Customizing and Hardwiring On-Chip Interconnects in FPGAs

Customizing and Hardwiring On-Chip Interconnects in FPGAs

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen

op maandag 28 februari 2011 om 10:00 uur

door

Jae Young HUR

Master of Science in Communications Engineering,
Munich University of Technology
geboren te Jeju, South Korea

Dit proefschrift is goedgekeurd door de promotor:

Prof.dr. K.G.W. Goossens

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter	Technische Universiteit Delft
Prof. dr. K.G.W. Goossens, promotor	Technische Universiteit Delft
Dr. ir. J.S.S.M. Wong	Technische Universiteit Delft
Prof. dr. ir. A. -J. van der Veen	Technische Universiteit Delft
Prof. dr. B. H. H. Juurlink	Technische Universität Berlin
Prof. dr. H. Corporaal	Technische Universiteit Eindhoven
Dr. T. P. Stefanov	Universiteit Leiden
Prof. dr. ir. D. Stroobandt	Universiteit Gent
Prof. dr. ir. H. J. Sips, reservelid	Technische Universiteit Delft

Jae Young, Hur
Customizing and Hardwiring On-Chip Interconnects in FPGAs
Computer Engineering Laboratory
PhD Thesis Technische Universiteit Delft
Met samenvatting in het Nederlands.
Subject headings: FPGAs, Interconnects, Crossbars, Network on chip

Cover page: Mobile bridge Hambrug in Delft, depicted by Sun Young Park

ISBN: 978-90-72298-13-3

This thesis is dedicated to my family.

Customizing and Hardwiring On-Chip Interconnects in FPGAs

Jae Young Hur

Abstract

This thesis presents our investigations on how to efficiently utilize on-chip wires to improve network performance in reconfigurable hardware. A field-programmable gate array (FPGA), as a key component in a modern reconfigurable platform, accommodates many-millions of wires and the on-demand reconfigurability is realized using this abundance of wires. Modern FPGAs become computationally powerful as hardware IP (intellectual property) modules such as embedded memories, processor cores, and DSP modules are accommodated. However, the performance and the cost of the inter-IP communication remains a main challenge. We meet this challenge in two aspects.

First, conventional general-purpose on-chip networks suffer from high area cost when they are mapped onto the reconfigurable fabric. To reduce the area cost, we present a topology customization technique for a given set of applications. Specifically, we present an application-specific crossbar switch, crossbar schedulers, point-to-point interconnects, and circuit-switched networks-on-chip (NoCs) that reside on top of a reconfigurable fabric. As a result, by establishing only the necessary network resources, our customized interconnects provide significantly reduced cost compared to general-purpose on-chip networks.

Second, while the reconfigurability is a key benefit in FPGAs, it is traded off by decreased performance and increased cost. This is mainly because of the bit-level reconfigurable interconnects. To increase performance and reduce cost, we propose to replace the bit-level reconfigurable wires by hardwired circuit-switched interconnects for the inter-IP communication. Specifically, we present hardwired crossbars and a circuit-switched NoC interconnect fabric. We describe the advantages of the hardwired networks evidenced by the quantified performance analysis, network simulation, and an implementation. As a result, the hardwired networks provide two orders of magnitude better performance per area than the networks that are mapped onto the reconfigurable fabric.

Acknowledgments

It is my privilege to have an opportunity to conduct a research at computer engineering (CE) laboratory in TU Delft. This thesis eventually appeared due to the help from great teachers. First of all, I owe deep gratitude to Professor Kees Goossens for the significant guidance. I am most honored to be participated in his pioneering work related to network-on-chip. I learned a lot from him, especially an importance of realistic assumptions as well as how to do the co-work. The technical discussions with him have been definitely a joy. He provided not only a promising idea and keen remarks to improve the scientific quality but also kind suggestions and a warm compassion to help his students.

I am most grateful to Dr. Stephan Wong for his daily supervision, support, trust, and patience during my entire stay in Delft. Even during weekends, he replied to emails with elaborate comments. He taught me an importance of a conceptualization, how to write technical papers, and many other things. When I had problems, he provided a way to simplify the problem and encouraged me to proceed to reach the successful finish.

The experience in Delft was most valuable in my professional career. This has been possible since Professor Stamatis Vassiliadis[†] accepted me as a PhD student. Until April 2007, he supervised me and provided a project with a challenging research topic. He showed me how to conduct a research. I appreciate of committee professors for an acceptance to be members of the defense even though the time schedule was tight. I would like to thank Dr. Koen Bertels for providing opportunities to review papers. I would like to thank Dr. Arjan Genderen who helped me with ASIC tool setup and showed interests in the Korean culture. I am specially grateful to Professor S.Y. Hwang in Sogang University for introducing the computer engineering field. I appreciate faculty professors in Cheju National University in my home for their helpful suggestions.

This work was supported by Dutch Science Foundation (STW). Accordingly, I had a great opportunity to collaborate with excellent people. I thank student members of the Artemisia project, Hristo and Mark for interesting discussions during we

conducted the project. I specially thank Dr. Todor Stefanov in Leiden University for the support regarding the ESPAM tool. With his support, I was able to write a first conference paper. The train trip between Delft and Leiden was always exiting. Part of this thesis is based on the collaborative work with Æthereal NoC development team in NXP and TU Eindhoven. I would like to thank Martijn Coenen and Andreas Hansson for explaining the Æthereal tool chain. I had worked together with a nice colleague Aqeel for the hardwired NoC. We had good experience. I am thankful to him for discussions and co-work.

It is lucky for me to have nice colleagues at an international environment in CE group. I thank Stefan, Ricardo, Mahmood, Radu, Thomas, Yi Lu, Asad, Kamana, Nadeem, Zubair, Ozana, Lotfi, Ioannis, George, Pepijn, Filipa, Tariq, Chunyang for technical and non-technical discussions. Roel helped to translate the initial Abstract into Dutch. It was fun to play tennis with Christopher. With these nice friends, I was able to enjoy the research in a cheerful atmosphere, for which I feel indebted. I thank Dr. Yong Dou for many helpful discussions when he was a visiting researcher. I appreciate Neil Steiner for providing me data from his thesis. I would like to thank CICAT, the international liaison office, which helped me to settle down when I first came to the Netherlands. I thank Bert and Erik for their help to fix computer problems. I specially would like thank Lidwina Tromp for her helping to solve all administrative problems.

I thank Korean students in Delft, especially C.J. Kim, Dr. Bang, and Dr. Jang for get-together from time to time. The previous manager Dr. Suh, B. Y. Kim at CPU department, H. Choi, and S. M. Hong in Samsung Electronics encouraged me when I decided to go to Europe for the doctoral study. With all the supports from those people, I felt stable and tried to do my best. Dr. Kim and Dr. Um at the system interconnect team allowed me to trip back for the defense ceremony, for which I am thankful.

Finally, my family supported me in an everlasting and a dedicated way. I thank Sun Young, my wife, for gladly drawing the cover page. Everyday early in the morning, she made a traditional Gimbap lunch. I hope the first baby to be born will be proud of his/her father. I thank parents-in-law. They regularly sent packets that contain Gimbap materials. My sisters always supported their youngest brother. Last but not least, I am thankful to my parents for supporting the endeavor of their only son. They always accepted my requests and motivations. From now on, I hope to accept their requests and play the better role as a son.

Jae Young Hur Delft, The Netherlands, 2011

Contents

List of Tables

List of Figures

List of Acronyms

CLB	Configurable logic block
CSN	Circuit-switched network
CCSN	Customized circuit-switched network
CPS	Customized parallel scheduler
ESPAM	Embedded system-level platform synthesis and application mapping
FPGA	Field-programmable gate array
FPS	Full parallel scheduler
GT	Guaranteed throughput
HCSN	Hardwired circuit-switched network
HFBAR	Hardwired full crossbar
HWNoC	Hardwired network on chip
IP	Intellectual property
KPN	Kahn process network
LUT	Look-up table
MPSoC	Multi-processor systems-on-chip
NI	Network interface
NoC	Network on chip
NOQ	Network of queues
ρ**-P2P**	Reconfigurable point-to-point
SCCSN	Soft customized circuit-switched network
SCSN	Soft circuit-switched network
SCPS	Shard customized parallel scheduler
SQS	Sequential scheduler
SCBAR	Soft customized crossbar
SFBAR	Soft full crossbar
TDM	Time-division-multiplexing
VOQ	Virtual output queueing

Chapter 1

Introduction

Advances *in the semiconductor technology enable us to integrate increasingly more (intellectual property (IP)) cores on a single chip. The design of a modern system-on-a-chip (SoC) is increasingly becoming based on utilizing multiple IPs. At the same time, the system-on-a-chip requires a short time-to-market, low development cost, adaptability for targeted applications, and flexibility for post-fabrication reuse. At the forefront of silicon technology scaling, the field-programmable gate array (FPGA) is an integrated circuit that contains regular logic cells interconnected by reconfigurable wires. By exploiting the reconfigurability, any IP functionality can be implemented. Consequently, modern FPGAs are increasingly more capable in supporting applications with a short time-to-market and low development cost. Accordingly, FPGAs meet the above-mentioned requirements and are emerging as a main component in modern SoC platform. Moreover, modern FPGAs accommodate hardwired IP modules such as embedded memories and processor cores. Subsequently, FPGAs become computationally powerful as these hardwired modules are running at increasingly higher frequency. However, the performance of the inter-IP communication remains a problem in that communication latencies are becoming increasingly dominant in SoCs due to the continued growth of chip densities. This led to our quest to improve the performance of inter-IP communication in FPGAs.*

In this chapter, we present a short background in interconnects leading to the definition of the scope of this thesis. Subsequently, we define problem statements, design objectives, and our methodologies. Finally, we present lists the major contributions and an overview of this thesis.

1.1 Interconnects in FPGAs

In this section, we present a short introduction of various interconnects at the logical and physical abstraction layers. An FPGA fabric mainly contains reconfigurable resources such as configurable logic blocks (CLBs), wire segments, and switches. Using the reconfigurable resources, many functionalities can be implemented leading towards re-usable IP blocks. We refer to such IP blocks as *soft* in order to distinguish them from *hard* IP blocks that are hardwired. Examples of hard IP blocks are PowerPC processor cores and embedded memories in the Xilinx FPGAs [7]. The existence of hard IP blocks on an FPGA chip next to the reconfigurable resources stemmed from the need for additional performance of very commonly used (soft) IP blocks. Figure 1.1 depicts a mapping of an application onto the FPGA.

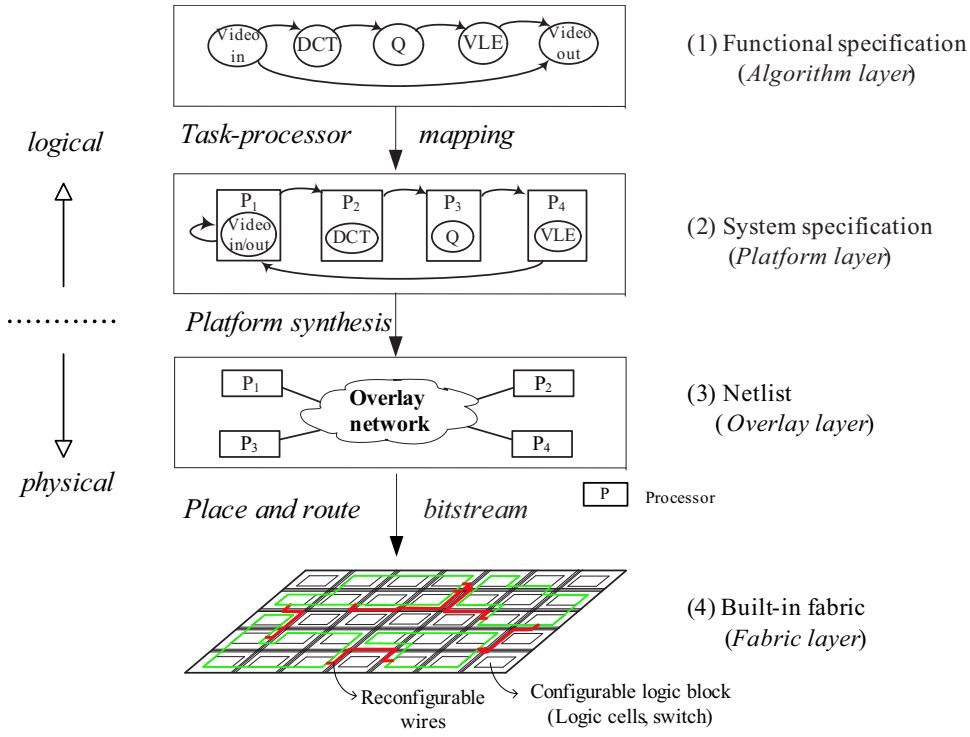


Figure 1.1: Logical and physical networks in different layers.

In the algorithm layer, the communication topology is specified by the task graph of the targeted application(s) as depicted in Figure 1.1(1). In the platform layer, the tasks are assigned to IPs as depicted in Figure 1.1(2). The edges in Figures 1.1(1) and 1.1(2) represent the *logical* networks that an application (or system) designer had in mind. The logical network functionality is implemented in physical network IPs such as shared buses, crossbars, or a network-on-chip (NoC). The physical network functionality is typically described in a synthesizable hardware description languages (HDL) by the system designer. These network IPs are synthesized into netlists as depicted in Figure 1.1(3). We define the synthesized netlists as *overlay* interconnects because they reside on top of underlying fabrics. Typically, these overlay interconnects are mapped, placed, and routed onto FPGA fabrics as depicted in Figure 1.1(4). We define the overlay interconnects mapped onto reconfigurable resources as *soft* interconnects because any network IP can be implemented on the reconfigurable fabric. In the following sections, we briefly review an FPGA interconnect fabric and typical overlay interconnects.

1.1.1 Reconfigurable interconnect fabric

In the fabric layer, the switches and wire segments constitute the reconfigurable interconnect fabric viewed as an electrically switched circuit network. A designer can implement any logical function by configuring the logic blocks and interconnect fabrics. The most abundant reconfigurable resources in FPGAs are regularly structured, dedicated through-routed point-to-point wires. Figure 1.2 depicts the number of logic tiles and wires in modern FPGA device families. FPGAs accommodate multi-millions of abundant wires. However, we observe from the trend in Figure 1.2 that as the logic density linearly grows, the number of wires grows in a similar *linear* manner. This is due to the fact that logic blocks and wires are regularly structured in the Manhattan style [89]. Intuitively, the *number of wires* should grow more than in a linear manner to maintain the point-to-point wirability between (especially long-distance) logic tiles. This means that the *long point-to-point* wires in FPGA become increasingly limited. We are motivated by this trend to devise efficient utilization of existing (rich but increasingly limited) wiring resources in modern FPGAs.

1.1.2 Overlay interconnects

The communication functionalities that constitute overlay interconnects on top of FPGA fabrics are categorized by following:

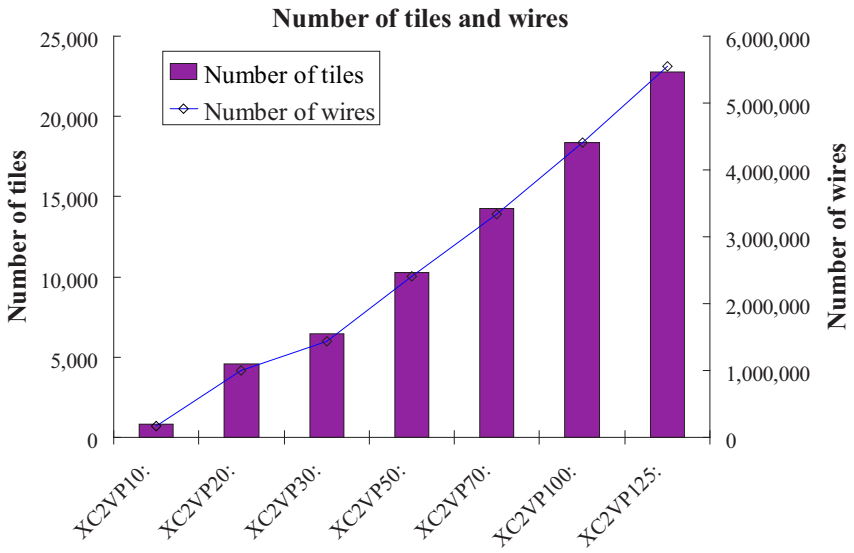


Figure 1.2: Number of wires and logic tiles in Virtex-II Pro devices [64][65].

Point-to-point interconnects: The point-to-point (P2P) interconnect is defined as ad-hoc dedicated links. The P2P interconnect establishes signal wires between the nodes. Since the dedicated links are established without traffic congestion, it does not require arbitration. The P2P wires are inherent interconnects for intra-IP[1] interconnects. Since the FPGA fabric contains bit-level wires, the P2P signal wires can be suitably implemented for intra-IP interconnects. The P2P interconnect is often utilized for the inter-IP communication. A widely used example for the inter-IP communication is the Fast Simplex Link (FSL) for MicroBlaze processors [95]. For inter-IP communication, the P2P interconnect is often suitably mapped onto the FPGAs when the physical distance of the wire is relatively short and the wiring congestions do not occur. In many cases, however, performance can be degraded because of the long wire length and the delay variation. This possibly limits the scalability of the P2P interconnect due to wiring congestion when the number of nodes increases and when the nodes are interconnected with long wires.

Shared bus: A shared bus is defined as an interconnect that establishes a global (dedicated) line after a single arbitration stage. When there are multiple requests from multiple nodes, the central arbiter arbitrates the requests. Subsequently, a ded-

[1] An IP can be a computational processing module or a network module. In this thesis, an IP refers to the computational processing or storage module unless it is stated by a *network IP*.

icated link is established between the nodes, while other nodes wait until the shared link becomes available again. Shared buses are inexpensive and have been widely used in practice, because they provide an adequate performance with a low area cost especially when the interconnect size is small and the traffic pattern is fairly sparse. To improve the performance, the state-of-the-art bus provides sophisticated features, such as pipelining, burst transfer, multiple outstanding transactions, and out-of-order transactions. However, the traditional shared bus is sequentially operated. The network performance is degraded because only one transaction is possible at a time. In other words, shared buses are limited in scalability in terms of performance. Table 1.1 shows the modern Xilinx on-chip buses. In these shared buses, the maximum number of masters (or slaves) is limited. A typical number of IPs connected to the PLB and OPB buses is between 2 and 8 [94]. The maximum total bandwidth that a bus offers is 800 MB/s for PLB and 500 MB/s for OPB which do not meet the dense traffic requirements [25]. This means that the aggregate bandwidth becomes limited as the number of nodes grows.

Table 1.1: On-chip shared buses in Xilinx FPGAs [94].

Feature	PLB	OPB	DCR	OCM	LMB
Processor	PPC	PPC, MB	PPC	PPC	MB
Data width	64	32	32	32	32
Address width	32	32	10	32	32
Masters (max)	16	16	1	1	1
Slaves (max)	16	16	16	1	16
Data rate (max, MB/s)	800	500	500	1500	500

Crossbar: A crossbar is defined as a switch connecting multiple inputs to multiple outputs in a matrix manner. If the switch has M inputs and N outputs, then a crossbar has a matrix with $M \times N$ crosspoints. The crossbar can be referred to as a matrix of buses and provides parallel transactions. A crossbar is composed of a switch fabric and a scheduler. Compared to shared buses, the performance increases due to the parallel nature of communication. The major problem of the crossbar switch is limited scalability due to a high area cost. Traditional $M \times N$ crossbars require $O(MN)$ wires. This means that the area cost quadratically increases as the number of nodes increases. Figure 1.3 depicts the area of the *i*SLIP crossbar scheduler [63], which is widely used for the commercial crossbar switches. As the number of ports increases, the area of the crossbar increases in an unscalable manner due to the all-to-all interconnects in the crossbar.

Network-on-chip: Network-on-chip (NoC) is defined as a network that establishes segmented (shared) interconnects using a set of routers or switches with a single

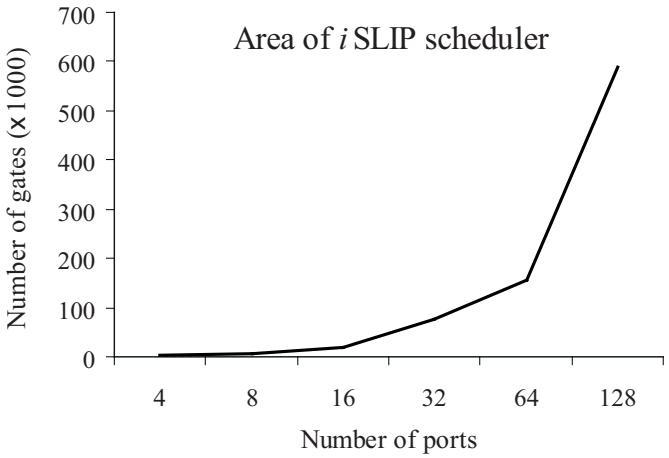


Figure 1.3: Area of *i*SLIP scheduler [63].

or multiple arbitration stages. A NoC can be referred to as a network of crossbars. A NoC achieves the scalability by sharing inter-crossbar wires, over which serialized packets are communicated on a multi-hop basis. The arbitration of the shared busses and crossbar switches does not scale with the number of attached IPs. Moreover, interconnects are physically distributed over the chip and deep-submicron problems related to long wires (such as low speed, signal degradation) complicate timing closure between IPs. NoC addresses these issues by a globally-asynchronous locally-synchronous design style and by replacing long global wires with optimized segmented wires. An arbitration is distributed over segmented links and therefore scalable. An aggregate bandwidth grows in a scalable manner as number of nodes grows. The link speed is unaffected by the number of nodes. In addition, the NoC provides separate abstraction layers such as network and transport layers.

1.2 Scope

In Figure 1.1, logical and physical networks in different layers are depicted. In this thesis, we focus on the *overlay* layer and the *fabric* layer. In this section, we present a model of computation, a platform model, a design parameter, and the targeted technology. First, in the algorithm layer, we target streaming applications in the media and telecommunication domains, because these applications are of practical importance. We consider the Kahn process network (KPN) [36] as a model of parallel computation because the KPN is suitable for the streaming

applications. KPN is a network of concurrent processes that communicate over FIFO (First In, First Out) channels and synchronize by a status of the FIFO [36]. Second, in the platform layer, we consider reconfigurable SoC platform. We consider that the reconfigurable SoC is based on the master-slave architecture, because the master-slave architecture is widely utilized in modern system on a chip. Third, in the overlay layer, we mainly focus on topology as a design parameter, since a network topology plays a key role for the performance and area cost in the modern reconfigurable platform. We aim to design and implement on-demand topology for the state-of-the-art and future generations of reconfigurable hardware. Fourth, in the fabric layer, we target modern FPGA technology. While our approach can be targeted to reconfigurable hardware in general, we specifically consider fine-grained, Manhattan-style FPGA interconnect technology [95]. It can be noted that a comparison between different interconnects is out of scope, since it depends on a particular application, a topology, and traffic requirements [16][30].

1.3 Problem statements

In this section, we describe the problem statements that we aim to solve in the above-mentioned scope.

Soft overlay interconnects: As described in the previous section, an inter-IP communication functionality is implemented as shared buses, crossbars, NoCs, or point-to-point interconnects. In most cases, however, the general-purpose interconnects exploit neither the traffic patterns that given applications exhibit nor the underlying technologies. In other words, the general-purpose soft interconnects do not efficiently utilize the available communication resources in FPGAs. Furthermore, the static or dynamic reconfigurability of FPGAs is desired to be efficiently utilized. A subsequently arising question is:

How can we exploit the static or dynamic reconfigurability in the state-of-the-art FPGAs to efficiently develop application-specific soft interconnects?

When the static reconfiguration is exploited, the soft interconnect should be application-specific to reduce the area cost and/or increase performance. When the dynamic reconfiguration is exploited, the reconfiguration time is often a bottleneck. As a chip density grows, the configuration bitstream size for the entire chip increases accordingly [95]. It is increasingly desired to adapt to the new communication behavior using a small partial bitstreams rather than storing many

large complete bitstreams. Therefore, it is an important problem to efficiently adapt to applications and reduce the reconfiguration overhead.

Interconnect fabric: The fine-grained reconfigurability is a valuable asset to implement any functionality with the desired granularity. However, FPGAs are slow compared to their ASIC counterpart. This is mainly due to the low performance of reconfigurable interconnect fabrics. When the overlay interconnect is mapped onto the reconfigurable resources such as bit-level interconnects and look-up tables (LUTs), performance is degraded due to the long wire length and the delay variation. Additionally, significant computational resources such as LUTs are utilized for communication purposes. Moreover, an inter-IP communication is mostly required to be coarse grained. It can be noted that the interconnect fabric in modern FPGAs does not distinguish between intra-IP and inter-IP interconnects. Therefore, these underlying bit-level wires in the FPGAs are not as efficient for the inter-IP communications as for intra-IP interconnects. A subsequently arising question is:

How can we improve on-chip network performance and reduce the area cost in the FPGA fabric itself?

Due to the mentioned different requirements, inter-IP and intra-IP interconnects should be designed differently. Therefore, it is an important problem to devise the interconnect fabric specially for the inter-IP communication.

1.4 Design objectives and methodologies

Our objective is to reduce the area cost and increase the performance for the inter-IP communication by solving the afore-mentioned problems. To achieve this, we develop adaptive soft interconnects and devise high-performance interconnect fabrics. Our approach is that the logical and physical networks in different layers are *as close as possible*, such that the performance of a given application can be improved. We target (but not limit ourselves to) streaming applications in the media and telecommunication domains. This section presents the main design objectives and proposed methodologies in the context of a design of the adaptive interconnects in the FPGA.

Reducing area cost: The area cost of the general-purpose crossbar switch and NoC is a main bottleneck in FPGAs. Our method to reduce the area cost of these

general-purpose interconnects is to *establish only necessary network resources* that an application requires. To achieve this, we present customization techniques for the soft crossbar as well as the NoC.

Increasing performance: Network performance[2] is usually measured by throughput and latency. Our approach to increase the network performance is to replace the bit-level wire fabric for the inter-IP communication by *coarse-grained hardwired interconnects*. By hardwiring the inter-IP interconnect fabric, the interconnect does not occupy the reconfigurable logic resources, such as look-up tables (LUTs).

Enhancing adaptivity: We aim to maintain adaptability while the reconfiguration overhead is mitigated. Our method to achieve the adaptability is to *statically or dynamically customize the topology* that applications require. We are motivated by the fact that different applications require different topologies as depicted in Figure 1.4. In addition, when the entire system is configured, the configuration bitstream size for the largest devices exceeds 45 Mbits [95] as depicted in Figure 1.5. Our approach to dynamically reconfigure topologies is to partially update bitstreams for the reconfigurable interconnects.

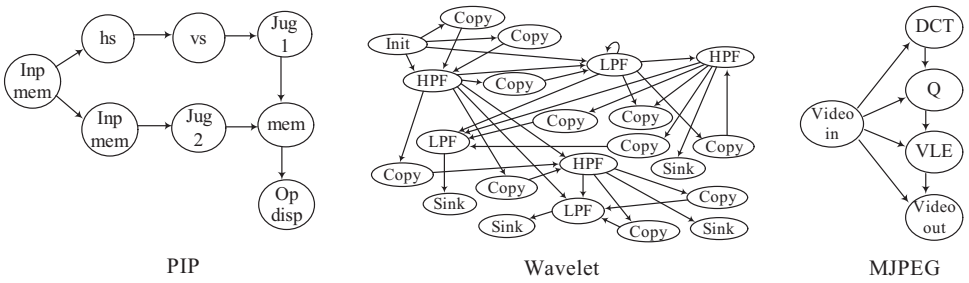


Figure 1.4: Logical topologies for different applications.

1.5 Contributions

The main contributions are summarized as follows:

- **Soft and hard interconnects**: We presented a trade-off study between general-purpose/application-specific and soft/hard interconnects. To compare soft and hard interconnects, we conducted a queuing performance anal-

[2] In this thesis, performance refers to network latency and throughput.

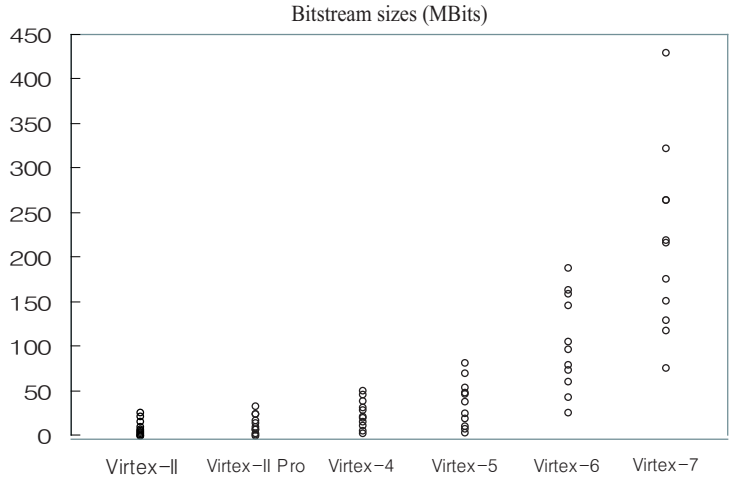


Figure 1.5: Configuration bitstream sizes of different generations of the FPGA devices [95]. Small circles indicate device products in each generation of FPGAs.

ysis. From our analysis and implementation, we presented that soft overlay networks should be (statically or dynamically) application-specific to maintain flexibility and reduce the cost. Additionally, we discussed the general advantages of the hardwired interconnect fabric. Despite of certain loss of flexibility, we demonstrated that hardwired interconnect fabric provides better scalability and performance than the soft networks.

- **Customizing soft overlay crossbar interconnects**: We proposed a topology optimization technique to implement crossbar switches and schedulers, using which the crossbar provides *identical* physical topologies to arbitrary topologies that an application requires. Our design technique is generic such that arbitrary topologies can be implemented for given applications. Specifically, we proposed a custom switch that establishes only necessary interconnects and the custom parallel scheduler (CPS) that accommodates only necessary arbiters for the established custom switch. In addition, we proposed a custom parallel scheduler with shared arbitration scheme (SCPS). Compared to the CPS, the area cost of the SCPS can be further reduced. The SCPS alleviates the scalability problem of full crossbar schedulers by sharing wires. Additionally, the presented custom crossbars have been verified by a prototype. The prototype results indicate that our custom crossbar network increases performance, significantly reduces the area (in the functional and configuration layers) and the power consumption, compared to reference crossbars.

- **Customizing soft overlay circuit-switched NoC**: We proposed a topology customization technique for the soft NoCs to reduce the area cost. Using our table-based technique, only necessary inter-router and intra-router network resources are established. As a result, our experiment indicates that 71% of the area is reduced when compared to the general-purpose soft NoC.

- **Utilizing the partial reconfiguration technique to implement on-demand topology**: We presented a novel use of partial reconfiguration technique to implement on-demand network topologies of dynamically reconfigurable FPGA interconnects. We analyzed the wiring resources in the functional plane. We presented that arbitrary topologies can be realized by updating a partial bitstream for the reconfigurable point-to-point (ρ-P2P) native interconnects. The experiments on the Virtex-II Pro device indicate that the utilization of our ρ-P2P interconnects is feasible and the topology reconfiguration latency can be significantly reduced using a partial reconfiguration technique.

- **Hardwiring crossbar interconnect fabric**: We proposed that crossbars are built in FPGAs to increase the inter-IP communication performance. We described and quantified the general advantages of the hardwired interconnect fabric in terms of the functional performance, area, granularity, wire delay, wire variation, partial reconfiguration time, and resource utilization. Considering a soft crossbar as a reference, an analysis was conducted for the MJPEG application to evaluate hardwired crossbar fabric. As a result, the hardwired crossbar is significantly better in throughput and system throughput, compared to the soft crossbar.

- **Analyzing hardwired circuit-switched NoC interconnect fabric**: We presented to use scalable hardwired circuit-switched NoC interconnect fabric. We presented an analysis, a simulation, and an implementation of soft and hardwired NoCs. We derived an approximated delay and applied the Jackson's queuing model to derive the relative network performance of (virtual) circuit switched NoCs. The analysis and the simulation results indicate that hardwired NoC provides 4.2$\times$ better network latency for the MJPEG task graph, when compared to soft NoC. We showed that the configuration memory and the on-chip logic resources are better utilized by hardwiring the inter-IP network.

1.6 Thesis overview

In the remainder of this thesis, we present various interconnects from the following perspective:

General-purpose versus application-specific: Considering general-purpose interconnects as references, we design and implement application-specific interconnects. As depicted in Figure 1.6, we foresee that the application-specific interconnects provide better performance per cost.

Hard versus soft: A hardwired network is expected to provide better performance than soft interconnects as sketched in Figure 1.6. We present hardwired and soft crossbars as well as NoCs.

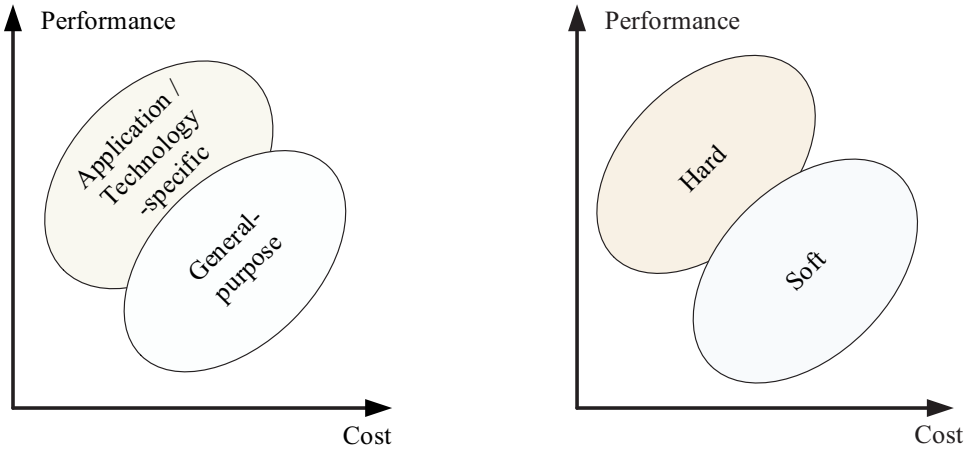


Figure 1.6: Performance and cost of application-specific, general-purpose, hard, and soft interconnects.

The chapters in this thesis are organized as shown in Table 1.2.

Chapter 2 describes the background necessary to better understand topics in this thesis. First, the interconnect fabric in the Xilinx FPGA is studied. Second, the overlay interconnects are summarized. Third, we discuss a model of the reconfigurable platform. Fourth, a general background on the queuing performance analysis is described. Finally, the related work is surveyed, where existing soft/hard, general-purpose/application-specific interconnects are studied.

Table 1.2: Thesis organization.

Type	Application	Layer	Technology	Chapters
Crossbar	application-specific	overlay	soft, static	3
Point-to-point	application-specific	overlay	soft, dynamic	4
Crossbar	general-purpose	fabric	hard	5
NoC	application-specific	overlay	soft, static	6
NoC	general-purpose	fabric	hard	6
Background				2
Experimental results				7
Conclusions and future work				8

Chapter 3 presents an application-specific soft overlay crossbar interconnects. We present an implementation of a crossbar that is customized at design time for given applications. Our method to construct on-demand topologies is presented, using which the crossbar switch and the schedulers are customized.

Chapter 4 presents the dynamically reconfigurable soft overlay point-to-point interconnect. First, we present a wiring analysis and several motivational examples. Second, we present our topology implementation and describe the experiments.

Chapter 5 presents our hardwired crossbar interconnect fabric. First, we present the general advantages of hardwired interconnect fabric. Second, these advantages are quantified by an analysis and an implementation. Finally, we compare the hardwired crossbar with soft crossbar interconnect.

Chapter 6 presents the soft and hard NoC. First, we present our topology customization technique for an application-specific soft overlay NoC. Second, we present the hardwired NoC interconnect fabric. Finally, we analyze the network performance by conducting a case study.

Chapter 7 presents the implementation results of the presented interconnects.

Chapter 8 concludes the thesis by summarizing our investigations, discussing our main contributions, and proposing future research directions.

