THESE DE DOCTORAT DE

L'UNIVERSITE PIERRE ET MARIE CURIE

Spécialité

Informatique-Microélectronique

(EDITE)

Présentée par

Abbas SHEIBANYRAD

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Implémentation Asynchrone d'un Réseau-sur-Puce Distribué

soutenue le 19 Mars 2008

Devant le jury composé de :

Prof. David ATIENZA ALONSO	(Rapporteur – UCM/DACYA, Espagne)
Prof. Alain GREINER	(Directeur de la thèse – LIP6/SoC, France)
Dr. Riccardo LOCATELLI	(Examinateur – STMicroelectronics, France)
Prof. Frédéric PETROT	(Président du jury – TIMA/SLS, France)
Prof. Christian PIGUET	(Rapporteur – EPFL, Suisse)
Dr. Pascal VIVET	(Examinateur – CEA-LETI, France)

Asynchronous Implementation of a Distributed Network-on-Chip

a thesis

to obtain the degree of Doctor of Philosophy (PhD)

submitted to the University of Pierre et Marie Curie (UPMC)

in the Computer Science Laboratory of Paris 6 (LIP6)

supervised by Professor Alain GREINER

Abbas SHEIBANYRAD

Paris / Winter of 2008

Résumé

En raison de limitations physiques, il est désormais extrêmement difficile, si pas impossible, de faire distribuer un signal d'horloge globale synchrone sur une vaste zone de la puce. Comme une solution, les Réseaux-sur-Puce (NoCs) qui utilisent les techniques de Globalement Asynchrone Localement Synchrone (GALS), divisent la puce en plusieurs zones synchrones indépendantes. Chaque zone est cadencée par un signal d'horloge différente, et de cette façon le problème est réduit à un certain nombre de sous-problèmes plus petits. Le réseau pourrait être l'infrastructure de communication globale asynchrone du système. Mais, comment le réseau lui-même doit être cadencé et comment nous pouvons traiter le problème de la synchronisation des horloges en frontières. Cette thèse de doctorat, dans le fond, essaie de répondre à ces deux questions.

Un réseau avec une conception entièrement asynchrone, qui n'implique pas la question de la synchronisation, est une approche naturelle pour construire des architectures GALS. Un NoC asynchrone limite la demande de synchronisation seulement aux interfaces du réseau, où les données synchrones doivent entrer dans le réseau asynchrone et les données asynchrones dans les sous-systèmes synchrones.

ASPIN (Asynchronous Scalable Packet-switching Integrated Network), présenté dans ce manuscrit, est un réseau asynchrone qui utilise deux FIFOs spéciales pour connecter les IPs synchrones au réseau asynchrone. Au début, l'implémentation détaillée de l'architecture de routeur ASPIN et de deux nouvelles conceptions pour le FIFO synchrone-asynchrone (SA_FIFO) et le FIFO asynchrone-synchrone (AS_FIFO) comme les interfaces du réseau sont élaborées. Tous les dessins ont été physiquement implémentés, et les caractéristiques électriques ont été évaluées par la simulation SPICE poste-layout.

Malgré que les NoCs sont beaucoup plus évolutif que les interconnexions traditionnelles, lorsque le nombre de composants générant du trafic augmente, le seuil de saturation du réseau diminue, et parfois il devient le goulot d'étranglement du système. Nous évaluons le seuil de saturation des réseaux ASPIN et DSPIN. DSPIN (Scalable Distributed Packet-switching Integrated Network) est un réseau multi-synchrone bien adapté au paradigme GALS. En réalité ASPIN est l'implémentation asynchrone de DSPIN. Dans l'évaluation de seuil de saturation

l'influence des deux paramètres est prise en compte: la capacité de stockage des flits et le débit du réseau.

Dans les grands systèmes, où il y a de nombreux éléments à interconnecter, le seuil de saturation du réseau a une faible valeur et déclare un problème. Nous proposons une nouvelle méthode pour améliorer le seuil de saturation dans les réseaux asynchrones rapides: en utilisant un algorithme Quasi-Store-and-Forward (QSF) au lieu de routage bout-en-bout wormhole. Dans cette approche, tous les flits d'un paquet s'accumulent dans le format asynchrone avant d'entrer dans le réseau.

Comme DSPIN et ASPIN utilisent la même architecture générale et fournissent les mêmes services, une comparaison systématique entre leurs paramètres de performances physiques, présentée dans cette thèse, peut aider à répondre à cette question que quelle type de l'architecture pourrait être plus satisfaisant à implémenter, synchrone ou asynchrone? Les caractéristiques physiques sont la surface en silicium, la latence de paquet, le débit de communication, et la consommation d'énergie. Comme un facteur prédominant, dans les évaluations les effets des longs fils ont été pris en compte.

Abstract

Because of physical limitations, henceforth it is extremely hard, if not impossible, to distribute a global synchronous clock signal over a wide chip area. As a solution, Networks-on-Chip (NoCs) using Globally Asynchronous Locally Synchronous (GALS) techniques divide the chip area into several independent synchronous clusters. Each cluster is clocked by a different clock signal and thereby the problem is reduced to a number of smaller sub-problems. The network could be the asynchronous global communication infrastructure of the system. But, how the network itself must be clocked and how we can deal with the problem of synchronization on clock boundaries. This PhD thesis, basically, tries to answer these two questions.

A network with a fully asynchronous design, which does not involve the issue of synchronization, is a natural approach to construct GALS architectures. An asynchronous NoC limits the synchronization failure only at the network interfaces, where the synchronous data has to enter into the asynchronous network and the asynchronous data into the synchronous subsystems.

ASPIN (Asynchronous Scalable Packet-switching Integrated Network), introduced in this manuscript, is an asynchronous NoC using two special FIFOs to connect synchronous IP cores to the asynchronous network. At first the router implementation of the ASPIN architecture is detailed and two new concepts for Synchronous-to-Asynchronous FIFO (SA_FIFO) and Asynchronous-to-Synchronous FIFO (AS_FIFO) as network interfaces are elaborated. All designs have been physically implemented, and the electrical characteristics have been evaluated by post layout SPICE simulation.

Even though NoCs are much more scalable than traditional on-chip interconnects, when the number of components generating traffic increases, the network saturation threshold decreases and sometimes becomes the system bottleneck. We evaluate the saturation threshold of the ASPIN and DSPIN network architectures. DSPIN (Distributed Scalable Packet-switching Integrated Network) is a multi-synchronous NoC well-suited to the GALS paradigm. Actually ASPIN is the asynchronous implementation of DSPIN. In the saturation threshold evaluation the influence of two parameters are considered: the flit storage capacity of the network and the network throughput.

In large systems, where there are many components to interconnect, the network saturation threshold has a low value and declares a problem. We propose a new method to improve the saturation threshold in fast asynchronous networks: using a Quasi-Store-and-Forward (QSF) algorithm instead of end-to-end wormhole routing. In this approach all flits of a given packet accumulate in the asynchronous form before entering the network.

As DSPIN and ASPIN use the same general architectures and provide the same services, a systematic comparison between their physical performances parameters, presented in this thesis, may help to answer this question that which architecture type could be more adequate to implement, synchronous or asynchronous? The related physical characteristics are silicon area, packet latency, communication throughput, and power consumption. As a predominant factor, in the evaluations the effects of long wires have been taken into account.

Reconnaissance

Depuis toujours j’ai pensé que le texte de reconnaissance et de remerciement qui se trouve au début du manuscrit des thèses n’est qu’un cliché, et je me disais qu’il faudrait éviter ces remerciements superficiels. Maintenant, c’est mon tour, et, comment je peux exprimer mes gratitudes autrement ! Comment je peux raconter mon parcours !

Si c'est mon dernier diplôme universitaire, comment je peux ne pas dire ici qui m'a formé pendant mes années d’études !

Comment je peux ne pas présenter ici mes remerciements à Mme. Gorgikia ! Celle qui m'a appris à lire et à écrire.

Comment je peux ne pas présenter ici mes remerciements à Mme. Fahim ! Celle qui m'a donné l'enthousiasme de me tourner continuellement vers la science.

Comment je peux ne pas présenter ici mes remerciements à M. Mohammad Shakibaei ! Celui qui m'a fait connaître le physique et l'électronique. Celui qui s'est éteint juste le lendemain de mon examen de physique du BAC.

Comment je peux ne pas présenter ici mes remerciements au Prof. Alain Greiner ! Celui qui m'a fait l'honneur de me nommer son stagiaire de DEA et puis son thésard. Celui qui m'a fait confiance pendant presque quatre années de ma présence dans son laboratoire avec une grande liberté d’action. Celui qui m'a aidé, guidé, et conseillé pour préparer cette thèse. Celui qui m'a appris comment faire la recherche et comment réfléchir scientifiquement. Celui qui m'a encouragé d'être sévère dans la vie professionnelle malgré tous les problèmes de la vie personnelle.

Comment je peux ne pas présenter ici mes remerciements au Prof. Christian Piguet et au Prof. David Atienza ! Ceux qui m'ont fait l'honneur d'être rapporteurs de ce manuscrit. Ceux qui m'ont donné plus de confiance en moi avec leurs rapports aussi précis.

Comment je peux ne pas présenter ici mes remerciements au Prof. Frédéric Petrôt, au Dr. Pascal Vivet, et au Dr. Riccardo Locatelli ! Ceux qui m'ont manifesté leur gentillesse d'être examinateurs de la soutenance de cette thèse.

Comment je peux ne pas présenter ici mes remerciements au Dr. Jean-Lou Desbarbieux, au Dr. Franck Wajsburt, et au Prof. François Anceau ! Ceux qui m'ont donné des conseils pratiques sur ce travail de thèse.

Comment je peux ne pas présenter ici mes remerciements aux relecteurs anonymes de mes publications ! Ceux qui m'ont aidé à améliorer le niveau de travail de cette thèse, en me donnant des suggestions et commentaires très utiles.

Comment je peux ne pas présenter ici mes remerciements au Dr. Pirouz Bazargan ! Celui qui m'encourageait constamment à continuer, et surtout à mon arrivée en France et lorsque ma faible connaissance en français me décourageait.

Comment je peux ne pas présenter ici mes remerciements à Mme. Shahin Mahmoodian ! Celle qui m'a appris à être toujours prêt à s'engager comme elle l'est.

Comment je peux ne pas présenter ici mes remerciements à Hassan Aboushady, Adrijean Adriahantenaina, Wahid Bahroun, Nicolas Beilleau, Sophie Belloeil, Mounir Benabdenbi, Cécile Braunstein, Jean-Paul Chaput, Hervé Charlery, Alexandre Coveliers, Laurent Delamarre, Anne Derieux, Nathalie Drach-Temam, Damien Dupuis, Daniela Genius, Karine Heydemann, Ramy Iskander, Marie-Minerve Louerat, Zied Marrakchi, Habib Mehrez, Ivan Miro, Hayder Mrabet, Ludovic Noury, François Pecheux, Corinne Poutier, Patricia Renault, Mathieu Rosiere, Marek Sroka, Sami Taktak, Matthieu Tuna, et plein d'autre gens du laboratoire ASIM ! Ceux qui ont mis une telle ambiance que le travail au sein de ce laboratoire pendant ces presque quatre ans m'est paru si captivant.

Si c'est mon dernier diplôme universitaire, comment je peux ne pas dire ici qui m'a supporté pendant cette longue durée de mes études !

Comment je peux ne pas présenter ici mes reconnaissances à Akbar ! Celui qui m'a donné l'existence. Celui qui m'a appris à être persévérant pour aboutir. Celui qui me soutenait et subvenait à tous mes besoins.

Comment je peux ne pas présenter ici mes reconnaissances à Mahboobeh ! Celle qui m'a fait naître. Celle qui a consacré et sacrifié sa vie pour sa famille. Celle qui m'a montré la beauté de la vie, même lorsque mon père était aux fronts de la guerre. Celle qui m'incitait toujours à continuer mes études.

Comment je peux ne pas présenter ici mes reconnaissances à Mahnaz, Mehrdokht, Mehraneh, et Maryam ! Celles qui m'ont donné la fierté d'être leur frère. Celles qui m'ont transmis leur envie de me voir réussir le plus loin possible dans ma vie personnelle et professionnelle.

Comment je peux ne pas présenter ici mes reconnaissances à Farideh et à Behrooz ! Ceux qui m'ont hébergé gentiment au début de mon arrivée en France. Ceux qui m'ont témoigné leur amitié par leur présence et leur grand soutien.

Comment je peux ne pas présenter ici mes reconnaissances à mes amis ! Ceux qui ont coloré ma vie ! Ceux qui m'ont partagé leurs bonheurs. Ceux qui m'ont partagé mes malheurs.

Enfin, et surtout, comment je peux ne pas présenter ici mes reconnaissances à Sahar ! Celle qui m'a fait l'honneur d'être ma chère épouse. Celle qui m'accepte comme je suis. Celle qui m'a restitué la motivation de continuer mes études supérieures, alors que je l'avais perdue pendant ma licence. Celle qui me comprend et me soutient dans les moments les plus difficiles, surtout pendant les derniers mois de la rédaction de ce manuscrit.

Non ! Moi je ne peux pas ! Et, je présente donc ici tous mes remerciements et toutes mes reconnaissances.

Acknowledgement

For a long time I thought that the text of acknowledgement and gratitude which is at the beginning of the manuscript of the theses is only a cliché, and I thought that it would be necessary to avoid these superficial thanks. Now, it's my turn, and how I can express my gratitudes otherwise ! How I can tell my route !

If it is my last university degree, how I can not say here who formed me during my years of studies !

How I can not present here my gratitude to Ms. Gorgikia ! The one who taught me how to read and write.

How I can not present here my gratitude to Ms. Fahim ! The one who gave me the enthusiasm for turning constantly to the science.

How I can not present here my gratitude to Mr. Mohammad Shakibaei ! The one who made me aware of the physics and electronics. The one who just passed away the day after my physics examination of BAC.

How I can not present here my gratitude to Prof. Alain Greiner ! The one who honored me with appointing me his trainee of MSc and then PhD. The one who relied me during nearly four years of my presence in his lab with a great freedom of action. The one who helped me, guided me and advised me to prepare this thesis. The one who taught me how to do research and how to think scientifically. The one who encouraged me to be tough in the professional life in spite of all the problems of the personal life.

How I can not present here my gratitude to Prof. Christian Piguet and Prof. David Atienza ! Those who honored me with being rapporteurs of this manuscript. Those who gave me more confidence in myself with their reports so precise.

How I can not present here my gratitude to Prof. Frédéric Petrôt, Dr. Pascal Vivet, and Dr. Riccardo Locatelli ! Those who showed me their kindness to be examiners of the defense of this thesis.

How I can not present here my gratitude to Dr. Jean-Lou Desbarbieux, Dr. Franck Wajsburt, and Prof. Francois Anceau ! Those who gave me practical advices onto this thesis work.

How I can not present here my gratitude to the anonymous reviewers of my publications ! Those who helped me to improve the standard of this thesis work, by giving me suggestions and very useful comments.

How I can not present here my gratitude to Dr. Pirouz Bazargan ! The one who constantly encouraged me to continue, especially in my arrival in France and when my weak knowledge in French discouraged me.

How I can not present here my gratitude to Ms. Shahin Mahmoodian ! The one who taught me to be always ready to make a commitment as she is.

