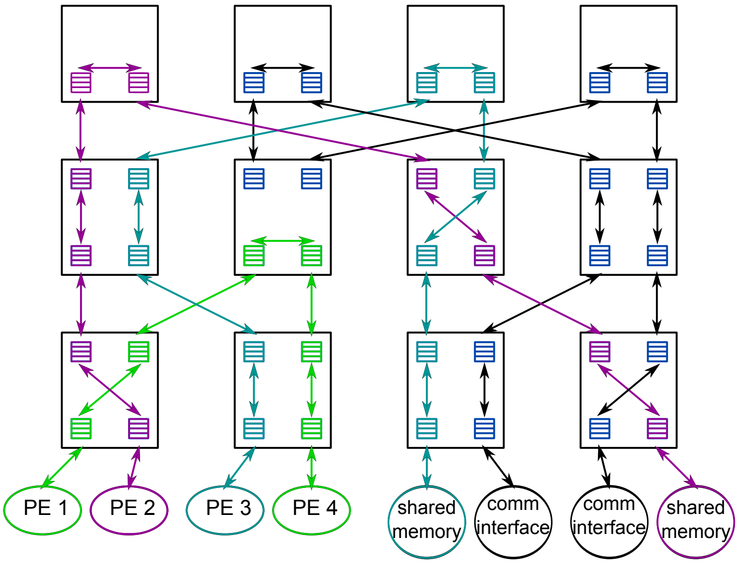


Figure 2.2 : Constrained placement of the shared elements (shared memories and communication interfaces). This placement create a linear data traffic in the right side of the network: data never turn back in the right half to reach an element of the same half of the network. Arrows depicted in routers show the routing path followed by data.

In every application, all the PEs are not dynamically implemented: some of them are static. This is notably the case of the shared memories and of the communication interfaces. If they are always statically implemented, or at least part of them, these elements also present some specificities regarding their communication schemes. Indeed, two shared memories never receive/transmit data directly from/to each other but only from/to microprocessors and IP blocks. This is the same for communication interfaces that never communicate directly with each other but always with microprocessors or IP blocks. This is very important because thanks to both their nature and their communication specificities, their placement can be constrained. As presented in Figure 2.2, these elements can be located in one half of the network, thus producing a purely ”vertical” traffic of data in this half. For example, a data coming from a shared memory will climb the structure of the right half of the network never turning back before the top level router. Indeed there is no PE in the right half the shared memory can communicate with. Since it is the same for other right half connected elements, transmitted data travel through the right half of the network never turning back excepted at the top of the network.

Considering the only vertical directions taken by data in the right half of the fat-tree, the idea came that the routing algorithm could be constrained. Indeed, it

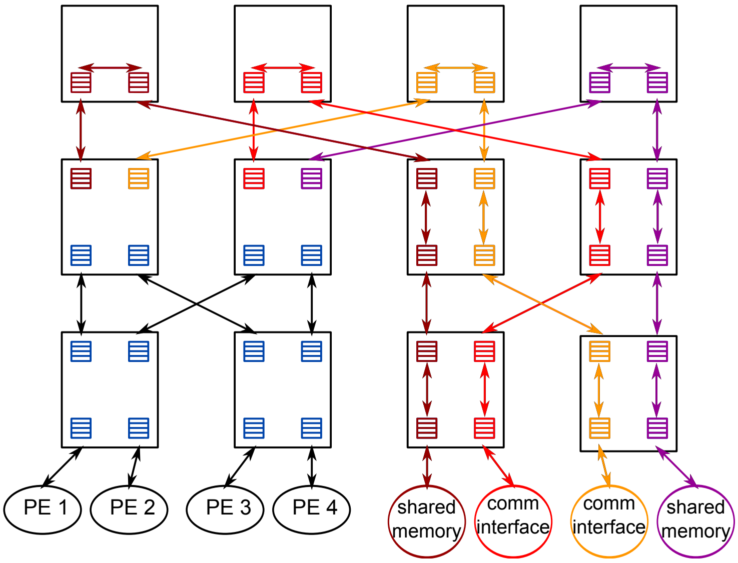


Figure 2.3 : Both constrained placement and routing algorithm. Each shared element has a unique and reserved communication route. All the determination of the routes to take for end-to-end communications is realized in the left half of the network.

is possible with an appropriate routing algorithm to create dedicated route for each shared element. These routes should be accessible directly from the left half of the tree. This way, every communication port of a top level router would be dedicated to a single shared element. This principle is shown in Figure 2.3.

Considering both placement and routing constraints, the routing of the data coming from/to the PEs would be determined in the left half of the network. Since the routing is always the same in the right side of the network and in top level routers, the main idea of DRAFT consists in removing these routers. Previously right side connected PE are then connected directly to the new top level routers. This leads to a $r - radix(n-1) - stages$ fat-tree with half of interconnected PEs directly connected to the top level routers. The topology of the DRAFT network is presented in Figure 2.4

The first advantage of the DRAFT network is the significant reduction of the required number of routers to interconnect a same number of PEs when compared with a conventional fat-tree based network. Thanks to this reduction, the scalability of DRAFT is quite improved. Indeed, considering complete DRAFT networks, a number N of interconnected PEs with $N \geq 3$ leads to a number p of communication ports which is determined by

$$p = 2^{\lceil log_2(N) \rceil}. \tag{2.1}$$

If the radix of DRAFT is r, then the number R_{DRAFT} of required routers is

$$R_{DRAFT} = \frac{p}{r}((\log_{\frac{r}{2}} p) - 1). \tag{2.2}$$

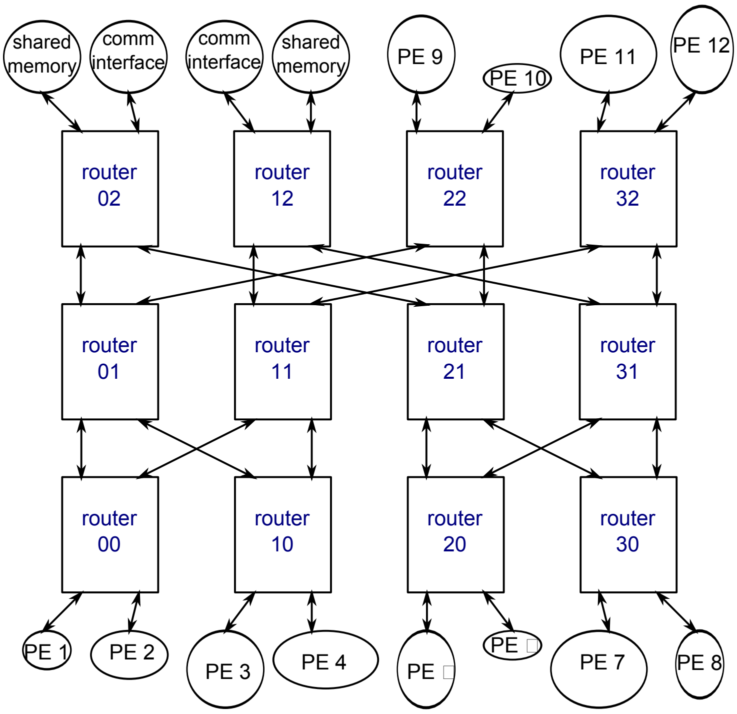


Figure 2.4 : Presentation of the DRAFT network in its $4 - radix3 - stages$ configuration. Half of the PEs are directly connected to the top level routers at the cost of following assumption: top level connected PEs can not communicate together but only with base level connected ones.

Similarly, the number L_{DRAFT} of communication links is

$$L_{DRAFT} = \frac{p}{2}(\log_{\frac{r}{2}} p). \tag{2.3}$$

The scalability of DRAFT compared with these of a conventional fat-tree and a square mesh is provided in Figure 2.5. From this figure, DRAFT appears with a number of required routers significantly lower than for a conventional fat-tree. Furthermore, this number of required routers is very close from the requirements of a square mesh, even if higher between 32 and 64 PEs.

Concerning network performances, the routing of data inside the structure of DRAFT is more constrained than in a conventional fat-tree. However, since the number of routers to be crossed by the data is more reduced, overall performances are still very high as presented in Section 4.3.

Definition0: *DRAFT efficiency is based on a principle that must be respected by application designers: top level connected PEs do not communicate together.*

If the routing algorithm allowed communications between top level PEs, it would result in a dramatic fall of overall performances due to the restriction of the number of data paths. However, if this assumption is respected by designers, there is in fact no limitation concerning the nature of the top level PEs. They can be either

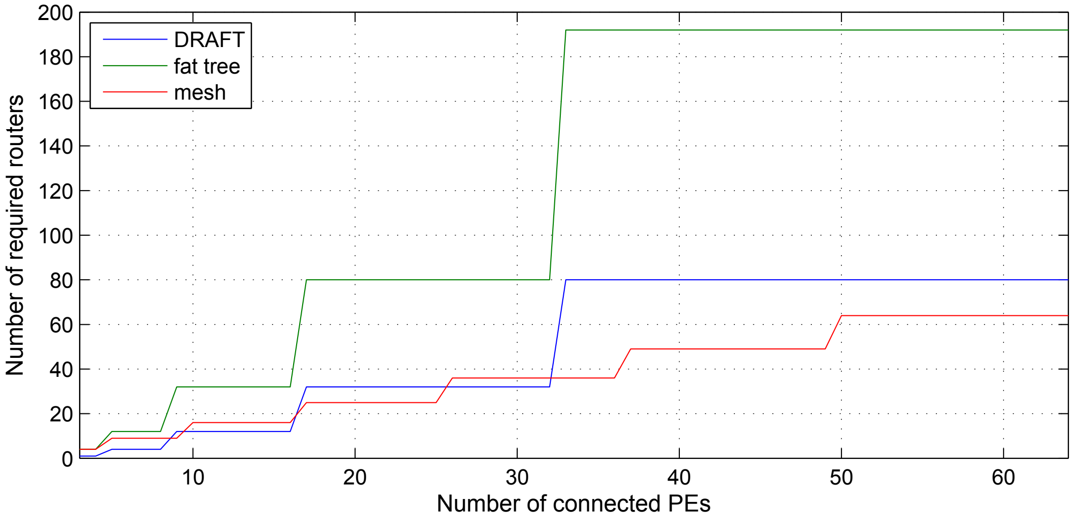


Figure 2.5 : Comparison of the number of required routers between the DRAFT network, a conventional fat-tree, and a square mesh, according with equations 2.2, 1.6, and 1.3.

shared memories or communication interfaces but also microprocessors or IP blocks. Furthermore, top level PEs can be either statically or dynamically implemented. In the case of dynamically implemented PEs, the implementation of these components is just constrained in order for the definition to remain valid.

Concerning the flexibility of DRAFT, there is no restriction for the communications of the dynamically implemented PEs as soon as they are connected to the base level stage of the network. So, base level PEs have no restriction at all leading to a flexibility similar to the one of a conventional fat-tree. Concerning the top level PEs, the flexibility is limited due to the assumption concerning the communications.

DRAFT was built considering that half of connected PEs (including shared memories and communication interfaces) do not require to communicate together. The case where more than half of the PEs do not communicate together is obvious: some of them can be connected to the base level routers. However, the case where more than half of the PEs require no communication restriction directly impacts the NoC. In this latter case, the size (the $n - stages$ value) of the network should be chosen accordingly with the number of PEs simultaneously connected that require to communicate together. Consequently, the overall flexibility of the DRAFT network is less important than for the conventional fat-tree based networks. The definition of the topology of DRAFT is more like a trade off between overall flexibility and the scalability of the network. Furthermore, DRAFT is defined considering present applicative requirements leading some PEs to have reduced communication needs (in terms of number of PEs they communicate with).

Concerning the DRAFT network topology itself, one can see many similarities with multi-stages networks. If a folded Beneŝ is similar to a conventional $4 - radixn - stages$ fat-tree [46], a Butterfly network looks similar with the DRAFT network [33]. The arrangement of the routers is similar in a Butterfly network and in DRAFT. However, there is a major difference between these two networks: routers and communication ports in a Butterfly network are unidirectional whereas they are bidirectional in DRAFT. This induces that DRAFT interconnects two times more PEs than the Butterfly network, and routing strategies are completely different (and so are the data traffics). This is why even if they look similar, they are different networks.

Table 2.1 : DRAFT API provided to each PE. Directions are defined considering the network, so an input of the network should be connected to an output of the PE.

Region	Signal name	Size (bit)	Direction	Function
Data reception	clock_rx	1	In	clock reference on which others incoming signals are synchronized
	data_in	16/32/64	In	data to be transmitted are injected here
	rx	1	In	notifies that valid data are present on data_in
	credit_out	1	out	indicates to sender if further data can be sent
Data emission	clock_tx	1	Out	clock reference on which outgoing signals are synchronized
	data_out	16/32/64	Out	transmitted data arrive here
	tx	1	Out	notifies that valid data are present on data_out
	credit_in	1	In	receiver indicates if further data can be received

The topology of a network does not define only the way the routers are arranged and interconnected, it also defines the transmitted signal along with their size. So, every router has four communication ports that are each composed of two parts namely the reception part ”rx” and the transmission part ”tx”. Thus, all links inside the network are bidirectional. Both rx and tx parts have the same interface. They are composed of a data line which size is user defined fitting the required data sizes (16, 32, or 64 bits). Along with this data line, two control signals are provided.

One (rx or tx) is to signify that a valid data is present on the data line. The other signal can have two different significations and thus names. Depending on the flow control strategy, this signal indicates that a valid data path was created from the source to the destination of the data (ack), or that next node (either a router or a PE) has sufficient storage capacities to receive the data (credit). Finally, a clock line is also provided. All these signals are summarized in Table 2.1.

2.3 Router architecture

The DRAFT network is based on a $4 - radixn - stages$ fat-tree topology composed of generically defined routers. So, each router has exactly the same architecture. Every router is composed of four input/output communication ports, a controller, and a crossbar. The generic architecture of a router is presented in Figure 2.6. Each router is identified by a unique address. This address gives the relative position of the router inside the NoC through its XY coordinates. The first part of the address correspond to the number of the router in its *stage* starting from the left, while the second part correspond to the *stage* number. This addressing of the router is presented in Figure 2.4.

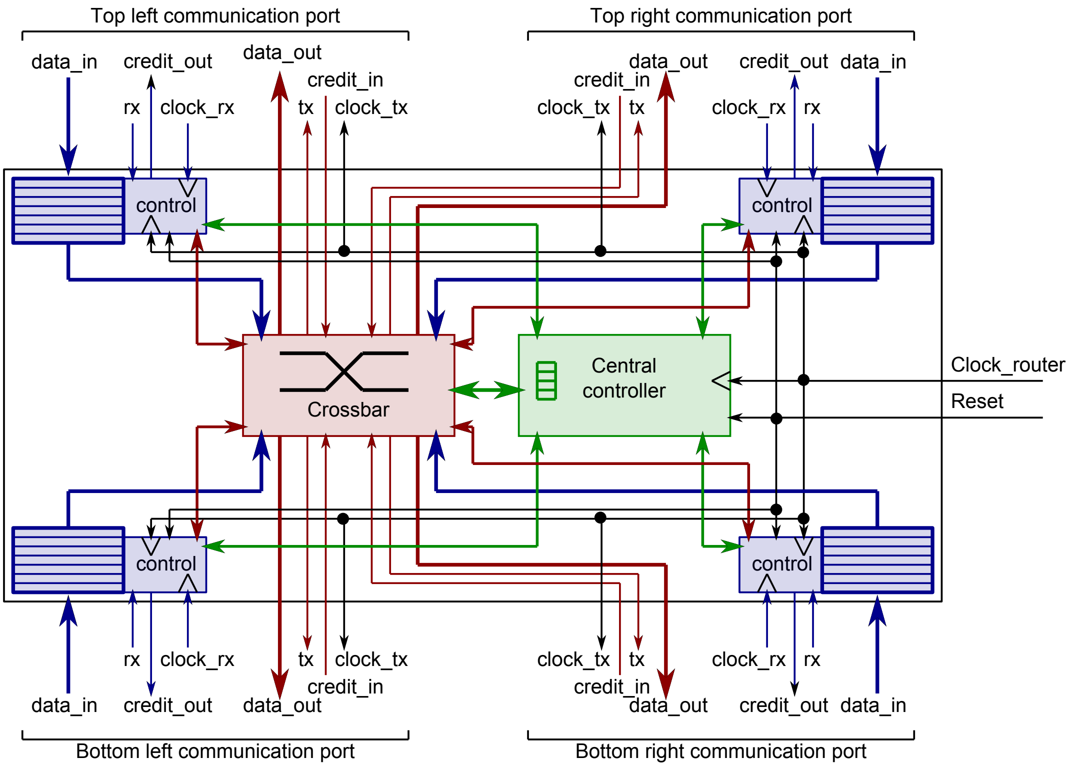


Figure 2.6 : Generic architecture of DRAFT routers. A router is composed of four input/output ports, a central controller and a crossbar. Inputs are buffered for asynchronous communications.

Each router receives clock and reset signals. If the reset signal can be used to force the router in its initial state, the clock signal is very important. Indeed, each input port of a router is composed of a bi-synchronous buffer making the relation between the clock domains of the sender neighbour (clock_rx) and of the router (clock_router). The central controller is synchronized on clock_router in order to guarantee a valid behaviour. Transmitted data are always clocked using the router's internal clock reference: clock_tx = clock_router. This clock adaptation has two goals: first it can realize the clock adaptation between the PEs and the network if clock references are different, and second it improves the QoS of the NoC. Indeed, providing exactly the same clock reference to all hardware resources embedded in a chip is very difficult. Due to parasitic effects, clock signals can be degraded and skew can occur between two regions of the chip. This phenomenon is called jitter. The fact that each router synchronizes incoming data on its internal clock using buffers guarantees the coherency of transmitted data and a valid behaviour of the central controller. Furthermore, if a router is not solicited at all by the data traffic, its frequency can be lowered (and even set to zero) for power consumption limitation purpose.

The bi-synchronous buffers located at each input communication port are based on a First In First Out (FIFO) strategy meaning that incoming data are stored and read in their order of arrival. Each buffer has a capacity that is defined by designers at synthesis time. A single buffer can store 4, 8, 16, or 32 incoming flits. When a buffer is full, and so no further data can be stored until at least one is read, the credit signal (if any) is set to "0". This indicates to the sender that no further data should be transmitted. Each buffer has a dedicated controller that makes the relation with both the central controller (for example informing that a new communication has arrived), and the crossbar. When the central controller acknowledges the creation of a data path, the buffer controller sends the first flit to the crossbar. Only made of combinatorial logics, the crossbar is used to route data from an input port to the appropriate output port accordingly with informations provided by the central controller. If for some reasons a data can not be transmitted (incoming credit line to zero), the information is immediately notified to the source buffer controller in order for it to stop the reading (and thus the sending) of stored data. This permits to handle 4 communications in parallel inside every router.

The central controller performs the routing algorithm, and manages accordingly the crossbar in order to route incoming data to their destination. Further explanations concerning how functions the central controller are provided in Section 2.4.

2.4 Routing and Flow Control

Data are packet switched through the DRAFT network. It means that instead of creating an exclusive communication link between source and destination PEs (circuit switch), DRAFT transmits data through its structure only using addresses that are included at the beginning of each packet. So, data paths are used (successively) by multiple couples of source/destination PEs. DRAFT is based on a Wormhole flow control. This means that packets of data (composed of several flits) travel through the network without being stored completely in one router. So, if a packet is composed of sufficient flits, these ones are distributed in each router all along the chosen data path. This strategy significantly reduces the requirements in terms of storage capacities since routers do not need to store the whole packet nor a large amount of flits.

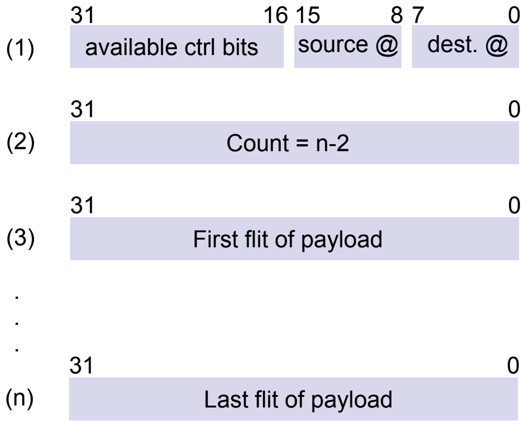


Figure 2.7 : Presentation of the various flits composing a packet of data. The first one, called header contains source and destination addresses. The second one (called count) contains the number of following flits of data. Then next flits contain the payload.

Concerning the routing of data, two addresses are used: the source PE address and the destination one address. These addresses are always included in the first half of the first flit that is called Header (see Figure 2.7). These addresses are dependent on where PEs are connected to the DRAFT network. Indeed, each communication port of DRAFT, and thus each PE, is numbered linearly starting from the left of the base level to the right of the top level. Binary addresses of both PEs and routers are presented in figure 2.8.

The width of the addresses is defined by the scale of the network. However, these addresses are also linked with the width of the flits, and thus the data width: the size of an address represents a maximum of quarter of the header. Since large scale networks require large addresses, the size of the network influences the minimal data width. For example a DRAFT network interconnecting 32 PEs $(4-radix4-stages)$

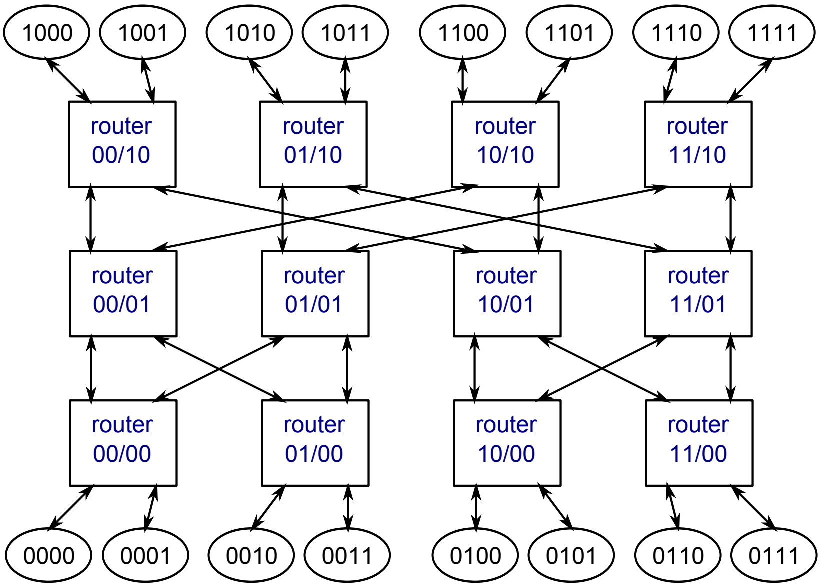


Figure 2.8 : Addressing of both PEs and routers in a $4 - radix3 - stages$ DRAFT network.

do not support 16 bits data width because 5 bits are necessary to code the addresses of connected PEs. 16 bits data are only compliant with DRAFT networks embedding between 1 and $3 - stages$ of routers (respectively from 2 to 4 bits necessary to code the addresses of the PEs). The remaining half of the header is provided to PEs if some control informations are to be transmitted. These control informations can be the internal address inside a shared memory, a software process id, etc.

The routing algorithm is based on two phases. The first one is quite common for fat-tree topologies, it is called the TurnBack algorithm [46]. This phase is used for the communication between two bottom level connected PEs. Data are routed toward upper stages until the destination PE is reachable only climbing down the structure of the NoC. This algorithm is used for example in the conventional fat-tree based network we used for comparison purpose, see Chapter 4. In this phase, when a data should continue on climbing the structure of the tree to reach its destination (bottom level connected PE), the emission port is chosen depending only on the source addresses. A router from the first stage reads the first bit of the source address starting from the right, i.e. the Least Significant Bit (LSB). If "1", data is transmitted through the top right communication port of the router. If "0", the top left communication port is chosen. A router located in the second stage of DRAFT reads the second bit of source address, and so on to the top stage of routers. Since the same destination PEs (base level ones) can be reached from both top left and right communication ports, this algorithm allows to balance the data traffic: data from different PEs always take different paths to climb the structure of the tree.

Obviously, since only one path is possible to reach the destination PE when data climb down the network, chosen data path is only dependent from the destination addresses. To continue latter example, when a data climbs down the network and arrives in a router located in the second stage, the second bit from the destination address starting from the right is checked. If ”0” data is transmitted through the bottom left communication port. If ”1”, the bottom right port is chosen. When arriving in a router from the first stage, similarly the first bit of the destination address is checked. This algorithm minimizes the likelihood of conflicts when data climb down the structure of the tree. It also ensures that no deadlock nor livelock can occur and chosen data paths are always minimal. This phase of the routing algorithm is presented inside the DRAFT network in Figure 2.9.

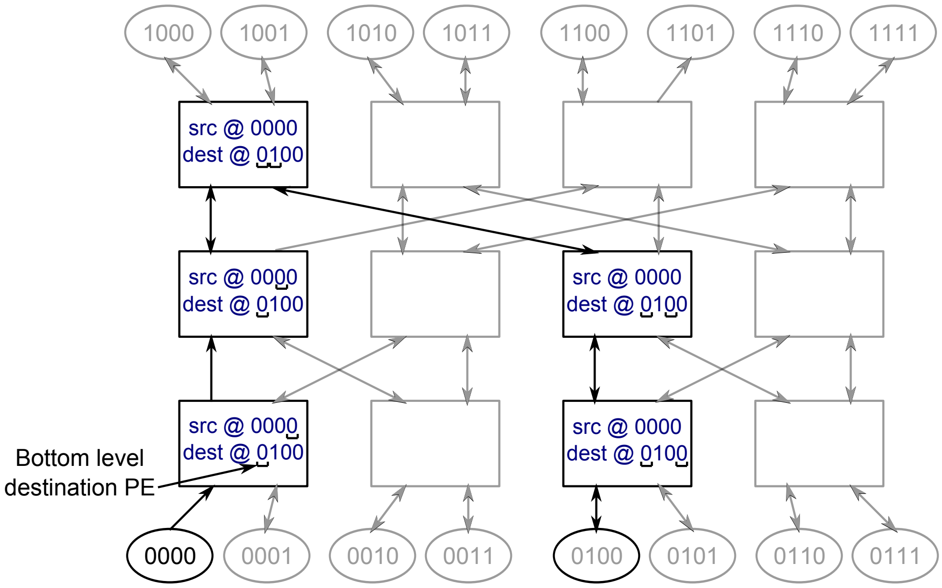


Figure 2.9 : Routing of data toward a bottom level connected PE. The MSB of the destination address indicates its connection at the base of the network. Source address is used to determine the data path when data climb the structure of the network. When they climb it down, the destination address is considered.

The second phase is specific to the DRAFT network since it consists in routing the data from base level PEs to top level ones using the only one possible data path. The routing is realized in each router using masks that are specifically generated depending on their own addresses. Thanks to these masks, the location of the destination PE is first determined depending on the destination address. For example, in the case of the $4-radix3-stages$ DRAFT network presented in Figure 2.8, the top level location of the PE is verified through the first bit starting from the left of the destination address, i.e. the Most Significant Bit (MSB). The determination of the data path in a $4-radixn-stages$ DRAFT network is realized as follow: if the router is part of the first stage of the network, the second bit of the destination address (starting from the right) is checked. If the router is part of the second stage, the third bit starting from the right of the destination address is checked. This continues the same way until the top stage routers. If the router is part of the top stage, then the LSB of the destination address is checked. For every router, wherever they are located, if checked bit is equal to ”0”, data is transmitted through the top left communication port. If equal to ”1”, similarly, the data is routed toward the top right port. This guarantees that the data can reach its destination for every couple source/destination PEs. This routing is presented inside the DRAFT network in Figure 2.10.

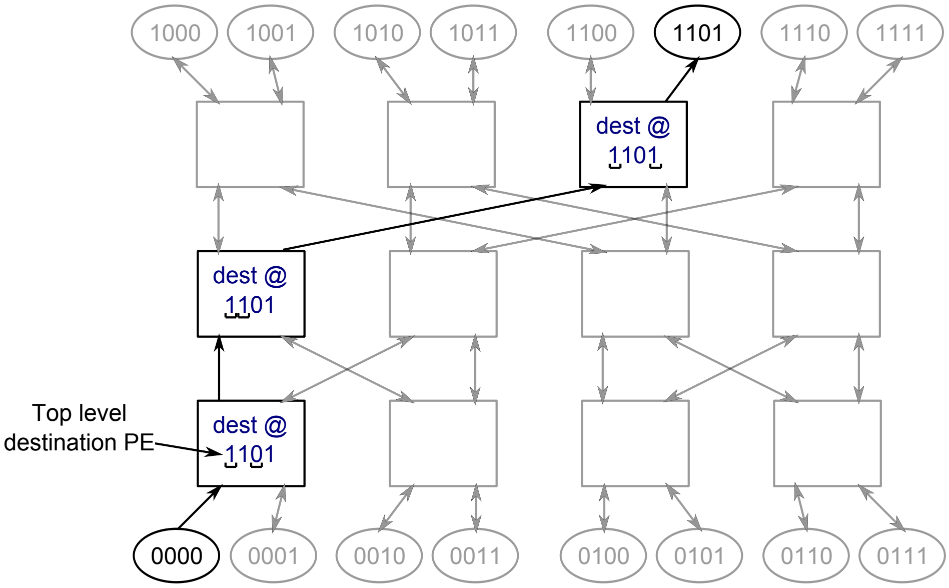


Figure 2.10 : Routing of data toward a top level connected PE. The MSB indicates the connection at the top of the network, other bits of the destination address are used to define the data path.

The link between the two phases of the algorithm is realized by the central controllers. Routing is performed as follow: if data are to be transmitted to a top level PE, the second phase is used, else it is the TurnBack algorithm which is used. In case of a conflict between two communications, the action depends on the destination of the communication that encounters the conflict. If the destination PE is connected to a top level router and the only possible communication port is already used, then there is no other choice than waiting for its availability. However, if the destination PE is located in the base of DRAFT and data should climb to upper stages, then the communication uses the other top level communication port of the router. Due to the $4-radix$ nature of DRAFT, since only the two bottom level ports can access the two top level ones, the remaining top level communication port of the router is always available. Furthermore, the fat-tree based structure of DRAFT guarantees that no deadlock, nor livelock can occur even in case of multiple conflicts. Finally, also due to the fat-tree based structure of DRAFT, chosen data paths are always ensured to be minimal.

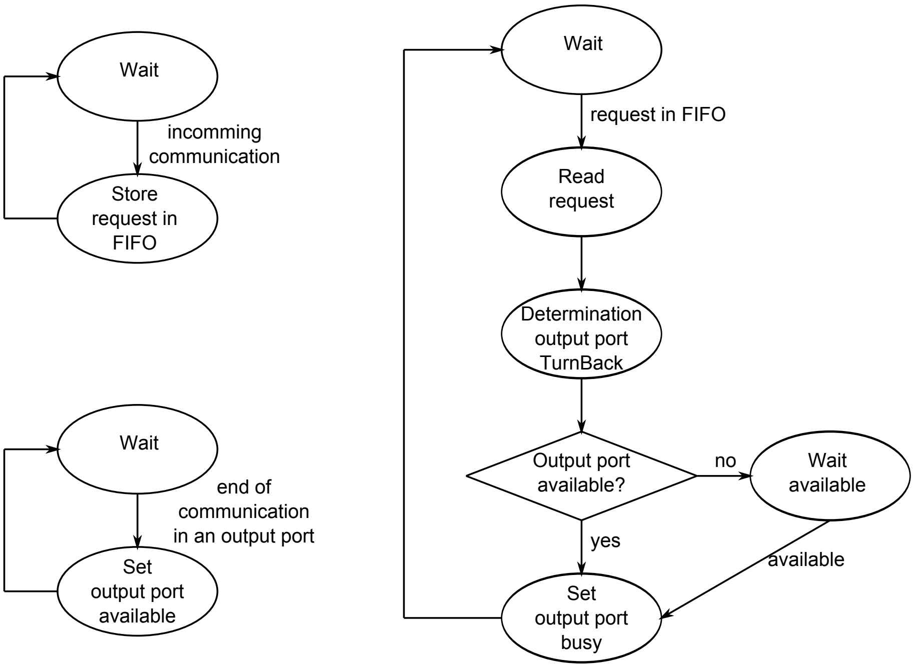


Figure 2.11 : Routing algorithm (TurnBack) from a conventional fat-tree based network.

Concerning the data that climb down the structure of DRAFT, if a conflict happens between two communications trying to use the same data path (or a portion of), then one is privileged by the concerned central controller and the second one waits for the other to complete. Without minding if data climb up or down the structure of the NoC, central controllers are based on FIFO scheduling policies giving to the various communications a fair probability to be realized. Routing algorithm of a conventional fat-tree based network is provided for comparison purpose in Figure 2.11. The complete routing algorithm of the DRAFT network is presented in Figure 2.12.

In order for the router to know when a communication is finished, a count information is always included in the second transmitted flit. This count indicates the number of following flits forming the payload. The count can not be inferior to 1 corresponding to at least one flit of payload. In each router, this count information is used to calculate if a communication is completed depending on the number of transmitted flits. As it could be seen from Figure 2.12, the central controller does not wait for current communication to be completed before processing a new communication

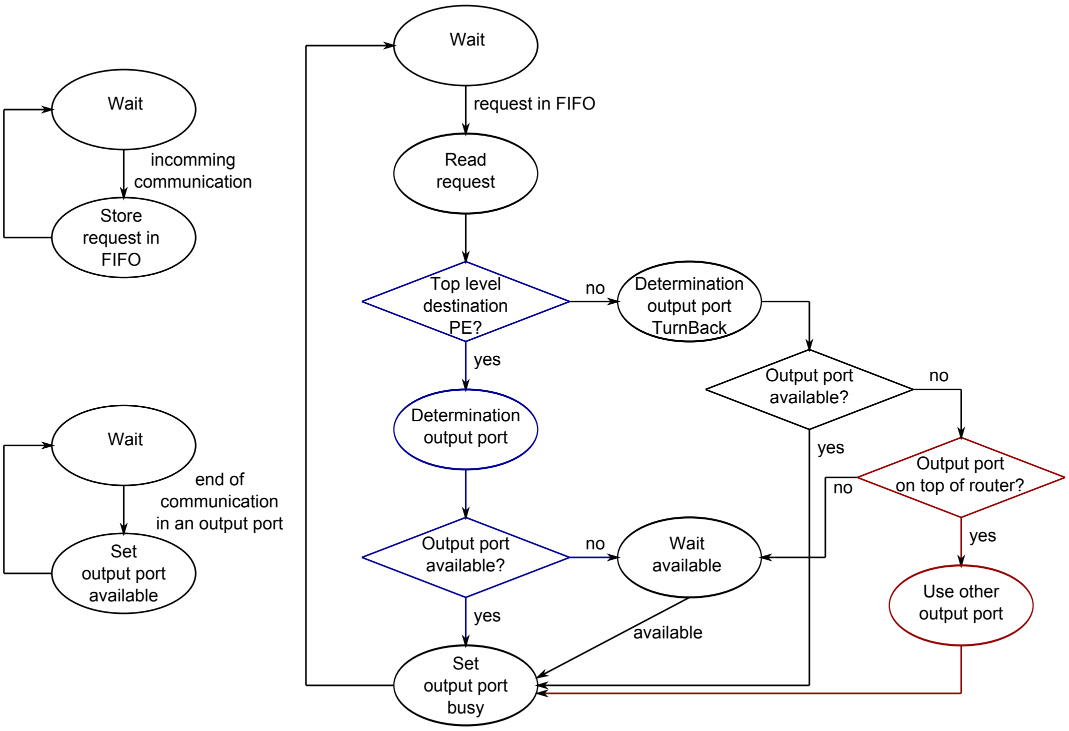
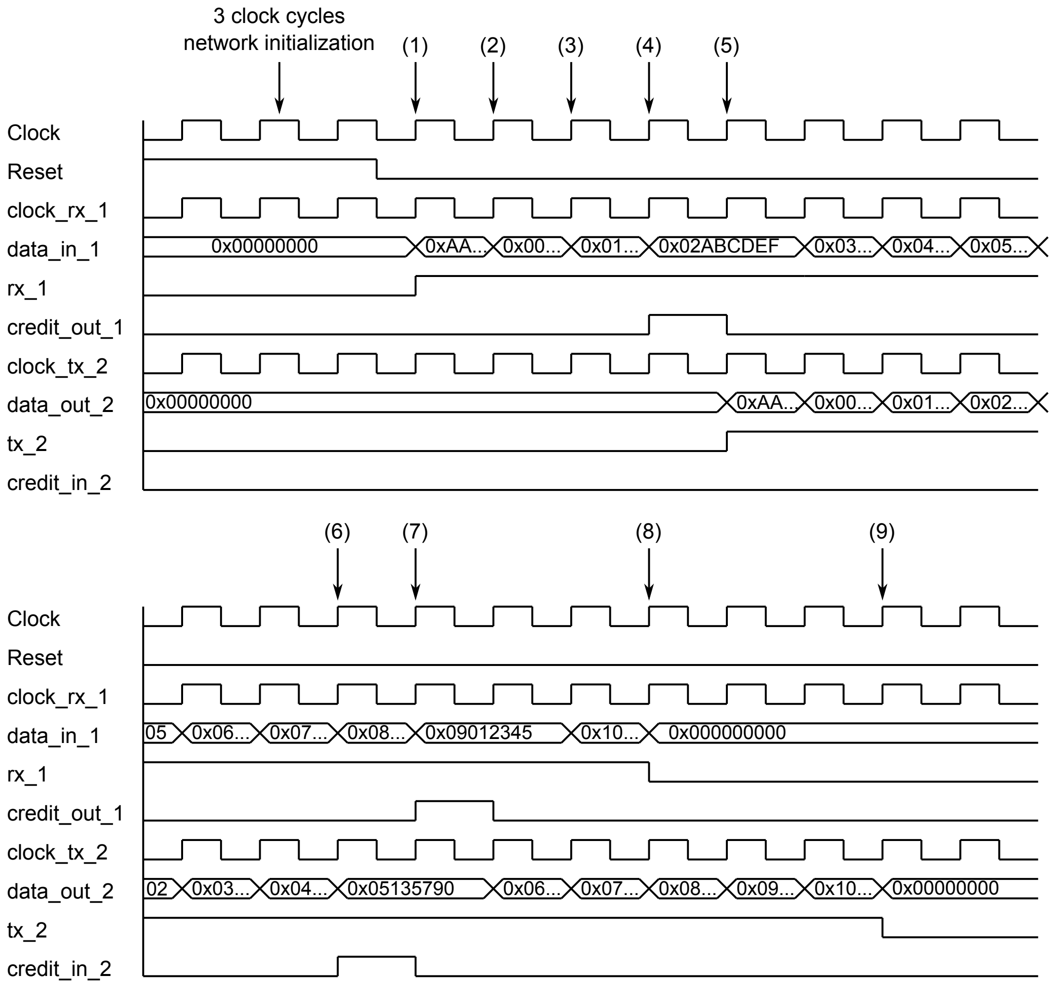


Figure 2.12 : Routing algorithm from the DRAFT network: the classical TurnBack phase is depicted in black while the second phase is in blue. Conflicts resolution is pictured in red.

request. Excepted in case of a conflict, this permits to save a lot of time for routing purpose. Thus, depending on destination addresses, up to four communications can take place simultaneously in every router $(4 - radix)$.

Chronograms of a communication are presented in Figure 2.13. In this scenario, PE 1 initiates a communication toward PE 2. The initialization of DRAFT takes 3 clock cycles. So, reset should be set to ”1” during 3 clock cycles each time the chip embedding DRAFT is powered on. The communication starts in (1) with the rx_1 line indicating that a data is present on data_in_1 line: the header of the communication. Count information and first payload flit are respectively transmitted in (2) and (3). Routing can take several clock cycles, in this case if the input buffer is full, credit_out_1 line is set to ”1” to prevent further data from arriving (4). In (5) the header has been transmitted through communication port 2 (data_out_2) so that the buffer from communication port 1 can receive one more data. Data are transmitted until (6) where neighbour element connected to communication port 2 informs that no more data should be transmitted (credit_in_2 = 1). If the buffer from communication port 1 becomes full, once again credit_out_1 line is set to ”1”. Last data arrives to the router in (8) and is transmitted in (9), thus ending the communication.



(1) Communication Header: 0xAAAA0001 (AAAA: user defined address ; 00: source PE 1 ; 01: destination PE 2)
(2) Count: 0x0000000A (10 transmitted flits of payload)
(3) First flit of payload: 0x01234567
(4) Input (4 flits capacity) buffer is full, delay during routing
(5) Header arrival time to PE 2
(6) 1 clock cycle saturation of PE 2 for some reasons
(7) Input buffer is full, wait for PE 2 to be available again
(8) End of communication for sender PE 1
(9) End of communication

Figure 2.13 : Chronograms presentation of DRAFT for a communication initiated by PE 1 (network address 00) toward PE 2 (network address 01) with 4 flits depth buffers.

2.5 The DRAGOON environment

DRAGOON is an environment inherited from ATLAS [81] allowing automatic generation and simulation of networks. The development of the DRAGOON environment was realized in parallel with the conception of the DRAFT network itself. Since network performances of DRAFT had not been evaluated at this time, the choice was made for DRAGOON not to generate directly DRAFT but a conventional fat-tree. Indeed, the DRAFT structure can be easily extracted from a fat-tree, but the reverse is not so trivial. This concept is explained in Figure 2.14.

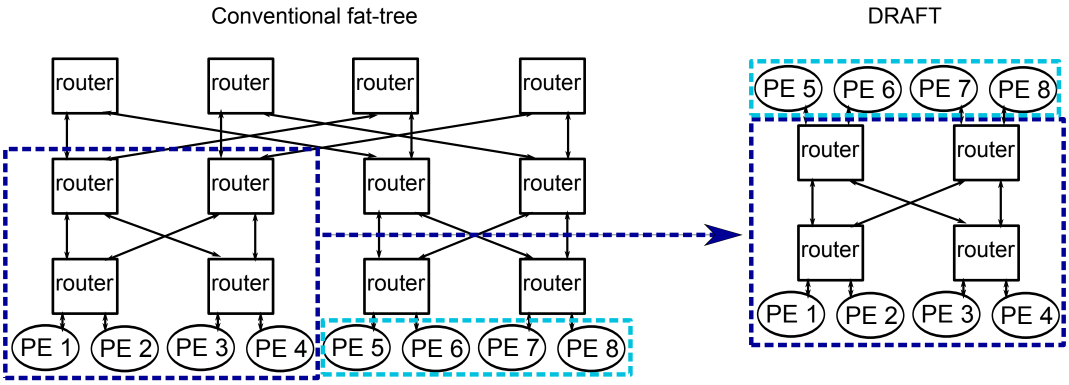
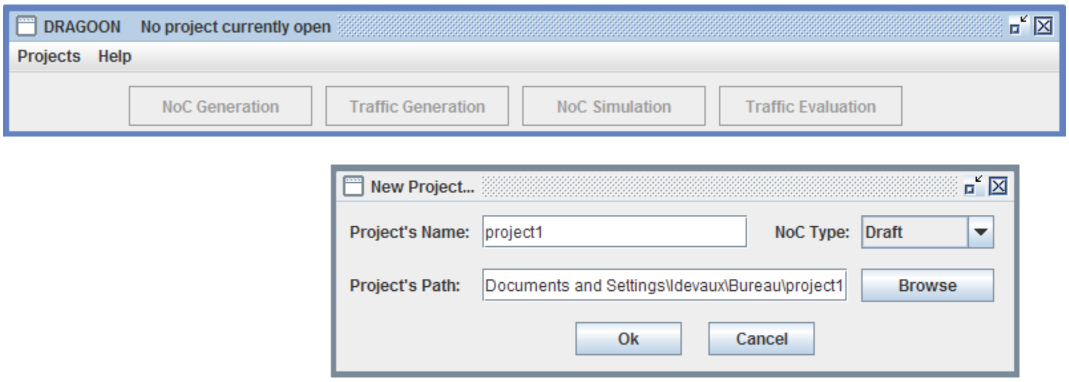


Figure 2.14 : Conventional fat-tree generated by DRAGOON. This fat-tree embeds in its structure the DRAFT network.