Chapter 2

On-chip Interconnects Background

As *described in Chapter 1, our objective is to reduce the cost and increase the performance of the inter-IP communication in the state-of-the-art and future generations of reconfigurable hardware. This chapter describes a general background and a literature survey on various on-chip interconnection networks. First, we need to study the underlying fabric of the reconfigurable hardware. In our work, we consider the Xilinx Virtex-II Pro as a target device. We review popular crossbars and NoCs as overlay interconnects to map onto the targeted FPGA. Second, a model of computation from the system's perspective must be determined. We utilize the Kahn process network (KPN) as a model of computation for the SoC platform. Third, we determine a model to analyze hard and soft interconnects.*

Section 2.1 describes the interconnect fabric structure of the targeted Virtex-II Pro device. Section 2.2 reviews the conventional crossbars and their scheduling schemes. Additionally, we describe details of NoCs considering the Æthereal [48] as an example. Section 2.3 describes a KPN-based platform model and the tool chain that we use. Section 2.4 reviews a general queuing analysis. We classify the surveyed networks based on the targeted technology and the application. Finally, Section 2.7 summarizes and concludes this chapter.

2.1 FPGA fabric

Figure 2.1 depicts a conceptual diagram of an FPGA comprised of a *functional plane* and a *configuration plane*. The functional plane contains the configurable logic blocks (CLBs), the input/output blocks (IOBs), and the reconfigurable interconnects. The configuration plane contains a configuration controller and a datapath including the configuration memory cells. In an actual FPGA, the configuration memory cells and elements in the functional plane are spread over the chip. Each element in the functional plane is configured by writing bitstreams onto associated configuration memory cells in the configuration plane. Typically, configuration memory cells are externally configured from the external memory through the configuration I/O port. Alternatively, configuration memory cells can be internally configured through the internal configuration access port (ICAP) from the functional plane.

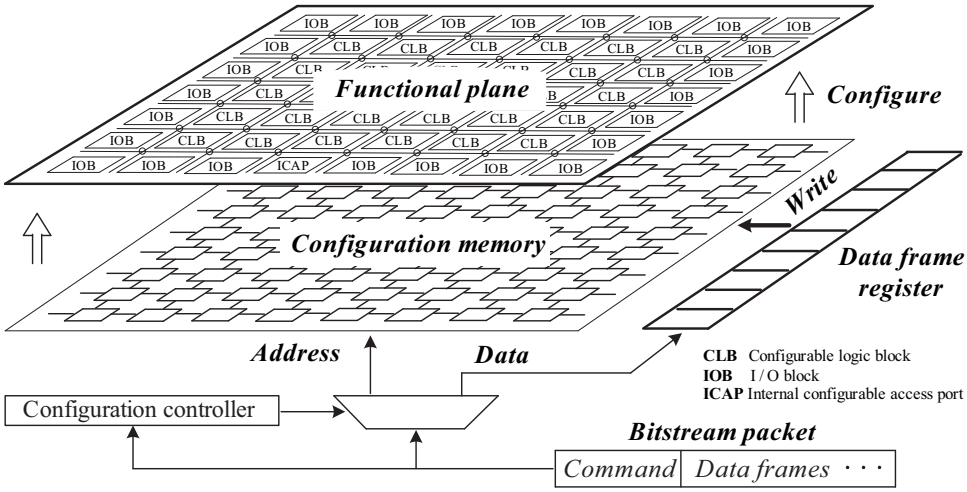


Figure 2.1: Simplified diagram of an FPGA.

2.1.1 Functional plane

Any overlay system (or network) functionality can be mapped on the functional plane by exploiting the reconfigurability of the device. The performance of the functional plane is represented by the latency or throughput. Additionally, the area cost[1] of the functional plane is typically represented by the $\frac{\text{Occupied logic slices}}{\text{Total number of logic slices}}$.

[1]The *cost* is the *area* unless mentioned otherwise.

The functional plane in FPGAs is dominated by millions of regularly structured wire segments. We focus on the interconnect fabric of the Xilinx FPGA as described in the following. Figure 2.2 depicts a primitive CLB cell of the Virtex-II Pro device consisting of wiring resources and logic slices. Wiring resources include the switch-box and various types of wires. As depicted in Figure 2.2, the logic slices geographically occupy much less than 10% of the CLB cell. The rest are wiring resources. A logic slice cell contains look-up tables (LUTs), flip-flops, and associated logic gate resources. It can be noted that even the majority of the logic slice is also composed of wires.

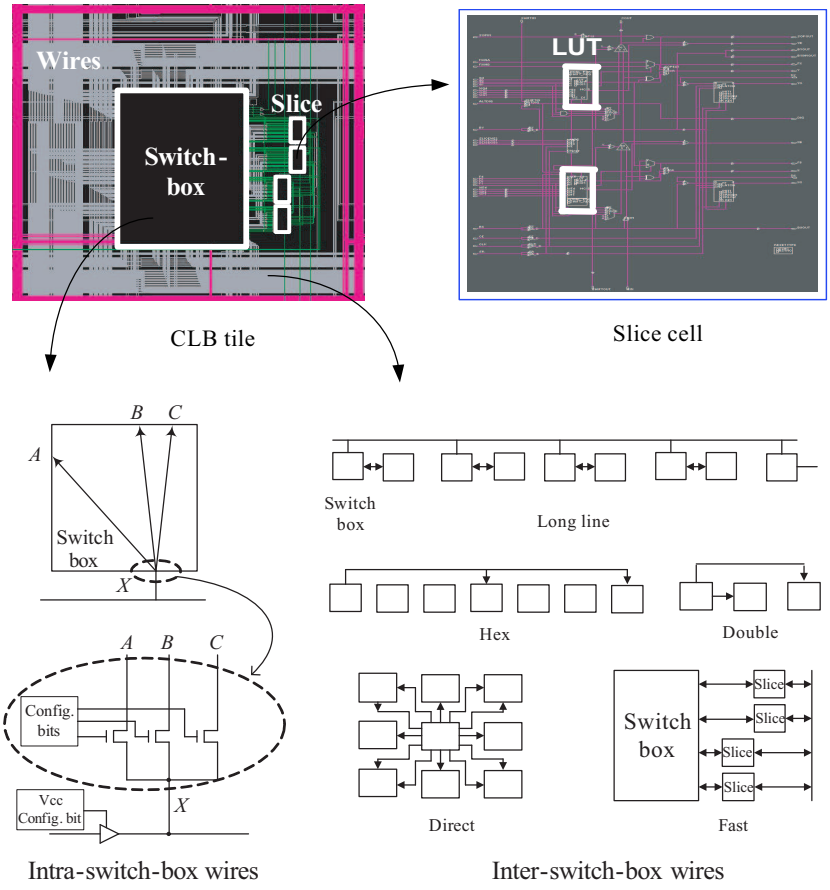


Figure 2.2: Various types of wires in the Xilinx FPGAs [7].

A point-to-point signal wire in the overlay interconnect is mapped onto single or multiple wire segments. The geographical topology is determined after the placement and routing stage. The point-to-point signal wire between look-up tables

is named a *net*. To construct a *net*, a single or multiple *programmable interconnect points (pip)* are configured in the underlying wiring fabric. In a Virtex-series device, there are two classes of wires, namely intra-switch wires and global inter-switch wires. First, there are wires inside the switch-box. The switch-box has hundreds of bit-level I/O pins. Second, there are four types of inter-switch wires[2], namely *direct*, *double*, *hex*, and *long* lines as depicted in Figure 2.2. As the name says, the long lines span the width or height of the entire device. The double line is spaced 2 and 4 switch-boxes apart. The hex line is spaced 6 and 12 switch-boxes apart. The direct wire is utilized to connect neighbors. Summarizing, an abundant number of wires[3] pass through each switch-box.

2.1.2 Configuration plane

The flexibility in the functional plane is realized by the circuitry in the configuration plane. Modern FPGAs have the capability to reconfigure only part(s) of its resources. This operation is called *partial reconfiguration*. In addition, this operation is allowed while the device is operational. This is called *run-time reconfiguration* and allows an efficient utilization of the available resources. The two main methods of reconfiguration are *difference-based* and *module-based* (partial) reconfiguration. In difference-based reconfiguration, small changes to a design are supported by generating a bitstream based only on the differences between the two designs. In module-based reconfiguration, a modular fraction of the FPGA is completely reconfigured and we utilize this method, since the difference-based method is typically allowed only for small design changes, such as LUT programming [5]. For modules that communicate with each other, a special *bus macro* is typically used to allow signals to cross over reconfiguration boundaries. The bus macro is utilized to establish fixed routing paths between modules and guarantee correct inter-module routing. To reconfigure (a portion of) an FPGA, a configuration bitstream is required. The configuration bitstream consists of packets. Each packet contains commands and configuration data that specify the configuration operation. There are two kinds of configuration operations, namely *register writes* and *data frame writes*. The *data frame write* operation is an actual configuration onto the configuration memories. The internal configuration space of the FPGA is partitioned into primitive segments, namely *frames*, which is the smallest load unit [95]. In the configuration plane, the configuration time is derived by following:

$$Config\_time = Number\_frames \times Config\_time\_per\_frame \tag{2.1}$$

[2]Typically, *wires* in an FPGA refer to the *inter-switch wires*.

[3]In Virtex-II Pro, 16 direct, 80 double, 240 hex, and 48 long lines pass through a switch box.

where *Config_time* is the configuration time overhead for a given number of frames *Number_frames*. *Config_time_per_frame* is the configuration time per frame. The cost of the configuration plane can be represented by the bitstream size as derived by following:

$$Config\_cost = Number\_frames \times Size\_frame \tag{2.2}$$

where *Config_cost* is the size (in bits) of the bitstream that physically occupies the configuration memory for a given number of frames *Number_frames*. *Size_frame* is the size of a single frame in bits.

Example: Figure 2.3 depicts the organization of the Virtex-II Pro xc2vp30 device. A single frame contains 206 words and each word is 32 bits wide. The configuration interface operates at 50 MHz. Subsequently, the ICAP controller configures the bitstream in a rate of 400 Mbps (= 8-bit interface $\times$ 50 MHz). The configuration time for a single frame *Config_time_per_frame* can be derived by $\frac{206 \text{ words} \times 32 \text{ bits}}{400 \times 10^6 \text{ bps}}$ = 16.5 μs. A total number of frames is 1756 frames. Therefore, the configuration time for an entire chip can be derived by 1756 frames $\times$ 16.5 us = 29 ms. The configuration memory size is derived by 1756 frames $\times$ 206 words $\times$ 32 bits = 11.6 Mbits.

2.2 Overlay interconnects

In this section, we describe the overlay interconnects that will be mapped onto the underlying fabric.

2.2.1 Crossbar

Crossbars[4] can communicate transactions from multiple input ports to multiple output ports simultaneously. In the traditional shared bus, access to the bus is given to a single IP at a time. Because of the sequential nature of data transfers, the shared bus has a bandwidth limitation. Therefore, the crossbar performs better than the shared bus. The main components in the crossbar are the FIFO queues, a switch fabric, and a scheduler as depicted in Figure 2.4. First, FIFO queues are used to temporarily store arriving packets before being transferred to output ports. Second, the switch fabric is organized as a matrix to connect input ports to output ports.

[4]The crossbar is widely utilized for interconnects in modern SoCs [1]. A main difference of crossbars in the internet and modern SoC is the granularity of the traffic. Crossbars in the internet work on packets and crossbars in SoCs work on transactions.

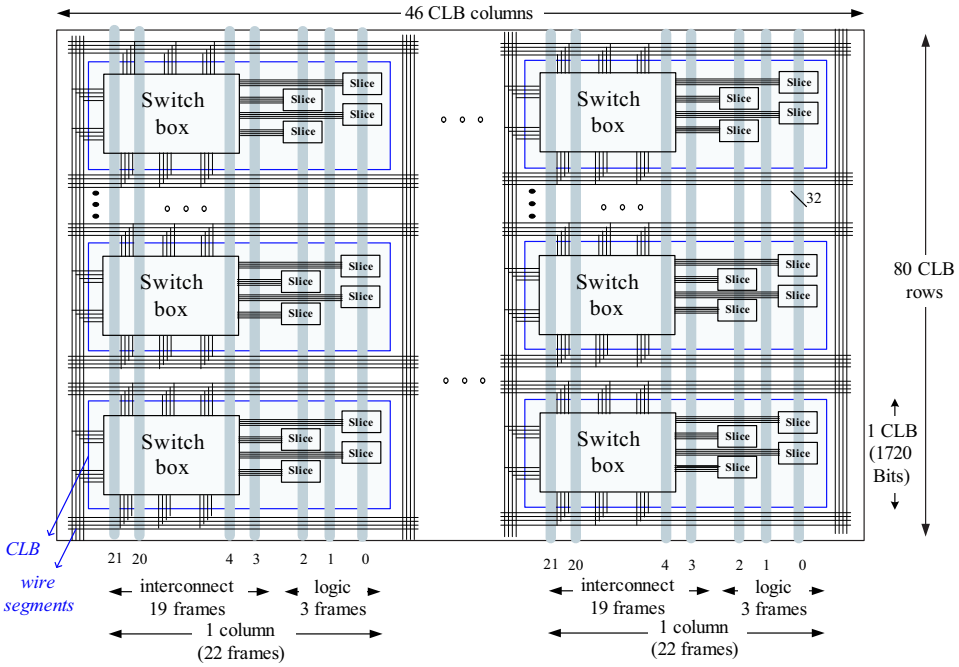
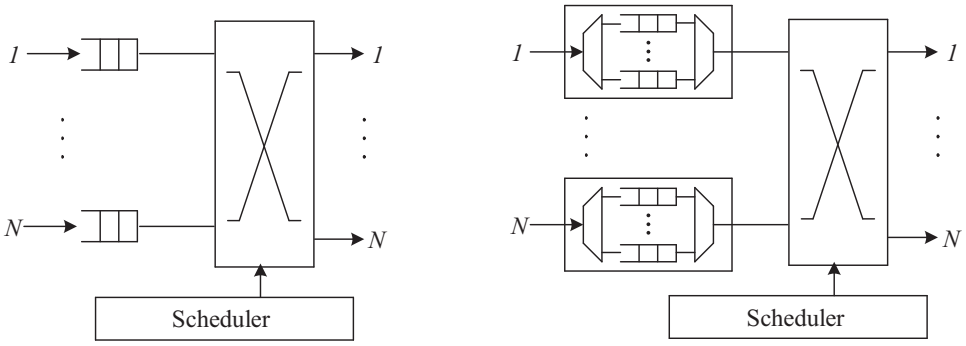


Figure 2.3: Virtex-II Pro xc2vp30 organization.

Each pair of input and output ports has single-hop dedicated routed-through wires. Third, the scheduler is a traffic controller to connect the set of input ports to the set of output ports of a crossbar. The scheduler determines *when* and *which* network resource (such as wires or ports) are allocated to certain transfers.



(1) Input queue (IQ) crossbar (2) Virtual output queue(VOQ) crossbar

Figure 2.4: Input queued crossbars.

Typically, the packets are queued at the input ports before the switching as depicted in Figure 2.4(1). Although the traffic inside the switch fabric of these input queued crossbars is inherently non-blocking, it is susceptible to *head-of-line* (HOL) blocking at input queues due to the possible contention at the output port. Since the scheduler considers the packet only when it reaches the head of the FIFO queue, all packets must wait behind the contended packet. This occurs when the scheduler uses a shared FIFO at the input port for incoming packets. In this case, performance is degraded especially when the traffic pattern is dense. The HOL blocking is reduced in the crossbar architecture by maintaining dedicated FIFO buffers for each input-output port pair. This is called a *virtual output queued* (VOQ) crossbar as depicted in Figure 2.4(2). Rather than maintaining a single shared FIFO queue for all packets, each input port maintains a separate queue for each output port. Maximally, there can be a total of N^2 input queues for the $N \times N$ crossbar. The performance of the VOQ crossbar highly relies on the scheduling algorithm. Typically, the scheduler performs its matching in a three step process, known as the *Request-Grant-Accept* (RGA) handshaking protocol. With suitable scheduling algorithms, an input queued switch using virtual output queueing can increase the throughput. The most well-known RGA-based algorithm is the *i*SLIP [63] used by CISCO routers. The *i*SLIP is based on the round-robin arbitration and is operated as follows:

1. *Request*: When a new packet arrives, each input port sends a request to the scheduler whether or not it has packets to be transmitted to the certain output port. Figure 2.5 depicts an example of the scheduler for the 4×4 crossbar, where there are 9 requests. As an example, input ports 2 and 3 concurrently request to the output port 1.

2. *Grant*: If the output port is available, the scheduler grants the request. When there are multiple requests to the output port, the requests are arbitrated (based on the grant pointer) and each output port grants one of the requests. As an example in Figure 2.5, output port 1 grants the request from input port 2. Note that output port 4 also grants the request from input port 2. Subsequently, the *i*SLIP scheduler updates its grant pointer to the next of previous grant pointer that was *accepted*.

3. *Accept*: The input port accepts a grant. When there are multiple grants to the same input port, the grants are arbitrated (based on the accept pointer) and the input port accepts one of the received grants. Similar to the grant step, the input port accepts a grant based on round-robin arbitration. Figure 2.5 depicts that the input port 2 has two grants from the output ports 1

and 4. These grants are arbitrated and the grant from the output port 2 is accepted. The *i*SLIP scheduler updates its accept pointer to the next of the previously accepted pointer. Simultaneously, the scheduler sends signals to the switch fabric to configure the input-output matrix. Figure 2.5 depicts that two connections are concurrently matched.

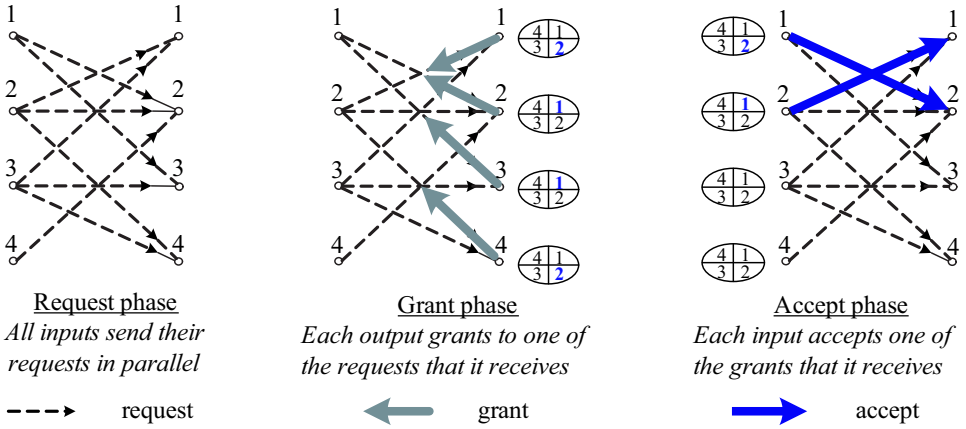


Figure 2.5: Operation in the 4×4 *i*SLIP crossbar scheduler.

The *Request-Grant-Accept* (RGA) operation is repeated whenever there are unmatched requests. For each pointer, the arbiter arbitrates N possible requests (or grants). Therefore, the scheduling algorithm has a time complexity of $O(N^2)$. Figure 2.6 depicts an implementation of an *i*SLIP scheduler [63]. The main components are $2N$ arbiters and the $O(N^2)$ switch wires. The arbiter is implemented using a priority encoder. A main operation in the priority encoder is the comparison. The logic complexity of the switch wires and the arbiter increase as $O(N^2)$ as the number of ports increases.

2.2.2 Network-on-chip

A network-on-chip (NoC) consists of routers (Rs) and network interfaces (NIs) as depicted in Figure 2.7. The IPs communicate with each other using *transactions*. A transaction consists of the request messages and the optional response messages. A request message can be a write or a read request. A response message can be *data* coming back as a result of a read operation or an *acknowledgment* as a result of a write operation. The NIs translate the transaction messages (transport layer in an IP) into packets (network layer in a router network), or vice versa. The routers

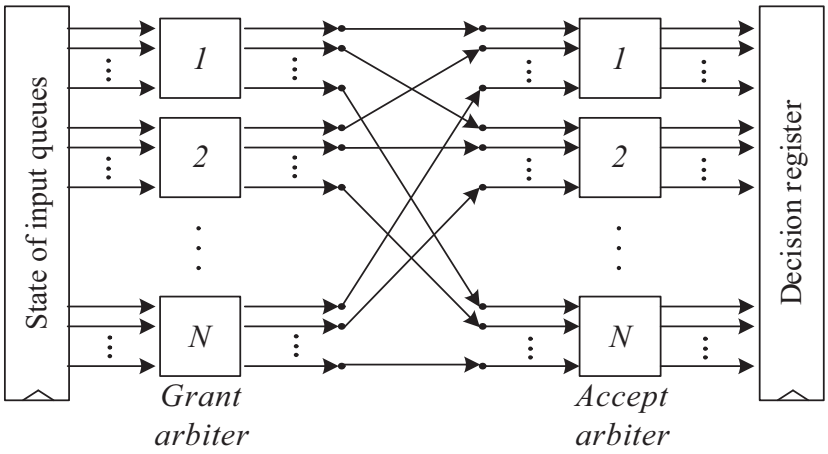


Figure 2.6: Implementation of $N \times N$ *i*SLIP crossbar.

forward packets from one NI to another and are connected among themselves with a certain topology. The NoC can be a packet-switched and circuit-switched network. A typical packet-switched network provides only the best-effort (BE) service. It has problems such as the unpredictable delay and throughput mainly due to blocking of a traffic inside a network. The blocking problem can be alleviated by virtual channels using multiple queues in routers, but this incurs an area cost [90]. Consequently, we focus on a circuit-switched NoC. We consider the Æthereal NoC [48] as it provides the guaranteed throughput (GT) and many real-time applications require such a predictable performance. In the following, the NIs and the routers of the Æthereal NoC are described.

Network interface [15]: In Æthereal, transactions are performed on *connections*. The connection is defined as the bi-directional logical link between IPs. In other words, the connections represent the peer-to-peer *logical topology* required by a system designer. A connection consists of a request channel and an optional response channel. The NIs are responsible for the implementation of the connections. The NI offers a standard interface (for example, AXI) and transport-layer communication services to the IP modules. Guarantees are obtained by means of *TDM slot reservations* (see below). The design of an NI is split in two parts, namely the NI *shell* at the IP side and the NI *kernel* at the network side [15]. The NI shell receives a transaction from an IP and converts it into a sequentialized pieces of message. The NI kernel receives the message from the NI shell, packetizes, and transports them to the router network. Packets may be of different lengths

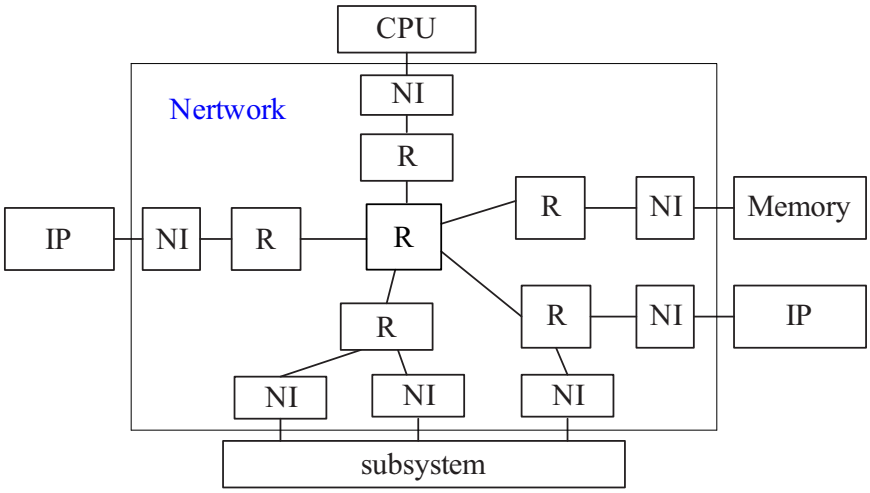


Figure 2.7: NoC example.

and can be further split into *flits*, the minimum flow control unit. The architecture in Figure 2.8 depicts an example NI kernel with two ports and two connections.

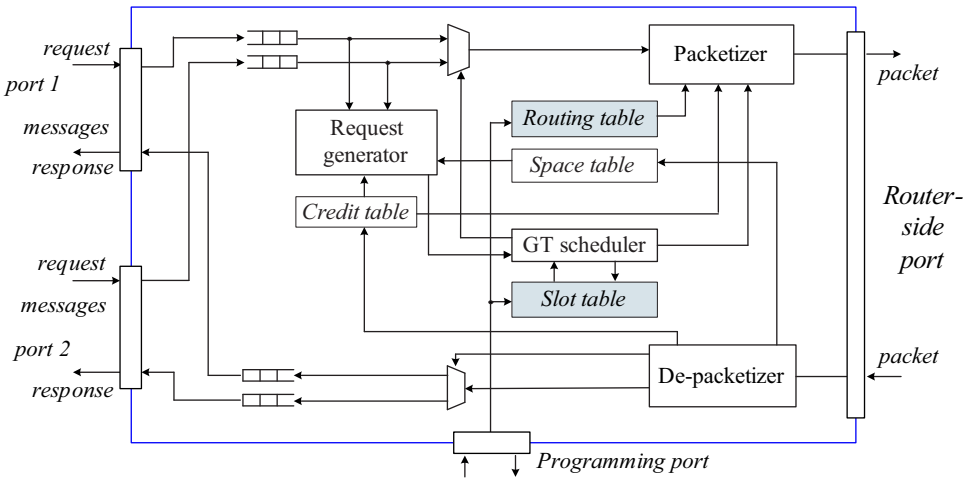


Figure 2.8: Network interface (NI kernel).

A connection consists of one request channel and one response channel. The FIFO size and the maximum number of connections are determined at design time. A channel can be individually *programmed* in terms of a slot reservation (in the *Slot*

table) and routing path information (in the *Routing table*)[15]. In this context, the programming NoC is a register-write operation to set up the *connections* and *routing paths* in the NI using the memory-mapped IO (MMIO) ports. For each channel, there is a counter tracking the available *buffer space*, namely a *credit*, of the FIFOs of the source and the remote NIs. In the *Credit table*, an available buffer space value of the *remote* NI is stored to notify to the source NI. In the *Space table*, the local space credit value of the source NI is stored to notify to the remote NI. This end-to-end flow control ensures that packet is sent only if there is enough space in the remote queue. Whenever a source queue contains a sendable amount of data, the request generator issues a signal specifying that the source queue can be scheduled. A scheduler arbitrates the channels that have data to be transmitted. The scheduler checks whether the current slot is reserved for a GT channel. If the current slot is reserved and there is sendable data in the queue, the source queue is scheduled. After the queue is scheduled, the data is packetized and sent to the routers.

Router: The router is responsible for forwarding the packets to the designated next hop or the destination. To achieve this, the Æthereal router uses the source routing in that the packet header contains information about the intermediate paths to the destination. The router is organized as packet header parsing units, FIFO queues, switch wires, and the controller, as depicted in Figure 2.9.

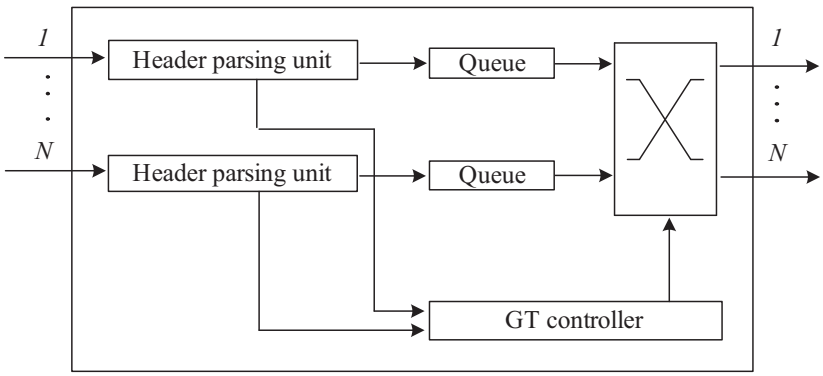


Figure 2.9: Router in Æthereal.

2.2.3 Guaranteed performance and design flow of NoC

The predictable performance of a network is desirable for real-time applications. In this section, we describe the performance guarantees and the design flow of the Æthereal NoC.

Contention-free routing and guaranteed throughput: Guaranteeing a certain level of performance (in terms of throughput and latency) for a communication requires resource reservation in the NoC. The Æthereal NoC uses contention-free routing, or pipelined time-division-multiplexed (TDM) circuit switching to implement the guaranteed throughput [48]. The guaranteed performance of GT connections results from wire and buffer reservations. Figure 2.10 depicts a contention-free routing with a snapshot of a router network and the corresponding slot tables [48]. R represents routers with input ports i and output ports o. T represents slot tables with a size of S. S denotes the total number of entries (for an output port) of a slot in a table. In a slot table T, rows indicate time slots s and columns indicate output ports o of a router. In a slot s, a network node (that is a router or a network interface) can read and write at most one block of data per input and output ports, respectively. In the next slot $(s + 1)$ modulo S, the network node writes the read blocks to their appropriate output ports. The slot table entries map outputs to inputs for every slot: $T(s, o) = i$, which means that blocks from input i (if present) proceed to output o at each $s + kS$ slot, where k is an integer. In Figure 2.10, the network contains three routers, R_1, R_2, and R_3 at slot s = 2, where s is the pointer to the third entry in each table. The size of the slot table S is 4 and Figure 2.10 depicts only the relevant columns. The three arrows **a**, **b**, and **c** represent connections. The three circles labelled a, b, and c represent blocks on the corresponding connections. Router R_1 switches block b from input i_1 to output o_2, as slot table $T_1(s = 2, o = o_2) = i_1$ indicates. Similarly, R_2 switches block a to output o_2, and R_3 switches block c to output o_1. In this way, the pipelined multi-hop NoC is operated. Accordingly, the network contention is avoided because there is at most one input block per output for each slot. It can be noted that a connection is arbitrated only *once* at the NI.

Æthereal NoC design flow: Figure 2.11 depicts the Æthereal NoC design flow [49]. Taking the communication requirements as an input, the tool flow generates the NoC hardware and software instances. The automatic design flow is split into three main steps: *hardware generation*, *software configuration*, and *performance verification*, depicted as boxes in Figure 2.11. An input of the design flow is the communication requirements of the given application, represented by weighted task

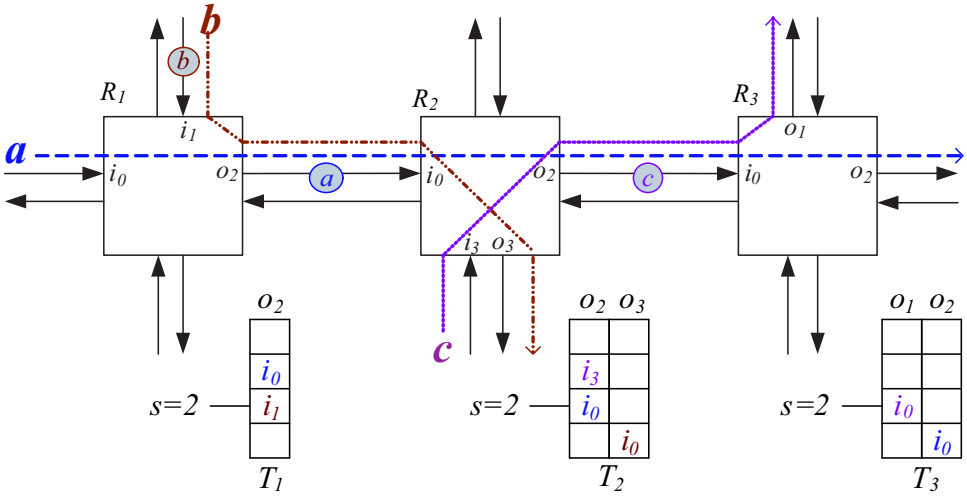


Figure 2.10: Contention-free routing: network of three routers (R_1, R_2, and R_3) at slot s = 2, with corresponding slot tables (T_1, T_2, and T_3) [48].

graphs. In the weighted task graphs, nodes, edges, weights correspond to IPs, connections, bandwidth (and latency) requirements, respectively. The input also specifies a list of all IPs connected to the router network and the IP ports. Each port has a number of attributes, such as the protocol (for example, AXI) and data word width.

The first step is the NoC *hardware generation*. Taking the above-mentioned input specification, hardware instances of routers and network interfaces are generated. Based on the specified application, network parameters are determined. As an example, the number of slots in the TDM table and the physical topology are determined. In the first step, the mapping is also performed to assign IP ports to NI ports. As a result, the tool produces RTL VHDL descriptions and the NoC can be synthesized for the back-end hardware implementation. The second step is the NoC *software configuration*. Using the output of the first step, the NoC configuration tool computes the run-time software that contains all the information to program the hardware. For each connection, a list of routing paths is generated. In addition, slots are allocated using a heuristic using a combination of each connection path length and required bandwidth. The third step is the NoC *performance verification*. Given the output of the first and second steps, the verification tool computes the worst-case (minimum) throughput, (maximum) latency, and (minimum) buffer sizes per GT connection. The verification step checks whether the NoC topology and configuration are guaranteed to fulfill the application requirements. If the re-

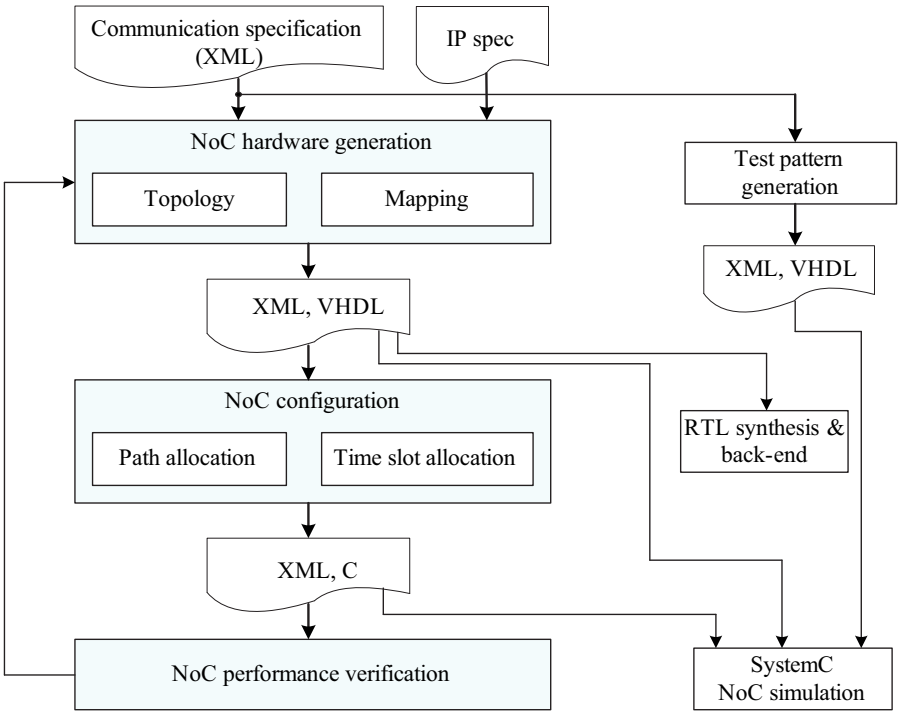


Figure 2.11: Æthereal NoC design flow [49].

quirements are not met, the design flow can be back-annotated. Finally, to assess the average performance for a particular execution trace, the SystemC simulation is conducted based on the generated topology, mapping, NoC configuration, and IP configuration. The NoC can be simulated at the cycle-accurate level.

2.3 System overview

In the previous sections, an underlying fabric and the overlay interconnects were studied. In this section, we describe a SoC platform model that we utilize.

2.3.1 Model of computation

In our work described in this thesis, the Kahn Process Network (KPN) [36] is considered as a model of computation. A KPN is a network of concurrent processes that communicate over unbounded FIFO channels and synchronize by a blocking read [36]. From the communication perspective, a communication channel is

mapped onto a FIFO as depicted in Figure 2.12. In this figure, there are 5 tasks and 7 communication FIFOs.

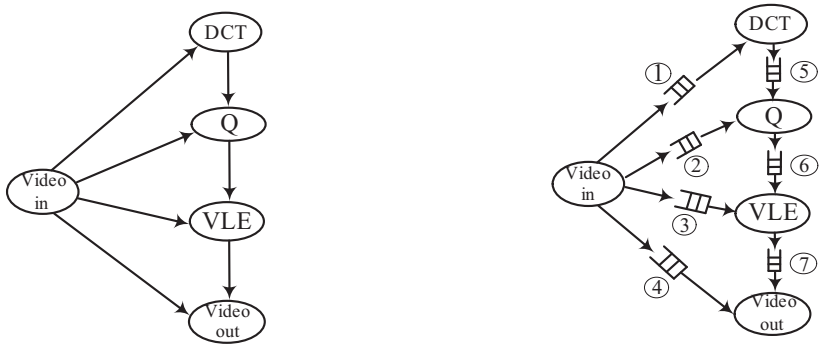


Figure 2.12: The KPN model of computation.