How I can not present here my gratitude to Hassan Aboushady, Adrijean Adriahantenaina, Wahid Bahroun, Nicolas Beilleau, Sophie Belloeil, Mounir Benabdenbi, Cecile Braunstein, John-Paul Chaput, Hervé Charlery, Alexandre Coveliers, Laurent Delamarre, Anne Derieux, Nathalie Drach-Temam, Damien Dupuis, Daniela Genius, Karine Heydemann, Ramy Iskander, Marie-Minerve Louerat, Zied Marrakchi, Habib Mehrez, Ivan Miro, Hayder Mrabet, Ludovic Noury, Francois Pecheux, Corinne Poutier, Patricia Renault, Mathieu Rosiere, Marek Sroka, Sami Taktak, Matthieu Tuna, and many other people of the laboratory ASIM ! Those who put such an atmosphere as the work within this laboratory during these almost four years is seemed to me so fascinating.

If it is my last university degree, how I can not say who supported me during this long period of my studies !

How I can not present here my acknowledgement to Akbar ! The one who gave me the existence. The one who taught me how to be persevering to succeed. The one who supported me and met all my needs.

How I can not present here my acknowledgement to Mahboobeh ! The one who gave me life. The one who has dedicated and sacrificed his life for his family. The one who showed me the beauty of the life, even when my father was at the fronts of the war. The one who always incited me to continue my studies.

How I can not present here my acknowledgement to Mahnaz, Mehrdokht, Mehraneh and Maryam ! Those who gave me the pride to be their brother. Those who passed on to me their urge to see me achieving as far as possible in my personal and professional life.

How I can not present here my acknowledgement to Farideh and Behrooz ! Those who kindly hosted me at the beginning of my arrival in France. Those who have shown their friendship by their presence and their great support.

How I can not present here my acknowledgement to my friends ! Those who have colored my life ! Those who have shared me their joys. Those who have shared me my misery.

Finally, and especially, how can I not present here my acknowledgement to Sahar ! The one who honored me with being my dear wife. The one who accepts me as I am. The one who returned me the motivation to continue my higher educations, whereas I had lost it during my

BSc. The one who understands me and supports me in the most difficult moments, especially during the last months of writing this manuscript.

No ! I can not ! And therefore I present here all my gratitudes and my acknowledgements !

To … which does not exist …

Preface

This PhD thesis has been done at the computer science laboratory of the university of Paris 6 (LIP6: Laboratoire d'Informatique de Paris 6 / UPMC: Université Pierre et Marie Curie), at the System-on-Chip department (SoC, previously ASIM: Architecture des Systèmes Intégrés et Microélectronique). The work started in October 2004 and has been supervised by Professor Alain Greiner that his experience within the field of Networks-on-Chip (NoCs) and Multi-Processor Systems-on-Chip (MP-SoCs) has deeply helped to advance the project. In fact, my PhD thesis is in the continuation of the SPIN Micro-Network project presented as three PhD theses, all supervised by Prof. Alain Greiner:

1. Pierre Guerrier, "Un Réseau d'Interconnexion pour Systèmes Intégrés (An Interconnect Network for Integrated Systems)", 05/2000
2. Hervé Charlery, "Intégration d'un Micro-Réseau à Commutation de Paquets dans un Système Multiprocesseurs à Mémoire Partagée Intégré sur Puce (Integration of a Packet-Switching Micro-Network in a Shared-Memory Multi-Processor System-on-Chip)", 12/2005
3. Adrijean Adriahantenaina, "Implémentation Matérielle d'un Micro-Réseau SPIN à 32 Ports (Physical Implementation of a SPIN Micro-Network with 32 Ports)", 01/2006

In parallel with my work at LIP6, another PhD project, initially entitled "Implantation sur silicium d'un micro réseau intégré sur puce (Silicon Implementation of a Network-on-Chip)", under supervisory of Prof. Alain Greiner was introduced. STMicroelectronics and CEA-LETI in Grenoble host the project and Ivan Miro Panades prepares the thesis. He has contributed also on certain parts of my research works. As a result, we have published some papers listed in the end of the thesis manuscript. The word of "we", in the present article, signifies our NoC research group consisting of Alain Greiner, Ivan Miro Panades, and me.

Contents

Figures

Introduction

Networks-on-Chip (NoCs) are a new design paradigm for scalable high-throughput communication infrastructures, in Systems-on-Chip (SoCs) with billions of transistors. The idea of NoCs is dividing a chip into several independent clusters (subsystems) connected together by a global communication architecture.

The NoC technology attempts to solve the bandwidth bottleneck of traditional interconnects. In fact the classical interconnects such as shared busses do not scale when number of components to interconnect increases. NoCs are suitable to have a much more modular and flexible design flow. In additional, NoCs try to overcome fundamental physical issues introduced when using deep submicron (DSM) technology. In nanometer fabrication processes the largest part of delays is related to global wires and the interconnect propagation delay may exceed a clock cycle time needed by high-speed applications.

Furthermore, because of physical limitations, henceforth it is extremely hard and even in some cases impossible to distribute a synchronous clock signal on the entire wide chip area. NoCs using Globally Asynchronous Locally Synchronous (GALS) techniques address this difficulty. In these techniques the chip area is divided into several independent clusters and each cluster is clocked by a different clock signal. The NoC architecture is the asynchronous global communication infrastructure which spreads on the entire chip.

One of the most important issues in design of a NoC providing GALS system is to have a robust solution to interface clock boundaries. The main problem is to overcome the phenomenon of metastability and physical synchronization failure. The trade-off between robustness and latency/throughput is a major concern in design of such interfaces.

One technique which seems the most compatible with the GALS paradigm is the use of a NoC with a fully asynchronous circuit design, in which the need of the asynchronous global

communication is naturally provided. This type of NoC can construct a GALS architecture by providing synchronous compliant interfaces to each local subsystem. Although the metastability could not be totally suppressed, an asynchronous NoC might reduce its possibility by limiting the place of its occurrence just to the interfaces between network and subsystems.

This PhD thesis tries to describe the problems being encountered when designing a NoC for GALS architectures. The focus is on the distributed micro-networks developed at the LIP6 laboratory (DSPIN: "Distributed Scalable Packet-switching Integrated Network" and, in particular, ASPIN: "Asynchronous Scalable Packet-switching Integrated Network"). In reality, DSPIN and ASPIN are two different implementations of the same generic NoC architecture. They are two different approaches to comply with GALS constraints. The first way (DSPIN) is to use a distributed multi-synchronous network, in which the routers are synchronous circuits. The second (ASPIN) approach is to have a fully asynchronous network architecture. As the general architectures and provided services of both networks are identical, DSPIN vs. ASPIN performances comparison, presented in this thesis, may help to answer this question: "Will future Networks-on-Chip be synchronous or asynchronous?"

Thesis Overview

In the following chapters and sections, I will first talk about the definition of problems concerning constructing Globally Asynchronous Locally Synchronous (GALS) architectures and the motivation of implementing asynchronous Networks-on-Chip. Chapter 1 describes the issues about the clock distribution network in large synchronous systems, and explains the problem of metastability. It focuses on the definition of problems relating to the ASPIN implementation, and it gives an overview of the questions this thesis will address.

Chapter 2 tries to introduce the state-of-the-art methods for GALS compatibility. The discussion will include the solutions for the problem of clock boundaries and synchronization failure. In this chapter some existing micro-network architectures compatible with GALS paradigm will be indicated, including DSPIN.

As an asynchronous implementation of DSPIN, the ASPIN architecture will be discovered in chapter 3. Firstly a general architecture will be presented and it will be explained how ASPIN addresses the problem of long wires. Then, the architectures will be detailed and the behavior specification of different modules will be explained.

Our robust solutions to interface the mixed and hybrid timing domains in GALS architectures are demonstrated in chapter 4. This chapter presents the detailed architectures and circuit implementations. Some experimental results are shown, as well as some physical layout examples.

In chapter 5, I will explain the phenomenon of the network saturation. The network saturation threshold will be considered in our two approaches of DSPIN and ASPIN. This chapter describes

how the network saturation threshold could be improved by increasing the flit storage capacitance of the routers. It claims that the price may be much cheaper in ASPIN approach. I will discuss about the importance of the throughput on the network saturation threshold and the influence of the speed ratio between network and flit injectors, i.e. subsystems. Some simulation results are displayed. Finally this chapter proposes a new approach to improve the saturation threshold in large high-speed networks.

A systematic physical performance comparison between DSPIN and ASPIN which are two implementation ways to have a NoC-based GALS architecture, will be presented in Chapter 6. The considered physical performance parameters are Silicon Area, Communication Throughput, Packet Latency, and Power Consumption. As a critical issue the problem of long wires is taken into account. The experimental results demonstrated in this chapter could be thought as physical characteristics of ASPIN.

One could find a summary of my thesis in the *Summary* section of this manuscript, just before appendixes.

For the readers who are not familiar with Networks-on-Chip, Appendix A presents the principle ideas about the NoC design paradigm, motivations, and perspectives. In this chapter the NoC design methodologies will be elaborated, and the current and future challenges in Systems and Networks on-Chip will be explained.

As an introduction for who have no idea about asynchronous circuits, Appendix B briefly introduces the asynchronous circuits, the design methodologies, and in particular the asynchronous data communication protocols.

Appendix C presents three different views of the specific basic cells of the asynchronous circuits, used in the designs illustrated in this thesis: the behavior as a VHDL model, the transistors net-list as a SPICE model, and the physical layout.

And finally, at the end of the manuscript, the publications relating to this PhD thesis are listed.

Chapter 1

Problem Definition

The shrinking of processing technology in the Deep SubMicron (DSM) domain aggravates physical issues. As chip technologies scale down, the effect of wires on delay and power becomes predominant. Decreasing clock cycle time and increasing die sizes make it henceforth extremely hard, if not impossible, to distribute a single global clock signal over the entire chip area. In addition to the clock skew which is claiming a larger relative part of the total cycle time, the clock distribution network, needed to implement a globally synchronized signal, is demanding increasing portions of the power and area budget [1].

The clock distribution network distributes the clock signal from a common point to all the elements that need it. The skew of the clock signals can severely limit the maximum performance of the entire system and create catastrophic race conditions in which an incorrect data signal may latch within a register. Highly symmetric and hierarchical clock trees are used to minimize clock skew, but an increase in clock signal delay is incurred, which does not make for high speed applications. Symmetric fractal H-Tree (demonstrated in Figure 1) is a common design example of clock distribution network for routing the clock signal to all parts of a chip with equal propagation delays to each part [2].

Even with H-Tree structured clock distribution networks the real amount of clock skew depends strongly on the physical size, the control of the fabrication process, and temperature variations. The difficulty in scaling such clock distribution networks is the primary reason for the recent effort placed on finding some new effective solutions, including the obvious solution of using globally asynchronous communication between locally synchronous regions [3]. A Globally Asynchronous Locally Synchronous (GALS) chip is divided into several independent clusters and each cluster is clocked by a different clock signal. The advantageous aspect of this method is the reducing of the problem of clock distribution to a number of smaller sub-problems.

Networks-on-Chip offers a structured approach to design a GALS architecture. Since NoCs span the entire chip, the network could be globally asynchronous part of the system, while the subsystem modules are the locally synchronous parts. Moreover, the principle idea of a NoC, in which a system is divided into several independent subsystems, helps to physically distinguish different timing domains. In other words, each subsystem could have its own clock domain without imposing any timing constraints on other clusters. Naturally this opportunity in such systems introduces some new design challenges, which will be addressed in the following chapters of this thesis.

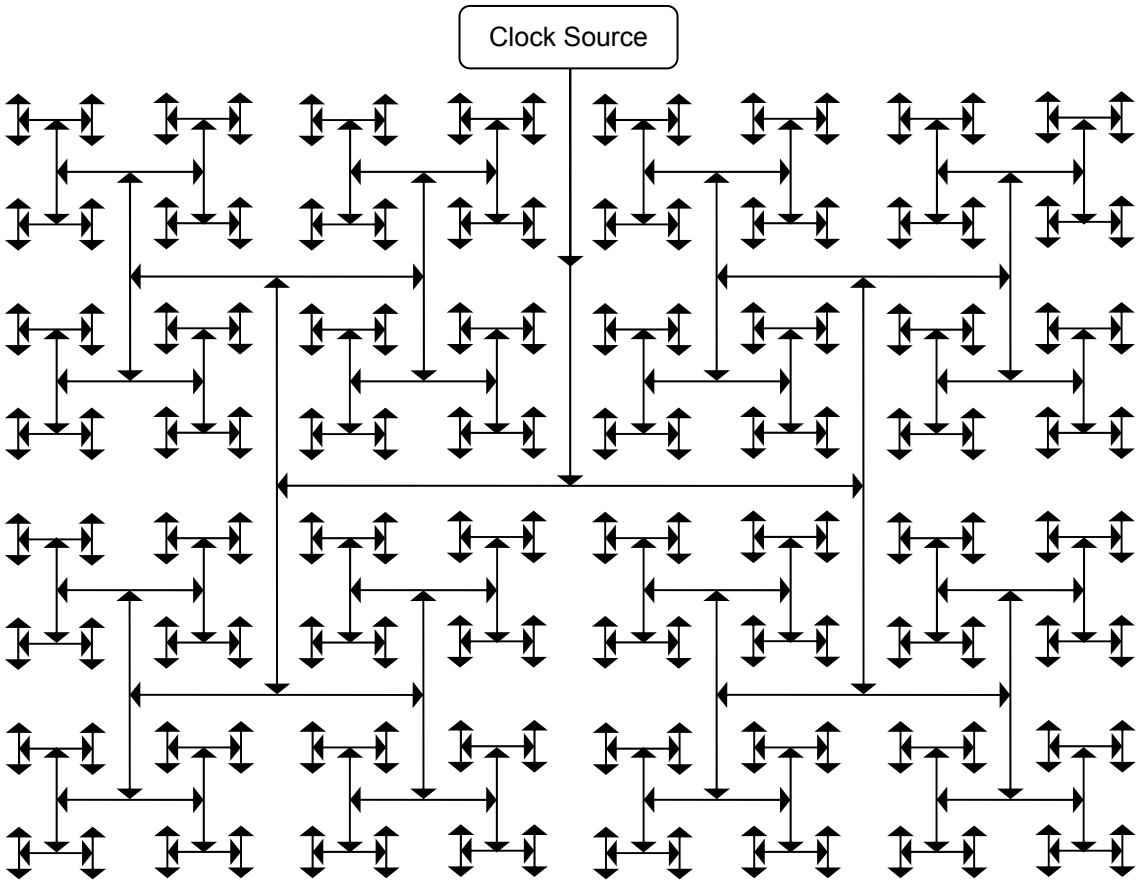


Figure 1. The first eight levels of an H-Tree (Clock Distribution Network)

1.1 Clock Boundaries

Figure 2 shows a 9-tile multi-clocked system arranged into a two-dimensional mesh. In the design of such GALS systems the main concern is the problem of clock boundaries, i.e. how separated synchronous domains can robustly communicate together. Transferring data between different timing domains requires safe synchronization. In fact sampling an asynchronous data into synchronous domain is similar to synchronizing the input data with the clock signal. Sampling the

data just at the time that the data is changing (i.e. when there are setup-time and hold-time violations) may possibly cause synchronization failure. The most important issue in GALS paradigm is to prevent synchronization failures.