In the remaining of this section, the various functions composing DRAGOON are detailed. Every time needed, further explanations will be given in order to switch from the conventional fat-tree to DRAFT.

2.5.1 DRAGOON main interface



Project's Name: project1

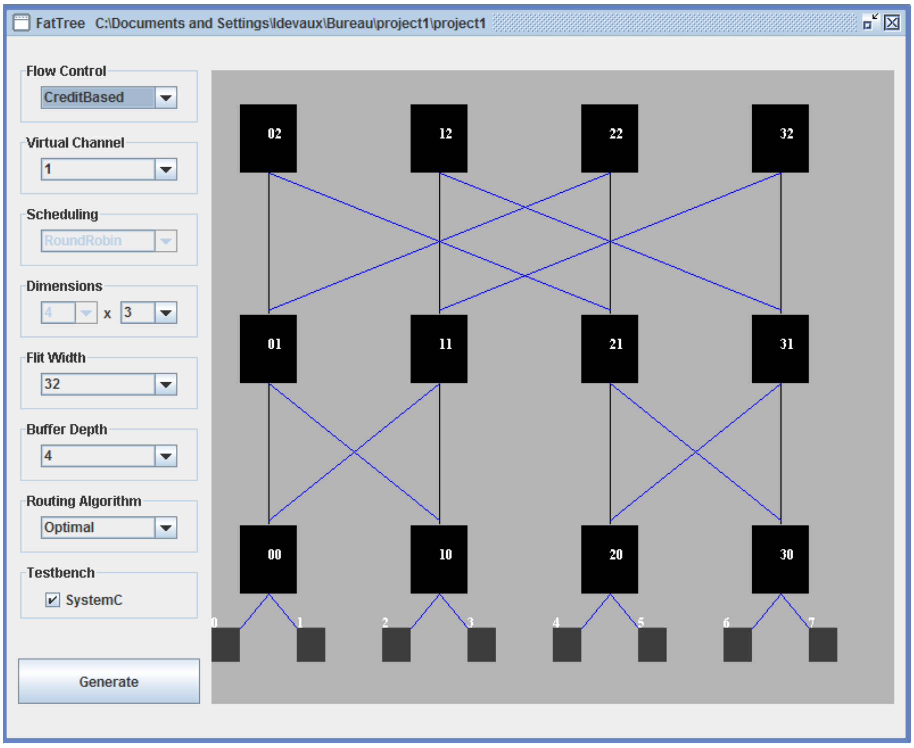
NoC Type: Draft

Project's Path: Documents and Settings\ldevaux\Bureau\project1

Figure 2.15 : Main panel from the DRAGOON environment. This is an overlayer for embedded tools.

The DRAGOON environment is composed of four main functionalities that can be chosen from the main panel. This panel, presented in Figure 2.15 is indeed an overlayer for four distinguished tools: the NoC generator, the traffic generator, the simulator, and the traffic evaluator. Using these tools requires to have a project that can be created from this main panel. In this first version of DRAGOON, only the DRAFT type of network can be selected (generating a fat-tree). However, improvements are scheduled in order to make this panel as an overlayer for both the DRAFT network and the fat-tree, and also interfacing with the overall ATLAS environment.

2.5.2 The NoC generator


FatTree C:\Documents and Settings\ldevaux\Bureau\project1\project1

Flow Control: CreditBased

Virtual Channel: 1

Scheduling: RoundRobin

Dimensions: 4 x 3

Flit Width: 32

Buffer Depth: 4

Routing Algorithm: Optimal

Testbench: ☑ SystemC

Figure 2.16 : NoC generator tool allowing to parameterize the fat-tree that will be generated.

The NoC generator presented in Figure 2.16 allows users to automatically create a conventional fat-tree with desired configuration. Produced fat-tree is generated at the RTL level (coded in VHDL). Several parameters of the generated fat-tree can be specified from this tool:

- The dimension: this parameter defines the number of hierarchical levels constituting the fat-tree, and so the number of PEs that can be connected to it. A representation of the fat-tree that will be generated is pictured on the right side of this panel. Fat-tree can be generated interconnecting from 4 up to 128 PEs.

- The flit width: this is the width of the data that are exchanged through the network. Three data width are available: 16, 32, and 64 bits.

- The buffer depth: this depth represents the number of data (flits) that can be stored in every buffer. Buffer depths can be parameterized to 4, 8, 16, or 32 flits. This depth should be configured depending on the data traffic that can be foreseen from the application.

- The number of virtual channels: for high network performances purposes, virtual channels can be defined. Three values are possible: 1, 2, or 4 representing respectively 0, 1, or 3 additional path(s) for every port of every router. This parameter should be used with extreme caution because of resulting hardware overhead that dramatically increases with the number of virtual channels.

- The flow control policy. By default CreditBased, this parameter can also be set to Handshake, thus replacing the credit signals in every communication port.

- The Testbench option. This option, asserted by default, indicates if the generated code of the network will include additional signals (coded in SystemC) for simulation purpose. These lines are only for simulation with the DRAGOON environment. For any other purpose, this option should be un-checked or SystemC lines should be removed from the NOC.vhd file.

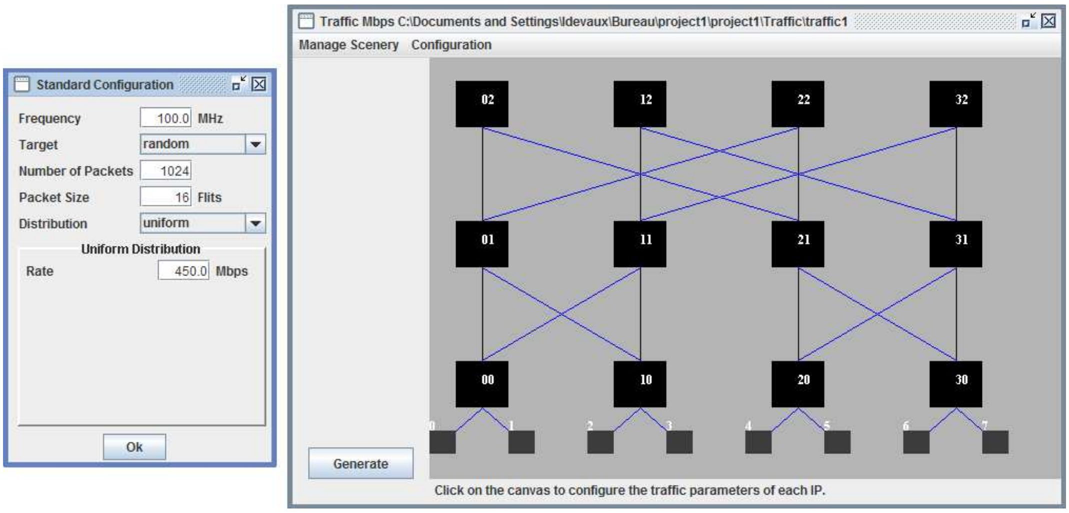
When clicking ”Generate”, a fat-tree is generated fitting user specifications in the project directory. At this point, if the DRAFT network was expected to be generated instead of a conventional fat-tree, modifications should be done manually in the NOC.vhd file. Two cases should be distinguished:

- the NoC is to be simulated with the DRAGOON environment. In this case, additional lines in SystemC are included at the end of the VHDL file (NOC.vhd). These lines make the interface between the RTL level NoC and the simulator connecting every router and every port of the network. In order to avoid any problem and thus to save time, no routers should be removed. So, to obtain the DRAFT network, all top level routers also with all routers from the right half of the network should have their clock signals connected to a constant value (0). Furthermore, input/output ports from the right half should be connected in the same order directly to the top of remaining left half routers, thus in place of top level routers.

- the NOC is destined to hardware implementation or simulation in another environment. In this case, all top level routers can be removed also with right half routers. Input/output ports should then be connected to the new top level routers (see Figure 2.14).

This need to modify manually the vhdl file (NOC.vhd) is quite impractical for designers. However, the direct generation of DRAFT will be integrated in the next version of the tools that will also be compliant with ATLAS.

2.5.3 The traffic generator


Standard Configuration

Frequency 100.0 MHz

Target random

Number of Packets 1024

Packet Size 16 Flits

Distribution uniform

Uniform Distribution

Rate 450.0 Mbps

Traffic Mbps C:\Documents and Settings\ldevaux\Bureau\project1\project1\Traffic\traffic1

Click on the canvas to configure the traffic parameters of each IP.

Figure 2.17 : Overview of the traffic generator tool.

The traffic generator, presented in Figure 2.17 is undoubtedly one of the key feature of DRAGOON. The traffic generator allows to generate a traffic pattern that will be injected into the NoC for simulation purpose. Before generating a traffic, a scenery should be created. This scenery is in fact a view of the NoC (pictures in front end window) also with the configuration of the traffic. Every PE located at the based level can be parameterized individual simply clicking on it in order to generate a specific traffic. Otherwise, clicking on ”Configuration”, the traffic pattern will be parameterized for all PEs. Several parameters can be configured:

- The frequency: this indicates the operating frequency of the PEs. It is usually set to the frequency that is expected from hardware implementation.
- The targets: this parameters indicates if destination of generated data packets should be chosen randomly or specifically.
- The number of packets: this represents the number of data packets that will be sent by every PE.
- The packet size: this parameter indicates how many data (flits) should be transmitted in every packet.

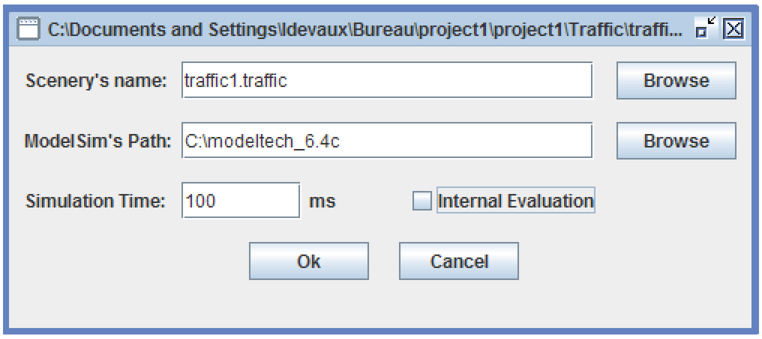
- The transmission rate. In bits per second, this value indicates at which rate the generated packets should be injected in the network. This constitutes the offered traffic.

- The distribution. This is the most important parameter to be configured. Indeed it indicates how packets should be transmitted by each PE: uniformly in the time, following a Normal distribution (periods of low emissions followed by a peak of high emission, then returning to low emissions), or in burst mode (bursts of data followed by periods of silence).

When generating the traffic, the generator creates a text file for each PE containing data to be sent also with the instants they are to be injected in the network (in clock cycles). An example is provided in Appendix A.

2.5.4 The NoC simulator

NoC simulations are not realized completely by DRAGOON. Indeed, the NoC simulator is an overlayer using ModelSim 6.4c. The NoC simulator configures ModelSim and provides generated network and traffic patterns. As presented in Figure 2.18, users have the possibility to choose which generated traffic (if several) should be injected in the NoC, also with the duration of the simulation.


C:\Documents and Settings\ldevaux\Bureau\project1\project1\Traffic\traffi...

Scenery's name: traffic1.traffic

ModelSim's Path: C:\modeltech_6.4c

Simulation Time: 100 ms

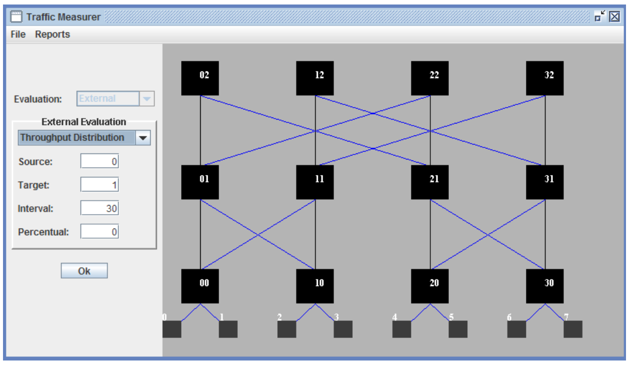
□ Internal Evaluation

Figure 2.18 : Presentation of the simulator of networks that acts as an overlayer fo ModelSim 6.4c

From ModelSim simulation, output text files are generated for each PE. These files contain the received packets also with the time the last data of these packets were received at. An example of simulation is provided in Appendix A.

2.5.5 The traffic evaluator

Finally, the last tool is the traffic evaluator which is presented in Figure 2.19. This tool takes into account the files generated both by the traffic generator and ModelSim



Evaluation: External
Source: 0
Target: 1
Interval: 30
Percentual: 0

Draft NoC
Flow Control: CreditBased (1 cycle per flit)
Virtual Channels: 1
Scheduling: RoundRobin
Dimensions: 4 x 2 (4 routers)
Flit Width: 32
Buffer Depth: 4
Routing Algorithm: AlgorithmOptimal

		Generated			Measured									
			Throughput (%)	Latency (clock cycles)		Throughput (%)				Latency (clock cycles)				
Source	Target	Packets	Average	Ideal	Packets	Average	Standard Deviation	Minimum	Maximal	Average	Standard Deviation	Minimum	Maximal
0	1	0	0.0	25.0	570	17.43	10.59	1.0	32.0	29.11	0.75	29.0	40.0
0	2	0	0.0	35.0	487	15.49	11.07	2.0	46.0	43.47	7.67	39.0	62.0
0	3	0	0.0	35.0	505	16.41	11.90	1.0	60.0	46.92	11.01	39.0	79.0
1	0	0	0.0	25.0	528	16.75	10.66	1.0	31.0	24.0	0.0	24.0	24.0
1	2	0	0.0	35.0	543	16.94	11.24	2.0	55.0	38.41	8.59	34.0	62.0
1	3	0	0.0	35.0	491	15.86	11.14	1.0	48.0	43.46	9.44	34.0	62.0
2	0	0	0.0	35.0	540	17.61	11.42	2.0	37.0	41.06	6.00	34.0	49.0
2	1	0	0.0	35.0	534	17.34	11.76	2.0	45.0	42.01	8.34	34.0	54.0
2	3	0	0.0	25.0	488	16.08	10.54	1.0	36.0	24.31	1.65	24.0	33.0
3	1	0	0.0	35.0	1	NA	NA	NA	NA	35.0	0.0	35.0	35.0
		0			4687	14.99				36.77			

Report generated on 20:30:21 Monday, June 13, 2011, AD by TrafficMeasurer

Figure 2.19 : View of the traffic evaluator and example of generated results.

in order to compute latencies and throughputs. These two parameters are summarized in a web page giving performances both for each couple source/destination PEs, and for the whole system. Number of received packets also with minimal, maximal and average throughputs and latencies (with standard deviation) are the key results provided by this tool.

All characterization results of DRAFT also with comparisons with traditional fat-tree and HERMES are detailed in Chapter 4.

2.6 DRAFT implementation advices

Since many applications require data transfers between static and dynamic PEs, they should be connected to DRAFT. For this purpose, designers can connect the static PEs directly to the routers, or with a bus based sub network (multiple buses for example) connected to a single router, like for the XGFT network. Then, the sub network and its connected PEs (statically implemented shared memories, communication interfaces, etc.) are viewed by DRAFT as a single PE. On the other hand, the static PEs which do not exchange data with dynamic ones should be interconnected through a separate communication architecture optimally designed for this purpose. This communication architecture can be shared multiple buses, or NoC based. Similarly, the shared elements which do not communicate with dynamic PEs but only with static ones should be connected to a sub network rather than to DRAFT. These principles are advised to reduce the size of DRAFT and so its resource consumption. Doing so, DRAFT can be seen as an independent core of network or even as an IP block interconnecting each part of the application.

For applications using the DPR, DRAFT does not directly connect the dynamically implemented PEs through their network interfaces, but through Bus Macros (BMs). This connection is presented in Figure 2.20.

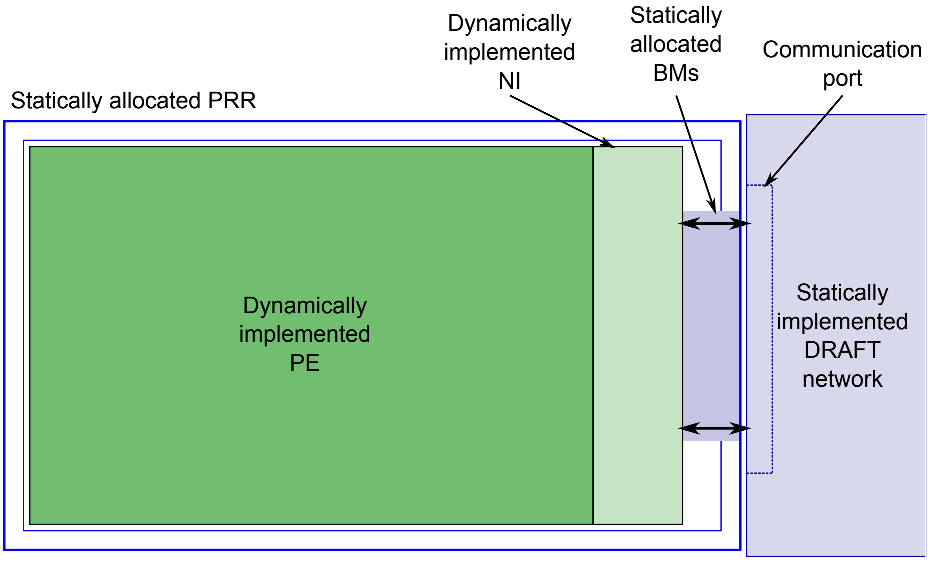

Dynamically implemented NI
Statically allocated BMs
Communication port
Statically allocated PRR
Dynamically implemented PE
Statically implemented DRAFT network

Figure 2.20 : PRR receiving a dynamic PE connected to DRAFT through the dynamic NI and the static BMs.

The concept of dynamically reconfigurable NIs is important because they can be designed optimally for their corresponding PEs. This allows to reduce the hardware cost of the NIs when a PE does not have the same interface as the others. So during the dynamic reconfiguration of a given PRR, DRAFT interface remains the same even if the newly allocated PE presents a specific interface. This concept makes DRAFT more generic and more flexible considering the location of the PEs.

DRAFT placement is important in the conception of a system using the DPR, because it impacts the use of the reconfigurable resources as well as the network performances. Thus, the solution presented Figure 2.21 is to implement DRAFT as a central column into the FPGA. This solution is particularly adapted to current technologies supporting the DPR: Xilinx Virtex series. PEs (static or dynamic) are implemented into both halves of the FPGA. Since DRAFT is not distributed into the FPGA, the designer is less constrained by the network for the definition of the PEs in terms of sizes and locations. This is an advantage for the implementation of heterogeneously sized PEs. Thus, the implementation of DRAFT is fully compliant with current technology and the DPR requirements.

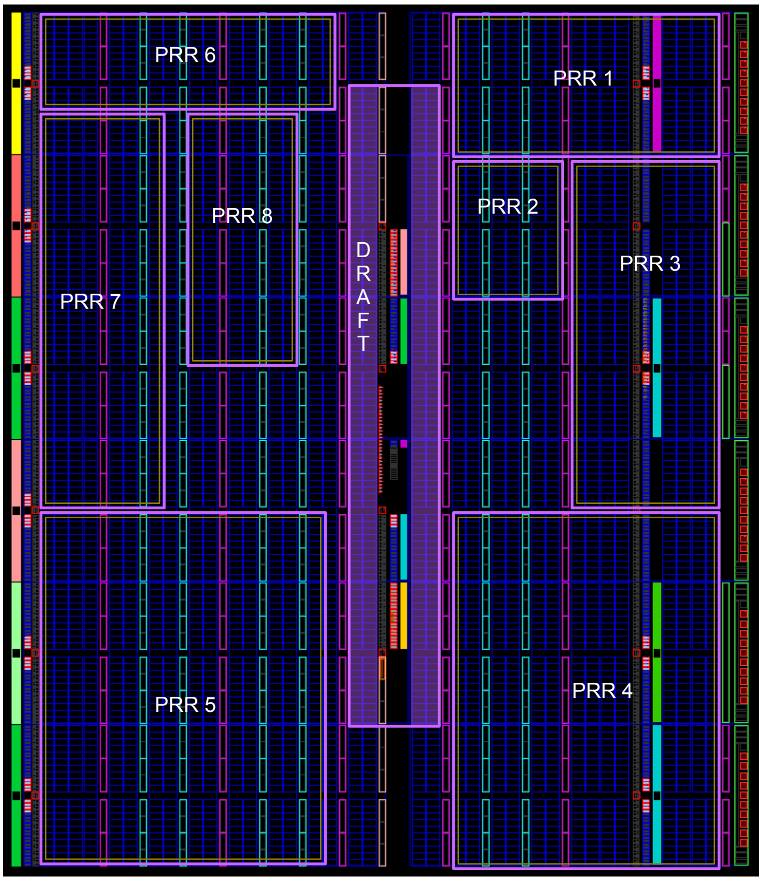


Figure 2.21 : Implementation of DRAFT as a central column interconnecting PEs located into both halves of the FPGA. DRAFT implementation is captured from Xilinx PlanAhead 12.4 design software targeting a Xilinx Virtex V implementation (XC5VSX50T).

Furthermore, in Xilinx FPGAs, communication interfaces should be located into the central Input Output (IO) bank column. Similarly, we recommended to locate the shared memories into the nearest BRAM columns of the central column (for small capacities but high performances, otherwise DDR memory should be used). Thereby, the communications between the PEs and the shared elements encounter a minimal latency.

2.7 DRAFT integration: interface DRAFT/AHB

More than just defining a NoC adapted for applications using DPR, the integration of this NoC into present applications is a key element of this PhD. For this purpose, the compliance of DRAFT with existing communication architecture was studied. Into the framework of the FOSFOR project, along with partners, the use of a LEON 3 processor for control purpose was decided. Indeed, LEON 3 processors are defined using AHB buses for communication purpose. This is why a bridge making the relation between DRAFT and the AHB bus was designed. However, one should notice that several bridges can be found on Internet making the relation between AHB and PLB or OPB buses. This is important because through these bridges DRAFT is directly accessible from MicroBlaze, PicoBlaze, or PowerPC microprocessors.

An AHB Bus typically contains the following components:

- AHB master: Bus masters initiate read or write operations providing address and control signals. Only one bus master is allowed to actively use the bus at once.
- AHB slave: Bus slaves respond to a read or write operation and signals back to the master the success, failure or waiting of the data transfer.
- AHB arbiter: Bus arbiters ensure that only one master at a time is granted access to the bus and to initiate data transfers.
- AHB decoder: The AHB decoder is used to decode the address of each transfer and provide a select signal for the slave involved in the transfer.

General architecture of an AHB bus is presented in Figure 2.22. Data bursts are supported by the AHB bus. Data burst is a transmission technique mainly used to transfer data to/from shared memory. It consists in creating a data path on which several packets are transmitted without any interruption. This avoids to request a new data path for each packet. Data bursts can be of a predetermined length (4, 8 or 16 packets) or of undefined length otherwise. The bursts can be incrementing or wrapping bursts. The difference between them is that in incrementing bursts, the address is continuously incremented after each data packet sent, while a wrapping burst wraps the address after the designated length of transfer. All burst transfers are limited to a 1 kByte address boundary. A dedicated signal allows to configure the bursts. This is particularly useful for memorization purpose of data. This way, data transfers from software to hardware or hardware to software communications are accelerated.

To successfully integrate a DRAFT network in a system based on an AHB Bus it must contain an AHB slave interface so that data can be transferred to it. It also

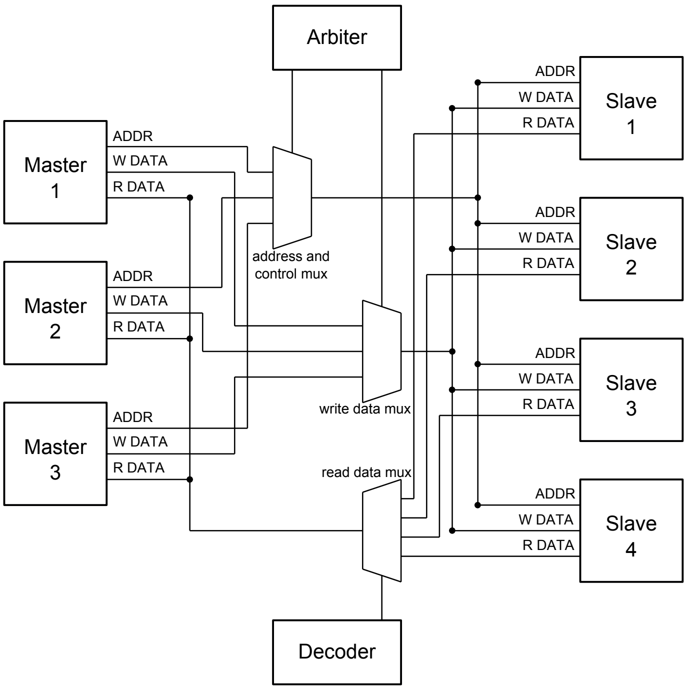


Figure 2.22 : Presentation of the general architecture of an AHB bus interconnecting 3 master to 4 slave elements.

must contain an AHB master interface in order to transmit the data received toward other components. An AHB slave interface responds to data transfers initiated by an external AHB master. The slave determines if it has been selected to perform a communication by checking its selection signal. The address, control and data signals are generated by the AHB master. After it detects the selection, the slave processes the data available on its inputs and drives the appropriate response. In case it does not manage to process the data in one cycle, it can request the master to maintain the data available for one or more cycles by driving low a ”ready” signal. An AHB slave requires an address, so it can be referenced by a master.

The AHB slave interface processes any incoming data and sends them to the DRAFT network by driving the appropriate control signals needed by the network, in the required order. This implies that the AHB to DRAFT bridge has AHB slave inputs and outputs and also signals to connect to an input port of the DRAFT network. The AHB to DRAFT bridge implements the functionality of responding with a busy signal when the DRAFT network encounters a communication delay. An overview of the AHB to DRAFT bridge is presented in Figure 2.23.A. An AHB master wanting to send data through the DRAFT network should send the packets presented in Figure 2.23.B.

Similarly, the overview of the DRAFT to AHB interface is presented in Figure

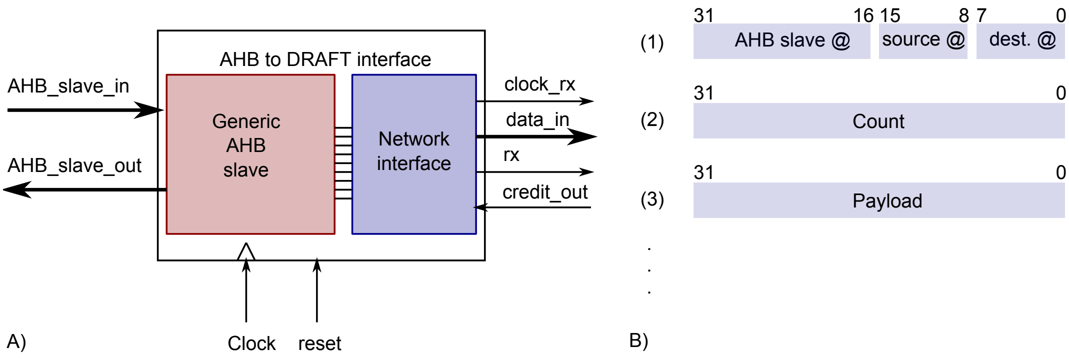


Figure 2.23 : A) Overview of the AHB to DRAFT bridge embedding an AHB slave that forwards data received from an AHB master to the DRAFT network. B) Data transmitted by the AHB master to the slave (and thus through DRAFT).

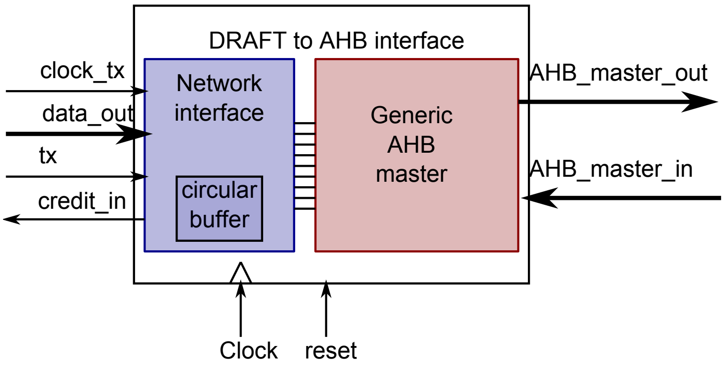


Figure 2.24 : Overview of the DRAFT to AHB bridge embedding an AHB master that forwards data received from DRAFT to the AHB bus accordingly with the AHB arbiter.

2.24. An easy approach of building such an interface for the DRAFT network implies the use of the generic AHB master interface provided by the GRLIB library [40]. Generic AHB slave interfaces are also provided by this library. The generic interface of an AHB master contains the logic necessary for the handling of inputs and outputs from AHB arbiter and AHB slave. The AHB master interface contains logic for requesting access, driving addresses, data and control signals toward the AHB arbiter and receiving response signals from it and from an AHB slave. Also an AHB master contains logic to wait and maintain valid data until the AHB slave finishes processing the current transaction.

In order to connect this generic master interface to the DRAFT network, a DRAFT to AHB controller was built. The DRAFT to AHB controller has the role of reading the input data received from DRAFT and driving the appropriate control signals to the AHB master interface. It uses a buffer (FIFO) to store the data packets received (an unknown number of data packets) and then it sends them to the master interface, driving the inputs as required by the AMBA AHB specification.

The whole bridge, with both AHB to DRAFT and DRAFT to AHB interfaces, was validated in RTL simulations using ModelSim 6.6c and implemented in a Xilinx Virtex V (XC5VSX50T) FPGA using the Xilinx ISE 12.4 design tool software. This way, communications using bursts of data from a LEON 3 microprocessor to a shared memory through the DRAFT network could be realized. For this purpose, the automatic incrementation of destination addresses was used by the master part of the bridge.

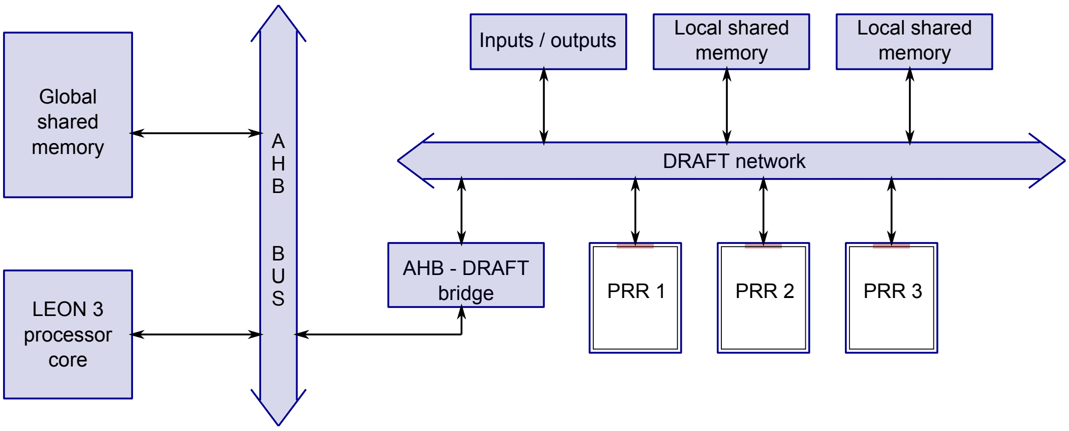


Figure 2.25 : Presentation of the FOSFOR platform using a LEON 3 processor core, a shared memory connected to the processor through an AHB bus, and PRRs interconnected by DRAFT. PEs corresponding to hardware tasks are dynamically implemented in the PRRs. Communications between the AHB bus and the DRAFT network are realized thanks to the AHB-DRAFT bridge.

Following the definition of a generic communication interface making the relation between the DRAFT network and the AHB bus, DRAFT was implemented inside the FOSFOR application. The global architecture of the considered hardware platform is presented in Figure 2.25. It is composed of a LEON 3 processor core, which manages the platform, and reconfigurable resources fully connected with DRAFT. DRAFT also ensures the connection between the processor core and the reconfigurable resources through the AHB-DRAFT bridge. This permits to support data exchange between software tasks (running on processor core), and hardware tasks (PEs instantiated in the PRRs). The bridge should be at the base of the network to have no communication restrictions.

2.8 DRAFT communication service

An OS supports numerous software communication techniques, such as signals, pipes, semaphores and shared memory which are the usual techniques. Similarly, this kind of communications can be implemented for PEs to improve the flexibility of the DPR compliant systems. In order to support such communications, an hardware communication service was studied in order to enable and manage the communications between pairs of PEs. This communication service is particularly adapted to the DRAFT network, thus easing its integration inside applications (FOSFOR platform or other applications with a different OS). The communication is supported by establishment of a communication channel between two tasks. The communication service abstracts the information about the PRRs where the PEs are implemented and establishes the data communication between the tasks. Thus, the communications are ensured without any designer's knowledge about the location of the PEs.

The communication service provides several functions. The first one is the "info/signal Handling". The communication service has the information about the current state of all the tasks (software or hardware) to take appropriate decisions on the system calls or signals from the other tasks. The status is maintained in a shared service table. This table is the central database where all the system calls are registered. The communication service has a dedicated Finite State Machine (FSM) for each port of DRAFT monitoring the communication specific system calls from the running PEs. Thus, it can support the simultaneous communications between different pair of tasks.

The second function concerns the service initialization and termination. The communication service creates a virtual communication channel between the pair of PEs which requires a connection. The PEs use the OPEN system call to open the channel. If the communication service finds two open requests from PEs (one in send mode, the other in receive mode) with the same channel number, a communication channel is established between them. This is referred to as service initialization. OPEN system call requests the communication service to open a particular channel in either read or write mode. Whenever a PE requests to open a new channel, the communication service acknowledges it with a channel ID for further communication using this reserved communication channel. Once the data communication is over between a pair of PEs, the communication channel is closed using a CLOSE system call. This is referred as service termination. The communication channel checks whether the data communication is over between the two PEs before closing the channel. Once the channel is closed, it is available for the other PEs to establish a new communication.

The third function concerns inter-task communications. Once the communication channel is opened and acknowledged by the communication service, a Send-Data or Receive-Data signal along with the Task ID, Memory Pointer, Port Address, Data Size are sent to the communication service. These informations are updated in the service table. SEND/RECEIVE system calls request the communication service for the physical address (NoC port address) of the receiving/sending PE along with the channel ID received during OPEN call. The communication service acknowledges the SEND/RECEIVE system calls with the physical addresses and initiates data communications through DRAFT NoC.

To ensure the communication between two PEs, the communication service manages the following configurations:

- The two PEs are instantiated at the same time. In this case, the data exchange can be supported by DRAFT and no additional storage is needed for the data. The data are directly transferred from (internal memory of) the sender PE to (internal memory of) the receiver. These two internal storages are completely masked, and the Network Interfaces use Direct Memory Access (DMA) to exchange data at high speed. For this purpose, sender PE should receive from the communication service the network address of the destination one.

- The source PE is not instantiated when the receiver one calls the communication in receive mode. In this case, the destination PE is suspended until the sender calls the communication channel in send mode. At this moment, the two DMAs can be initialized and the transfers between the two PEs can start.

- The destination PE is not instantiated when the sender starts the communication. In this case, the OS can suspend the sender until the receiver becomes ready to receive data. This type of behavior delays the execution of the sender PE.

Since the last case is not satisfactory, several new possibilities are offered to the OS when the destination PE is not instantiated while the sender PE starts the communication:

- A part of the global shared memory can be chosen for the temporary data storage. In this case, the OS allocates sufficient space in the memory and provides the address of this memory to the interface of the source PE. For the source PE, it seems that the destination is ready and the data transfer can start. The interface of sender PE sends the data through the NoC, and DRAFT drives the data to the allocated global shared memory through the bridge. This communication scheme is presented in Figure 2.26).

- A local shared memory can be chosen for the data storage. This case is very similar with the previous one, but the allocation is done within a memory placed in the reconfigurable area. In Figure 2.25, for example, this memory is connected at the top of DRAFT. The OS provides local memory network address to the source PE. So, for the source PE, it seems that the destination

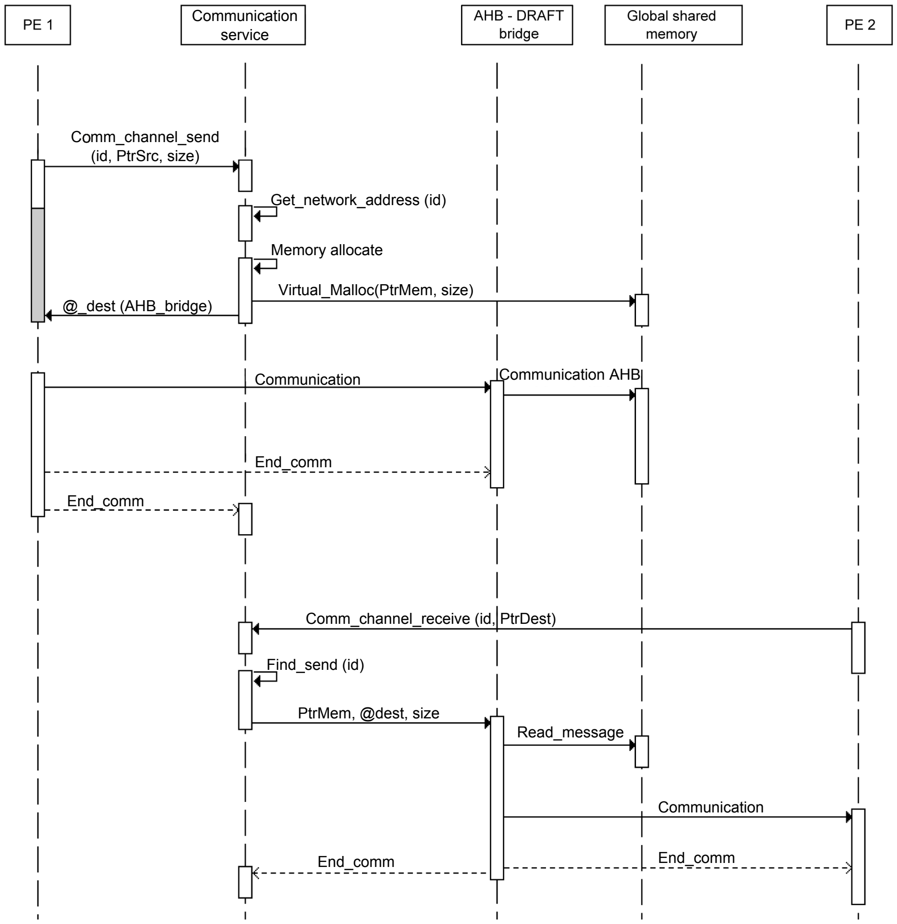


Figure 2.26 : Sequence diagrams of communications between two PEs using the global shared memory.

one is ready and the transfer can start. DRAFT drives the data to the allocated local shared memory.

- DPR allows to propose a solution by the way of dynamic creation of temporary memory. To ensure a maximum flexibility to the application, the OS is extended with temporary memory allocation within the available memories in the platform. If a PRR is not used to support a PE during a time interval, its logical resources can be used for temporary data storage. For example, it is possible to use the memory blocks (BRAM) present in a PRR to store locally data during task exchange. If no PRR are available, the global shared memory can be used for this purpose. To do that, the memory capacity of each PRR should be evaluated. This way, each PRR is defined compliant with

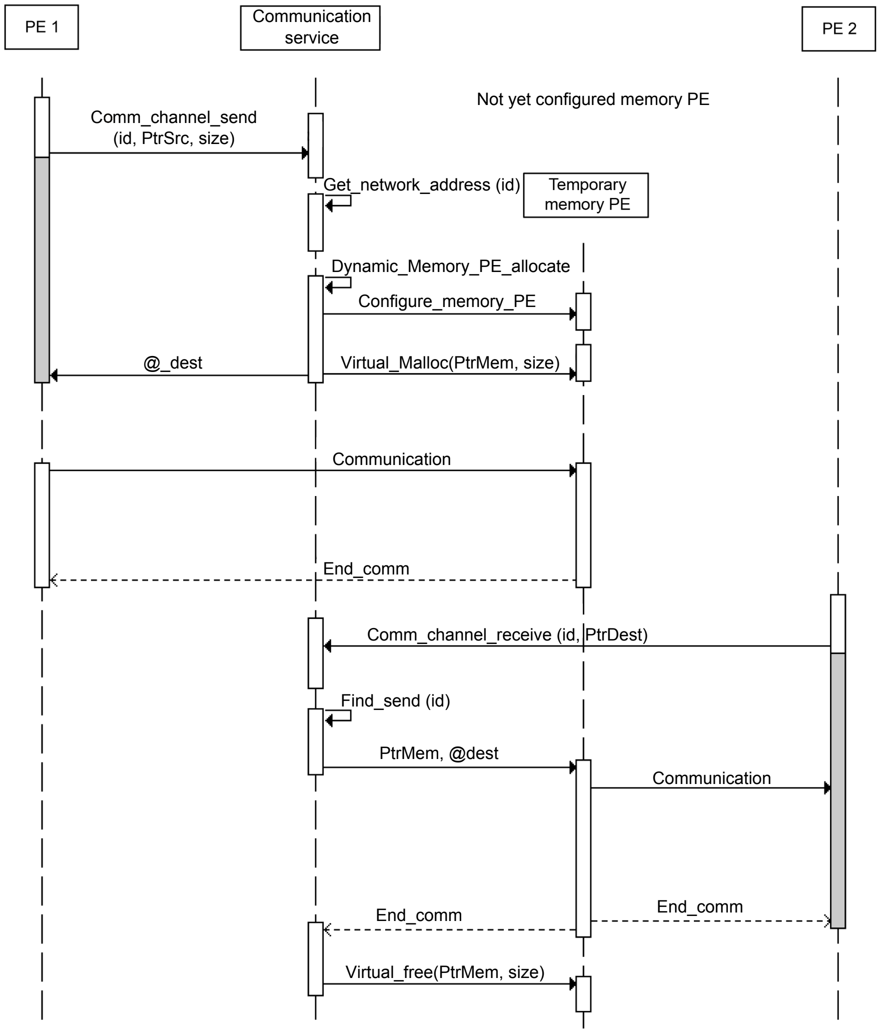

PE 1
Communication service
PE 2
Not yet configured memory PE
Comm_channel_send (id, PtrSrc, size)
Get_network_address (id)
Temporary memory PE
Dynamic_Memory_PE_allocate
Configure_memory_PE
@_dest
Virtual_Malloc(PtrMem, size)
Communication
End_comm
Comm_channel_receive (id, PtrDest)
Find_send (id)
PtrMem, @dest
Communication
End_comm
End_comm
Virtual_free(PtrMem, size)

Figure 2.27 : Sequence diagrams of communications between two PEs with creation of a memory PE temporarily storing the data.

the instantiation of a set of PEs. The synthesis of each PE of a set leads the synthesis tool to include BRAM within the PRR. When all PEs are mapped in the PRR, the memory size included in the PRR is at least equal to the maximum required internal memory from the PE set. In this case, an additional PE memory is defined for each PRR. A memory PE is a PE including logic resources to ensure read and write operations within the BRAM memory included in the PRR. Each set of PEs is then increased with this new module. In this case, the OS can allocate and configure each PRR accordingly with the application needs in terms of communication. The memory of one PRR can be chosen and a memory PE needs to be instantiated dynamically. In this case, the OS finds, between the current available PRRs, which one have sufficient memory space to temporary store the data. After the selection of one PRR, the OS provides the network address of the selected PRR to the interface of source PE. For this PE, it seems that the destination one is ready, and the transfer can start (Figure 2.27). DRAFT drives the data to the selected PRR previously configured as a memory PE.