2.3.2 Platform model

The SoC platform is based on the master-slave architecture [39, 40, 41] and is depicted in Figure 2.13. The platform implements the KPN model of computation. It consists of processors and interconnects. A single or multiple tasks are assigned to a processor. The tasks (for a processor) are statically scheduled and the processor sequentially performs the assigned tasks. A communication channel is mapped to a physical FIFO. The transaction is based on the *local write, remote read* scheme. The master processor locally writes data to the local slave FIFOs. The master processor remotely requests data to the remote slave FIFOs. If the remote FIFOs are not empty, the master processor remotely reads the FIFOs. The interconnects forward these *requests* (from the master processor) and *data* (from the slave FIFOs). The KPN is a suitable model of computation on top of the platform, mainly because the synchronization scheme is relatively simple. Processors synchronize only with the full/empty status of the hardware FIFO.

Figure 2.14 depicts a system organization using a crossbar as an example. Figure 2.14(1) depicts a task graph of a 4-node MJPEG application. There are 5 connections (named $\mathbf{F}_1$ to $\mathbf{F}_5$) and each connection is mapped to a physical FIFO. Figure 2.14(2) depicts the system organization, where processors locally write and remotely read. The crossbar transfers *requests* (from processors) and *data* (from FIFOs). As an example, the communication between the processors $\mathbf{P}_1$ and $\mathbf{P}_2$

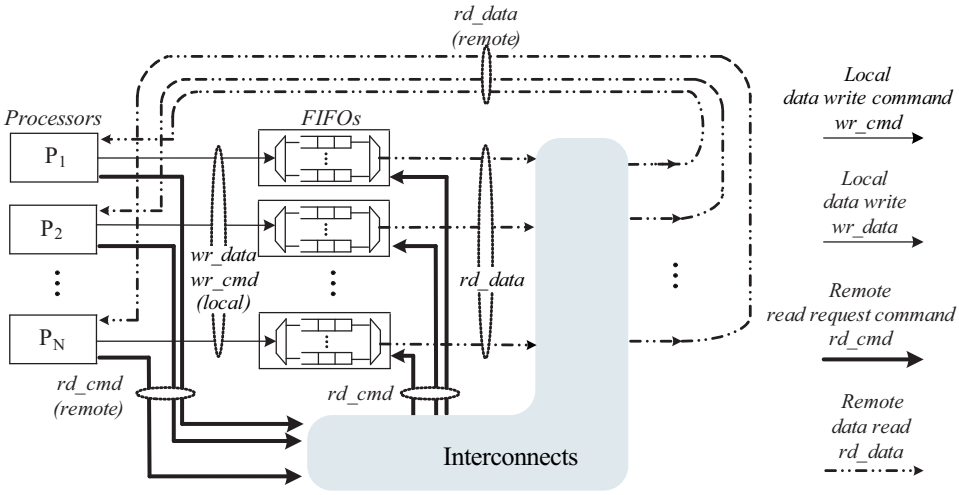
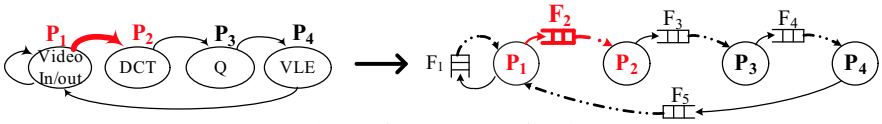


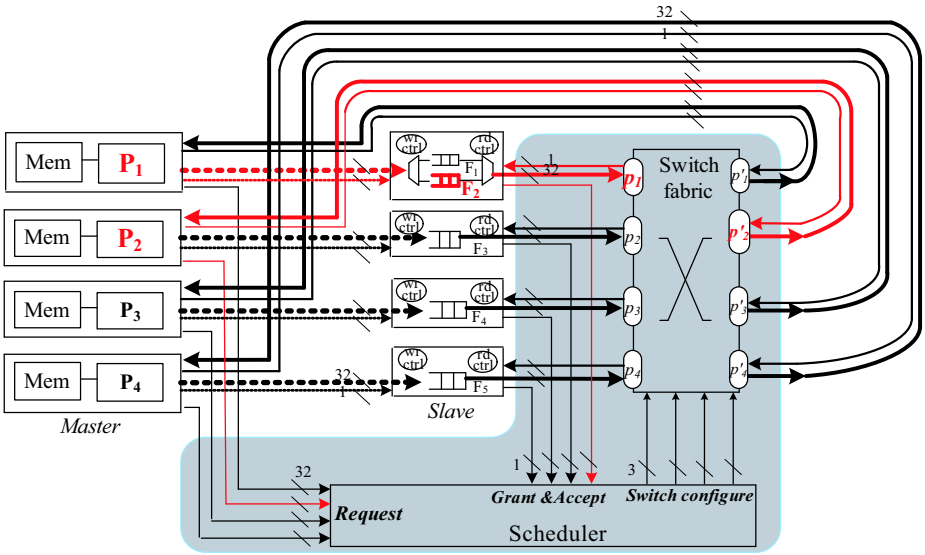
Figure 2.13: A multiprocessor SoC platform model for the KPN.

operates as follows. First, $\mathbf{P}_1$ locally writes data in FIFO $\mathbf{F}_2$. The processor $\mathbf{P}_2$ sends a *request* to the scheduler by designating the target port index p_1 and target FIFO index $\mathbf{F}_2$. The request signal requires $\{$ $\log_2$ (Number of processors) + $\log_2$ (Number of FIFOs in targeted port)$\}$ bits. In this work, we use a 32-bit processor and the request signal contains 32 bits as depicted in Figure 2.14(3). The designated FIFO $\mathbf{F}_2$ responds to $\mathbf{P}_2$ whether the FIFO is empty or not. If the FIFO is not empty, the scheduler generates control signals to establish a communication line between p_1 and p_2' in the switch fabric.

We use the communication controller interface in [41]. Figure 2.15 depicts the detailed system organization to implement the communication between $\mathbf{P}_1$ and $\mathbf{P}_2$. The communication between $\mathbf{P}_1$ and $\mathbf{P}_2$ is depicted by the thick line. The interface controller selects which FIFO to write, based on the signals received from processors. The FIFO controller checks the full/empty status of the FIFO(s) and update the pointers (or FIFO addresses) using read/write counters. Each FIFO has its own read/write controllers. Read and write controllers (denoted by rd_ctrl and wr_ctrl in Figure 2.14) are also depicted in Figure 2.15 in detail. Figure 2.15 also depicts detailed ports to implement p and p'. As an example, $\mathbf{P}_2$ sends read command signal to the port Read'(2) at the processor-side port $p'(2)$. Empty'(2) indicates an empty status signal sent to $\mathbf{P}_2$. At the FIFO-side, Empty(1) indicates an empty status signal sent from FIFO at port p_1.



(1) 4-node MJPEG application



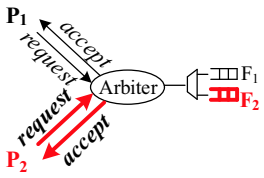
(2) System organization

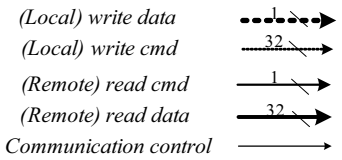
31	30 downto 16	15 downto 0
1 (request), 0(Clear)	Port ID to read	FIFO ID (in targeted port) to read

(3) 32-bit request signal format

① $\mathbf{P_1}$ locally writes $\mathbf{F_2}$ using dedicated local wires
② $\mathbf{P_2}$ requests remote FIFO space in $\mathbf{F_2}$ via crossbar arbiter
③ Crossbar arbitrates
④ Crossbar accepts the request of $\mathbf{P_2}$.
⑤ FIFO $\mathbf{F_2}$ sends data to $\mathbf{P_2}$.

(4) Operation





P : *32-bit processor*
F : *FIFO*
p' : *processor-side port*
p : *FIFO-side port*

rd/wr cmd: *request – accept handshaking*

Figure 2.14: A system organization example.

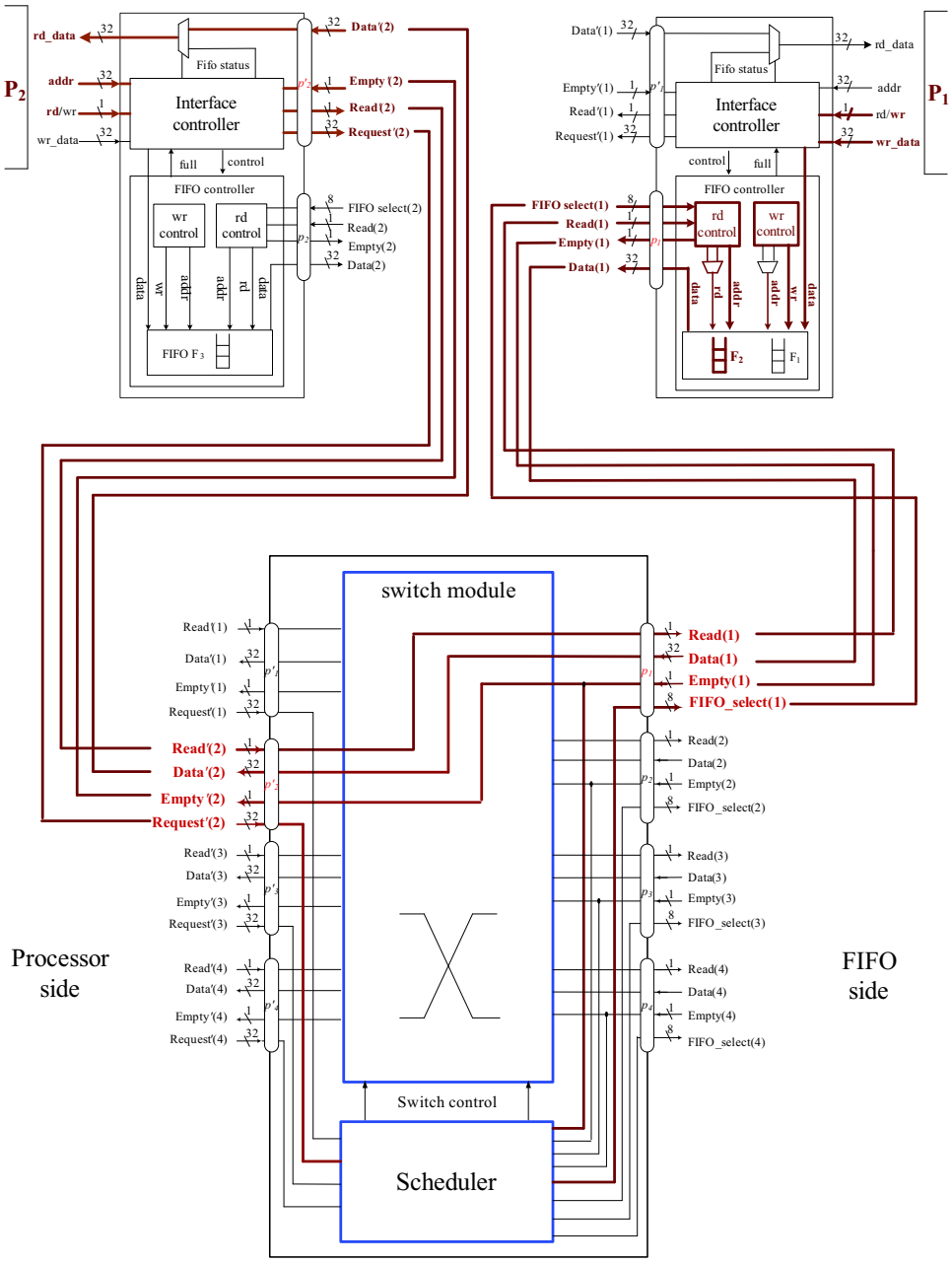


Figure 2.15: A detailed micro-architecture to implement the P_1-P_2 connection.

2.3.3 ESPAM design flow

In Chapter 3, a soft customized crossbar is presented and integrated as a modular communication component in the ESPAM tool chain [39, 40, 41] depicted in Figure 2.16. In ESPAM, 3 input specifications are required, namely *application*, *mapping*, and *platform* specification in XML. An application is specified as a KPN. A KPN specification is automatically generated from a sequential Matlab program using the COMPAAN tool [17]. We define the network topology in the KPN specification as the *logical topology* for an application, which consists of tasks and logical channels. In the mapping specification, a single task or multiple tasks are assigned to a physical processor. In the platform specification, a network type and the port mapping information are specified.

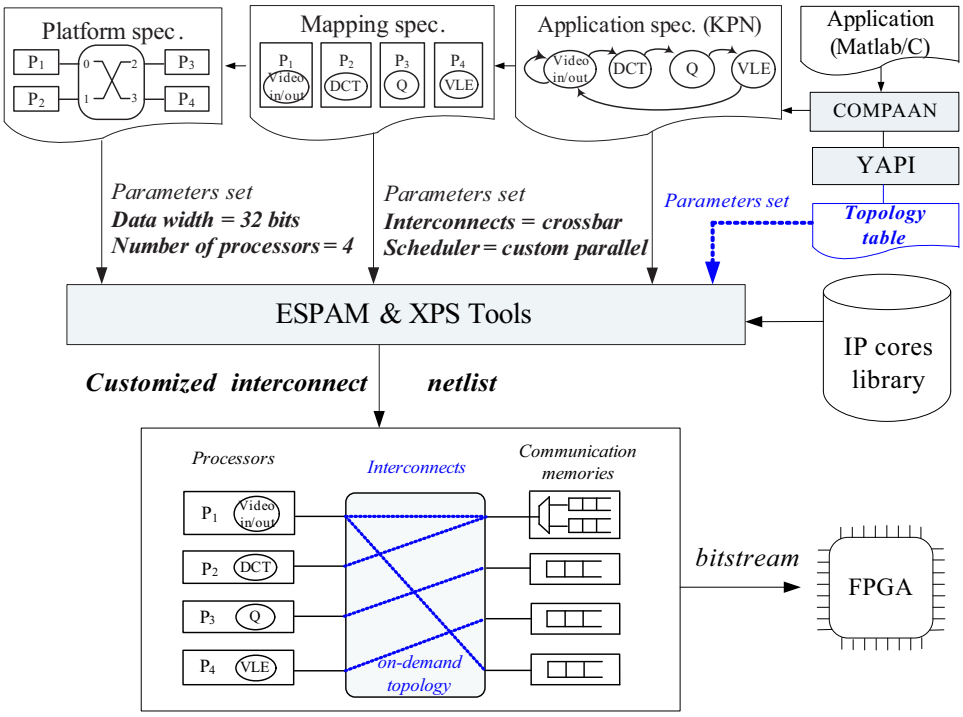


Figure 2.16: ESPAM design flow [39, 40, 41].

Figure 2.16 depicts how customized crossbars can be implemented from a 4-node MJPEG application specification. In the platform specification, four processors are a port-mapped on a crossbar. From the mapping and platform specifications, port-mapped network topology is extracted as a static parameter and passed onto

ESPAM. We define the extracted port-mapped network topology as the *on-demand topology* that an application requires, which consists of processors and physical links. Additionally, the bandwidth information is obtained from the YAPI tool [28]. Subsequently, ESPAM refines the abstract platform model to an elaborate parameterized RTL (hardware) and C/C++ (software) models, which are inputs to the commercial Xilinx Platform Studio (XPS) tool. The XPS tool generates the on-demand netlist (depicted in Figure 2.16) with regard to the parameters passed from the input specifications. Finally, a bitstream is generated for the FPGA prototype board to check the functionality and measure the performance.

2.4 Queuing analysis

Queuing analysis is a widely used modeling method in telecommunication networks and provides a reasonable fit to the reality with relatively simple formulation [73]. We use a queuing model to compare soft and hard on-chip interconnect instances. It can be noted that the analysis of the SoC and telecommunication networks have different implications. First, unlike Internet traffic, traffic information can be extracted from the application specification. This means that a priori logical information such as topology and bandwidth can be exploited for the analysis and design. Second, we usually reuse pre-verified IP components and their specifications. This means that the physical information such as the area, the clock speed, and/or latency of IPs are available at design time. In the following, we describe a general background on the queuing analysis.

2.4.1 Single queueing system

The two central elements of the queueing system are a *server* and a waiting *queue* as depicted in Figure 2.17. Tokens arrive at the queue to be served. A *token* is the primitive communication unit. When a token arrives at the queue, a token is dispatched to the server. If the server is idle, the token is served immediately. When the server has completed serving the token, the token departs. As described in the next sections, we mainly consider the queuing server as an interconnection network, while it can be any entity.

Additionally, Figure 2.17 illustrates parameters associated with a queuing model. The analysis takes the following as an input:

- Arrival rate, λ

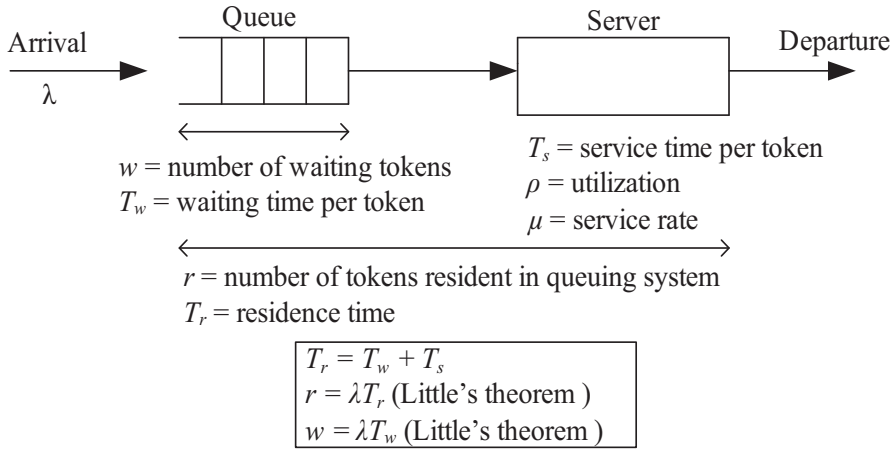


Figure 2.17: A simplified queuing system.

• Service time, T_s

Tokens arrive at the queue at a rate of λ, where the λ is the *number of tokens arriving per second*. At any given time, w tokens wait in the queue, with the mean waiting time T_w. The server handles incoming tokens with a service time T_s. T_s is the time interval between the dispatching of a token to the server and the departure of the token from the server. Utilization ρ is the fraction of time that the server is busy, measured by $\frac{\lambda}{\mu} = \lambda T_s$. The analysis generates the following information as output:

• Number of tokens residing in a system, r
• Time resident in a system, T_r

The number of tokens resident in the system, r, includes the token being served and the tokens waiting in a queue. The residence time T_r is the time that a token spends in the queuing system, waiting and being served. A convenient notation to characterize a queuing model is $X/Y/N$, where X refers to the distribution of the inter-arrival times, Y refers to the distribution of service times, and N refers to the number of servers. The most common distributions are denoted as follows:

G = general independent arrivals or service times
M = negative exponential distribution
D = deterministic arrivals or fixed length service

In many cases, the traffic pattern can vary over time and entails an inter-dependence between tokens. To derive this information, the probability distribution of the time-varying arrival rate and service time is required. Since the required formulas are exceedingly complex, typically simplifying assumptions are made in order to make the analysis tractable. The most important assumption is that the arrival process obeys the *Poisson* distribution. This means that the inter-arrival times are exponential, which is equivalent to stating that the arrivals occur randomly and independently from one another. When the standard deviation is equal to the mean, the service time distribution is exponential and the $M/M/1$ model can be used. With this assumption, many useful results can be obtained as shown in Table 2.1 if the arrival rate and service time are known.

Table 2.1: $M/M/1$ queuing model.

Number of tokens	Residence time
$r = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}$	$T_r = \frac{T_s}{1-\rho} = \frac{1}{\mu-\lambda}$

2.4.2 Network of queues and Jackson's model

In a distributed environment such as an SoC, the system to be analyzed is more complex than an isolated single queueing system. In many cases, the system consists of multiple interconnected queues and constitutes a network of queues (NOQ). Figure 2.18 illustrates this situation. Each node represents an individual queuing system when combined with the interconnecting lines. When the traffic flow adheres to a Poisson distribution and the service times are exponential, the Jackson's model [45] can be used to analyze the average performance of a network of queues. The theorem is based on the following assumptions:

- The queuing network consists of nodes, each of which provides an independent exponential service.
- Tokens arriving from outside the system to any one of the nodes arrive at a Poisson rate.
- Once served at a node, a token goes (immediately) to one of the other nodes with a fixed probability.
- The queue size is sufficiently large to avoid the stall of the data flow.

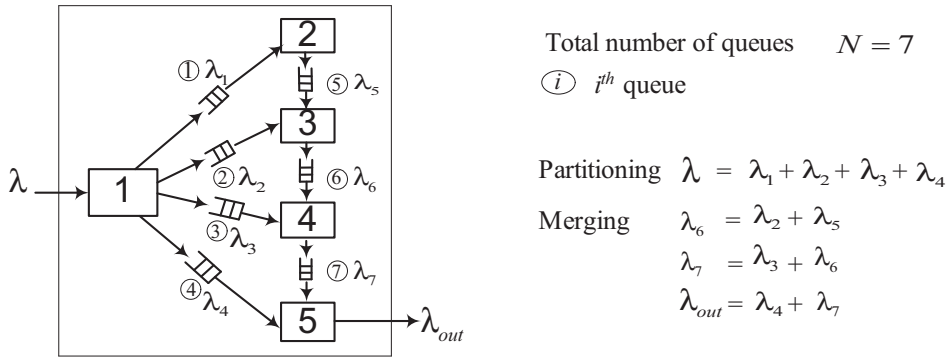


Figure 2.18: Network of queues.

In the Jackson's model, each combination of the node and the link is an independent queuing system, with a Poisson input determined by the principles of partitioning and merging. As an example, in Figure 2.18, the traffic flow is partitioned from the node 1 and merged in the nodes 2 to 5. In addition, the arrival and the departure processes at each node i are Poisson processes with the same rate λ_i. As an example, in Figure 2.18, the Poisson arrival and the departure traffic at node 2 have the same rate, or $\lambda_1 = \lambda_5$. Subsequently, each queueing system can be analyzed separately using the $M/M/1$ model and the results can be combined by ordinary statistical methods. For $M/M/1$ queueing model, the mean residence time in the i^{th} queueing system, $E(T_r^i)$ can be formulated as [26]:

$$E(T_r^i) = \frac{1}{\mu_i - \lambda_i}, \tag{2.3}$$

where μ_i is the service rate and λ_i is the arrival rate for the i^{th} queueing system. The total number of tokens in the entire NOQ, $E(n)$ is [45]:

$$E(n) = \sum_{i=1}^{N} E(n_i), \tag{2.4}$$

where $E(n_i)$ is the number of tokens in the i^{th} queueing system. N is the total number of queues in the NOQ. As an example, N is 7 in Figure 2.18. The entire NOQ also can be treated as a single queueing system. By applying Little's theorem, the mean end-to-end residence time in NOQ, $E(T_r)$ can be represented as:

$$E(T_r) = \frac{E(n)}{\lambda}, \tag{2.5}$$

where λ is the total arrival rate from an external source. Since $E(n) = \sum_{i=1}^{N} E(n_i)$ and $E(n_i) = \lambda_i E(T_r^i) = \frac{\lambda_i}{\mu_i - \lambda_i}$, the mean end-to-end residence time for a token in the NOQ is:

$$E(T_r) = \frac{E(n)}{\lambda} = \frac{1}{\lambda} \sum_{i=1}^{N} E(n_i) = \frac{1}{\lambda} \sum_{i=1}^{N} \frac{\lambda_i}{\mu_i - \lambda_i} \tag{2.6}$$

where the end-to-end residence time corresponds to the *total response time* for a token in the entire network of queues. Equation (2.6) states that the number of tokens in the entire NOQ is the summation of the number of tokens in the individual queueing system. Subsequently, the total response time is derived by dividing the number of tokens (in the NOQ) by the arrival rate of the incoming traffic. In the remainder of this thesis, we use Equation (2.6) to conduct our performance analysis. The Jackson's model can be utilized when the service distributions are independent. In practice, the arrival process to each queue can be correlated to the service process, which makes the subsequent tokens dependent on each other. Though the analysis is an approximate, we use this model (in Chapters 5 and 6) to derive the rough comparison of performance between hard and soft interconnects.

2.4.3 Applying Jackson's model

Given an arrival rate λ_i for each connection in Equation (2.6), the performance of a queueing system can be obtained by deriving a service rate μ_i for the connection. In Figure 2.19(1a), the *Video_In* process is mapped onto the processor P_1 and the *DCT* process is mapped onto the processor P_2. The communication channel in the KPN is mapped onto a FIFO. Figure 2.19(1b) depicts that the processor P_2 reads the data of P_1 and performs *DCT* computation.

Figure 2.19(2) depicts our application of the queuing model. Note that the queueing model is general in that the server can be any entity or it can be a combination of entities. The server depicted in Figure 2.19(2) is a *network* that provides a *transmission service*. In this case, we can derive only the network performance and the computation time is assumed to be zero. Accordingly, the queueing system consists of a queue and a network. A *token* refers to a set of data words, which is our primitive granularity of a communication. The residence time of a single token in a queueing system is denoted as T_{token}. Since the service rate is reciprocal of the residence time in the Jackson's model, the service rate μ_{token} is derived by $\frac{1}{T_{token}}$.

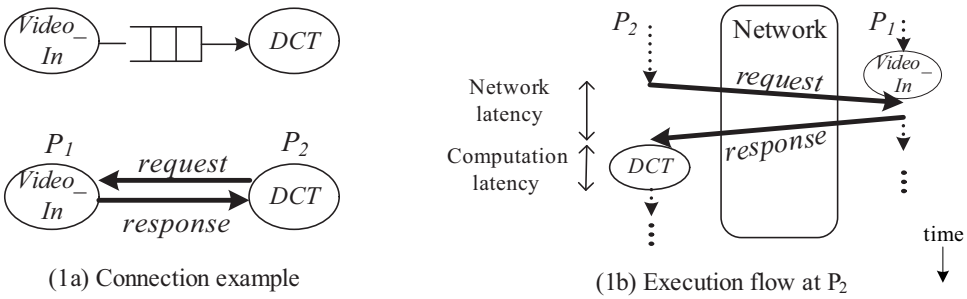


Figure 2.19: Our application of Jackson's model.

2.5 Related work

Recently, a number of on-chip interconnection networks were reported. In this section, we survey the literature and categorize with regards to the targeted technology and type of interconnects as shown in Table 2.2. An overlay interconnect IP is *soft* when after synthesis it is *mapped*, *placed*, and *routed* using *reconfigurable resources* in FPGAs. An overlay interconnect IP is *hard* when it is *implemented* in bare silicon.

2.5.1 Hard interconnects

General-purpose crossbars: The *i*SLIP crossbar [63] is the most popular general-purpose crossbar. Gupta, *et al.*, present in [66] a design and an implementation of fast round-robin *i*SLIP crossbar scheduler for high performance computer networks. In these conventional round-robin schedulers, all-to-all interconnects are established because the traffic pattern is unknown. These conventional full crossbar schedulers accommodate the circuitry to arbitrate all possible requests from all ports.

Table 2.2: Categorization of on-chip interconnects.

Layer	Hard/soft	Application	Type	Sections
Overlay	hard	general-purpose	crossbar	2.5.1
		application-specific	crossbar	
		general-purpose	NoC	
		application-specific	NoC	
	soft	general-purpose	crossbar	2.5.2
		general-purpose	NoC	
		application-specific	NoC	
Fabric	hard	general-purpose	NoC	2.5.3

Application-specific crossbars: Recently, a couple of application-specific crossbars were reported. Loghi, *et al.*, present in [60] the STbus crossbar customization and verified it through simulation. Murali, *et al.*, present in [76] that the STbus crossbar is customized based on the analysis of simulated traffic patterns in windows. In [76], an arbiter of the bus-based custom crossbar is connected to the IP. In [77], Murali, *et al.*, propose a methodology to generate a partial crossbar, in which an application is traced in simulation and the graph clustering technique is used. Pasricha, *et al.*, present in [78] a design method to synthesize an application-specific partial bus matrix is presented. Multi-layer AHB [4], based on the AHB protocol [1], provides an interconnection scheme that enables parallel access paths between multiple masters and slaves in a system.

General-purpose NoCs: Numerous NoCs (surveyed in [85]) employ rigid and general-purpose underlying networks. Examples of such NoCs include aSOC [46], Octagon [31], Proteo [27], Xpipes [56], T-SoC [68], Eclipse [57], QNoC [29], SPIN [8], MANGO [84]. Figure 2.20 summarizes these NoCs in terms of topology/routing, flit sizes, buffering schemes, type of network interfaces, areas per router, peak performances, QoS schemes, and target technologies. Typically, packet routers constitute tiled NoC architectures, where a pair of router and network interface (NI) is connected to an IP. The hard overlay NoCs perform better than the soft overlay NoCs. As Figure 2.20 indicates, however, each NI is implemented for certain protocols *fixed* at design-time. In other words, rigid NIs support only fixed message formats or communication types. Leroy, *et al.*, proposed an NoC using the Spatial-Division Multiplexing (SDM) [13][12]. The SDM technique allocates only a subset of the link wires to a given connection. Messages are serialized on a group of wires in the link. Unlike the TDM, data in the SDM is serialized on a number of wires proportional to the allocated bandwidth. In SDM, an $M \times M$

router with the N bit-wide links would require a $(MN)\times(MN)$ bit switch when the bandwidth is allocated in a bit level. Since the full crossbar inside a router has a high complexity to be used in a SDM router, Multi-stage Interconnection Network (MIN) switches are used in [13][12]. The MIN can reduce the number of crosspoints down to O($N\log_2 N$). In [96], Yang, *et al.*, proposed a dynamically reconfigurable SDM-based NoC, where the number of crosspoints in a router is reduced to O(N) at the cost of certain routability.

NoC	Topology / Routing	Flit Size	Buffering	Interface	Switch Area	Performance	QoS Support	Target
aSOC	2D mesh (scalable) / Determined by application	32 bits	None	n.a.	50000 transistors	n.a.	Circuit-switching	0.35um
Octagon	Chordal ring / Distributed and adaptive	Variable data + 3 bits control	n.a.	n.a.	n.a.	40 Gbits/s	Circuit-switching	n.a.
Proteo	Bi-directional ring / n.a.	Variable control, Data sizes	Input queue + Output queue	VCI	n.a.	n.a.	n.a.	0.18um
Xpipes	Arbitrary / Source static	32, 64 or 128 bits	Virtual output queue	OCP	(Estimated) 0.33 mm^2	64 Gbits/s per switch	n.a.	0.1um
T-SoC	Fat-tree / Adaptive	38 bits max.	Input queue + Output queue	Custom/ OCP	27000 to 36000 NAND gates		4 virtual channels	n.a.
Eclipse	2D sparse hierarchical mesh / n.a.	68 bits	Output queue	n.a.	n.a.	n.a.	n.a.	n.a.
QNOC	2D mesh regular or irregular / XY	16 bits data + 10 bits control	Input queue + Output queue	Custom	(Estimated) 0.02 mm^2	80 Gbits/s per switch	Virtual channels	0.9um
SPIN	Fat-tree / Deterministic and adaptive	32 bits data + 4 bits control	Input queue + 2 shared output queue	VCI	0.24 mm^2	2 Gbits/s per switch	n.a.	0.13um
Æthereal	configurable / Source	32 bits	Input queue	AXL, DTI	0.13mm^2 (6x6 router)	17Gbits/s per link	BW reservation	0.13um
Mango	2D mesh / XY, source	32 bits	Input queue	OCP	0.188 mm^2	16.5Gbits/s per link	Virtual channels	0.12um
SDM	2D mesh / Adaptive	n.a.	n.a.	AMBA	0.024 mm^2	n.a.	BW reservation	0.13um

Figure 2.20: Hard general-purpose NoCs.

Application-specific NoCs: Recently, a couple of application-specific NoCs were reported. Srinivasan, *et al.*, present in [54] that a customized network is synthesized by recursively bisecting the channel intersection graph. Meloni, *et al.*, present an application-specific NoC, where an individual switch is customized in terms of routing paths [67]. An arbiter size is also configured for the customized switch interconnects. The customized switch in [67] is integrated in the Xpipes design flow. In the Æthereal NoC [48], topologies, routing paths, and bandwidth allocations are customized for a given number of IPs and applications. While the Æthereal NoC in [48] does not customize the arbiters and switches themselves, we present the application-specific switches and arbiters in Chapter 6.

2.5.2 Soft interconnects

General-purpose crossbars: Figure 2.21(1) depicts soft overlay general-purpose crossbars. Nikolov, *et al.,* present in [39] a multiprocessor system that accommodates a overlay crossbar, where the centralized crossbar interconnects are systematically constructed. Wee, *et al.,* present in [82] that each of distributed small-sized crossbars is connected to embedded hardwired PowerPC core. Brebner, *et al.,* present in [34] a large-sized (928×928 bits) crossbar utilizing native programmable interconnects and look-up tables (LUTs). The work in [34] was motivated by the fact that current FPGA interconnects are fine-grained and electrically programmable physical circuit switched networks. Hübner, *et al.,* present in [58] that a dynamically reconfigurable network is implemented based on a LUT based macro. These macros are not affected by placement/routing tools and typically provides higher performance compared to afore-mentioned soft implementation, since the network is better optimized for a targeted device. Bobda, *et al.,* present in [21] four types of communication schemes, which are bus macros, shared memories, linear array multiple bus (RMB) and external crossbars are utilized for different communication modes. In [34, 58, 21], the network is implemented utilizing a modern partial and/or dynamic reconfiguration technology.

General-purpose NoCs: NoC-based systems targeting FPGAs are summarized in Figure 2.21(2), in terms of system sizes, clock frequencies, target devices, data widths, buffer depths, number of I/O ports, occupied network areas per switch (or router) and design characteristics of [24, 32, 86, 19, 83, 62, 22]. Most of these systems employ packet switched networks. Each of them entails specific design goals and different characteristics. As an example, Marescaux, *et al.,* designed a network in [86] as a partially reconfigurable module supported by an operating system. Kapre, *et al.,* utilized time-multiplexed switches in [62]. Hilton, *et al.,* present in [22] a circuit switched network with 4 different topologies, where a circuit switched router can be connected with multiple IPs. Pionteck, *et al.,* present in [87] a partially reconfigurable packet switched NoC for FPGA, where a packet router module can be added or removed at run-time. These soft overlay interconnects provide a higher adaptivity to meet the requirements of various IPs. The drawbacks of these soft overlay general-purpose NoCs will be a decreased performance and an increased area cost, compared to the hard NoCs. As an example, 16 routers with 4 virtual channels in [14] occupy 25481 slices, while a common device such as the xc2vp30 Virtex-II Pro contains only 13696 slices in total.

Application-specific NoCs: Little has been reported regarding the soft overlay

FPGA Switch	Type	System cores	Freq. MHz	Chip	System network			Network interface
					DataWidth	Network area	Characeristic	
Nikolov *et al*.	soft	4	100	V2Pro20-6	32-bit	379 slices	Automated design	custom
ATLAS	soft	5	100	V2Pro70	32-bit	n.a.	Transaction memory	custom
Brebner *et al*.	native	928x928bit	155	V2 XC2V6000	1-bit	26912 LUTs	all-to-all crossbar (928x928)	n.a.
Huebner *et al*.	hard macro	5	66	V2 XC2V3000	32-bit	n.a.	Crossbar bus macro	custom
Erlangen Slot Machine	hard/soft	4	n.a.	V2 XC2V6000	16-bit	1536 slices / slot	Bus macro, Circuit router, Crossbar	custom

(1) Crossbars

FPGA soft NoCs	System cores	Freq. MHz	Chip	Router				
				DataWidth	BufferSize	I/O	Area	Characeristic
SoCIN	2X2	n.a.	EPF10K200S	8-bit	4 flits	5	800 logic cells	parameterized
Hermes	2X2	25	V2-1000-4	8-bit	8 flits	5	316 slices	worm-hole
Marescaux *et al.*	3X3	33	V2-6000	16-bit	BRAM	5	446 slices, 5 BRAMs	virtual cut-through
LiPaR	3X3	33	V2Pro30	8-bit	BRAM	5	352 slices, 10 BRAMs	parallel routing
Bartic *et al*.	3X3	50	V2Pro40	16-bit	BRAM	5	552 slices, 5 BRAMs	flexible topology
Kapre *et al*.	8	166	V2-6000-4	32-bit	no buffer	4	732 or 1464 slices	time-multiplexed
PNoC	8	134	V2Pro30-7	16-bit	no buffer	8	1223 slices, 1 BRAM	circuit switching

(2) NoCs

Figure 2.21: Soft overlay interconnects.

application-specific NoCs. Bartic, *et al.*, present in [83] a topology adaptive parameterized network component. The physical topology in [83] is constructed between packet routers. As we will describe in the next chapters, the network occupies the reconfigurable resources. Moreover, the inter- and intra-router network resources are often under-utilized by an application. Therefore we present the topology customization for the crossbar (in Chapter 3) as well as NoC (in Chapter 6).