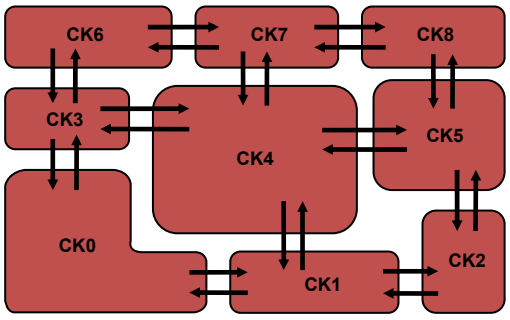


Figure 2. A Multi-Clocked System arranged in Two-Dimensional Mesh Topology

1.1.1 Metastability

The metastability is the cause of the synchronization failure. The metastability is the possibility of a non-stable state to persist for a long (theoretically unlimited) period of time. In other words the metastability is an inevitable result of any attempt to map a continuous domain to a discrete one. There will always be points in the continuous domain which are equidistant from the points of the discrete domain, making a decision concerning which discrete point to select a difficult and potentially lengthy process.

In effect a bistable device with hysteresis can enter into a metastable state and has a positive probability that it will remain indefinite for any given period of time, though over time the probability exponentially decreases. To better understand the difference between stability states Figure 3.a presents an example of a bistable system consisting of a ball and a hill. Clearly the location of the ball at the bases of the hill is much more stable than at the top of the hill. Actually at the hilltop the ball is in a metastable state because just for example the slightest air current would eventually cause the ball to roll down one side or the other. The ball at the hillsides is absolutely unstable.

The flip-flop is an electronic device that is susceptible to metastability. It has two well-defined stable states, traditionally designated 0 and 1. Assuming the use of a positive edge triggered master-slave D type flip-flop (as displayed in Figure 3.b), when the rising edge of the clock (CK) occurs at a point in time when the input (A) is causing its master latch to transition, the flip-flop is highly likely to end up in a metastable state. This rising clock causes the master latch to try to capture its current value, where Y follows the value of X and X follows the value of Y. Either of these two signals cannot instantaneously change the level and it must transition through the analog region where the other is transitioning through its analog region. Figure 3.c plots the

transition of X versus Y and Y versus X. As can be seen, these two curves have three intersection points, showing three stable points. Two of these three points correspond to the stable states of logic 0 and logic 1. But, though in theory the third point is also a stable state and a latch could remain there indefinitely, in reality thermal and induced noise will jostle the state of the latch causing it to move from this metastable state into either the logic 0 or logic 1 state. The most perfectly caught metastable state (on the very top of the hill) results in the longest time required for the flip-flop to resolve itself to one of the stable states.

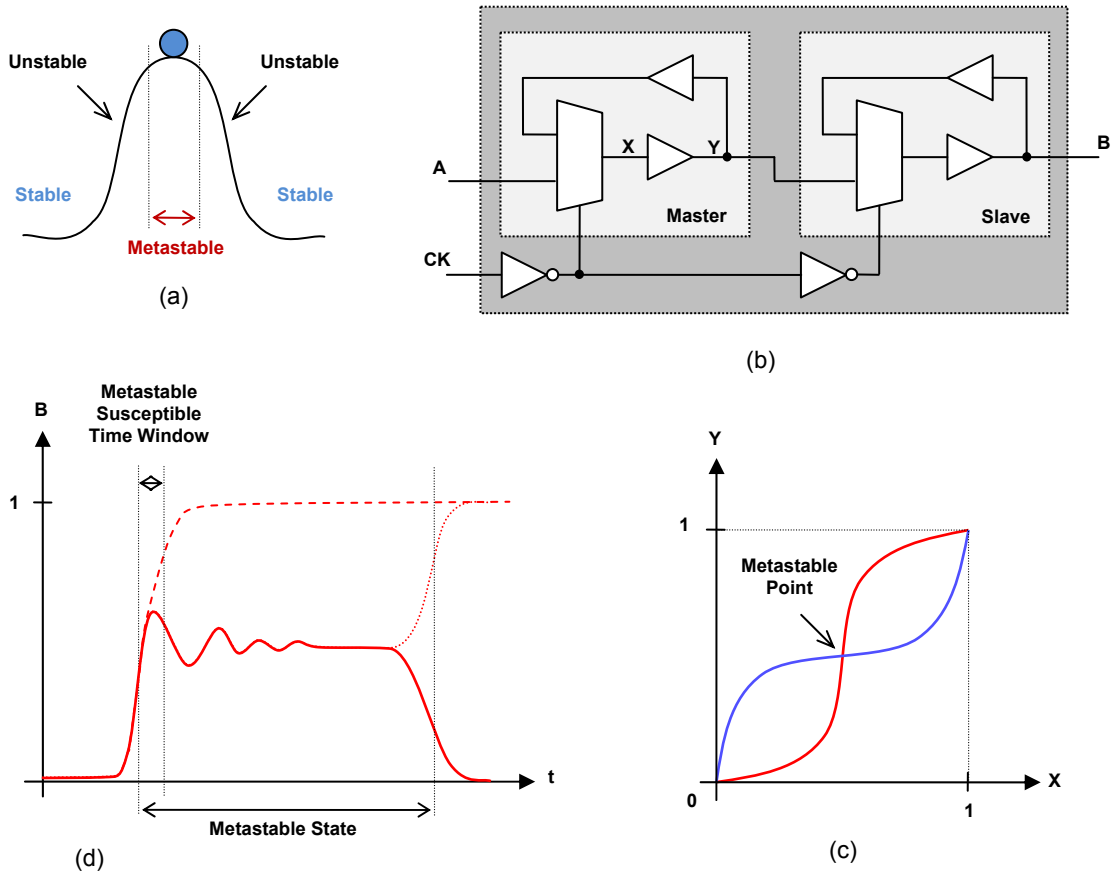


Figure 3. Metastability: (a) Ball and Hill, (b) Master-Slave D type Flip-Flop, (c) X vs. Y and Y vs. X, and (d) Flip-Flop Output in Metastable state

After the rising edge of the clock the slave latch is opened allowing the flip-flop output (B) to follow the latched value of the master. Figure 3.d shows the chronograph of the flip-flop output when it is in a metastable state. The oscillating output has undesirable intermediate values between logic 1 and logic 0. Here the output finally settles down to 0, though unpredictably it was probable to settle down to 1. How long it takes to settle down (duration of the metastable state) is not predictable, and depends on the technology of the flip-flop. When the transition curves of X and Y are sharp the probability of remaining in the metastable state is reduced, though it is always

unpredictable. The undesirable value of the output can propagate in combinational parts of the circuit and causes the failure in the functionality of the system.

1.2 Asynchronous Network-on-Chip

Presenting novel NoC architectures to cope with Globally Asynchronous Locally Synchronous paradigm issues is now a very high active research domain of NoC technologies. Potentially, NoCs are compatible with the idea of GALS that needs to clusterize the chip into several independent subsystems. But the question that remains is how the network itself must be clocked, and how we can deal with the problem of synchronization and metastability on clock boundaries.

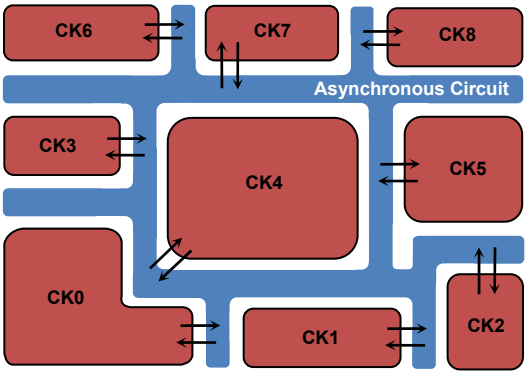


Figure 4. A Multi-Clocked System using an Asynchronous Network

Since one obvious way to eliminate the problem of clock skew is to use asynchronous logic, one can suppose the use of an asynchronous circuit for the global communication architecture. This is not the only way as will be seen in the next chapter. But the asynchronous design is a natural approach to construct GALS architectures. A large number of locally synchronous islands can communicate together via a global asynchronous network which does not involve the issue of synchronization. The possibility of the synchronization failure will be limited only at the network interfaces, where synchronous data has to enter into the asynchronous network, and where asynchronous data is obligated to go into the synchronous subsystems. In Figure 4 the black arrows resemble required synchronizing modules.

1.2.1 Clock Boundary Interfaces

Connecting synchronous IP cores to the asynchronous network requires designing some special interfaces involving certain serious problems. As an important responsibility, this kind of interfaces has to provide a robust synchronization. The increase of synchronization safety is often accompanied by a penalty on latency. To achieve performance, with any choice of synchronizer, such interfaces must fulfill the maximum throughput of one transmitted data per clock cycle, at the same time of optimizing silicon area and minimizing latency. Furthermore, the clock boundary

interface is involved to the flow control issues too. Overcoming all these problems, Chapter 4 describes the design of new interfaces well-suited to be instantiated at clock boundaries of an asynchronous NoC.

1.2.2 Design Complexity

In synchronous circuits an unpredicted variation in propagation delays possibly results in a serious malfunction of the system. Delay-insensitive asynchronous circuits can operate correctly in presence of variable delays in gates and wires. As a result a delay-insensitive asynchronous NoC will strongly simplify the work of the system designer. The network's modules can be instantiated by a plug-and-play fashion, without any timing constraint.

Usually a delay-insensitive asynchronous circuit relies on a delay-insensitive data encoding. The use of delay-insensitive data encoding (such as dual-rail) is often accompanied by more logic, as needed for extra data lines and required completion detection units. The area optimization is a major concern in design of NoCs which are extremely cost-constrained circuits.

The design of a robust and delay-insensitive asynchronous circuit is not evident and due to the heavy restrictions such systems are complex to implement in a CMOS technology. Additionally there is very few synthesis tools for asynchronous circuits. A Signal Transition Graph (STG) is a common way to formalize the timing diagram, although defining an STG is not easy, especially for large systems. Chapter 3 presents the asynchronous implementation and corresponding design complexities for ASPIN distributed NoC architecture.

1.2.3 Long Wires Issue

In the shrinking of process technology the length of local wires usually shrinks accordingly. In the contrary, since the die size does not decrease, global wire lengths do not reduce. The largest part of the delays is related to the global (long) wires now.

In a distributed Networks-on-Chip the global wires are the wires connecting routers. Long wires in a NoC likely have large propagation delays incompatible with the required throughput. Long wires could be a performance bottleneck and are a real crucial issue in design of NoC architectures.

1.3 Performance Comparison

According to the fact that both synchronous and asynchronous circuits have their own advantages and disadvantages, it is not clear what is the best choice to implement a NoC-based system compatible with GALS paradigm. Chapter 5 and Chapter 6 are an attempt to predict whether future NoCs will be synchronous or asynchronous [4]. We believe that the answer of this question could be found by analyzing performance parameters of two synchronous and asynchronous

network providing the same services and having the same general architectures. DSPIN (briefly described in Section 2.6) and ASPIN (detailed in Chapter 3 and Chapter 4) are compared from the viewpoint of silicon area, power consumption, communication throughput, communication latency, and network saturation threshold.

1.3.1 Saturation Threshold

Any interconnect saturates when the average offered load reaches a point called saturation threshold, where communication latency becomes unpredictable and exponentially increases. Saturation threshold improvement is the main motivation supporting the NoC paradigm.

The exact value of saturation threshold depends on the traffic load, average packet length, destination distribution, and distributed buffering capacity. These arguments should be taken into account as parameters of a generic simulation platform in order to evaluate the value of saturation threshold.

Another issue in this kind of simulation is the difference between levels of modeling: Traffic generator and analyzer ask for a system-level modeling, while gate-level models are appropriate for the asynchronous network.

The low value of saturation threshold (an important limiting factor for delay sensitive applications) convinces the system designers to pay more for ameliorating this key feature. A well known and efficient solution to improve the network saturation threshold is to grow buffering capacity of the network. But as a drawback, extra buffering capacity means extra power and area.

On the other hand, although Networks-on-Chip are much more scalable than traditional on-chip interconnects, network saturation can still become the system bottleneck. Scalability in NoCs means that the value of saturation threshold seems to be roughly independent on the number of communication units. Nevertheless when number of cores generating traffic augments the saturation threshold degrades and in huge systems really becomes a significant limiting factor. As a result the need of some new approaches seems necessary.

1.3.2 Physical Performance Parameters

Abstraction is mandatory to manage complexity in design and modeling. While abstract modeling and automated synthesis enables complex system design, such an approach increases the variability of the physical and electrical parameters. Such parameters can determine if a NoC is cure or curse.

In order to evaluate physical performance parameters close to the exact values, a post layout model of the design components is needed. That is to say before extracting transistor-level electrical models, gate-level structural models must be developed and then the cells must be placed and the design must be routed.

As emphasized before, due to the DSM technology long wires are now the dominant factor of delay and power consumption, and as a consequence in the evaluation of physical performance parameters they also have to be considered. In a NoC the exact value of the physical parameters directly depends on the cluster size.

1.4 Summary

Because of physical limitations, henceforth it is extremely hard, if not impossible, to distribute a global synchronous clock signal over a wide chip area. As a solution, Networks-on-Chip using Globally Asynchronous Locally Synchronous techniques divide the chip area into several independent synchronous clusters. Each cluster is clocked by a different clock signal and thereby the problem is reduced to a number of smaller sub-problems. The network could be the asynchronous global communication infrastructure of the system. But two basic questions remain which are how the network itself must be clocked and how we can deal with the problem of synchronization at clock boundaries. Accordingly, this PhD thesis will address the following questions:

- Presenting a novel asynchronous NoC architecture to cope with the issues of the Globally Asynchronous Locally Synchronous paradigm and to reduce the possibility of metastability along the packet path. This architecture must also address the crucial issue of the global long wires which likely have large propagation delays, incompatible with the required throughput.
- Presenting a new design of some special interfaces to be instantiated at clock boundaries. These architectures have to provide a robust synchronization and with any choice of synchronizer must fulfill the maximum throughput of one data per clock cycle.
- Presenting an evaluation of the network saturation threshold which is a key feature and the main motivation supporting the NoC paradigm. This evaluation has to consider the influence of two parameters: the flit storage capacity of the network and the network throughput. Moreover, since when the number of cores generating traffic augments the saturation threshold degrades and in huge systems becomes the system bottleneck, a new approach to improve the saturation threshold must be addressed.
- Presenting a systematic comparison between physical performance parameters in two synchronous and asynchronous networks providing the same services and having the same general architectures, as it is not clear which architecture type is the best choice to implement. As a predominant factor, in the evaluations the long wires effects must be taken into account too.

Chapter 2

State of the Art

Regarding the problems in multi-clocked systems and the synchronization failure, several solutions have been proposed. Related solution varies according to various hypotheses on phases and frequencies of the clock signals. Some authors (e.g. [5], [6], and [7]) have suggested plesiochronous solutions which rely on exact or nearly exact frequency and phase matching of clocks. The Globally Pseudochronous Locally Synchronous scheme (GPLS) is proposed in [8]. This clocking scheme (quasi-synchronous) distributes a clock with a constant phase difference between clusters. Mesochronous solutions are more general. In these approaches (e.g. [7], [9], [10], and [11]) it is argued that distributing the same frequency in several clock domains is not too difficult. The main problem is the undefined skew between clock phases. In heterochronous approaches (e.g. [7], [12], [13], and [14]), all clock signals can have different frequencies, but with fixed and integer ratios. Table 1 summarizes these conditions.