2.9 Synthesis of the chapter

In this chapter, the DRAFT network is presented. This network is based on the $4 - radixn - stages$ fat-tree topology but the main difference lies in the fact that DRAFT has $n - 1$ stages and half of interconnected PEs directly connected to the top level routers. This simplification of the topology is based on the observation that, in both current and foreseen applications, some of interconnected PEs do not share data between themselves. These PEs can be shared memories, communication interfaces, etc. Connecting these PEs directly to the top level routers allows to significantly reduce the hardware overhead of a fat-tree based topology, and so to improve its scalability. Observing the assumption that no communication can take place between two top level connected PEs, there is in fact no limitation concerning their nature: memories, processors, IP blocks, etc. These PEs can be ever statically or dynamically implemented. DRAFT does not aim to optimize a performance criteria compared with the others, but being a viable trade off between hardware resources consumption, network performances, and flexibility when facing applications using DPR. After defining the DRAFT NoC, the DRAGOON environment is presented. DRAGOON is a framework allowing to generate and simulate both DRAFT and a conventional fat-tree.

Then, the integration of DRAFT inside current applications was studied. For this purpose, a DRAFT-AHB bridge was designed. This bridge makes the relation between PEs connected to DRAFT and microprocessor(s) like the LEON 3 also with shared memories that communicate through an AHB bus. Using both an AHB slave and an AHB master interface, the bridge is compliant with bursts of data with incremental addresses.

Finally, in order to ease the design of applications, an hardware communication service was studied. The communication service that is part of an OS gives to users, and thus designers, an abstraction of the communication architecture and protocol also with the flexibility inherited from the use of the DPR. Based on communication channels, the communication service handles the flexibility of the DPR using memory spaces in the global or local shared memories. If for some reasons sufficient memory spaces can not be found in these memories and a PRR is not used, then a temporary memory PE is instantiated. The memory PEs are in fact the definition of additional PEs that use the BRAM resources included inside the PRR. Thus extra memory spaces can be created on demand.

Chapter 3

Toward new flexible NoCs

This chapter presents new interconnection networks based on circuit switched protocol. The efficiency of these networks is provided by the dynamic reconfiguration of their switches. First presented network is called R2NoC. This network is designed for applications running on dynamically reconfigurable FPGAs. Due to technological constraints, several versions of the OCEAN network were inherited from R2NoC. These networks aim both ASIC and FPGA implementations. OCEAN v3.0 and OCEAN v4.0 are specially designed for FPGA implementations with a synchronous control network reconfiguring an asynchronous data network. The difference between these two versions lies in the routing algorithm that impact the structure of the data switches. The OCEAN v3.1 and OCEAN v4.1 versions are respectively adaptations of the OCEAN v3.0 and OCEAN v4.0 networks in order to fit ASIC implementations. A test platform was also created in order to characterize the four versions of OCEAN and to compare them with a conventional fat-tree simulating an FPGA implementation.

3.1 Objectives and motivations

When considering communication architectures, their hardware cost and required performances steadily scale up along with applications. The main idea of this chapter is thus to define new interconnection networks using the DPR to improve performances, flexibility, and hardware costs. The objective is to design a network on which connected PEs do not suffer any location constraint: one can communicate with another one whatever are their respective locations and independently of current data traffic. A high degree of flexibility would also be highly appreciated in order to improve complex data flow graphs. For example, the ability for a PE to send a packet of data to several destinations at a time (multicast) can dramatically improve performances of parallel applications where a data is shared by multiple PEs. This kind of flexibility is not supported by the DRAFT network. Furthermore, this latter imposes that half of connected PEs (the top level ones) do not communicate together. It should be particularly interesting for a new interconnection network not to present such a limitation. This way this new network should be even more flexible.

Concerning the hardware cost of an interconnection architecture, it is highly linked with the number of PEs that are to be interconnected at a time. Considering that this number is lightened by the DPR, even if interconnection architectures must foresee an increase of the number of communication ports, best effort can be made on low scaled interconnections to minimize their cost. Consequently to previously enumerated drawbacks of DRAFT and of existing reconfigurable Networks-on-Chip (NoCs), the Reconfigurable Routers based Network-on-Chip (R2NoC) network was designed.

3.2 Topology of the R2NoC network

Just like for DRAFT, the way switches/routers are linked together is a major concern in the definition of an interconnection network. It highly influences resulting network performances and both routing and flow control algorithms must be defined accordingly. For the R2NoC network, the main concern is to use the dynamic reconfiguration to reduce the interconnection logic. The first idea could be to use a big crossbar on which every PE is connected and made of only one Partially Reconfigurable Region (PRR). This PRR should contain only direct links reconfigured accordingly with desired communications. However, several concepts must be taken into account:

- First, a PRR is not an empty region were direct links can be implemented. Indeed, every set of four links entering or exiting a PRR have to pass through a Bus Macro (BM) using one full slice of the FPGA. BMs are implemented inside the PRR. Furthermore, a signal entering a PRR through a BM must encounter dynamically implemented logics before exiting through another BM. This is due to Xilinx design software tools that require every signal to be modified or at least transferred between two BMs by some logic. So, the more communication ports there are entering a PRR, and the larger they are in terms of number of wires, the more hardware resources will be involved in every possible configuration of the PRR.

- Second, a PRR is the finest dynamically reconfigurable area. When initiating the reconfiguration of a PRR, the whole resources it contains are reconfigured. However, during the reconfiguration process, if a configuration bit has the same value in the novel configuration as in the previous one, dedicated resource is not affected and no glitch disrupts its behaviour [73]. This is a major issue because assuming that every dynamically implemented logic remains at the same location, existing communications are not impacted by the configuration of a new data path. However, the more logical resources are implemented in a PRR, the more difficult it is to guarantee that they will remain exactly at the same locations for every possible configuration of the PRR.

- Third, the number of possible configurations for a PRR is directly linked with the number of communication ports. If the crossbar contains N communication ports (with both input and output parts), the number of possible configurations is equal to $factorial(N)$ (usually noted $N!$). Every generation of Xilinx

> FPGAs contains hidden resources that are to be revealed only in subsequent generations. So, for industrial protection of their technologies, Xilinx design software tools only generate encrypted bitstreams. Bitstreams, also called configuration files, can not be generated nor modified online. So, in the case of the R2NoC network which should be compliant with present and future FPGAs, every configuration file has to be generated offline and stored in the board.

Considering previous concepts, it appears that if the number of communication ports is high for a single PRR, many logical resources are involved leading to the generation of big configuration files. Furthermore, the number of these files to generate and to store increases dramatically with the number of communication ports. Moreover consequent storage capacities are then required to store these configuration files. Finally, configuration files become more and more difficult to generate guaranteeing the constant placement of every logical resource. For all these reasons, a crossbar like the one presented in Figure 3.1.A is not a viable solution due to the $8! = 40320$ configuration files required for its correct behaviour despite its low number of communication ports. The number of port per switch/router must be minimized.

A mesh topology based network was proposed defining each switch as a PRR [80]. This network is composed of two sub-networks: one dynamically reconfigurable transferring data and one statically implemented for control purpose. In Figure 3.1.B, only the data network is represented. Considering the number of communication ports for each switch, the number of configuration files necessary to perform all possible data transfers between the 8 connected PEs is equal to $5!+4*4!+3*3! = 234$. In this particular case of 8 connected PEs, a 4x2 mesh based network would have reduced the number of needed configuration files to $4*4!+4*3! = 120$. However, a square mesh was chosen to show that the central switch which possesses 5 communication ports heavily impacts the number of configuration files. Unfortunately, such 5 ports switches can not be avoided in larger mesh based networks.

Mesh based networks are often preferred to fat-tree based ones because of the hardware cost of such networks notably when they scale up. However, in the context of dynamically reconfigurable applications, the limited size required from the interconnection architecture can be an advantage for the fat-tree topology. Indeed the fat-tree topology presented in Figure 3.1.C uses unitary switches with only 4 communication ports leading, for the connection of 8 PEs, to the generation and storage of $8*4! = 192$ configuration files. Considering that the fat-tree topology allows high network performances, the reduction of the configuration files is one more reason leading this topology to be used for R2NoC.

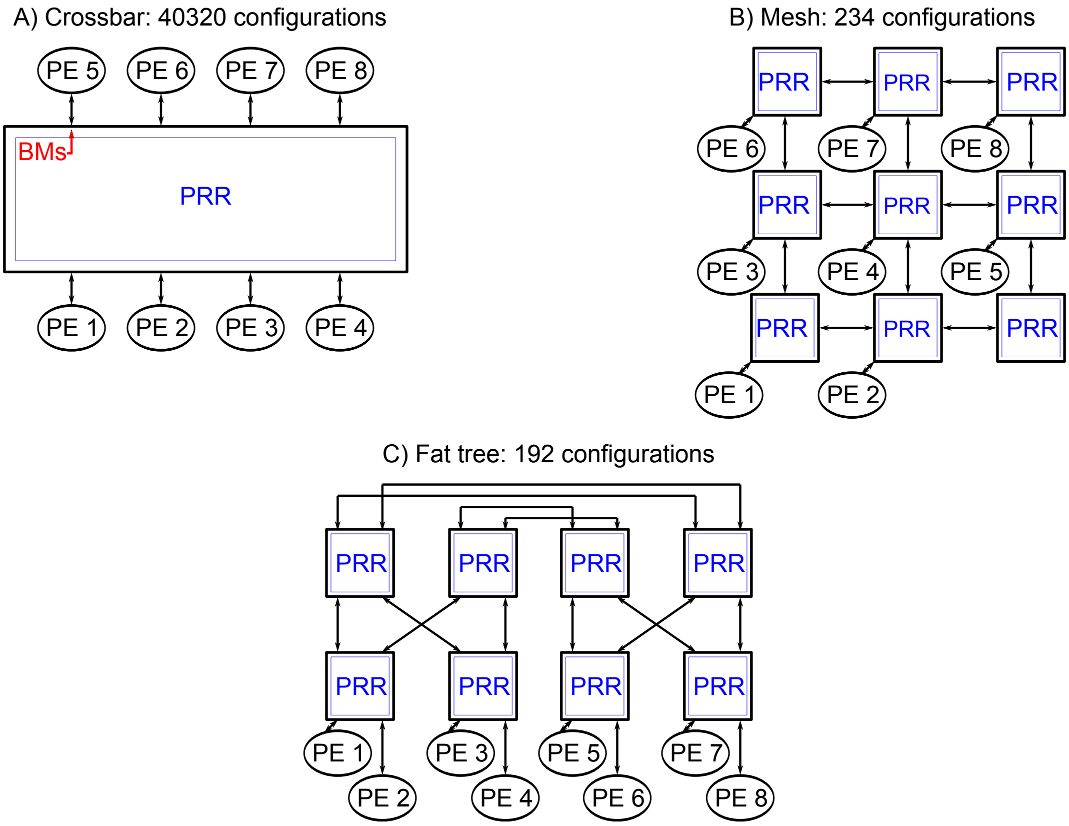


Figure 3.1 : Interconnection architectures based on dynamically reconfigurable switches (PRRs). A) Crossbar network connecting 8 PEs. B) 3x3 square mesh based network with one communication port remaining free. C) fat-tree topology used for R2NoC interconnecting 8 PEs

Remark 1: *If Xilinx provided way to relocate easily a bitstream from a PRR to another one, the number of configuration files to store should be of* $8! = 40320$ *for a crossbar,* $5! + 4! + 3! = 150$ *for a mesh based network and only* $4! = 24$ *for a fat-tree taking advantage of its generic switches.*

Each switch of the R2NoC network is composed of 4 communication ports themselves constituted of an input port and an output one. So, every switch has 4 inputs and 4 outputs which is defined as the radix of the network. The version of R2NoC presented in Figure 3.1.C embeds only two stages of switches. This version is then a $4 - radix2 - stages$ fat-tree. Each port (ever input or output) is made of a data line (32 bits width) and 3 control signals. So, each port has a width of 35 bits.

Switches of the R2NoC network are defined as PRRs in which routing configurations are dynamically configured. The objective is thus to create a direct path between each couple source/destination PEs. So, data are circuit switched through the R2NoC network. Originally, NoCs emerged embedding buffers allowing to store data all along their paths from sources to destinations. However, DPR allows R2NoC

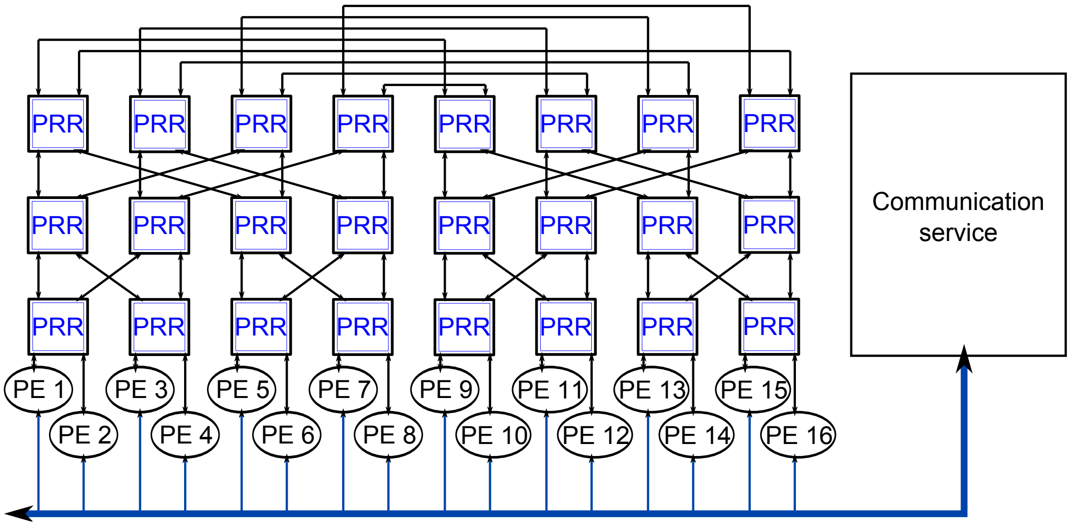


Figure 3.2 : Presentation of R2NoC in its $4 - radix3 - stages$ version (16 connected PEs) with its dedicated communication service.

to circuit switch data so that buffers are no longer necessary. When configured, a path is nothing else than a direct line interconnecting only the desired PEs. R2NoC is designed for a relatively low number of connected PEs. Its reduced size avoids any risk of long lines parasitic effects: fall of maximum frequency or transmission errors. Indeed, the network interfaces (NIs) that are part of every PE connected to R2NoC, embedding buffers and sequential logics, are enough to ensure that long lines are avoided. However, if no buffer nor sequential logics are present along the network, a communication service is required to compute the routing algorithm and to reconfigure the PRRs. Communication service is directly linked with interconnected PEs. Complete R2NoC network with its communication service is presented in Figure 3.2. Further details about the switches and the communication service are given in following sections.

3.3 R2NoC switch architecture

R2NoC switches are dynamically reconfigurable. This assumption greatly impacts the hardware definition of their architecture. Indeed, the DPR allows to create at ease direct routes between one port to another one. No buffer is necessary to store incoming packets. Data should just be transferred directly to their destination port. Each line entering or exiting a PRR is interfaced by a BM. Every BM is able to interface 4 wires using 1 full slice of a Xilinx Virtex V FPGA. A switch interconnects 4 input to 4 output ports with 35 bits width each. Then, $(4+4)*35 = 280$ lines must be interfaced by BMs. So 70 BMs are used per switch consuming 70 slices (thus 35

Configurable Logical Blocks (CLBs), that are defined as set of 2 slices, are used).

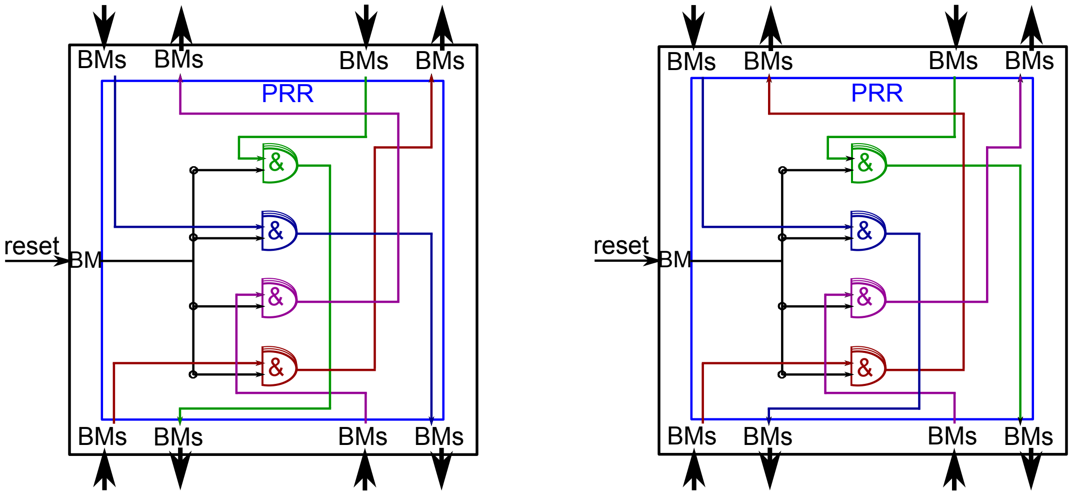


Figure 3.3 : Presentation of 2 possible configurations for a switch over the 24 possible. Each switch is made of a PRR containing BMs as interfaces and dynamically instantiated AND gates. The extra reset line is also pictured.

Direct links can not be instantiated between two BMs: logical elements must be implemented. Taking advantage of these indispensable resources, extra functionalities can be added. In the case of R2NoC switches, AND logical gates are dynamically instantiated between every pair of BMs. Each AND gates receives a unique signal acting as a reset line. If this signal is set to 0, no data can pass through the switch. On the contrary, when it is set to 1, data are transferred without any modification at the cost of the AND gate latency. The reset line should be controlled outside of the switch, so an extra BM is necessary. The architecture of a switch is presented in two different configurations (that can be dynamically exchanged) in Figure 3.3.

3.4 R2NoC Routing and Flow Control

Dynamic reconfiguration of the switches is performed by a centralized communication service linked to the PEs by direct links. The communication service can be implemented as a fully hardware implemented service, or using a processor running a software algorithm. A MicroBlaze processor running a software communication protocol was implemented for practical reasons.

The hardware part of the communication service is composed of a MicroBlaze, the ICAP interface, a configuration DDR memory, and an interface proposing general purpose interfaces (GPIO) that are used to communicate directly with the PEs for control purpose (Figure 3.4). The MicroBlaze computes and establishes the route between source and destination PEs reconfiguring the involved switches through

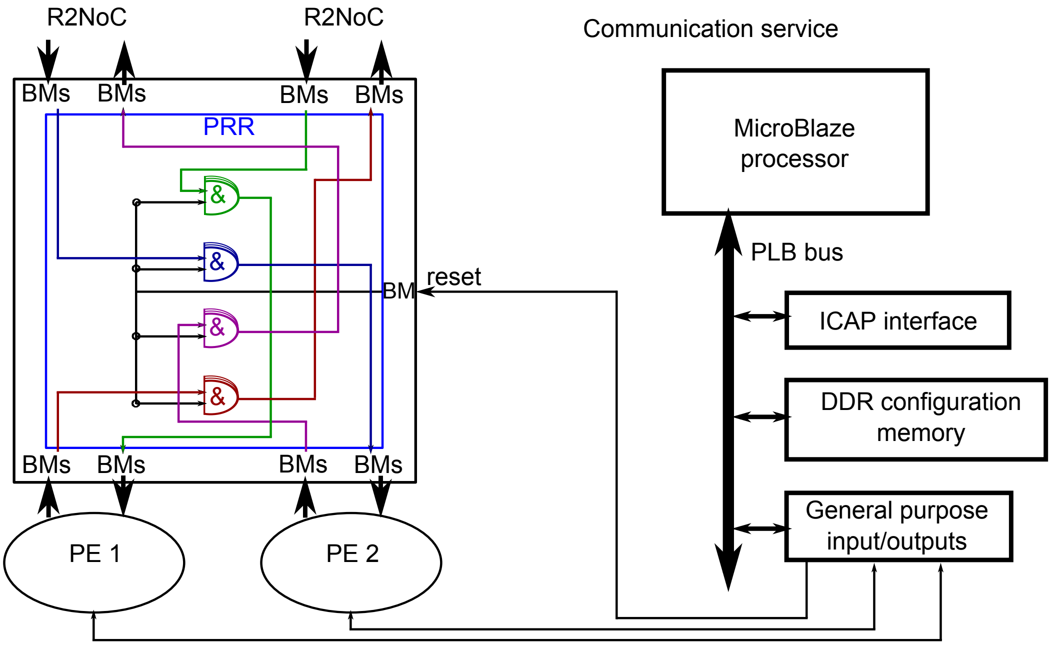


Figure 3.4 : Presentation of the communication service controlling the R2NoC network. For practical reasons, only 2 PEs are depicted here. The communication service is centralized and composed of a MicroBlaze processor that performs the communication management, the routing algorithm, and switches reconfiguration. Configuration files are stored in a DDR memory, and loaded by the MicroBlaze processor directly to the ICAP interface.

the ICAP interface. For this purpose, configuration files are loaded from the DDR configuration memory directly to the ICAP interface.

Communication protocol is proposed as follow. When a PE initiates a communication, it notifies the communication service to open a communication channel in send mode. While the answer to this channel request is not acknowledged, the PE suspends its execution (blocking communications). Since PEs are dynamic, the communication service waits for the destination PE to request a communication channel in receive mode. This point allows to synchronize the PEs and to make sure they are both available for a communication. It also allows PEs not to have the knowledge about their respective locations. This way, no distributed routing tables are necessary because all the knowledge about implemented PEs, including their local addresses, is centralized in the communication service. When both source and destination PEs have requested the same communication channel, before acknowledging, the communication service performs the routing protocol.

The centralized communication service computes the routing protocol considering the topology of the network. Since R2NoC is based on a fat-tree topology, the routing protocol consists in a TurnBack algorithm. Data are routed toward upper

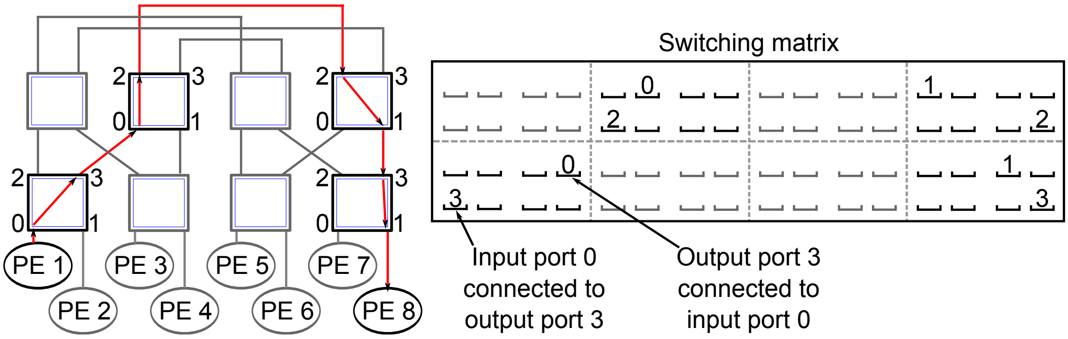

Switching matrix
PE 1
PE 2
PE 3
PE 4
PE 5
PE 6
PE 7
PE 8
Input port 0 connected to output port 3
Output port 3 connected to input port 0

Figure 3.5 : Establishment of a one way communication route in the R2NoC network and the corresponding switching matrix.

level switches until the destination is reachable only going down from this level. The routing protocol is realized thanks to a matrix filed with the state of each communication ports of each switch (parts of the switching matrix in Figure 3.5). For the $4-radix2-stages$ R2NoC network presented in Figure 3.1.C, the switching matrix consists in a 4x16 array of values. These values are coded using two bits indicating the port number they are connected with. For example, a connection of the PE1 and PE8 hardware elements (that can themselves be statically or dynamically allocated) is presented in Figure 3.5.

When establishing a communication route, the algorithm uses the switching matrix in order to recursively find an available way compliant with the TurnBack algorithm. This guarantees that communication routes are always minimized in length. For this purpose, a local array of 2x4 values is created corresponding to every communication port states of the switch the sending PE is connected with. Then, the possibility to reach the destination PE directly from the ports of the switch is checked. If not, using the TurnBack algorithm, the communication service uses the recursivity to determine if a communication way can be created through available neighbour switches. When an available path is found, the communication route is created updating the switching matrix, and corresponding partial bitstreams are loaded from the memory to the ICAP interface. The fat-tree based topology of R2NoC guarantees that there is always a communication path available between two PEs requesting to communicate: no livelock and no deadlock can occur. Furthermore, taken communication path is ensured to be minimal. Thus, the communication service does not need any priority mechanism to manage communication failures.

3.5 Constraints of the R2NoC network

From the characterization of the R2NoC network concerning the network performances and the hardware cost, several constraints appeared.

- First is the number of configuration files to generate and to store for correct operation of the network. Every switch requires $4! = 24$ configuration files. Thus the scalability of R2NoC can be dramatically affected by the storage capacity: a $4-radix2-stages$ (connecting 8 PEs) requires 192 configuration files, a $4-radix3-stages$ (connecting 16 PEs) necessitates 576 configuration files, and a $4-radix4-stages$ (connecting only 32 PEs) implies the generation of 1536 bitstreams. Relocation of bitstreams should highly reduce the configuration files to only 24 whatever is the size of the network, but it is not currently supported by available FPGAs.

- Second is the waste of hardware resources just to implement BMs: a $4-radix2-stages$ uses 280 CLBs for the bus macros while a $4-radix3-stages$ consumes 840 CLBs, and a $4-radix4-stages$ 2240 CLBs which represents merely half of the CLB provided in middle sized Virtex V FPGAs.

- Third is the constant placement of BMs and logical gates inside every PRR. Until ISE 11, BMs had to be located manually using the PlanAhead 10.1 software. This is a very long and boring work for designers. From ISE 11, BMs are placed automatically. This was a very welcomed improvement of design tools but the constant location of the logical gates remains an important drawback for industrial applications.

- Fourth is the use of a centralized communication service that scales badly with the network inducing latency to perform the routing algorithm thus limiting the parallelism of the communications. However a centralized communication service was implemented because only two ICAP interfaces are embedded in most recent FPGAs. The restricted number of ICAP interfaces is indeed the major bottleneck for the R2NoC network since the various dynamic reconfigurations needed when creating a data path have to be made sequentially.

- Fifth is the reconfiguration latencies. Indeed, we measured a reconfiguration delay of 11.3ms for each 22KB configuration files. Some works like [72], or [35] obtained much better reconfiguration times modifying the hardware ICAP. Accordingly with their results, reconfiguration times from the order of 50-60μs can be reached. However, a reconfiguration time of 50μs represents 6250 clock cycles with a frequency of 125MHz. This latency per switch is too much important in order for R2NoC to have an interest in a real life application. If considering R2NoC and a equivalent conventional fat-tree interconnecting 8 PEs, the average time of flight in the fat-tree can be estimated to be close from 50 clock cycles. However, if 4 switches of R2NoC must be reconfigured, the reconfiguration delay is at least of $4 * 6250 = 25000$ clock cycles with an optimized ICAP. So, even if when configured the time of flight is inferior to one clock cycle, R2NoC is only interesting if more than $\frac{25000}{50} = 500$ successive communications occur using the same data path. Similarly, with genuine Xilinx ICAP, more than 113000 successive communications are required for R2NoC to be attractive.

Further implementation results are provided in Appendix C, but it is important to notice that all these constraints are caused by current technologies. However, unless technologies evolve in the future, R2NoC (like other networks using Xilinx dynamic reconfiguration to perform circuit switched data transfers) will remain impractical for most of industrial applications.

3.6 Motivations for the OCEAN network

Currently, most of present digital systems are limited by their interconnection [33]. This observation is notably due to the constant increase of both density and performances of the computational logics embedded in Systems-on-Chip (SoCs). Consequently, interconnection architecture are more and more solicited for data transfer purpose. These architectures become a bottleneck when the amount of offered data outperforms their capacity of communication. Foreseeing more and more complex applications with larger numbers of involved PEs and larger communication needs leads to search for new interconnection networks.

A NoC adapted to current and future applications should present following characteristics:

- a reduced consumption of hardware resources,
- high level network performances in the sens of extremely low latencies along with high bandwidth,
- a high degree of flexibility in order to support a large range of application,
- and finally no induced configuration memory.

The main idea of the R2NoC network was to use the DPR to create data paths dedicated to each communication. This idea had the advantage of significantly reducing the number of hardware resources required by the switches. Furthermore, the fact that data are circuit switched in a fat-tree based NoC was expected to induce high level network performances. Indeed, the offered bandwidth remains constant between each hierarchical level of the NoC all along from the base to the top. Consequently, if each PE (located at the base level) has a dedicated connection link, at least one data path is guaranteed to exist between two PEs requesting to communicate. Thus, dynamically reconfigurable switches combined with the advantages inherent to the fat-tree topology seem very interesting to fit the first three requirements.

However, R2NoC suffers many constraints that are caused by currently available industrial solutions. Consequently, the main idea for a new interconnection should be to reproduce the behaviour of a network with dynamically reconfigurable switches, but independently from industrial devices. Indeed, an efficient NoC should be able to dynamically reconfigure its switches without using the Xilinx DPR. Furthermore, it should do so with reduced hardware and temporal overheads. The independence from any industrial solution leads to another motivation: the benefits of a DPR like behaviour should not only be reserved to systems implemented in FPGA. It would be very interesting for a new NoC to be compliant with both FPGAs and ASICs implementations.

The requirement of flexibility does not imply only the compliance with dynamically and partially reconfigurable PEs. Indeed it also implies for the NoC to support several modes of data transfers. For example, the communications between an IP block and a shared memory would be significantly improved if the interconnection network supports the use of a DMA. DMAs are commonly encountered in applications receiving a constant flow of data that must be stored in memory. DMAs are characterized by very long transmission times so that the bandwidth offered by the network becomes more important than data time of flight. The opposite example should be the possibility for a microprocessor to send control informations to a PE with a minimal latency. In latter example, only a few data are transmitted so that a minimized time of flight is much more important than a large bandwidth. There is another communication scenario that should be supported by a flexible NoC. Due to the increasing parallelism of the applications, it becomes more and more necessary for some PEs to send the same data to several others. It should thus be very interesting for the interconnection network to be able to transfer data to multiple destinations simultaneously (multicast). In addition to being more flexible, a network supporting multicast would provide significantly better network performances.

All these motivations for a new NoC led to the On-Chip Efficiently Adaptive Network (OCEAN) network. In remaining of this chapter, the OCEAN network is presented in detail along with the answers it brings to current and foreseen interconnection challenges.

3.7 Topology of the OCEAN networks

The fat-tree topology presents numerous advantages concerning offered flexibility and performances. For R2NoC, the choice of a fat-tree topology was made considering the constraints imposed by Xilinx partial DPR and notably the number of needed configuration files. The OCEAN network aims to be independent from industrial solutions concerning DPR exactly to avoid these reconfiguration files. So, the main reason for OCEAN to use the fat-tree topology lies in the large and constant bandwidth it implies. This way, direct data paths can be created to make PEs communicate without any risk of livelock, deadlock, or even of non-minimal path. Furthermore, the fat-tree topology ensures that there is always an available data path between PEs requesting to communicate. Consequently, thanks to this topology, the high performances expected from circuit switched data transfers can become a reality.

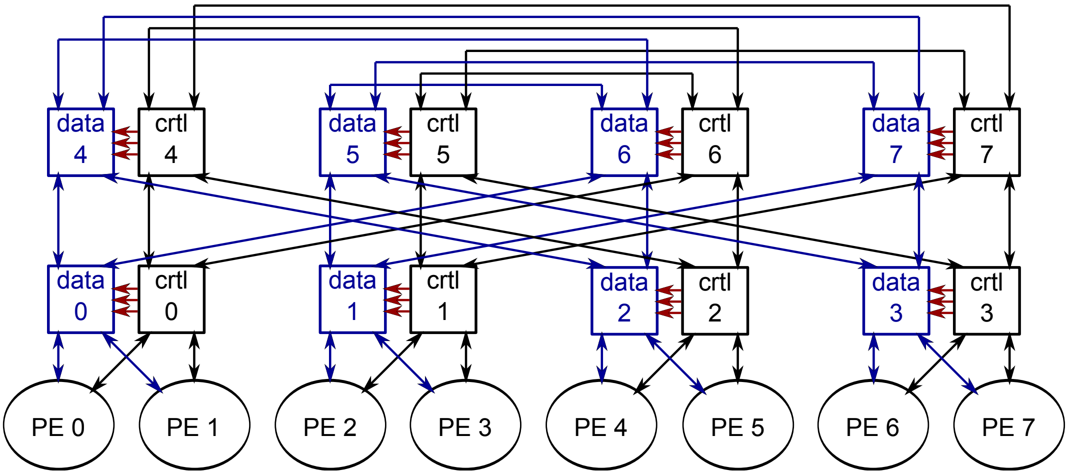


Figure 3.6 : Presentation of the OCEAN network with its two sub-networks. The data network depicted in blue is dynamically reconfigured to circuit switch data. The control network (in black) distributively manages the dynamic reconfiguration of the data network through reconfiguration interfaces (in red).

Hence, just like R2NoC, OCEAN is composed of a $4 - radixn - stages$ network whose switches are dynamically reconfigured. The first main idea of the OCEAN network lies in the management of this dynamic reconfiguration of the switches. Indeed, a centralized communication service using a unique reconfiguration interface (ICAP) rapidly appeared to be a limiting factor of scalability. This is why, in OCEAN, the control and thus the dynamic reconfiguration of each switch is realized by dedicated controllers. Consequently, the communication service is no longer centralized but distributed all along the data network. This is why OCEAN is in fact composed of two sub-networks. One is dedicated to the transportation of data through dynamically reconfigured dedicated routes. This is the data network. The second one is dedicated to the control and the management of the data network. This is the control network. In order for each switch of the control network to manage one switch of the data network, both sub-networks have the same $4 - radixn - stages$ topology with n the number of hierarchical levels. Furthermore, each switch of the control network masters the dynamic reconfiguration of its associated switch from the data network through a specific reconfiguration interface. The structure of the OCEAN network with its two sub-networks linked through reconfiguration interfaces is presented in Figure 3.6.

3.8 The data network

3.8.1 Communication principles and interfaces

The main particularity of the data network is to transmit data through dynamically configured data paths. Each switch of the data network can be dynamically reconfigured independently from the others. Each switch presents four data interfaces (communication ports), and one reconfiguration port. Depending on informations arriving on the reconfiguration port, data paths are dynamically created between the communication ports. This way, data incoming from a port are forwarded to one or several other ports through a dedicated data path. Reconfiguring every switch from source to destination PEs creates a direct route between these elements.

Interfaces of the data network are constituted of two ports each. One port is for data emission (generically noted ”tx”), and one port for data reception (generically noted ”rx”). The set of these two ports allows full-duplex communications (simultaneous communications in both ways: emission and reception). Both rx and tx ports have the same architecture: a 32 bits width data line plus 2 control signals each. These 2 control signals are respectively to notify the receiver PE(s) that a data is available on dedicated line, and to notify the sender that a new data can be received. Data interfaces for both emission and reception of data from the OCEAN network point of view are presented in Table 3.1.

3.8.2 Data switches

The objective for data switches was to define an architecture dynamically reconfigurable but without using any component linked to present industrial solutions. Wished behaviour for these switches is only to directly connect its 4 reception to the 4 emission ports depending on required configuration. Since Xilinx DPR of direct wires revealed itself to induce numerous constraints, a solution producing the same behaviour but without any constraints was looked for. This solution, the second

Table 3.1 : OCEAN API provided to each PE concerning the emission/reception of data. Directions are defined considering the network, so an input for the network should be connected to an output for the connected PE.

Region	Signal name	Size (bit)	Direction	Function
Data reception	rx_data	32	In	data to be transmitted are injected here
	rx	1	In	notifies that valid data are present on rx_data
	rx_credit	1	In	receiver indicates if further data can be received
Data emission	tx_data	32	Out	transmitted data arrive here
	tx	1	Out	notifies that valid data are present on tx_data
	tx_credit	1	Out	indicates to sender if further data can be sent

main idea of the OCEAN network, is inherited from FPGA reconfiguration. Indeed, inside the reconfigurable fabric of an FPGA, there are small resources called switch-matrices allowing to connect a CLB to a metal made wire linking desired region of the FPGA: for example another CLB. Switch-matrices are configured depending on witch metal wire should be used, and so which CLB should be reached. Switch-matrices are not directly user programmable, their behaviour is programmed by design tools during the place and route phase of design compilation. Then their configuration is included in partial bitstreams with user programmable components i.e. CLBs [118].

Concerning OCEAN data switches, the principle remains the same. Each data switch is defined as a switch-matrix allowing to select a data path accordingly with received dynamic configuration. Configurations are online generated by the switches from the OCEAN control network and transmitted on the fly to data switches through the reconfiguration port. As presented in Figure 3.7, in a data switch, only logical gates AND and OR are necessary to route a reception port toward correct emission port(s). For example, we consider a data switch located at the base stage of the data network. A data coming from a PE must have the possibility to be routed toward one or several of the three other communication ports. For this purpose, three AND gates are connected to each wire of the reception port the data

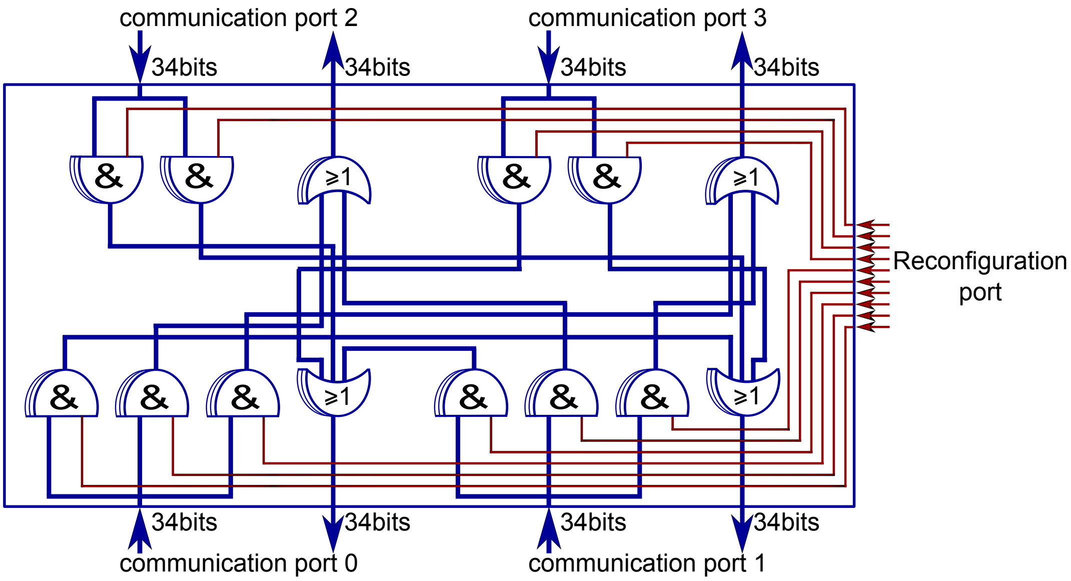


Figure 3.7 : Data switch from the OCEAN network. Appropriate emission ports are dynamically connected to the reception ones through logical gates OR and AND. Latter ones assuming the selection of the route(s) accordingly with configurations issued from the reconfiguration port.

comes from. Each AND gate receives as second input a wire coming from the reconfiguration port. Every AND gate addresses a specific emission port. So, if the three wires coming from the reconfiguration port are set to ”1”, the incoming data is routed toward the emission part of the three other communication ports. Similarly, if only one AND gate receives a ”1” from the reconfiguration port (”0” for the two others), data are only allowed to cross this gate, and then to reach the dedicated communication port. Every wire coming from a reception port should be routed toward the same destination(s). So, only a unique wire from the reconfiguration port is required to route the 34 wires of reception port toward dedicated emission port. Finally, since emission port can receive data from several reception ports, OR gates are implemented on each wire. This way, assuming that wires coming from the reconfiguration port do not allow several reception ports to access simultaneously the same emission port, data are transmitted without any change.

In a fat-tree topology, a data climbing down the structure of the tree before going up in a switch does not make sense. It would result in non-minimal data paths and a significant waste of bandwidth. This is why, in Figure 3.7, only two emission ports are accessible from communication ports 2 and 3. This saves both logical gates and wires from the reconfiguration port. So, the reconfiguration port is defined with a width of 10 bits.