2.5.3 Hardwired interconnect fabric for FPGAs

A general approach on the future FPGA utilizing hardwired packet-switched NoCs is envisioned in [71] by Hecht *et al*. In [71], a system model is explored with a SystemC abstraction. Gindin, *et al.*, propose in [70] the hardwired networks, where packets are routed on top of fixed underlying physical router networks. In addition, part of the network interface is allowed to be *soft* to adapt to different applications at configuration time. In Chapter 6, we present the analysis of the performance and the cost in the configuration layer as well as the functional layer.

2.6 FLUX interconnection network

In this section, we briefly summarize the FLUX network [79] as a related work. Generally, a parallel programmer develops algorithms having a particular interconnection network in mind. Traditionally, interconnection networks are rigid and often (actually usually) the interconnection network changes from one design point to the next. A consequence is that algorithms and software, when ported to a new family of multiprocessor parallel systems, will not scale in terms of performance and new software development has to be undertaken to adapt to the (new) rigid underlying interconnection network. The general approach of the FLUX network is diametrically opposite to the existing network proposals, for adaptable networks stated by the following: *Interconnection networks are provided (dynamically) on demand to suit the needs of an application/algorithm/program* [79]. Figure 2.22(1) depicts an example of the FLUX network for reconfigurable hardware, where the on-demand topology is *reconfigured* while running the application. A programming paradigm is described in [80] as depicted in Figure 2.22(2), which shows an execution of the SET instruction before or during different phases of an application. Our work inherits general approaches of the FLUX interconnection networks [79]. In Chapter 3, we present a design-time custom crossbar to implement such an on-demand network in the reconfigurable hardware.

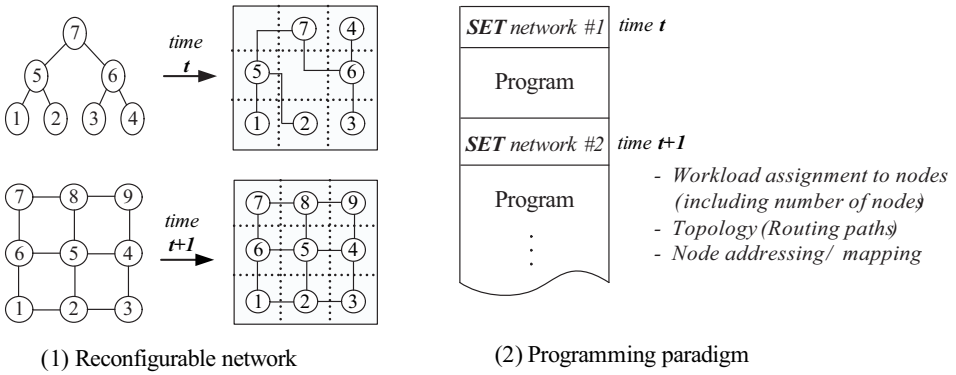


Figure 2.22: The reconfigurable FLUX interconnection network [80][79].

2.7 Summary

In this chapter, we studied the physical fabric, the overlay interconnects, a parallel computation model, a platform model, an analytical model, and a tool chain.

Among these, our focus is the physical fabric, the overlay interconnects, and the analysis. We studied the Æthereal NoC that is operated by an end-to-end flow control using credits and connections. We studied the KPN-based platform model is implemented by the ESPAM tool chain. Additionally, we studied an analytical model. We aim to derive a comparative performance of hard and soft interconnects. For this, we use a Jackson's queuing model. We surveyed on-chip interconnection networks in literature and categorized them. In the overlay layer, we differentiated the hard and soft interconnects though they have same architectural property to highlight different implications from the technology perspective. The trade-off is that the hard interconnect performs better with lack of flexibility, compared to soft interconnects.

Chapter 3

Soft Application-specific Crossbars

A crossbar provides high interconnect performance and minimum traffic congestion. The relatively simple implementation makes the crossbar popular as an internet switch [2]. However, a conventional crossbar is limited in terms of scalability because of the $O(N^2)$ of hardware complexity and the scheduling algorithm complexity. Therefore, it is required to reduce the area cost while we maintain the high performance of the conventional crossbars. To achieve this, we present an application specific interconnect topology construction for a reconfigurable on-chip multiprocessor platform. We also present various customized scheduling schemes and their trade-offs.

This chapter is organized as follows. Section 3.1 presents the motivations of our work. Section 3.2 and Section 3.3 present the design of the customized switches and schedulers, respectively. Section 3.4 presents the implementation results. We conduct the MJPEG case study on the prototype board to evaluate the area cost, performance, and the power consumption. Finally, Section 3.5 summarizes and concludes this chapter.

3.1 Introduction

Traffic patterns are in most cases unknown. Accordingly, the crossbar establishes all-to-all interconnects to accommodate all possible traffic patterns. Due to the all-to-all interconnects, a major problem of the conventional crossbar is the high area cost due to the high number of wires. As the number of ports increases, the area of the crossbars increases in a quadratic manner. This work alleviates the high cost problem in a crossbar by *constructing arbitrary topologies* in a reconfigurable platform. First, we present a design of a customized crossbar, where physical topologies are *identical* to the logical topologies for a given application. Second, we propose that crossbar scheduler only arbitrates the actually established on-demand interconnects for a given topology. This work is motivated by the following key observations:

- The logical topology and traffic information can be *derived* from the parallel *specification* of an application.
- Communication patterns of different applications represent *different logical topologies*. Figure 3.1 depicts task graphs of realistic applications. In Figure 3.1, the numbers between braces in the depicted topologies indicate the number of nodes and the number of required links. As an example, MJPEG$\{6,14\}$ indicates that the MJPEG application requires 6 nodes and 14 links. As depicted in Figure 3.1, logical topologies are application-specific and require *only a small portion* of the all-to-all interconnects. Single applications can be specified differently as observed in the MJPEG (Figure 3.1(5)(6)(7)).
- Table 3.1 shows the number of nodes, the number of links, and their ratio for the task graphs of the benchmark applications. It indicates that an average *number of links per node* is only 1.6 and the variance is 0.96.

In the remainder of this chapter, we present a design, an analysis, and an implementation of our customized crossbar. The presented interconnect combines the high performance of a conventional crossbar and the reduced area of fully customized interconnects for raw data communications.

3.2 Customized switch

A crossbar consists of a switch module and a scheduler. Our goal is to design customized interconnects, where the physical topology and logical topology are iden-

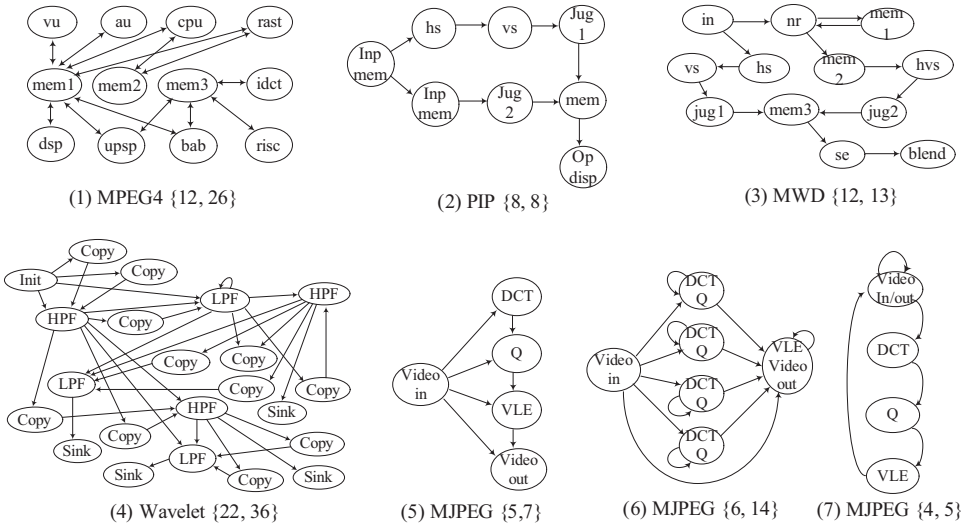


Figure 3.1: Parallel specifications of practical applications.

Table 3.1: Benchmark topologies.

Application	H.264 [93]	MP3 enc [43]	TCP Chk [52]	Hyper LAN [35]	MJPEG [47]	MPEG2 [37]	H.263 [43]
Nodes	5	5	5	6	6	7	7
Links	6	10	14	5	14	13	14
Links/node	1.2	2	2.8	0.8	2.3	1.9	2
Application	IPSEC [88]	PIP [25]	802.11 [53]	DVB-T [20]	DRM rec [69]	MWD [25]	MPEG4 [25]
Nodes	8	8	9	9	12	12	12
Links	7	8	20	23	16	13	26
Links/node	0.9	1	2.2	2.6	1.3	1.1	2.2
Application	VOPD [75]	Wavelet [47]	MMS [43]	ASTB [51]	AV [44]	ROBOT [38]	Avg. (Var.)
Nodes	16	22	25	31	40	88	
Links	20	36	47	30	56	131	
Links/node	1.3	1.6	1.9	0.97	1.4	1.5	1.6 (0.96)

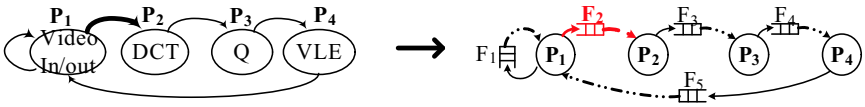
tical. The physical interconnects are required to be instantly switched to adaptively meet the dynamic traffic patterns. In this section, we present an implementation of our switch module. We exploit the fact that the logical topology is represented by a task graph (KPN), in which each node has possibly a different number of incoming and outgoing links. In this work, a parameterized multiplexer array has been implemented for switch modules as a design technique. Topology-specific and different sized multiplexers ensure that required interconnects are established. The switch module has been implemented with the following two steps:

1. *Topology extracting*: The topology table is extracted from the parallel application specification.

2. *Parameter passing*: The extracted topology tables are passed as static parameters to the switch module. multiplexer select signals are generated in the scheduler and also passed to the switch module as a dynamic parameter.

First, the graph topology is extracted from the KPN specifications. Each processor port has a set of incoming and outgoing links, as depicted in Figure 3.2(2). Table A indicates the number of incoming links and a list of ports from which the links originate for the MJPEG$\{4,5\}$ application (see Figure 3.2(1)). As an example, port p_2' has one incoming link from port p_1, indicating that processor $\mathbf{P}_2$ can possibly read the data located in the FIFOs connected to port p_1. Table B indicates the number of outgoing links and a list of ports to which the links are directed. As an example, port p_1 has two outgoing links to ports p_1' and p_2', indicating that either processor $\mathbf{P}_1$ or $\mathbf{P}_2$ reads the data located in the FIFOs connected to port p_1. Tables A and B are used to systematically implement customized multiplexer arrays instead of full multiplexers. The unused inputs of each multiplexers are tied off and optimized away during the synthesis step. There are two types of multiplexers, namely processor-side multiplexers and FIFO-side multiplexers, as depicted in Figure 3.2(2). Table A is used to implement processor-side multiplexers controlled by $CTRL\_FIFO$ signals and Table B is used to implement FIFO-side multiplexers controlled by $CTRL\_PROC$ signals.

Second, the two tables described above are passed to a VHDL function as parameters to actually establish a circuit link. The function generates parameterized multiplexer arrays. Once the request is granted, 2 cycles are required to establish a circuit link between the source and the designated target port. Once a link is established, a remote memory behaves as a local memory until the link is cleared. Figure 3.2(3) depicts the finally customized switch module, in which 12 multiplexers are instantiated. It can be noted that an N-port full crossbar contains N-way full multiplexers per port, while our interconnect contains variable-way multiplexers per port, depending on the graph topology. The aforementioned two signals $CTRL\_FIFO$ and $CTRL\_PROC$ are dynamically generated by the scheduler, as described in the next section.



(1) 4-node MJPEG{4,5} application

Parameter 1
Topology tables

Table A : Incoming links

Port	NUM_LINK	LIST
p'_1	2	p_1, p_4
p'_2	1	p_1
p'_3	1	p_2
p'_4	1	p_3

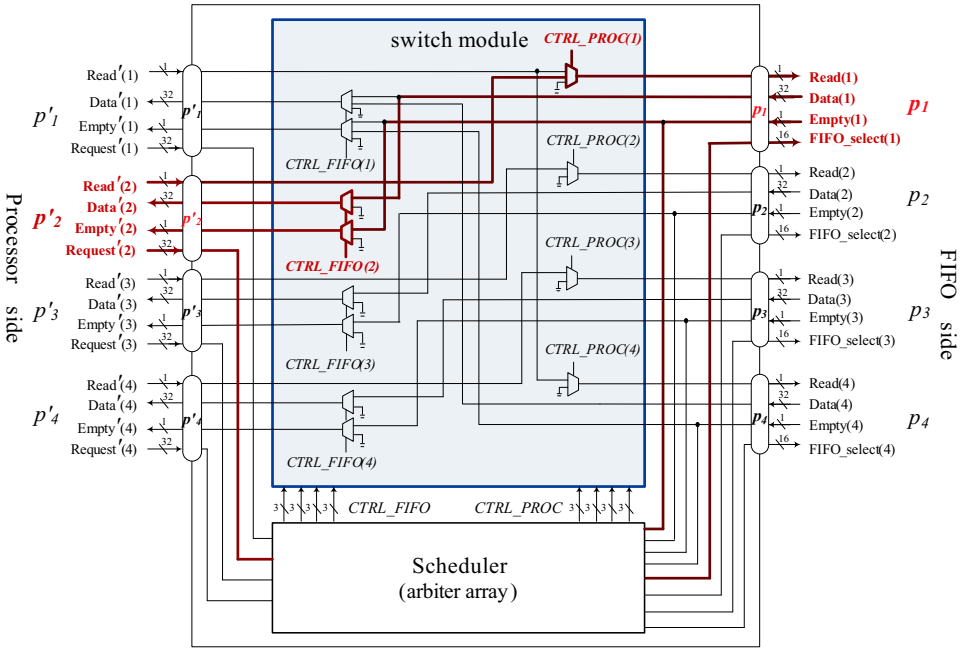
Table B: Outgoing links

Port	NUM_LINK	LIST
p_1	2	p'_1, p'_2
p_2	1	p'_3
p_3	1	p'_4
p_4	1	p'_1

Parameter 2
Traffic control signals

CTRL_FIFO (CTRL_PROC)	MUX_OUT
0	clear
1	p_1 (p'_1)
2	p_2 (p'_2)
3	p_3 (p'_3)
4	p_4 (p'_4)

(2) Implementation of switch modulefor the channel between $\mathbf{P_1}$ and $\mathbf{P_2}$



(3) Customized switch module for MJPEG {4,5}

Figure 3.2: Parameterized switch module for the MJPEG{4,5}.

3.3 Customized schedulers

The scheduler plays a key role in achieving high performance in terms of latency and throughput. A commercial crossbar scheduler typically accommodates an arbiter per port and each arbiter arbitrates the incoming requests in parallel [63]. This scheduler is called a fully parallel scheduler (FPS)[1]. As discussed earlier, a major bottleneck of the fully parallel scheduler is the high cost due to the all-to-all interconnects inside the scheduler module. In addition, the crossbar scheduler is an important basic building block in a router of a modern NoC [85]. In some cases, such schedulers contain a *single central arbiter* that *sequentially arbitrates* a single request at a time, while data transmission can be parallel. This scheduler is called a sequential scheduler (SQS)[2]. Accordingly, the arbitration latency in the SQS can increase as the number of crossbar ports increases. In this section, we present a design and an analysis of application-specific crossbar schedulers. The presented scheduler *arbitrates only the necessary requests* instead of all requests.

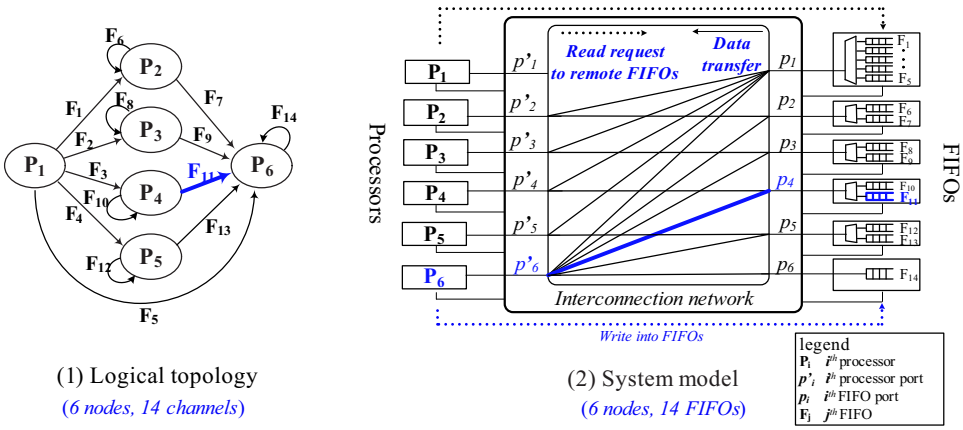
3.3.1 Reference scheduling schemes

We consider a sequential scheduler (SQS) and a fully parallel scheduler (FPS) as references to compare with our custom schedulers. Figure 3.4 depicts the behavior of the SQS, FPS, and our custom schedulers for the MJPEG application in Figure 3.1(6). Figure 3.3(1) depicts the logical topology (or data flow graph) with 6 nodes and 14 connections. In our model of computation, each communication connection is mapped onto the FIFOs labelled by $\mathbf{F}_1$ to $\mathbf{F}_{14}$. Figure 3.3(2) depicts the system model. The physical system consists of 6 nodes and 14 links (or 14 FIFOs). The i^{th} node is connected to processor port p'_i and FIFO port p_i. For example, in the 6^{th} node, processor $\mathbf{P}_6$ is connected to the processor port p'_6. The FIFO $\mathbf{F}_{14}$ is connected to the FIFO port p_6, as depicted in Figure 3.3(2). A FIFO index corresponds to the connection index, as depicted in Figure 3.3(1). A bold line in Figure 3.3(1) represents that processor $\mathbf{P}_6$ remotely sends the *request* signal to FIFO $\mathbf{F}_{11}$. Then, FIFO $\mathbf{F}_{11}$ sends data to processor $\mathbf{P}_6$. Note that data in FIFO $\mathbf{F}_{11}$ is locally written by the processor $\mathbf{P}_4$. Figure 3.4(1) depicts the bipartite graph based on the system model. The thick and thin lines depict all possible requests according to the topology in Figure 3.3(1). The thick lines represent an example request pattern. In this example, 4 processors request to 4 FIFOs.

[1]The term *parallel* means that the *grant* and *accept* operations can be concurrent for multiple requests because there is a arbiter per port.

[2]The term *sequential* means that the *grant* and *accept* operations are sequentially performed for each request because there is only a single arbiter.

Figures 3.4(2) to (5) depict the cycle-by-cycle behavior of four different schedulers for the request patterns (bold links) in Figure 3.4(1). To establish the link between the processor and the target FIFO, three steps are required. First, a processor sends a *request* to an arbiter. Second, the arbiter *grants* the request when the target port is idle and the round-robin pointer points to the requesting processor. Third, the request is *accepted* when the target FIFO contains data. These operations are denoted by R, G, and A. The round-robin pointer is denoted by the oval. For the sake of simplicity, the data is assumed to be requested by a processor in the first cycle. The arbiter is assumed to perform a circular round-robin arbitration in the order of p'_1, p'_2, p'_3, and so on. After the request is granted, a link between a processor and a FIFO port is established using a handshaking protocol, which is assumed to take 2 cycles. The bold lines in Figures 3.4(2) to (5) represent actual data transmission, which is assumed to take 5 cycles.



(1) Logical topology
(6 nodes, 14 channels)

(2) System model
(6 nodes, 14 FIFOs)

Figure 3.3: 6-node MJPEG{6,14} application example.

Figure 3.4(2) depicts the behavior of a SQS, where one port is arbitrated at a time. A crossbar contains a central arbiter that sequentially grants a single request at a time. A request is served after a request in the previous port index is arbitrated and/or the link is established. Subsequently, 48 cycles are required to serve those requests, as depicted in Figure 3.4(2). Figure 3.4(3) depicts FPS, where homogeneous arbiters are located in each port. Unlike the SQS, multiple requests can be arbitrated in parallel. Each arbiter checks for all ports whether there is a request or not. Consequently, 40 cycles are required in total, as depicted in Figure 3.4(3). The circular round-robin pointer in our FPS implementation is updated when the

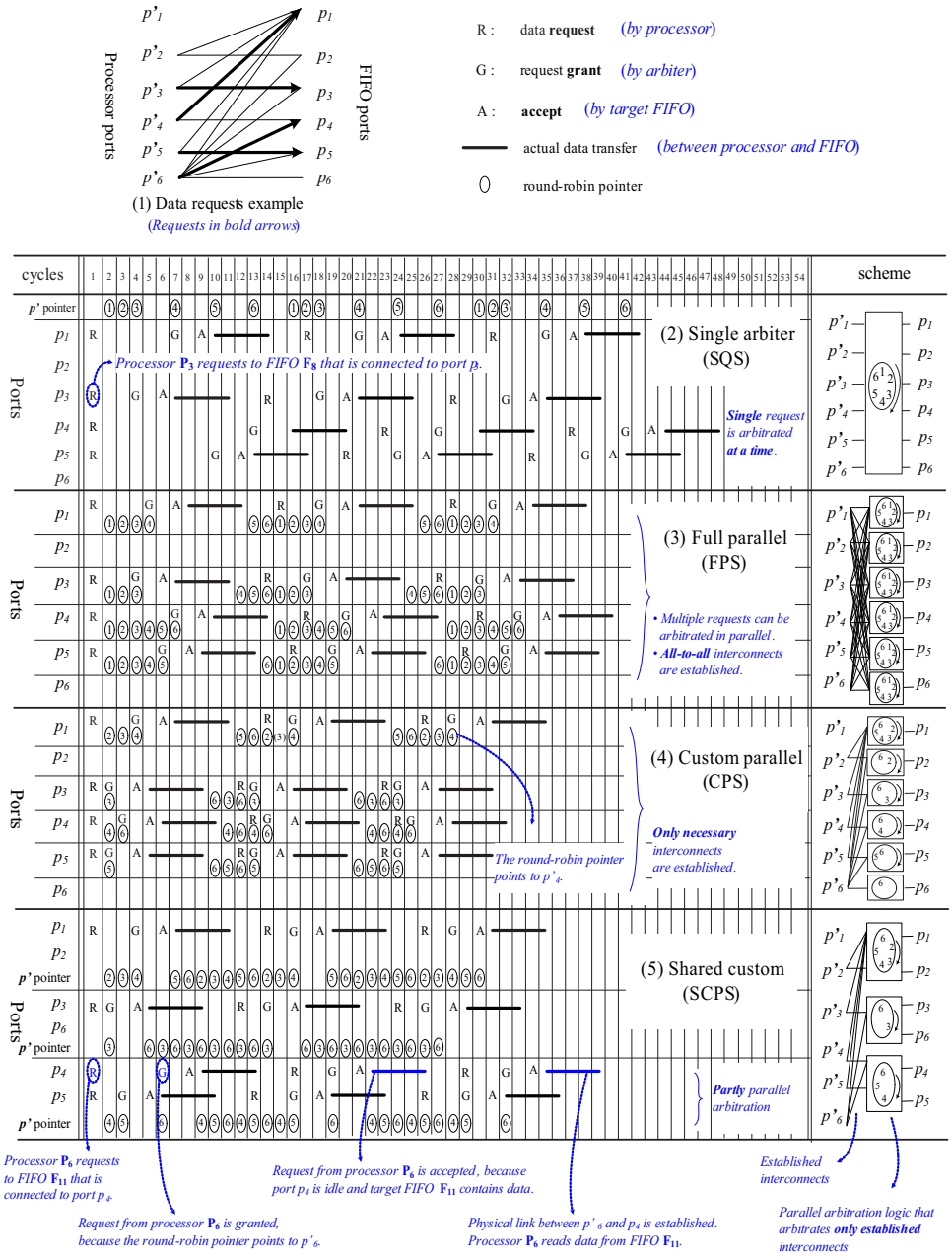


Figure 3.4: Different scheduling schemes for MJPEG{6,14}.

request is accepted. As depicted in Figure 3.4(3), FPS performs better than SQS, since concurrent requests can be served in parallel.

3.3.2 Custom parallel scheduler (CPS)

Our proposed custom parallel scheduler (CPS) scheme is similar to the FPS in that the scheduler consists of arbiter arrays. In our CPS, however, the round-robin pointer update operation is performed only for the on-demand interconnects. Additionally, we exploit the fact that the application is specified by a task graph, in which each node has possibly a different number of connected links. As a design technique, each arbiter is parameterized with respect to the logical topology. Application-specific and differently sized arbiters ensure that the topology of the physical interconnects is identical to the logical topology specified by the application partitioning. Given the logical topologies from the application specifications, our CPS operates as follows:

1. *Request*: A processor issues a request by designating the target FIFO port and FIFO index.

2. *Grant and Accept*: If there is a request in the round-robin pointer and the target FIFO port is idle, the request is granted. Afterwards, the target FIFO status is checked. If the target FIFO is not empty, the request is accepted and the link is established. The round-robin pointer is updated to the one that appears next in a round-robin schedule, where the round-robin schedule is determined by *the topology of an application.*

If the pointed request is a *Clear_Request*, the connection is cleared. If there is no request, the round-robin pointer is incremented. The arbiter is implemented with a three-state finite state machine as described above. The CPS module is implemented with parameterized arbiter arrays. The crossbar with our scheduler operates with the following specific steps. First, the topology table is extracted from the application specifications. Figure 3.5 depicts an example for the 6-node MJPEG application. Each FIFO port has possibly different set of request links, as depicted in Figure 3.5(1a). The topology table in Figure 3.5(1b) represents the number of links and a list of ports from which the links are directed. As an example, the round-robin pointer in arbiter A_2 points to either p'_2 or p'_6, as depicted in Figure 3.5(1c), indicating that the data in FIFOs connected to p_2 is transferred to either processor $\mathbf{P}_2$ or processor $\mathbf{P}_6$. In other words, arbiter A_2 has two possible requests in total, from processor $\mathbf{P}_2$ and $\mathbf{P}_6$. Therefore, arbiter A_2 searches for only two links. Note that the FPS arbiter at each port searches for 6 links. Second, given

the topology table, the arbiter generates two control signals, $CTRL\_PROC$ and $CTRL\_FIFO$. Figure 3.5(2) depicts that the processor $\mathbf{P}_6$ reads from a remote memory connected to p_2, as represented by the bold line. In case there is a request, the request is registered. The registered 32-bit request signal contains a target port and a target FIFO index. If the target port is idle and the designated FIFO contains data, two control signals are generated. In this way, control signals dynamically configure the switch fabrics. Figure 3.4(4) depicts the scenario of the CPS. As a result, Figure 3.4(4) indicates that only 35 cycles are required. In general, CPS performs better than SQS, since the arbitration is performed in parallel. Note that the request search space of CPS is a subset of the full search space of the FPS. Only when an application requires all-to-all communication, the CPS is identical to the FPS. An area reduction also can be expected, since only on-demand links are physically established. Moreover, in many cases, only a single link is connected to the crossbar port, indicating that no arbitration is necessary for those ports.

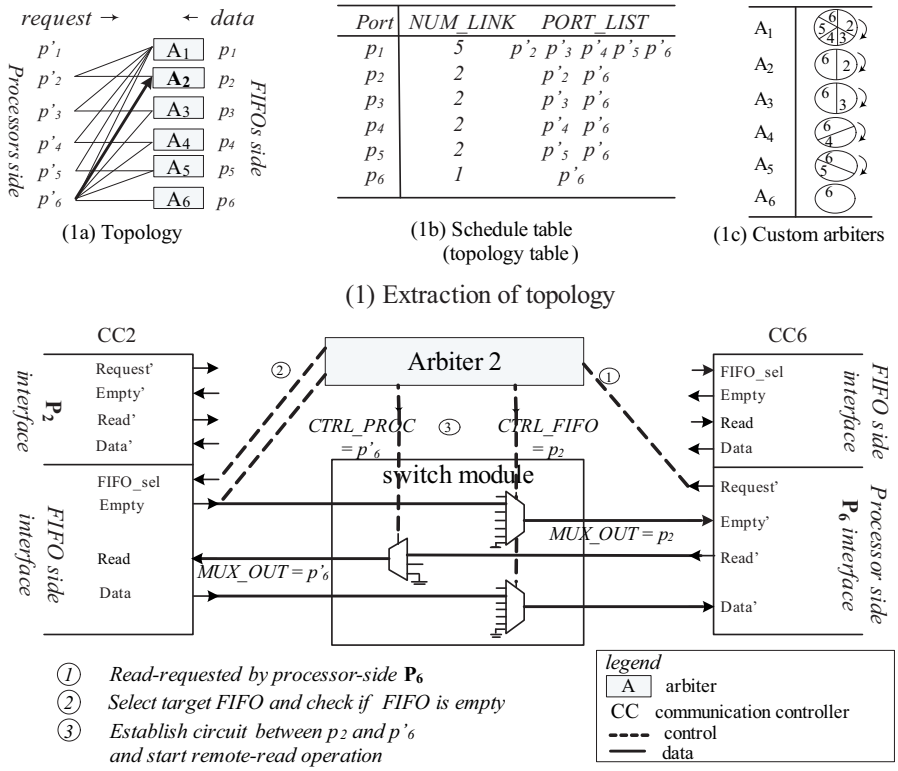


Figure 3.5: A customized crossbar for MJPEG{6,14}.

3.3.3 Shared custom parallel scheduler (SCPS)

In our CPS scheme, an arbiter is accommodated at each crossbar port to perform parallel arbitration. This means that the CPS is a topologically customized version of the FPS. When the number of links per node increases, however, the CPS suffers from the scalability problem, similar to the FPS. In this case, sharing network resources (such as arbiters and wires) can be an alternative solution to alleviate the scalability problem. In this section, we propose a custom scheduler with the novel shared arbitration (SCPS) scheme. In SCPS, multiple arbiters are shared, such that each arbiter sequentially performs arbitration while multiple arbiters operate in parallel. Moreover, our SCPS scheme combines the advantages of the low cost in SQS and increased performance in CPS. Figure 3.6 depicts an example of CPS and SCPS for a 6-node MJPEG application. Figure 3.6(1) depicts CPS, where an arbiter is established per port. Figure 3.6(2) depicts SCPS, where arbiters for port (p_1, p_2), (p_3, p_6), and (p_4, p_5) are shared. The traffic requirement is also depicted in Figure 3.6, derived from the YAPI tool [28]. λ is the total incoming traffic bandwidth or an arrival rate to the system. Figure 3.6 indicates that the number of arbiters in SCPS is 3 and the number of links is 10. For comparison, the number of arbiters in CPS is 6 and the number of links is 14. This means the area cost can be reduced by sharing the resources. Moreover, our SCPS is constructed over on-demand interconnects. Compared to CPS, the interconnect performance can be likely decreased. This is due to the fact that a shared arbiter is accommodated for multiple ports and the round-robin search space can be increased for the shared arbiters. Figure 3.4(5) depicts the example scenario for the SCPS, where 3 sequential arbiters operate in parallel. As Figure 3.4(5) shows, 39 cycles are required to serve the request patterns.

The arbiters can be shared in a way that the traffic bandwidth is balanced. Given a CPS scheme, our method to cluster (or share) the arbiters is described as follows:

1. *Calculate cost function*: Calculate the individual cost using a cost function for arbiter A_j, where $1 \leq j \leq p$. p denotes the number of CPS arbiters.
2. *Sort*: Sort arbiters in an increasing order based on the derived cost function.
3. *Iterate clustering*: Iteratively cluster arbiters with the lowest and the highest cost function in the sorted list.

The cost function is a relative metric to represent the utilization of an arbiter. We define the cost function for the arbiter A_j as follows:

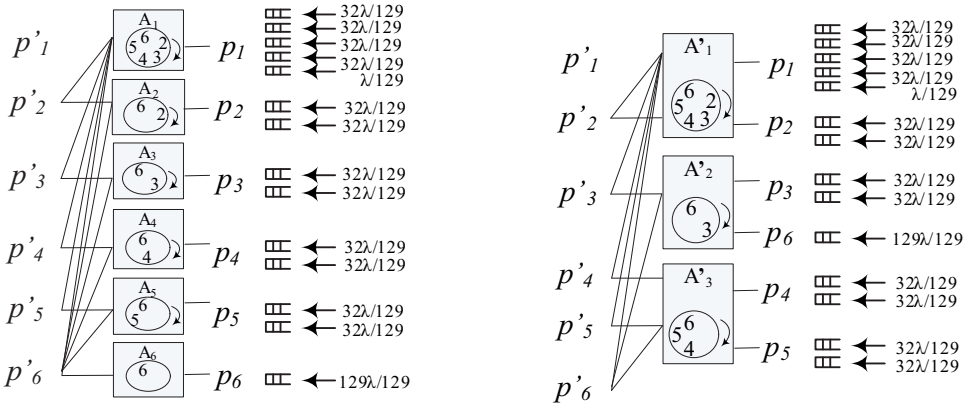


Figure 3.6: Shared custom parallel scheduler for MJPEG{6,14}.

$$C\{A_j\} = \frac{links_j}{2} \times \mathrm{U}_j, \tag{3.1}$$

where $C\{A_j\}$ refers to the cost function of a CPS arbiter in the j^{th} port. $links_j$ refers to the number of physical links which are connected to the arbiter A_j. $\frac{links_j}{2}$ corresponds to the relative arbitration latency as described in the next section. U_j refers to the summation of the traffic for the arbiter A_j and corresponds to relative arbiter utilization. The individual connection utilization for U_j is depicted in Figure 3.6. As an example, U_1 is derived by $\frac{32+32+32+32+1}{129}$ (see Figure 3.6(1)). We multiply U_j by $\frac{links_j}{2}$ to reflect the actual utilization of an arbiter. As an example in Figure 3.6, we cluster arbiters in the following way:

1. *Calculate cost function*: We calculate the cost function for each arbiter. As an example, $links_1 = 5$ and $\mathrm{U}_1 = \frac{32+32+32+32+1}{129}$ for arbiter A_1. Therefore C$\{A_1\}$ = $\frac{5}{2} \times (\frac{32+32+32+32+1}{129}) = 2.5$. Similarly, C$\{A_2\}$ = C$\{A_3\}$ = C$\{A_4\}$ = C$\{A_5\}$ = 0.496, and C$\{A_6\}$ = 0.5.

2. *Sort*: Arbiters are sorted in an increasing order. We obtain ($\{A_2\}$,$\{A_3\}$,$\{A_4\}$,$\{A_5\}$,$\{A_6\}$,$\{A_1\}$)

3. *Iterate clustering*: We cluster $\{A_1$,$A_2\}$, $\{A_3$,$A_6\}$, and $\{A_4$,$A_5\}$.

In this way, a highly utilized arbiter and a less utilized arbiter can be clustered. We considered clusters of two arbiters only. More than 2 arbiters (or variable-sized arbiters) can be shared in a similar way from the sorted list.

Comparison with related work: The presented custom crossbar is similar to [60][76][77][78] in that an application-specific interconnects are established between masters and slaves. A simulation-based approach is adopted by [60][78]. In [78], an application traffic trace is conducted to synthesize the application-specific topology, which requires hours of design space exploration time. In [77][78], it is reported that the full crossbar performs better. Our crossbar differs from [60][76] in that variable sized custom (shared) arbiters are connected to slaves and different sized multiplexers are utilized. Our design method differs from [77][78] in that our on-demand topology information is systematically and automatically extracted in the application specification step. Additionally, a multiprocessor system combined with our custom crossbar is rapidly prototyped on the reconfigurable hardware using the ESPAM [39][40][41] tool chain. Finally, we obtain the cost and performance in the configuration layer as well as the functional layer.