Table 1. Timing-Dependent Methods

	Δ Frequency	Δ Phase
Synchronous	0	0
Plesiochronous	ε	ε
Pseudochronous	0	Constant
Mesochronous	0	Undefined
Heterochronous	Rational	Undefined
Multi-synchronous	Undefined	Undefined
Asynchronous	-	-

Most of these timing-dependent methods roughly avoid the need for global synchrony, and in particular, popular mesochronous clock distribution has been used in a variety of commercial products. For instance Arteris's NoC, called Danube, utilizes asynchronous bridges between locally mesochronous network clusters [15] and Intel's 80-core tera-flap processor (mentioned in A.1) uses a 5-port wormhole packet-switching router with mesochronous clocking scheme.

2.1 Multi-Synchronous NoCs

In design of a high-performance SoC, a fundamental challenge may be the capability of operating under totally independent timing assumptions. Typically the clusters (subsystems) of a large system are synthesized independently and have individual timing characteristics. To achieve maximum performance, each cluster should operate with its own timing limitations.

Such a multi-synchronous system contains several synchronous subsystems clocked with completely independent clock signals. The routers are distributed in each subsystem and are connected to the north, south, east, and west neighbors by means of bidirectional point-to-point links. This architecture represents the problem of communication and synchronization failure at each clock boundary between each two neighbor routers.

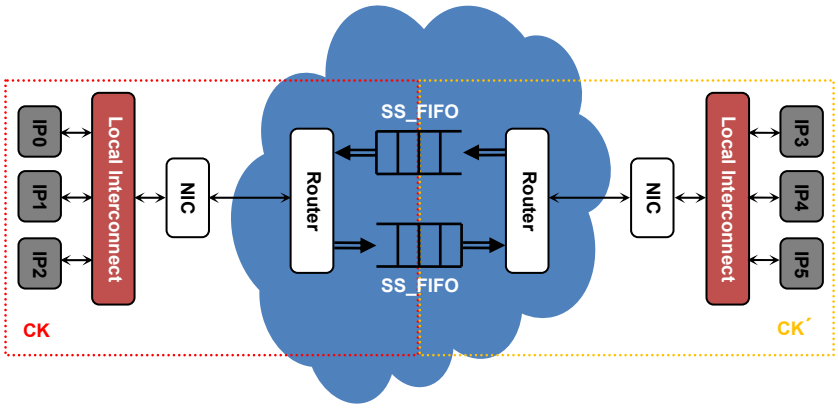


Figure 5. Two adjacent clusters in a multi-synchronous system

Relating to this problem, several authors (e.g. [7], [16], [17], and [11]) have proposed different types of special FIFO architectures that can be used as robust interfaces. In these FIFOs the producer and consumer sides use two different clock signals, hence called bisynchronous FIFOs. As shown in Figure 5, bisynchronous FIFOs (SS_FIFOs in the figure) can be instantiated between each two adjacent routers. In this approach each router is a synchronous circuit and clocked by the local clock signal of the encompassing subsystem. In this thesis in Chapter 4 the architecture of a new bisynchronous FIFO will be presented too.

One other possibility of clocking could be that the routers use a dedicated mesochronous clock signal, as displayed in Figure 6. Two advantages can be accounted for these new clocking ways. First, if the routers are clocked by different clock signals with different frequencies, the performance of the network in terms of latency and throughput is not easy to be predicted. Each path between a source and a destination includes a number of routers operating with individual speeds. Additionally the slow clusters become bottlenecks for all paths crossing them. The use of a single and dedicated clock signal for the network resolves both problems, as all routers will work with the same speed. The second advantage could be that the routers can likely use a faster clock signal than the subsystems. The use of a fast mesochronous dedicated clock signal for the network reduces the network latency and as will be explained in Chapter 5 could provide a better saturation threshold.

Nevertheless, the mesochronous network slightly complicates the design as it creates another clock boundary between the router and the Network Interface Controller (NIC). As robust interfaces, two bisynchronous FIFOs can be instantiated between the router and NIC of each subsystem. See Figure 6. Whereas mesochronous clocking way assumes that the clock frequency over the entire network is the same and it accepts only undefined phase skews, in order to optimize the network cost, within the network architecture we can use mesochronous solutions which have less complex design. Figure 6 proposes to use two mesochronous FIFOs (SS_FIFO´) (e.g. that of [11]) between each two adjacent routers, instead of bisynchronous FIFOs.

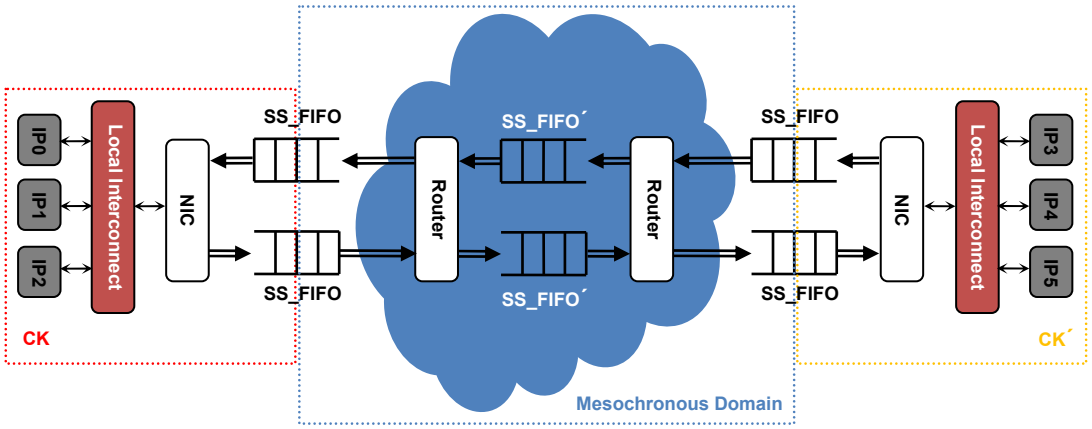


Figure 6. Two adjacent clusters in a multi-synchronous system using a mesochronous network

2.2 The Asynchronous Approach

A fully asynchronous NoC profits from the same advantages indicated above for a mesochronous network. The asynchronous network operates with a speed independent from the subsystems

clocks, and in particular the network works as fast as possible. Different from synchronous circuits based on the worst cases, an asynchronous circuit works at the average case conditions. Furthermore, as explained in Appendix B an asynchronous network offers some other beneficial features, such as zero idle state power dissipation, actual-case latency, plug-and-play reusability, delay insensitivity, etc.

An asynchronous NoC reduces the number of synchronizing interfaces required to be instantiated in the network. In fact an asynchronous NoC can construct a GALS architecture if it provides synchronous compliant interfaces to each local subsystem. In this case, the synchronization failure can only happen at the network interfaces in the first and last steps of each packet path, i.e. in the source cluster where the synchronous data must be converted to an asynchronous form and in the destination cluster where the asynchronous data must be entered into a synchronous subsystem. Selecting a well-behaved handshake protocol in an asynchronous circuit reduces the risk of metastability.

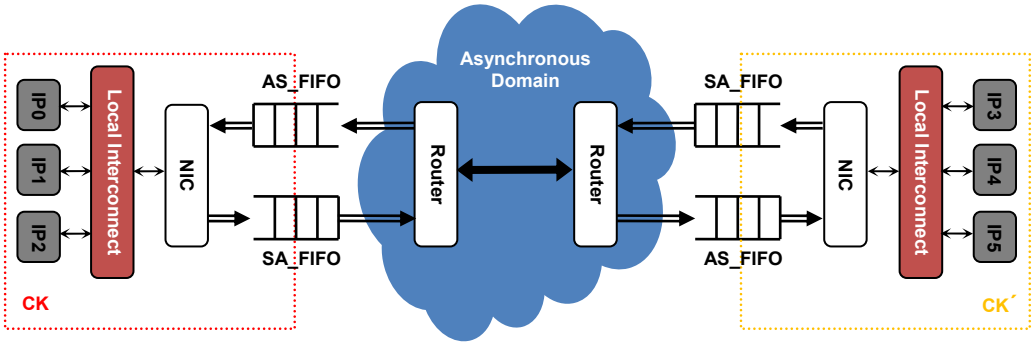


Figure 7. Two adjacent clusters in a multi-clocked system using an asynchronous network

Figure 7 illustrates the use of two special types of FIFOs instantiated at the network boundaries, between the synchronous network interface controllers and the asynchronous routers. Asynchronous-to-synchronous FIFO (AS_FIFO in the figure) is that which has an asynchronous data producer and a synchronous data consumer, and likewise the other FIFO is called synchronous-to-asynchronous FIFO (SA_FIFO).

Using FIFOs to interface mixed timing systems couples two fundamental issues which need to be considered in designing such interfaces: flow control (high level issue) and synchronization (low level issue). As will be explained this coupling reduces the need of hardware synchronizer to the handshake signals that are used for the flow control. Until now the design of a variety of Async-to-Sync and Sync-to-Async FIFO architectures have been presented (e.g. [16], [18], [19], [20], and [21]). The architecture of most of these published solutions is strongly dependent on the choice of a specific synchronizer. Designing two new Sync⇔Async FIFO architectures independent from the selected synchronizer is one of the principle purposes of this thesis and Chapter 4 is associated to this problem.

2.3 Synchronizer

The goal of a synchronizer is to prevent metastability. An interesting bibliography of the metastability and the related solutions is assembled in [22]. Some authors (e.g. [3], [19], [20], [21], and [23]) have recommended stretching the clock signal by dynamically modifying the cycle time (generating stoppable or pausible clocks) in order to satisfy the setup-time and hold-time. In these methods, instead of synchronizing asynchronous inputs with the clock, the clock is synchronized with asynchronous inputs [18]. The synchronizer must be able to detect that it will be in the metastable state and it stretches the clock cycle of the local system until the probability of metastability is zero. For more than one asynchronous input, clock must be stretched until all synchronizers ensure that the metastable states would not occur. As a consequence, as it is explained in [12] and [24], these solutions are not well suited for high speed designs with IP cores having large clock buffer delays.

Some others (e.g. [25], [26], and [27]) suggest using different kind of Schmitt triggers to avoid the metastability. In a Schmitt trigger when the input is higher than a certain chosen threshold the output is high, when the input is below another chosen threshold (lower) the output is low, and when the input is between the two the output retains its value. The trigger is so named because the output retains its value until the input changes sufficiently to trigger a change. Although the output of such synchronizers has well defined values (VDD or VSS) and undesirable metastable state values are prevented to propagate, this does not solve the problem because the precise duration of the metastable state remains unpredictable. The transition of the output is asynchronous compared with the clock signal of the next stage.

Additionally, as shown in [28], the Schmitt triggers can themselves go to a metastable state. In reality, the metastability remains poorly understood in some circles, and various engineers have proposed their own pet circuits said to solve or filter out the metastability. Typically these circuits simply shift the occurrence of the metastability from one place to another [29]. Apparently a widely held belief is that the metastability is an unavoidable characteristic and cannot be totally eliminated [30].

However, the probability of the metastability, typically expressed in terms of Mean Time Between Failures (MTBF), can be bounded to an acceptable value by a carefully designed synchronizer [31], [32]. MTBF gives the average time interval between two successive failures. In quantitative terms, the rate of entering metastable state for a simple flip-flop is $T_W \times f_D \times f_C$, where T_W is a parameter related to the metastable susceptible time window of the flip-flop, f_D is the frequency of pushing data across the clock domain boundary, and f_C is the clock frequency. For a 0.18 µm technology (where $T_W \approx 50$ps [29]) with a clock frequency of 200 MHz and receiving data every 1000 cycles, that rate is 2000 per second (MTBF=0.5 ms) ! This is clearly not acceptable as MTBF is typically designed to be at least ten times the expected life of the product.

The simplest (and the most common) way to increase MTBF is to use two cascaded Flip-Flops, as shown in Figure 8.a. This approach allows for an entire clock period for the metastable states in the first synchronizing flip-flop to resolve themselves. As the probability that the metastable state continue longer than a time T decreases exponentially with T, quantitatively, the allowed settling time (T) provides an extra safety factor of $e^{T/\tau}$, where τ is the settling time constant of the flip-flop determined with regard to the technology [31]. So:

$$MTBF = \frac{e^{T/\tau}}{T_W f_D f_C}$$

The MTBF for two cascaded Flip-Flops in a 0.18 μm technology (where $\tau \approx 10$ps [29]) with a clock frequency of 200 MHz and receiving data every 1000 cycles can be estimated to $e^{500}/2000$, or about 10^{206} years!

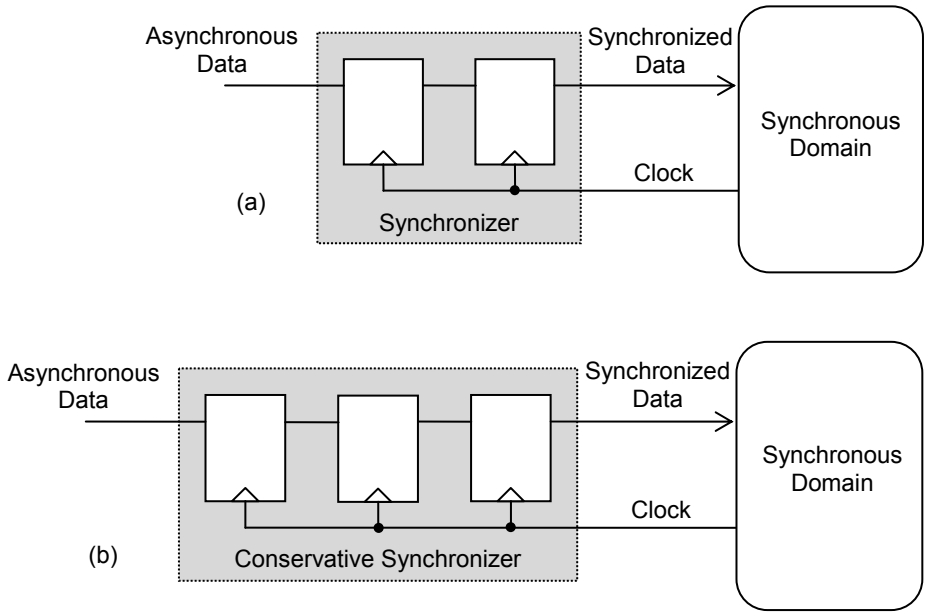


Figure 8. Cascaded Flip-Flops: (a) Robust synchronizer and (b) Conservative synchronizer

As conservative synchronizers, conservative designers can use more than two flip-flops. The MTBF can be improved by using several cascaded flip-flops (e.g. that displayed in Figure 8.b), but if synchronization latency is not an issue. Increasing the latency is the usual penalty for obtaining extra safety. We believe that the synchronizer choice must be a design decision depending on the application requirements. A generic architecture should support the trade-off between latency and robustness.

2.4 A Case Study of a Mixed-Timing FIFO Design

Tiberiu Chelcea and Steven M. Nowick have presented a mixed-timing FIFO design [16] using a modular approach: they defined a set of basic interfaces, both synchronous and asynchronous, that can be assembled to obtain a FIFO that meets the desired timing assumptions on both the senders and receivers end. Thus, the design of a mixed-timing FIFO is reduced to reusing and assembling a few pre-designed components. In other words these designs are adaptable to scenarios where communication can be between mixed-clock synchronous modules, asynchronous-synchronous modules, or just between asynchronous-asynchronous modules.