Remark 2: *Hardware resources are statically implemented. However, in Xilinx like architectures, configurable elements are physically engraved inside the chip and* *only the resulting behaviour is dynamic accordingly with configuration bits. This is exactly the same principle in OCEAN data switches.*

3.8.3 Discussions over the data network

Considering the structure of data switches, a data reaching its destination PE only encounters combinatorial logics along the data path. No buffer nor sequential logics are used to synchronize the data. Thus data are transferred asynchronously through the network. This is a major specificity from OCEAN network. Even if data are injected in and read from the network using a clock reference, no further synchronization is made during the transmission. This allows to reduce the time of flight to logic and wires propagation delays. For applications using DMA or transmitting large volumes of data, the time of flight is not so important. However, for control informations requiring extremely low latencies, time of flight minimized to propagation delays is very interesting. Furthermore, since every bit of data crosses the same logic gates and follows the same data path (same wire length), very short transitory states are expected at data arrival time.

Another consequence from data paths made only of combinatorial logics lies in the reconfiguration time. Indeed, the drawbacks from Xilinx DPR are completely avoided in OCEAN since the reconfiguration latency is equal to the respond time of the AND gates.

Furthermore, the data network has the particularity that no clock signals are transmitted as references along with data. This is why data interfaces do not include any clock signal. This particularity lies on the observation that it is more cost efficient to synchronize the Network Interfaces (NIs) that send/receive data on/from the network, than to propagate clock references. Indeed, even if OCEAN data network propagated clock references, different communication frequencies from the PEs would require asynchronous buffers to be implemented inside their NIs. So, unifying the communication frequency, which can be different from PEs' does not imply an important hardware overhead. Furthermore, the absence of propagated clock references allows to save hardware resources all over the network. Considering the steadily increase of the chip size, it is more and more difficult to provide exactly the same clock to all embedded resources. This is why a convention is asserted: data are both transmitted and read on the rising edge of the communication clock (in NIs) so that the impact of propagation delays and of clock jitter are less sensitive. Indeed, the maximal communication frequencies (that depend of the size of the OCEAN network) presented in Chapter 4 guarantee that data are stable and then can be read at each falling edge assuming ideal clocks. Since in real life clocks are not ideal, reading data on rising edges increases the jitter tolerance. As demonstrated in Subsection

4.4.2, this convention is sufficient to ensure the coherency of transmitted data with 0% error.

Consequently to these observations, from the data transmission point of view, OCEAN answers well to the initial motivations:

- low hardware cost because only combinatorial logics are implemented (no buffers, no sequential logic, size of the switches limited to 4 communication ports),
- high network performances due to both the fat-tree topology (ensuring the availability of a least one data path) and the time of flight reduced to propagation delays,
- independence from industrial solutions concerning DPR (extremely low reconfiguration times),
- high flexibility due to both the absence of control logics along the data path (no specific communication protocol is imposed) and the times of flight allowing transfers of critical control informations,
- and finally the support of the multicast (several destinations can be accessed at a time depending on informations issued from the reconfiguration ports).

3.9 The control network

3.9.1 Operation principles and interfaces

The control network performs the dynamic reconfiguration of the data network. For this purpose, every control switch disposes of a reconfiguration interface allowing to dynamically load desired configurations in associated data switches. To prevent inefficiencies, dynamic reconfiguration of a data switch is initiated by a control switch only when an end-to-end route has been found and acknowledged. The control network operates as follow. Before starting the transmission of data, a PE requests the control network to create a data path toward destination PE(s). First switch receiving the request along with destination address(es) computes the routing algorithm and forward the request to appropriate neighbour switch(es). The propagation of the request continues until it reaches destination PE(s), thus creating a control route. Destination PE acknowledges the request when it is ready to receive data. Receiving the acknowledgment, every switch reconfigures its associated data switch and propagates it along the control route. This way a data path is created and kept valid until the source PE cancels the request returning in to ”0”.

In addition to the reconfiguration port (10 bits width), each control switch has 4 control ports. Each control port is composed of two parts: the reception part (rx) to receive requests and destination address(es), and the emission part (tx) to transmit these control signals. The reception part also embeds an acknowledgment signal which is to be transmitted to the source PE. Similarly, the emission part is able to receive the acknowledgment from destination PE(s). One of the motivations for the OCEAN network is to support multicast. When a PE wants to send data to multiple PEs at a time, it should not transmit their addresses sequentially to the control switches. This is why the address signals contain as much wire as there are interconnected PEs. Every wire designates a particular PE. So, if several wires of the address signal are set to 1, every corresponding PE will receive the data (multicast). On the contrary, if only one wire is set to one, corresponding PE will be the only one to receive the data (unicast). The multicast functionality from OCEAN networks is very practical for parallel applications, but it has an hardware cost: the number of wires for the address signals scales exactly the same way as the network itself. For example, the $4 - radix2 - stages$ version of OCEAN uses 8 bits width address signals while the $4 - radix5 - stages$ version requires 64 bits width address signals. Interfaces provided to PEs by the control network are presented in Table 3.2

Table 3.2 : OCEAN control interface provided to each PE. Directions are provided considering the network, PEs should then oppose their outputs to the inputs of OCEAN. The size of the addresses are given for the $4 - radixn - stages$ versions of OCEAN interconnecting "N" PEs.

Region	Signal name	Size (bit)	Direction	Function
Control reception	rx_task_id	N	In	destination address(es)
	rx_req	1	In	requests a data path toward destination address(es)
	tx_enable	1	Out	acknowledgment when data path is configured
Control emission	tx_task_id	N	Out	destination's own address + co-receivers' if any
	tx_req	1	Out	a data path toward receiver is requested
	rx_enable	1	In	acknowledgment, receiver is ready to receive data

OCEAN control network operates synchronously using on a clock signal which is distributed to every switch. Furthermore, a reset signal is also required to initialize all transmitted signals along with embedded control logics.

3.9.2 OCEAN Routing algorithm

Since data should cross a network to reach their destinations, a routing algorithm is needed. An adaptive routing algorithm is particularly interesting because it can significantly improve the QoS. Indeed, thanks to an adaptive routing the more solicited areas of a network can be avoided, thus contributing to make the traffic repartition more uniform. Furthermore, if for some reasons a switch or a link of the network becomes faulty (no transmission occurs through this switch/link), an adaptive routing will automatically search for another route. However, the difficulty in packet switching lies in the fact that the first data packets (containing control information (header, etc.) and the beginning of the payload) are already inside the network. So most common solution consists in trying to foresee through additional signals the condition of neighbour switches in order to detect which one of them is less stressed by the data traffic. In the case of OCEAN, an adaptive routing algorithm can be much more simple and cost effective to implement. Indeed, all its functioning is based on requests and acknowledgments (handshake). So, a request can be forwarded to multiple neighbour switches. The route without any faulty switch will be acknowledged as soon as the request reaches destination PEs. When acknowledged, every other path that was tested sees its request signal returning to ”0”, thus making previously solicited emission parts of the control ports available for other communications. This selection of multiple control routes, as well as the configuration of the data switches and the cancelling of the unused control route are presented in Figure 3.8.

The particularity concerning the routing of requests in the OCEAN control network lies in the fact that the routing algorithm is distributed in every switch, but moreover in every control port of every switch. In a switch, the bottom level control ports are noted 0 (bottom left) and 1 (bottom right) while the top level ones are noted 2 (top left) and 3 (top right).

This kind of adaptive routing strategy is interesting to be implemented in OCEAN in order to improve its reliability making it more fault tolerant. The routing algorithms implemented in reception parts of the control ports 0 and 2 are presented in Figure 3.9. In the case of the control port 0, the routing algorithm can be explained as follow:

- First, when receiving a request signal, the destination address is read. Every control port has two specific masks (specifically defined for each switch) indicating which PEs can be reached from top level control ports (2 and 3) and

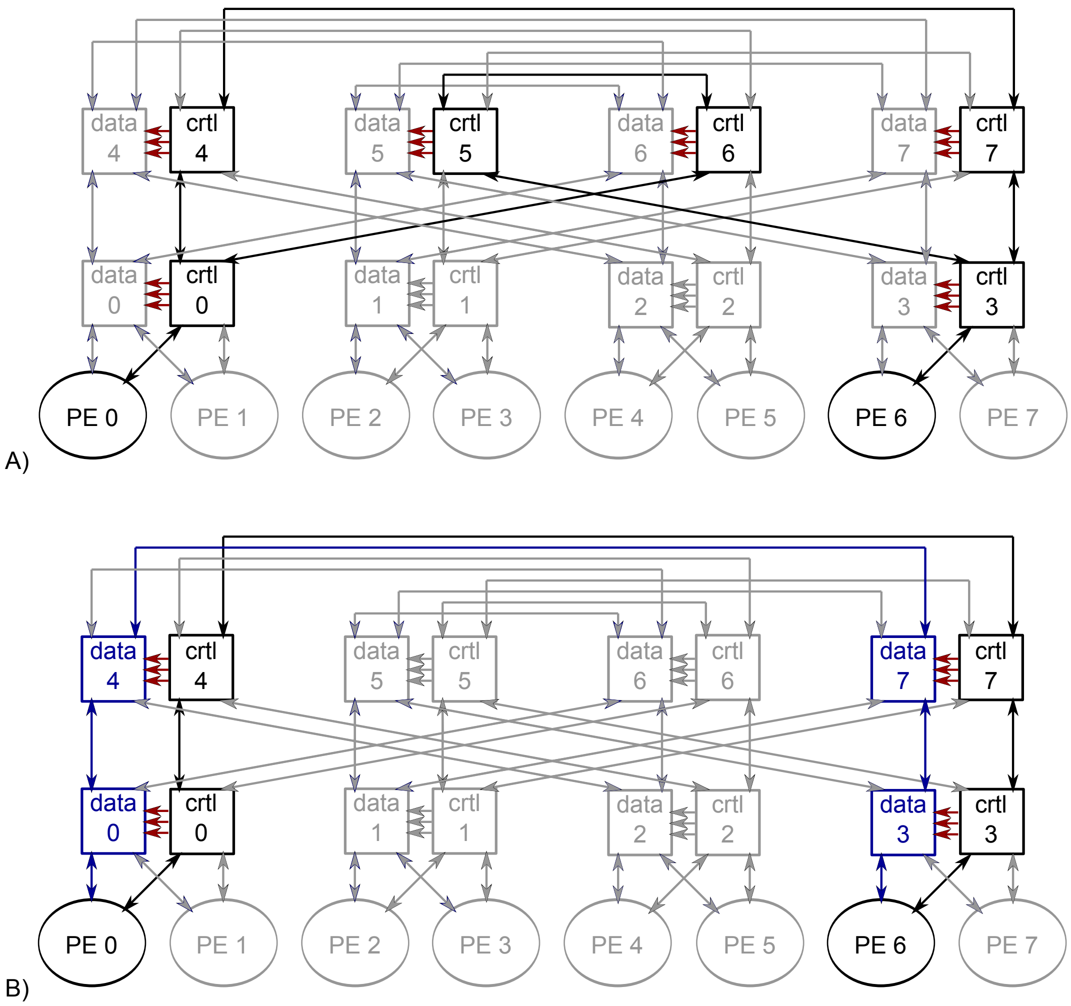


Figure 3.8 : Adaptive selection of control routes in the OCEAN network. In A) multiple routes are tested by control switch 0. So, if for some reasons one switch (4, 5, 6, or 7) becomes faulty, PE 0 is guaranteed to find another control route to reach PE 6. In B), when PE 6 acknowledges a communication request, the route that first transmitted the request configures corresponding data switches, while the remaining request from switch 0 is cancelled.

bottom level ones (0 and 1). Simply comparing incoming address with these two masks allows to know if a route should be requested toward top level neighbour switches, bottom level ones, or both in case of multicast. As it was assumed by both data and control switches, in order to avoid any problem of cyclic testing a request coming from top level neighbours can only be routed toward bottom level ones.

- If the comparison indicates that the request should be forwarded toward a top level switch, in fact it is always forwarded to both control ports 2 and 3, themselves transmitting the request to neighbour switches. Since there are

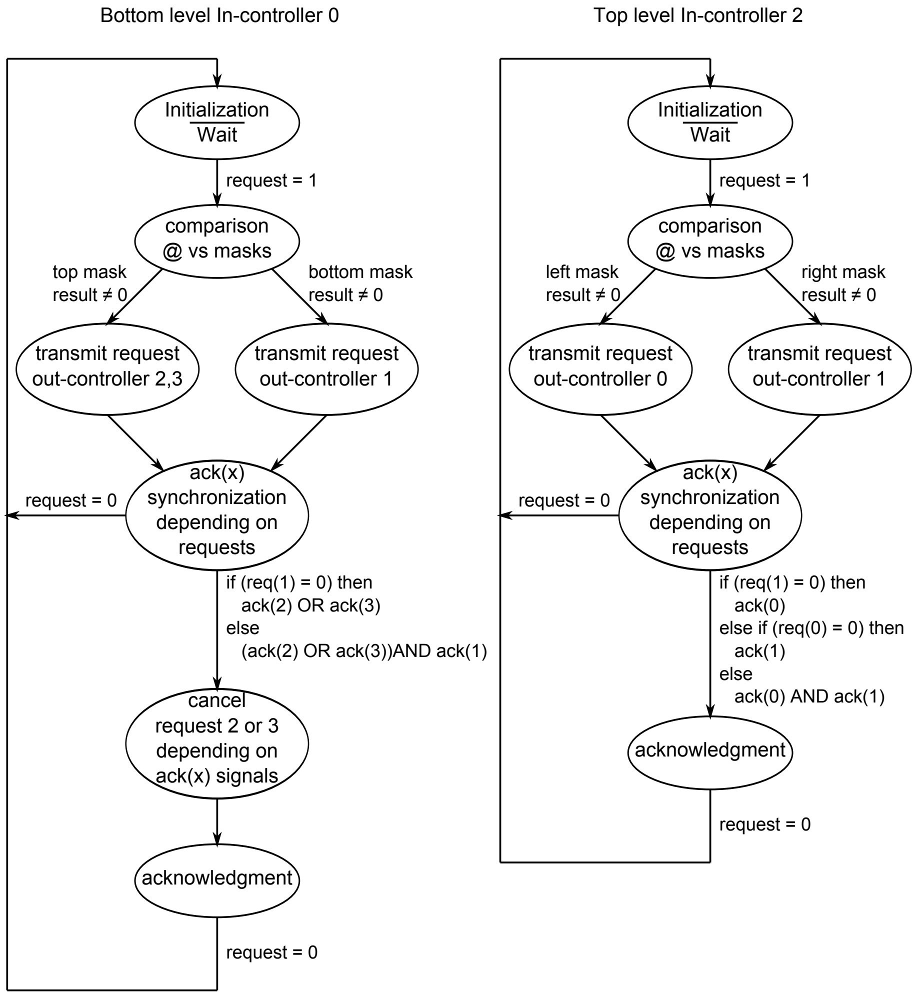


Figure 3.9 : Adaptive routing algorithms processed in control ports 0 (bottom level In-controller) and 2 (top level In-controller) of the OCEAN control switches.

2 top level control ports, and since they can only be accessed from the two bottom level ones, at least one of the control ports 0 or 1 is sure to see its request transmitted immediately. Thanks to the fat-tree topology, the same PEs are always accessible through the two top level control ports. So, as soon as control port 2 or 3 receives an acknowledgment, it informs requesting control port (0 or 1). This latter will immediately cancel its un-acknowledged request. For example, if control port 2 receives the acknowledgment, requesting control port 0 cancels the request it transmitted to control port 3. In this example control port 3 is now available to transmit a request from control port 1.

- If the comparison indicates that request should be forwarded toward bottom level neighbour switch(es) or PE(s), no multiple paths can be tested. Indeed, each bottom level control port permits to access different PEs. So, if requests are sent to the two bottom level control ports, then both acknowledgments are required to acknowledge the control route. When two requests reach a unique bottom level control port, the first arrived takes the priority.

Considering this adaptive routing algorithm, combined with the fat-tree topology of the control network, an acknowledgment from a PE is ensured to be always transmitted through a unique and minimal control route which is reserved for the communication. Since the data network has the same topology, the configured data path is also ensured to be unique and minimal. Furthermore, this algorithm ensures that no deadlock nor livelock could occur whatever the data traffic is.

3.9.3 Control switches

Control switches are both the heart and brain of the OCEAN network. Indeed they propagate incoming requests and compute the routing algorithm. Control switches have two main particularities making OCEAN an original and innovative network. Usually in NoCs, each switch has a unique control block performing the routing algorithm and configuring accordingly a unique crossbar. In OCEAN, each part (emission and reception) of each control port has a dedicated controller. So, any critical section is avoided. The second particularity lies in the fact that control switches do not possess any buffer. All the operation of a control switch is based on handshake. The architecture of a control switch with its embedded controllers is presented in Figure 3.10.

The 8 controllers work in parallel, only sharing signals based on the handshake functioning. Controllers have different functions depending on if they are affected to an emission or a reception part of a communication port. So, they are respectively referred as In-controllers and Out-controllers.

In-controllers perform the routing algorithm accordingly with incoming address and request signals. After computing the routing algorithm, they first transmit the request to appropriate Out-controllers. As explained in previous section, the routing algorithm of In-controllers 0 and 1 has the possibility to test several routes to reach a destination. This is only possible when the request climbs the structure of the control network. Thus, several requests can be generated. As soon as acknowledgments are received from Out-controllers (coming from all aimed PEs), remaining requests (if any) are cancelled. Furthermore, at this point, dedicated lines of the reconfiguration port are configured and an acknowledgment is sent answering the inputted request signal. This propagates the dynamic reconfiguration of the data path. When acknowledgments reach the source PE, the overall data path is configured and the communication can take place in the data network. As soon as the request signal coming from source PE returns to ”0”, the lines of the reconfiguration port are

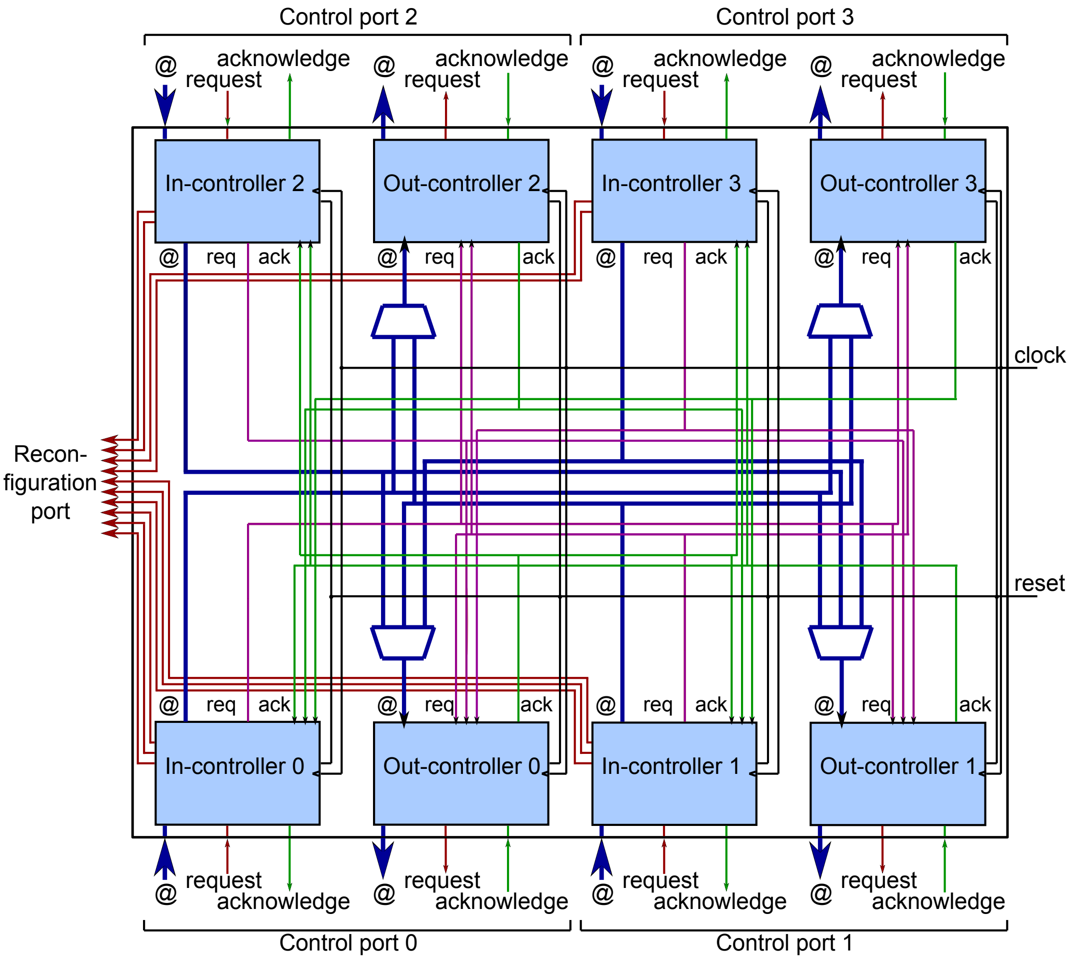


Figure 3.10 : Control switch from the OCEAN network. This switch is composed of 8 controllers each dedicated to a part (emission or reception) of a control port. Controllers work in parallel.

re-initialized to ”0” (cutting off the data path), transmitted requests are cancelled, and the In-controller waits for a new request to arrive. The FSM performing this functioning of the In-controllers 0 and 1 is presented in Figure 3.11.

When a request climbs down the fat-tree based structure of the control network, assuming it will not go up again, only one route can be taken to reach destination PEs. Consequently, all requested routes must be acknowledged before the acknowledgment to be sent to previous control switches. This is why In-controllers 2 and 3 presented in Figure 3.12 do not possess the ”cancel not-acknowledged requests” step in their FSMs.

Out-controllers can receive requests from all In-controllers they are connected with. For example, Out-controller 0 receives requests and addresses from In-controllers 1,2,3 while Out-controller 2 receives these signals only from In-controllers 0 and 1 (U-turn not allowed when requests climb down). The rule of an out-controller is to

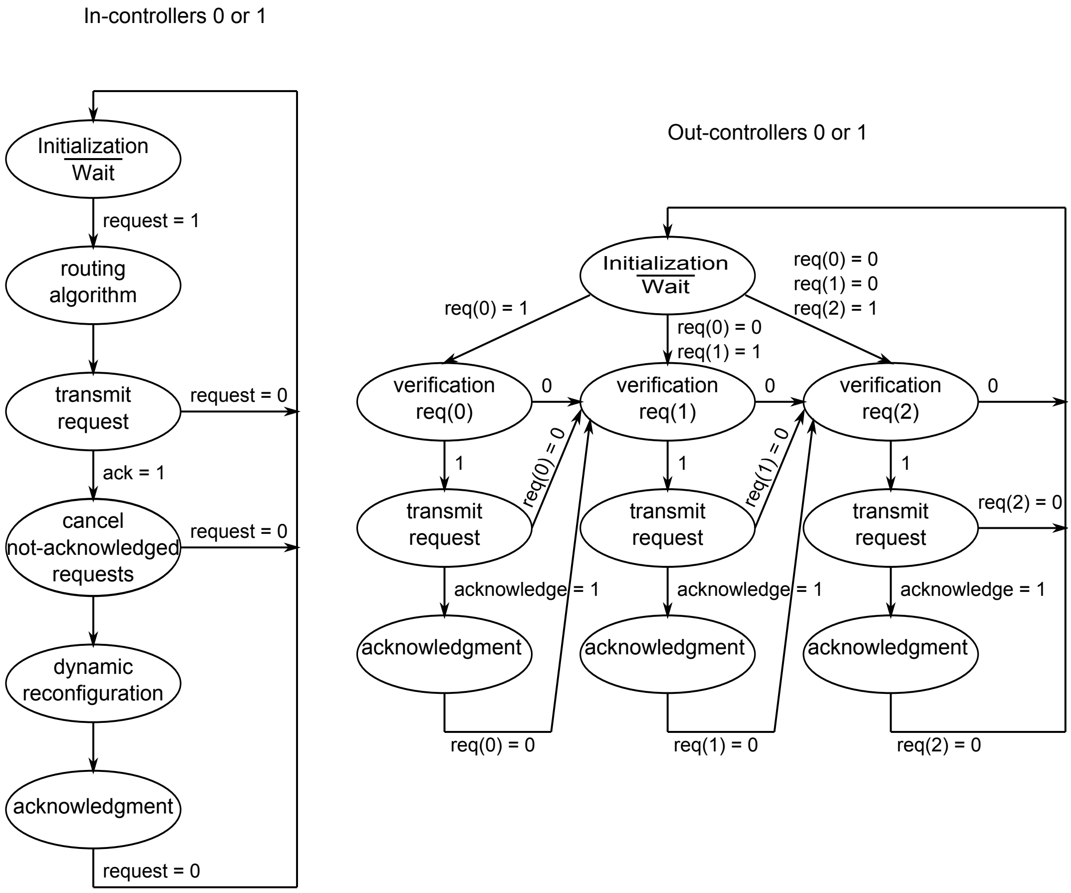


Figure 3.11 : Presentation of the FSMs defining the functioning of In-controllers and Out-controllers 0 or 1 from OCEAN control network.

forward received requests to neighbour switch or PE. Since all requests can not be forwarded at the same time, the FSM of Out-controllers cyclically and sequentially considers the requests, thus giving a fair priority to every In-controller. If at least one request signal is set to "1", the FSM is always the same:

- First, the request signal is verified to be equal to 1.
- Second, request and address signals are transmitted to neighbour control switch or PE.
- Third, an acknowledgment is waited from neighbour switch or PE.
- Fourth, the acknowledgment is forwarded to the In-controller that initiated the request.
- Fifth, if received request signal returns to "0", the transmitted request is cancelled and next In-controller is checked.

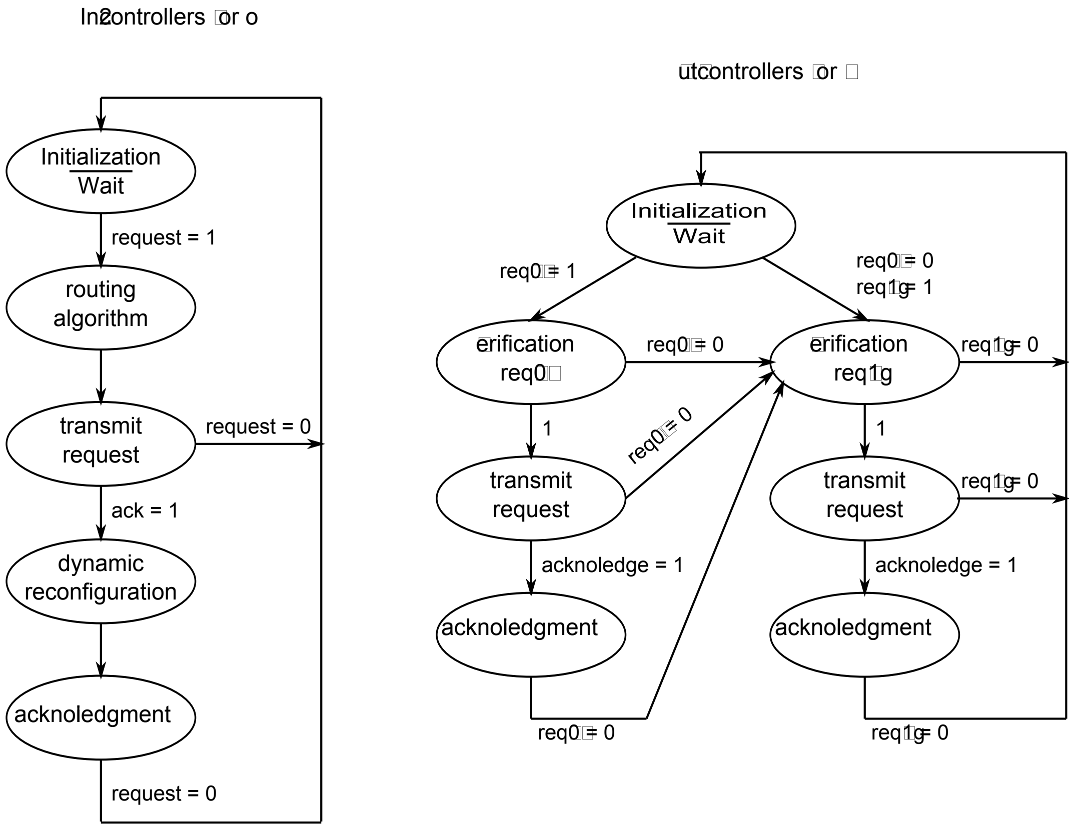


Figure 3.12 : FSMs implemented in In-controllers and Out-controllers 2 or 3 from OCEAN control network.

The FSMs characterizing Out-controllers 0 and 1 are presented in Figure 3.11 while those corresponding to Out-controllers 2 and 3 are presented in Figure 3.12. Due to the $4 - radixn - stages$ fat-tree based structure of the control network, latter Out-controllers could be simplified. Indeed, when a request is climbing down the structure of the tree, assuming it does not go up again, Out-controllers 2 and 3 can only be accessed from In-controllers 0 and 1.

3.10 Variations of the OCEAN network

The OCEAN network that has just been presented corresponds in reality to the version 3.0 of the network. Three other versions are issued from OCEAN v3.0: OCEAN v3.1, v4.0, and v4.1. Since these three versions are directly inherited from OCEAN v3.0, their specificities are now presented.

3.10.1 OCEAN v4.0

The main difference between OCEAN v4.0 and OCEAN v3.0 lies in the routing algorithm. Indeed, an adaptive algorithm confers a better reliability to the network, but may results in decreasing network performances in normal functioning conditions. When several routes are tested for adaptive routing purpose, they are reserved during the test phase and so are not available for other incoming communication requests. This is why OCEAN v4.0 embeds an oblivious (and deterministic) routing algorithm: the Turnback algorithm. The general functioning is quite simple: a request climbs the structure of the control network (ascendant route) until all destinations (if several) are reachable only climbing down the appropriate routes (descendant routes). In order to avoid conflicts, taken ascendant route only depends on the source PE. So, every PE has a dedicated ascendant route: every In-controller 0 transmits requests to Out-controller 2 while In-controller 1 only uses Out-controller 3. Naturally, descendant routes are chosen accordingly with destination addresses. This algorithm has the advantage to make uniform the repartition of the control routes inside the structure of the control network (and consequently also the data one). This algorithm is only made possible thanks to the fat-tree topology and the induced constant bandwidth. Just like for the adaptive algorithm, this strategy guarantees that no deadlock nor livelock can occur. Furthermore, contentions only occur when two PEs want to communicate with the same destination. In this case, obviously the second communication will have to wait for the first one to be finished. The routing algorithms processed in In-controllers 0 and 2 of OCEAN v4.0 control switches are presented in Figure 3.13.

Consequently to the change of routing algorithm, several improvements are provided by OCEAN v4.0 concerning its hardware cost. The first one concerns the controllers embedded inside the control switches. Indeed, if an ascendant request arriving to the In-controller 0 can only be routed up to Out-controller 2 or down to Out-controller 1 (and similarly for In-controller 1), the architecture of all in-controllers can be defined generically. Their FSM is then the same as the one presented in Figure 3.12 for all In-controllers. Out-controllers 0 and 1 can still receive requests from In-controllers 2, 3, and 1 or 0 respectively. Their FSMs are thus the same as the ones presented in Figure 3.11. However, Out-controllers 2 and 3 are deeply impacted. Indeed Out-controller 2 can only receive requests from In-controller 0, and respectively for Out-controller 3. Corresponding FSMs could thus be simplified to the one presented in Figure 3.14.

The data network from OCEAN v4.0 is also impacted by this oblivious routing algorithm. Indeed, a data incoming a data switch from the communication port 0 will never be transmitted through the port 3. Same goes for data arriving from

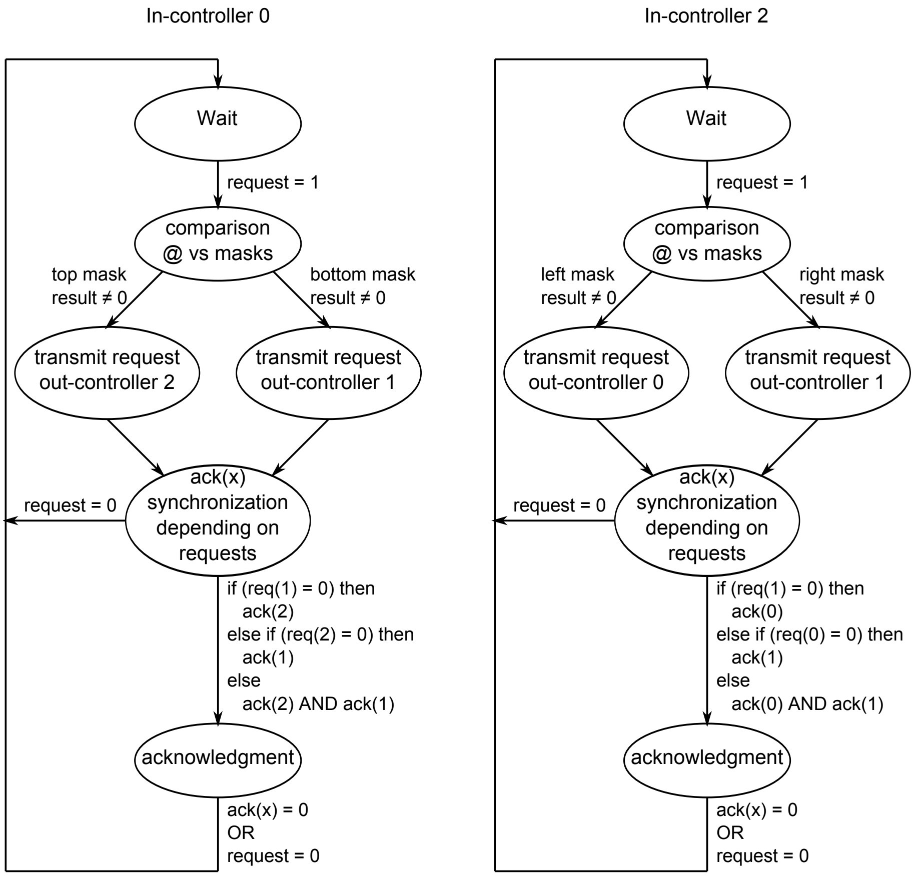


Figure 3.13 : Oblivious routing algorithms processed in In-controllers 0 and 2 of OCEAN v4.0 control switches.

communication port 1. So, as presented in Figure 3.15, only two AND gates are required for each of the bottom level reception ports. Furthermore, the number of wires coming from the reconfiguration can thus be reduced to 8.

3.10.2 OCEAN v3.1 and v4.1

As it can be expected from their names, OCEAN v3.1 and v4.1 are both small evolutions respectively from OCEAN v3.0 and v4.0. Indeed, OCEAN v3.0 and v4.0 target FPGA implementations while OCEAN v3.1 and v4.1 are for ASIC implementations. This impacts how the controllers operate. Since there is no buffer embedded in the control sub-network from the OCEAN versions, a particular care had to be taken concerning parasitic effects that can be caused by transitory states of the various control signals. Transitory states can not be avoided due to hardware reasons (different length of metal wires, propagation delays of the gates, etc.). This is why the synchronization of the controllers is particularly important. For this purpose, FSMs

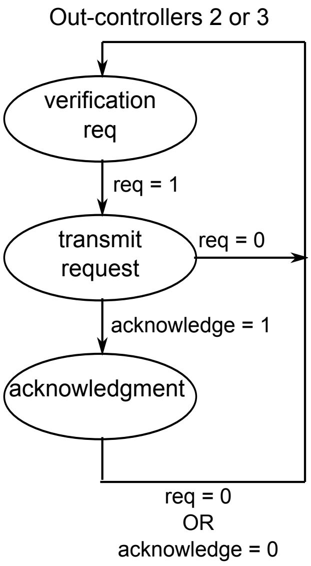


Figure 3.14 : Presentation of the FSM defining the functioning of Out-controllers 2 or 3 from OCEAN v4.0 control switches.

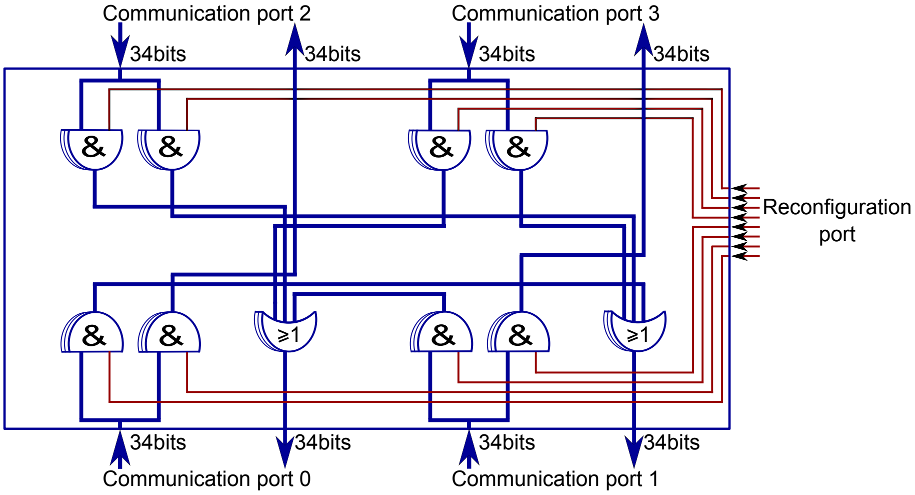


Figure 3.15 : Data switch from OCEAN v4.0 network. The reduction of the number of possible configurations leads to a minimization of the combinatorial gates.

from all OCEAN versions were written as Moore machines. A Moore machine is defined as a finite state machine where outputs only depend on current state and not on inputs (this is the difference with the Mealy machines). Usually, a Moore machine is composed of three main parts:

- The input logic. This logic is composed of combinatorial gates determining the future state that the machine will take depending on the current state and the inputs .

- The state register. Composed of flip-flops, the state register updates, at a specified clock edge, the current state of the machine using the future state calculated by input logic. Current state is sent to the input logic in order to determine again the future state of the machine.
- The output logic. this logic is composed of combinatorial gates. It determines the output of the system considering the current state.

Concerning OCEAN v3.0 and v4.0, FSMs are defined as very classic Moore machines (3.16.A) but OCEAN v3.1 and v4.1 received an extra synchronization layer after the output logic, as presented in Figure 3.16.B. This ensures that all output signals are transmitted only when they are stable. However, in order to minimize the delay to generate an output, synchronization edges of the machine were adapted. The state register operates on a falling edge of the clock reference, while the final synchronization of outputs is realized on a rising edge. This extra synchronization implies an increase of both hardware cost and transmission latencies, but it significantly improves the reliability.

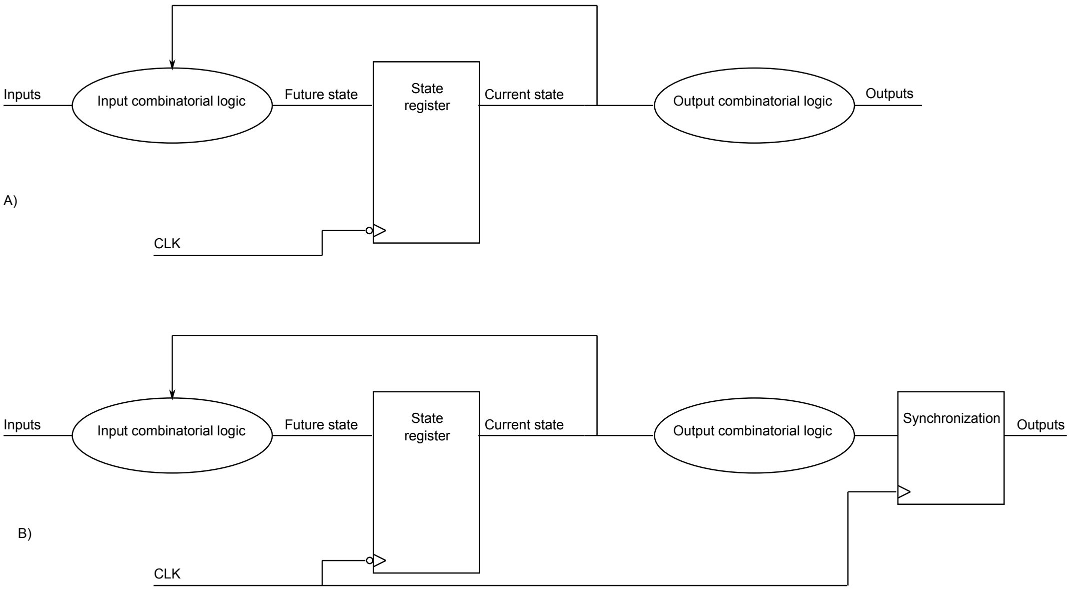


Figure 3.16 : A) Traditional Moore machine used for the FSMs from OCEAN v3.0 and v4.0 control switches. B) Extra synchronization is provided to output signals from OCEAN v3.1 and v4.1 avoiding the propagation of transitory states.

Considering previously defined Moore machines, for all OCEAN versions, but moreover for ASIC oriented ones, a convention was asserted: PEs wanting to create a communication route should send informations (address and request or acknowledgment) to their control switch on a rising edge of their clock reference. However, detection of requests by receiving PEs (and acknowledgments by sending ones) should be performed on a falling edge of the clock. This convention ensures signals to be stable a long time before and after the moment they are read. This way, control switches are more tolerant with jitter and parasitic effects.

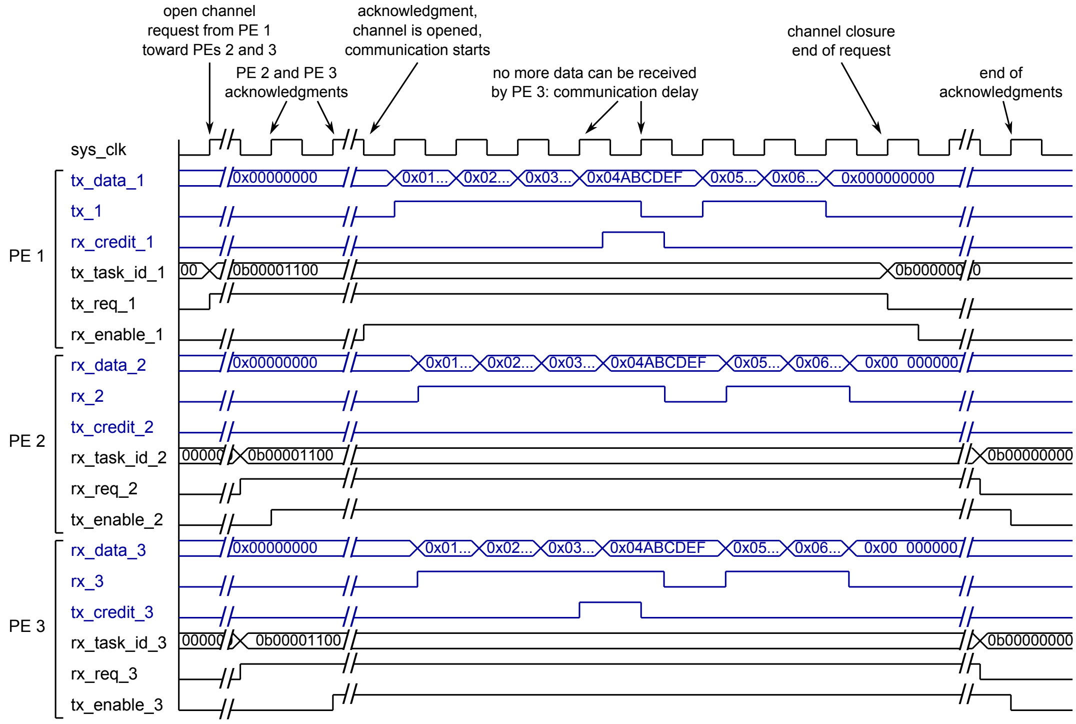


Figure 3.17 : Chronograms presentation of the OCEAN network for a multicast communication initiated by PE 1 toward PEs 2 and 3.