3.4 Implementation

A full crossbar and our custom crossbars are implemented in VHDL to measure the area. The implementations are generic in terms of data width, number of ports, and on-demand topologies. Assuming each processor is associated with a single crossbar port, the full and custom crossbars are synthesized, placed and routed using the Xilinx ISE tool on Virtex-II Pro FPGA[3] and the areas have been obtained.

Soft custom switch: Our custom switches are implemented using parameterized multiplexer arrays for the task graphs shown in Table 3.1. Figure 3.7 depicts the number of nodes, the number of required links, the area of the switch module. As a result, the custom switch requires on average 83% less area compared to the full switch. As Figure 3.7 shows, the area of the interconnect highly depends on the number of nodes and links. The area of our interconnect not only depends on the number of nodes that determine its size but also on the topology. It can be observed that the higher area reduction is obtained as the interconnect size increases. This is due to the fact that the average number of links per node is only 1.6.

Soft custom schedulers: The soft custom scheduler modules are implemented with parameterized arbiter arrays. We experimented with the benchmark topolo-

[3]We targeted a commercial Xilinx Virtex-II Pro device, particularly because the Xilinx device supports a partial reconfiguration, while any reconfigurable hardware can be a targeted device. The state-of-the-art Virtex-4 or Virtex-5 devices have reduced configuration frame size and increased logic density. However, all of these devices stem from the same origin and the architectural difference is minor.

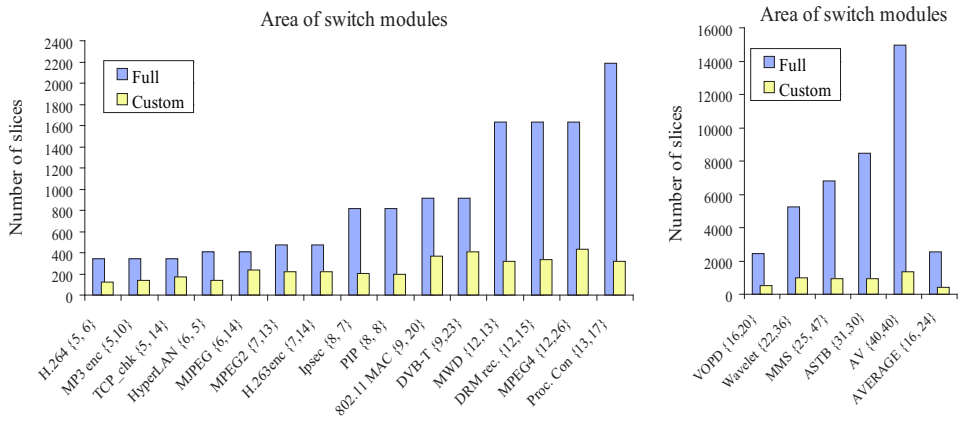
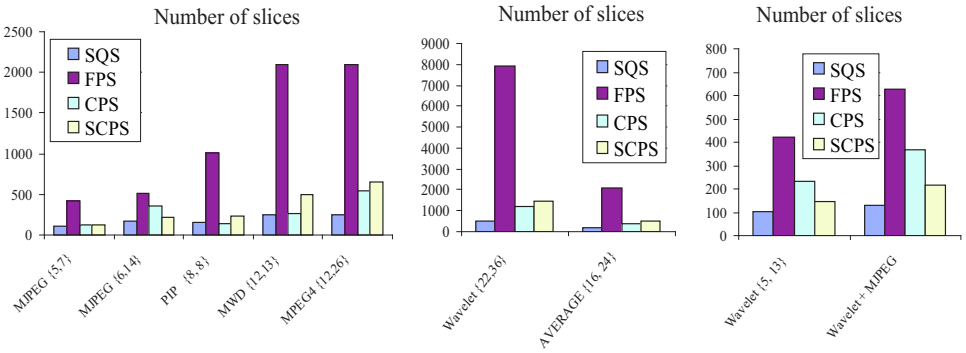


Figure 3.7: Area of switch modules in a soft crossbar.

gies depicted in Figure 3.1. The implementation results are depicted in Figure 3.8(1). Our custom parallel scheduler (CPS) requires on average 83% less area compared to the FPS. The CPS requires on average 28% more area compared to the sequential scheduler (SQS). The shared custom parallel scheduler (SCPS) occupies 14% more area than CPS. As described earlier, this is due to the fact that in many cases only one link is connected to arbiters in CPS scheme. Due to this, the CPS arbiter logic can be greatly simplified, especially when a single task is mapped onto a processor. In addition, the single SCPS arbiter contains relatively more complex decoding logic than single CPS arbiter.

In Figure 3.4, multiple cycles are required for an arbitration. In fact, the number of arbitration cycles is typically a single cycle. Our major goal is to reduce the area of the interconnects. Additionally, we consider multi-cycle arbiters for the sake of simplicity. In the ESPAM design flow, arbiters are systematically implemented for any topology.

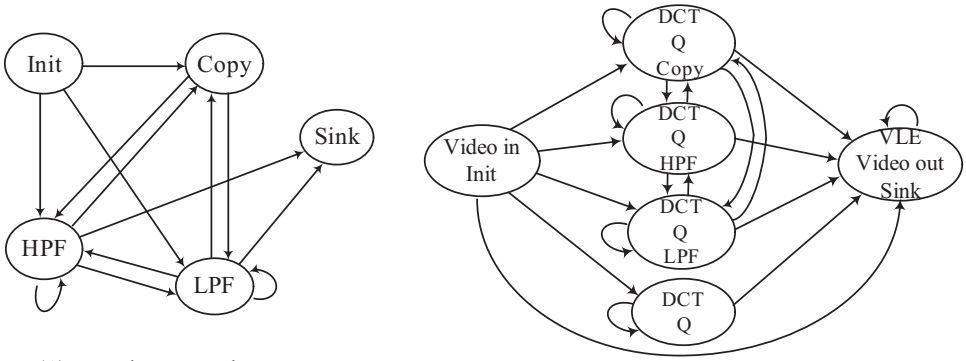
In Figure 3.8(1), we considered that a single node is mapped onto a single processor. In this case, the CPS scheme is sufficient. The number of tasks is often greater than the number of physical processors for given applications. Therefore, multiple tasks are required to be mapped onto a physical processor because a target device is limited in size. In this case, the required number of links per port may increase. Figure 3.9(1) depicts the 5-node representation for the 22-node Wavelet application. Figure 3.9(1) is derived by clustering the same tasks onto a single processor. As a result, the number of links per node is $\frac{13}{5}$, or 2.6. Note that the number of



(1) One-task to one-processor mapping (2) Multiple-to-one mapping

Figure 3.8: Area cost of soft crossbar schedulers.

links per node is $\frac{36}{22}$, or 1.6 when a single task is mapped onto one processor. Figure 3.9(2) depicts a topology of a synthetic application that combines the 6-node MJPEG and the 5-node Wavelet specification. In this case, the number of links per node is $\frac{20}{6}$, or 3.3. In both cases, the number of links per node increases when compared to one-to-one mapping. We implemented different schedulers for these cases and compared the area cost. As depicted in Figure 3.8(2), the SCPS scheme requires 70% less area than the CPS scheme. This experiment suggests that when the number of links per port increases, the SCPS can be beneficial.



(1) 5node-Wavelet {5, 13}

(2) 6-node MJPEG + 5-node Wavelet {6, 20}

Figure 3.9: Topology after multiple tasks are mapped onto a processor.

Wire utilization: To measure the wiring complexity[4] in FPGAs, we implemented

[4]Wiring complexity can be indicated by the number of utilized pips (programmable interconnect

the crossbars for the benchmarks for the targeted Virtex-II Pro device. As a result, Figure 3.10 depicts that the custom crossbar requires 84% less pips (or wire segments) than the full crossbar.

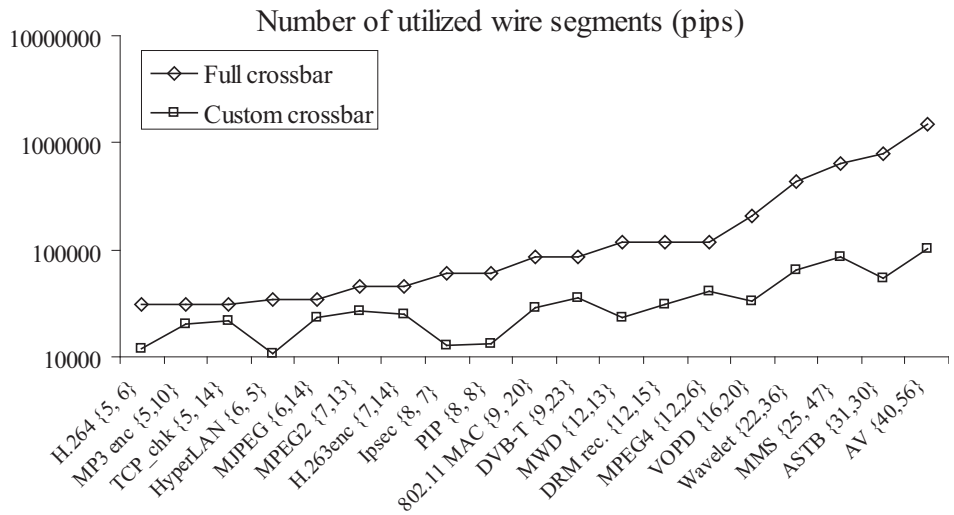


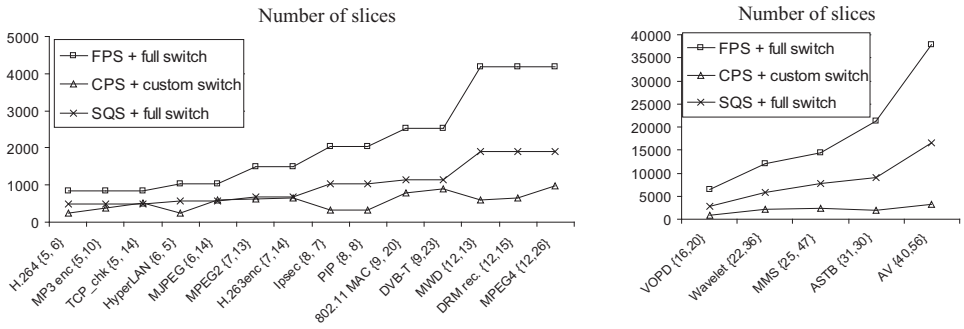
Figure 3.10: Wire utilization in FPGAs.

It can be noted that our custom crossbar is soft because they constitute overlay interconnects on top of underlying fine-grained reconfigurable FPGA interconnect fabric. In this work, the interconnect is customized at compile-time. In the ESPAM design flow, the topology information can be statically derived from the application specification and does not change for the entire lifetime of an application. The design flow rapidly instantiates on-demand interconnects and scheduler processors, based on the extracted traffic information. The interconnect architecture uses the reconfigurability of the FPGAs which is an inherent part of the design flow.

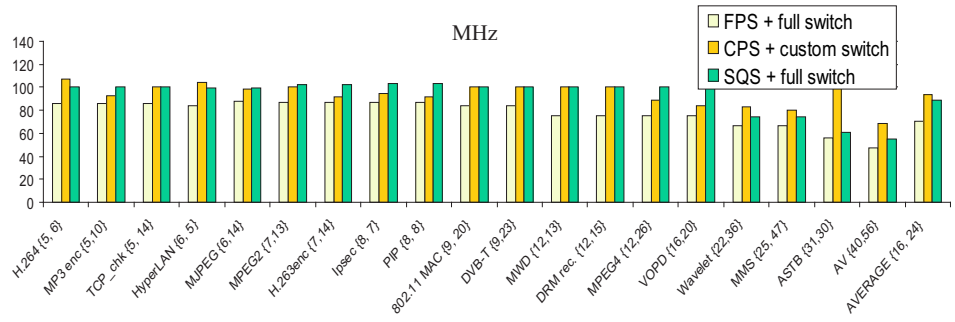
Customized crossbars: We describe the implementation results of different soft crossbar interconnects to compare the area cost and the clock frequency. First, Figure 3.11(1) depicts the area for different topologies. As a result, our custom crossbar occupies on average 84% less area than the crossbar with FPS module and 67% less area than the crossbar with SQS module. In addition, the custom crossbar for the 88-node ROBOT benchmark occupies only 27% of the xc2vp100

points). This is due to the fact that a signal wire between look-up tables (LUTs), namely a *net*, is mapped onto a single or multiple *wire segments* and *pip*s. The number of utilized pips is available from the XDL tool [95], while the wiring information is proprietary.

device, while the other crossbars do not fit in our target device. Second, Figure 3.11(2) depicts the clock frequency of different crossbars. The clock frequency of our custom crossbar is 94 MHz on average. In addition, our custom crossbar operates at 91 MHz for the 88-node ROBOT benchmark. The experimental results indicate that our custom crossbar provides better performance per cost, compared to reference crossbar interconnects.



(1) Area



(2) Clock frequency

Figure 3.11: Area and clock frequency of soft crossbar interconnects.

Prototyping: Our soft custom crossbar has been integrated as a modular communication component in the ESPAM tool chain. Using the ESPAM tool chain, an actual system has been prototyped onto the ADM-XPL FPGA board [6]. We experimented on an MJPEG encoder application that operates on a single image with size 128×128 pixels. We experimented with the three alternative task graphs for MJPEG applications, where our custom crossbar provided the on-demand topologies. The implemented system is homogeneous in that each node contains a 32-bit MicroBlaze processor.

Figure 3.12 depicts the prototype results, in terms of performance, area, and power consumption. Figure 3.12(1) depicts the occupied area for the custom and full switch. The custom switch requires on average 68% less area, compared to the full switch. Figure 3.12(2) depicts the system cycles for different topologies. The system cycles decrease as the number of IP cores increase. When the system incorporates a 6-node crossbar, the system performs $4.6\times$ better than a sequential processing. It can be observed that the topology plays an important role for the system performance. Figure 3.12(3) depicts a performance comparison between different schedulers. When the token size is 1 word, the custom scheduler (CPS) reduces the number of cycles by 10% compared to reference schedulers. When the token size is 64 words, the CPS is still better, while the improvement is negligible. This is due to the fact that the transmission latency (due to the large token size) is a dominant factor to determine the performance. Figure 3.12(4) depicts a comparison of power consumption. We use XPower tool [95] to measure the dynamic power consumption. As a result, the custom crossbar reduces the power by 42% compared to a crossbar with SQS and by 71% compared to a crossbar with FPS. The custom crossbar significantly reduces the power consumption due to the fact that the signal switching activity occurs only for on-demand interconnects. In addition, the capacitive loads of the custom switch is much less than the loads of the full switch.

3.5 Conclusions

In this chapter, we presented topologically customized crossbar switches and schedulers designed for reconfigurable hardware. By (systematically) constructing a physical topology that an application requires, we showed that the proposed custom crossbar reduces the high are cost problem. Our customized interconnect efficiently utilizes the bandwidth by establishing on-demand on-chip resources. Only in case the application requires an all-to-all topology, the interconnect is identical to a full crossbar. We showed that the custom interconnect can be implemented using parameterized multiplexer and arbiter arrays. In our implementation, the switch module and the scheduler are generic in terms of data widths, number of processors, and custom topologies. In this way, the interconnect is adapted to an arbitrarily specified logical topology and arbitrary number of processors, without modifying the interconnect implementation itself. In this work, we consider the KPN system model while the presented design techniques can be generally utilized in other systems. We presented a custom scheduling scheme for the on-demand topology. We prototyped the soft crossbars. As a result, our custom crossbar maintains better performance and significantly reduces the area cost and power consumption compared

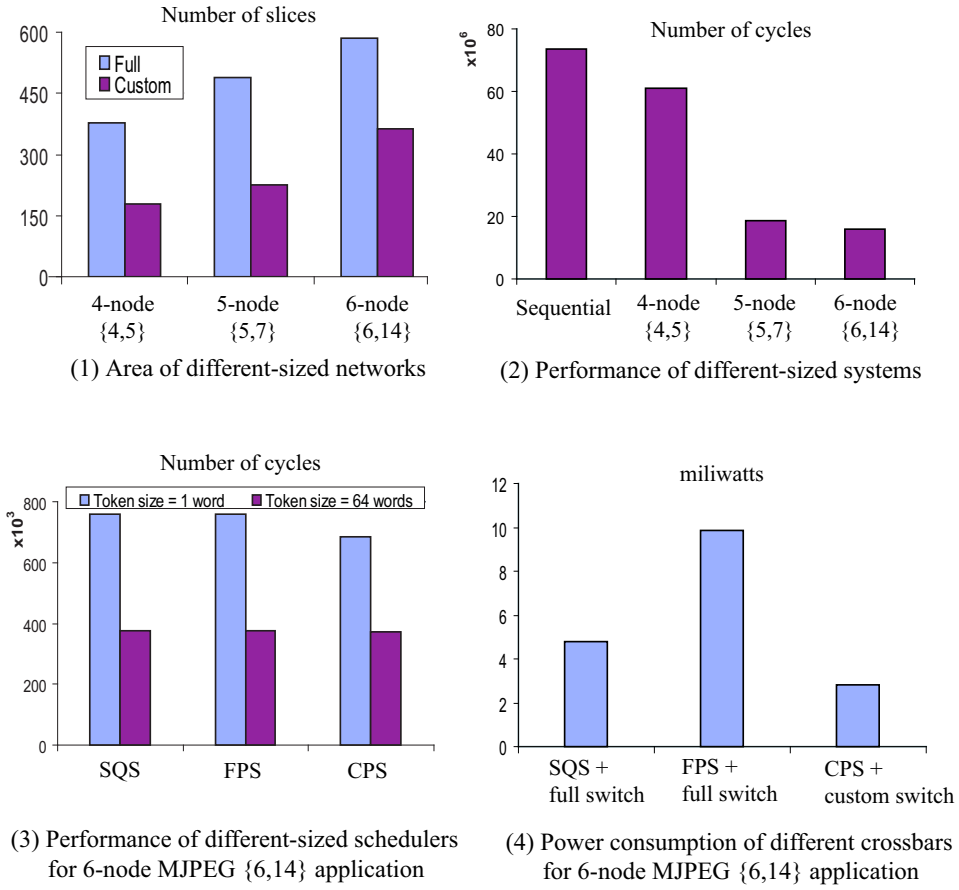


Figure 3.12: Experiments on the prototype for MJPEG applications.

to the conventional full crossbar.

Chapter 4

Partially Reconfigurable Soft Interconnects

In the previous chapter, we presented the customization of soft crossbars, where all necessary wires are configured before application execution. The custom crossbar can be suitable when the required topology is relatively simple. However, an application often entails complex topologies and they may change dynamically. Consequently, we investigate a method to implement such a dynamic topology. In this chapter, we investigate the wire reconfigurability in FPGAs to implement reconfigurable point-to-point (ρ-P2P) interconnects. We conduct an experimental study for the viable implementation of dynamic soft ρ-P2P interconnects. Additionally, we reduce the reconfiguration time, because the reconfiguration overhead is a major bottleneck of the soft networks in FPGAs. To do this, we present a novel use of the partial reconfiguration technique in modern FPGA technology to implement a dynamic network topology.

The organization of this chapter is as follows. Section 4.1 presents the motivations of our work. Section 4.2 presents an wire analysis and motivational examples. Section 4.3 presents our implementation and experimental results. Finally, Section 4.4 summarizes and concludes this chapter.

4.1 Introduction

In modern on-chip multi-core systems, the communication latency of the interconnects is increasingly becoming a significant factor hampering system performance. Many multi-core chips, however, incorporate rigid interconnects, i.e., mostly utilizing a 2D-mesh as the underlying physical network topology combined with packet routers. More specifically, it is necessary for the designer to either (i) *modify* algorithms to suit the underlying fixed topology or (ii) *embed* the logical topology (intended by the algorithm) onto the physical topology. The topology embedding techniques are well-studied [33] and usually require the introduction of a router module to handle network dilations and congestions. As a result, these chips that still utilize rigid interconnects have the following limitations. First, the programmer must have intricate knowledge of the underlying physical interconnect to fully exploit it. Second, communication latency can be increased due to topology embedding that is also likely to incur traffic congestions. Therefore, we aim to design and implement (dynamically) adaptive interconnects to alleviate these limitations. Figure 4.1 depicts our general approach and the interconnects are adaptively changed to better suit different traffic patterns. The ability to *construct* topologies on-demand (at application start time or even during run-time) is likely to improve performance. In this chapter, we implement an arbitrary topology by updating small-sized partial bitstreams for point-to-point (P2P) interconnects. Furthermore, the topology reconfiguration latency is significantly reduced using a partial reconfiguration technique. Our work is motivated by several key observations:

- Different applications (or algorithms) generate different traffic patterns that require different topologies to handle them in the best possible manner. In addition, sub-routines within a single application even require different traffic patterns.
- A parallel application is actually specified by a peer-to-peer (P2P) data flow or a task graph. In addition, modern FPGA interconnect fabrics inherently comprise reconfigurable, circuit switched, point-to-point interconnects. The P2P overlay interconnects eliminate the afore-mentioned overhead of embedding topologies.
- Modern reconfigurable hardware fabrics contain rich intra-chip wiring resources and additionally provide a capability to change the interconnections (possibly) in run-time. These wire segments are heavily under-utilized and they can be efficiently utilized through partial configuration techniques.

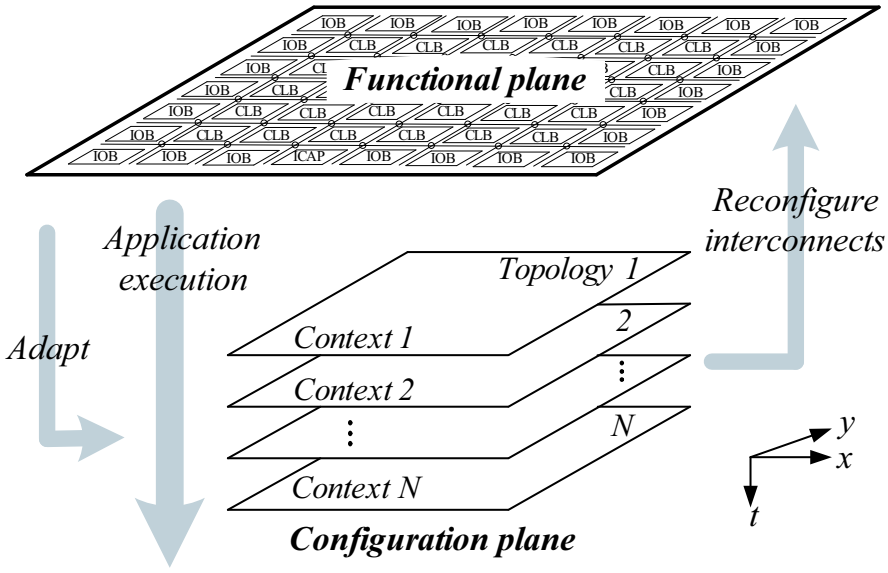


Figure 4.1: Adapting interconnects to an application in a spatial (x, y) and temporal (t) manner.

4.2 Wire analysis and motivational examples

In [92], a network architecture using the reconfigurable optical connections is proposed. The network in [92] consists of a base network with fixed regular topology (for background traffic), augmented with reconfigurable extra direct links (for high-volume burst traffic).

In this section, we describe the relationship between the number of wires and CLB tiles. We study how the wires are utilized in the bitstream. Subsequently, we present motivational examples to use dynamic interconnects depicted in Figure 4.1.

Functional plane: In the functional plane, the relationship between an average number of terminals and blocks in a partitioned design can be described by the Rent's rule [18]. This is represented by $T = t\,B^p$, where T is the total number of terminals, t is an average number of terminals per logic block, B is the number of logic blocks, and p is the Rent exponent. An experimental study in ASIC technology indicates that the typical value of p is between 0.5 and 0.75 [18]. To derive the p value in modern FPGAs, we counted the number of wires around the CLB tile in the Virtex-II Pro device. As a result, we obtained the value t of 936. Figure 4.2 depicts the relationship between the number of

wires and the number of CLB tiles. The number of wires is taken from [64]. In addition, the number of CLB tiles are depicted between parentheses on the x-axis. Figure 4.2 depicts that as the number of tiles increases, the number of wires increases in a *linear* manner. This is due to the fact that CLB tiles and wires are regularly structured in an island style. The value of Rent's exponent p is in the range of [0.78, 0.87], as depicted in Figure 4.2, which is greater than the typical range [0.5, 0.75]. This indicates that the routing wires are dominant (but become critical) resources in the functional plane of the Virtex-II Pro FPGA device.

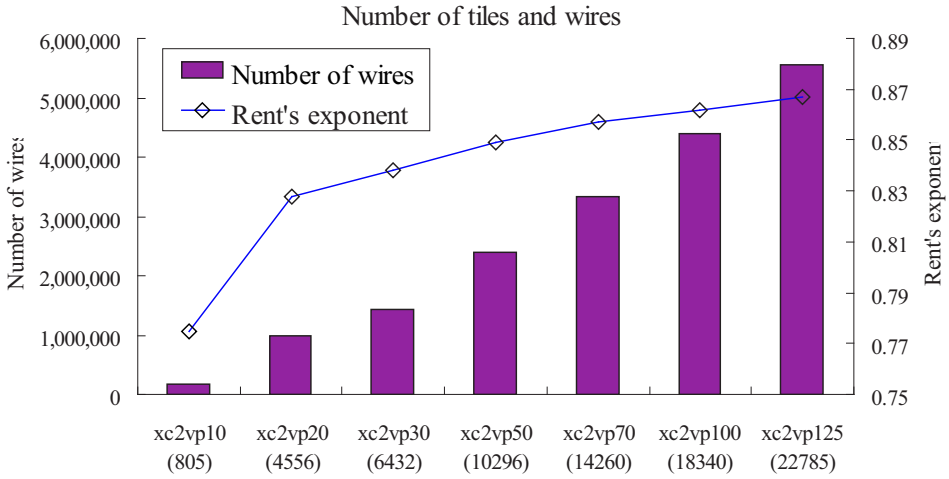


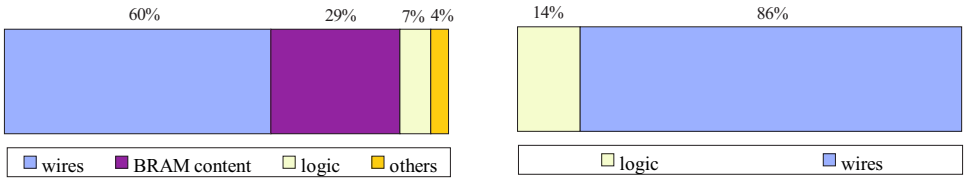
Figure 4.2: Total number of wires in the Virtex-II Pro device series.

Configuration plane: In the configuration plane of the FPGA, a bitstream is dominated by the configuration data for wires. This can be quantified by $\frac{\text{Number of frames for wires}}{\text{Total number of frames}}$ because the frame is the atomic configuration unit. Table 4.1 shows the portion that on-chip resources represent in the bitstream for the Virtex-II Pro xc2vp30 device. Accordingly, wires represent $(46 + 2 + 8) \times 19$ or 1064 frames. The xc2vp30 device contains 1756 frames in total. The routing wires represent $\frac{1064}{1756}$ or 60% of the bitstream, while the logic slices represent only 7% as depicted in Figure 4.3(1). Furthermore, 22 frames are required to configure a CLB cell and 19 frames are dedicated to only wires as depicted in Figure 4.3(2). Therefore, the majority of the bitstream (or the configuration memory) is represented by the un-utilized wire resources. Figure 4.4 depicts the configuration time and the portion that wires consume. The configuration time is derived by (number of frames)$\times$(required time per frame), where the required time per frame

is 16.5 us as explained in Chapter 2. Figure 4.4 clearly indicates that the routing wires significantly contributes to the configuration time. Therefore, it is required to efficiently utilize the abundant wires by *configuring only necessary resources at the desired time*.

Table 4.1: Components of a bitstream for Virtex-II Pro xc2vp30 device.

Components	Columns	Frames per column	Frames for wires per column
IOB	2	4	0
IOI	2	22	19
CLB	46	22	19
BRAM	8	64	0
BRAM interconnects	8	22	19
GCLK	1	4	0



(1) Configuration memory (2) CLB

Figure 4.3: Percentile occupation of a bitstream.

Motivational examples: Dynamically changing network topology is often required [92]. These dynamic wiring is beneficial in cases such as:

- *Dynamic module replacement*: New functionalities can be dynamically plugged in the device as presented in [55]. In this case, it is required to interconnect to (or between) these new modules.
- *Multi-step computation*: Even a single application likely contains multiple subroutines that benefit from different topologies. In this case, the dynamic wiring can be beneficial especially when the communication patterns of these subroutines are temporally local. As an example, the Linpack benchmark application mainly consists of two routines, namely *dgefa* (factorization of matrix) and *dgesl* (solving linear equation with back-substitution). The main subroutines in *dgefa* routine are *idamax* (finding maximum) and *dgemm* (matrix multiplication). Additionally, *dgemm* is a main subroutine in *dgesl* rou-

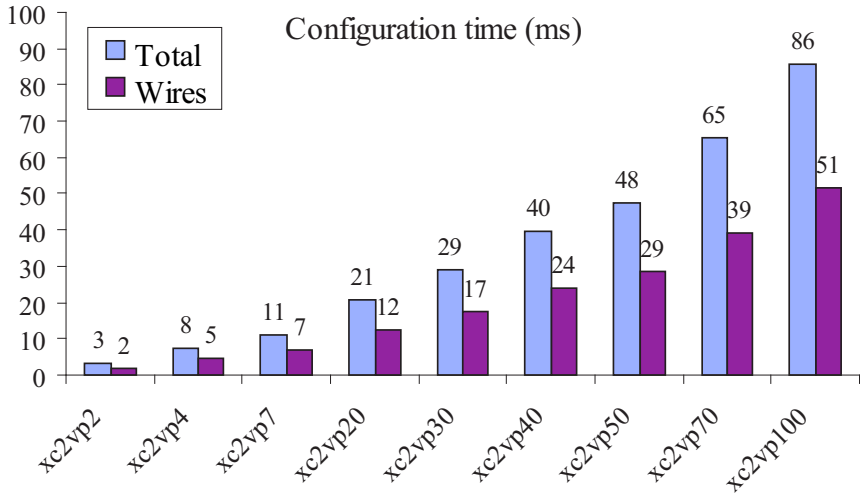


Figure 4.4: Configuration time in Virtex-II Pro device series.

tine. In this case, possible topologies for the subroutines can be a binary tree (for *idamax*) and a 2D-torus (for *dgemm*), as depicted in Figure 4.5.

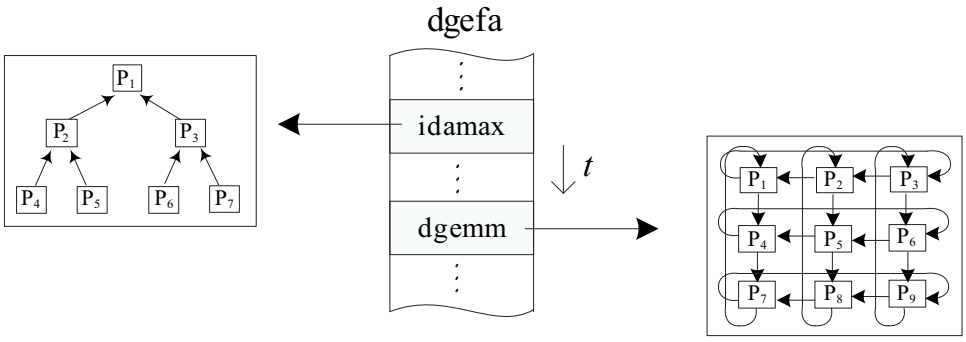


Figure 4.5: Applications and topologies.

4.3 Implementation

This work is based on the general concept of on-demand reconfigurable interconnects in [79], in which the interconnections are established on demand before or during program execution. In this section, reconfigurable point-to-point (ρ-P2P)

interconnects are presented to realize on-demand interconnects as an viable implementation methodology utilizing reconfigurable wires in the modern FPGAs.

Overview: The proposed ρ-P2P interconnect directly[1] interconnects processing elements (PEs) as depicted in Figure 4.6(1). The network topology is implemented as a partially and dynamically reconfigurable component in a chip. PEs are located in a static region and interconnects are located in the reconfigurable region. We utilize pre-fabricated native wire segments to construct the topology. The topology component is modular and can be replaced by other topologies. Anchor points reserve wires to interconnect the reconfigurable components and the static components. Figure 4.6(2) depicts the exemplified reconfiguration steps where the PEs are initially interconnected in a 2D-mesh. Subsequently, the on-demand topology is reconfigured by updating the partial bitstreams only for the reconfigurable topology component during the respective times $t_2 - t_1, t_3 - t_2$.

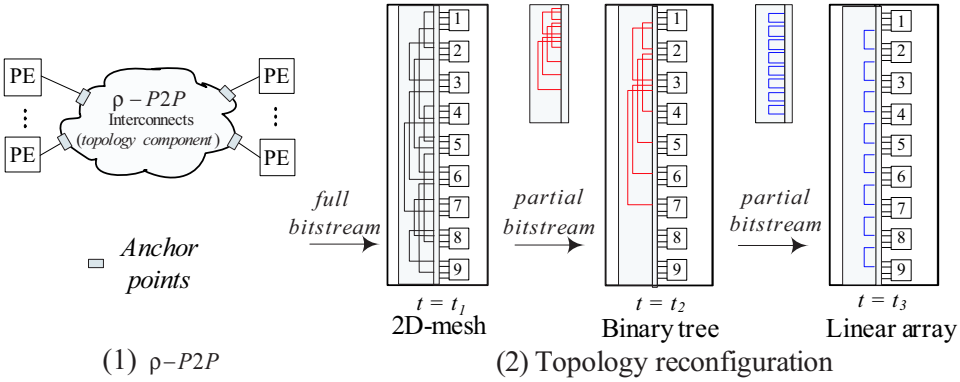


Figure 4.6: The ρ-P2P interconnects.

When the required communication patterns change, the physical interconnects can be (quickly) adapted. The reconfiguration latencies are directly proportional to the corresponding partial bitstream sizes. The bitstream size is determined by the required on-chip resources. We exploit the partial reconfiguration technique in modern FPGAs, using which we can implement our topology reconfiguration approach as a proof of concept. The layout of the static region can be identical for each system configuration and remains unchanged during the interconnects reconfigu-

[1]Logically, the P2P can be classified as zero-hop interconnects because no arbitration is required in the interconnect. Physically, the point-to-point link is implemented using a single or multiple *pip* switches and typically requires multiple (switched) wire segments.

ration. The small-sized topology components can be reconfigured while PEs are in operation within the static region. These static regions can be composed of IPs such as application-specific PEs or general-purpose processors. Though the design flow allows the reconfiguration of these IPs, we consider these IPs are static and the configuration of the static region is out of the scope.

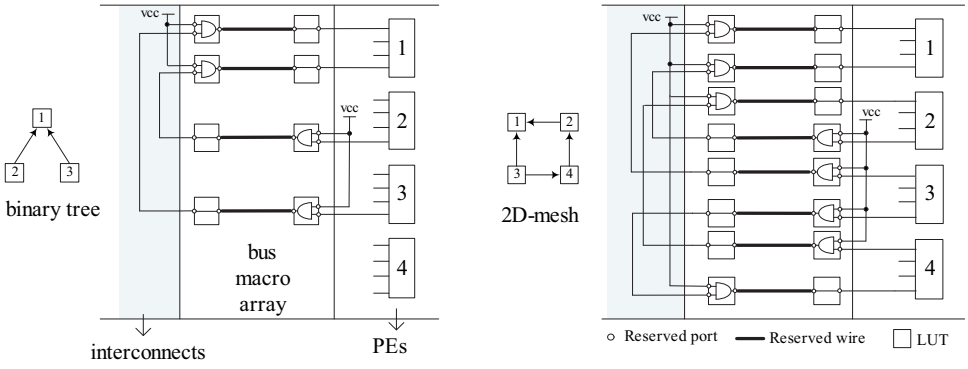


Figure 4.7: The topology implementation using the LUT-based bus macro array.