The FIFO employs a ring of storage elements in which tokens are used to indicate full or empty state. This simplifies detection of the state of the FIFO (full or empty) and thus makes synchronization robust. During continuous steady-state data transmission the probability of synchronization failure is zero. Synchronizers are added to the two global control signals (full and empty). The added latencies through the synchronizers may cause overflow/underflow. For example, when the FIFO becomes empty, the consumer interface is stalled two clock cycles later. So in the next clock cycle the consumer might request and read an empty cell. As a solution, the definition of full and empty are redefined and the main concern now is to detect when the FIFO is heading towards a full or empty state, and stop the respective interface in time (i.e. full means 0 or 1 cell being unused, while empty means only 0 or 1 cells being used). This helps in hiding the synchronization latency introduced between the state detection and the input/output handshaking.

Due to the early detection of full the producer may see an n-place FIFO as an n-1 place one. And, the early detection of empty may cause the FIFO to deadlock: it is possible that the FIFO still contains one data item, but the requesting consumer is still stalled. As the possibility of stalling the consumer when there is a single valid data item in the FIFO should be avoided, a bimodal empty detector is used. The detector, in addition to computing the new empty definition, also computes the true empty one.

Although these architectures are basically designed for only a pair of cascaded flip-flops as a synchronizer, for arbitrary robustness the designer might use more than two. In these new designs the definitions of new full and new empty have to change in order to avoid overflow/underflow. For example for k cascaded flip-flops the FIFO is full when there are no more than k empty cells, and the FIFO is empty when there are no more than k full cells. Notice that to avoid deadlock the normal empty detector must still be used.

However, the new implementation (as a solution for providing robustness) may have a negative impact on the throughput of the consumer interface, as for example for three cascaded flip-flops, when the FIFO has two data items and there have not been any recent read request, the two data items will be dequeued in three clock cycles, whereas with the basic FIFO (with two cascaded flip-flops) these two data items can be dequeued in two clock cycles.

Finally, the implementation of these mixed-timing FIFOs presented in [16] is based on ad-hoc design. This design uses a tri-state bus for the read action and complex cells for full/empty detection, and thus cannot be handled easily by standard tools.

2.5 Case Studies for GALS Compatible NoCs

Most of previously published NoCs (such as SPIN [33], Dally's NoC [34], AEthereal [35], Nostrum [36], xPipes [37], CLICHÉ [38], Octagon [39] and Spidergon [40], SoCBUS [41], HERMES [42], aSoC [43], SoCIN [44], BFT [45], BONE [46], and Proteo [47]) are based on synchronous circuit techniques. However, in the literature until now there are some on-chip interconnects which employ fully asynchronous circuits. These include Chain [48], QoS [49], QNoC [50], Nexus [51], ANoC [52], and MANGO [53]. As two case studies I briefly describe MANGO and ANoC architectures as follows.

2.5.1 MANGO

The MANGO (Message-passing Asynchronous Network-on-chip providing Guaranteed services over OCP interfaces) architecture, developed at the Technical University of Denmark, is an asynchronous NoC, targeted for coarse-grained GALS-type SoC [54]. MANGO provides connection-less Best Effort (BE) routing as well as connection-oriented Guaranteed Services (GS). Guaranteed services connections are established by allocating a sequence of Virtual Channels (VCs) through the network. The routers implement VCs as separate physical buffers. A scheduling scheme called ALG (Asynchronous Latency Guarantees), schedules access to the links, allowing latency guarantees to be made.

The router consists of two separate routers: the BE router and the GS router. The BE router implements a source routing scheme. The three MSBs of the packet header indicate one of five output ports. After passing the router, the header is rotated three bits, positioning the header bits for the next hop. With a flit size of 33 bits (of which one is the end-of-packet bit) it is thus possible to make only 10 routing hops.

While the routers themselves are implemented using area efficient bundled-data circuits, the links implement delay-insensitive dual-rail data encoding. This makes global timing robust, because no timing assumptions are necessary between routers. However pipelining is necessary in order to keep performance.

2.5.1.1 Network Adapter

In the MANGO architecture the router has a number of unidirectional ports of which two are local ports. The local ports connect to a module called the Network Adapter. As a network interface controller, the Network Adapter (NA) provides OCP-based standard socket interface, based on the

primitive routing services of the network. The NA implements a Core Interface (CI) at the core side and a Network Interface (NI) at the network side. The GALS architectures are enabled as the NA implements synchronization between the clocked OCP interfaces and the asynchronous network. Its design is a balanced mix of synchronous and asynchronous circuit parts.

Employing a two cascaded flip-flops as a synchronizer on signals requiring synchronization incurs an overhead on system performance (in terms of packet latency and communication throughput). In the NA a two-phase handshake channel is implemented in order to reduce the synchronization overhead. This is converted in the asynchronous domain to a four-phase protocol compliant with the handshaking of MANGO.

2.5.2 ANoC

ANoC, a complete Asynchronous NoC architecture adapted to GALS systems, is proposed and developed by CEA-LETI (Commissariat à l'Energie Atomique – Laboratoire d'Electronique et de Technologie de l'Information) of Grenoble [52]. ANoC uses Virtual Channels (VCs) to provide Quality-of-Service (QoS), and is implemented in Quasi-Delay-Insensitive (QDI) asynchronous logic using four-phase handshake protocol and one-hot (1-of-4) data encoding.

The ANoC communication architecture is composed of nodes, links between nodes, and computation resources. The NoC asynchronous nodes are the basic switching elements of the network. They are responsible to handle the wormhole protocol and arbitrate between any conflicting packets. Each flit is composed of 32-bit data and 2 control bits where the 34^{th} bit encodes the begin-of-packet and the 33^{rd} bit encodes the end-of-packet.

The packet header contains a field path-to-target in order to perform the packet routing from one initiator to a target. The static path between initiator and target resources are programmed and stored in the initiator resources. Then the nodes arbitrate between incoming requests. Each node uses the two LSB bits and shifts the path-to-target field for the following node. The path-to-target field is encoded on 18 bits, which allows crossing at most 9 different nodes in the network topology.

Even if the routing is deterministic, the routing paths between the resources are determined using a dynamic routing algorithm called "odd-even turn model" adaptive routing algorithm [55]. By using a virtual channel priority scheme and an adaptive routing algorithm some packets can be guaranteed through the network. The data paths are separated in real-time and best-effort packets. The real-time packets must not overlap while the best-effort packet may overlap. Real-time constraints are checked at system level by simulating the complete application on the network architecture.

ANoC and its related design methodology have been applied to the design of a prototype chip in a 130 nm STMicroelectronics CMOS technology, called FAUST. The FAUST chip (Flexible Architecture of Unified System for Telecommunication) [56], arranged in a two-dimensional

mesh topology, integrates 20 asynchronous NoC nodes, 23 synchronous units including ARM946 processor core, embedded memories, various IP blocks, reconfigurable data paths engines, and one clock management unit to generate the 23 distinct clock domains. The 20 NoC nodes represent about 15% of the overall area.

2.5.2.1 Network Interface

To integrate any synchronous IP within ANoC architecture a dedicated network interface performs two main tasks: the synchronization between the synchronous and asynchronous logic domains, and providing all facilities to access the NoC communication infrastructure including network routing path programming, network data packetizing, and IP core configuration. In design of such an interface two technical points have been addressed: On-chip GALS interfaces between the synchronous and asynchronous NoC domains, and Off-chip interfaces to communicate off-chip at NoC level [57].

The on-chip GALS interfaces are based on a pre-existing multi-clock synchronous FIFO using gray code [58]. Even if less efficient compared with Chelcea and Nowick FIFO, the aim of this design has been mainly to be adapted as much as possible to standard synchronous tools. Because the timing domains between the synchronous and asynchronous sides are distinct, it is mandatory to resynchronize, for example in the asynchronous-to-synchronous FIFO, the write pointer with the read clock in order to generate the empty information. This synchronization is done by a standard two cascaded flip-flop synchronizer. Using a gray code to encode the read and write pointers guarantees that if a metastability problem occurs in the first flip-flop stage, the empty decision is always correct, but at most generated one clock cycle later. Using a multi-clock domain gray FIFO in an asynchronous environment needs to generate a local clock pulse to the FIFO when a new transfer is required by the asynchronous side, and to handle differently the full and empty flag of the asynchronous side within the FIFO.

The Asynchronous-to-Synchronous gray FIFO is composed of:

- 8 × 34-bit registers
- A write pointer and a write clock which is generated by the asynchronous side of the GALS interface
- A read pointer and a read clock which is actually equivalent to the clock of the synchronous domain
- Write enable, read enable, full, and empty FIFO control signals

And similarly the Synchronous-to-Asynchronous gray FIFO is composed of:

- 8 × 34-bit registers
- A read pointer and a read clock which is generated by the asynchronous side of the GALS interface

- A write pointer and a write clock which is actually equivalent to the clock of the synchronous domain
- Read enable, write enable, empty, and full FIFO control signals

On the asynchronous side of, for example, A-to-S FIFO, since the write clock is generated only when a new asynchronous write is requested, it would be possible to set the full signal after the last write, but it would not be possible to clear the full signal after a read on the synchronous side. As a solution, the full information is regenerated according to the relative phases of the write clock and read clock: the full signal is cleared when full was true, read clock is low (read pointer register value is stable), and the comparison between write and read pointers shows a non-full situation.

Some timing margins must be respected: the read clock and write clock high-pulses must be long enough so that the gray pointer register outputs are stable, as well as the combinational test between the read and write pointers. Additionally, a trickier timing requirement is to guarantee that the input FIFO signals (e.g. write data and write enable) are stable before the write clock pulse is generated. This has been tuned by the clock tree insertion time. As a drawback, a special care must be given to the clock tree synthesis during place and route within the GALS interfaces in order to manage properly the clock tree synthesis and insertion time.

The ANoC off-chip interface shows a dual synchronous/asynchronous NoC port mode. The architecture is built with the on-chip NoC A-to-S and S-to-A interfaces, for the synchronous port, and with the asynchronous Quasi-Delay-Insensitive (QDI) to Bundled Data (BD) protocol converters, for the asynchronous port. In fact internal QDI protocol does not fit off-chip communication, because four-rail is too costly in number of pads, and four-phase is too slow as pad latency occurs four times per data transfer.

2.6 DSPIN

Looking for cost-effectiveness, in continuation of the SPIN micro-network, LIP6 developed a new distributed Network-on-Chip (called DSPIN, i.e. Distributed Scalable Packet-switching Integrated Network). DSPIN has been designed generally to support large-scale clusterized shared memory MP-SoCs. This new architecture is an answer to the problems encountered during the physical implementation of SPIN micro-network architecture [59], the first published attempt of NoC technology design. Among these faced difficulties are: implementation complexity of adaptive routing algorithms and eventual lack of in-order-delivery, inflexibility of fat-tree topologies, inappropriate synthesizability of hard macro-cells, and GALS incompatibility of a centralized NoC architectures.

Different from the SPIN architecture in which the network topology was a 4-ary n-dimensional fat-tree and the routers were centralized as a hard macro-cell, the DSPIN network topology is a two-dimensional mesh and the routers are physically distributed within the clusters. The size and

shape of the clusters have no constraints, but the mesh (grid-based) topology has to be respected. Refer to Figure 2.

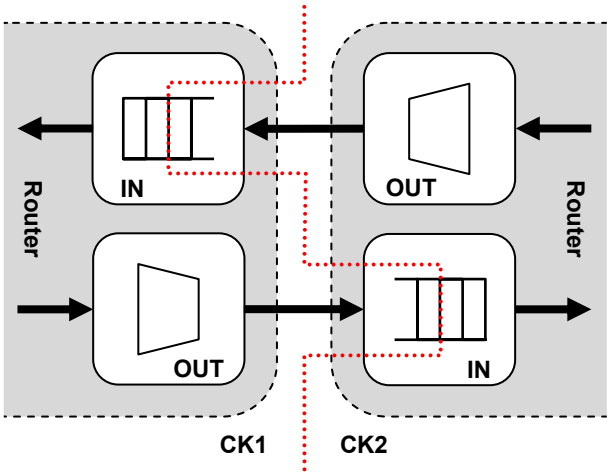


Figure 9. Clock Boundary between two adjacent DSPIN Routers

The first and most important projection of the DSPIN architecture was to respect the GALS paradigm. DSPIN exploits the multi-synchronous approaches to solve the clock boundary issues. In order to provide point-to-point asynchronous (or mesochronous) links the connections between routers are implemented as special bisynchronous (or mesochronous) FIFOs. Refer to Figure 5 and Figure 6. As DSPIN uses the storage strategy of input buffering, the special FIFOs are instantiated inside the input ports (IN). Figure 9 displays the clock boundary between two adjacent routers in the DSPIN architecture.

DSPIN is a wormhole packet-switching network. Packets are divided into flits (flow control units) which consist of only one phit (physical unit). The smallest physical unit handled by the routers, a phit (or a flit) in DSPIN is a single 32-bit data word accompanied by 2 control bits. Figure 10 shows the format of DSPIN's packet. The first flit of a packet is the packet header. It includes the destination cluster address, defined by absolute coordinates of (Y, X). When an input port of the router receives the header of a packet, where the flag indicating the beginning of the packet (BOP) is 1, the destination address field (the eight LSB bits) is analyzed and the flit is forwarded to the corresponding output port. The rest of the packet is also forwarded to the same port until the trailer of packet, where the flag marking the end of the packet (EOP) is 1. When there are simultaneous requests for the same output port, the Round-Robin algorithm will be used to schedule the requests in order to avoid starvation.

According to the fact that DSPIN is a direct network, each router in addition to the north, south, east, and west neighbor routers, is connected to the local subsystem. To route packets between these five different sides, from input ports to output ports, DSPIN uses the distributed X-First algorithm guaranteeing the in-order-delivery property for the network. With this algorithm,

packets are first routed on the X direction and then on the Y direction. As a result, there is no need to connect the north and south input ports to the west and east output ports (see Figure 12), decreasing the hardware complexity.

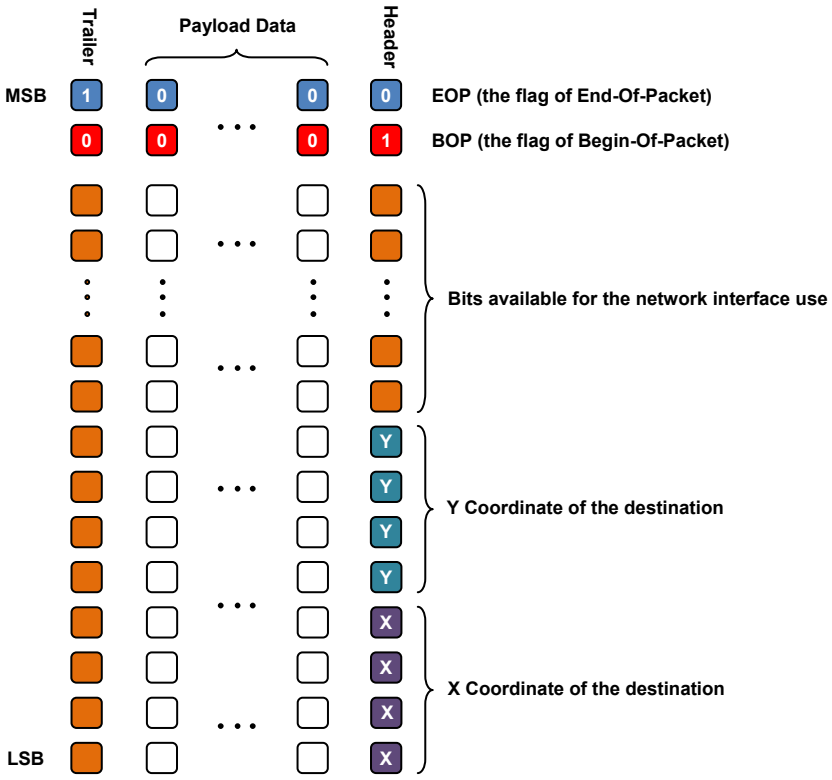


Figure 10. DSPIN's Packet format

The interconnection between the input and output ports in DSPIN is composed of a set of four different signals, *Out_Req*, *Data*, *Put*, and *Get*, as illustrated in Figure 11. *Out-Req*, connecting the routing algorithm module (implemented in the input port) to the scheduling algorithm module (implemented in the output port), indicates that which output port must be allocated to the incoming packet. With regards to the Round-Robin algorithm, the output port then selects the corresponding input. Afterward, the sequence of flits appeared on *Data* moves from the input port to the output port, whenever *Get* and *Put* are both asserted. In fact *Put* is the ROK (not empty) signal of the input FIFO and so it declares that there is a flit to be transferred. On the other hand, as *Get* follows the WOK (not full) signal of the next router's FIFO, it states that the flit can be accepted by the next stage. Since at the same time for each input port there is only one *Get* following states of the next hops, a simple OR Gate can be used to provide the Read signal of the input port's FIFO buffer.