Chronograms of a multicast communication are presented in Figure 3.17. In this scenario, PE 1 initiates a communication toward PE 2 and PE 3. Latter receivers acknowledge the communication only when they are ready to do so. Chronograms are exactly similar for every version of the OCEAN network

3.11 Discussions concerning the OCEAN networks

When considering the whole OCEAN network (whatever the version is) with both data and control sub-networks, an extra functionality is supported: the creation of communication channels. Indeed, an OS always gives an abstraction of communication architecture providing communication channels to the applicative tasks (either hardware or software). When a task wants to communicate with another one, it requests a communication channel (access to the physical medium reserved for its data transfers). Using this channel, the task can send its data to the destination one with a complete abstraction of routing protocols. This is exactly the same principle in OCEAN. A PE (hardware element analogue to the OS vision of an applicative task) requests a communication channel to communicate with one or several others PE(s) through the control network. When channel is created, source PE receives an acknowledgment and can start communicating on this channel (reserved data path) independently from any imposed protocol. So, if the source PE creates a channel toward several destination PEs but wants to send data to only one of them, a bus like protocol of communication can be used. Communication protocol is up to source and destination PEs but does not impact OCEAN data network at all. Similarly, once a communication channel is created, source PE can keep it open even after the end of its data transfer if it wants to send new data in the future. Naturally, it should verify if the data flow graph of the application indicates that it is the only one PE sending data to this destination. This is a significant gain of flexibility for the OCEAN network compared with other proposed NoCs.

The control network that is part of OCEAN versions is synchronized on a clock reference, but the data network remains asynchronous. This has two consequences: first, data can be injected inside the data network at a frequency superior to control network's. Second, the creation of communication channels takes all its sense in this context where its creation induces higher latencies than communications. An opened channel can then be used for several sequential communications with minimal latencies each. This is notably useful for transmissions of control informations where the time of flight is critical. Indeed, in reality the dynamic reconfiguration delay of the data path is negligible compared with the control route establishment time.

3.12 OCEAN test platform

OCEAN is a very innovative network with both interfaces and internal architecture completely different from already existing NoCs. As far as we could investigate, no existing NoC simulator supporting the OCEAN network paradigm could be found. This is why a new test platform was designed. However, since the comparison of different NoCs is always difficult due to their specific interfaces and protocols, the designed test platform aims to be generic (thus supporting as much different NoCs as possible).

The main idea behind this platform is to create a system, defined at the RTL level, in which NoCs are incorporated and characterized from the network performances point of view. The platform contains generic PEs generating a data traffic. Since NoCs may have different interfaces and communication protocols, the injection of data in the network is realized through NoC specific NIs. One PE is connected to every communication port of the network to characterize using these specific NIs. Every PE has the same behaviour: an emission part generates and injects a data traffic in the network, and a reception part interprets received data.

Every PE is synchronized on a global clock signal. This forbids PEs to operate at different frequencies. This point can seem to be a drawback for the platform, but it is most cost efficient for an application to possess synchronous communications (additional signal are then avoided in the network) even if PEs have different frequencies. The clock adaptation is realized in the NIs through asynchronous buffers. So, if in real life applications PEs can thus have specific frequencies depending on the function they realize, in this platform they can be synchronized in order to avoid additional buffer implementations.

The emission part of a PE needs several informations that should be provided through a standard interface:

- The source address (needed by several routing algorithms, this is the address of the emission part of the PE).

- The destination address of the packet to generate.

- The number of data (flits) that have to be transmitted in this packet.

- A delay information, in clock cycles, indicating the time that should be elapsed between the end of a transmission and the beginning of the following one. Indeed, with this platform, users should be able to master the data rate of the traffic injected in the NoC: the offered traffic. For example, if delay is set to 0, every communication starts immediately at the end of the previous one. The offered traffic is thus maximal, corresponding to an injection rate of 100%. Similarly, if the number of data to transmit is equal to the delay information, data are injected half of the time, leading to a 50% injection rate.

- A counter: in order for the receiving parts of the PEs to calculate latencies, time informations should be included in transmitted data packets. This is why a 32 bits counter was created. Shared by every PE (and both emission and reception parts), this counter brings coherent time values that are used to date the various phases of a communication: emission and reception of the very first control signal (header, channel request, etc.), the emission of the first data packet, and the reception of the last transmitted data.

In order for the emission part of a PE to produce different data traffics, an additional signal is defined: every PE indicates to an address controller when the destination address can be changed for next packets. This address signal is used to guarantee the integrity of sent destination addresses making sure that this address will not change during its transmission.

The reception part of a PE computes several informations from received data. Comparing the emission time of the first control signal with the counter at the time it arrives, the latency corresponding only to the path creation is determined. In some networks like DRAFT, HERMES or the conventional fat-tree, it corresponds to the time of flight of the first flit: the header. However for networks like the four versions of OCEAN, it indicates the time in which a data path (communication channel) is configured for incoming communication(s). Similarly, comparing the time from which the first data flit was sent and the moment the last arrives gives informations about transmission times. Some delays could result for example from contentions in the network structure during the time data were going through it. Finally, the addition of these two latencies gives the transport latency: time elapsed between the beginning of the communication and the reception of the last data. This is the latency that will be presented in Chapter 4.

The reception part also calculates two values: the number of clock cycles during which data arrived giving information about the number of received flits (and thus of received bytes), and the number of clock cycles with no incoming data. When related with the simulation time, the number of clock cycles with incoming data is used to calculate the data rate provided by the NoC. This is the accepted traffic. When normalized, in percentage, this gives the throughput of the network. In case the simulation time would be unknown, the number of clock cycles elapsed without receiving any data allows to calculate the same throughput.

The reception part of a PE thus provides following informations:

- The transport latency.
- The number of clock cycles with incoming data.
- The number of clock cycles without incoming data.

The platform is presented in Figure 3.18. Since results (latencies and numbers of clock cycles) are produced by every PE for each packet of data, they are assigned as outputs of the platform. When simulated with ModelSim 6.6b, all the results can be stored in a list file (containing every result from every packet that has been transmitted through the network). Then, global network performances can be calculated from the list file using Matlab.

Network performances are highly linked with injected data traffic. Depending on the distribution of the traffic, in time and space, the network will be more or less stressed. For this platform, a traffic fully repeatable was expected in order to test different NoCs in exactly the same conditions. In this first version of the platform, the destination addresses are cyclically chosen. This principle is pictured in Figure 3.19 for a PE numbered n interconnected through a network with $m-1$

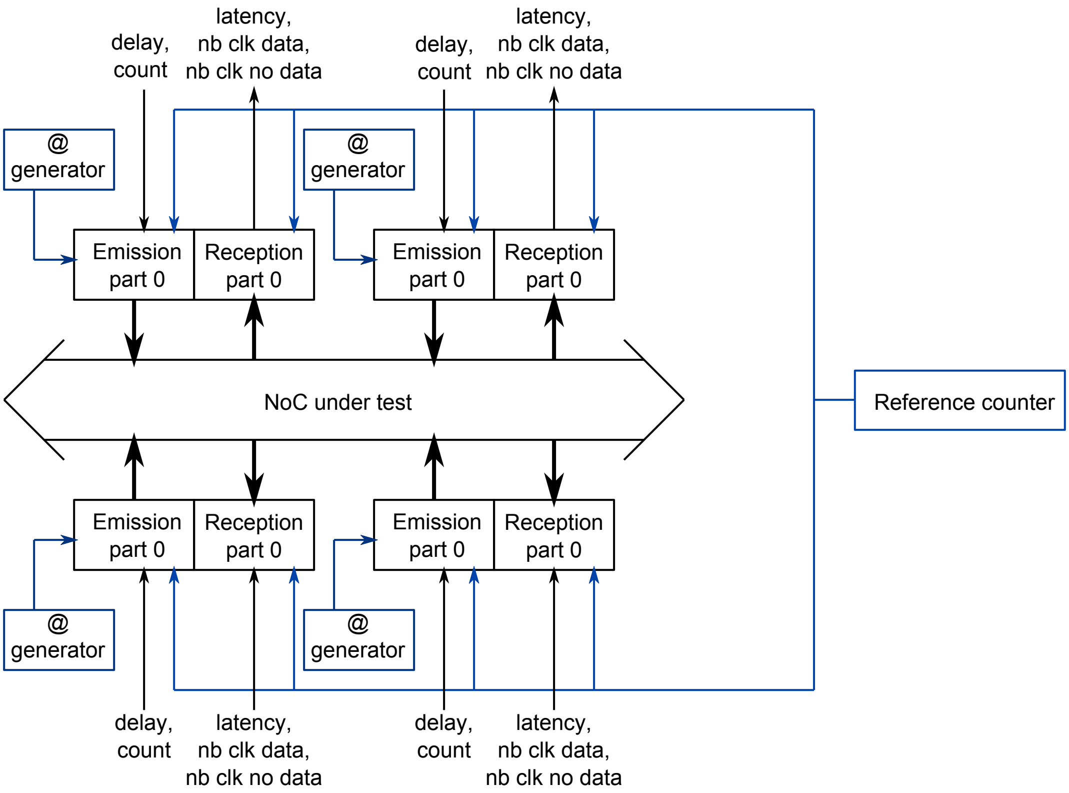


Figure 3.18 : Presentation of the characterization platform for a NoC with 4 input/output ports. The entire platform is coded in VHDL. In can be simulated with ModelSim or implemented in FPGA. Generated results should be processed with Matlab in order to extract global network performances.

PEs. Furthermore, if PEs are numbered from 0 to m-1, the PE number n always starts sending packets to the (m-1)-n address. In order to stress even more the network, increasing the probability of conflicts and of contentions, several sizes of packets are injected. This traffic is called Permutation. Quarter of PEs will send packets of 8 flits, another quarter will send 16 flits per packet, another one will send 32 flits per packet, and finally the last quarter will send 64 flits per packet. This will create a pseudo random data traffic. If the number of flits per packet differs from a PE to another, delay informations are chosen accordingly in order for every PE to inject exactly the same rate of data in the network.

Finally, there are two main differences between network performances measured from DRAGOON (presented in Chapter 3) and the characterization platform. The first one is obvious, generated data traffics are different so that obtained performances such as latencies will differ. This is why, the conventional fat-tree generated with DRAGOON will be characterized along with OCEAN networks using this platform. The conventional fat-tree was chosen instead of HERMES because its topology is exactly the same as for the four versions of OCEAN, so that the same data traffic

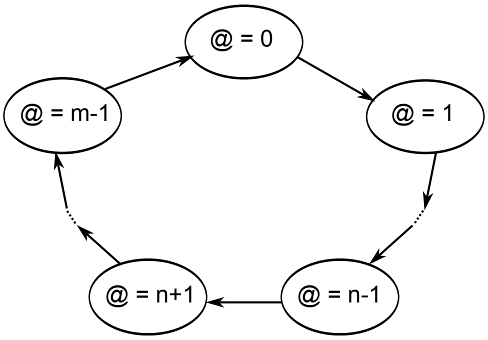


Figure 3.19 : Cyclic addressing of the packets' destination injected by the PE number *n*.

can be used. The second difference concerns the dating of the packets. In DRAGOON, the dating is generated with the traffic before the simulation, accordingly with desired injection rate. With such dating, when a network saturates, many packets are injected a very long time after the moment they should have been in an ideal network. Resulting latencies explode. However, even when a network saturates, data are still transmitted even if the accepted traffic is lower than the offered one. Since the saturation of a network can also be observed with the number of transmitted packets (reaching a maximum in stable networks, or decreasing although offered traffic increases in unstable networks), the choice was made to date the packets at the time they are really transmitted. This way, calculated latencies are only those of the transmissions. Thus, informations can be gathered from behaviour of the networks when they saturate. This is a fundamental difference with most of research papers: it is normal, and done on purpose, that latencies do not explode when a network saturates. Complete characterization results are presented in Chapter 4.

3.13 Synthesis of the chapter

In this chapter, several new interconnection networks were presented. First was presented the R2NoC network. This network is designed for DPR compliant FPGAs. Main particularity of R2NoC lies in the definition of its switches that are only constituted of a PRR where direct links are dynamically configured. This network is controlled by a centralized communication service. Data are circuit switched through the network. However, technological limitations make this network quite impractical for industrial utilization. This is why the OCEAN network was proposed. The four OCEAN versions embed two networks, one dynamically reconfigurable circuit switching data, and one processing the communication service while managing the reconfiguration. OCEAN v3.0 is characterized by an adaptive routing algorithm. It is particularly designed for FPGA implementations. The OCEAN v3.1 in an adaptation of OCEAN v3.0 in order to be compliant with ASIC implementations. OCEAN v4.0 is inherited from OCEAN v3.0 but possess an oblivious routing algorithm allowing a significant reduction of its hardware cost. OCEAN v4.0 is also designed for FPGA implementations. Finally, the OCEAN v4.1 is an adaptation of the OCEAN v4.0 network in order to be ASIC compliant. OCEAN networks are independent from any industrial technology concerning dynamic reconfiguration. The behaviour of dynamically reconfigured data paths is provided by combinatorial logics. From the control point of view, OCEAN networks dispose of a deadlock free, and minimal routing algorithm. Multicast also with creation of communication channels are supported by the OCEAN networks. A specific platform was also created in order to compare the OCEAN networks with a conventional fat-tree simulating an FPGA implementation.

Chapter 4

Results and comparisons

In this chapter, all previously presented NoCs are characterized and compared. The DRAFT network is first compared with a conventional fat-tree (from which it is inherited), and to the most popular NoC: the mesh topology based HERMES [81]. Details concerning the hardware overhead induced by these networks when implemented in FPGA are also provided. Furthermore, the four versions of the OCEAN network are characterized in FPGA. They are compared with the same conventional fat-tree that was used for DRAFT. The two versions of OCEAN which are ASIC oriented were also characterized specifically for these architectures.

4.1 Objectives

The comparison of different topologies of networks leads to some problems that must be considered before any experimentation. When the number of interconnected Processing Elements (PEs) varies, the size of the networks should be chosen accordingly. However, if considering a mesh based network interconnecting 8 PEs, there are several ways to do so: designing a 4x2 mesh, or a 3x3 one but with an unconnected router. Similarly, a fat-tree interconnecting 9 PEs have many routers that will remain unused, and that could be removed for optimization purpose. This leads to a very high number of configurations to be tested. In order to avoid that, we evaluate only full networks. It means that whatever the number of connected PEs is, the networks will always have a complete structure: a complete binary tree for fat-trees, and a square matrix for meshes. Thus, sometimes, several routers will not connect any PEs but the comparison will be fairer. Unconnected routers are always located in the same corner of a mesh based network (south east), at the bottom right end of the trees, and at the top right end of DRAFT.

When characterizing networks, several parameters must be evaluated:

- the hardware resources consumption,
- the scalability,
- the latencies,
- the throughput,
- the maximum data rates.

However, comparing existing networks is a very difficult task. Indeed, most of existing Networks-on-Chip (NoCs) are proposed with a dedicated environment for generation and simulation purpose. As an example, this is the case for HERMES with the ATLAS environment. The two most famous simulator of networks are undoubtedly NS2 and Orion [106], [108]. These two simulators are very powerful and allow to simulate a large variety of networks. However, they are based on models for general networks but not for NoCs. Making them supporting NoCs would require a very significant investment in both manpower and time. Several simulator were designed for NoCs, such as GpNoCsim [50], Nirgam [6], and Noxim [90]. GpNoCsim was designed to support several topologies of NoCs: butterfly fat-tree, mesh, torus, and extended fat-tree. However, this simulator is based on models concerning both the behavior of every component of a NoC, and the performances estimation. Implementing new NoCs, with new topologies and novel router architectures would require to re-design every models. This is also the case for the simulator based on SystemC models presented in [63]. Nirgam and Noxim (which is the most popular simulator in research community and was extended to support new routing algorithms) are very limited in offered possibilities of simulation. Noxim does not use models for performance estimation but is compliant only with routers composed of 5 communication ports related to mesh based networks. Nirgam is very close from Noxim, notably adding the support of torus based topologies. Latter simulator supports only SystemC written NoCs and is not compliant with RTL level designed networks. This drawback, combined with the limited number of supported topologies (that may lead to require substantial improvements of the simulators themselves) are the reasons why these simulators were not retained as practical solutions for NoC simulation purpose. Considering this problem of simulation environments, only HERMES and a conventional fat-tree are used for comparison purpose. More networks should have been of a high interest to be compared with, but the lack a generic characterization environment supporting a large variety of NoCs avoids such comparisons.

The Dynamically Reconfigurable Architectures compliant Generator and simulatOr Of Network (DRAGOON) environment was designed in order to measure performances of both DRAFT and the conventional fat-tree. Further details on the DRAGOON environment are presented in Chapter 2, .

4.2 Experimental conditions

This section describes the conditions in which all networks were characterized and compared. The hardware resources consumption of the networks highly depends on the targeted chips. Indeed used resources are different between an FPGA and an ASIC so that implementation results will differ (hardware resources consumption, maximal frequency, etc.). Furthermore, even considering the same target, results can differ between two versions of the software design tools even if they target the same chip. Every network is characterized in FPGA. Since DPR is the leitmotiv of this PhD, Xilinx architectures and in particular the Virtex V ”XC5VSX50T” is considered for all FPGA implementations. Furthermore, the Xilinx ISE 12.4 design suit is used to measure hardware costs along with maximal operating frequencies targeting this specific FPGA. Two versions of the OCEAN network (versions 3.1 and 4.1) were designed for ASIC implementations. So, results are specifically provided for these two versions targeting the 130nm CMOS technologies from STMicroelectronics.

Concerning network performances, DRAFT and the conventional fat-tree were evaluated and compared using the DRAGOON environment (see Section 2.5). Similarly, the HERMES mesh network is evaluated through its dedicated ATLAS environment. Both environments allow to create similar data traffics since DRAGOON is directly inherited from ATLAS. Furthermore, they both use the same external simulator for performance evaluations: ModelSim 6.4c. Unfortunately, OCEAN versions could not be characterized inside DRAGOON or at the cost of a complete re-design of the whole environment. This is why a new platform was designed (see Section 3.12). This platform supports both the four versions of OCEAN and the same conventional fat-tree as for the comparison of DRAFT. A repeatable data traffic is generated thanks to this platform, thus allowing to simulate and compare NoCs with exactly the same data. Raw simulation results are provided by ModelSim 6.6c. They are then processed using Matlab R2010b leading to final results.

Latencies, throughput, and accepted data rates are the main expected results (see Section 1.2). The main problem for the comparison of network performances is the data traffic. Indeed, all these metrics are intrinsically linked with the data traffic: number of simulated PEs requesting to access the same destination at a time, length of transmitted data, uniform repartition of the traffic in terms of couple source/destinations, etc. For the comparisons between DRAFT, the conventional fat-tree and HERMES, network performances are evaluated using a basic data traffic with randomly chosen destination addresses and a uniform temporal distribution of the communications. Transmitted data packets are composed of 16 flits of payload with a data with of 32 bits. Concerning DRAFT, since no communication is allowed between top level communication ports, destination addresses of the top level simulated PEs were constrained aiming only bottom level destinations. For the comparison of the four versions of OCEAN with the conventional fat-tree, a data traffic stressing more the networks was chosen. This so called permutation traffic is based on cyclically permuted destination addresses. Data packets injection is realized following a uniform temporal repartition but with various lengths (8, 16, 32, 64 data flits respectively per quarter of the connected PEs).

For all networks, simulations are realized at a frequency of 100MHz during 10ms. This frequency is assigned to both injected data (simulated PEs) and network internal clock references. However, when expressed in μs, final results are provided taking into account the maximal frequencies that can be obtained from FPGA implementations.

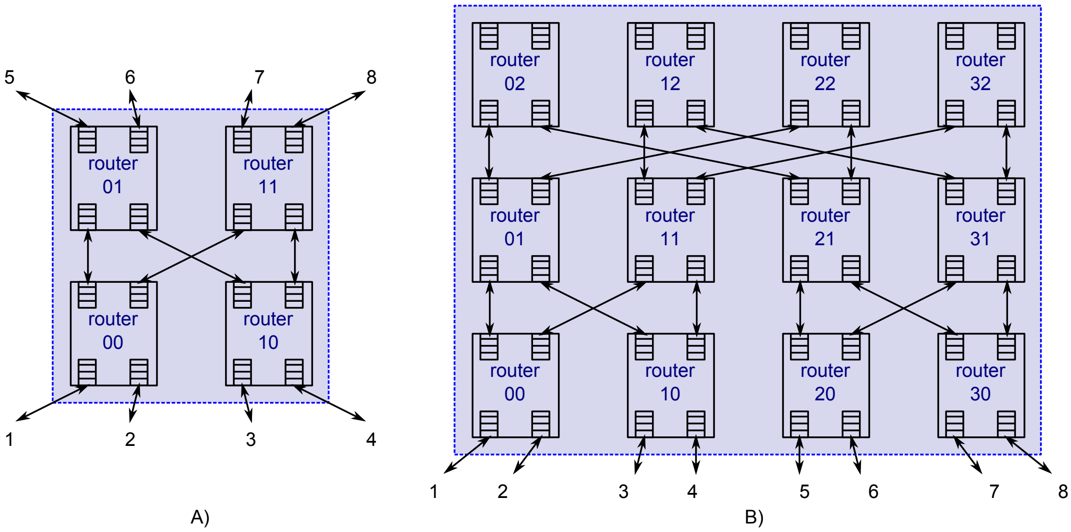


Figure 4.1 : A) Basic DRAFT network from which characterizations and comparisons are performed. B) Conventional fat-tree used for comparison purpose.

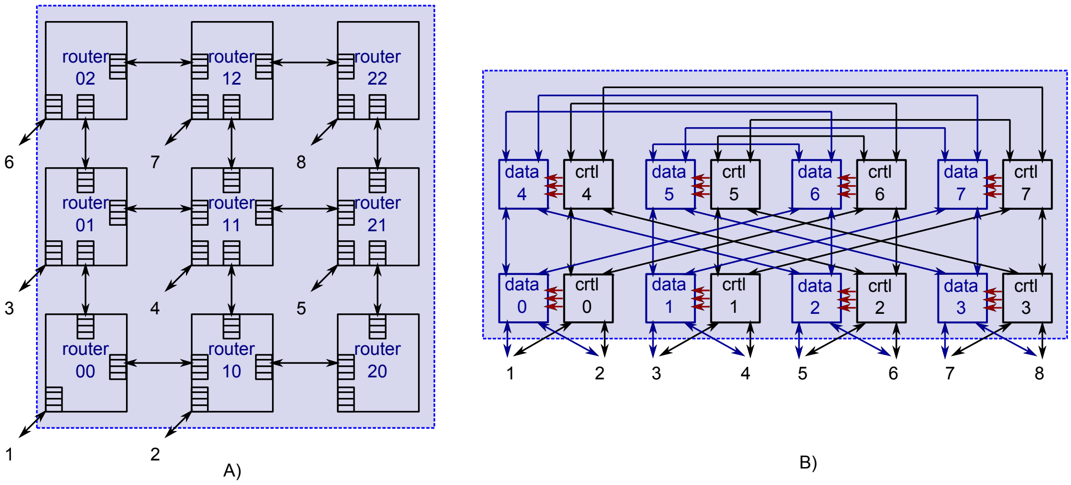


Figure 4.2 : A) Square mesh network with 8 communication ports. B) Basic OCEAN network from which characterizations and comparisons are performed.

Every network is evaluated starting from a basic configuration with well defined characteristics. Only one characteristic is changed at a time so that its influence can be isolated. Every basic version of the various networks interconnects 8 simulated PEs, as presented in Figures 4.1 and 4.2. These basic networks are implemented and simulated with 32 bits data width, a buffer depth of 4 flits (for packet switched ones), and no virtual channels. The credit based flow control protocol was chosen for DRAFT, the fat-tree and HERMES.

4.3 DRAFT performances and comparison

In this section, main results concerning DRAFT performances are presented. For each evaluated performance, a comparison is realized with the HERMES NoC and a conventional fat-tree. Experimental conditions are always the same for the three networks.

4.3.1 Hardware resources consumptions

In order to compare the hardware consumption of the DRAFT, fat-tree and HERMES NoCs, several configurations were generated in order for every one to interconnect from 2 to 64 PEs. As presented in Section 4.1, only complete networks are implemented: square meshes and complete trees. This principle is sometime more favorable to a network compared with another one (and sometime it is the opposite) because different topologies do not scale the same way with the number of PEs to interconnect. However, globally, it allows a fair comparison. Results are presented in Figure 4.3.

The number of used links is the only results that is not generated by ISE. Indeed it corresponds to the number of full-duplex links required to interconnect both the routers and the PEs. So, every link is made of 70 wires. This result is provided because FPGA matrices are designed with more and more CLBs, but the interconnection wires are starting to become a bottleneck for on-Chip implementations. From the results, links and CLBs consumptions of DRAFT and HERMES are very close. DRAFT consumes less CLBs and less links from 2 to 16 connected PEs. Then from 16 to 25 PEs, the mesh has a lower but very close hardware cost. From 25 to 64 PEs, hardware consumptions remain very similar, denoting a scalability almost identical for both networks. Indeed DRAFT allows only an average gain of 0.3% of required CLBs compared with HERMES. This is a very important result for DRAFT because the minimization of its hardware cost was a key challenge for its conception. As it could be expected, a fat-tree uses more hardware resources (both CLBs and links) than the two other networks, notably between 32 and 64 connected PEs. In average, DRAFT presents an hardware cost 55% less than the conventional fat-tree. Furthermore, DRAFT reduces by 52% the number of used links compared with the fat-tree. Even though, between 2 and 32 PEs, the fat-tree remains a viable candidate for on-Chip implementations.

ISE also provides informations concerning the maximal frequencies that are supported by the networks. From these results, it appears that both DRAFT and the fat-tree can operate at similar frequencies that are higher than for HERMES. The difference is never more than 8MHz but increases network performances of both DRAFT and the fat-tree. Indeed, maximal operating frequencies of both DRAFT

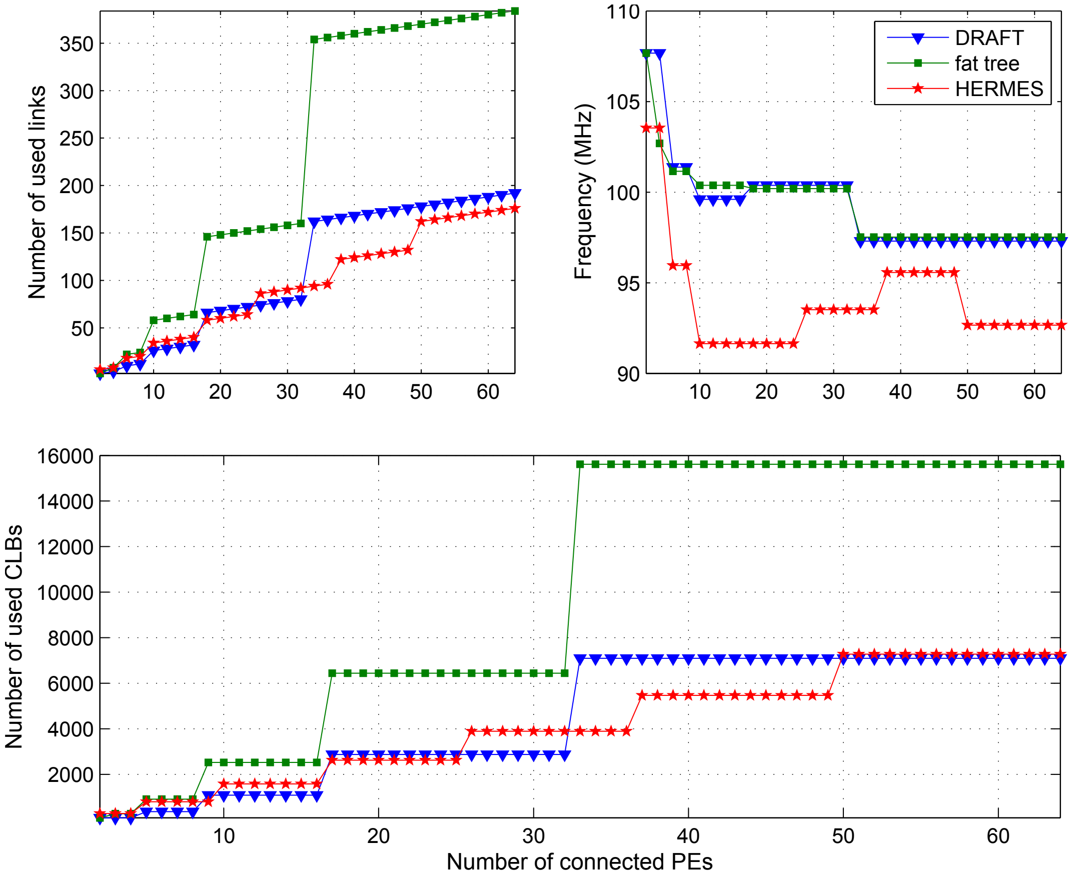


Figure 4.3 : Hardware costs to implement the DRAFT, fat-tree, or HERMES NoCs depending on the number of connected PEs.

and the fat-tree present an average gain factor of 5% compared with HERMES.

4.3.2 Network performances

Considered network performances are both average latencies and throughputs. Since offered data are dated from the moment they are generated, and not from the moment the network can accept them, resulting latencies explodes when the networks saturate. Indeed, when a network saturates, no more than a limited number of packets are accepted (and then transmitted). If more packets are offered to the network, since they can not be immediately accepted, they induce long delays. Since some of them will never be transmitted, resulting latency tends to the infinite. Saturation of a network can also be noted from the throughput which is the expression of the accepted traffic normalized in percentage.

As presented in Figure 4.4, DRAFT provides lower latencies and supports higher throughputs than the other networks. Due to its reduced number of routers, data

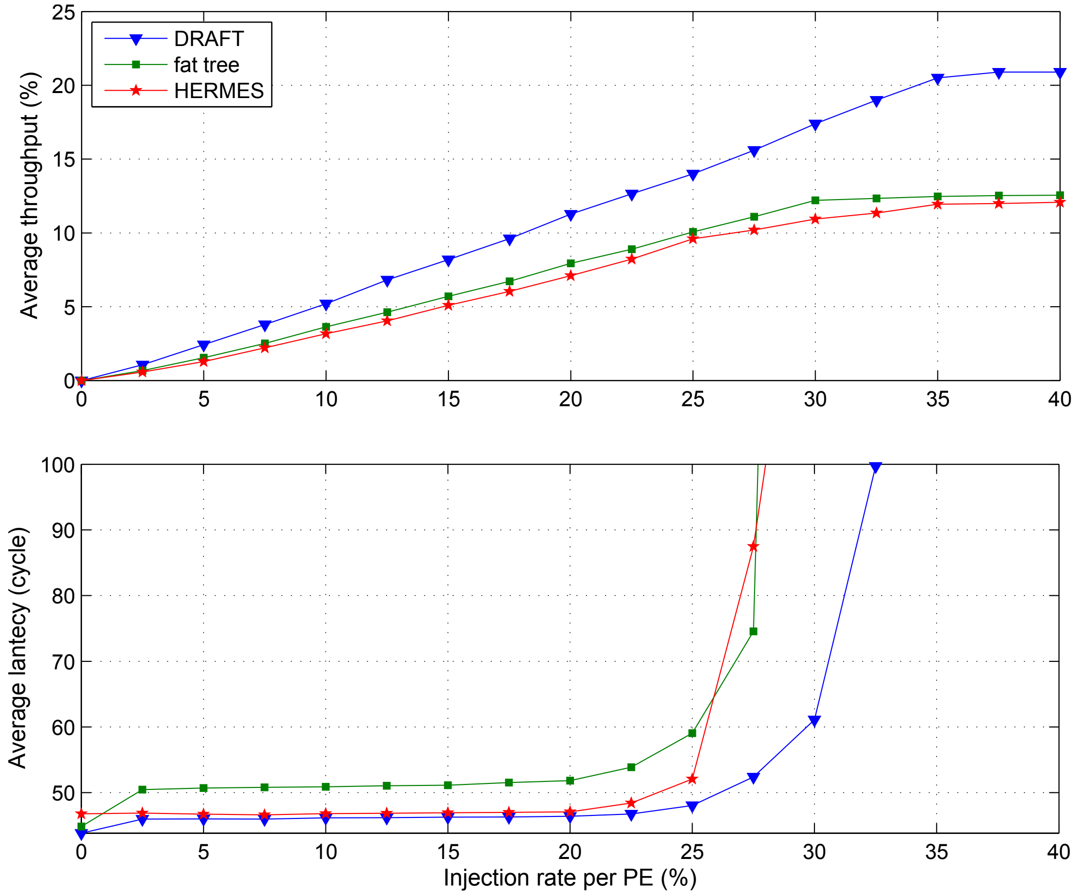


Figure 4.4 : Average latencies and throughputs depending on the offered transfer rates per PE. All PEs and networks are clocked at 100 MHz.

reach their destinations with lower latencies (in average) than in a fat-tree or in HERMES. From the latencies point of view, HERMES and DRAFT are very close while the conventional fat-tree suffers from its higher number of routers. The saturation zone is almost the same for every type of networks: latencies significantly increase between 25 and 30% of injection rates corresponding respectively to 800 and 960 Mbit/s transfer rate per PE. When networks do not saturate, DRAFT offered latencies present an improvement of 10% compared with these of a fat-tree and 3% compared with HERMES.

From the throughput point of view, the networks start saturating in the same range of injection as for latencies (obviously) but limit values are reached between 30% and 35% injection rates (corresponding respectively to 960 and 1120 Mbit/s per PE). For practical reasons, performances corresponding to Injection rates higher than 40% (corresponding to 1280 Mbit/s) were not pictured, but average throughputs remain stable for every networks. Networks are then qualified to be stable.

Furthermore, before data rates reach the saturation zone, DRAFT throughput is 28% higher than for the fat-tree and also 35% higher than for HERMES. This traduces higher accepted traffics. When considering the saturation zone, from 1120 to 3200 Mbit/s (the maximum possible data rate per simulated PE clocked at 100 MHz with a data width of 32 bits), DRAFT accepted traffic is 15% and 16% higher respectively than conventional fat-tree and HERMES. However, these very interesting performances should be balanced with the traffic limitation inherent to the DRAFT network that top level connected PEs can not communicate together.

4.3.3 The scalability

The scalability is the study of performances when the number of interconnected PEs increases. The impact on hardware resources consumption was already presented in Section 4.3.1. This is why only network performances: average latencies and maximum data rates per PE are considered here. Results are presented in Figure 4.5.

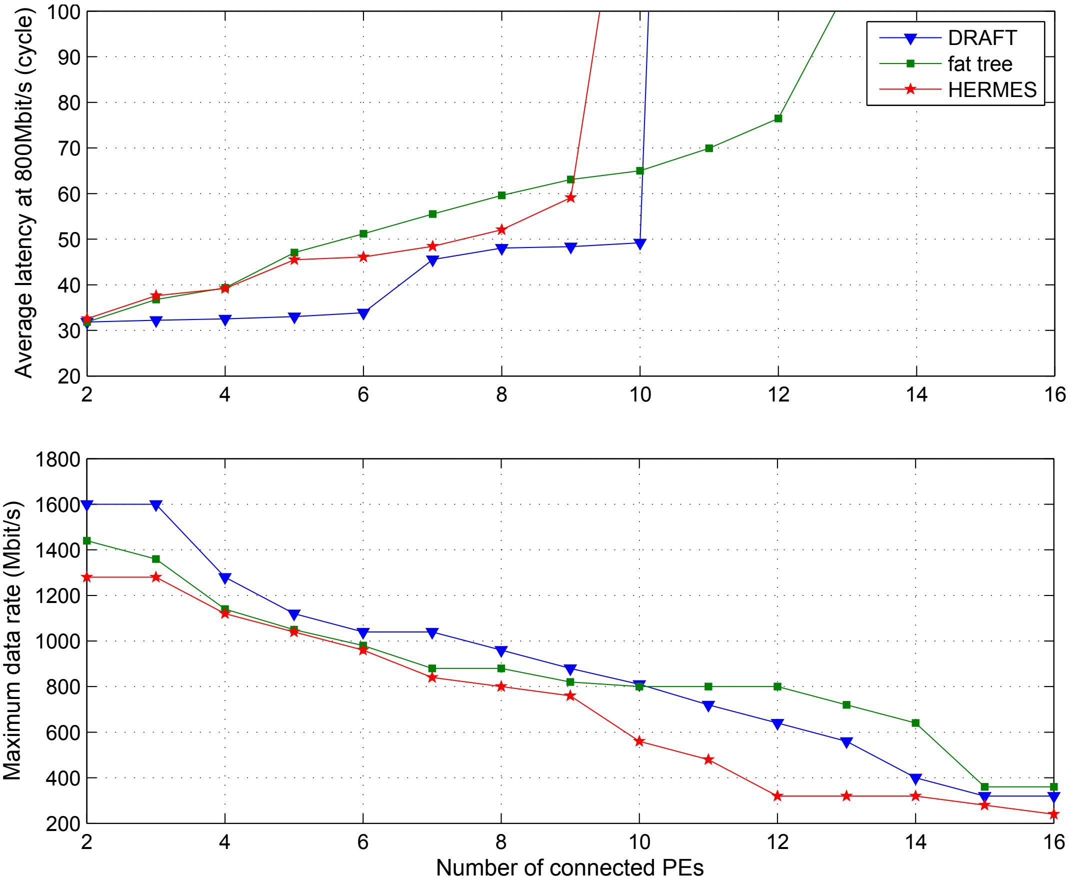


Figure 4.5 : Average latencies measured for an offered traffic of 800 Mbits/s per PE, and maximum achievable data rates depending on the number of connected PEs.

From these results, the conventional fat-tree supports higher data rates per PE than the other networks, notably when their number increases. This observation could be expected because fat-tree based networks are mostly designed to support high number of connected elements with high network performances. So, at a fixed data rate per PE of 800 Mbit/s, a fat-tree supports 31% more connected PEs then HERMES and 23% more than DRAFT. When fixing the number of connected PEs and observing the maximal accepted data rates, until 10 connected PEs DRAFT supports respectively 9% and 16% higher data rates than the fat-tree and HERMES. However starting at 10 connected PEs, the fat-tree outperforms DRAFT supporting 7% higher data rates. In the same range of connected PEs, DRAFT is compliant with 14% higher data rates per PE than HERMES. However, the ability of a fat-tree to support efficiently high number of connected PEs is counter-balanced by a significant hardware cost. Considering the close hardware resources consumption of HERMES and DRAFT, from the scalability point of view this latter appears to be an interesting trade off between achievable network performances and hardware resource consumption.

4.3.4 The data width

The data width directly impacts the architecture of a network. Indeed, all the router architecture depends on this width. Obviously, the buffers are dramatically impacted since they have to store several data. Crossbars performing the routing of data also depend on the data width. To a lesser extent, switching controllers are also impacted since the size of used addresses and masks is defined accordingly with the data width (width of both header and count flits). To determine the influence of the data width over both hardware resources consumption and network performances, the offered data rate per PE could not be fixed to 800 Mbits/s because the three networks saturates with a data width of 16 bits. This is why network performances are obtained from an offered data rate equal to 400 Mbits/s per connected PE. Results are presented in Figure 4.6.

From the hardware results, DRAFT appears to be just a bit less impacted by the data width than HERMES when considering the CLBs consumption. This is due to its reduced number of routers. If increasing the data width from 16 to 64 bits implies an hardware overhead of 62% for HERMES, the overhead is of 61% for both DRAFT and the fat-tree. However, the main difference between the three networks concerning hardware implementation concerns the maximum operating frequencies which decrease when the data width scales up. Frequencies are very close but it is interesting to observe that HERMES frequency (that is lower than those of DRAFT and the fat-tree) is less influenced by increasing data widths. HERMES maximum

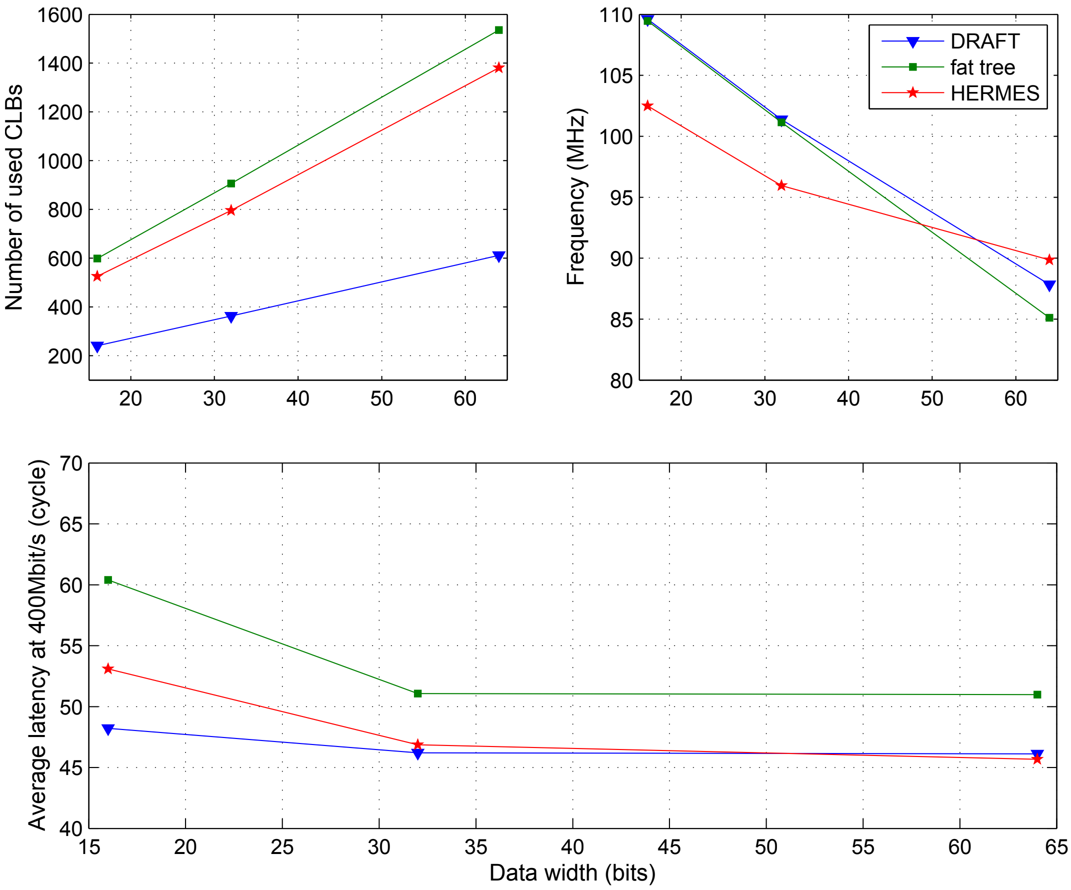


Figure 4.6 : Impact of the data width over hardware resources consumption (used CLBs and operating frequencies), and average latencies with an offered data rate of 400 Mbits/s per PE.

frequency becomes even higher than those of DRAFT and fat-tree for a 64 bits data width. Indeed the maximum frequency of HERMES only decreases from 12% although those of DRAFT and the fat-tree respectively decrease from 19 and 22%.