Topology implementations: In this work, we target the Virtex-II Pro device, in which the configuration frame spans the full vertical height. We use bus macros to implement the anchor points in Figure 4.6(1). The bus macro can be implemented using symmetrical buffer arrays. The reconfigurable region and bus macros constitute the topology component. In this work, we implemented LUT arrays to construct the 16-bit bus macros in a handcrafted manner. Hübner, *et al.*, introduced the use of LUT-based bus macro in [59], where interconnects are implemented utilizing a modern partial and/or dynamic reconfiguration technology. In [59], the buses are static and these LUTs are operated as an interconnect itself to interconnect computation components. However, LUTs in our work operate as anchor points to decouple computational components and communication components. Figure 4.7 depicts the design of a topology component using bus macros as an example. The interconnects are enabled or disabled by controlling the LUT input signals. The interconnects do not require logic resources such as slices except the power and ground signals.

Experiment: The run-time reconfiguration of the interconnects was realized in the Virtex-II Pro xc2vp30 device on the Digilent XUP-V2P prototyping board. Figure 4.8 demonstrates the example procedure of the partial run-time reconfiguration. Each sub-module was implemented in VHDL using the ISE 8.2 tool. Actual

interconnects in Figures 4.8(2) and 4.8(4) correspond to the topology components depicted in Figure 4.6. As an example, we reconfigured the binary tree interconnects by updating partial bitstream (Figure 4.8(4)). The layout of the static region (Figure 4.8(1)) is identical for each system configuration and remains unchanged during the interconnects reconfiguration.

Table 4.2 shows the routing analysis, which shows the number of utilized nets with delays, bitstream utilization, and configuration latency. An an example, 896 nets require less than 2 ns delay for the 7-node binary tree interconnects. As described in Chapter 2, the Virtex-II Pro xc2vp30 device contains 1756 frames in total. The partial bitstream size in number of frames is only 26 (for 7-node binary tree) and 48 (for 9-node mesh) out of 1756. This means that updating of 1.5% (for binary tree) and 2.7% (for mesh) of frames in the entire chip is only required. The xc2vp30 device requires 29 ms to configure an entire chip. The required reconfiguration latencies to change topology are 670 us (for binary tree) and 1011 us (for mesh). This means that only 2.3% (for binary tree) and 3.5% (for mesh) of the configuration time were only required. These partial reconfiguration latencies includes *overheads*, such as initialization, padding frames, and CRC checks. Therefore, the reconfiguration latency can be significantly reduced by utilizing these partial bitstreams. Additionally, we have counted the actual *set bits* in the bitstream. The total bitstream size for the entire chip is 11 Mbits. In the partial bitstream, only 3620 bits (for binary tree) and 6439 bits (for mesh) were set to 1. As shown in Table 4.2, the set bits for the partial bitstream is 0.03% (for binary tree) and 0.06% (for mesh) out of total bitstream. This means the actually necessary on-chip resources are small.

Table 4.2: Routing analysis.

	Net delay(ns)				Partial bitstream				Config. time	
	<2	<3	<4	<5	Frames	%	Set bits	%	us	%
7-node Tree	896	16	32	16	26	1.5	3620	0.03	670	2.3
9-node Mesh	1040	0	16	0	48	2.7	6439	0.06	1011	3.5

4.4 Conclusions

In this chapter, we presented a novel use of wiring flexibility in modern FPGA technology to implement dynamic network topology. We presented partially reconfigurable FPGA interconnects to implement on-demand network topologies. In our implementation, arbitrary topologies can be realized by updating a partial bit-

stream for the ρ-P2P interconnects. We considered P2P-based soft interconnects to make the overlay layer and fabric layer as close as possible, while the underlying fabric itself is the P2P interconnect. We experimented with the Virtex-II Pro device while our approach can also be suitably applied to LUT-based FPGAs such as partially reconfigurable Xilinx Virtex series devices. The experiments show that the utilization of our ρ-P2P interconnects is feasible and the topology reconfiguration latency can be significantly reduced using a partial reconfiguration technique.

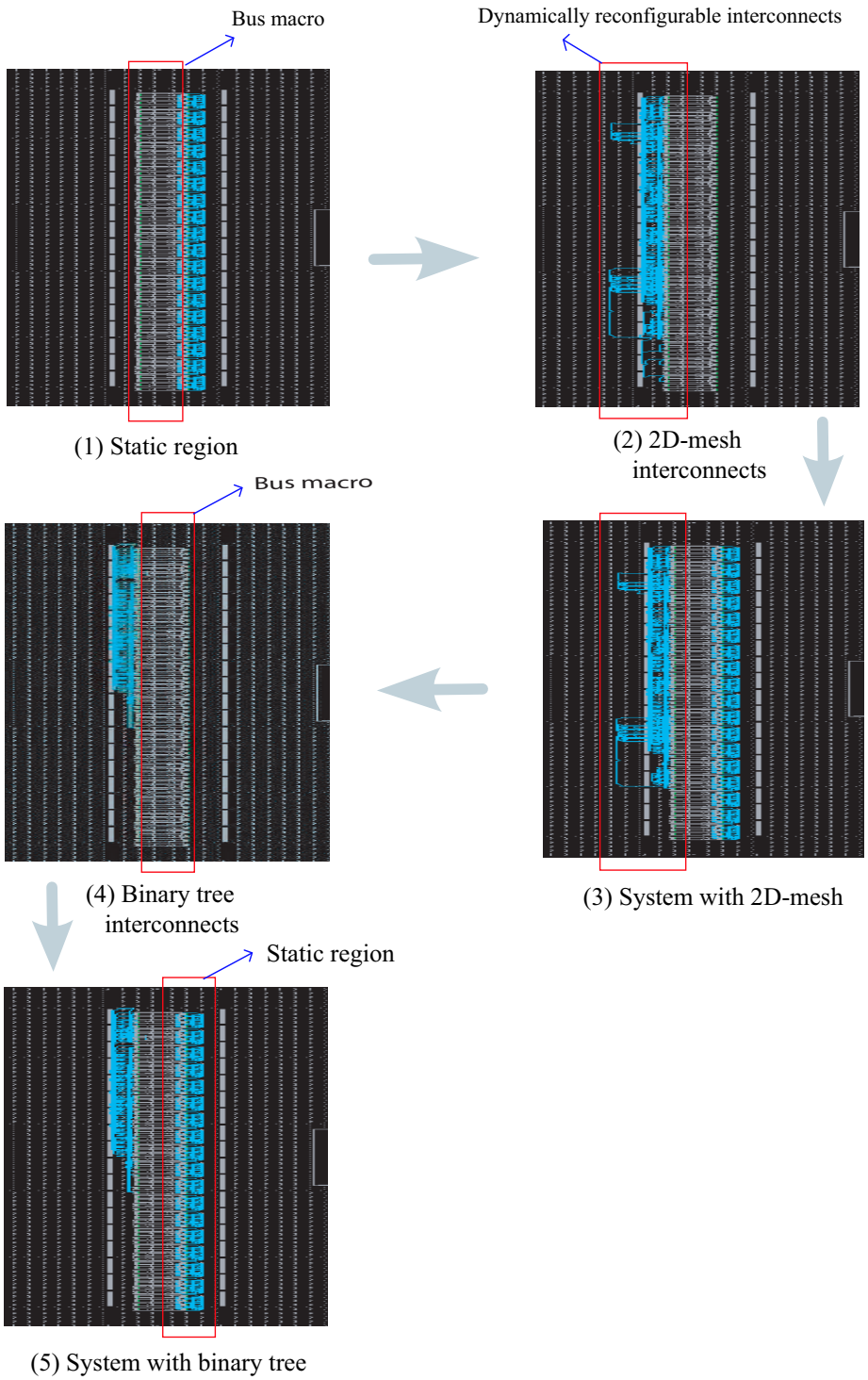


Figure 4.8: Partial run-time reconfiguration.

Chapter 5

Hardwiring Crossbar Interconnect Fabric

In the previous chapters, we studied the soft application-specific overlay interconnects implemented using reconfigurable on-chip resources. However, the reconfigurability is traded with the reduced functional performance and configuration overheads. This is mainly due to the bit-level reconfigurable interconnects. In this chapter, we propose to hardwire crossbars as a interconnect fabric to improve the functional performance and reduce the configuration overheads. We discuss the general advantages of the hardwired interconnect fabric for the inter-IP communication. Considering a soft interconnect as a reference, an analysis is conducted to analyze the performance and cost of the hardwired interconnect fabric. For this purposes, we conduct an MJPEG case study using the Jackson's queuing model to analyze the interconnect performance.

This chapter is organized as follows. Section 5.1 presents our motivations and describes the hardwired crossbars. Section 5.2 presents our performance analysis of the hardwired and soft crossbar interconnects with a case study. Section 5.3 presents the experimental results. Considering the performance per cost as a metric, the soft and hardwired crossbars are compared. Finally, Section 5.4 summarizes and concludes this chapter.

5.1 Introduction

In FPGAs, any overlay network functionality can be implemented utilizing the reconfigurable resources in the fabric layer. However, the interconnect fabric itself has the following limitations [50]:

1. **Functional performance and cost**: Compared to an ASIC, FPGAs are slow. FPGA implementations are reported to occupy 35 $\times$ more area, operate 3.5 $\times$ slower, and use 14$\times$ more energy [42], when compared to ASIC implementation using the same technology. This is mainly due to the bit-level reconfigurable interconnects. Interconnects account for more than 60% of delay, 75% of area, and 80% of power consumption [9]. Moreover, inter-IP communication functionality requires (usually significant) on-chip logic resources.

2. **Granularity**: In the functional plane, inter-IP communication is mostly required to be coarse grained (for example words, flits, packets, or messages). However, the underlying fabric is still operating at bit-level. The fine-grained interconnects must be used to implement any intra-IP functionality with the desired granularity. Due to these different requirements, inter-IP and intra-IP interconnects should be designed differently. It can be noted that the interconnect fabric in modern FPGAs does not distinguish between the intra-IP and inter-IP interconnects.

3. **Wire delay and variation**: The (bit-level) wire delay in the soft interconnects are highly unpredictable before placement and routing step. Subsequently, it may be difficult to meet the timing requirements at design-time due the net delay skew.

4. **Partial reconfiguration**: When a chip is dynamically reconfigured, the functional interconnect must be (partially) reconfigured. The computation IPs are typically rectangle-shaped and they can be efficiently configured by the frame-(or module-) based configuration circuit. Typically, bus macros are required to be geographically spread out between different modules [95]. The partial reconfiguration of the soft interconnects is subsequently not efficient because the interconnect is spread over large surface area. In this case, the network IPs occupy significant reconfigurable resources and a significant portion of the configuration memory (or bitstream) is unnecessarily allocated for the partial reconfiguration of the inter-IP interconnects.

These problems can be solved by implementing the interconnects directly in (*hard*) silicon rather than configurable elements of the FPGA. First, the performance of the hardwired interconnect is improved and occupies less area than the soft interconnects, as described in the following sections. Second, the granularity problem can be solved by implementing the hardwired interconnect at a coarser grain level. Third, the hardwired interconnect is a pre-verified IP and provides highly predictable timing information. Fourth, the partial reconfiguration is highly efficient because the hardwired fabric and reconfigurable fabric are by nature decoupled. In addition, no bus macro is necessary for the partial reconfiguration. Furthermore, the configuration memory is better utilized for the intra-IP functionality because only the soft IP requires configuration. However, the interconnect in the FPGA becomes less flexible because some silicon area is committed to a fixed function. For example, a number of design parameters such as bit-widths, crossbar switch sizes are fixed at fabrication time. The loss of flexibility in the inter-IP communication can be compensated by the increased flexibility in the intra-IP implementation. This is due to the fact that the hardwired interconnect (mostly) does not require reconfigurable resources and these resources can be utilized for the intra-IP implementation. Furthermore, the problem of the loss of flexibility can be alleviated by designing the hard interconnect for the worst-case. Therefore, the gain in performance and cost can outweigh the loss of flexibility.

A soft shared bus is widely used for FPGA platforms. When the bus fabric is hardwired, the available bandwidth in the hardwired bus significantly increases because of the increased clock frequency. The bus component is only needed to be instantiated as a hard macro. Accordingly, the contention probability of the interconnect is reduced, which means that the hardwired bus performs better than the soft bus. However, because many buses are sequential they suffer from traffic congestion before concurrent interconnects do. Goossens, *et al.,* proposed that the *unified* and *reprogrammable* NoCs are directly implemented in silicon [50]. Though the regular hardwired NoC (HWNoC) is a promising interconnect fabric, in this chapter, we propose to hardwire crossbars in FPGAs to bridge the gap between the soft interconnects [95] and the future hardwired NoCs [50]. The main advantage of a crossbar is that minimum traffic congestion occurs inside the crossbar because the dedicated interconnects are physically established. Data transactions inside a crossbar can be fully parallel. Though an area cost is a bottleneck, the area of the crossbar can be adequate for small-sized, for example up to 16 ports. Figure 5.1(1) depicts the hardwired crossbars as built-in components in FPGAs. Figure 5.1(2) depicts the transaction protocol in [40]. Figure 5.1(3) and (4) depict a possible physical hard & soft interface for the Xilinx FPGA layout.

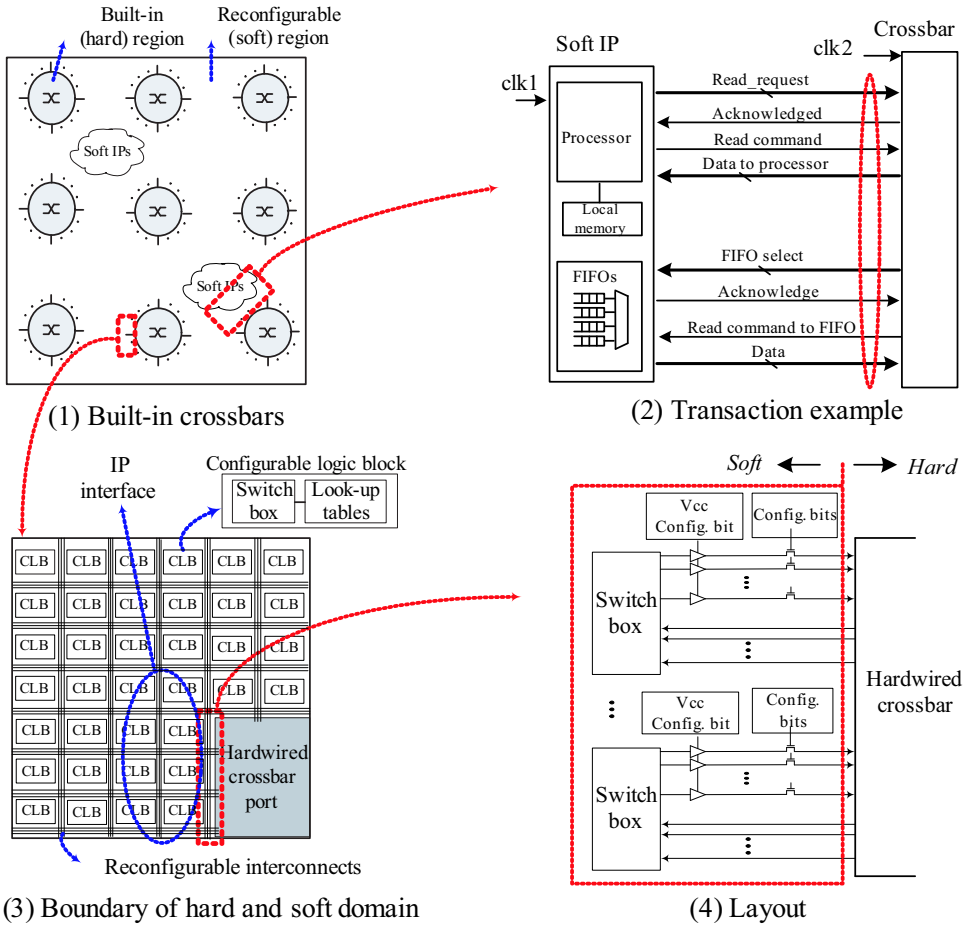


Figure 5.1: Built-in crossbars and physical interface in FPGAs.

5.2 Performance analysis

In this section, we present the performance analysis of hard and soft on-chip crossbars with a case study. As described earlier, we utilize a priori information such as the traffic information (specified in the application) and the physical information (provided by the pre-verified IP components). To achieve this, we derive an approximated service time and utilize the Jackson's open queuing model [45] for the comparative analysis. For the analysis, we reuse physical specifications of network IPs in [47][48]. As described in Chapter 2, the network response time (or the sojourn time) for the Jackson's model is represented as:

$$T_{sojourn} = \frac{1}{\lambda} \sum_{i=1}^{N} \frac{\lambda_i}{\mu_i - \lambda_i}, \tag{5.1}$$

where N is the number of individual queuing systems. λ is the incoming arrival rate of the entire system. λ_i is the incoming arrival rate of the i^{th} queuing system. μ_i is the service rate of the i^{th} queuing system. In addition, this model is useful in that an average buffer size can be directly obtained from the formulation. $\frac{\lambda_i}{\mu_i - \lambda_i}$ corresponds to the buffer size of the i^{th} queuing system. Subsequently, $\sum_{i=1}^{N} \frac{\lambda_i}{\mu_i - \lambda_i}$ corresponds to the number of items in the entire queuing system.

Figure 5.2 depicts our system model for an MJPEG application. A task graph with 7 logical connections is depicted in Figure 5.2(1a), where the numbers on the edge indicate the minimum bandwidth requirement of an application. The bold line represents the streaming data path for an application. The corresponding network of queues is depicted in Figure 5.2(1b). There are 7 queuing systems (numbered (1) to (7)), i.e., $N = 7$. Figure 5.2(1c) depicts the individual queuing systems for each logical connection. The queuing system comprises of the waiting queue and the server. A server is the network component that provides transportation service. To compute λ_i in Equation (5.1), we utilize the bandwidth distribution information in Figure 5.2(1a). As an example, λ_1 is computed by $\frac{62}{62+0.6+1+0.6}\lambda = 0.97\lambda$. Therefore, for a given λ, the network response time is determined from μ_i. Figure 5.2(2) and (3) depict the traffic mapping onto different crossbars, where a connection consists of two logical channels, a *request* and a *response* channel. Our aim is to derive a relative performance and determine network parameters by an application mapping. The network parameters (for example, buffer sizes) are usually dimensioned at design time in a conservative manner.

5.2.1 Network service rates

We formulate the *network* service rate μ_{token} that does not include the computation time.

Network service rate: The network service rate μ_{token} for a single token can be derived by:

$$\mu_{token} = T_{token}^{-1} = (T_{arbit} + T_{transmit})^{-1}, \tag{5.2}$$

where T_{token} is the delivery time for a single token, from the first word (in the departure queue) to the last word (absorbed by the destination IP). A *token* is the

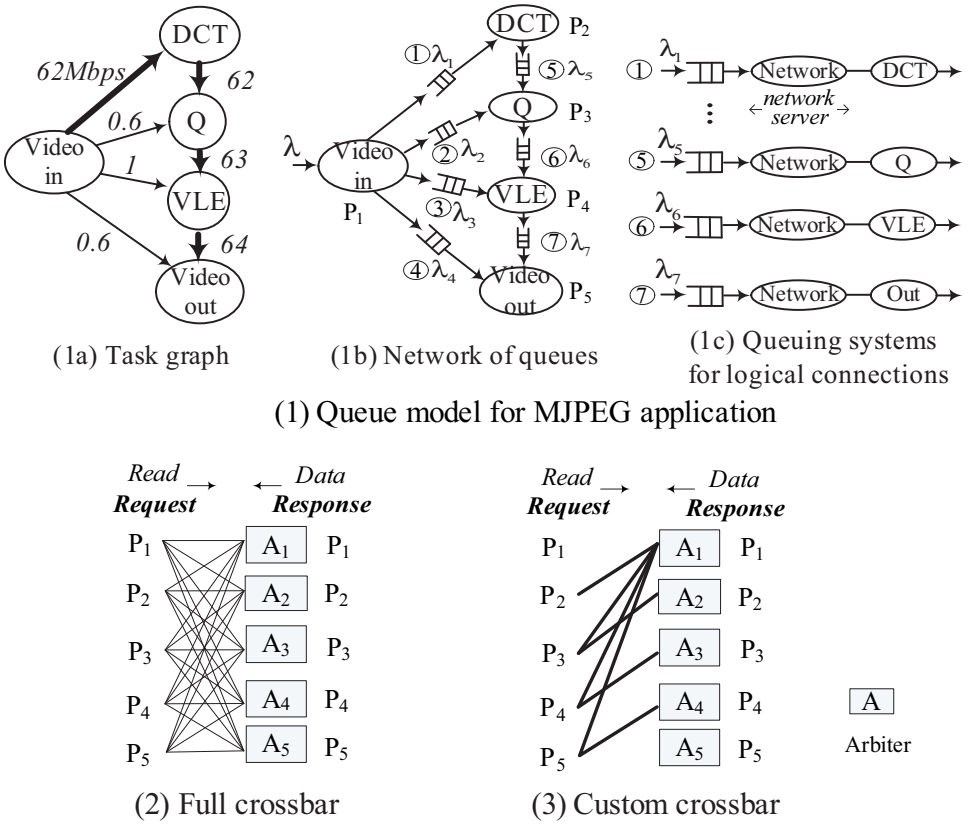


Figure 5.2: Queue model for MJPEG application and mapping onto networks.

primitive communication unit and consists of set of words. T_{arbit} denotes the arbitration time. $T_{transmit}$ is the actual data transmission time.

Network performance: The network response time can be derived by substituting the μ_{token} in Equation (5.1). We consider that the token size is 3 words to make the communication arbitration-intensive. As an example, when token size is 3 words and data block size is 64 words, 22 ($=\lceil \frac{64}{3} \rceil$) arbitrations are required.

5.2.2 Crossbar analysis

The soft crossbars were presented in Chapter 3. The soft full crossbar (SFBAR) contains all-to-all interconnects and the soft custom crossbar (SCBAR) contains on-demand interconnects. The on-demand interconnects for the MJPEG application

are represented in bold lines in Figure 5.2(3). The arbitration time for a full crossbar T_{arbit_FPS} and a custom crossbar T_{arbit_CPS} is restated as:

$$T_{arbit\_FPS} \approx (\lfloor \frac{\text{\#ports}}{2} \rfloor + C_{hand}) / (Clk_{net}) \tag{5.3a}$$

$$T_{arbit\_CPS} \approx (\lfloor \frac{\text{\#links}}{2} \rfloor + C_{hand}) / (Clk_{net}), \tag{5.3b}$$

where a request check latency is approximated as $\lfloor \frac{\text{\#ports}}{2} \rfloor$ or $\lfloor \frac{\text{\#links}}{2} \rfloor$ cycles. C_{hand} refers to the handshaking latency in number of cycles. The arbitration time T_{arbit} in our crossbar varies with number of ports #ports or logical channels #links. The transmission time $T_{transmit}$ in Equation (5.2) corresponds to the token size, as derived by following:

$$T_{transmit} = \frac{S_{token}}{Clk_{net}}, \tag{5.4}$$

where S_{token} denotes the token size or the number of words. Clk_{net} refers to the clock frequency of the interconnect, which is equivalent to the word rate.

5.2.3 MJPEG case study

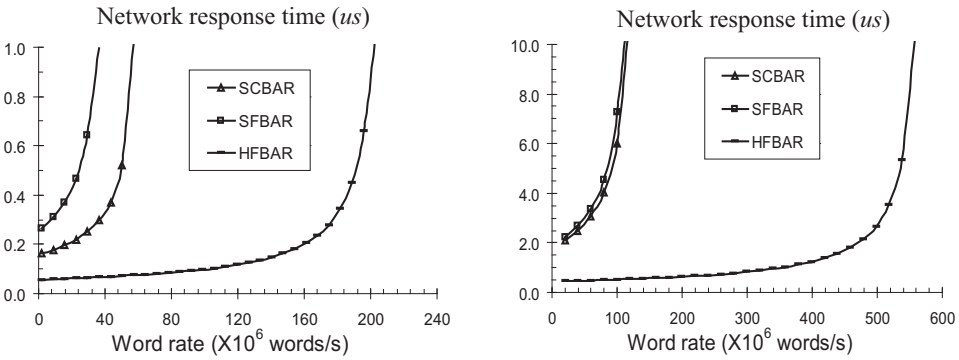
We derive the crossbar and system performance for the MJPEG specification depicted in Figure 5.2(1a). We consider a hardwired full crossbar (HFBAR) for the token size S_{token}=3 words and the number of ports #ports=8 ports. The handshaking latency C_{hand} is 2 cycles and the clock frequency Clk_{net} is 446MHz (see Section 5.3).

Network service rate: The network service rate μ_{token} in the HFBAR is derived as follows. Since C_{hand} = 2 cycles and Clk_{net} = 446MHz, T_{arbit_FPS} is derived by $\frac{\lfloor \frac{8}{2} \rfloor + 2}{446 \times 10^6}$ in seconds for Equation (5.3a). Since S_{token} = 3 words, the transmission time $T_{transmit}$ is derived by $\frac{3}{446 \times 10^6}$ seconds. The μ_{token} is derived by substituting T_{arbit_FPS} and $T_{transmit}$ in Equation (5.2). Subsequently, μ_{token} = $\frac{446 \times 10^6}{\lfloor \frac{8}{2} \rfloor + 2 + 3}$= 49×10^6 tokens/s, which is equivalent[1] to 147×10^6 words/s.

Network performance: The network response time is derived by substituting the network service rate in Equation (5.1). Figure 5.3(1) depicts the network performance for soft and hard crossbars. As a result, the throughput of the HFBAR is

[1]Word rate is derived by (Token rate) $\times$ (Token size in words).

5× better than the soft full crossbar (SFBAR) and 3× better than the soft custom crossbar (SCBAR). This is mainly because of the higher clock frequency and faster arbitration. Figure 5.3(2) depicts the network performance for token size of 64 words. In this case, an arbitration occurs only once for the 64-word data block. Accordingly, the transmission time $T_{trnasmit}$ and the clock frequency Clk_{net} are dominant terms to determine the network performance. As a result, the throughput of HFBAR is 4.5× better than the soft crossbars.



(1) Network response time for a single token (token size = 3 words)

(2) Network response time for a single token (token size = 64 words)

Figure 5.3: Crossbar interconnect performance for MJPEG{5,7} application.

5.3 Implementation

In the previous section, we presented the functional performance of the hardwired crossbar fabric. In Section 5.1, we discussed the advantages of the hardwired networks. In this section, we compare hard and soft crossbars in terms of functional cost, configuration cost, and wire delay variations.

Area cost: We implemented the soft and hard crossbars with the 32-bit data-widths to compare the area cost in the functional plane. Table 5.1 shows the result. The area cost of the hard crossbar is 0.11 mm^2 (for 8-port) and 0.29 mm^2 (for 12-port) in a 130nm CMOS technology. For comparison, the die size of the Virtex-II Pro is estimated to be 397 mm^2 (for xc2vp30 device) and 1158 mm^2 (for xc2vp100 device) in [3]. This indicates that the area overhead of the hardwired crossbar is minimal.

Table 5.1: Hardware implementation results.

Crossbar	Type	Size	Area (slices)	Clock Freq. (MHz)
Full crossbar	Soft	8 ports	2048	97
		12 ports	4182	75
Custom crossbar	Soft	MJPEG{5,7}	284	101
Crossbar	Type	Size	Area (mm^2)	Clock Freq. (MHz)
Full crossbar	Hard	8 ports	0.11	446
		12 ports	0.29	410

Wire delay and variation: To analyze the wire delay variation, we placed and routed the soft crossbars for the MJPEG{5,7} topology in the Virtex-II Pro xc2vp30 device. Subsequently, the number of nets and the net delay were obtained. From this data, the average and the standard deviation are calculated. As a result, Figure 5.4 indicates that the variation of the soft interconnects is significant. It can be noted that wire delay and variation of the soft interconnects can be obtained only after the PAR (placement and routing) step. For comparison, the timing information of the hard interconnects are known at design-time.

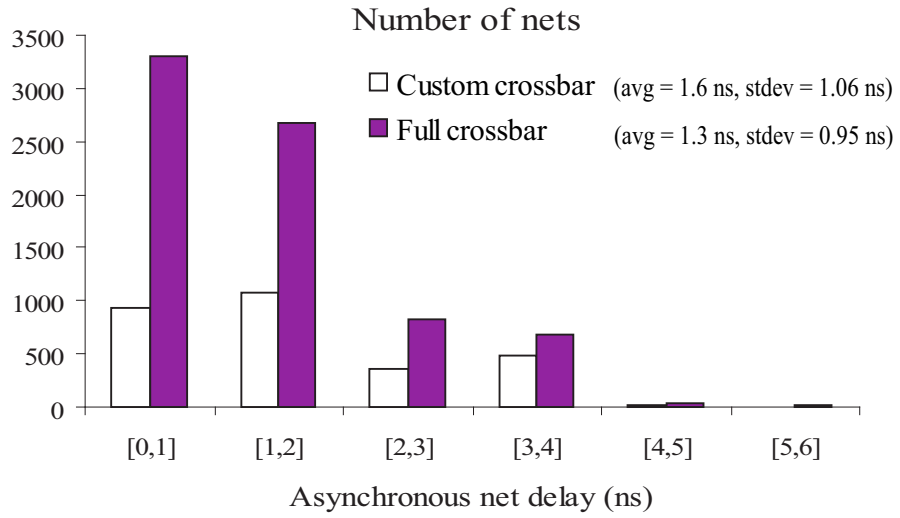


Figure 5.4: Distribution of net delays in soft crossbars for MJPEG{5,7} topology.

Partial reconfiguration: Soft interconnects by nature entail configuration overheads in terms of configuration time, configuration memory, and bitstream. To

analyze the configuration overhead of the soft interconnects, we derive an approximated configuration time based on the utilized area. The configuration time is determined by the required number of frames. The required number of frames varies with placement and routing policies. However, the *lower bound* of the configuration time (or the number of frames) can be derived from the utilized logic slices. Assuming that the utilized logic slices are maximally packed into each frame, the lower bound of the number of frames can be derived as $\lceil \frac{\text{Number of utilized slices}}{\text{Number of slices per CLB} \times \text{Number of CLBs per column}} \rceil \times (\text{Number of frames per column})$. This is due to the fact that the CLB column is the basic coherent unit for the configuration. Table 5.2 shows the mapping result for the MJPEG$\{5,7\}$ topology. As an example, the custom crossbar occupies 284 slices. This means that *at least* $\lceil \frac{284 \text{ slices}}{4 \text{ slices per CLB} \times 80 \text{ CLBs per column}} \rceil \times (22 \text{ frames per column}) = 22$ frames are required. Since a single frame requires 16.5 μs, the configuration time is derived by $22 \times 16.5 = 363$ μs. Figure 5.5 depicts these configuration overheads of our benchmark topologies. The required bitstream size is derived by $(\text{Number of frames}) \times (\text{Number of bits per frame})$. Note that these overheads do not occur when these interconnects are *hard*. By hardwiring interconnects, the reconfigurable resources can be fully utilized for the intra-IP functionality. The on-chip resources in the functional plane and the configuration plane are better utilized when the inter-IP interconnects are hard.

Table 5.2: Mapping soft crossbars in Virtex-II Pro for MJPEG$\{5,7\}$ topology.

Soft crossbar	Resource utilization and lower bound of configuration time		
	Area	Number of frames	Config. time
Full	852 slices	66	1089 μs
Custom	284 slices	22	363 μs

Comparison of soft and hardwired crossbars: We derive the *performance per cost* of the soft and hardwired crossbar interconnects for MJPEG$\{5,7\}$ topology. The performance per cost can be represented by the $\frac{throughput \text{ (words/s)}}{area \text{ (mm}^2\text{)}}$. The throughput is obtained from Figure 5.3 in Chapter 5. The area of the hardwired full crossbar (HFBAR) is obtained from the ASIC implementation. We used the Cadence Encounter tool for 130nm CMOS technology and 0.11 mm^2 is obtained. Note that the Virtex-II Pro adopts 90nm CMOS technology. We derived the 90nm-equivalent area of the 130nm hardwired crossbar by dividing by two ($\frac{0.11}{2} = 0.055$). In Virtex-II Pro, the area of the SCBAR and SFBAR is 284 and

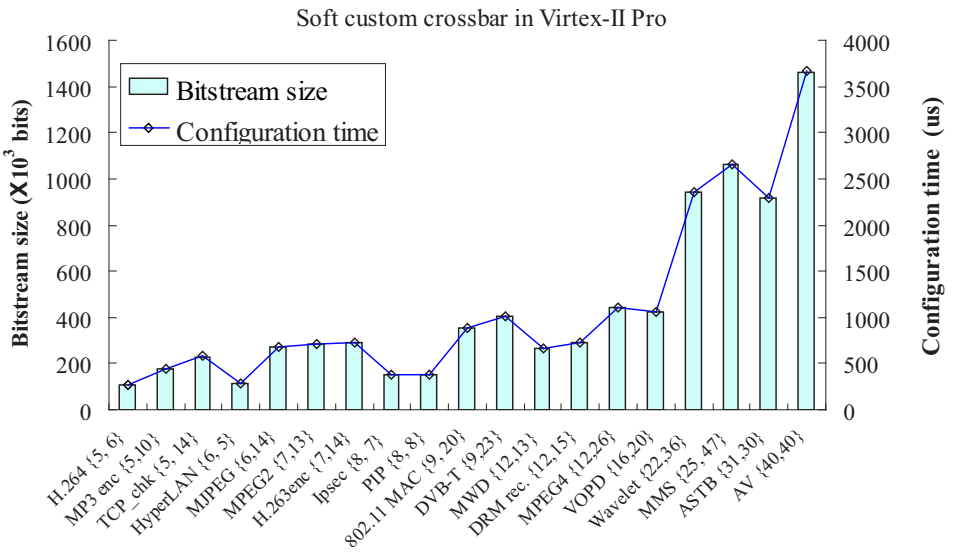


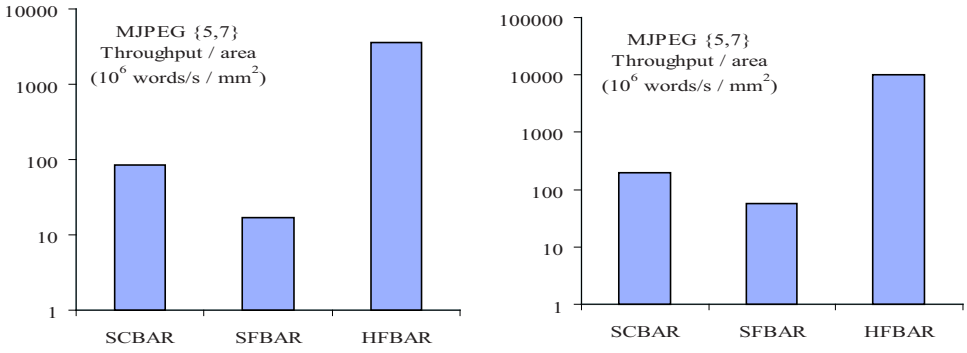
Figure 5.5: Lower bound of bitstream size and configuration time overheads for soft custom crossbars.

852 slices, respectively. We derived the equivalent area[2] of the SFBAR by multiplying by 35 ($0.055 \times 35 = 1.93$ mm^2). We derived the equivalent area of the SCBAR by multiplying by the area ratio ($1.93 \times \frac{284}{852} = 0.64$ mm^2). Table 5.3 shows the throughput and area cost of the soft and hardwired crossbars. In this way, $\frac{throughput\ (\text{words/s})}{area\ (\text{mm}^2)}$ is derived as depicted in Figure 5.6 for the token sizes of 3 words and 64 words. As a result, the HFBAR provides two order of magnitudes better than the SFBAR in terms of $\frac{throughput}{area}$.

Table 5.3: Throughput (words/s) and area (mm^2) for MJPEG{5,7}.

	SCBAR		SFBAR		HFBAR	
Token size	3-words	64-words	3-words	64-words	3-words	64-words
Throughput	54×10^6	128×10^6	33×10^6	109×10^6	195×10^6	550×10^6
Area	284 slices ($\approx 0.64mm^2$)		852 slices ($\approx 1.93mm^2$)		$\approx 0.055mm^2$	

[2]To obtain the area ratio of $\frac{\text{FPGA}}{\text{ASIC}}$, a detailed technology data is required. Kuon, *et al.*, report in [42] that the area ratio of $\frac{\text{FPGA}}{\text{ASIC}}$ is 35. We used this number, while we consider this is a conservative comparison. Another study in [74] reports the area ratio of 31 to 362 and on average 80.



(1) Token size = 3 words

(2) Token size = 64 words

Figure 5.6: Performance per area of crossbars for MJPEG{5,7}.