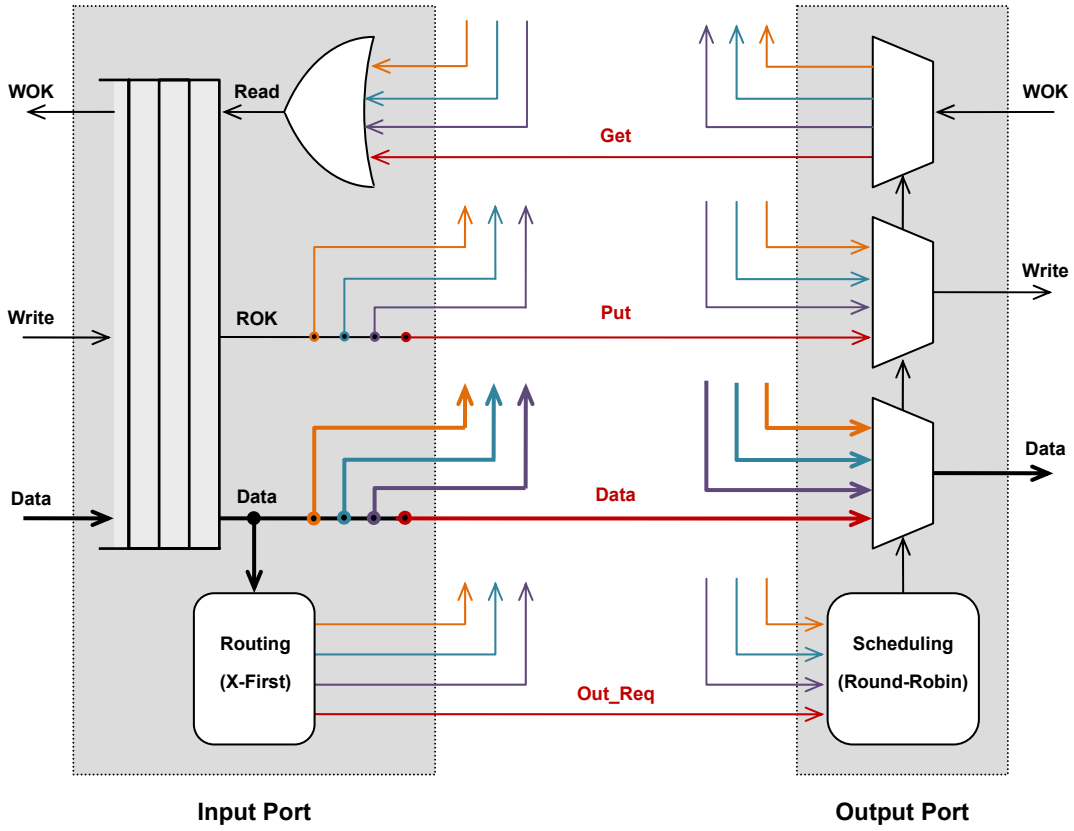


Figure 11. General Architecture of DSPIN's Input / Output Ports

2.6.1 Long Wire Issue

As emphasized before, in deep submicron processes the largest part of the delays is related to the long wires. Seeing that the place and route tools have difficulties to cope with long wires, in multi-million gates SoCs, the timing closure can become a nightmare [60]. DSPIN architecture attempts to solve this problem by partitioning the SoC into isolated clusters. This allows performing physical synthesis and timing closure analysis for each cluster independently, without any time constraints between different clusters.

As shown in Figure 12, the DSPIN router is not a centralized macro-cell. The router is split in five separated modules (North, South, East, West, and Local) that are physically distributed on the cluster borders. This feature allows us to distinguish two classes of wires:

- *Intra-Cluster Wires*, which connect the modules of the same router (black arrows)
- *Inter-Cluster Wires*, which connect the modules of two adjacent routers (white wide arrows)

As shown, the intra-cluster wires are spread out along the cluster. Though the length of these wires is bounded by the physical area of a given synchronous domain, they are long (even the longest) wires and likely require to be latched and use repeaters. On the contrary, as in most of the times the corresponding modules of each two neighbor routers, for example the east module in cluster (Y, X) and the west module in cluster (Y, X+1), could be very close to each other, the inter-cluster wires are usually short wires. Once the clusters are individually synthesized, the pre-placed and pre-routed clusters can be placed one beside the other and be connected one to the other, in a plug-and-play fashion.

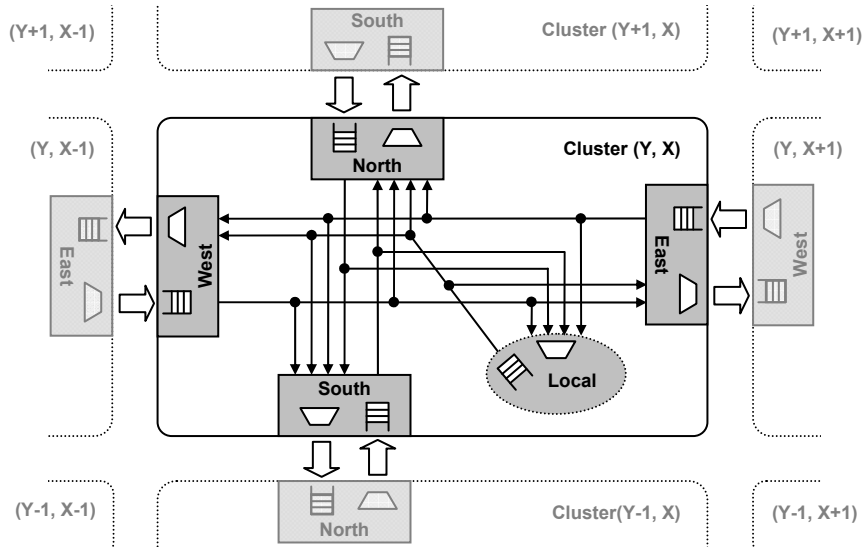


Figure 12. DSPIN's Router Architecture

2.7 Summary

The principal ideas and the state-of-the-art methods regarding GALS compatibility and synchronization failure have been introduced. Networks-on-Chip offer a structured approach to design a multi-clocked system, and as relevant approaches multi-synchronous and fully asynchronous NoC paradigms have been described. While in a packet path a multi-synchronous approach represents the possibility of synchronization failure at each clock boundary between each two neighbor routers, an asynchronous NoC limits the synchronization failure only at the network interfaces, where the synchronous data has to enter into the asynchronous network and the asynchronous data into the synchronous subsystems.

As robust interfaces, in a multi-synchronous NoC a bisynchronous FIFO (in which the consumer and producer sides use independently their own clock signals) can be instantiated at clock boundaries, and in an asynchronous NoC two special FIFOs: synchronous-to-asynchronous

FIFO (which has a synchronous data producer and an asynchronous data consumer) and asynchronous-to-synchronous FIFO (in which the data producer is asynchronous and the data consumer is synchronous).

Finally, I indicated some existing micro-network constructing GALS architectures, including DSPIN developed in the LIP6 laboratory. DSPIN is a distributed multi-synchronous NoC that has been designed generally to support shared-memory Multi-Processor Systems-on-Chip. As an important feature, DSPIN addresses the crucial issue of the global long wires. In deep submicron technologies the largest part of the delays is related to the long wires.

Chapter 3

Asynchronous Implementation

The general architecture of DSPIN, a multi-synchronous NoC well-suited to the GALS paradigm, was introduced in Chapter 2. An asynchronous implementation, called ASPIN (Asynchronous SPIN), is presented in this chapter. As said, the NoCs using asynchronous circuits seem the most compatible techniques to support the GALS paradigm. As ASPIN's router has a fully asynchronous design, ASPIN uses two special async-to-sync and sync-to-async FIFOs in order to provide synchronous compliant interfaces at each local port. The router implementation is described in the present chapter and the design of AS_FIFO and SA_FIFO will be considered in the next chapter.

3.1 General Architecture

As DSPIN, ASPIN has been designed to support clusterized shared-memory MP-SoC architectures. The network topology is a two-dimensional mesh and the routers are physically distributed within clusters. Each router is connected to the north, south, east, and west neighbor routers, as well as to the local subsystem. To route packets from input ports to output ports of these five different sides, ASPIN uses the distributed X-First routing algorithm. ASPIN is a wormhole packet-switching network and uses the storage strategy of input buffering. When there are simultaneous requests for the same output ports, a Round-Robin algorithm will be used to schedule the requests in order to avoid starvation.

3.1.1 Long Wires Issue

As a predominant physical factor in deep submicron technologies the delays incurred by the long wires can become the limiting factor for the network throughput. As DSPIN, ASPIN's routers are

split in five separated modules of North, South, East, West, and Local. Those modules can be physically distributed over the cluster area, in order to balance the length of the long wires.

However, in large clusters the long wires need to be pipelined. In ASPIN architecture an Intermediate Pipeline Stage (IPS) is instantiated as a relay station inside the router between the input and output modules, dividing the length of the intra-cluster wires in two. Figure 13 demonstrates ASPIN's router architecture. Supplementary pipeline stages can be instantiated on the inter-cluster wires (wide white arrows).

Depending on the routing, long wires can have various lengths for different bits of a flit, and thus the resulting skew is not predictable. In order to guarantee delay insensitivity, the long wires in ASPIN use double-rail data encoding and the communication employs four-phase handshake protocol. Dual-rail data encoding is a delay-insensitive code since the validity information is carried along every bit in the data word, and thereby the receiver is enable to unambiguously detect the word completion, regardless of delays.

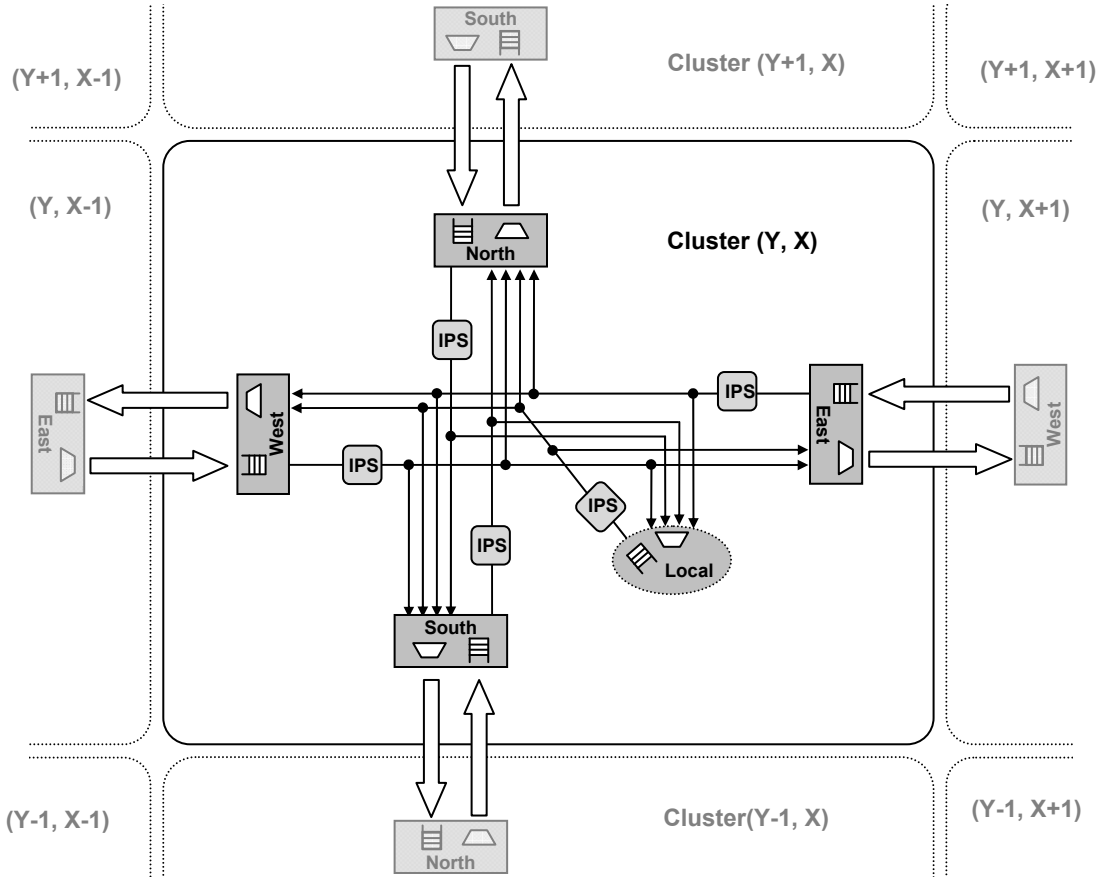


Figure 13. ASPIN's Router Architecture

3.1.2 Packet Structure

In ASPIN packets are divided into flits (flow control units) and each flit consists of only one phit (physical unit). A phit is a single N-bit data word accompanied by one control bit indicating the end of the packet. Figure 14 shows the structure of ASPIN's packet. The first flit of a packet is the packet header including the destination address. The address is defined by absolute coordinates (Y, X). When the packet header is entered to an input port of the router, the destination address field (the eight LSB bits) is analyzed and the flit is forwarded to the corresponding output port. The rest of the packet is also forwarded until the packet trailer, where the flag marking the end of the packet (EOP) is 1.