Concerning network performances, at 400 Mbits/s, the influence of the data width is only relevant between 16 and 32 bits. With this offered data rate, configuring the data width to 32 bits decrease average latencies (thus increasing network efficiency). Increasing the data width from 16 to 64 bits decreases the average latencies of a fat-tree from 16%. However, concerning DRAFT and HERMES, they decrease respectively their transport latencies by a factor of 4% and 14%. So, at 400 Mbit/s per PE, reducing the data width in DRAFT impacts less performances than for other networks. This is interesting because considering the significant hardware overhead induced by the data width, this overhead can be minimized in DRAFT with similar performances. However, the interest of large data widths (like 64 bits) is much more interesting when an application requires data rates per PE reaching limit values presented in the Section 4.3.3 (for example 960 Mbit/s for a DRAFT connecting 8 PEs). Only in this applicative case 64 bits data width is advantageous.

4.3.5 Buffer depth

Just like for the data width, the influence of the the buffer depth is now studied. The depth of buffers defines their capacity to store 4, 8, 16, or 32 flits. Contrarily to data width experimentations, concerning buffer depths network performances could be obtained from simulation with 800 Mbits/s per PE. Results are presented in Figure 4.7.

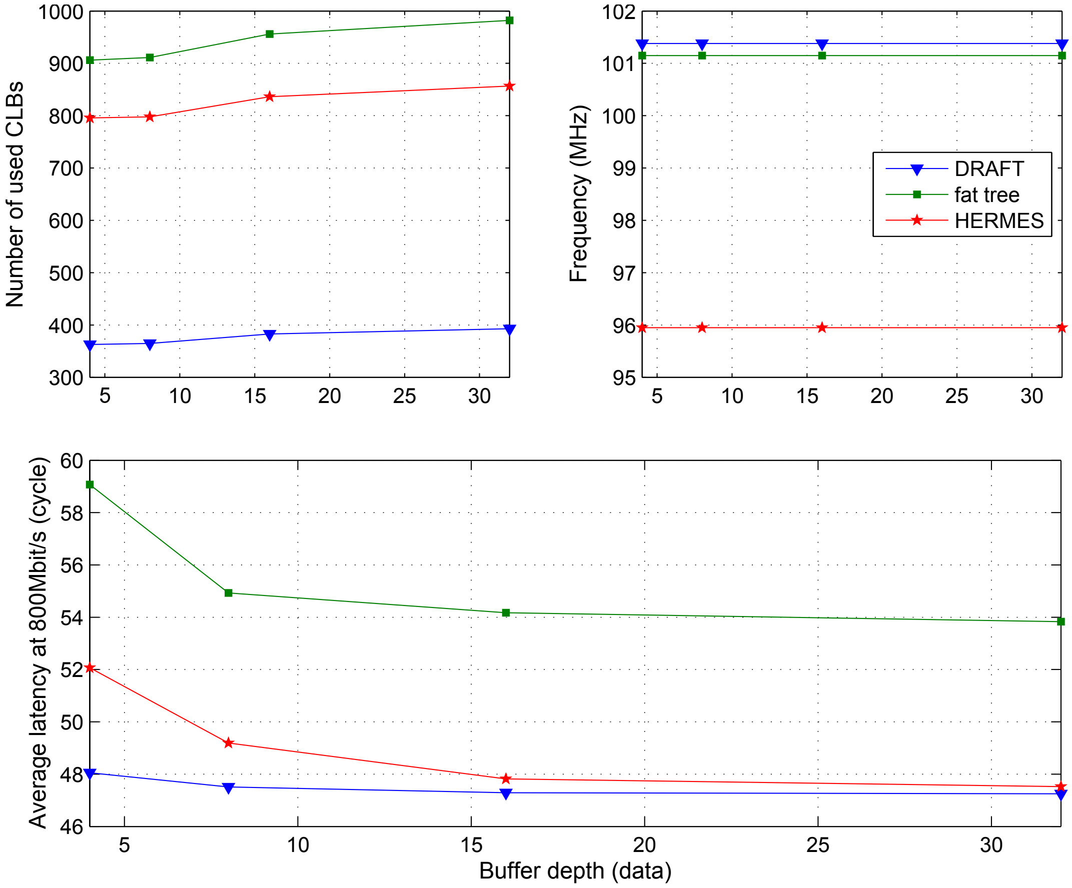


Figure 4.7 : Impact of the buffer depth over hardware resources consumption (used CLBs and operating frequencies), and average latencies with an offered data rate of 800 Mbits/s per PE.

From these results, it appears that the buffer depth much less impacts the hardware resources consumption than the data width. Only buffers are impacted, not the crossbars nor the switching controllers. So, the hardware overhead induced by increasing the buffer depth is reduced to 8% for both DRAFT and the conventional fat-tree, and to 7% for HERMES. Furthermore, maximum operating frequencies are not influenced at all by the buffer depth. The impact over networks performances is much significant in the low capacities: between 4 and 16 data. Higher capacities can be useful when expected offered data rates are close from limit rates presented in Section 4.3.3. Like for the data width, DRAFT is less influenced concerning network performances when the buffer depth is minimized. Indeed both fat-tree and HERMES present an impact of 9% over average latencies between buffer depths of 4 and 32 flits, while DRAFT presents only a difference of 2%.

From the applicative point of view, when expected data rates reach the limits of the networks, the buffer depth should be configured to reach and guarantee required performances more than the data widths. The objective is to limit the hardware impact while supporting desired performances. When high buffer capacities are not sufficient to support required performances, then data width can be adapted. Otherwise, both parameters should be minimized, thus taking advantage of DRAFT which is less influenced regarding the network performances.

4.3.6 Types of data traffics

Thanks to DRAGOON and ATLAS, several data traffics could be generated for each network. Their influence over network performances (average latencies) is now studied. Two data rates are studied: 400 and 800 Mbits/s per PE in order to differently stress the networks. Source and destination addresses are randomly generated, so that the types of traffic represent the distribution in the time of the communications. Results are presented in Table 4.1.

Table 4.1 : Influence of the different data traffics over the latency (average clock cycles). The Δ ranges corresponds to the latency difference between Uniform traffic and corresponding traffic.

Networks	DRAFT		fat-tree		HERMES	
Type of traffic	Latency (clk)	Δ (%)	Latency (clk)	Δ (%)	Latency (clk)	Δ (%)
Uniform (800Mbit/s)	48.27	0	58.22	0	51.91	0
Normal (800Mbit/s)	43.40	4.87	56.71	1.51	49.21	2.7
Burst (800Mbit/s)	33.11	15.16	42.42	15.80	38.40	13.51
Uniform (400Mbit/s)	46.20	0	51.07	0	46.87	0
Normal (400Mbit/s)	35.58	10.62	43.28	7.79	38.65	8.22
Burst (400Mbit/s)	31.43	14.77	39.83	11.24	35.64	11.23

From these results, the three types of traffics are supported by the networks, even at 800 Mbits/s. However, when calculating the difference between performances obtained with a Uniform traffic and those from other traffic (Δ ranges with $\Delta = Latency_{uniform} - Latency_{consideredtraffic}$), DRAFT appears to be more sensitive than the other networks. This is a good point for DRAFT because it denotes decreasing communication latencies. For example, DRAFT supports a bit more bursts of data than HERMES and even the fat-tree at 400 Mbit/s due to a fall close of 15% of the latencies. Even if the difference between the three networks is quite low, it is normal for a network with a reduced number of routers (thus concentrating more the data traffic) to be more sensitive to the traffic types. Since this higher sensitivity indicates transport latencies decreasing more than for other networks, DRAFT is well adapted for applications presenting various types of traffics, even with high data rates.

4.4 Ocean performances and comparison

In this section, the four versions of the OCEAN network are characterized and compared with the same fat-tree as for the comparison of DRAFT. Since all compared networks have thus the same topology, they can be compared using exactly the same data traffic. All versions of OCEAN, including ASIC oriented ones (OCEAN v3.1 and OCEAN v4.1), are first evaluated considering FPGA implementations using Xilinx Virtex V architectures. Then specific results will be presented concerning the implementation in ASIC of these two versions.

4.4.1 Hardware resources consumption

The hardware resources consumption of the four versions of the OCEAN network are presented in Figure 4.8. For comparison purpose, the hardware costs of the conventional fat-tree as well as those of DRAFT and HERMES are also presented in this figure.

From these results, it appears that the four versions of the OCEAN network along with both DRAFT and HERMES have similar resources consumptions between 2 and 32 interconnected PEs. So, even if OCEAN versions are composed of two fat-tree based sub-networks, the conventional fat-tree presents an higher hardware cost. Indeed, in average, the original version of OCEAN (OCEAN v3.0) presents an hardware gain of 48% compared with a conventional fat-tree. DRAFT presents only a gain of 6% of hardware resources compared with OCEAN v3.0. Furthermore, when confronting the various versions of OCEAN, it appears that the version v4.0 with its oblivious routing algorithm allows an average gain of 8% of hardware resources compared with the version v3.0 and its adaptive routing algorithm. The ASIC oriented versions (v3.1 and v4.1), although requiring more registers, take less LUTs and then less CLBs than FPGA oriented ones. Average gains of 5% and 6% are

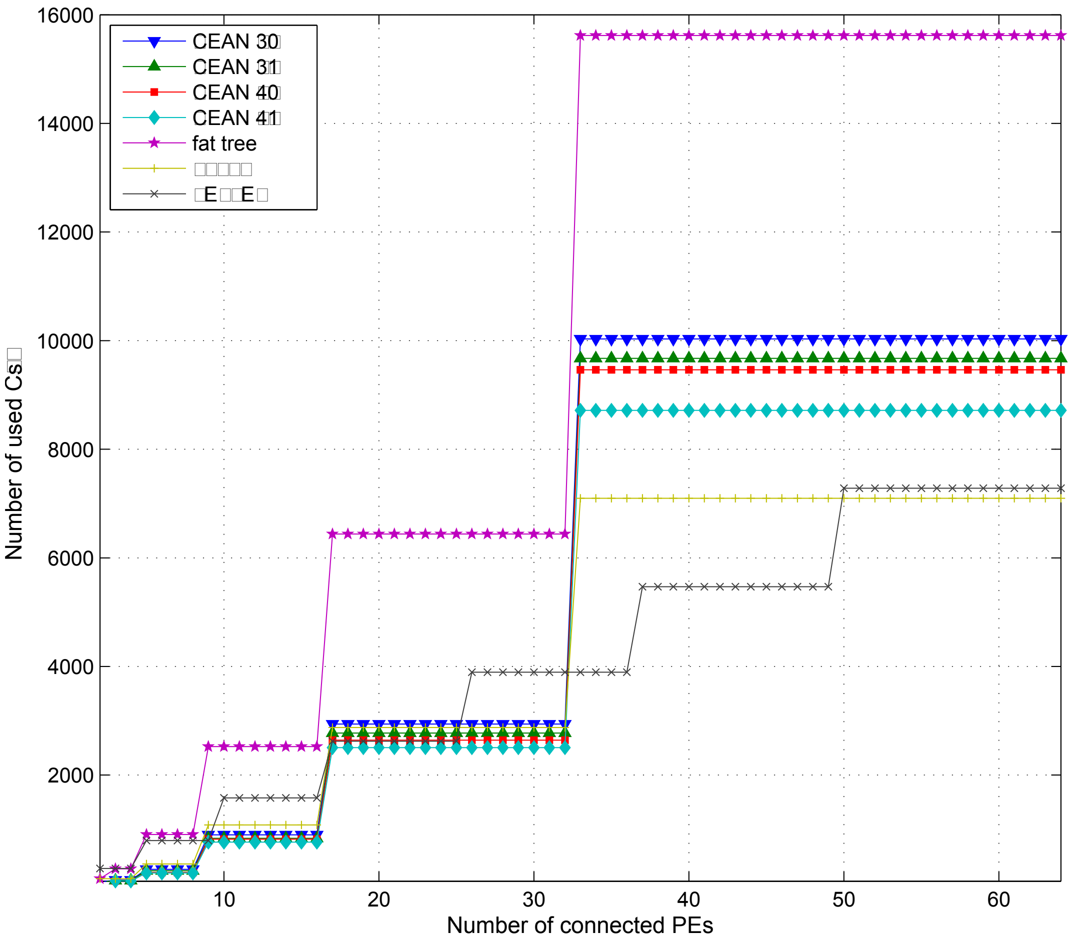


Figure 4.8 : Hardware resources consumption of OCEAN networks compared with the conventional fat-tree, DRAFT, and HERMES NoCs. Results are provided for FPGA implementations targeting the Xilinx Virtex V (XC5VSX50T).

observed respectively between OCEAN v3.0 and v3.1, and between OCEAN v4.0 and v4.1.

All these results are provided in average from 2 up to 64 connected PEs. However, if considering only small regions like between 32 and 64 PEs, gain percentages are quite different. In this specific range, for example DRAFT presents a hardware gain of 14% compared with OCEAN v3.0. So, OCEAN versions are more cost efficient for low numbers of connected PEs (between 2 and 32). However, in this range of 32 to 64 PEs, OCEAN networks present still a much reduced overhead than the fat-tree (18% gain for OCEAN v3.0).

Considering these hardware results, OCEAN networks are very well suited for on-Chip implementations due to their limited hardware overheads. Furthermore, it is very interesting to notice that the improved reliability characteristic from OCEAN v3.0 (and its ASIC oriented version) only implies a 8% overhead compared with version v4.0. This is a major result because thanks to the specific structure of the OCEAN networks, an improved reliability can be implemented with very affordable overheads compared with conventional networks. Table 4.2 presents the maximum operating frequencies of both OCEAN networks and the conventional fat-tree.

Table 4.2 : Maximum operating frequencies (in MHz) measured through ISE 12.4 depending on the size of the networks (number of connected PEs).

Network sizes	8 PEs	16 PEs	32 PEs	64 PEs
	max frequency (MHz)	max frequency (MHz)	max frequency (MHz)	max frequency (MHz)
OCEAN v3.0	196.23	176.30	173.53	173.18
OCEAN v3.1	157.68	155.52	153.36	151.70
OCEAN v4.0	171.17	173.13	170.80	170.45
OCEAN v4.1	178.76	167.67	161.24	152.21
fat-tree	101.15	100.38	100.20	97.75

From these results, the OCEAN networks appear to support much higher operating frequencies than the conventional fat-tree. Indeed maximum frequencies of OCEAN v3.0 are in average 44% higher than those of the fat-tree. Similarly the fat-tree has a maximum frequency 41% lower then OCEAN v4.0. The additional synchronization process, characteristic from ASIC oriented versions of OCEAN, also impacts their maximum frequencies compared with FPGA oriented ones. So, operating frequencies of OCEAN v3.1 are 14% lower than those of OCEAN v3.0. Similarly OCEAN v4.1 has 3% lower operating frequencies than the version v4.0. This difference of operating frequencies will significantly impact the overall network performances such as latencies and data rates.

4.4.2 Network performances

As presented in Section 3.12, every network is tested with exactly the same Permutation data traffic.

Average latencies

The main performances criteria used to characterize NoCs is the transport latency. Results are presented in Figures 4.9 and 4.10. All these results are presented in μs taking into account the maximum operating frequencies obtained from FPGA implementations. Four sizes of networks are studied: 8, 16, 32, and 64 connected PEs. Results are presented depending on the standardized injection rate.

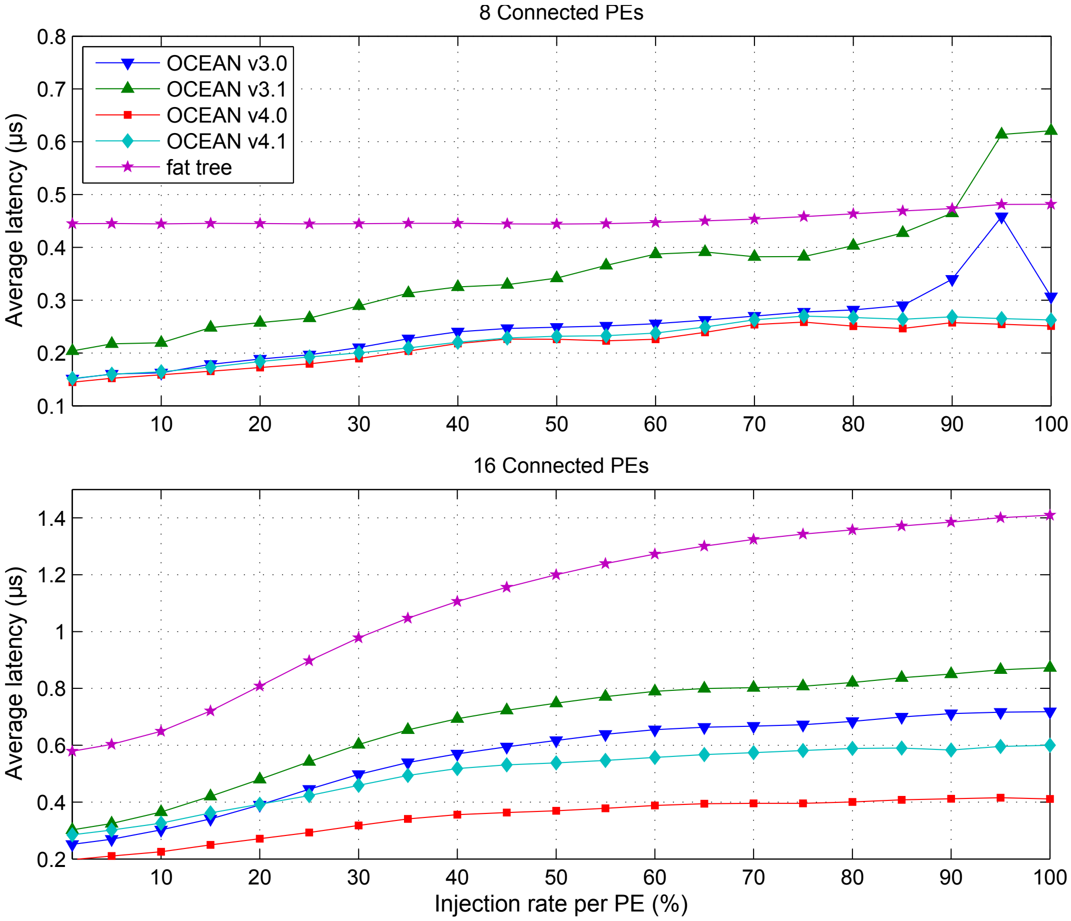


Figure 4.9 : Average latencies expressed in μs depending on the injection rate for 8 and 16 connected PEs. These results take into account the different operating frequencies of the networks.

The main thing that may seem strange is that latencies do not explode when the networks saturate. This is due to the dating of the packets that is done on the fly. So it is interesting to see that the latencies of transmitted packets reach some "kind" of limit values specifically for each network. Concerning the results themselves, measured latencies of OCEAN versions are inferior to those of the fat-tree. Indeed, considering results obtained for 8 PEs, OCEAN v3.0 presents a 45% average gain of transport latencies compared with the fat-tree. Furthermore the version v4.0 of OCEAN has a gain of 52% compared with the same fat-tree. However, if the gain of OCEAN v4.0 remains relatively the same for 16, 32, and 64 connected PEs (respectively 60%, 63%, and 62% gains), the behaviour of the version v3.0 is different. Indeed the performance gain of this version compared with the fat-tree decreases when networks scale up. This leads for 64 connected PEs to the reverse

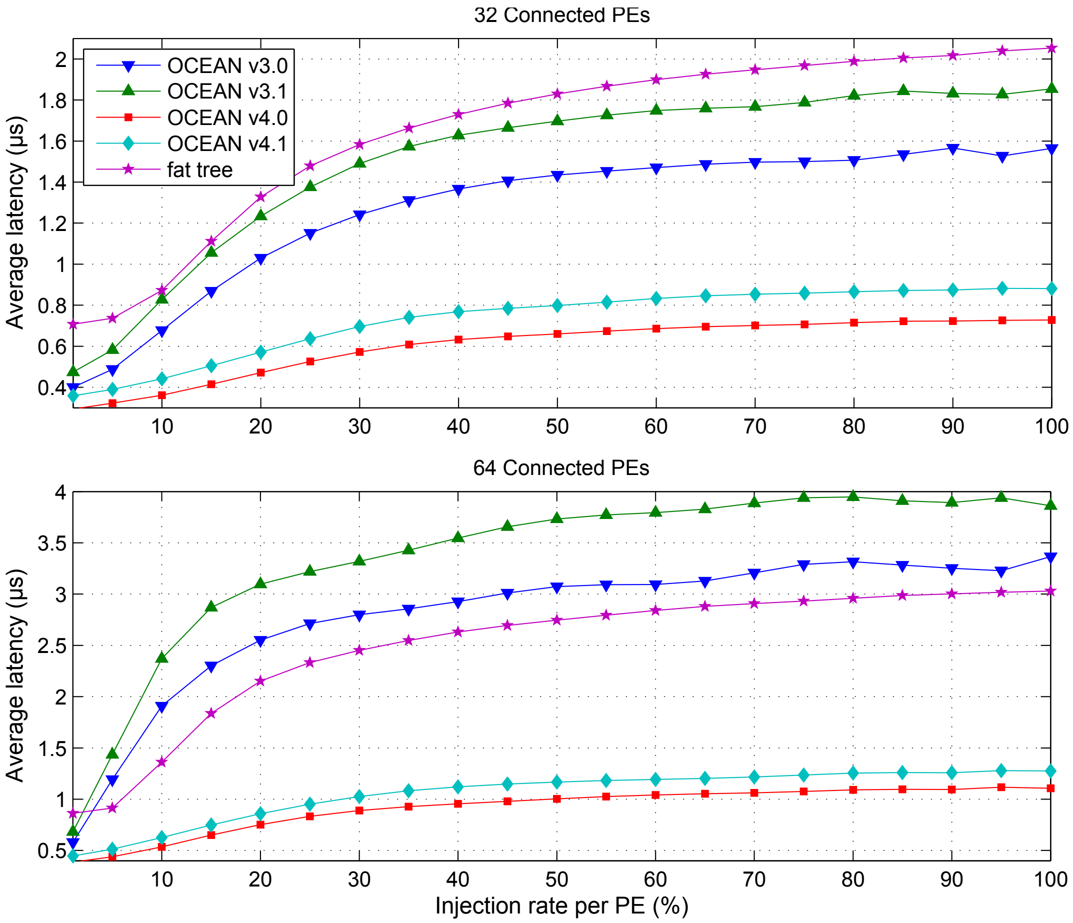


Figure 4.10 : Average transport latencies expressed in μs depending on the injection rate for 32 and 64 connected PEs.

situation where the fat-tree provides 9% lower latencies than OCEAN v3.0.

When comparing OCEAN v3.0 and v4.0, it appears that the difference between them scales up along with their scale. Indeed, if the version v4.0 presents a 11% gain of performances compared with the v3.0 for 8 connected PEs, the difference increases reaching 65% for 64 connected PEs. Thus, it is interesting to notice that if an improved quality of service (fault tolerance) has a low hardware overhead, it impacts performances increasing average latencies. However, from these comparisons, OCEAN networks and moreover the versions with an oblivious routing algorithm significantly outperform the conventional fat-tree. This constitutes a major result for the OCEAN networks: they both use less hardware resources and provide lower average latencies than a conventional packet switched fat-tree.

The difference between ASIC and FPGA oriented versions of OCEAN can be clearly observed for large scale networks: every ASIC oriented version presents higher average latencies than FPGA oriented ones. However, the difference between these

ASIC oriented versions and the networks from which they are inherited remains constant whatever is the size of the networks. So, OCEAN v3.0 presents in average 18% gain compared with the v3.1, while between versions v4.0 and v4.1, the gain is of 16% for the v4.0. However, versions vx.1 do not target FPGA implementations. The ASIC implementation of these networks allows them to operate at higher frequencies. Further details are provided in following sections.

Data rates

Usually throughputs are given as an indicator of network performances. Throughput corresponds in fact to the data traffic transferred by a network (also called accepted traffic) normalized in percentage. Since throughputs are normalized, they do not depend on operating frequencies. Regarding networks like DRAFT, HERMES, or the conventional fat-tree, since their operating frequencies are pretty similar, throughputs are the exact reflections of accepted data rates. However, when comparing OCEAN networks with the conventional fat-tree, significant differences can be observed between throughput and accepted data rates. This is due to the operating frequencies that are very different: for example the maximum frequency of the fat-tree is 44% lower than these measured for OCEAN v3.0. This is why, for fair comparison purpose we do not compare throughputs but accepted data rates (in MByte/s). Corresponding results are presented in Figures 4.11 and 4.12.

From these results, OCEAN v4.x networks accept much higher data rates than the conventional fat-tree. Indeed, 65% and 60% average gains were measured between the v4.x versions and the fat-tree for respectively 8 and 16 connected PEs. For 32 and 64 PEs, the performance gains are a little bit lower (48 and 41%) but these versions are very efficient and scalable. Indeed they substantially reduce the hardware costs while offering lower latencies and higher accepted data rates than a packet switched fat-tree even for 64 connected PEs. This is a major contribution of this PhD answering initial motivations.

Concerning OCEAN v3.x and their adaptive routing algorithm, it appears that accepted data rates become lower than for the fat-tree when the networks scales up. Indeed, if version v3.0 presented a gain of 62% concerning data rates when compared with the fat-tree, this gain decreases to 41% and 7% respectively for 16 and 32 connected PEs. Furthermore, for 64 connected PEs, the conventional fat-tree outperforms the versions v3.x from a gain factor of 27%. This is an interesting result concerning the impact of an improved quality of service: when more and more PEs are interconnected, the number of tested control routes scales up leading to a decrease of overall performances. However, even if accepted data rates become lower than fat-tree's for 64 connected PEs, OCEAN v3.x remain very interesting due to

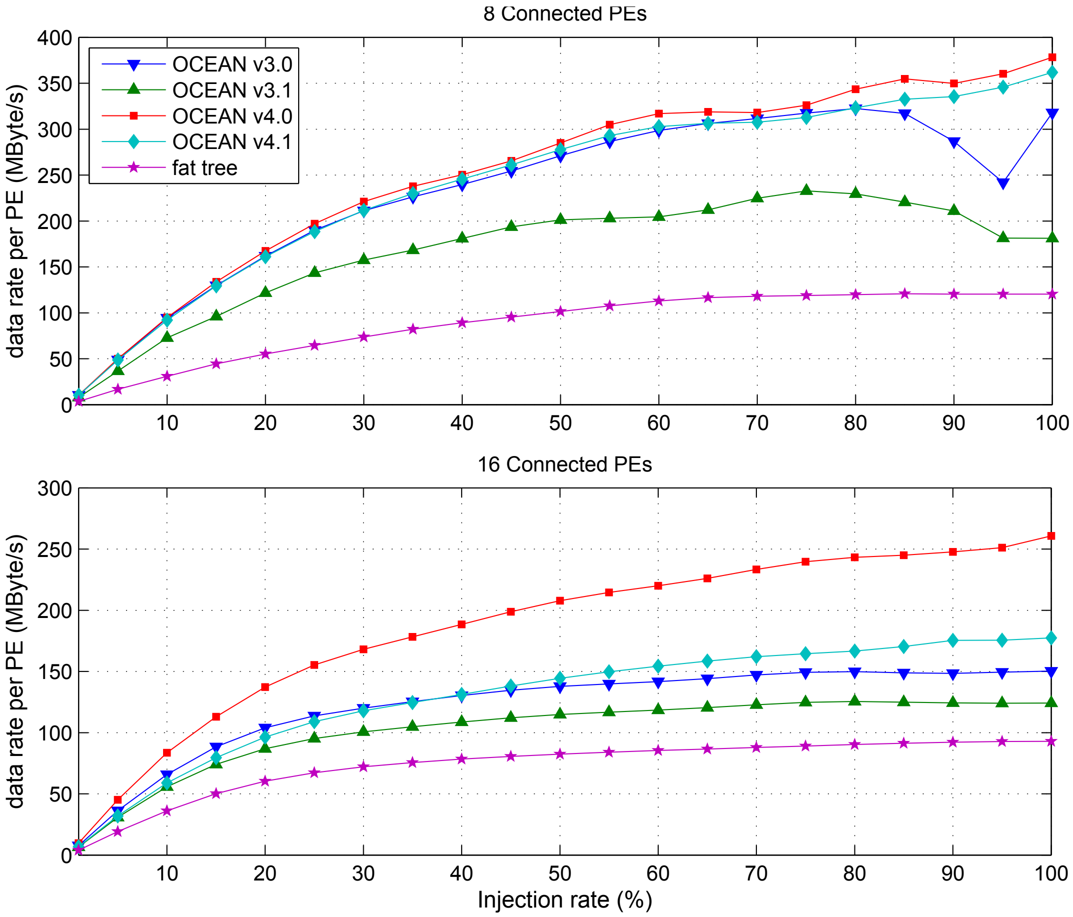


Figure 4.11 : Average accepted data rates per PE expressed in Mbytes/s depending on offered injection rate. This figure presents results obtained for the 8 and 16 connected PEs versions of the networks.

their reduced hardware costs, their low latencies, and supported QoS.

Concerning the saturation zones, despite of their different routing algorithms, their different switching protocols and the absence of any buffer in OCEAN versions, every network saturates at the same offered traffics. This denotes a similar behaviour from all these networks which is due to the fat-tree topology.

Maximum accepted data rates are summarized in Table 4.3.

From these results, the significantly improved scalability of OCEAN v4.0 is demonstrated. This network is very well adapted for on-Chip implementations, including ASICs with its v4.1 variation. These two versions outperform conventional packet switched fat-tree considering all performance metrics (hardware cost, latencies, data rates). Furthermore, even if the conventional fat-tree accepts higher data rates than OCEAN v3.x, these latters are also very well adapted to be implemented inside SoCs and RSoCs. Thanks to their specific architectures, OCEAN v3.x

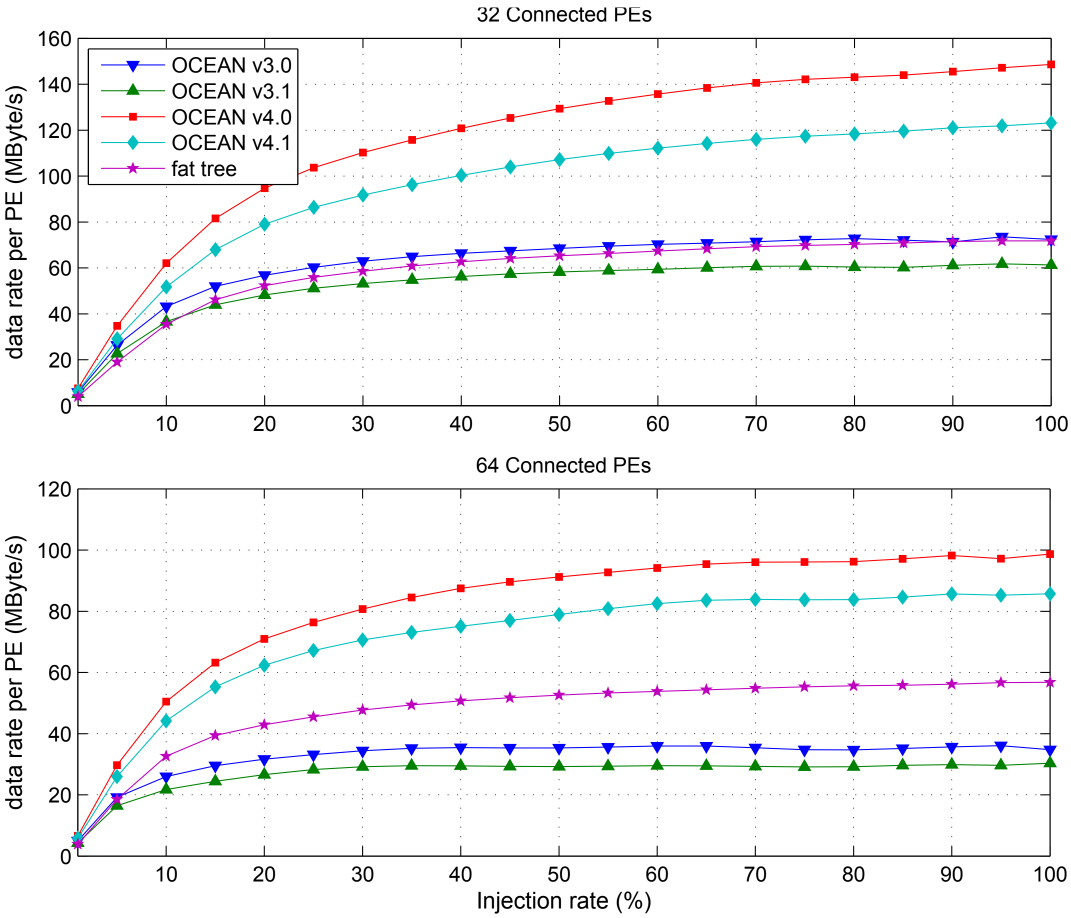


Figure 4.12 : Average accepted data rates for 32 and 64 connected PEs depending on offered injection rate.

constitutes an affordable way to improve the quality of service of a communication architecture with very low hardware overheads, even when compared with OCEAN v4.x.

The fault tolerance that improves the QoS of OCEAN v3.0 and OCEAN v3.1 was not characterized in terms of performances when one or several switches become faulty. However, excepted if one switch on which PEs are directly connected become faulty, every couple source/destination can communicate with one faulty switch (or link). Performances would be impacted due to a restriction of available data paths, but communications can still happen. The maximum number of bearable faulty switches depends both on the size of the network and on where they are located into the structure of the networks. The more switches are high in the structure of OCEAN v3.x, the more they are allowed to become faulty without isolating PEs. While there is at least one data path reaching every connected PEs, communication can occur in OCEAN v3.x.

Table 4.3 : Maximum accepted data rates per connected PE (expressed in MBytes/s) depending on the size of the networks.

Network sizes	8 PEs	16 PEs	32 PEs	64 PEs
	data rate (MB/s)	data rate (MB/s)	data rate (MB/s)	data rate (MB/s)
OCEAN v3.0	326,06	150,30	73,45	36,34
OCEAN v3.1	239,41	125,71	61,71	30,33
OCEAN v4.0	378,22	260,70	148,64	98,70
OCEAN v4.1	361,94	177,39	123,16	85,92
fat-tree	120,42	92,97	71.81	56.78

4.4.3 FPGA validation

ISE design tools provide maximum operating frequencies considering the sequential logics like registers that are embedded inside the NoC. However, the data subnetwork do not includes such registers but only combinatorial logics. this is why a test application was implemented in a Xilinx Virtex V FPGA (ML506 development platform). In fact this application is directly inherited from the platform that was used for characterization and comparison purpose (see Section 3.12). The difference between this platform and the test application lies in both its hardware implementation (instead of a simulation at the RTL level), and in the role of generic PEs. Instead of using informations included in the flits to calculate performances, generic PEs read the first flit and verify that following one are well incremented. If a received flit do not corresponds to the value of the previous one plus ”1”, then an error signal is turned on.

Similarly to the test platform, a permutation based traffic is injected in the NoC. However very much larger packets of data were also injected in addition to small ones (until 1024 flits). Since PEs could not be clocked at higher frequency than 282 MHz, transmissions over the data network were realized at this frequency. For these experimentations, the control sub-network was clocked at 125 MHz. In order to verify the good behaviour of the application, some transmission errors can be injected on demand. All these errors injected on purpose were well detected. Then, with this simple application, data were injected successively in the four OCEAN versions (without user generated errors) during at least a dozen of minutes and sometimes even merely an hour. This way, during these one hour experimentations representing more than 40 billion communications, no error was detected leading to 0 % transmission errors.

In all this section concerning network performances, only one to one communications where created. So, the extra functionalities supported by OCEAN networks, such as multicast or communication channels handling several communications, were not presented. Indeed these functionalities can highly improve performances depending on the application requirements. For example, latencies can be reduced to 1 clock cycle thanks to already opened communication channels. Indeed the time of flight is then reduced to only the propagation delays on the data sub-networks. All the Handshake phase for control purpose is realized only once. Using already opened communication channels allow PEs to inject data at full speed which corresponds in FPGA with an injection frequency of 282 MHz (the maximum we could validate) to 1.1280 GB/s.

Furthermore, accepted data rates can also be significantly increased thanks to the compliance of the networks with the multicast. However, the use of this mechanism is intrinsically linked with the real life applications in which the networks are implemented. Furthermore, no comparison could be done since the fat-tree, DRAFT, or HERMES networks do not support multicast. Further characterizations concerning the impact of these mechanisms are expected from future applications embedding an OCEAN network.

4.4.4 ASIC implementation

OCEAN v3.1 and v4.1 have an internal structures adapted for ASIC implementation. These two networks could be synthesized and validated at the gate level. Unfortunately neither implementation on silicon could be realized nor simulation at the transistor level. However, in this section, results and first characterizations at the gate level are presented. For practical reasons, syntheses were realized through Synopsys DesignVision 2007.03 using the 130nm CMOS technology from STMicroelectronics.

The architecture of both an out-controller and an in-controller from OCEAN v4.1 are presented in Figure 4.13. Remaining hardware block constituting a full network are presented in Appendix D.

Concerning performances, results provided by both the synthesis tool and simulations at the gate level through Modelsim 6.6c are summarized in Tables 4.4 and 4.5. The Ports column represents the number of input/output wires constituting the interface of the OCEAN networks. The Nets column is to signify the number of wires used inside the network. The Area column indicates the total area taken by logical gates. The "F_{max} ctrl" column indicates the maximum operating frequency that can be taken by the control part of the networks. These values were obtained from simulation at the gate level with modelsim with a resolution of 1ps. Obviously, results take into account all the timing informations provided by both STMicroelectronics and Synopsys. These frequencies guarantee that the networks operate with 0% error. The control delay column indicates the maximum number of clock cycles a request can take to reach its destination (if there is no conflict with another existing communication). The data delay column indicates the propagation delay of a data using the longest path of the network (worst case). Just like for "F_{max} ctrl", the "F_{max} data" column indicates the maximum frequency on which data can be transmitted through the networks. These frequencies guarantee that data reach their destination and are perfectly stable in one clock cycle. Indeed data reach their destination before half of the clock period, but this allows to guarantee the stability of these data when they are read. Higher frequencies can be used by designers (until two times indicated F_{max} data) counting on low skews concerning the clocks of the PEs, and low parasitic effects. Hence, the use of such frequencies is at designers' own risks and are not ensured to lead to 0% error communications.

Table 4.4 : Synthesis results at the gate level concerning OCEAN v3.1 and OCEAN v4.1.

	Ports	Nets	Area (mm^2)
OCEAN v3.1 8 PEs	706	1844	0.227
OCEAN v3.1 16 PEs	1666	6068	1.019
OCEAN v3.1 32 PEs	4354	20228	3.959
OCEAN v3.1 64 PEs	12802	72004	16.151
OCEAN v4.1 8 PEs	706	1828	0.219
OCEAN v4.1 16 PEs	1666	6020	0.851
OCEAN v4.1 32 PEs	4354	20100	3.310
OCEAN v4.1 64 PEs	12802	71684	13.515

As it could be expected from FPGA implementations, OCEAN v4.1 uses fewer nets and fewer gates than OCEAN v3.1. However, the overhead of OCEAN v3.1 concerning both area and power is quite reduced. This is particularly interesting considering the improved QoS (fault tolerance) of this network. Concerning network performances measured in simulation at the gate level, very high data rates from 800 MB/s for OCEAN v3.1 (32 bits data width; 64 PEs) to 2.5 GB/s for OCEAN v4.1 (32 bits data width; 8 PEs) are supported per PE also with very low latencies (from 0.8 ns to 2.5 ns times of flight). From simulations, it is interesting to notice that thanks to appropriate timing constraints at synthesis time, there is always less than 1ps delay between the various wires constituting a signal. So, transitory states are

Table 4.5 : Simulation results at the gate level concerning OCEAN v3.1 and OCEAN v4.1.

	F_{max} ctrl (MHz)	Control delay (clk cycle)	F_{max} data (MHz)	Data delay (ps)
OCEAN v3.1 8 PEs	416.67	9	500	1000
OCEAN v3.1 16 PEs	416.67	13	333	1500
OCEAN v3.1 32 PEs	416.67	24	250	2000
OCEAN v3.1 64 PEs	416.67	39	200	2500
OCEAN v4.1 8 PEs	454.54	9	625	800
OCEAN v4.1 16 PEs	454.54	13	417	1200
OCEAN v4.1 32 PEs	416.67	24	313	1600
OCEAN v4.1 64 PEs	416.67	39	250	2000

limited to less than 1ps. This is important when designing a complete application aiming ASIC implementation.

From these results, OCEAN v4.1 and OCEAN v3.1 are flexible network providing very high level performances with a reduced hardware cost to ASIC implemented SoCs. Due to their support of multicast, communication channels, and improved fault tolerance for OCEAN v3.1, new high performances applications requiring flexibility from the communication architecture can be foreseen. This is also a major contribution of this PhD.

4.5 Synthesis of the chapter

In this chapter, a characterization and comparison of all created networks was made. From the comparisons, DRAFT appeared to be a viable way to reduce significantly the hardware resources consumption of fat-tree networks. Sometime DRAFT reveals itself to be as advantageous as a mesh topology based network. Concerning network performances, under the assumption that there is no data traffic between the top level connected PEs, DRAFT provides high levels of performances. Both hardware consumption and network performances make DRAFT a viable solution for real life applications using DPR in current FPGAs.