5.4 Conclusions

Hard interconnects does not use the configurable elements of the FPGA. Aiming to fill the gap between soft interconnects and hardwired NoCs, we proposed to hardwire crossbars as built-in inter-IP interconnect components in FPGAs. We conducted a performance analysis of the hardwired crossbar fabric using the Jackson's queuing model. Our analysis and implementation results indicate that the hardwired interconnect is significantly better in speed, throughput, resource utilization, and partial reconfiguration at an acceptable cost.

Chapter 6

Soft and Hardwired Network-on-Chip

In the previous chapters, we presented the crossbar overlay interconnects and the underlying fabric that can be soft or hard. In general, the crossbars are not scalable in both of the overlay and fabric layers. To solve the scalability problem, this chapter presents the network-on-chip (NoC) overlay and the interconnect fabric. First, we propose a soft customized circuit-switched NoC (CCSN) because the general-purpose NoCs occupy significant on-chip reconfigurable resources. Second, we present the hardwired NoC (HWNoC) interconnect fabric and its effectiveness for the inter-IP communication. Considering the Æthereal NoC [48] as an example, Jackson's queuing model is used to analyze the functional performance of the soft and hard NoCs.

This chapter is organized as follows. Section 6.1 presents the motivations of our work. Section 6.2 presents the soft application-specific NoCs in the overlay layer. Section 6.3 presents the hardwired NoCs in the fabric layer. Section 6.4 presents performance analysis of soft and hardwired NoCs. Section 6.5 presents our implementation for the soft and hard NoCs. Section 6.5 presents simulation results. Finally, Section 6.7 summarizes and concludes this chapter.

6.1 Introduction

In general, wires do not scale as well as logic [72][23]. This scalability problem also holds for the FPGAs. As discussed in the previous chapters, FPGAs contain multi-millions of (short) wire segments. These wire segments are useful for the local communications and are in most cases heavily under-utilized by the applications. The problem lies in the global wires. In Chapter 1, we briefly discussed the general trend, in which the number of wires *linearly* increases as the number of tiles increases. As a result, *the number of wires per tile* does not increase as much as *the number of tiles* increases. Figure 6.1 depicts this relationship. While the number of tiles increases by 28 ($\approx \frac{22785}{805}$) times, the number of wires per tile increases by only 1.18 ($\approx \frac{244}{206}$) times. This means that the (global) wires clearly become the limiting factor in FPGA.

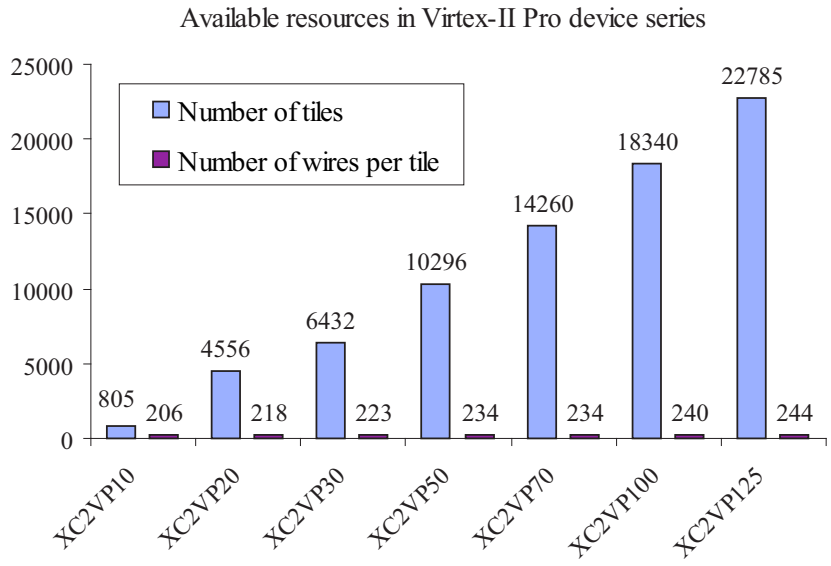


Figure 6.1: Number of wires per tile in Virtex-II Pro device series.

As the global wires are utilized, these are not scalable in terms of wire delay or wire length. Donath, *et al.*, demonstrated in [89] that the average point-to-point wire length increases as number of logic blocks increases. An experimental study in [89] indicates that even in the same technology, an average wire length proportionally grows with $B^{p-0.5}$, where B is the number of tiles and p is the Rent exponent. This means that the average wire length proportionally grows as the Rent's exponent p grows. Note that the Rent's exponent p value is in the range of [0.78, 0.87] in FPGA

which is higher than typical ranges [0.5, 0.75]. Therefore, not only the lack of the global wires but also the average wire delay (or length) become the limiting factor in FPGAs. To solve these problems, a mesh of tree fabric topology is presented in [10] to increase the locality of wires. While the interconnect scheme in [10] is based on dedicated routed-through wires, our approach is to share the global wires using the circuit-switched network on chip. In this chapter, we present the circuit-switched network on chip that is either directly implemented in (*hard*) silicon [50] or customized for the reconfigurable (*soft*) fabric.

6.2 Soft customized circuit-switched NoC

In this section, to reduce the high area cost in the general purpose NoCs, we present a soft *customized circuit-switched NoC* (CCSN) targeting the reconfigurable fabric. Our main approach is to establish only the necessary network resources. We consider the non-customized 2D-mesh circuit-switched-network (CSN) as a reference. Figure 6.2 depicts an example. Figure 6.2(1) depicts a logical topology of a 5-node MJPEG application. Figure 6.2(2) depicts an embedding of the logical topology on the 2×3 2D-mesh topology. The logical topology is embedded in a way that the dilation is minimized. Subsequently, the maximum dilation for the MJPEG application is 2, (for example, from P_1 to P_3). Figure 6.2(3) depicts the corresponding physical 2D-mesh CSN after the topology embedding, where each tile corresponds to the router-IP pair. The general-purpose $M \times N$ 2D-mesh router network has $(6MN - 2M - 2N)$ inter-router half-duplex links. A router internally accommodates network resources such as buffers, intra-router links, and associated control logic. We define the switch wires inside a router as the intra-router links. As an example, the 3-port general-purpose router R_5 in Figure 6.2(3) establishes 6 links around the router, internally 3 buffers, $9\ (= 3 \times 3)$ intra-router links. In this way, total amount of network resources are calculated. Table 6.1 shows the number of network resources for different sized 2D-mesh general-purpose router networks.

The CCSN is obtained using the table-based customization technique as described in the following. A logical connection consists of two channels, a *request* and a *response* channel. We assume that the XY-routing is used for both of the request and response channels. First, the path utilization table is extracted for each router instance after the topology embedding. Second, unnecessary network resources are eliminated in each router using the extracted path utilization table. As an example, Figure 6.2(4) depicts how router R_5 is customized. Router R_5 requires

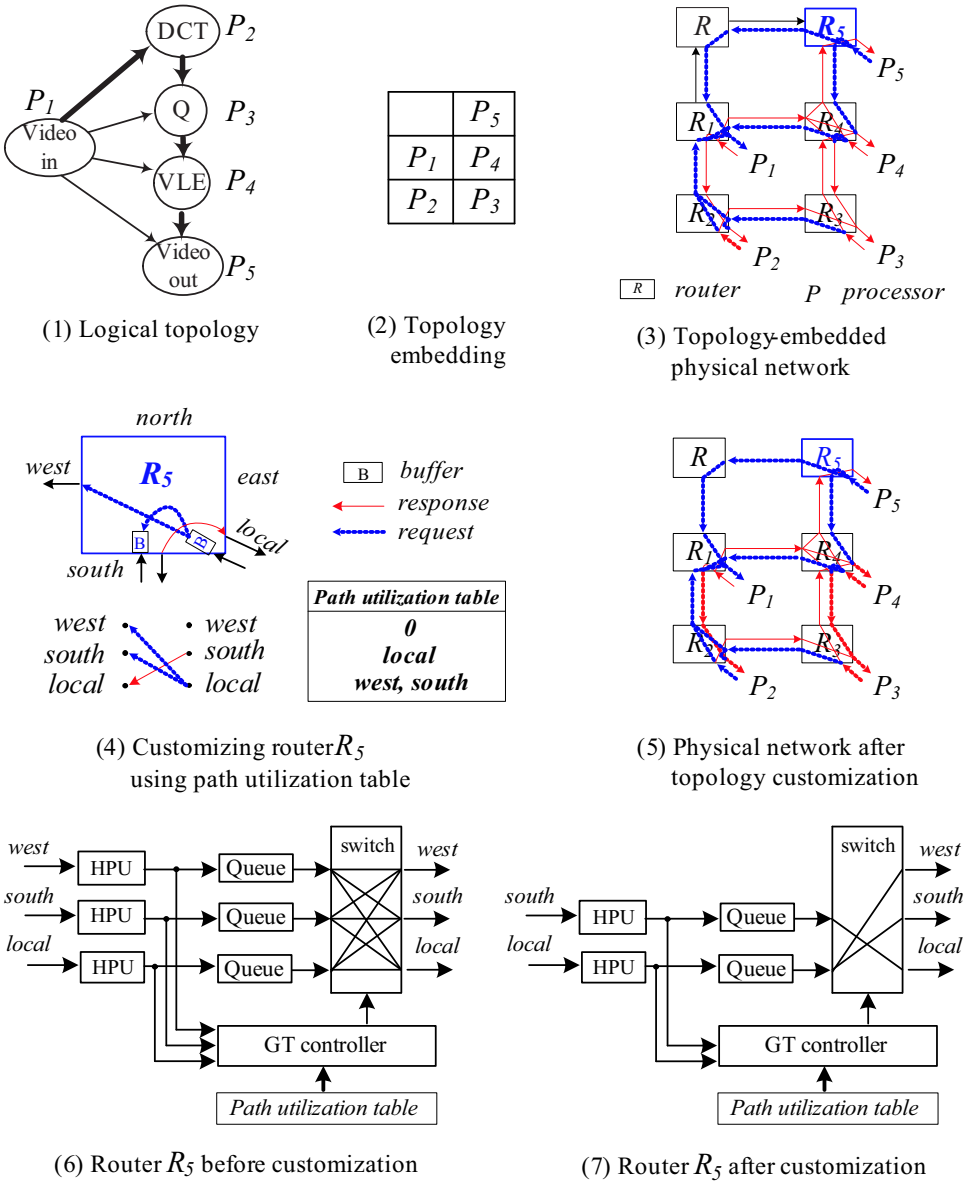


Figure 6.2: Topology embedding of MJPEG application onto physical 2D-mesh topology (1)(2)(3), path utilization table to customize router R_5 (4), customized network (5), router R_5 before customization (6) and router R_5 after customization (7). Note that the topology is customized for both request and return channels.

Table 6.1: Network resources in general-purpose 2D-mesh network.

2D-mesh		2×3	3×3	3×4	4×4	5×5	6×6	6×7	7×7
Number of routers	3-port	4	4	4	4	4	4	4	4
	4-port	2	4	6	8	12	16	18	20
	5-port	0	1	2	4	9	16	20	25
Inter-router links		26	42	58	80	130	192	226	266
Intra-router links		68	125	182	264	453	692	824	981
Number of buffers		20	33	46	64	105	156	184	217

only 3 intra-router links ($south \rightarrow local$, $local \rightarrow west$, $local \rightarrow south$ links), instead of 3^2 links. The path utilization table contains the connectivity information. Moreover, unnecessary buffers are eliminated. As an example, two buffers are required for the $south, local$ ports in router R_5. Figures 6.2(6) and (7) depict how the Æthereal router architecture is optimized for router R_5 using the path utilization table. Consequently, the CCSN is derived and depicted in Figure 6.2(5). Furthermore, unnecessary inter-router links and their associated control logic are also eliminated, because these resources are logically/physically disconnected. As an example, one inter-router link to the $west$ port is optimized away. Table 6.2 shows the utilized resources of the CCSN for the MJPEG$\{5,7\}$ topologies. This indicates that the CCSN utilizes significantly less resources of the general-purpose CSN.

Table 6.2: Comparison between CSN and CCSN for MJPEG $\{5,7\}$ topology.

2×3 2D-mesh	CSN	CCSN	Utilization (%)
Inter-router links	26	22	85
Intra-router links	68	27	40
Number of buffers	20	17	85

Similar to this case study, we obtained the CCSN from the CSN for the benchmark topologies. To achieve this, logical topologies with n-nodes are embedded onto a $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ 2D-mesh. As a result, on average 70% of buffers, 28% of the intra-router links, and 70% of inter-router links are utilized by the applications as depicted in Figure 6.3. The area cost can be accordingly reduced, as presented in the next chapter. It can be noted that our customization method can be applied to the general CSN with any topology. Our topology construction method is similar to [67] in that the switch wires are customized for individual routers. In [67], parameters are specified for an individual multiplexer instance and an arbiter instance.

In our work, a generic topology table is extracted from the topology embedding. Our customization method additionally optimizes the intra-router buffers as well as inter-router half-duplex links. In other words, our table-based customization allows the systematic removal of un-utilized buffers and interconnects in an entire NoC.

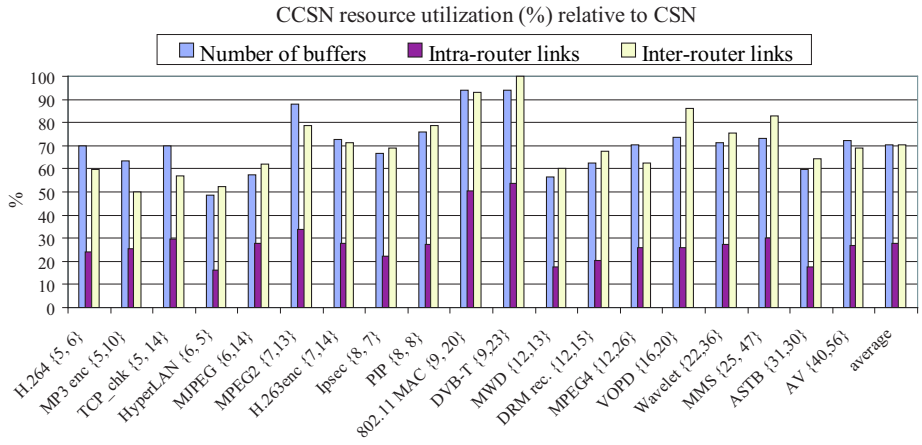
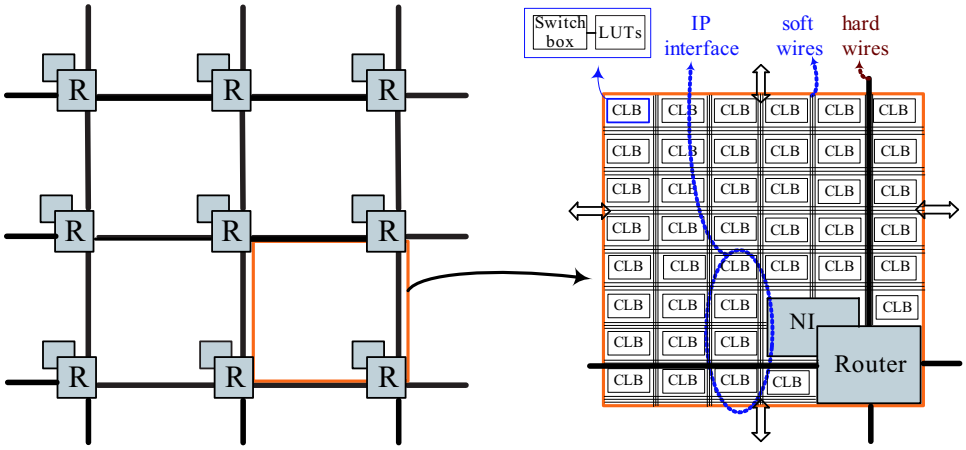


Figure 6.3: Network resource utilization of CCSN relative to CSN.

6.3 Hardwired circuit-switched NoC fabric

In this section, we describe how an FPGA utilizes the hardwired NoC [50] fabric for the global inter-IP communication. This is depicted in Figure 6.4, where the regular 2D-mesh is considered. The legacy reconfigurable wires are utilized for the application-specific intra-IP interconnects. The hardwired NoC (HWNoC) fabric is dedicated to the inter-IP global communication. In Chapter 5, we described the advantages of the hardwired interconnect fabric. In addition to those advantages, the HWNoC interconnect fabric solves the scalability problem of the global wiring in FPGAs. The HWNoC replaces the long global wires with optimized segmented wires [61][91]. This is only possible when the NoC is hard. In the HWNoC, the packets are typically communicated over links, which requires multiple cycles. However, the clock frequency in the HWNoC is significantly higher than the soft networks as described in the next sections. This means that the actual time for the communication in HWNoC can be faster. The traffic locality is also beneficial in the HWNoC because the communication hops are accordingly reduced.



(1) HWNoC in FPGAs (2) Tile

Figure 6.4: HWNoC-based FPGAs.

6.4 Performance analysis

In this section, we present the performance analysis of hard and soft on-chip networks with a case study using Jackson's model. Figure 6.5(1) depicts our model for an MJPEG{5,7} application. Figure 6.5(2) depicts individual queuing systems for each logical connection. Figure 6.5(3) depicts the traffic mapping onto NoCs with 2D-mesh topology.

6.4.1 Network service time

In this section, considering the GT (guaranteed throughput) Æthereal NoC [48], we derive the delay (or the service time) in the circuit-switched network to obtain the latency and throughput performance. The required bandwidth for each logical connection is reserved by allocating time-division-multiplexed slots. The global scheduler in the network interface arbitrates channels based on the allocated slot table and the remote buffer space. When the channel is arbitrated, the transmission time in the router network is derived similarly to a crossbar. We assume that connections are long-lived and ignore the time associated with their set-up and tear-down [11]. The service time is the summation of the arbitration time and the transmission time as derived in the following.

Arbitration time: The arbitration time is determined by the slot size, the slot ta-

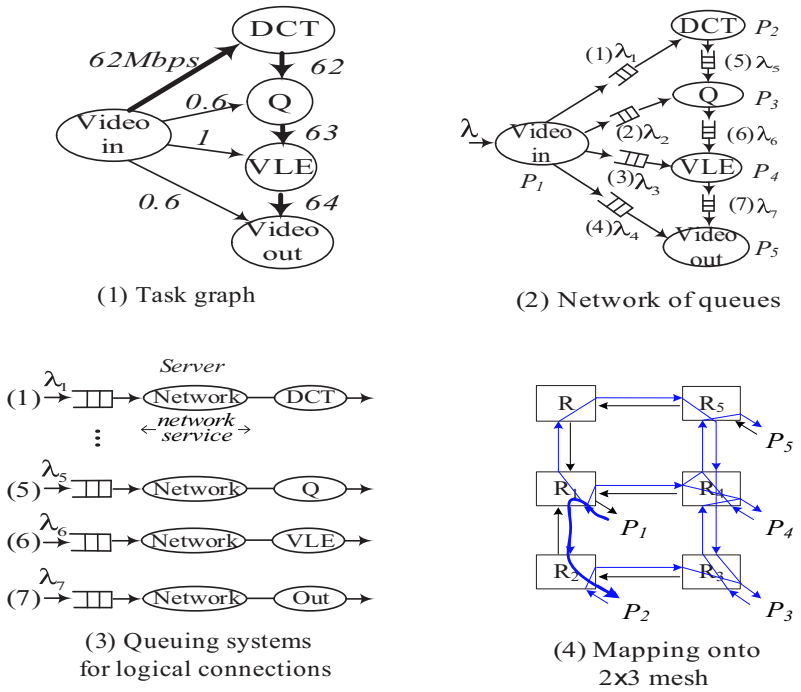


Figure 6.5: Queue model for MJPEG {5,7} application and mapping onto networks.

ble size, and the number of allocated slots for a channel. Assuming that slots are equally distributed in the slot table, the arbitration time T_{arbit} for a token is approximated by:

$$T_{arbit} = \frac{S_{slot} \times \lceil \frac{S_{tab}}{2 \times A_{slot}} \rceil}{f_{net}}, \tag{6.1}$$

where S_{slot} denotes the slot size in number of words. S_{tab} denotes the slot table size in number of slots. A_{slot} denotes the number of slots that is reserved for a channel in the slot table. f_{net} refers to the clock frequency of a network. In this work, a *slot* contains 3 words (1 header and 2 payload words). We divide by 2, since the circular round-robin pointer is statistically located in the middle of the search space.

Transmission time: The transaction consists of *read request* and *data response* channels. The transmission time consists of the packetization time, the time to send data, and the pipeline delay, which can be approximated by:

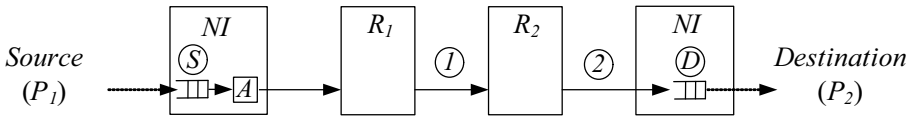
$$T_{transmit} = \begin{cases} \frac{\lceil \frac{S_{req}}{S_{slot-1}} \times \frac{S_{tab}}{A_{slot}} \rceil + \#\text{hop} \times C_{SW} + C_{misc}}{f_{net}} & \text{for } \textit{request} \\ \frac{\lceil \frac{S_{resp}}{S_{slot-1}} \times \frac{S_{tab}}{A_{slot}} \rceil + \#\text{hop} \times C_{SW} + C_{misc}}{f_{net}} & \text{for } \textit{response}, \end{cases} \quad (6.2)$$

where $(S_{slot} - 1)$ refers to the slot size in number of payload words, while a slot contains 1-word of header. S_{req} and S_{resp} denote the token size in the number of words for the request and response channel, respectively. The first term $\lceil \frac{S_{req}}{S_{slot-1}} \times \frac{S_{tab}}{A_{slot}} \rceil$ refers to the number of cycles to send data. This is an approximation because it assumes slots are equally spaced. However, otherwise the first term needs to be split in a *div* term for the number of table revolutions, and a *mod* term for the delay in the last revolution. The second term refers to the pipeline delay. #hop refers to the number of intermediate routers in the routing path. C_{SW} denotes the number of cycles for the switching per router hop. The third term C_{misc} denotes the number of cycles spent in the network interfaces for packetization and de-packetization.

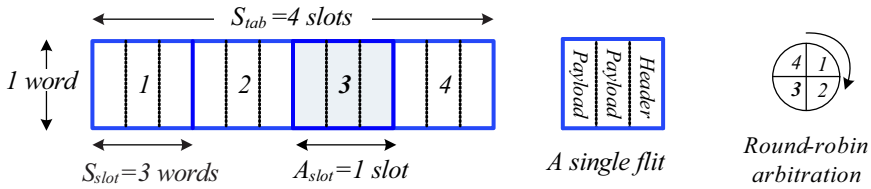
6.4.2 MJPEG case study

We derive the performance of the hardwired circuit-switched network (HCSN). The MJPEG task graph is mapped onto the HCSN with 2×3 2D-mesh topology depicted in Figure 6.5(4). Figure 6.6(1) depicts the connection between P_1 and P_2, where P_1 sends data to P_2 via the response channel. Design parameters of the Æthereal NoC are the following. The slot size S_{slot} is 3 words. The clock frequency of the hardwired network $f_{net} = 500$ MHz from the implementation (see Section 6.5). The switching latency per router hop C_{SW} is 3 cycles from the implementation. The slot table size S_{tab} and number of the reserved slots per channel A_{slot} are derived from the bandwidth distribution. The miscellaneous cycles C_{misc} is 3 cycles, since the request, packetization, and de-packetization requires 1 cycle each. We used the automated design flow of [49] to obtain S_{tab} and A_{slot} for an MJPEG task graph. As a result, S_{tab} is 4 slots and A_{slot} is 1 slot per channel. The arbitration time and transmission time are derived as follows:

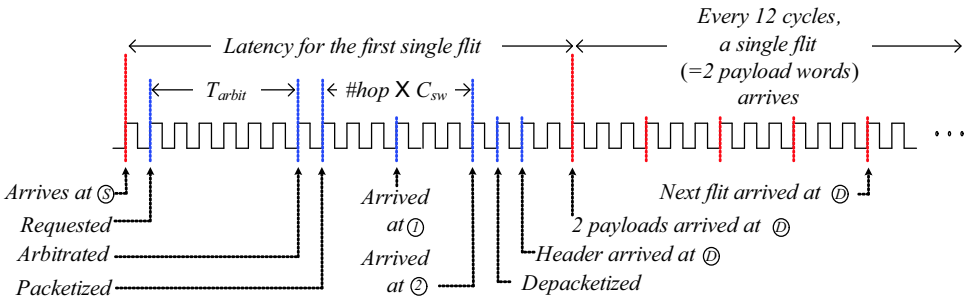
Arbitration time: The arbitration time is derived by substituting the S_{slot}=3 words, S_{tab}=4 slots, and A_{slot}=1 slot in Equation (6.1). Figure 6.6(2) depicts an example. The round-robin pointer points to the 3^{rd} slot and the $1^{th}, 2^{nd}, 4^{th}$ slots are occupied by other channels. Since each slot contains 3 words, the arbitration requires approximately $(3 \times \lceil \frac{4}{2 \times 1} \rceil)$=6 cycles. Subsequently, the arbitration time T_{arbit} is derived by $\frac{3 \times \lceil \frac{4}{2 \times 1} \rceil}{500 \times 10^6}$ = 12 ns per channel.



(1) An established channel between P_1 and P_2



(2) GT scheduling in the source NI



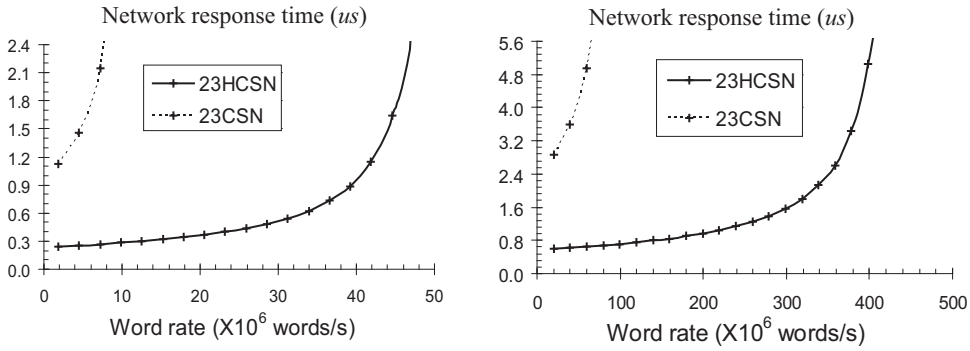
(3) A latency from the source queue to the destination queue

Figure 6.6: An example of the delay model.

Transmission time: Figure 6.6(3) depicts a latency from the source queue to the destination queue. Since S_{slot}=3 words and S_{req}=S_{resp}=3 words, the time to send data is $\frac{\lceil \frac{3}{3-1} \times \frac{4}{1} \rceil}{500 \times 10^6} = 12$ ns. This means that a single flit (2 payload words) can be transmitted per every revolution of 12 (=3 words $\times$ 4 slots) cycles. From the topology mapping and the routing strategy, the number of hops #hop is obtained for each channel. The routing paths are depicted in Figure 6.5(2a) for the response channels. As an example, #hop between P_1 and P_2 is 2 (see bold line in Figure 6.5(4)). Since C_{SW}=3 cycles and C_{misc}=3 cycles, $T_{transmit}$ is derived by $\frac{\lceil \frac{3}{3-1} \times \frac{4}{1} \rceil + 2 \times 3 + 3}{500 \times 10^6}$ = 30 ns for a channel between P_1 and P_2.

Network performance: The network service rate μ_{token} can be derived by $\frac{1}{T_{arbit} + T_{transmit}}$. As an example, the service rate μ_1 for the connection P_1 and P_2 is derived by $\frac{1}{2 \times (12+30)\text{ ns}}$ = 12×10^6 tokens/s. Note that a connection consists

of two channels, the *request* and the *response* channel. The total network response time is derived by substituting individual service rates in $\frac{1}{\lambda}\sum_{i=1}^{N}\frac{\lambda_i}{\mu_i-\lambda_i}$. Similarly, the performance of hard and soft networks with different topologies are derived, as depicted in Figure 6.7(1). In addition, the hardwired NoC is significantly better in latency and throughput than the soft NoC.



(1) Network response time for a single token (token size = 3 words)

(2) Network response time for a single token (token size = 64 words)

Figure 6.7: The network performance for MJPEG $\{5,7\}$ task graph. 23(H)CSN denotes a soft (hardwired) circuit-switched network with 2×3 2D-mesh topology.

6.5 Implementation

In this section, we describe the four experiments we conducted to analyze the cost of soft and hard networks. First, in the functional plane, the soft and hard CSNs are implemented. Table 6.3 shows the results. The clock frequency of HWNoC is $4.3\times$ higher than the soft NoCs. The soft CSN occupies significant logic resources, while the HWNoC does not utilize them. As an example, 3×4 2D-mesh CSN occupies 9802 slices and 22% of total logic resources in the xc2v2p100 device. For comparison, the area of the 3×4 2D-mesh HCSN is $1.21mm^2$ and this is a small part of the die size[1] of the same device. Second, the CCSNs are implemented and the area cost is analyzed for the benchmark topologies. Figure 6.8(1) depicts the area cost. As a result, the CCSN reduces the area by 71% of logic slices on average. Third, in the configuration plane, we derive the configuration cost in terms of the bitstream size or the configuration memory size. Figure 6.8(2) depicts the lower bound of bitstream sizes and the configuration time of the CCSN for our

[1] 1158 mm^2 estimated in [3] for xc2vp100 device.

benchmark topologies. Even though CCSN reduces the configuration bitstream size, still a large bitstream with millions of bits is required. For comparison, the HWNoC does not have these configuration overheads.

Table 6.3: Hardware implementation results.

Soft (90nm CMOS FPGA Virtex-II Pro, XC2VP100)			
Type	Size	Area (slices)	Clock Freq. (MHz)
CSN	2×3 2D-mesh	3450	115
	3×4 2D-mesh	9802	115
Hard (130 nm CMOS ASIC)			
Type	Size	Area (mm^2)	Clock Freq. (MHz)
HCSN	2×3 2D-mesh	0.51	500
	3×4 2D-mesh	1.21	500

Fourth, we derive the *performance per cost* of the soft and hardwired NoCs. Table 6.4 shows the throughput and area cost of the soft and hardwired crossbars. The throughput is obtained from Figure 6.7. In this way, $\frac{throughput\ (\text{words/s})}{area\ (\text{mm}^2)}$ is derived as depicted in Figure 6.9 for the token sizes of 3 words and 64 words. As a result, the CCSN is $3.5\times$ better than the CSN in terms of $\frac{throughput}{area}$ for the MJPEG$\{5,7\}$ application. In addition, the HCSN provides two order to magnitudes better than the CSN.

Table 6.4: Throughput (words/s) and area (mm^2) for MJPEG$\{5,7\}$.

	23CSN		23CCSN		23HCSN	
Token size	3-words	64-words	3-words	64-words	3-words	64-words
Throughput	8.1×10^6	64×10^6	8.1×10^6	64×10^6	45×10^6	396.8×10^6
Area	5502 slices ($\approx 9.01mm^2$)		1298 slices ($\approx 2.13mm^2$)		$\approx 0.26mm^2$	

6.6 Simulation results

In this section, we present the simulation results. We conducted two experiments. First, to verify the analysis, we experimented with a cycle-accurate SystemC simulation for the Æthereal NoC [49] and compare it with our approximated latency. Figure 6.10(1) depicts an average of connection latencies for the MJPEG$\{5,7\}$. The average of connection latency in our analysis is represented by AN. The minimum/average/maximum simulated connection latencies are obtained from the design flow [49]. Max_Sim denotes the maximum experienced latency in the simulation. As depicted in Figure 6.10(1), our analysis provides the same trend as the

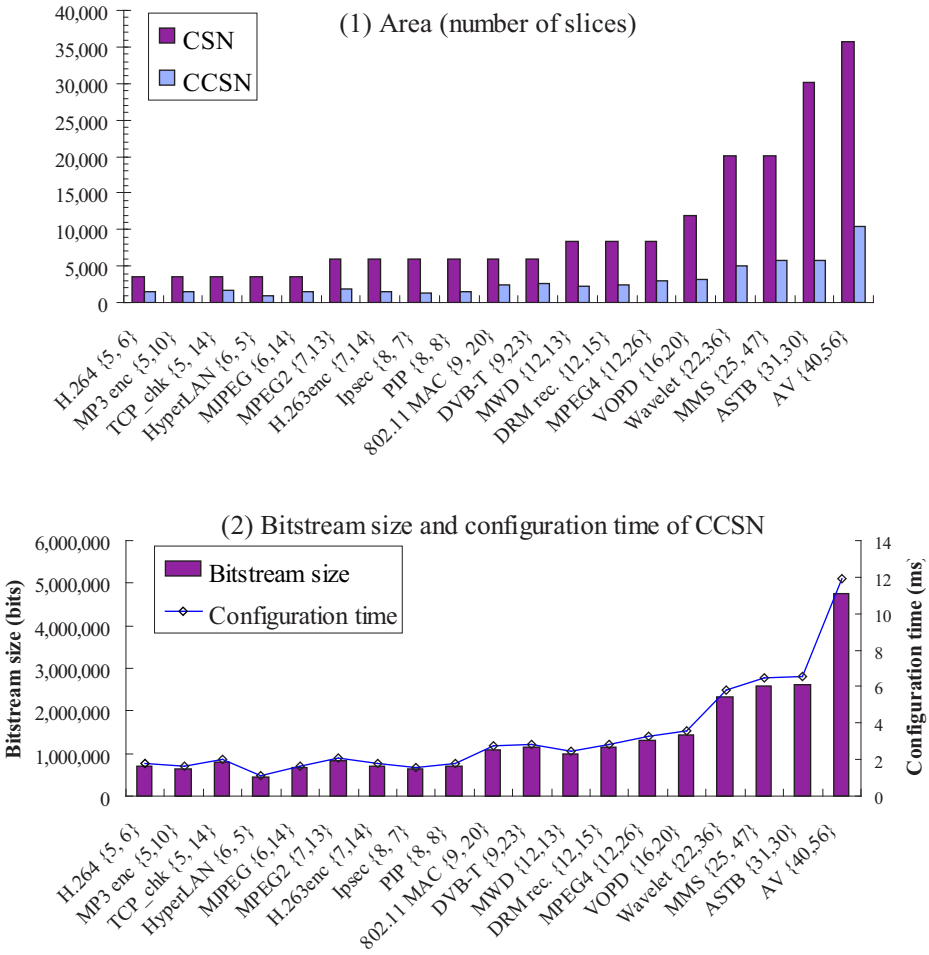
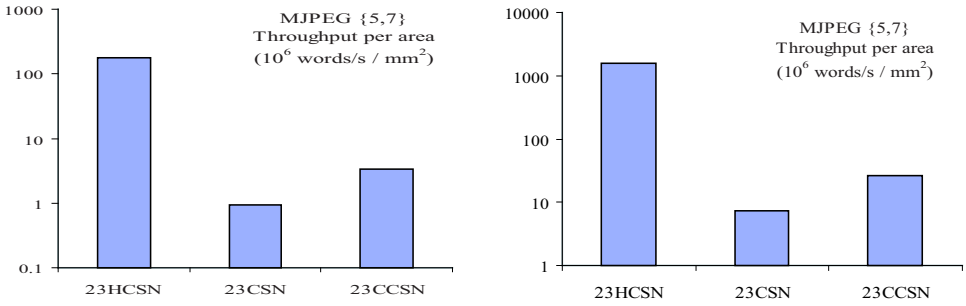


Figure 6.8: Area and configuration overheads of CSN and CCSN in Virtex-II Pro xc2vp100.

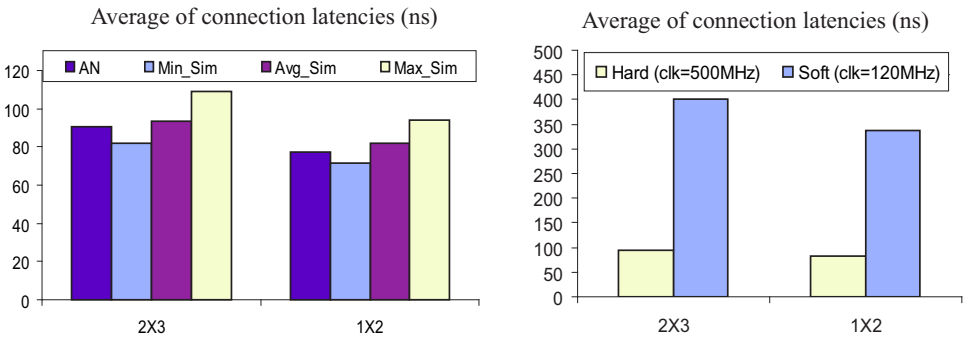
simulation. Second, we compared hard and soft NoCs in the simulation. Figure 6.10(2) depicts an average of connection latencies of hard and soft networks, by changing the clock frequency in the simulation. As a result, on average 4.2× of the latency is reduced in the hardwired network.