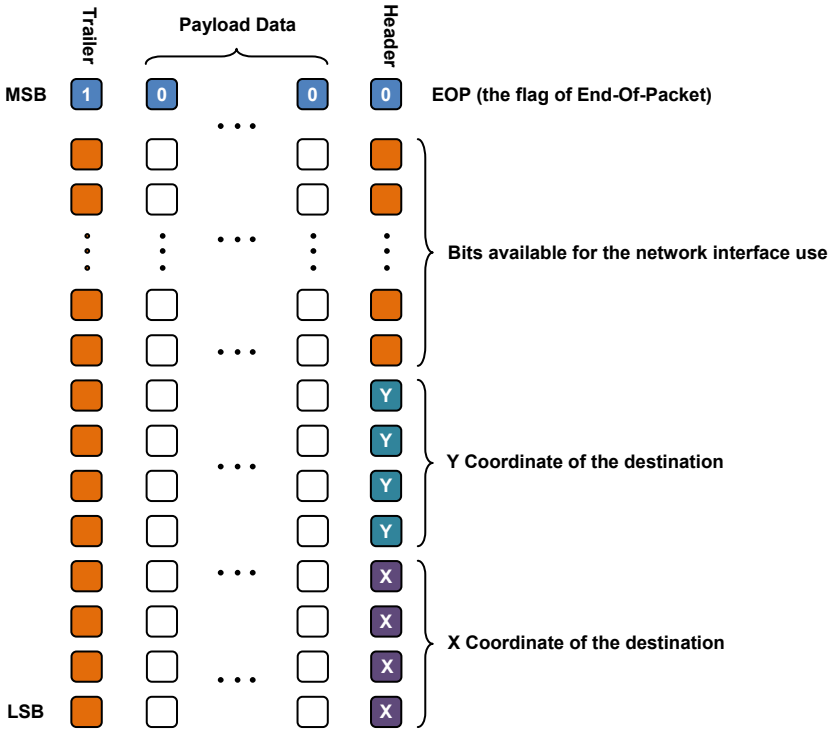


Figure 14. ASPIN's Packet Format

3.1.3 Block Diagram

As shown, an ASPIN router consists of three different kinds of blocks: Input Ports, Output Ports, and Intermediate Pipeline Stages. The interconnection between ASPIN's modules is depicted in Figure 15. Seeing that the communication over the long wires uses double-rail four-phase handshake protocol, signals transferring data between separated blocks are composed of two types of signals: *Data* and *ACK*.

Due to the fact that to represent each bit, *Data* uses two wires, a major concern may be the significant overhead in silicon area. In a router the storage places (FIFOs) are the most important area occupant. In order to minimize the silicon area overhead, the asynchronous FIFO (instantiated in the input port) is designed as an optimized hard-block (in which the designer can control the signal timings) and uses single-rail protocol (bundled-data encoding). To be compliant with dual-rail data encoding used by the rest of the router, two protocol converters, dual-rail to single-rail and single-rail to dual-rail, are implemented respectively before and after the asynchronous bundled-data FIFO.

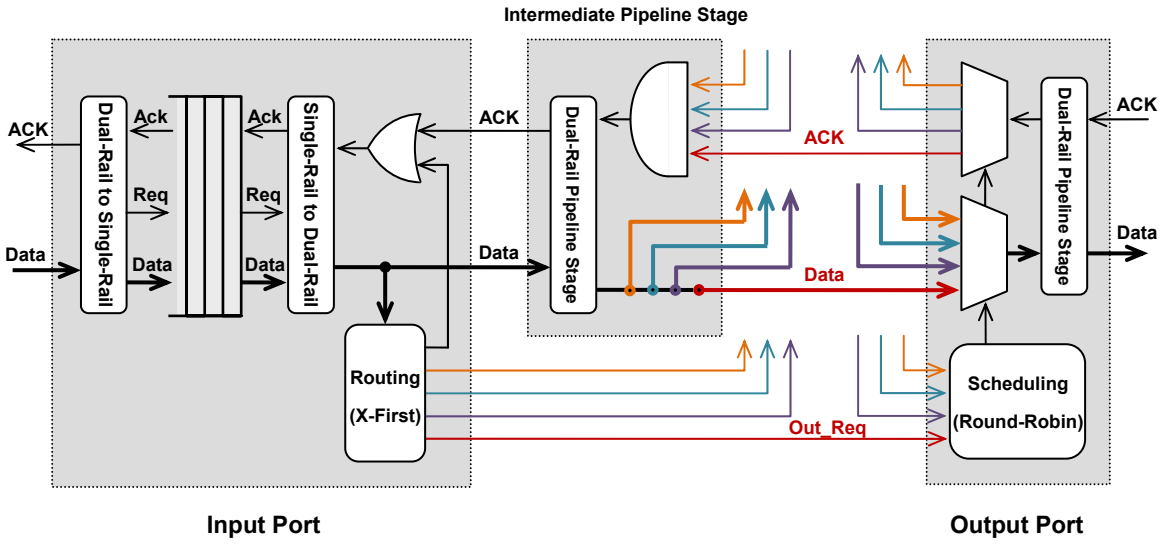


Figure 15. General Architecture of ASPIN's Modules

The *Out-Req* signals connecting the routing algorithm module (implemented in the input port) to the scheduling algorithm module (implemented in the output port) are the allocation request. With regard to the Round-Robin algorithm, the output port then selects the corresponding channel to establish the communication between the intermediate and output pipeline stages.

A simple AND gate can provide the acknowledge signal of the intermediate pipeline stage, because when a channel is unselected the corresponding *ACK* remains at high value. In fact the communication handshake sequence of the last flit (the packet trailer) does not finish and the output port does not produce the last *ACK*-. Thanks to that it resembles the output port has not yet ended the transmission of the last flit, and thus the intermediate pipeline stage does not take the next flit which is the header of the next packet and must be retained in the input port until the routing algorithm unit makes its decision and asserts the relative *Out-Req*.

When due to a transient error on the packet header there is a failure on the destination address it is possible that a packet aims to go to an invalid direction and thereby blocks the port (for example a packet incoming from the north intends to go to the east). Recall that in order to

optimize the design complexity, as in DSPIN, in ASPIN a packet is not permitted to return to the same port and there is not any physical connection from the vertical (i.e. north and south) input ports to the horizontal (i.e. east and west) output ports, as not required by the X-First routing algorithm. To resolve this blocking situation, the routing algorithm module of ASPIN determines the faulty packet and generates a virtual *ACK* (which is combined with *ACK* going out from the intermediate pipeline stage by an OR gate) to gather the flits from the path.

3.2 Input Port

The detailed architecture of ASPIN's input port is shown in Figure 16. As can be seen, an incoming packet is led to be retained in the storage unit consisting of three parts: dual-rail to single-rail converter, single-rail FIFO, and single-rail to dual-rail converter. When a packet header appears on the output of the single-rail to dual-rail converter, a comparison module compares the packet destination addresses (Y and X) with the local cluster addresses (Y0 and X0), and then for each coordinate asserts one of three output signals signifying the three possibilities: Y0>Y, Y0=Y, and Y0<Y, and, X0>X, X0=X, and X0<X.

Except for the header of packet, address comparison introduces an invalid operation. By the use of an upper asymmetric C-element the value high of *P* does not let outputs of *Address Comparator* be propagated. The *Controller* unit holds *P* at logic one during packet transmission. Waiting for beginning of a new packet, at the end of each packet transmission *Controller* sets *P* to zero. Before arriving *P* down, *Controller* should be assured that the *Address Comparator*'s outputs are zero and their previous values will not incorrectly be propagated. *nZero* produced by an OR gate is provided for this reason. Considering dual-rail data encoding, at data idle state, when the two wires of each address bit are zero, all six outputs of *Address Comparator* are set to zero.

When a new packet arrives, depending on the address comparison one of five signals indicating destination (i.e. *EAST*, *WEST*, *NORTH*, *SOUTH*, or *LOCAL*) will be asserted. As required by *Controller*, from these five signals a simple OR gate generates the *BOP* signal which informs the packet beginning.

The design example displayed in Figure 16 is the south input port. According to the X-First routing algorithm, a packet incoming from the south requires to be headed only to the north or local subsystem. Out_Req_N and Out_Req_L indicate request for the intended direction respectively. Nevertheless, if incorrectly another direction (i.e. east, west, or south) is demanded (for example due to transient errors on bits of the destination address field in the packet header), the packet will be considered as corrupted and must be removed. Using a state-holding element (C-element), *Err* remains high until the end of the packet. At the end of the packet *P* goes down and sets *Err* to zero. Note that the incorrect direction request (*EAST*, *WEST*, or *SOUTH*) is already set down.

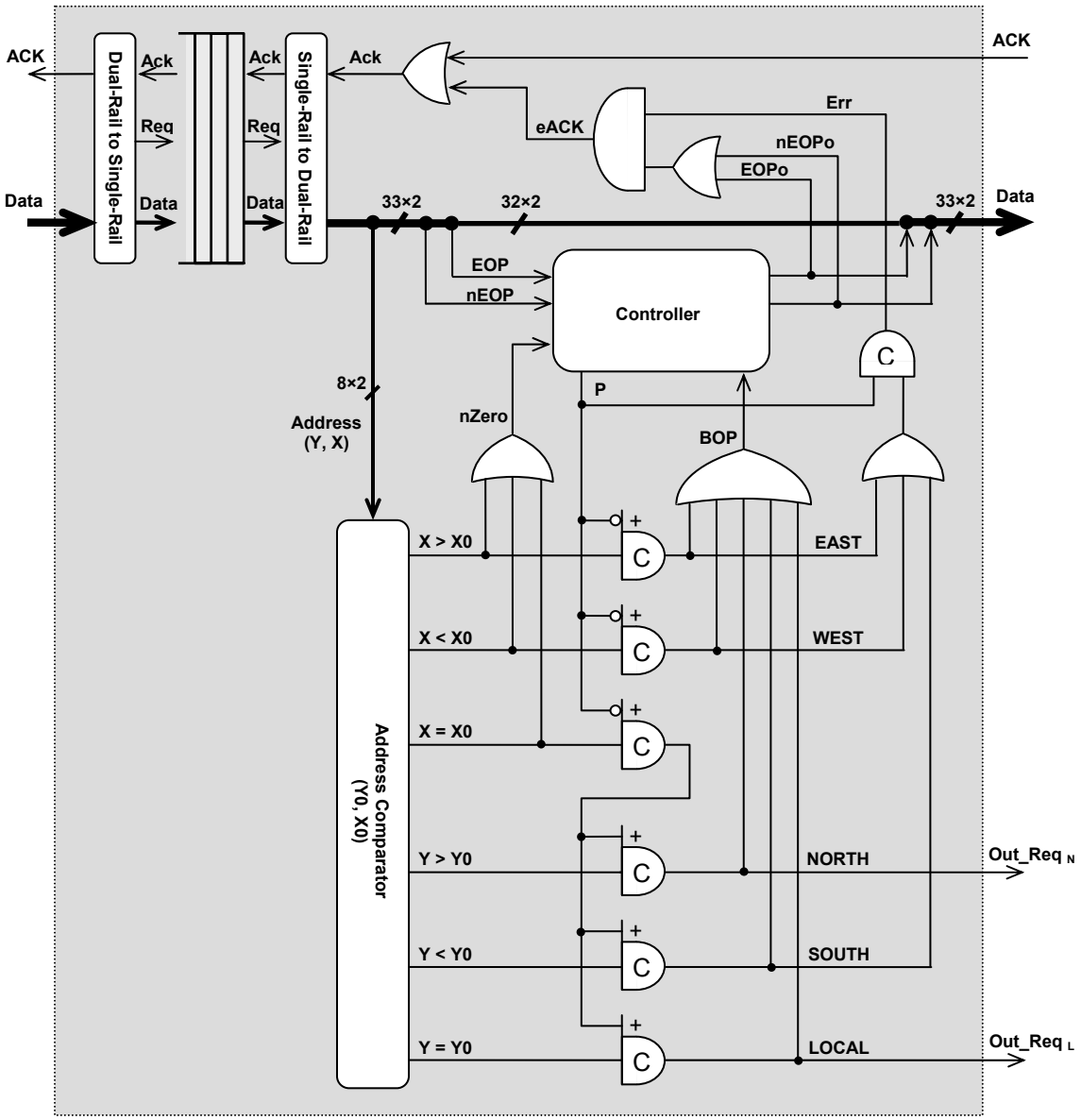


Figure 16. Input Port Architecture (South Port)

To control the flow of packet transmission the two wires of the bit indicating the end of packet (i.e. *EOP* and *nEOP*) are separated from data signals and enter *Controller*. They will be reproduced as two new signals (*EOPo* and *nEOPo*) and will be merged to other data signals when leaving *Controller*. Recall that in dual-rail four-phase handshake protocol packet transmission does not progress until all data bits are completed.

The assertion of one of two signals of *EOPo* and *nEOPo* means *Controller* has given the permission for transferring the current flit. As a consequence, the combination with *Err* via an AND gate can generate a local ACK signal (*eAck*) to complete the handshake sequence, removing flits of the faulty packet.

3.2.1 Controller

As explained, in order to allow the progression of packet transmission and to confirm the routing algorithm decision, the *Controller* generates two different indicators: a signal indicating the presence of a packet being transmitted (*P*), and flags indicating the end (or not) of the packet transmission (*EOPo* or *nEOPo*). Accordingly, the *Controller* needs three different input information signals: an indicator indicating the beginning of a packet (*BOP*), a signal signaling the presence of a non-zero value on *Address Comparator*'s outputs (*nZero*), and flags signifying the end (or not) of the incoming packet (*EOP* or *nEOP*). The circuit implementation of *Controller* is shown in Figure 17.a and its detailed behavior, specified as a Signal Transition Graph (STG), is depicted in Figure 17.b. In this STG the dashed-lines represent transitions outside the *Controller* and the solid-lines represent the inside transitions.

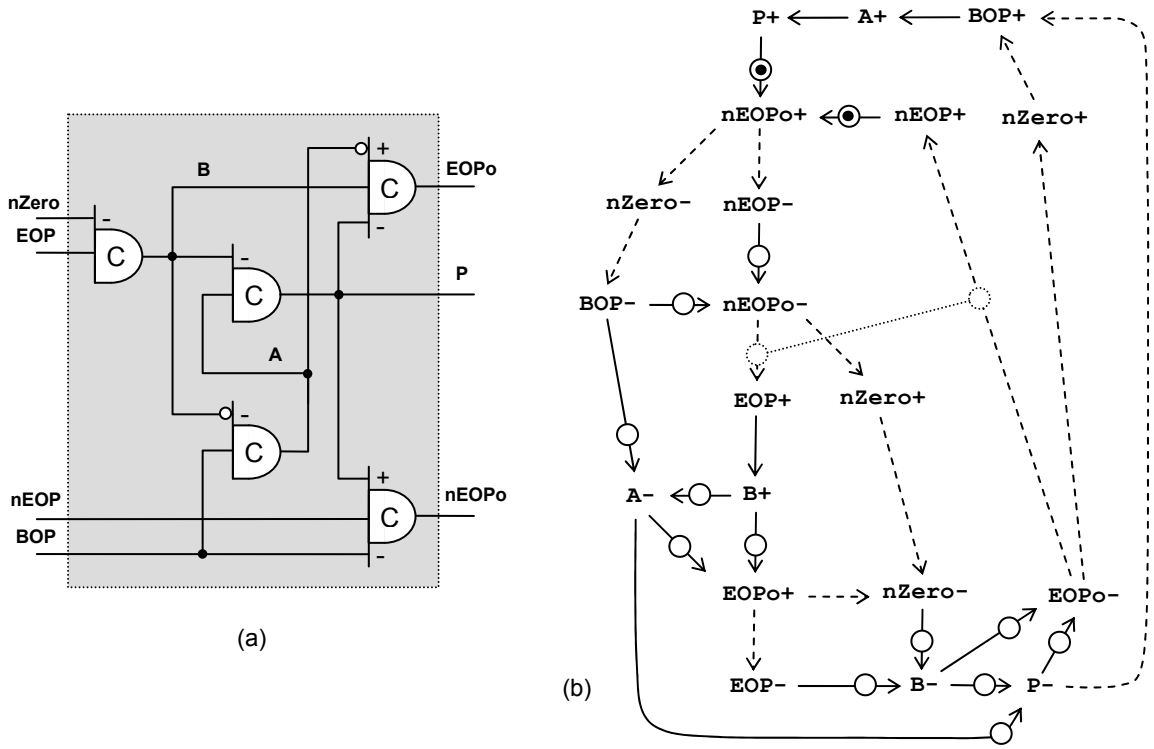


Figure 17. Input Port's Controller (a) Circuit Implementation and (b) STG

Briefly, when *BOP* is high *Controller* sets *P* to high and waits until the end of the packet. That is to say if *nEOP* is set to high, *nEOPo* will be set to high. *nEOPo*+ permits the continuation of the flit transferring and consequently causes *nEOP*-. Respecting delay insensitivity, if *BOP* is already set down, *nEOP*- causes *nEOPo*-. By the end of each flit transmission (*nEOPo*-) a new one can be started. The dotted-line represents this possibility that after *nEOPo*- a new *nEOP*+ can be happened.