In order to design a network with no restriction concerning the data traffic and with extra functionalities like multicast compliance, the OCEAN networks were evaluated. The OCEAN networks appeared as being efficient solutions to reduce the hardware cost of a fat-tree topology (thus increasing the scalability) while proposing very high levels of performances. OCEAN v3.0 proposes an improved QoS and is adapted for FPGA implementations. OCEAN v3.1 is the same network but adapted for ASIC implementations. OCEAN v4.0 is a simplified version of OCEAN v3.0 with an oblivious routing algorithm allowing higher level performances (hardware consumption and network performances). Finally, OCEAN v4.1 is the adaptation of OCEAN v4.0 targeting ASIC implementations. Characterizations of these networks were realized both in FPGA and in ASIC (at the gate level). These networks are particularly interesting for applications requiring very high level performances, and where data traffics can not be predicted. The use of communication channels or the multicast can even significantly improve offered performances.

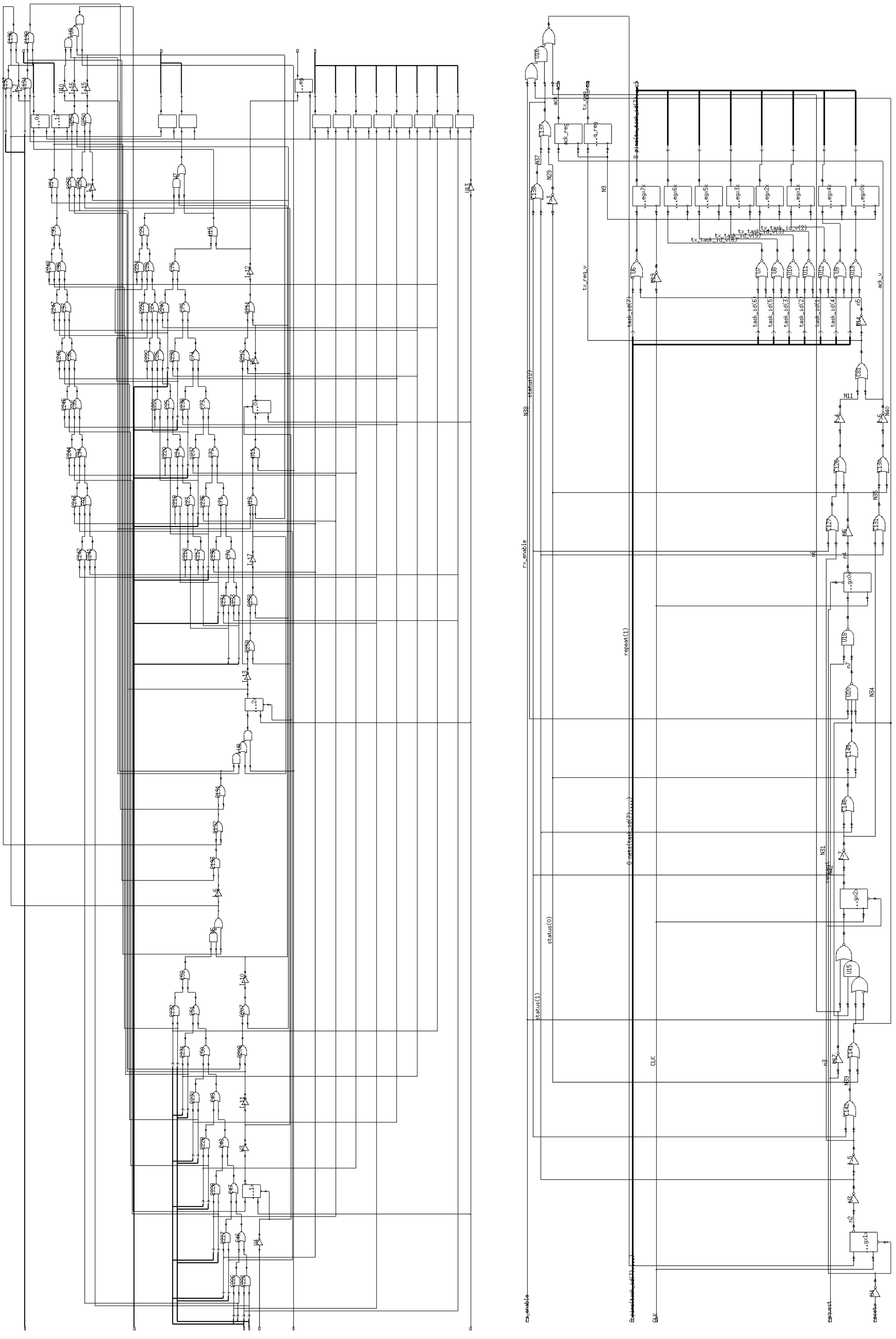


Figure 4.13 : Presentation of an In-controller (left) and an Out-controller (right) of the OCEAN v4.1 network at the gate level (90ř rotated.

Conclusion and perspectives

The first objective of this PhD was to propose an innovative interconnection architecture in order, into the scope of the FOSFOR project, to provide an efficient solution for communications in a DPR environment. For this purpose, the DRAFT network was proposed. DRAFT is based on a fat-tree topology but it has the particularity that some of the PEs are directly connected to the top level router. Indeed, DRAFT takes advantage of the existing SoCs and more precisely RSoCs where some elements never share data directly. This is for example the case for two shared memories or two communication interfaces. These sample elements are always accessed by computing ones like microprocessors or IP blocks, either for read and write operation. When interconnected by a network, this results in several data paths that are never used and can thus be simplified. Constraining the placement of these elements to the top of DRAFT allows to save one scale factor compared with a traditional fat-tree while providing high level performances. The flexibility of DRAFT is inherited from fat-tree's because of both the indirect nature of the network (avoiding PEs to be located inside its structure), and the bandwidth offered to the PEs that is constant all over the network. The internal architecture of DRAFT is more common than the topology. DRAFT is based on a packet switching politic with a routing algorithm inherited from a classic TurnBack but adapted to the new topology. When compared with a traditional fat-tree also with the mesh based network HERMES, DRAFT reveals itself to present a hardware cost similar to mesh's at least until 64 simultaneously connected PEs. Concerning network performances, DRAFT provides similar or just higher performances than HERMES. At fixed data rate concerning the offered traffic, DRAFT gives also better results than the traditional fat-tree. However, this latter reveals its strength concerning the maximum accepted data rate, notably when the number of interconnected PEs increases. Since the performances of a traditional fat-tree are so at the cost of a significant hardware cost, DRAFT appeared to be an interesting trade off between costs, performances, and flexibility. Considering its characteristics, DRAFT fits the expectations from the FOSFOR project. Furthermore, the integration of DRAFT in the FOSFOR demonstrator was significantly eased by the design of a bridge making the relation between an interface of DRAFT, and the standardized and well known interface of an AHB bus. This bridge uses both master and slave controllers in order to respectively send and receive data from an AHB connected element. The DRAFT-AHB bridge is also compliant with transfers of bursts of data.

The second objective of this PhD was to ease the parameterization, the generation, and the performance evaluation of the DRAFT network. For this purpose the DRAGOON environment was inherited from ATLAS. DRAGOON does not answer completely to this objective since it does not generate the DRAFT network but a traditional fat-tree. However, the DRAFT network can be obtained easily from generated fat-tree changing two VHDL files. However, even if this transformation should be realized manually, DRAGOON allows to simulate both fat-tree and DRAFT networks. The generation of DRAFT is then possible with user control over notably the number of interconnected PEs, the width of exchanged data, and the depth of the buffers. The perspectives for DRAGOON and so for DRAFT are double. Indeed we aim to improve DRAGOON in order to directly generate DRAFT without any human intervention, and we also would like to merge DRAGOON with the ATLAS environment from which it is inherited. Doing so, the resulting environment would offer more choice concerning the network topologies to the designers. Hence, they should be able to choose the one that best fits their application. We deeply would like this to happen shortly after the end of this PhD. However, for this purpose DRAFT is already part of the FPL benchmark.

The third objective was to propose a communication service, dissociated from the FOSFOR OS, acting as an overlayer to the DRAFT network. This communication service was also proposed providing new features that are allowed by the DPR ability of the FPGAs. The proposed service is based on virtual communication channels. When a task, either software or hardware, requires to communicate through its dedicated PE, it request the communication service to open a channel. When the other PE it wants to communicate with also opens the channel, the service provides the physical addresses to both PEs so that the communication can start. However, in the case that the destination PE is not allocated on the platform, or not ready to communicate, the service offers the possibility to store temporarily the data in a shared memory if there is enough free place to do so. When the destination PE is allocated and requests to open the communication channel, the service makes the relation between the PE and a shared memory. This way, the storage of the data in shared memory is completely transparent for the PEs. Furthermore, if not enough free place could be found in interconnected memory, the service offers the possibility to use the storage logics embedded in free PRRs allocating a temporary memory PE in this PRR. This PE acts then as a shared memory, thus increasing the memory space of the platform. Obviously it is removed once data have been collected by the genuine receiver PE.

Finally, the last but not least objective from this PhD was to propose a very high performance NoC that does not impose any constraints over the PEs. For example this network was expected not to induce placement constraints like for the top level connected PEs of the DRAFT network that can not share data with each others. Since the scalability was also a key feature for the aimed NoC, the R2NoC network was proposed. The idea behind R2NoC was to perform a circuit switching of the data dynamically reconfiguring the routers of a fat-tree. This way, the size of the router was expected to be reduced, and thus the scalability improved. Respecting technological issues imposed by present technology, this network is feasible but the storage of all needed configuration partial bitstreams also with the very important latency required for reconfiguration purpose make R2NoC impractical for industrial applications. This is why the OCEAN network was designed. OCEAN is composed of two fat-tree based sub-networks. One is dedicated to data while the other is for control purpose reconfiguring the first one. Indeed, the data network circuit switches the data through direct and minimal paths that are dynamically reconfigured by the control network. In order to avoid the drawback of R2NoC, OCEAN performs its own dynamic reconfiguration: configuration bits are generated online by the control routers and forwarded to corresponding data routers. In these latters, only combinatorial logics are implemented allowing data to come from a port to another one depending on the configuration bits. So, transfers of data on the data network are realized asynchronously while the control is performed synchronously. This allows a real quality of service thanks to the synchronous control while providing very high performances due to the asynchronous data transfers. OCEAN was not only simulated but implemented and validated in FPGA, and also at the gate level aiming ASIC implementations. Resulting performances are really promising because performances are significantly better than those of a traditional packet switched fat-tree with a far lower hardware cost. Furthermore, the version 3 of the OCEAN network was also proposed embedding adaptive routing algorithms that improves the reliability of the network. Indeed, if for some reason a router becomes faulty, the routing algorithms automatically searches for other data paths to reach the destination(s). However this improved quality of service is provided at the cost of network performances that are similar or even a little bit lower than those of the traditional fat-tree. It is also important to notice that every version of OCEAN is compliant with multicast which is particularly interesting for data flow based applications. OCEAN answers completely to the objective of the PhD due to its high level network performances, its scalability merely similar to these of a mesh network, its flexibility considering DPR environments, and its efficiency considering the data transfers on which no constraint nor protocol is imposed. Designers are then free to choose the protocol that best fits their application, for example performing DMA transfers.

Considering the perspectives to this research work, in addition to the improvement of the DRAGOON environment and its integration inside ATLAS, it should be of high interest to propose, like for DRAFT, a communication service able to exploit all the possibilities offered by the OCEAN network. The multicast combined with the non-closure of the data paths between two communications from the same couple source/destination PEs should be particularly interesting to improve communications. Furthermore, the high flexibility concerning the data transfers should be exploited to improve the performances of applications. We also think that a parameterization and generation environment dedicated to OCEAN should be particularly welcome in order to ease its integration inside industrial products. In addition with all of this, the evaluation of the fault tolerance in real life situations of OCEAN v3.x should be particularly interesting. More generally, it should of a very high interest to implement the various versions of OCEAN in a real RSoC with multiple processors in order use them in real environments.

Finally, the problem of the characterization environments has been a constant brake to the research process. Indeed it is very difficult to compare different networks because most of them have a specialized simulator which is not compliant with other networks. Furthermore, present generic simulators are not adapted to the simulation of a large range of NoC with different topologies. This is why it should be interesting to create a standardized on-Chip platform on which NoCs could be implemented independently from their structure and protocols. Performances should then not be estimated but in situ measured. This way, the platform should not be of interest only inside the research community, but also for industrials. Indeed they could verify the real performances that can be extracted from their communication architectures reproducing the behaviour of their application. The first specifications for such a platform were proposed in this PhD (see Appendix B). We think it would deserve both improvements and manpower to become a reality.

Bibliographic references

[1] Achronix (S. C.). *Speedster FPGA Family*, DS001 Rev. 1.2 - October 16, 2009. (referenced page 5)

[2] Agarwal (A.), Iskander (C.) and Shankar (R.), « Survey of network on chip (NoC) architectures & contributions », *Journal of Engineering, Computing and Architecture*, **3**, 2009, p. 21–27. (referenced page 33)

[3] Ahmadinia (A.), Bobda (C.), Ding (J.), Majer (M.) and Teich (J.), « A Practical Approach for Circuit Routing on Dynamic Reconfigurable Devices », *Workshop on Rapid System Prototyping.*, 2005, p. 84–90. (referenced page 31)

[4] Ahmadinia (A.), Bobda (C.), Bednara (M.) and Teich (J.), « A New Approach for On-line Placement on Reconfigurable Devices », *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 04)*, 2004, p. 134–140. (referenced page 16)

[5] Ahmadinia (A.) and Teich (J.), « Speeding up Online Placement for XILINX FPGAs by Reducing Configuration Overhead », *Proceedings of the IFIP International Conference on VLSI-SOC, Darmstadt, Germany*, 2003, p. 118–122. (referenced page 16)

[6] Al-Hashimi (M. G. B.), Laxmi (V.), Navaneeth (R.), Choudhary (N.), Jain (L.), Ahmed (M.), Paliwal (K.), Varsha, Rekha and Vineetha. *NIRGAM: A Simulator for NoC Interconnect Routing and Applications' Modeling*. EPSRC (UK) grant EP/C512804/1, 2010. (referenced page 117)

[7] Ali (M.), Welzl (M.) and Hellebrand (S.). « A dynamic routing mechanism for network on chip ». In *23rd Nordic Microelectronics event (NORCHIP) Conference*, p. 70–73. IEEE, 2005. (referenced page 34)

[8] Altera. *Stratix IV Dynamic Reconfiguration*, 2005. (referenced page 6)

[9] Altera. *Stratix IV Device Handbook - Volume 1&2, ver 4.0, Nov 2009.*, 11 2009. (referenced page 6)

[10] Altera. « Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs (WP-01137-1.0) ». Technical report, Altera corporation, 2010. (referenced page 6)

[11] Andriahantenaina (A.) and Greiner (A.). « Micro-network for soc: Implementation of a 32-port spin network ». In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, p. 11128. IEEE Computer Society, 2003. (referenced page 33)

[12] Angiolini (F.), Meloni (P.), Carta (S.), Benini (L.) and Raffo (L.), « Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness », *Proceedings of the conference on Design,Automation and Test in Europe : Proceeding*, **1**, 2006, p. 124–129. (referenced page 27)

[13] ANJO (K.), OKAMURA (A.), KAJIWARA (T.), MIZUSHIMA (N.), OMORI (M.) and KURODA (Y.). « NECoBus: A high-end SOC bus with a portable & low-latency wrapper-based interface mechanism ». In *IEEE 2002 custom integrated circuits conference*, 2002. (referenced page 27)

[14] ATMEL. *AT40K05/10/20/40AL. 5K - 50K Gate FPGA with DSP Optimized Core Cell and Distributed FreeRam, Enhanced Performance Improvement and Bi-directional I/Os (3.3 V).*, 2006. revision F. (referenced page 5)

[15] Baklouti (M.), Aydi (Y.), Marquet (P.), Dekeyser (J.) and Abid (M.), « Scalable mpNoC for massively parallel systems - Design and implementation on FPGA », *Journal of Systems Architecture*, **56**(7), 2010, p. 278 – 292. Special Issue on HW/SW Co-Design: Systems and Networks on Chip. (referenced page 33)

[16] Bartels (C.), Huisken (J.), Goossens (K.), Groeneveld (P.) and van Meerbergen (J.), « Comparison of An Æthereal Network on Chip and A Traditional Interconnect for A Multi-Processor DVB-T System on Chip », *Proceeding, IFIP Conference on Very Large Scale Integration (VLSI-SoC)*, 2006. (referenced page 29)

[17] Bazargan (K.), ICastner (R.) and Sawafzadeh (M.), « Fast Template Placement for Reconfigurable Computing Systems », *Design & Test of Computers, IEEE*, **17**, 2000, p. 68–83. (referenced page 16)

[18] Becker (J.), Donlin (A.) and Huebner (M.). « New tool support and architectures in adaptive reconfigurable computing ». In *Very Large Scale Integration, 2007. VLSI-SoC 2007. IFIP International Conference on*, p. 134–139. IEEE, 2007. (referenced page 16)

[19] Becker (J.), Hubner (M.), Hettich (G.), Constapel (R.), Eisenmann (J.) and Luka (J.), « Dynamic and Partial FPGA Exploitation », *Proceedings of the IEEE*, **95**, 2007, p. 438–452. (referenced page 13)

[20] Becker (J.), Pionteck (T.) and Glesner (M.), « Adaptive Systems-on-Chip: Architectures, Technologies and Applications », *Proceedings of the 14th Symposium on Integrated Circuits and Systems Design*, 2001, p. 2. (referenced page 13)

[21] Benini (L.) and Micheli (G. D.), « Networks on Chips: A New SoC Paradigm », *Computer*, **35**, 2002, p. 70–78. (referenced page 28)

[22] Bertozzi (D.), Kumar (S.) and Palesi (M.), *VLSI Design: Networks-on-Chip*. Hindawi Publishing Corporation, 2007. (referenced page 28)

[23] Bjerregaard (T.) and Mahadevan (S.), « A Survey of Research and Practices of Network-on-chip », *ACM Computing Surveys (CSUR)*, **38**, 2006, p. 1–51. (referenced page 28)

[24] Bobda (C.) and Ahmadinia (A.), « Dynamic Interconnection of Reconfigurable Modules on Reconfigurable Devices », *Design & Test of Computers*, **22**(5), 2005, p. 443–451. (referenced page 31)

[25] Bobda (C.), Ahmadinia (A.), Majer (M.), Teich (J.), Fekete (S.) and van der Veen (J.), « DYNOC: a dynamic infrastructure for communication in dynamically reconfigurable devices », *Field Programmable Logic and Applications.*, 2005. (referenced page 29)

[26] Bobda (C.), Ahmadinia (A.), Rajesham (K.), Majer (M.) and Niyonkuru (A.), « Partial Configuration Design and Implementation Challenges on Xilinx Virtex FPGAs », *18th International Conference on Architecture of Computing Systems, ARCS*, 2005, p. 61–66. (referenced page 13)

[27] Bouhraoua (A.) and Elrabaa (M.), « A High-Throughput Network-on-Chip Architecture for Systems-on-Chip Interconnect », *International Symposium on System-on-Chip*, 2006, p. 1–4. (referenced page 33)

[28] Braun (L.), Gohringer (D.), Perschke (T.), Schatz (V.), Hubner (M.) and Becker (J.), « Adaptive real-time image processing exploiting two dimensional reconfigurable architecture », *Journal of Real-Time Image Processing*, **4**, 2009, p. 109–125. (referenced page 13)

[29] Chou (S.), Chen (C.), Wen (C.), Chen (T.) and Lin (T.), « Hierarchical circuit-switched NoC for multicore video processing », *Microprocessors and Microsystems*, **35**, 2011, p. 182–199. (referenced page 31)

[30] Compton (K.), Li (Z.), Cooley (J.), Knol (S.) and Hauck (S.), « Configuration Relocation and Defragmentation for Run-Time Reconfigurable Computing », *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, **10**(3), 2002, p. 209–220. (referenced page 18)

[31] Cozzi (D.), Farè (C.), Meroni (A.), Rana (V.), Santambrogio (M.) and Sciuto (D.), « Reconfigurable NoC design flow for multiple applications run-time mapping on FPGA devices », *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, 2009, p. 421–424. (referenced page 38)

[32] Dally (W. J.) and Towles (B.), « Route Packets, Not Wires: On-Chip Interconnection Networks », *Design Automation Conference (DAC)*, 2001, p. 684–689. (referenced page 28)

[33] Dally (W. J.) and Towles (B.), *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004. (referenced pages 5, 18, 28, 32, 34, 49 and 87)

[34] Dally (W.), « Performance analysis of k-ary n-cube interconnection networks », *IEEE Transactions on Computers*, **39**, 1990, p. 775–785. (referenced page 29)

[35] Delorme (J.), Nafkha (A.), Leray (P.) and Moy (C.), « New OPBHW-ICAP interface for realtime Partial reconfiguration of FPGA », *International Conference on Reconfigurable Computing and FPGAs*, 2009, p. 386–391. (referenced pages 16, 86 and 174)

[36] EETimes. « Altera to offer partial reconfiguration at 28-nm ». `http://www.eetimes.com/showArticle.jhtml?articleID=222600544`. (referenced page 6)

[37] Evain (S.), Diguet (J.) and Houzet (D.). « μSpider: a CAD Tool for Efficient NoC Design ». In *Proceedings of the Norchip Conference*, p. 218–221, 2004. (referenced page 37)

[38] FOSFOR. « `http://users.polytech.unice.fr/~fmuller/fosfor/` », 2008. (referenced page 7)

[39] Freitas (H.) and Navaux (P.). « Evaluating on-chip interconnection architectures for parallel processing ». In *11th IEEE International Conference on*

Computational Science and Engineering Workshops (CSEWORKSHOPS'08), p. 188–193, 2008. (referenced page 35)

[40] Gaisler (A.). *GRLIB IP Core User's Manual v1.1.0*, 2011. (referenced pages 27 and 69)

[41] Gebali (F.), Elmiligi (H.) and El-Kharashi (M.), *Networks-on-chips: Theory and Practice*. CRC Press, Inc., 2009. (referenced page 28)

[42] Gericota (M. G.), Alves (G. R.), Silva (M. L.) and Ferreira (J. M.), « Run-Time Management of Logic Resources on Reconfigurable Systems », *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 03)*, 2003, p. 974–979. (referenced page 18)

[43] Gohringer (D.), Liu (B.), Hubner (M.) and Becker (J.). « Star-Wheels Network-on-Chip featuring a self-adaptive mixed topology and a synergy of a circuit- and a packet-switching communication protocol ». In *Field Programmable Logic and Applications, FPL'09*, 2009. (referenced page 32)

[44] Goossens (K.), Dielissen (J.) and Radulescu (A.), « Æthereal network on chip: concepts, architectures, and implementations », *Design & Test of Computers, IEEE*, **22**, 2005, p. 414–421. (referenced page 29)

[45] Hecht (R.), Kubish (S.), Michelsen (H.), Zeeb (E.) and Timmermann (D.), « A Distributed Object System Approach for Dynamic Reconfiguration », *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, p. 8. (referenced page 16)

[46] Hennessy (J. L.) and Patterson (D. A.). *Computer Architecture: A Quantitative Approach*, chapter Appendix E : Interconnection Networks. Morgan Kaufmann, 2006. (referenced pages 18, 28, 31, 32, 33, 34, 43, 49 and 53)

[47] Hilton (C.) and Nelson (B.). « Pnoc: a flexible circuit-switched noc for fpga-based systems ». In *Computers and Digital Techniques, IEE Proceedings*, 2006. (referenced page 30)

[48] Holsmark (R.), Kumar (S.), Palesi (M.) and Mejia (A.). « HiRA: A Methodology for Deadlock Free Routing in Hierarchical Networks on Chip ». In *3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS 2009)*, p. 10, 2009. (referenced page 36)

[49] Horta (E. L.), Lockwood (J. W.), Taylor (D. E.) and Parlour (D.), « Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfigu-

ration », *Design Automation Conference (DAC)*, 2002, p. 343–348. (referenced page 13)

[50] Hossain (H.), Ahmef (M.), Al-Nayeem (A.), Islam (T. Z.) and Akbar (M. M.). « gpNoCsim - A general purpose simulator for network-on-Chip ». In *International Conference on Information and Communication Technology (ICICT)*, 2007. (referenced page 117)

[51] IBM. *Cell Broadband Engine Programming Handbook*, 2008. Version 1.11. (referenced page 31)

[52] Ivanov (A.) and Micheli (G. D.), « The Network-on-Chip Paradigm in Practice and Research », *Design & Test of Computers, IEEE*, **22**(5), 2005, p. 399–403. (referenced page 28)

[53] Jantsch (A.) and Tenhunen (H.), *Networks on Chip.* Kluwer academic Publishers, 2003. (referenced page 28)

[54] Jara-Berrocal (A.) and Gordon-Ross (A.), « SCORES: A Scalable and Parametric Streams-Based Communication Architecture for Modular Reconfigurable Systems », *Design, Automation and Test in Europe Conference and Exhibition, 2009. Proceedings*, 2009. (referenced page 34)

[55] Jovanovic (S.), Tanougast (C.), Bobda (C.) and Weber (S.), « CuNoC: A dynamic scalable communication structure for dynamically reconfigurable FPGAs », *Microprocessors & Microsystems*, **33**, 2009, p. 24–36. (referenced page 30)

[56] Jung (E.-G.), Choi (B.-S.) and Lee (D.-I.), « High performance asynchronous bus for SoC », *Proceedings of the 2003 International Symposium on Circuits and Systems. ISCAS '03.*, **5**, 2003, p. 505–508. (referenced page 27)

[57] Kalte (H.), Porrmann (M.) and Rückert (U.), « System-on-Programmable-Chip Approach Enabling Online Fine-Grained 1D-Placement », *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 04)*, 2004. (referenced page 16)

[58] Kariniemi (H.). *On-Line Reconfigurable Extended Generalized Fat Tree Network-on-Chip for multiprocessor System-on-Chip Circuits.* PhD Thesis, Tampere university of Technology, 2006. (referenced pages 33 and 36)

[59] Kariniemi (H.) and Nurmi (J.). « New adaptive routing algorithm for extended generalized fat trees on-chip ». In *International Symposium on System-on-Chip*, p. 113–118, 2003. (referenced page 33)

[60] Kariniemi (H.) and Nurmi (J.), « Reusable XGFT interconnect IP for Network-on-Chip implementations », *Symposium on System-on-Chip.*, 2004, p. 95– 102. (referenced page 33)

[61] Kariniemi (K.) and Nurmi (J.). « Fault tolerant XGFT network on chip for multi processor system on chip circuits ». In *International Conference on Field Programmable Logic and Applications (FPL2005)*, p. 203–210. IEEE, 2005. (referenced page 33)

[62] Khalaf (M.) and Jagtiani (A.), « Making hardware more like software », *Embedded Systems Design (ESD)*, **24**(5), June 2011, p. 22–28. (referenced page 6)

[63] Khan (G.) and Dumitriu (V.), « A modeling tool for simulating and design of on-chip network systems », *Microprocessors and Microsystems*, **34**, 2010, p. 84–95. (referenced page 117)

[64] Koch (D.), Beckhoff (C.) and Teich (J.), « ReCoBus-Builder - A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs », *Field Programmable Logic and Applications.*, 2008, p. 119–124. (referenced page 27)

[65] Koh (S.) and Diessel (O.), « COMMA: A Communications Methodology for Dynamic Module Reconfiguration in FPGAs (Extended Abstract) », *Dagstuhl Seminar Proceedings 06141, Dynamically Reconfigurable Architectures*, 2006, p. 1–17. (referenced page 28)

[66] Kohler (A.) and Radetzki (M.). « Fault-Tolerant Architecture and Deflection Routing for Degradable NoC Switches ». In *3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS 2009)*, 2009. (referenced page 30)

[67] Krasteva (Y. E.), de la Torre (E.) and Riesgo (T.), « Reconfigurable Networks on Chip: DRNoC architecture », *Journal of Systems Architecture*, **56**(7), 2010, p. 293 – 302. Special Issue on HW/SW Co-Design: Systems and Networks on Chip. (referenced page 39)

[68] Kumar (R.), Zyuban (V.) and Tullsen (D. M.), « Interconnections in multicore architectures: Understanding mechanisms, overheads and scaling », *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA'05)*, 2005, p. 408–419. (referenced page 28)

[69] Kumar (S.), Jantsch (A.), Soininen (J.-P.), Forsell (M.), Millberg (M.), Oberg (J.), Tiensyrja (K.) and Hemani (A.), « A network on chip architec-

ture and design methodology », *IEEE Computer Society Annual Symposium on VLSI*, 2002, p. 105–112. (referenced page 29)

[70] LEE (S.), LEE (C.) and LEE (H.-J.). « A new multi-channel on-chip-bus architecture for system-on-chips ». In *IEEE International SOC Conference, Proceedings.*, p. 305–308, 2004. (referenced page 35)

[71] LEISERSON (C. E.), « Fat-trees: universal networks for hardware-efficient supercomputing », *IEEE Transactions on Computers*, **34**, 1985, p. 892–901. (referenced page 33)

[72] LIU (M.), KUEHN (W.), LU (Z.) and JANTSCH (A.), « Run-time partial reconfiguration speed investigation and architectural design space exploration », *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'09)*, 2009, p. 439–444. (referenced pages 16, 86 and 174)

[73] LYSAGHT (P.), BLODGET (B.), MASON (J.), YOUNG (J.) and BRIDGFORD (B.), « Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas », *Field Programmable Logic and Applications. FPL'06.*, 2006, p. 12–17. (referenced pages 6, 14 and 79)

[74] MARESCAUX (T.), BARTIC (A.), VERKEST (D.), VERNALDE (S.) and LAUWEREINS (R.), « Interconnection Networks Enable Fine-Grain Dynamic Multitasking on FPGAs », *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL), LNCS 2438, M. Glesner, P. Zipf, and M. Renovell (Eds.)*, **2438**, 2002, p. 795–805. (referenced page 30)

[75] MARESCAUX (T.), MIGNOLET (J.-Y.), BARTIC (A.), MOFFAT (W.), VERKEST (D.), VERNALDE (S.) and LAUWEREINS (R.), « Networks on Chip as Hardware Components of an OS for Reconfigurable Systems », *Field-Programmable Logic and Applications*, **177**, 2003, p. 595–605. (referenced page 16)

[76] MARKOVSKY (Y.), PATEL (Y.) and WAWRZYNEK (J.). « Using Adaptive Routing to Compensate for Performance Heterogeneity ». In *3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS 2009)*, 2009. (referenced page 37)

[77] MOADELI (M.) and VANDERBAUWHEDE (W.). « A communication model of broadcast in wormhole-routed networks on-chip ». In *International Conference on Advanced Information Networking and Applications*, p. 315–322. IEEE, 2009. (referenced page 32)

[78] Moadeli (M.), Vanderbauwhede (W.) and Shahrabi (A.). « A performance model of communication in the quarc noc ». In *14th IEEE International Conference on Parallel and Distributed Systems*, p. 908–913, 2008. (referenced page 32)

[79] Modarressi (M.), Sarbazi-Azad (H.) and Arjomand (M.). « A hybrid packet-circuit switched on-chip network based on sdm ». In *Proceedings of the Conference on Design, Automation and Test in Europe*, p. 566–569, 2009. (referenced page 30)

[80] Moller (L.), Fischer (P.), Moraes (F.), Indrusiak (L. S.) and Glesner (M.), « Improving QoS of Multi-Layer Networks-on-Chip with Partial and Dynamic Reconfiguration of Routers », *Field Programmable Logic and Applications. FPL'10.*, 2010, p. 1–5. (referenced pages 30, 39 and 80)

[81] Moraes (F.), Calazans (N.), Mello (A.), Moller (L.) and Ost (L.), « HERMES: an infrastructure for low area overhead packet-switching networks on chip », *Integration, the VLSI Journal*, **38**, 2004, p. 69–93. (referenced pages 29, 58 and 116)

[82] Moraes (F.), Calazans (N.), Möller (L.), Brião (E.) and Carvalho (E.). *New Algorithms, Architectures and Applications for Reconfigurable Computing*, chapter Dynamic and Partial Reconfiguration in FPGA SoCs: Requirements Tools and a Case Study, p. 157–168. Springer US, 2005. (referenced page 13)

[83] Murali (S.), *Designing Reliable and Efficient Networks on Chips*, volume 34. 2009. (referenced page 37)

[84] Muttersbach (J.), Villiger (T.), Kaeslin (H.), Felber (N.) and Fichtner (W.), « Globally-Asynchronous Locally-Synchronous architectures to simplifythe design of on-chip systems », *ASIC/SOC Conference, 1999. Proceedings. Twelfth Annual IEEE International*, 1999, p. 317–321. (referenced page 27)

[85] Neeb (C.) and Wehn (N.), « Designing efficient irregular networks for heterogeneous systems-on-chip », *Journal of Systems Architecture*, **54**, 2008, p. 384–396. (referenced page 37)

[86] Ngo (V.-D.) and Choi (H.-W.), « Analyzing the Performance of Mesh and Fat-Tree Topologies for Network on Chip Design », *Computeur Science*, **3824**, 2005, p. 300–310. (referenced page 33)

[87] Nikolic (T.), Stojcev (M.) and Djordjevic (G.), « CDMA bus-based on-chip interconnect infrastructure », *Microelectronics Reliability*, **49**, 2009, p. 448–459. (referenced page 27)

[88] Nollet (V.), Marescaux (T.) and Verkest (D.), « Operating-System Controlled Network on Chip », *Design Automation Conference, 41st Conference on (DAC'04)*, **35**, 2004, p. 256–259. (referenced page 16)

[89] OpenCores. *Wishbone B4, WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*, 2010. (referenced page 27)

[90] Palesi (M.), Patti (D.) and Fazzino (F.). *Noxim: the NoC simulator, User Guide.* University of Catania, 2010. (referenced page 117)

[91] Pande (P. P.), Grecu (C.), Jones (M.), Ivanov (A.) and Saleh (R.), « Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures », *IEEE Transactions on Computers*, **54**, 2005, p. 1025–1040. (referenced page 37)

[92] Park (K.) and Kim (H.). *Remote FPGA Reconfiguration Using MicroBlaze or PowerPC Processors.* Xilinx, Application Note: XAPP441 (v1.1) edition, 2006. (referenced page 16)

[93] Patooghy (A.), Miremadi (S. G.) and Fazeli (M.), « A low-overhead and reliable switch architecture for Network-on-Chips », *Integration, the VLSI Journal*, **43**, 2010, p. 268–278. (referenced page 30)

[94] Petrini (F.) and Vanneschi (M.). « k-ary n-trees: High Performance Networks for Massively Parallel Architectures ». In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, p. 87–93, 1997. (referenced page 33)

[95] Pionteck (T.), Albrecht (C.), Koch (R.), Maehle (E.), Hubner (M.) and Becker (J.). « Communication Architectures for Dynamically Reconfigurable FPGA Designs ». In *IEEE International Parallel and Distributed Processing Symposium*, p. 174–182, 2007. (referenced page 38)

[96] Pionteck (T.), Koch (R.) and Albrecht (C.), « Applying Partial Reconfiguration to Networks-on-Chips », *Field Programmable Logic and Applications.*, 2006, p. 1–6. (referenced page 38)

[97] Rana (V.), Atienza (D.), Santambrogio (M.), Sciuto (D.) and Micheli (G. D.). « A reconfigurable network-on-chip architecture for optimal multiprocessor SoC communication ». In *VLSI-SoC: Design Methodologies for SoC and SiP*, p. 232–250. Springer, 2010. (referenced page 39)

[98] Salminen (E.), Kulmala (A.) and Hamalainen (T. D.), « Survey of Network-on-chip Proposals », *OCP-IP White Paper,* `http://www.ocpip.org/whitepapers.php`, 2008. (referenced pages 28 and 43)

[99] Salminen (E.), Kangas (T.), Hämäläinen (T. D.), Riihimäki (J.), Lahtinen (V.) and Kuusilinna3 (K.), « HIBI Communication Network for System-on-Chip », *The Journal of VLSI Signal Processing*, **43**, 2006, p. 185–205. (referenced page 27)

[100] Samman (F.), Hollstein (T.) and Glesner (M.). « Planar adaptive router microarchitecture for tree-based multicast network-on-chip ». In *Network on Chip Architectures*, 2008. (referenced page 30)

[101] Schuck (C.), Lamparth (S.) and Becker (J.). « artNoC - A novel Multi-Functional router architecture for Organic Computing ». In *Field Programmable Logic and Applications, FPL'07*, 2007. (referenced page 29)

[102] Seceleanu (T.), Plosila (J.) and Liljeberg (P.), « On-Chip Segmented Bus: A Self Timed Approach », *IEEE ASIC/SOC Conference*, 2002, p. 216–220. (referenced page 27)

[103] Sekar (K.), Lahiri (K.), Raghunathan (A.) and Dey (S.), « Dynamically configurable bus topologies for high-performance on-chip communication », *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **16**, 2008, p. 1413–1426. (referenced page 27)

[104] Song (W.), Edwards (D.), Nunez-Yanez (J. L.) and Dasgupta (S.). « Adaptive Stochastic Routing in Fault-tolerant On-chip Networks ». In *3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS 2009)*, 2009. (referenced page 30)

[105] Steiger (C.), Walder (H.), Platzner (M.) and Thiele (L.), « Online Scheduling and Placement of Real-time Tasks to Partially Reconfigurable Devices », *Real-Time Systems Symposium, RTSS 2003, 24th IEEE*, 2003, p. 224–225. (referenced page 16)

[106] The VINT project. *The ns Manual.* UC Berkeley, LBL, USC/ISI, Xerox PARC, 2000. (referenced page 116)

[107] Thomas (A.) and Becker (J.), « Dynamic Adaptive Runtime Routing Techniques in multigrain reconfigurable Hardware Architectures », *Conference on Field-Programmable Logic And Applications*, **3203**, 2004, p. 115–124. (referenced page 37)

[108] Wang (H.-S.), Zhu (X.), Peh (L.-S.) and Malik (S.). *Orion: A Power-Performance Simulator for Interconnection Networks*. Princeton University, 2002. (referenced page 116)

[109] W.H.Hu, Lee (S.) and Bagherzadeh (N.). « DMesh: a Diagonally-Linked Mesh Network-on-Chip Architecture ». In *Network on Chip Architectures*, p. 14. citeseer, 2008. (referenced page 30)

[110] Wiklund (D.) and Liu (D.), « Design of a system-on-chip switched network and its design support », *IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions*, **2**, 2002, p. 1279–1283. (referenced page 29)

[111] Wiklund (D.) and Liu (D.). « Socbus: switched network on chip for hard real time embedded systems ». In *Parallel and Distributed Processing Symposium*, 2003. (referenced page 30)

[112] Winegarden (S.), « Bus architecture of a system on a chip with user-configurable system logic », *IEEE Journal of Solid-State Circuits*, **35**(3), 2000, p. 425–433. (referenced page 27)

[113] Wolkotte (P. T.), Smit (G. J.), Kavaldjiev (N.), Becker (J. E.) and Becker (J.), « Energy Model of Networks-on-Chip and a Bus », *International Symposium on System-on-Chip : Proceedings*, 2005, p. 82–85. (referenced page 30)

[114] Wolkotte (P.), Smit (G.) and Becker (J.). « Energy-Efficient NoC for Best-Effort Communication ». In *International Conference on Field Programmable Logic and Applications (FPL2005)*, p. 197–202, 2005. (referenced page 30)

[115] Xilinx. *Difference-Based Partial Reconfiguration, Application Note XAPP290*, 2007. (referenced page 15)

[116] Xilinx. *Virtex-4 FPGA Configuration User Guide*, v1.10 edition, April 8 2008. (referenced page 6)

[117] Xilinx. *Virtex-5 FPGA Configuration User Guide*, 2008. v3.5. (referenced pages 6, 14 and 17)

[118] Xilinx. *Xilinx : Virtex-5 User Guide*, v4.3 edition, September 23 2008. (referenced pages 14, 15 and 91)

[119] Xilinx. *LogiCORE IP On-Chip Peripheral Bus V2.0 with OPB Arbite*, 2010. v1.00b. (referenced page 27)

[120] Xilinx. *LogiCORE IP Processor Local Bus (PLB) v4.6*, 2010. v1.05. (referenced page 27)

[121] Xilinx. *PlanAhead User Guide (v12.4)*, december 2010. (referenced page 6)

[122] Xilinx. *Synthesis and Simulation Design Guide (v12.4)*, september 2010. (referenced page 6)

[123] Xilinx. *7 series FPGAs Configuration user Guide*, march 2011. (referenced page 6)

[124] Xilinx. *Partial Reconfiguration of Xilinx FPGAs Using ISE 12 Design Suite*, july 2011. (referenced page 6)

[125] Xilinx. *XST User Guide for Virtex-6 and Spartan-6 Devices*, December 2, 2009. UG687 (v 11.4). (referenced page 6)

[126] Yuan (X.), Nienaber (W.), Duan (Z.) and Melhem (R.), « Oblivious routing for Fat-Tree based system area networks with uncertain traffic demands », *SIGMETRICS '07 Conference Proceedings*, **35**, 2007, p. 337 – 348. (referenced page 34)

[127] Zeferino (C. A.), Kreutz (M. E.), Carro (L.) and Susin (A. A.). « A study on communication issues for system-on-chip ». In *15th Symposium on Integrated Circuits and Systems Design*, p. 121–126, 2002. (referenced page 28)

Personal publications

[128] Devaux (L.), Chillet (D.), Pillement (S.) and Demigny (D.). « Flexible communication support for dynamically reconfigurable FPGAs ». In *Southern Programmable Logic*, 2009.

[129] Devaux (L.), Sassi (S. B.), Pillement (S.), Chillet (D.) and Demigny (D.). « DRAFT: Flexible interconnection network for dynamically reconfigurable architectures ». In *Field Programmable Technologies*, 2009.

[130] Devaux (L.), Sassi (S. B.), Pillement (S.), Chillet (D.) and Demigny (D.). « Réseau d'interconnexion flexible pour architecture reconfigurable dynamiquement et partiellement ». In *RenPar'19 / SympA'13 / CFSE'7*, 2009.

[131] Devaux (L.), Pillement (S.), Chillet (D.) and Demigny (D.). « Mesh and Fat-Tree comparison for dynamically reconfigurable applications ». In *Reconfigurable Communication-centric Systems on Chip (ReCoSoC'10)*, 2010.

[132] Devaux (L.), Pillement (S.), Chillet (D.) and Demigny (D.). « OS services for Reconfigurable System-on-Chip Communication ». In *Proceedings of the XXV Conference on Design of Circuits and Integrated Systems (DCIS)*, 2010.

[133] Devaux (L.), Pillement (S.), Chillet (D.) and Demigny (D.). « R2NoC : dynamically Reconfigurable Routers for flexible Networks on Chip ». In *International Conference on ReConFigurable Computing and FPGA's, ReConFig'10*, 2010.

[134] Pham (H.-M.), Devaux (L.), Pillement (S.) and Demigny (D.). « Dynamic NOC-based MPSoC with Fault-Tolerance Support ». In *DAC Workshop on "Diagnostic Services in Network-on-Chips (DSNoC)*, 2010.