(1) Token size = 3 words

(2) Token size = 64 words

Figure 6.9: Performance per area of 2×3 2D-mesh NoCs for MJPEG{5,7}.



(1) Average of connection latencies in hard networks

(2) Average of connection latencies in hard and soft networks

Figure 6.10: Simulation results for MJPEG{5,7} task graph.

6.7 Conclusions

In this chapter, we presented the soft and hard NoCs. By hardwiring NoCs, some flexibility for the inter-IP communication will be lost. The fixed design parameters at the design time includes the network topology, network size, data width, and size of FIFOs in the routers. However, the loss of flexibility can be compensated by dimensioning the networks for the worst-case in the particular targeted device and/or targeted application domains. Moreover, the loss of flexibility in the inter-IP communication results in the increased flexibility for intra-IP logic and interconnects. This is due to the fact that more reconfigurable resources can be allocated to implement the IP functionalities. We derived the delay model and applied Jackson's analysis to analyze the soft and hard circuit-switched networks. Our analysis

and implementation results suggest that the hardwired NoCs significantly improve the performance compared to soft interconnects at an acceptable cost.

Chapter 7

Conclusions and Future Work

While *reconfigurability is a key benefit in FPGAs, it is traded-off by decreased performance and increased cost, mainly because of the bit-level interconnects. Our goal is to reduce the cost and increase the performance for the adaptive inter-IP communication. In the overlay layer, we presented a design and an implementation of the soft customized interconnects. We presented topology customization techniques for the soft crossbar switch, crossbar schedulers, point-to-point interconnects, and circuit-switched NoCs. Compared to the general-purpose interconnects, our customized interconnects maintain lower cost, by establishing only necessary network resources. In the fabric layer, we proposed to replace the bit-level reconfigurable wires by hardwired circuit-switched networks. We presented a scheme of hardwiring networks as an interconnect fabric. Compared to the soft networks, our hardwired network fabric significantly improves network performance with low cost.*

This chapter presents concluding remarks and possible future research directions. Section 7.1 summarizes this thesis. Finally, Section 7.2 presents possible future work.

7.1 Summary

We presented various soft and hardwired networks by presenting the trade-offs in terms of performance, cost, and flexibility, as summarized in the following.

In Chapter 3(**Soft Application-specific Crossbars**), we presented application-specific soft crossbars in the overlay layer. The performance of a parallel application increases when the underlying physical interconnects are *identical* to

the required communication behavior of the application. Our general approach is that the interconnects are traffic-aware and/or technology-aware, such that the logical and physical interconnects in different layers are as close as possible. We presented a topologically customized crossbar designed for reconfigurable platform. Our customized crossbar efficiently utilizes the bandwidth by establishing on-demand on-chip resources. Specifically, our presented crossbar is summarized as follows:

Custom switch: We presented a topology customization technique for crossbar switches, using which the crossbar provides *identical* physical topologies to arbitrary topologies that an application requires. We showed that the custom switch can be implemented using parameterized multiplexer arrays. By utilizing the logical topology as a parameter, the interconnect is adapted to a given application without modifying the implementation. A multiprocessor system using our custom crossbars were implemented and verified with the automated ESPAM design flow. Implementation results show that our custom interconnect reduces the area by 84% and the power consumption by 71%, compared to the full crossbar.

Custom schedulers: We presented a custom parallel scheduler (CPS) for the custom switch. We demonstrated that the CPS can be implemented using parameterized arbiter arrays. Considering sequential (SQS) and full parallel (FPS) schedulers as references, we conducted a comparative analysis using the Jackson's open queuing model. Additionally, we presented the shared custom parallel scheduler (SCPS) that can be beneficial when the number of links per port increases. Furthermore, the SCPS alleviates the scalability problem of a conventional crossbar interconnect by sharing wires.

In Chapter 4(**Partially Reconfigurable Soft Interconnects**), we investigated the wire reconfigurability to devise the adaptive interconnects. An application often entails complex topologies and they may change dynamically. We investigated a method to implement such a dynamic topology. We analyzed the wiring resources in the functional plane. Our analysis indicated that the FPGA is dominated by abundant wiring resources. Our study showed that the wiring resources occupy 60% of the bitstream or the configuration memory. Therefore, it is necessary to reduce the reconfiguration time. To this end, we conducted an experimental study for the viable implementation of reconfigurable point-to-point (ρ-P2P) interconnects using the partial reconfiguration technique.

In Chapter 5(**Hardwiring Crossbar Interconnect Fabric**), we proposed to

hardwire crossbars as an interconnect fabric to increase performance (in terms of latency and throughput) and reduce area cost. We described the general advantages of the hardwired interconnect fabric in terms of its functional performance, area, granularity, wire delay, wire variation, partial reconfiguration time, and resource utilization. Considering a soft interconnect as a reference, an analysis was conducted to evaluate hardwired crossbar fabric. We utilized the MJPEG application as a case study using the Jackson's queuing model to analyze the performance of the interconnect. Our analysis and implementation results indicated that the hardwired crossbar provides up to $4.2\times$ better throughput than the soft crossbar at an acceptable cost.

In Chapter 6(**Soft and Hardwired Network-on-Chip**), we presented an application-specific soft NoC for the overlay layer and hardwired NoCs for the fabric layer, as described by following:

Application-specific soft NoC: We proposed a soft customized circuit-switched NoC (CCSN) to reduce the area cost. We demonstrated that a customized NoC can be derived by optimizing away un-utilized intra-router buffers, the intra-router links, inter-router links, and associated control logic. We analyzed the utilized network resources from the topology embedding. Our implementation results showed that on average 71% of the area cost is reduced by applying our customization technique.

Hardwired NoC fabric: We presented a hardwired NoC (HWNoC) as an interconnect fabric to increase performance in terms of latency and throughput. We showed that physical (on-line) wiring over the configuration plane is slower than logical routing over the hardwired NoC. Hardwiring inter-IP communication is a viable and promising solution to the scalability problem. HWNoC not only performs better than soft network but also it provides efficient utilization of configuration bitstream and on-chip logic resources. Moreover, HWNoC inherently is suitable for the partial reconfiguration. As our analysis and implementation results indicate, the hardwired network is significantly better in speed, throughput, resource utilization, and an area cost. Finally, we discussed the loss of flexibility of the HWNoC.

7.2 Future work

In this thesis, we focused on how the soft overlay interconnects are *statically* mapped onto the (hardwired or soft) functional plane for a single application in

the FPGA fabric. As a possible future work, we could investigate how to devise the adaptive interconnects for *dynamic* applications from the system perspective. Subsequently, future research directions can be elaborated as follows:

Dynamically reprogrammable topologies: In our soft interconnects, the desired topologies are configured at design time. We presented the dynamically reconfigurable topologies in Chapter 4, where we focused on the physical (zero-hop) wiring by utilizing the reconfigurability of FPGAs. As an alternative, the logical on-line connection (or reprogramming) over the multi-hop hardwired fabric could be investigated. In addition, a network architecture that supports dynamic topologies could be integrated in the entire reconfigurable platform. As an example, the FLUX network [79] could be designed and integrated into the MOLEN programming paradigm [81].

Unified hardwired NoC: In this thesis, we examined the effect of the hardwired NoC mainly in the functional plane. However, the interconnect fabric in the configuration plane of the modern FPGAs comprise still dedicated point-to-point interconnects and is orthogonal to the functional plane. In this case, the global wires are spread over the entire chip and accordingly suffer from the low configuration performance and scalability problem. As an example, the configuration plane of the Virtex-5 device operates at 100 MHz [95], which is lower than the typical clock frequency in the functional layer. To solve these problems, the interconnects in the configuration plane and the functional plane can be *unified* as proposed in [50]. In the context of designing the unified hardwired NoCs, the proposed future work can be described as follows. First, when the unified NoC is employed, the FPGA architecture and the design method are required to be developed. Second, it is required to evaluate the unified NoC from the practical application perspectives. To achieve this, a system-level simulation framework can be developed. Third, the power consumption can be evaluated while we focused on performance and cost.

Task and resource management for hardwired NoC: Little has been reported regarding the task management and resource (re)allocation in the reconfigurable platform for dynamic applications. A major problem for the task (re)placement, migration, and scheduling is the reconfiguration cost for the interconnects. By hardwiring an interconnect fabric, the inter-task communication cost can be reduced because the reconfiguration is not required. Subsequently, it is required to develop the task/resource management methodologies for the HWNoC-based platform. The task/resource manager itself can be implemented in soft or hard, which can be investigated further.

Exact performance modeling: In this thesis, we derived the steady-state average delay model and applied Jackson's model. Subsequently, the formulations are approximated and have the following limitations. First, the formulations do not explicitly capture the pipelining operation of traffics. As an example, the derived performance of NoC is conservative and pessimistic. Second, to simplify the formulation, we assumed the FIFO sizes are sufficiently large. Accordingly, the waiting time in the queue due to the backpressure, the flow control, and the port sharing (for multiple FIFOs) is ignored. Third, to focus on the network analysis, we assumed that the token size before and after the node is the same. The token size can be different depending on the computation in the IP. Though our simplifications are mainly based on the implementation and can be sufficient to derive the relative network performance, it is desirable to derive an exact model to capture the dynamic behavior in the SoC.

Bibliography

[1] AMBA 3 Specification, ARM Ltd., http://www.arm.com/.

[2] Cisco Systems, Inc., http://www.cisco.com.

[3] FPGA FAQ, http://www.fpga-faq.org.

[4] Multi-layer AHB Protocol specification. ARM, Inc., http://www.arm.com/.

[5] Xapp 290: Two flows for partial reconfiguration: Module based or difference based, application note, http://www.xilinx.com, Sep.

[6] Alpha Data Parallel Systems, Ltd., http://www.alpha-data.com/adm-xpl.html, 2002.

[7] Virtex-ii pro handbook, http://www.xilinx.com, 2002.

[8] A. Andriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. Zeferino. SPIN: a Scalable, Packet Switched, On-Chip Micro-network. In *Proceedings of International Conference on Design, Automation and Test in Europe (DATE'03)*, pages 70–73, Mar. 2003.

[9] A. DeHon. Reconfigurable architectures for general-purpose computing, PhD dissertation, Massachusetts Institute of Technology, Sep. 1996.

[10] A. DeHon. Unifying mesh- and tree-based programmable interconnect. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(10):1051–1065, Oct. 2004.

[11] A. Hansson and K. Goossens. Trade-offs in the configuration of a network on chip for multiple use-cases. In *Proceedings of the 1st International Symposium on Networks-on-Chips (NOCS'07)*, pages 233–242, May 2007.

[12] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor. Concepts and Implementation of Spatial Division Multiplexing for Guaranteed Throughput

in Networks-on-Chip. *IEEE Transactions on Computers*, 57(9):1182–1195, Sep. 2008.

[13] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, and D. Verkest. Spatial Division Multiplexing: a Novel Approach for Guaranteed Throughput on NoCs. In *Proceedings of the International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS'05)*, pages 81–85, Sep. 2005.

[14] A. Mello, L. Tedesco, N. Calazans, and F. Moraes. Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC. In *Proceedings of 18th Symposium on Integrated Circuits and Systems Design (SBCCI'05)*, pages 178–183, Sep. 2005.

[15] A. Rădulescu, J. Dielissen, S. G. Pestana, O. P. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 24(1):4–17, Jan. 2005.

[16] Arteris Inc. A Comparison of Network-on-Chip and Busses, White Paper, http://www.arteris.com, 2005.

[17] B. Kienhuis, E. Rijpkema, and E. F. Deprettere. Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing Architectures. In *Proceedings of 8th International Workshop on Hardware/Software Codesign (CODES'2000)*, pages 13–17, May 2000.

[18] B. S. Landman and R. L. Russo. On a Pin Versus Block Relationship For Partitions of Logic Graphs. *IEEE Transactions on Computers*, C-20(12):1469–1479, Dec. 1971.

[19] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri. Lipar: A Lightweight Parallel Router for Fpga Based Networks on Chip. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI'05)*, pages 452–457, Apr. 2005.

[20] C. Bartels, J. Huisken, K. G. W. Goossens, P. Groeneveld, and J. V. Meerbergen. Comparison of an Æthereal Network on Chip and a Traditional Interconnect for a Multi-Processor DVB-T System on Chip. In *Proceedings of International Conference on Very Large Scale Integration (VLSI-SoC'06)*, pages 80–85, Oct. 2006.

[21] C. Bobda, A. Majer, A. Ahmadinia, T. Haller, A. Linarth, and J. Teich. The Erlangen Slot Machine: Increasing Flexibility in FPGA-based Reconfigurable Platforms. In *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT'05)*, pages 116–125, Dec. 2005.

[22] C. Hilton and B. Nelson. PNoC: A flexible circuit-switched NoC for FPGA-based systems. *IEE Proc.-Comput. Digit. Tech.*, 153(3):181–188, May 2006.

[23] C. Webb. 45nm Design for Manufacturing. *Intel Technology Journal, http://www.intel.com*, 12(2):212–230, Jun. 2008.

[24] C. Zeferino and A. Susin. SoCIN: A Parameteric and Scalable Network-on-Chip. In *Proceedings of 16th Symposium on Integrated Circuits and Systems Design (SBCCI'03)*, pages 169–174, Sep. 2003.

[25] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. D. Micheli. NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):113–129, Feb. 2005.

[26] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 2nd edition, 1991.

[27] D. Sigüenza-Tortosa and J. Nurmi. Proteo: A New Approach to Network-on-Chip. In *Proceedings of IASTED International Conference on Communication Systems and Networks(CSN'02)*, pages 355–357, Sep. 2002.

[28] E. A. de Kock, G. Essink, W. J. M Smits, P. van der Wolf, J.-Y. Brunel, W. M. Kruijtzer, P. Lieverse, and K. A. Vissers. YAPI: Application Modeling for Signal Processing Systems. In *Proceedings of the 37th Design Automation Conference (DAC'00)*, pages 402–405, Jun. 2000.

[29] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS Architecture and Design Process for Network on Chip. *Journal of Systems Architecture, Special Issue on Networks on Chip*, 50(2-3):105–128, Dec. 2004.

[30] E. Salminen, A. Kulmala, and T. D. Hämäläinen. On Network-on-Chip Comparison. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD'07)*, pages 503 – 510, Aug. 2007.

[31] F. Karim, A. Nguyen, and S. Dey. An Interconnect Architecture for Network Systems on Chips. *IEEE Micro*, 22(5):36–45, Sep. 2002.

[32] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. HERMES: an Infrastructure for Low Area Overhead Packet-switching Netwoks on Chip. *Integration, the VLSI Journal*, 38(1):69–93, Oct. 2004.

[33] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays·Trees·Hypercubes*. Morgan Kaufmann Publishers, Inc., 1992.

[34] G. Brebner and D. Levi. Networking on chip with platform FPGAs. In *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT'03)*, pages 13–20, Dec. 2003.

[35] G. K. Rauwerda, P. M. Heysters, and G. J. M. Smit. Mapping Wireless Communication Algorithms onto a Reconfigurable Architecture. *The Journal of Supercomputing*, 30(3):263–282, Dec. 2004.

[36] G. Kahn. The semantics of a simple language for parallel programming. In *Proceedings of the IFIP Congress, North-Holland publishing Co.*, pages 471–475, 1974.

[37] H. G. Lee, U.Y. Ogras, R. Marculescu, and N. Chang. Design space exploration and prototyping for on-chip multimedia applications. In *Proceedings of the 43th Design Automation Conference (DAC'06)*, pages 137–142, Jul. 2006.

[38] H. Kasahara, T. Tobita, T. Matsuzawa, and S. Sakaida. Standard Task Graph Set., http://www.kasahara.elec.waseda.ac.jp/schedule/.

[39] H. N. Nikolov, T. P. Stefanov, Ed F. Deprettere. Efficient Automated Synthesis, Programming, and Implementation of Multi-processor Platforms on FPGA Chips. In *Proceedings of 16th International Conference on Field Programmable Logic and Applications (FPL'06)*, pages 323–328, Aug. 2006.

[40] H. Nikolov, T. Stefanov, and Ed Deprettere. Multi-processor System Design with ESPAM. In *Proceedings of the 4th IEEE/ACM/IFIP International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS'06)*, pages 211–216, Oct. 2006.

[41] H. Nikolov, T. Stefanov, and Ed Deprettere. Systematic and Automated Multi-processor System Design, Programming, and Implementatio. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):542–555, Mar. 2008.

[42] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, Feb. 2007.

[43] J. Hu and R. Marculescu. Energy-Aware Mapping for Tile-based NoC Architectures Under Performance Constraints. In *Proceedings of the 8th Asia and South Pacific Design Automation Conference (ASP-DAC'03)*, pages 233–239, Jan. 2003.

[44] J. Hu, U. Y. Ogras, and R. Marculescu. System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(12):2919–2933, Dec. 2006.

[45] J. Jackson. Networks of waiting lines. *Operations Research*, 5(4):518–521, Aug. 1957.

[46] J. Liang, S. Swaminathan, and R. Tessier. aSOC: A Scalable, Single-Chip Communications Architecture. In *IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT'00)*, pages 37–46, Oct. 2000.

[47] J. Y. Hur, T. Stefanov, S. Wong, and S. Vassiliadis. Customizing Reconfigurable On-Chip Crossbar Scheduler. In *Proceedings of IEEE 18th International Conference on Application-specific Systems, Architectures and Processors (ASAP07)*, pages 210–215, Jul. 2007.

[48] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: concepts, architectures, and implementations. *IEEE Design & Test of Computers*, 22(5):414–421, Sep. 2005.

[49] K. Goossens, J. Dielissen, O. P. Gangwal, S. G. Pestana, A. Radulescu, and E. Rijpkema. A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification. In *Proceedings of International Conference on Design, Automation and Test in Europe (DATE'05)*, pages 1182–1187, Mar. 2005.

[50] K. Goossens, M. Bennebroek, J. Y. Hur, and M. A. Wahlah. Hardwired Networks on Chip in FPGAs to Unify Functional and Configuration Interconnects. In *Proceedings of the IEEE International Symposium on Networks-on-Chip (NOCS'08)*, pages 45–54, Apr. 2008.

[51] K. Goossens, O. P. Gangwal, J. Röver, and A. P. Niranjan. *Interconnect and Memory Organization in SOCs for Advanced Set-Top Boxes and TV — Evolution, Analysis, and Trends.* In Interconnect-Centric Design for Advanced SoC and NoC, Chapter 15, pp. 399–423, 2004.

[52] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey. Design of High-Performance System-On-Chips Using Communication Architecture Tuners. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(5):620–636, May 2004.

[53] K. Sekar, K. Lahiri, A. Raghunathan, and S. Dey. FLEXBUS: A High-Performance System-on-Chip Communication Architecture with a Dynamically Configurable Topology. In *Proceedings of 42th Design Automation Conference (DAC05)*, pages 571–574, Jun. 2005.

[54] K. Srinivasan and K. S. Chatha. A Low Complexity Heuristic for Design of Custom Network-on-chip Architectures. In *Proceedings of International Conference on Design, Automation and Test in Europe (DATE'06)*, pages 130–135, Mar. 2005.

[55] L. Braun, M. Hbner, J. Becker, T. Perschke, V. Schatz, and S. Bach. Circuit Switched Run-Time Adaptive Network-on-Chip for Image Processing Applications. In *Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL'07)*, pages 688–691, Aug. 2007.

[56] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini. Xpipes: Latency Insensitive Parameterized Network-on-Chip Architecture for Multi-Processor SoCs. In *Proceedings of International Conference on Computer Design (ICCD'03)*, pages 536–539, Sep. 2003.

[57] M. Forsell. A Scalable High-Performance Computing Solution for Networks on Chips. *IEEE Micro*, 22(5):46–55, Sep. 2002.

[58] M. Huebner, M. Ullmann, L. Braun, A. Klausmann, and J. Becker. Scalable Application-Dependent Network on Chip Adaptivity for Dynamical Reconfigurable Real-Time Systems. In *Proceedings of 14th International Conference on Field Programmable Logic and Applications (FPL'04)*, pages 1037–1041, Aug. 2004.

[59] M. Huebner, T. Becker, and J. Becker. Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration. In *Proceedings of 17th Symposium on Integrated Circuits and Systems Design (SBCCI'04)*, pages 28–32, Sep. 2004.

[60] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing On-Chip Communication in a MPSoC Environment. In *Proceedings of International Conference on Design, Automation and Test in Europe (DATE'04)*, pages 752–757, Feb. 2004.

[61] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, R. Rabaey, and A. Sangiovanni-Vincentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *Proceedings of the 43th Design Automation Conference (DAC'01)*, pages 667–672, Jun. 2001.

[62] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon. Packet Switched vs Time Multiplexed FPGA Overlay Networks. In *Proceedings of Field-Programmable Custom Computing Machines (FCCM'06)*, pages 205–216, Apr. 2006.

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

[64] N. Steiner. A Standalone Wire Database for Routing and Tracing in Xilinx Virtex, Virtex-E, and Virtex-II FPGAs, Master thesis, Virginia Polytechnic Institute and State University, Aug. 2002.

[65] N. Steiner and P. Athanas. An Alternate Wire Database for Xilinx FPGAs. In *Proceedings of Field-Programmable Custom Computing Machines (FCCM'04)*, pages 336–337, Apr. 2004.

[66] P. Gupta and N. McKeown. Designing and Implementing a Fast Crossbar Scheduler. *IEEE Micro*, 19(1):203–289, Jan.-Feb. 1999.

[67] P. Meloni, M. Camplani, L. Raffo, S. M. Carta, S. Murali, and G. De Micheli. Routing Aware Switch Hardware Customization for Networks on Chips. In *Proceedings of 1st International Conference on Nano-Networks (Nano-Net'06)*, pages 1–5, Sep. 2006.

[68] P. Pande, C. Grecu, A. Ivanov, and R. Saleh. Design of a Switch for Network on Chip Applications. In *Proceedings of International Symposium on Circuits and Systems (ISCAS'03)*, pages 217–220, May 2003.

[69] P. T. Wolkotte, G. J. M. Smit, and L. T. Smit. Partitioning of a DRM Receiver. In *Proceedings of the International OFDM-workshop*, pages 299–304, Sep. 2004.

[70] R. Gindin, I. Cidon, and I. Keidar. NoC-Based FPGA: Architecture and Routing. In *Proceedings of the 1st International Symposium on Networks-on-Chips (NOCS'07)*, pages 253–264, May 2007.

[71] R. Hecht, S. Kubisch, A. Herrholtz, and D. Timmermann. Dynamic Reconfiguration with hardwired Networks-on-Chip on future FPGAs. In *Proceedings*

of 15th International Conference on Field Programmable Logic and Applications (FPL'05), pages 527–530, Aug. 2005.

[72] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, Apr. 2001.

[73] R. O. Baldwin, N. J. Davis IV, S. F. Midkiff, and J. E. Kobza. Queueing network analysis: concepts, terminology, and methods. *The Journal of Systems and Software*, 66(2):99–117, May 2003.

[74] S. J. E. Wilton, C. H. Ho, P. H. W. Leong, W. Luk, and B. Quinton. A Synthesizable Datapath Oriented Embedded FPGA Fabric. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA'07)*, pages 33–41, Feb. 2007.

[75] S. Murali and G. D. Micheli. Bandwidth-Constrained Mapping of Cores onto NoC Architectures. In *Proceedings of International Conference on Design, Automation and Test in Europe (DATE'04)*, pages 896–901, Feb. 2004.

[76] S. Murali and G. D. Micheli. An Application-Specific Design Methodology for STbus Crossbar Generation. In *Proceedings of International Conference on Design, Automation and Test in Europe (DATE'05)*, pages 1176–1181, Mar. 2005.

[77] S. Murali, L. Benini, and G. D. Micheli. An Application-Specific Design Methodology for On-Chip Crossbar Generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(7):1283–1296, Jul. 2007.

[78] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Constraint-Driven Bus Matrix Synthesis for MPSoC. In *Proceedings of 11th Asia and South Pacific Design Automation Conference (ASP-DAC'06)*, pages 30–35, Jan. 2006.

[79] S. Vassiliadis and I. Sourdis. FLUX Networks: Interconnects on Demand. In *Proceedings of International Conference on Computer Systems Architectures Modelling and Simulation (IC-SAMOS'06)*, pages 160–167, Jul. 2006.

[80] S. Vassiliadis and I. Sourdis. FLUX Interconnection Networks on Demand. *Journal of Systems Architecture*, 53(10):777–793, Oct. 2007.

[81] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K.L.M. Bertels, G.K.Kuzmanov, E. Moscu Panainte. The Molen Polymorphic Processor. *IEEE Transactions on Computers*, 53(11):1363– 1375, Nov. 2004.

[82] S. Wee, J. Casper, N. Njoroge, Y. Tesylar, D. G. C. Kozyrakis, and K. Olukotun. A Practical FPGAbased Framework for Novel CMP Research. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA'07)*, pages 116–125, Feb. 2007.

[83] T. A. Bartic, J.-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins. Topology adaptive network-on-chip design and implementation. *IEE Proc. -Comput. Digit. Tech.*, 152(4):467–472, Jul. 2005.

[84] T. Bjerregaard and J. Sparsø. A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip. In *Proceedings of International Conference on Design, Automation and Test in Europe (DATE'05),*, pages 1226–1231, Mar. 2005.

[85] T. Bjerregaard and S. Mahadevan. A Survey of Research and Practices of Network-on-Chip. *ACM Computing Surveys*, 38(1):1–51, Mar. 2006.

[86] T. Marescaux, V. Nollet, J.-Y. Mignolet, A. B. W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Run-time support for heterogeneous multitasking on reconfigurable socs. *Integration, the VLSI Journal*, 38(1):107–130, 2004.

[87] T. Pionteck, R. Koch, and C. Albrecht. Applying Partial Reconfiguration to Networks-on-Chips. In *Proceedings of 16th International Conference on Field Programmable Logic and Applications (FPL06)*, pages 155–160, Aug. 2006.

[88] V. Ramamurthi, J. McCollum, C. Ostler, and K. S. Chatha. System-level Methodology for Programming CMP based Multi-threaded Network Processor Architectures. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI'05)*, pages 110–116, May 2005.

[89] W. Donath. Placement and Average Interconnection Lengths of Computer Logic. *IEEE Transactions on Circuits and Systems*, 26(4):272–277, Apr. 1979.

[90] W. J. Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, Mar. 1992.

[91] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 684–689, Jun. 2001.

[92] Wim Heirman. Reconfigurable Optical Interconnection Networks for Shared-Memory Multiprocessor Architectures, PhD dissertation, Ghent University, Jul. 2008.

[93] X. Jiang, W. Wolf, J. Henkel, and S. Chakradhar. H.264 HDTV Decoder Using Application-Specific Networks-On-Chip. In *Proceedings of International Conference on Multimedia and Expo (ICME'05)*, pages 1508–1511, Jul. 2005.

[94] Xilinx Inc. Local Memory Bus (LMB), DS445, http://www.xilinx.com, Apr. 2005.

[95] Xilinx Inc., http://www.xilinx.com.

[96] Z. J. Yang, A. Kumar, and Y. Ha. An Area-efficient Dynamically Reconfigurable Spatial Division Multiplexing Network-on-Chip with Static Throughput Guarantee. In *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT'10)*, Dec. 2010.

List of Publications

Journals

1. J. Y. Hur, S. Wong, and T. Stefanov, “Design Trade-offs in Customized On-Chip Crossbar Schedulers”, Journal of VLSI Signal Processing Systems, vol. 58, no. 1, pp. 69-85, Jan. 2010.

2. J. Y. Hur, S. Wong, and S. Vassiliadis, “Partially Reconfigurable Point-to-Point FPGA Interconnects”, Int’l Journal of Electronics, vol. 95, no. 7, pp. 725-742, July 2008.

International Conference / Workshop proceedings

1. T. Marconi, J.Y. Hur, K.L.M. Bertels, and G. N. Gaydadjiev, “A Novel Configuration Circuit Architecture to Speedup Reconfiguration and Relocation for Partially Reconfigurable Devices”, Proceedings of IEEE Symposium on Application Specific Processors (SASP), pp. 87-92, Anaheim, Jun. 2010.
2. A. Shahbahrami, J.Y. Hur, B.H.H. Juurlink, and S. Wong, “FPGA Implementation of Parallel Histogram Computation”, HiPEAC Workshop on Reconfigurable Computing, pp. 63-72, Göteborg, Jan. 2008.
3. K.G.W. Goossens, M. Bennebroek, J.Y. Hur, and M.A. Wahlah, “Hardwired Networks on Chip in FPGAs to unify Data and Configuration Interconnects”, IEEE Int’l Symposium on Networks on Chip (NOCS’08), pp. 45-54, Newcastle, April 2008.
4. J. Y. Hur, K. Goossens, and L. Mhamdi, “Performance Analysis of Soft and Hard Single-Hop and Multi-Hop Circuit Switched Interconnects for FPGAs”, IFIP/IEEE Int’l Conference on Very Large Scale Integration (VLSI-SoC’08), pp. 224-229, Rhodes, October 2008.

5. J. Y. Hur, T. Stefanov, S. Wong, and S. Vassiliadis, “Customizing Reconfigurable On-Chip Crossbar Scheduler”, IEEE Int’l Conference on Application-specific Systems, Architectures and Processors (ASAP07), pp. 210-215, Montreal, Canada, July 2007.

6. J. Y. Hur, T. Stefanov, S. Wong, and S. Vassiliadis, “Systematic Customization of On-Chip Crossbar Interconnects”, Int’l Workshop on Applied Reconfigurable Computing (ARC’07), Lecture Note in Computer Science, pp. 62-71, Mangaratiba, Brazil, March 2007.

7. J. Y. Hur, S. Wong, and S. Vassiliadis, “Partially Reconfigurable Point-to-Point Interconnects in Virtex-II Pro FPGAs”, Int’l Workshop on Applied Reconfigurable Computing (ARC’07), Lecture Note in Computer Science, pp. 49-60, Mangaratiba, Brazil, March 2007.

National Conferencees

1. S. Wong, S. Vassiliadis, and J.Y. Hur, “Parallel Merge Sort on a Binary Tree On-Chip Network”, Workshop on Circuits, Systems and Signal Processing (ProRISC’05), pp. 365-368, Veldhoven, The Netherlands, Nov. 2005.

Samenvatting

Dit proefschrift beschrijft ons onderzoek naar de vraag hoe men effici?nt on-chip draden kan gebruiken en de netwerkprestaties in herconfigureerbare hardware kan verbeteren. Field-Programmable Gate Arrays (FPGAs), als essentieel onderdeel in moderne herconfigureerbare multiprocessor platformen, bevatten miljoenen draden, welke herconfigureerbaarheid op vraag mogelijk maken. Moderne FPGAs worden steeds krachtiger naarmate hardware modules als ingebedde geheugens, processoren, en DSPs worden toegevoegd. Echter de prestaties en de kosten van inter-processor communicatie blijft de voornaamste uitdaging. In de context van interconnectie-netwerken in FPGA technologie, gaan we deze uitdaging aan op twee punten.

Ten eerste hebben conventionele algemene netwerken hoge oppervlaktekosten wanneer zij toegewezen worden aan een herconfigureerbaar medium. Om de area kosten te verminderen stellen wij een topologie specialisatie techniek voor specifieke applicaties voor. Concreet presenteren wij een applicatie-specifieke crossbar switch, crossbar schedulers, punt-punt interconnecties, en circuit-geschakelde netwerken op een chip (NoCs) die zich bevinden in een herconfigureerbaar medium. Door de netwerkprestaties per kosten als metriek te beschouwen voeren we een prestatieanalyse uit op een implementatie van onze techniek om een vergelijking te maken met algemene netwerken. Dientengevolge, door slechts de benodigde netwerkbronnen te instanti?ren, verschaffen onze customized netwerken betere prestaties per kostenpost met een grootteorde.

Ten tweede gaat een deel van het voornaamste voordeel van FPGAs verloren, doordat een verhoging van de herconfigureerbaarheid gepaard gaat met een afname van de prestaties en een toename van de kosten. Dit komt voornamelijk doordat de interconnecties herconfigureerbaar zijn op het niveau van bits. Om de prestaties te verhogen en de kosten te drukken, stellen wij voor de herconfigureerbare wires op bit-niveau te vervangen met vaste circuit-geschakelde netwerken voor inter-processor communicatie. Concreet presenteren wij vaste crossbars en een circuit-

geschakeld NoC interconnectie medium. We beschrijven de voordelen van de vaste netwerken ondersteund door een kwantitatieve prestatie-analyse, netwerksimulatie, en een implementatie. Vaste netwerken hebben betere prestaties per kostenpost, met twee grootteordes, dan de netwerken die geprogrammeerd worden in herconfigureerbare media.

Propositions

1. To make the most of scalable logical network on chip, the underlying fabric should be a physical network on chip.
2. Soft interconnects in FPGAs should be application specific as well as technology aware.
3. By providing identical physical topologies for arbitrary logical topologies, network performance increases, area cost decreases, and power consumption decreases.
4. Shared custom parallel schedulers can alleviate a scalability problem by sharing wires and interconnect resources.
5. Compared to soft interconnects, hardwired interconnects in FPGAs provide better configuration performance and cost.
6. Compared to soft interconnects, hardwired networks on chip increase flexibility for computational resources.
7. Hardwired crossbar fabrics perform better than soft buses at an acceptable cost.
8. As systems and traffic patterns become more complex, their analysis can often be simpler.
9. Although we prefer simplicity, complexity is often required for the acceptance of a paper.

Stellingen

1. Om optimaal gebruik te maken van schaalbare logische netwerken op de chip, moet het onderliggende communicatie medium een fysiek netwerk op de chip worden.

2. Geprogrammeerde interconnects in FPGA's moeten zowel toepassingsspecifiek als technologiespecifiek worden.

3. Door het verstrekken van identieke fysieke topologie?n voor willekeurige logische topologie?n, nemen de prestaties van het netwerk toe, neemt het minder oppervlak in beslag, en vermindert het energieverbruik.

4. Gedeelde parallelle schedulers kunnen het schaalbaarheidsprobleem verlichten door het delen van draden en interconnectmiddelen.

5. Vergeleken met geprogrammeerde interconnects, bieden vaste interconnects in FPGA's een betere configuratieprestaties en lagere kosten.

6. Vergeleken met geprogrammeerde interconnects, verhogen vaste netwerken op de chip de flexibiliteit voor de rekenmiddelen.

7. Hardwired crossbars presteren beter dan geprogrammeerde bussen tegen aanvaardbare kosten.

8. Als systemen en verkeerspatronen complexer worden, kan hun analyse vaak eenvoudiger.

9. Hoewel wij de voorkeur geven aan eenvoud, is voor de aanvaarding van een artikel het vaak nodig het complex te maken.

Curriculum Vitae

Jae Young Hur was born in Cheju island, South Korea on the 3^{rd} of June 1971. He received the B.S. degree in electronics engineering from Cheju National University, Cheju, South Korea in February 1995. He received the M.S. degree in electronics engineering from Sogang University, Seoul, South Korea in August 1998 and the M.S. degree in Communications Engineering from Munich University of Technology, Munich, Germany in October 2002. From January 1999 to August 2000, he was an engineer in System LSI division, Samsung Electronics, Ltd., South Korea.

From August 2001 to October 2001, he did an internship at CPD AA (Network Processor department) in Infineon Technologies, Munich, Germany. In April 2003, he was accepted as a PhD student by Professor Stamatis Vassiliadis in the computer engineering (CE) laboratory, Delft University of Technology, The Netherlands. Since November 2003, he has been with the CE group, where he was a research assistant. He was involved in Artemisia (the Architecture, Programming, and Exploration of Networks-on-Chip based Embedded System Platforms) project and was supported by Dutch Technology Foundation. His research focused on the interconnects in FPGAs and this thesis describes his study.

Since November 2008, he has been working as a senior engineer at SoC Platform department in System LSI division, Samsung Electronics, Ltd., South Korea. His research interests include embedded multi-processor system, network-on-chip, VLSI design, and reconfigurable computing.