[135] Devaux (L.), Sassi (S. B.), Pillement (S.), Chillet (D.) and Demigny (D.), « Flexible Interconnection Network for Dynamically and Partially Reconfigurable Architectures », *International Journal of Reconfigurable Computing*, **2010**(390545), 2010, p. 15.

[136] Muller (F.), Rhun (J. L.), Lemonnier (F.), Miramond (B.) and Devaux (L.), « A Flexible Operating System for Dynamic Applications », *Xcellence in New Applications (Xcell)*, **73**, 2010, p. 30–34.

[137] Narayanan (S.), Devaux (L.), Chillet (D.), Pillement (S.) and Sourdis (I.). « Communication Service for hardware tasks executed on dynamic and partial reconfigurable resources ». In *International Conference on Very Large Scale Integration (VLSI-SoC)*, 2011.

[138] Pham (H.-M.), Devaux (L.) and Pillement (S.). « Re2DA: Reliable and Reconfigurable Dynamic Architecture ». In *Reconfigurable Communication-centric Systems on Chip (ReCoSoC'11)*, 2011.

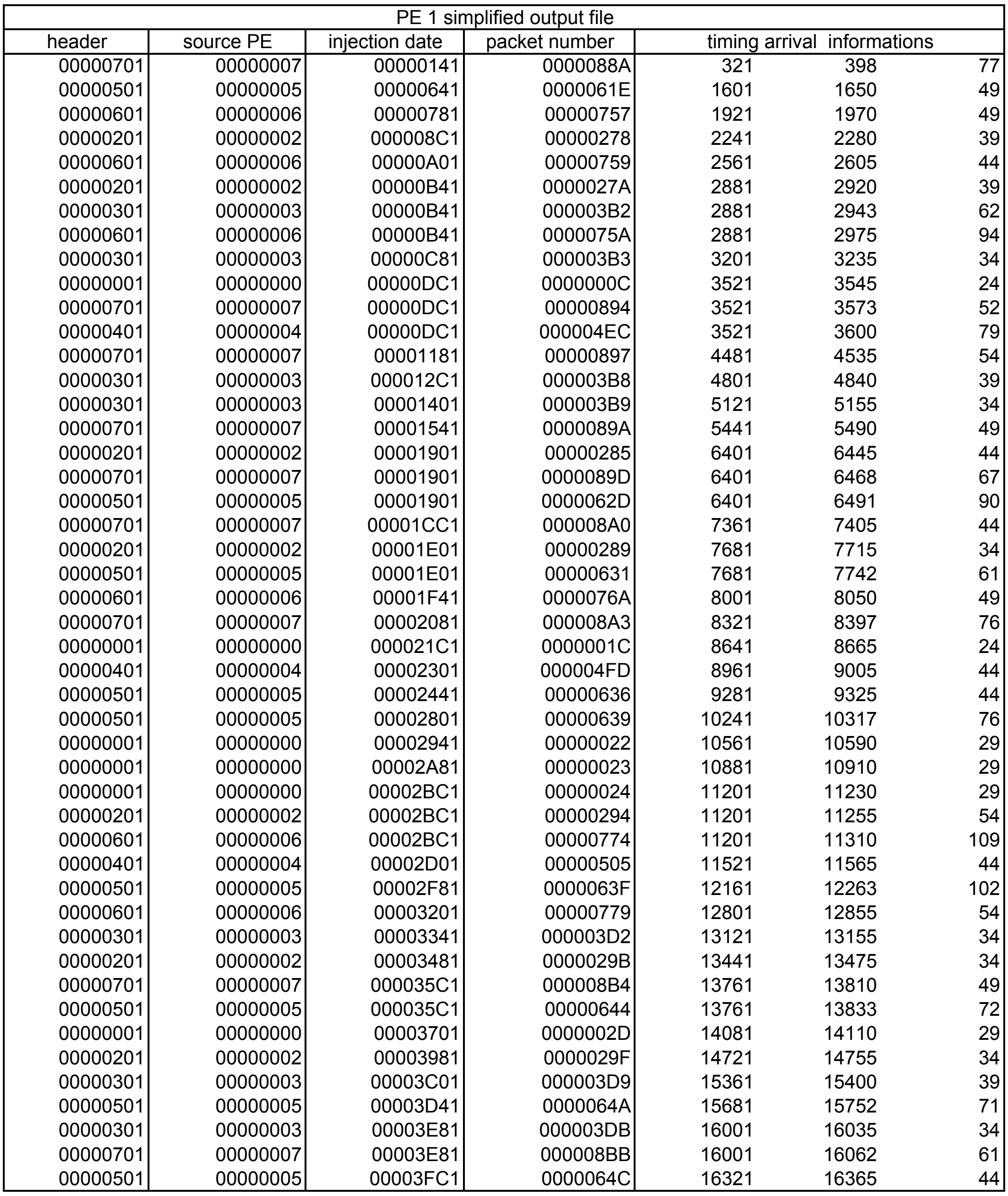
Appendix

Appendix A

Simulation files from DRAGOON

PE 1 input file																
Injection date	header	count	payload													
			source PE				injection date		packet number							
1	00000106	0000000E	00000001	00000000	00000000	00000000	00000001	00000000	00000139	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
141	00000100	0000000E	00000001	00000000	00000000	00000000	00000141	00000000	0000013A	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
281	00000106	0000000E	00000001	00000000	00000000	00000000	00000281	00000000	0000013B	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
3C1	00000100	0000000E	00000001	00000000	00000000	00000000	000003C1	00000000	0000013C	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
501	00000102	0000000E	00000001	00000000	00000000	00000000	00000501	00000000	0000013D	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
641	00000100	0000000E	00000001	00000000	00000000	00000000	00000641	00000000	0000013E	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
781	00000100	0000000E	00000001	00000000	00000000	00000000	00000781	00000000	0000013F	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
8C1	00000106	0000000E	00000001	00000000	00000000	00000000	000008C1	00000000	00000140	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
A01	00000100	0000000E	00000001	00000000	00000000	00000000	00000A01	00000000	00000141	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
B41	00000106	0000000E	00000001	00000000	00000000	00000000	00000B41	00000000	00000142	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
C81	00000103	0000000E	00000001	00000000	00000000	00000000	00000C81	00000000	00000143	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
DC1	00000100	0000000E	00000001	00000000	00000000	00000000	00000DC1	00000000	00000144	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
F01	00000100	0000000E	00000001	00000000	00000000	00000000	00000F01	00000000	00000145	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1041	00000107	0000000E	00000001	00000000	00000000	00000000	00001041	00000000	00000146	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1181	00000103	0000000E	00000001	00000000	00000000	00000000	00001181	00000000	00000147	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
12C1	00000100	0000000E	00000001	00000000	00000000	00000000	000012C1	00000000	00000148	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1401	00000103	0000000E	00000001	00000000	00000000	00000000	00001401	00000000	00000149	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1541	00000102	0000000E	00000001	00000000	00000000	00000000	00001541	00000000	0000014A	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1681	00000102	0000000E	00000001	00000000	00000000	00000000	00001681	00000000	0000014B	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
17C1	00000100	0000000E	00000001	00000000	00000000	00000000	000017C1	00000000	0000014C	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1901	00000103	0000000E	00000001	00000000	00000000	00000000	00001901	00000000	0000014D	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1A41	00000102	0000000E	00000001	00000000	00000000	00000000	00001A41	00000000	0000014E	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1B81	00000100	0000000E	00000001	00000000	00000000	00000000	00001B81	00000000	0000014F	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1CC1	00000102	0000000E	00000001	00000000	00000000	00000000	00001CC1	00000000	00000150	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
1F41	00000102	0000000E	00000001	00000000	00000000	00000000	00001F41	00000000	00000152	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2081	00000100	0000000E	00000001	00000000	00000000	00000000	00002081	00000000	00000153	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
21C1	00000100	0000000E	00000001	00000000	00000000	00000000	000021C1	00000000	00000154	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2301	00000106	0000000E	00000001	00000000	00000000	00000000	00002301	00000000	00000155	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2441	00000106	0000000E	00000001	00000000	00000000	00000000	00002441	00000000	00000156	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2581	00000107	0000000E	00000001	00000000	00000000	00000000	00002581	00000000	00000157	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
26C1	00000107	0000000E	00000001	00000000	00000000	00000000	000026C1	00000000	00000158	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2801	00000100	0000000E	00000001	00000000	00000000	00000000	00002801	00000000	00000159	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2941	00000107	0000000E	00000001	00000000	00000000	00000000	00002941	00000000	0000015A	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2A81	00000103	0000000E	00000001	00000000	00000000	00000000	00002A81	00000000	0000015B	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2BC1	00000100	0000000E	00000001	00000000	00000000	00000000	00002BC1	00000000	0000015C	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2D01	00000107	0000000E	00000001	00000000	00000000	00000000	00002D01	00000000	0000015D	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
2F81	00000100	0000000E	00000001	00000000	00000000	00000000	00002F81	00000000	0000015F	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
30C1	00000100	0000000E	00000001	00000000	00000000	00000000	000030C1	00000000	00000160	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
3201	00000107	0000000E	00000001	00000000	00000000	00000000	00003201	00000000	00000161	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
3341	00000103	0000000E	00000001	00000000	00000000	00000000	00003341	00000000	00000162	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
3481	00000105	0000000E	00000001	00000000	00000000	00000000	00003481	00000000	00000163	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
35C1	00000107	0000000E	00000001	00000000	00000000	00000000	000035C1	00000000	00000164	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
3701	00000105	0000000E	00000001	00000000	00000000	00000000	00003701	00000000	00000165	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
3841	00000104	0000000E	00000001	00000000	00000000	00000000	00003841	00000000	00000166	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
3981	00000105	0000000E	00000001	00000000	00000000	00000000	00003981	00000000	00000167	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
3AC1	00000106	0000000E	00000001	00000000	00000000	00000000	00003AC1	00000000	00000168	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E

Figure A.1 : Input simulation file example generated by DRAGOON for a PE (PE 1) connected to DRAFT. At each specified injection date, the corresponding packet is offered to the network. This packet is made of a header containing source and destination addresses, the count for a total of 16 transmitted flits, and the payload. Payload starts first with the source address, then followed by the injection date and a number (identifier) specific to this packet. Remaining of payload flits are not relevant.



PE 1 simplified output file						
header	source PE	injection date	packet number	timing arrival informations		
00000701	00000007	00000141	0000088A	321	398	77
00000501	00000005	00000641	0000061E	1601	1650	49
00000601	00000006	00000781	00000757	1921	1970	49
00000201	00000002	000008C1	00000278	2241	2280	39
00000601	00000006	00000A01	00000759	2561	2605	44
00000201	00000002	00000B41	0000027A	2881	2920	39
00000301	00000003	00000B41	000003B2	2881	2943	62
00000601	00000006	00000B41	0000075A	2881	2975	94
00000301	00000003	00000C81	000003B3	3201	3235	34
00000001	00000000	00000DC1	0000000C	3521	3545	24
00000701	00000007	00000DC1	00000894	3521	3573	52
00000401	00000004	00000DC1	000004EC	3521	3600	79
00000701	00000007	00001181	00000897	4481	4535	54
00000301	00000003	000012C1	000003B8	4801	4840	39
00000301	00000003	00001401	000003B9	5121	5155	34
00000701	00000007	00001541	0000089A	5441	5490	49
00000201	00000002	00001901	00000285	6401	6445	44
00000701	00000007	00001901	0000089D	6401	6468	67
00000501	00000005	00001901	0000062D	6401	6491	90
00000701	00000007	00001CC1	000008A0	7361	7405	44
00000201	00000002	00001E01	00000289	7681	7715	34
00000501	00000005	00001E01	00000631	7681	7742	61
00000601	00000006	00001F41	0000076A	8001	8050	49
00000701	00000007	00002081	000008A3	8321	8397	76
00000001	00000000	000021C1	0000001C	8641	8665	24
00000401	00000004	00002301	000004FD	8961	9005	44
00000501	00000005	00002441	00000636	9281	9325	44
00000501	00000005	00002801	00000639	10241	10317	76
00000001	00000000	00002941	00000022	10561	10590	29
00000001	00000000	00002A81	00000023	10881	10910	29
00000001	00000000	00002BC1	00000024	11201	11230	29
00000201	00000002	00002BC1	00000294	11201	11255	54
00000601	00000006	00002BC1	00000774	11201	11310	109
00000401	00000004	00002D01	00000505	11521	11565	44
00000501	00000005	00002F81	0000063F	12161	12263	102
00000601	00000006	00003201	00000779	12801	12855	54
00000301	00000003	00003341	000003D2	13121	13155	34
00000201	00000002	00003481	0000029B	13441	13475	34
00000701	00000007	000035C1	000008B4	13761	13810	49
00000501	00000005	000035C1	00000644	13761	13833	72
00000001	00000000	00003701	0000002D	14081	14110	29
00000201	00000002	00003981	0000029F	14721	14755	34
00000301	00000003	00003C01	000003D9	15361	15400	39
00000501	00000005	00003D41	0000064A	15681	15752	71
00000301	00000003	00003E81	000003DB	16001	16035	34
00000701	00000007	00003E81	000008BB	16001	16062	61
00000501	00000005	00003FC1	0000064C	16321	16365	44

Figure A.2 : Simplified presentation of the output file generated by Modelsim. This file contains all the packets that reached the destination (here PE 1). Relevant informations are for each packet, the header, a reminder of the source address, the injection date, the identifier of the packet, and finally timing informations (injection and arrival dates in decimal, clock cycles elapsed (latency)).

Appendix B

In situ characterization platform

B.1 Need of an In-Situ characterization platform

Every existing simulator requires an amount of modifications that can be important to support new networks. Such modifications can be delicate to guarantee the validity of the comparisons (different models, etc.) and can result in a very significant loss of time. This is how the idea of a standardized platform for characterization was born. Since every researcher uses its own methodology on designing NoCs, there are many different networks coded in many different languages, from RTL (VHDL or verilog) to object oriented languages such as java or C++. Many models were created to estimate performances but they can not cover the scope of all available NoCs. Since the finality of a NoC is not to be defined with a high level of abstraction but to be implemented in hardware, the idea came to create an open hardware platform. This platform would be on reconfigurable chip implemented, so that performances would not be estimated but measured in-situ. It is obvious that such a platform is not to be used during the definition of a network (high level simulators would save time in this context), but to efficiently compare an enough matured network with others. To do so, this platform should provide a standard interface. This way designers should only design a network interface (that is always specific to a NoC) making the link between this standard interface and the NoC. This platform would offer the possibility to create a testbench regrouping as much NoCs as possible. Every NoC would be characterized in the same conditions of traffic, of hardware target, etc. It would also provide the possibility to experiment in hardware the good functioning of a real life application from its data flow graph. Indeed, more than generating only random traffics of data, the possibility to specify specific traffics (in terms of source/destinations, data rates, etc.) would allow to replicate applications from the communication point of view. NoC designers should then find an interest in such a platform, but also application designers! For example, determining a satisfying placement of shared elements like memories in a network would be easily experimented thanks to this platform. Moreover, verifying the compliance between an interconnection architecture and an application, just starting from its data flow graph, would help saving time.

The general architecture of this in-situ characterization platform is presented in Figure B.1.

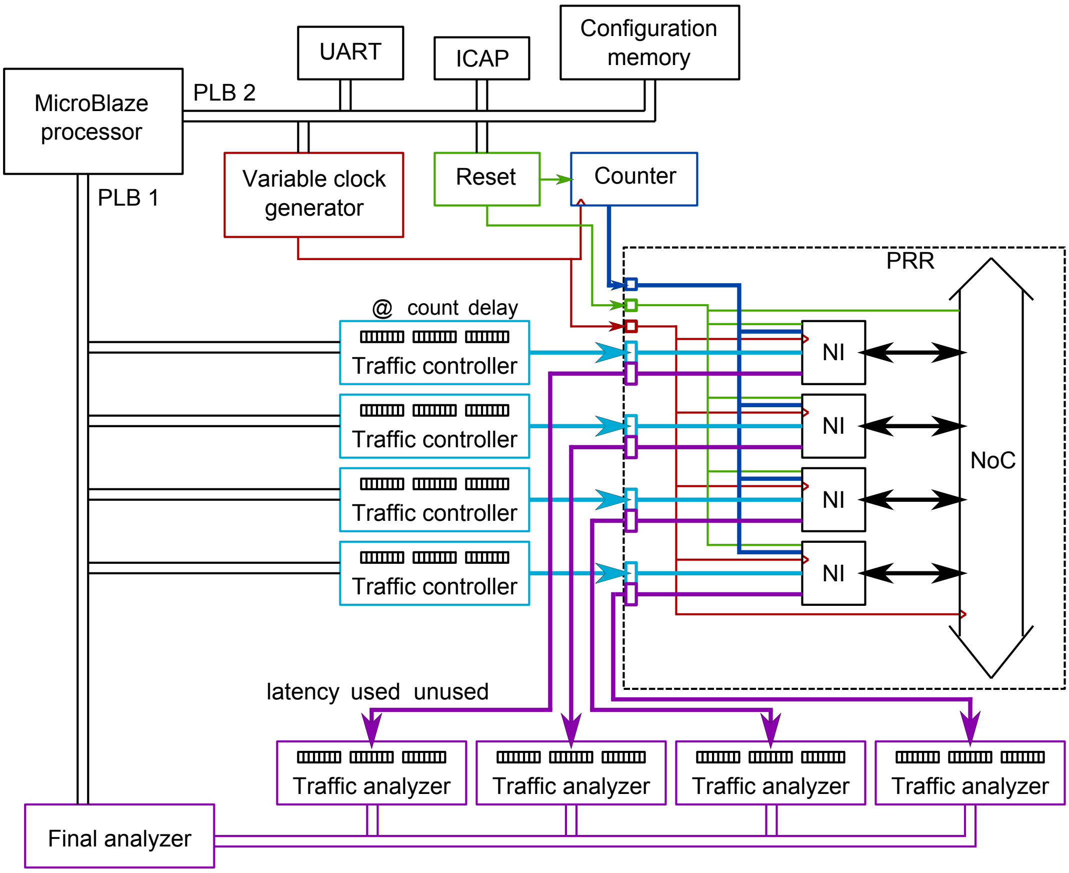


Figure B.1 : Architectural presentation of the in-situ NoC characterization platform.

The NoC to characterize is implemented inside a PRR. Doing so, several versions/configurations of the network can be evaluated without disturbing the remaining of the platform including user specified data traffics. Since Bus Macros (BMs) are required to enter/exit a PRR, they constitute the standard interface. NIs are specific for each NoC, this is why they are also implemented inside the PRR. Since only NIs have the knowledge of the communication protocols, they are responsible of the calculation of performances: latencies of the packet, number of clock cycles with and without incoming data. Their standard interface is defined as follow:

- a 6 bits width address used to specify the destination of the message to send (input),
- a 16 bits width count signal specifying the number of data to transmit (input),
- a 16 bits width delay information indicating the delay to observe between the end of current communication and the beginning of the following one (input),
- a 1 bit line specifying when address and count values can be changed without creating instability (output),
- a 1 bit line specifying when the delay information can be changed (output),
- a 16 bits width control latency, measured from the moment a communication is requested to the moment the first data can be sent (output),
- a 16 bits width global latency, measured from the moment a communication is requested to the moment the last data arrives at destination (output),
- a 16 bits width used count indicating the number of clock cycles during which data are received (output),
- a 16 bits width unused count indicating the number of clock cycles during which no data is received (output).

This standard interface is provided to each NI, thus one for each port of the network. In the in-situ platform, until 64 ports networks are supported. Thus, this interface is instantiated 64 times with a unitary cost of 13 Configurable Logical Blocks (CLBs). Furthermore, several signals are provided to the whole PRR (instantiated one time only):

- a variable clock that is user specified from 20 to 450MHz,
- a reset signal,
- a 32 bits width counter value for the time reference, clocked using the variable clock signal.

Thus, 4 additional CLBs are used for this interface. Behind these interfaces, the platform is organized as follow:

- a 32 bits counter clocked on a user configured clock reference,
- a clock generator, and a reset controller which are both linked by a PLB bus to a MicroBlaze processor,

- a MicroBlaze processor with three functions: controlling the dynamic reconfiguration of the PRR, receiving from the user the configuration to analyze (traffic, frequency, etc.), and displaying measured performances.
- an Universal Asynchronous Receiver Transmitter (UART) make the interface between the user and the MicroBlaze processor,
- an ICAP interface performing the reconfiguration of the PRR with configurations stored in a dedicated memory,
- several traffic controllers composed of 3 buffers: 1 storing a sequence of destination addresses, 1 for the count informations dedicated to each address in first buffer, and 1 for the delay informations,
- several traffic analyzers calculating average and maximal latencies (both control and global), the number of received data, and the elapsed time from their dedicated interfaces,
- a final analyzer centralizing informations from traffic analyzers and calculating final performances (latencies, data rates, throughput) weighting results by the number of received data and elapsed time.

Traffic controllers are one the most important part of the platform because they manage the data traffic of their dedicated NI. Every traffic controller is composed of three buffers allowing to store 64 informations each. Thus, through the UART and the MicroBlaze processor, users can specify a sequence of addresses that will be repeated until a new one is specified. Corresponding count and delay values acts the same way. With the knowledge of every PE's own data flow graph, the whole behavior of an application can be reproduced this way (or at least a typical test scenario). If more than 64 addresses are needed to characterize the behavior of a particular PE, informations are stored by the MicroBlaze in its memory and sent to traffic controllers at the time they are about to loop. If users do not want to specify a sequence of addresses, each traffic controller will have a random generator of addresses, even if informations in count and delay buffers are still required to control the offered data rate.

Appendix C

R2NoC measured performances

The generation of R2NoC induces 24 configurations (and so 24 partial bitstreams) per switch. Considering the limitations of present design tools, we decided to characterize first a single switch and the communication service (MicroBlaze processor) from the hardware consumption point of view. This way, the number of configurations to generate could be limited.

Implementation of the switch and the communication service were realized on a Xilinx ML506 development platform using a Xilinx XC5VSX50T FPGA. It was implemented using the Xilinx ISE 12.4 design software tool. Definition of the PRR corresponding to the switch is provided thanks to the Xilinx PlanAhead 12.4 software. To obtain the exact values of resource utilization, the Xilinx FPGA Editor software is used.

The Bus Macros used for communication between static and dynamic parts of the system represents the main resource utilization in the dynamic switches. They use 284 Look Up Tables (LUTs) while AND gates are implemented using only 140 LUTs. So, a total of 424 LUTs and no registers are used for the whole switches. The same switch, statically implemented, from the conventional fat-tree network have a cost of 657 LUTs and 207 registers. Resources utilizations of the switch and the communication service are summarized in Table C.1. The resources consumption of the communication service seems important when compared with a switch. However, when designing the R2NoC network, the cost of the routing part is multiplied by the number of switches while the cost of the communication service remains unchanged. This is true as the MicroBlaze processor is powerful enough to handle the control of the whole network. In small networks, the cost of the communication service can be significantly reduced by not using a MicroBlaze processor but a fully hardware implemented service, or at least a PicoBlaze processor.

To configure the switch, partial bitstreams of 22KB are generated whatever the chosen configurations. First, just after the initial configuration of the FPGA, bit-

Table C.1 : Hardware resources consumption of the R2NoC network.

	R2NoC switch alone		communication service		switch from static fat-tree
	Used	%	Used	%	used
Registers	0	0%	1221	4%	207
LUT	424	1%	1622	5%	657
DSP48E	0	0%	3	1%	0
BRAM36	0	0%	0	0%	0

streams are loaded from the original CompactFlash card to the DDR memory. We measured this operation to 951ms for the 24 bitstreams. Next, partial bitstreams are loaded from the DDR to the ICAP interface when the partial reconfiguration occurs. Each bitstream has the same size, thus we measured a reconfiguration time of 11.3ms for each configuration. This reconfiguration delay is very large considering present applications. However, in this implementation, we used the genuine ICAP interface provided by Xilinx. Some works like [72], or [35] obtained much better reconfiguration times modifying the hardware ICAP. Accordingly with their results, reconfiguration times from the order of 50-60μs can be reached. However, a reconfiguration time of 50μs represents 6250 clock cycles with a frequency of 125MHz. This latency per switch is too much important: in order for R2NoC to have an interest in a real life application, latencies from the range of conventional networks' (50 clock cycles for a conventional fat-tree interconnecting 8 PEs) are required.

When a switch is configured, it acts like a direct link with less than 1 clock cycle delay. This is an important result because when the pairs of sources and destinations do not often vary in a given application, the establishment time of the communications and the latencies are reduced to 1 clock cycle. If we consider a real life application requiring a network interconnecting 8 PEs, the establishment time of a communication with an ideal communication service should be of the order of 25000 clock cycles (200 μs). Considering the 1 clock cycle delays when a data path is already established, R2NoC is only interesting if more than 500 successive communications (considering the equivalent conventional fat-tree) occur using the same data path. Using Xilinx genuine ICAP interface, more than 113000 successive communications are required for R2NoC to be attractive.

Considering previous results, R2NoC is not adapted for real life applications using DPR, but only with statically implemented applications (in order to foresee the number of successive communications with the same couples source/destinations).

However, considering the 32 bits data width and the 125 MHz operating frequency, a bandwidth of 500 Mbytes/s per PE can be of an interest for pipelined applications where specialized PEs often communicate with the same destinations. Since we limited the scope of this PhD to applications using DPR where data traffics can not necessarily be predicted, no further investigations where realized on the R2NoC network.

Appendix D

ASIC implementation of OCEAN

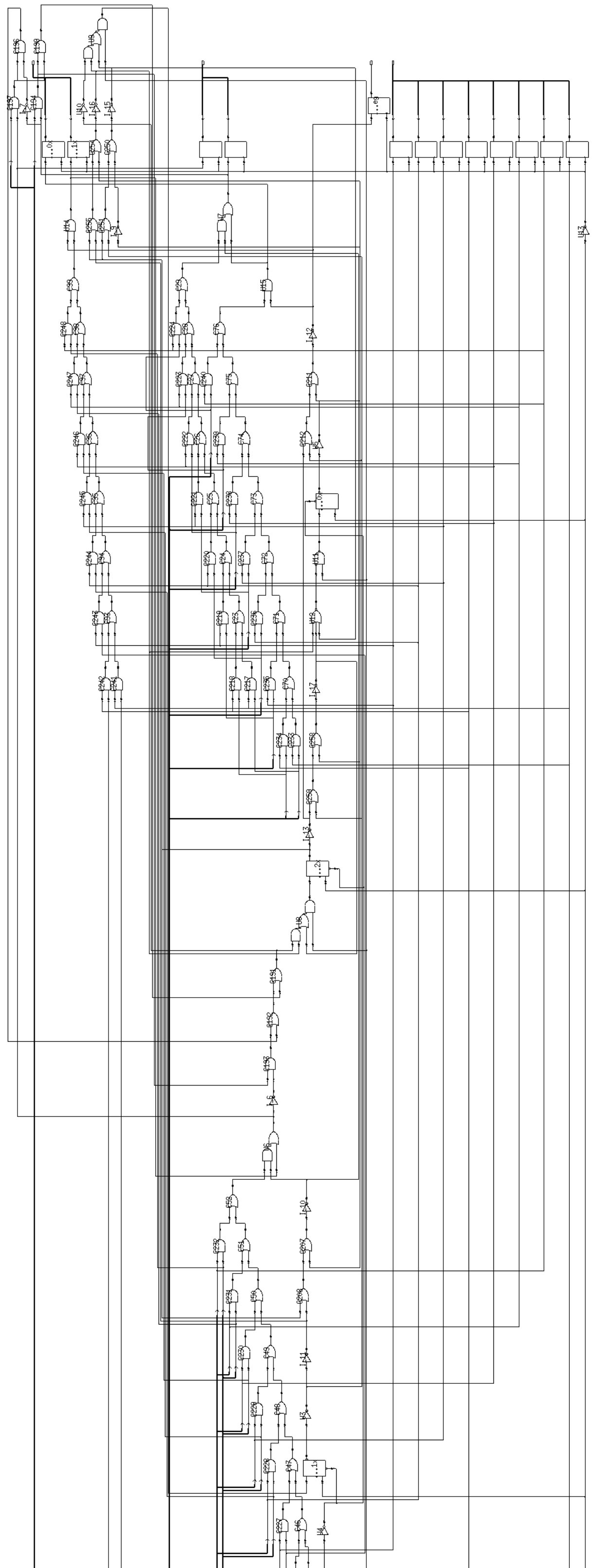


Figure D.1 : Presentation of an In-controller of the OCEAN v4.1 network at the gate level.

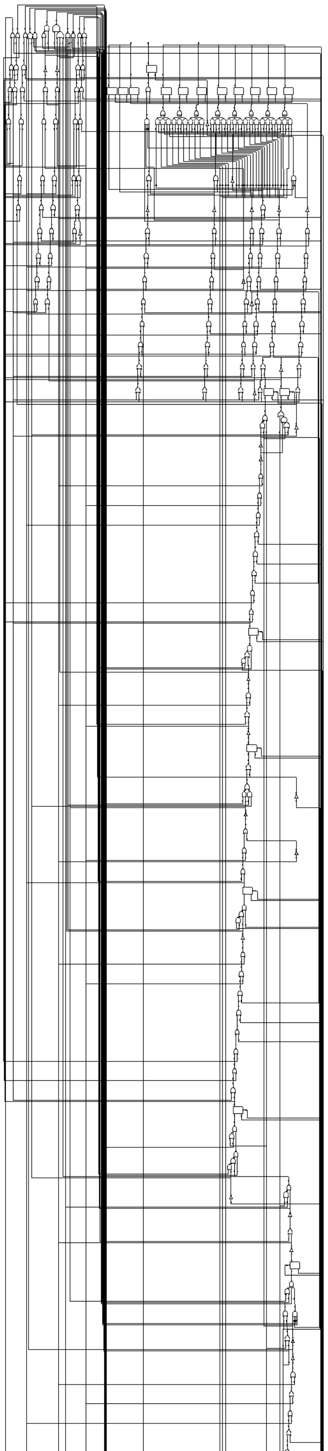


Figure D.2 : Bottom level Out-controller from the OCEAN v4.1 network.

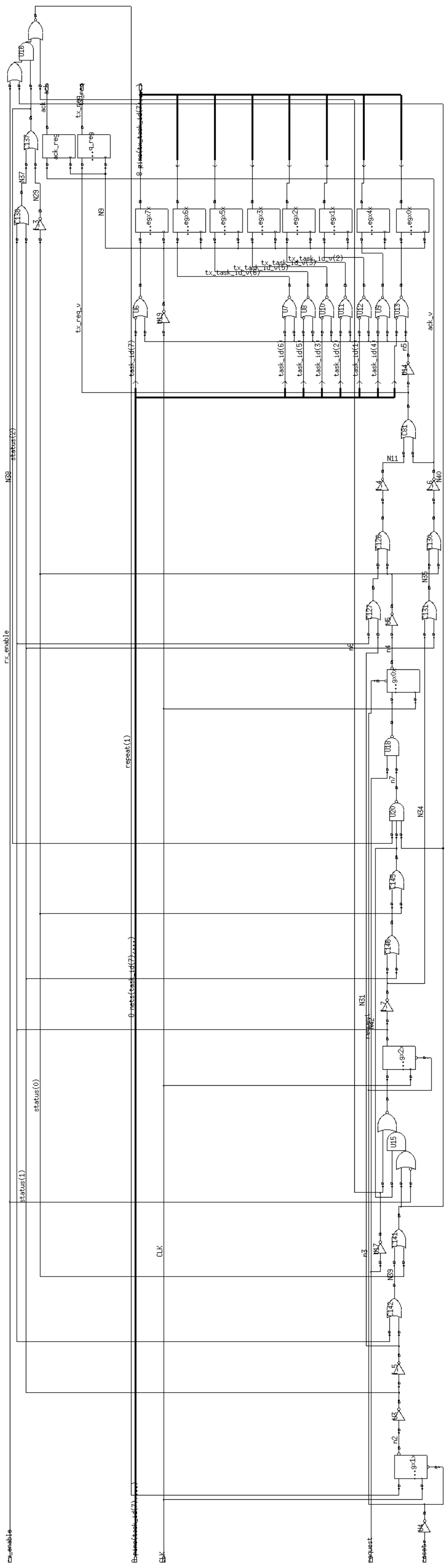


Figure D.3 : Top level Out-controller from the OCEAN v4.1 network.

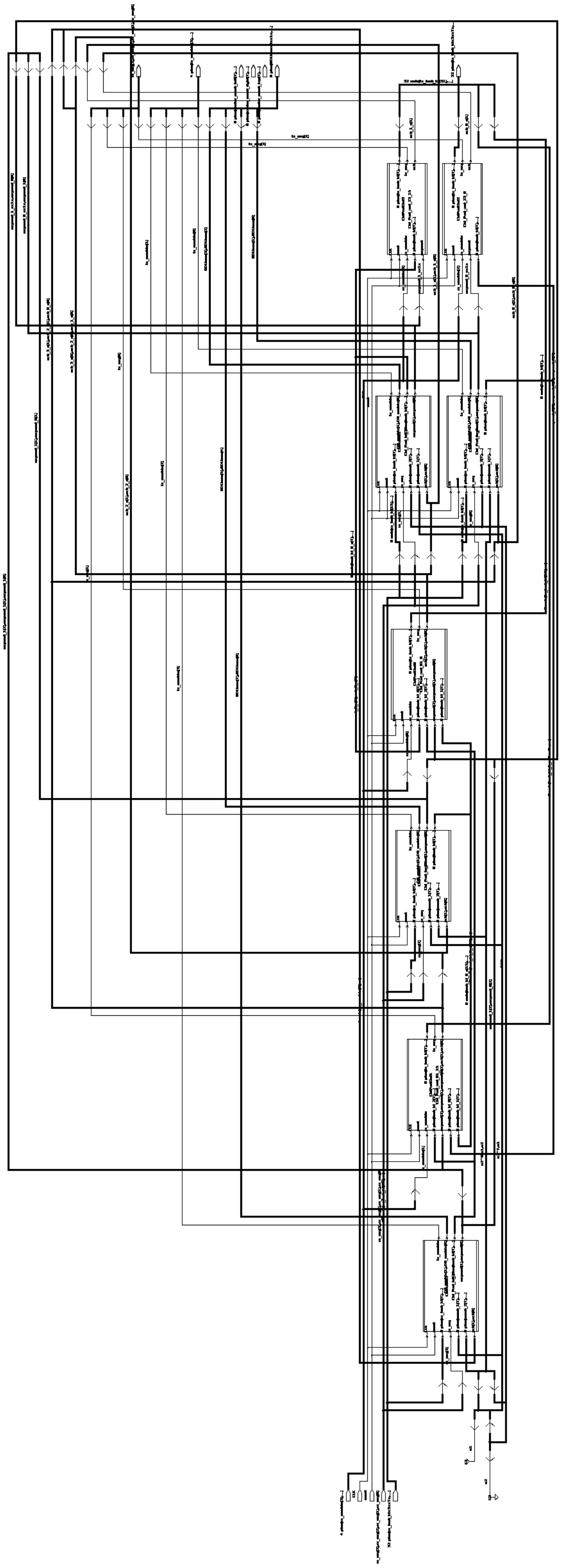


Figure D.4 : Overview of a single switch from the control part of the OCEAN v4.1 network. This switch uses both In and Out-controllers defined in previous figures.

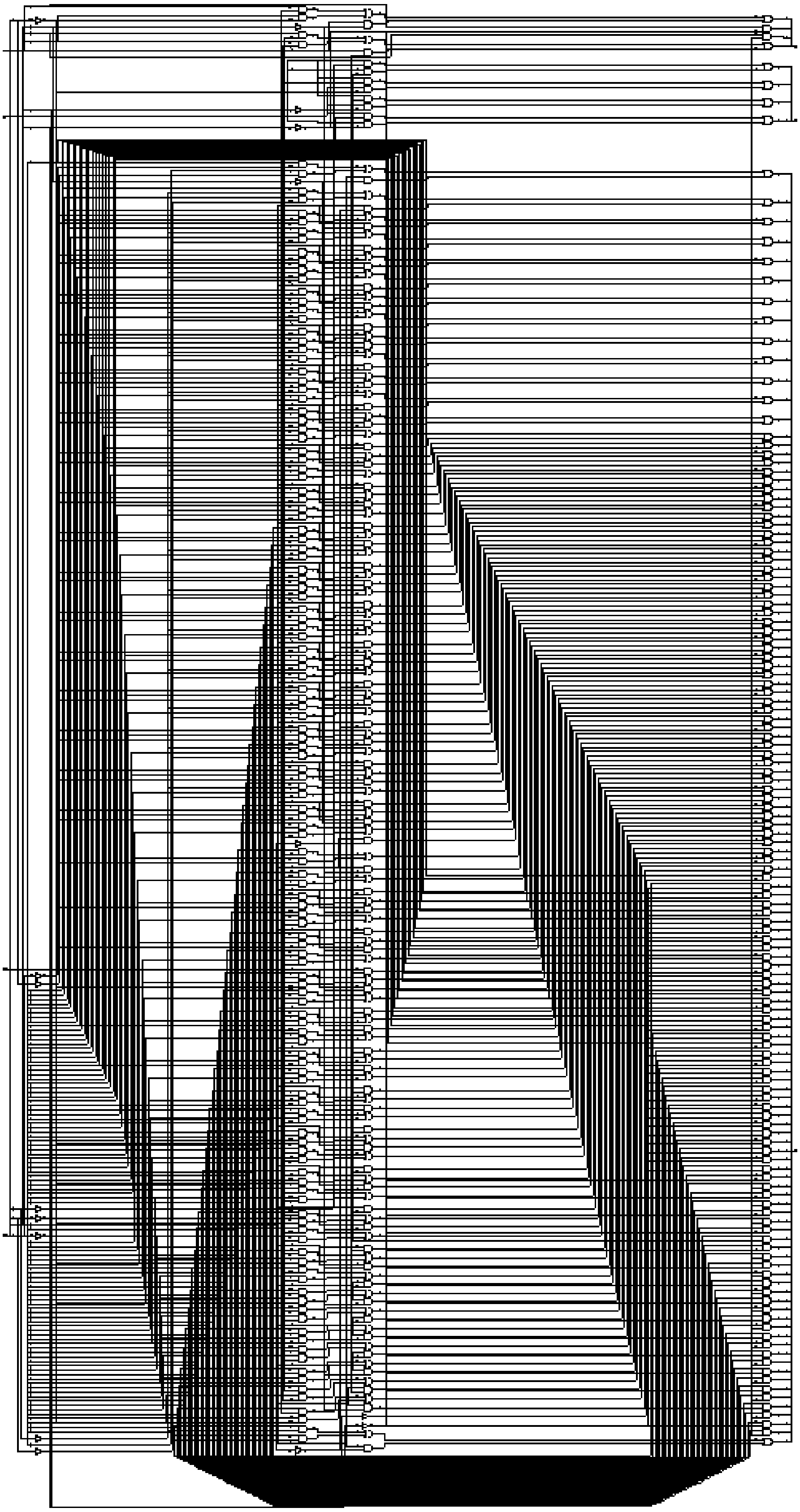


Figure D.5 : Overview the data sub-network from the OCEAN v4.1 network.

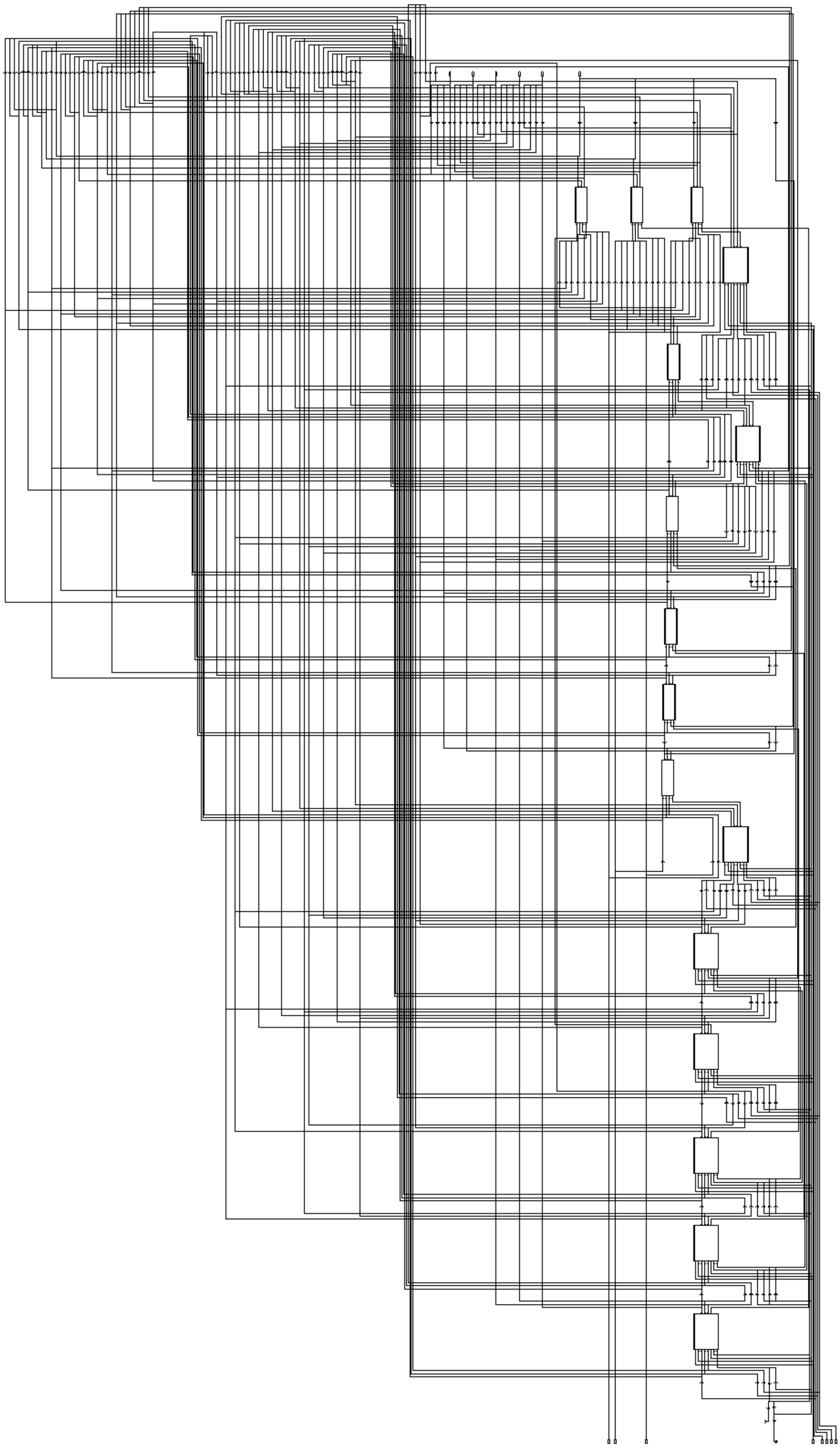


Figure D.6 : Presentation of the complete OCEAN v4.1 network interconnecting 8 PEs.