Otherwise, if the next flit is the packet trailer (i.e. after *nEOPo*-, instead of *nEOP*+, *EOP*+ is produced), *Controller* asserts *EOPo* and waits for *EOP*-. Afterward, when *EOP*- (signifying the request for end of the packet transferring) is occurred, and if *nZero* is set down, *P* and then *EOPo* go to zero, waiting for a new packet.

3.2.2 Address Comparator

In ASPIN the eight Least Significant Bits (LSB) of the packet header contain the destination address being composed of two absolute coordinates of Y and X. The responsibility of *Address Comparator* is to independently compare Y with Y0 (y-coordinate of the local cluster) and X with X0 (x-coordinate of the local cluster). Each bit of X and Y will be separately compared with the corresponding bit of X0 and Y0.

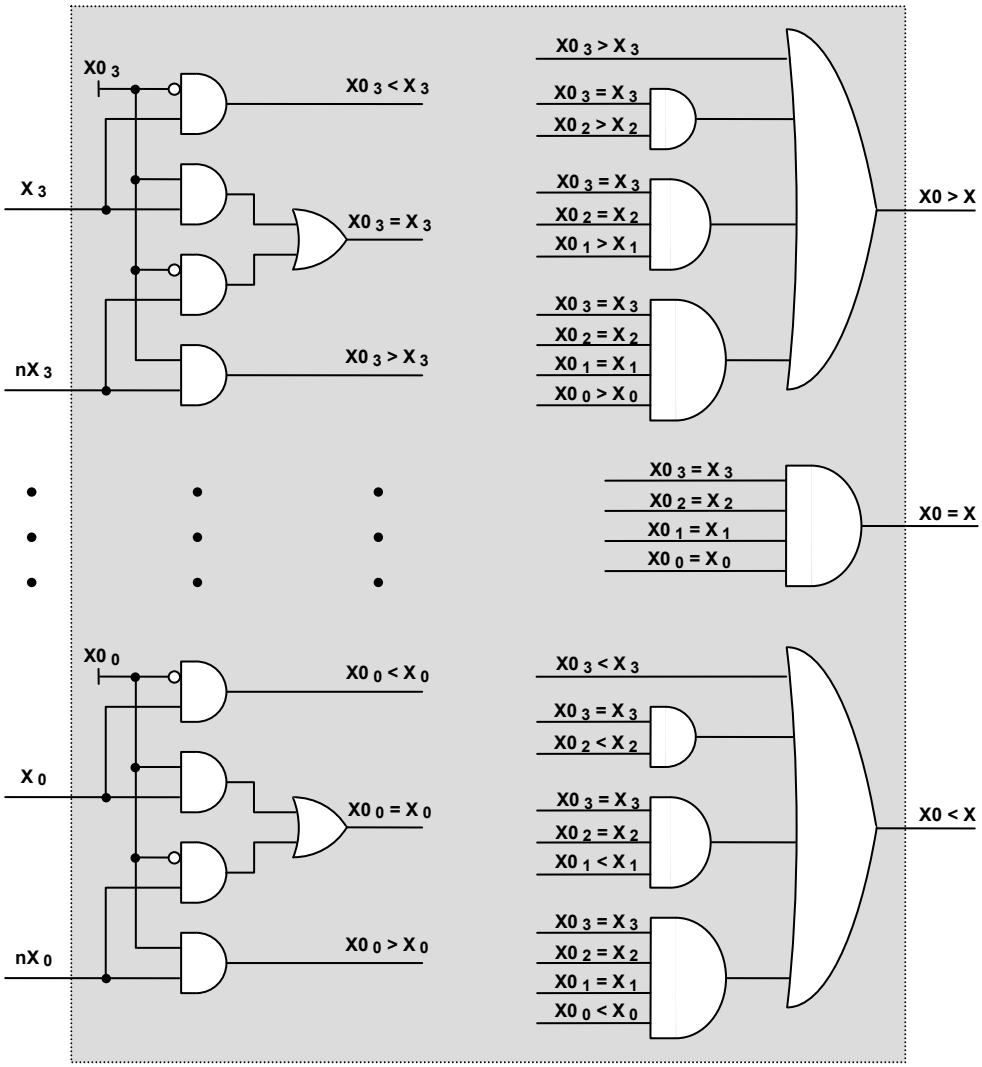


Figure 18. X Comparison in Address Comparator

A circuit fragment of *Address Comparator*'s internal architecture, corresponding to the X comparison, is shown in Figure 18. As ASPIN uses dual-rail data encoding, for each bit i of X, signal X_i represents the value one and nX_i signifies the value zero. So, if nX_i and $X0_i$ both are equal to one, it means $X0_i$ is greater than X_i. On the contrary, when X_i is one and $X0_i$ is zero, a signal indicating that $X0_i$ is less than X_i must be asserted. The two other cases (i.e. when X_i and $X0_i$ both are one and when nX_i is one but $X0_i$ is zero) represent the equality of X_i and $X0_i$.

Note that idle state (where both X_i and nX_i are equal to zero) all signals indicating $X0_i<X_i$, $X0_i>X_i$, and $X0_i=X_i$ are set to zero. As a consequence, in idle state all *Address Comparator*'s outputs will be set to zero, respecting the four-phase protocol principles.

3.3 Output Port

The main duty of an output port controller is to allocate the output port to one of the requesting input ports. The allocation policy must be starvation-free. Accordingly, ASPIN employs the Round-Robin algorithm. Due to the X-First routing algorithm the output ports of the west and east are connected to two input channels, while those of the north, south, and local are connected to four.

Figure 19.a shows the circuit implementation of an output port (the local port) with four input channels. For each channel *i*, the behavior of the output port can be specified as a Signal Transition Graph depicted in Figure 19.b. The dashed-lines are transitions outside the output port and the solid-lines are the inside transitions.

When the input channel *i* demands to communicate, Out_Req_i becomes high. This request will be accepted to be scheduled if the internal signal A_i is high, and then as a result Req_i will be set high. In fact A_i is an internal state accepting the allocation request. Req_i enters to the allocation scheduler performing a Round-Robin strategy. When it is the turn of channel *i* to communicate, the scheduler sets $Aloc_i$ to one. In Figure 19.b the dashed-dotted-dotted-line between Req_i+ and $Aloc_i$+ signifies that there may be some other transitions, as *Round-Robin Scheduler* may allocate the output port to some other channels before selecting channel *i*.

The value one of $Aloc_i$ lets ACK_i (which in the idle state is held at one) goes down and follows *ACKo*'s values. *ACKo i*s the acknowledge signal of the output pipeline stage. Additionally, $Aloc_i$, as a multiplexer select signal, allows $Data_i$ to be connected to the pipeline stage. Therefore, all packet flits coming from channel *i* can be transferred to the pipeline stage and the corresponding handshake sequences can be completed. The dotted-lines in the STG indicate possible transitions of *ACKo* and ACK_i during packet transmission.

As can be seen in Figure 19.a, EOP_i (the positive wire of bit indicating the end of packet) is separated from other data signals and combined with $Aloc_i$ via a C-element to generate $EOPO_i$. All $EOPO_i$ merge together and join with the multiplexer outputs to complete data wires. This

allows controlling the flow of the packet trailer. In fact when $Aloc_i$ is one, EOP_i+ causes $EOPO_i$+ and flit transmission continues, but EOP_i- cannot drive $EOPO_i$-.

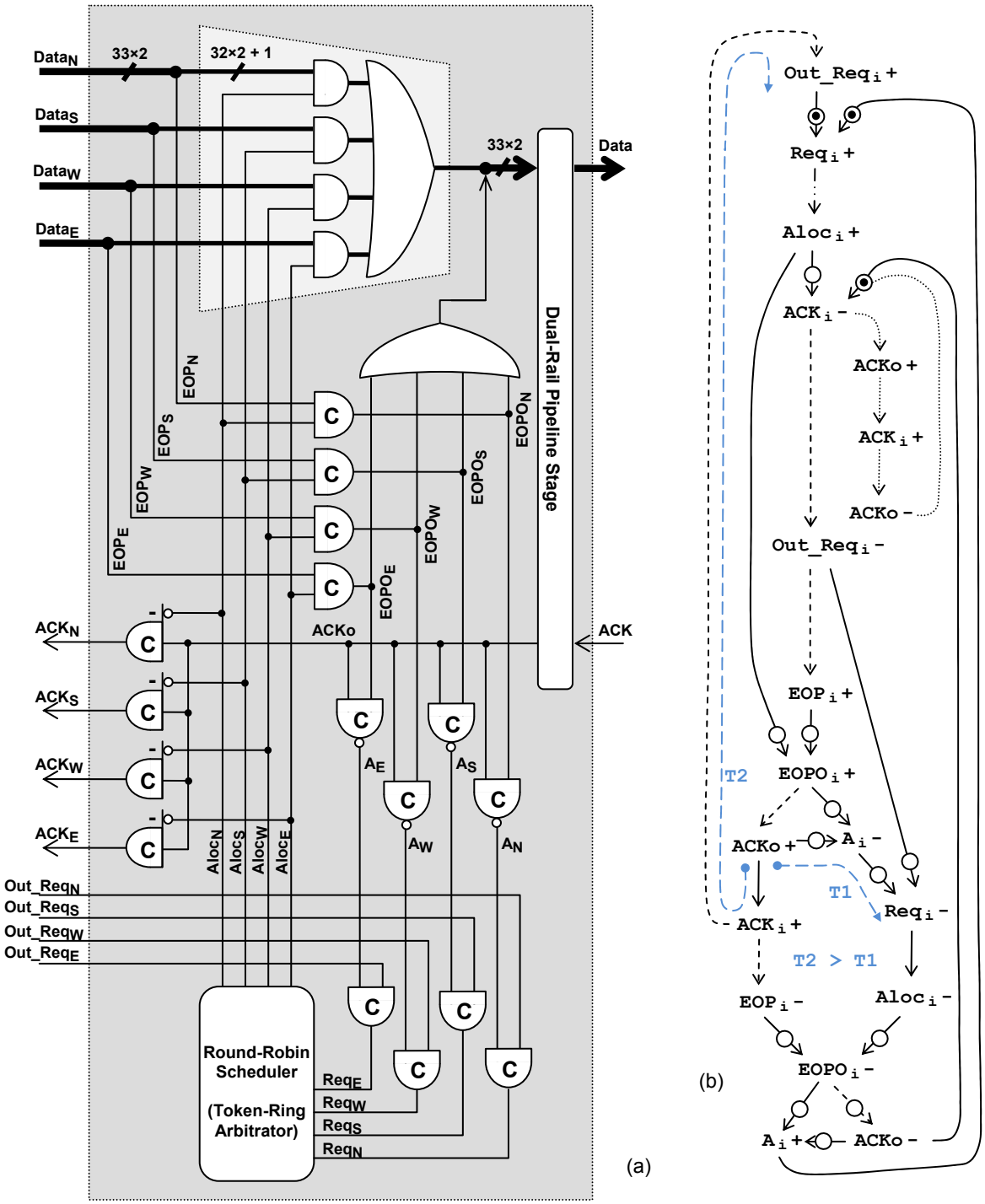


Figure 19. Output Port (a) Circuit Implementation and (b) STG

As said before, in ASPIN the Intermediate Pipeline Stages (IPS) of the router suppose that the transmission of the last flit of each packet is not completed. When ACK_i goes to one for the last flit, it has not to return to zero and must remains at high. After $EOPO_i$+ and when ACKo+ happens, A_i and consequently Req_i will be set to zero (note that at this time Out_Req_i has already changed to zero). When Req_i- happens the scheduler sets $Aloc_i$ to zero and thereby prevents the occurrence of ACK_i-, even if $EOPO_i$- and as a result *ACKo*- happen. The value low of A_i avoids an eventual new Req_i+ and $Aloc_i$+, until $Aloc_i$- is seen and $EOPO_i$- is occurred.

Here there is a timing constraint. *ACKo*+ causes A_i-, and as a consequence Req_i- must be happened. Req_i- needs that Out_Req_i remains at zero. Now the problem is: at the same time, due to *ACKo*+, ACK_i+ occurs and afterward there may be an eventual Out_Req_i+. Furthermore, Out_Req_i+ causes Req_i+ if A_i is still high. As a conclusion, a new Out_Req_i+ can happen after and only after A_i- and Req_i- occurred. Referring to the ASPIN general architecture (Figure 15) and the input port design (Figure 16), the delay between *ACKo*+ and Out_Req_i+ (*T2*) involves the propagation delays of tens of logical gates, plus the propagation delays of long wires traversing from one side to the opposite side of the cluster. While on the other hand, the delay between *ACKo*+ and Req_i- (*T1*) is the propagation delays of two logical gates. Clearly, this timing constraint that *T2* must be greater that *T1*, does not introduce any bother for the functionality of the circuit.

3.3.1 Token-Ring Arbiter as Round-Robin Scheduler

As a Round-Robin scheduler, we implement a Token-Ring arbiter. The design of an asynchronous Token-Ring Arbiter, firstly presented in [61], is displayed in Figure 20. It consists of several *Token Arbiter* arranged in a ring topology. Each Token Arbiter corresponds to a request (Req_i) and for each request there is a grant ($Aloc_i$).

When a request arrives to a Token Arbiter, which is not the Token owner, the request will be transferred to the next Token Arbiter on the ring. And if the next Token Arbiter also does not have the Token, the request will be propagated to the next in a same manner. On the other side, when a request, whether from the external (via the input port *R*) or from the previous Token Arbiter (via the input port *Ri*), enters to a Token Arbiter which has the Token, the corresponding acknowledge signal (*A* or *Ai*) will be asserted. And when a Token Arbiter which has launched a request, receives the acknowledge, depending on the original request (coming from the external or internal), sets *A* or *Ai* to one.

When a Token Arbiter sets its external acknowledge signal (*A*) to one, it means that the Token is moved to this place. In the circuit implementation, an RS flip-flop (as a state-holding element) is used to keep the Token. The flip-flop will be reset if the internal acknowledge (*Ai*) is asserted, and will be set when the assertion of *A* means the acceptance of the external request.

The arbitration between two or more requests that eventually arrive at the same time is the responsibility of a Token-Ring Arbiter. In fact if there are several simultaneous requests, the nearest request to the Token will be acknowledged. However it must be ensured that the arbitration between the internal and external requests is mutually exclusive. Accordingly, the Token Arbiter exploits a handshake Arbiter which uses MUTEX. The MUTEX ensures that even if its request signals *R0* and *R1* are set simultaneously, signals G0 and G1 are mutually exclusive. Following the MUTEX there are two AND gates whose purpose is to ensure that the handshakes on channel 0 and channel 1 are mutually exclusive, as for example the output request of channel 0 (*I0*) can only go high if *A1* is low. In this way, if handshaking is in progress along one channel, it blocks handshaking on the other channel.

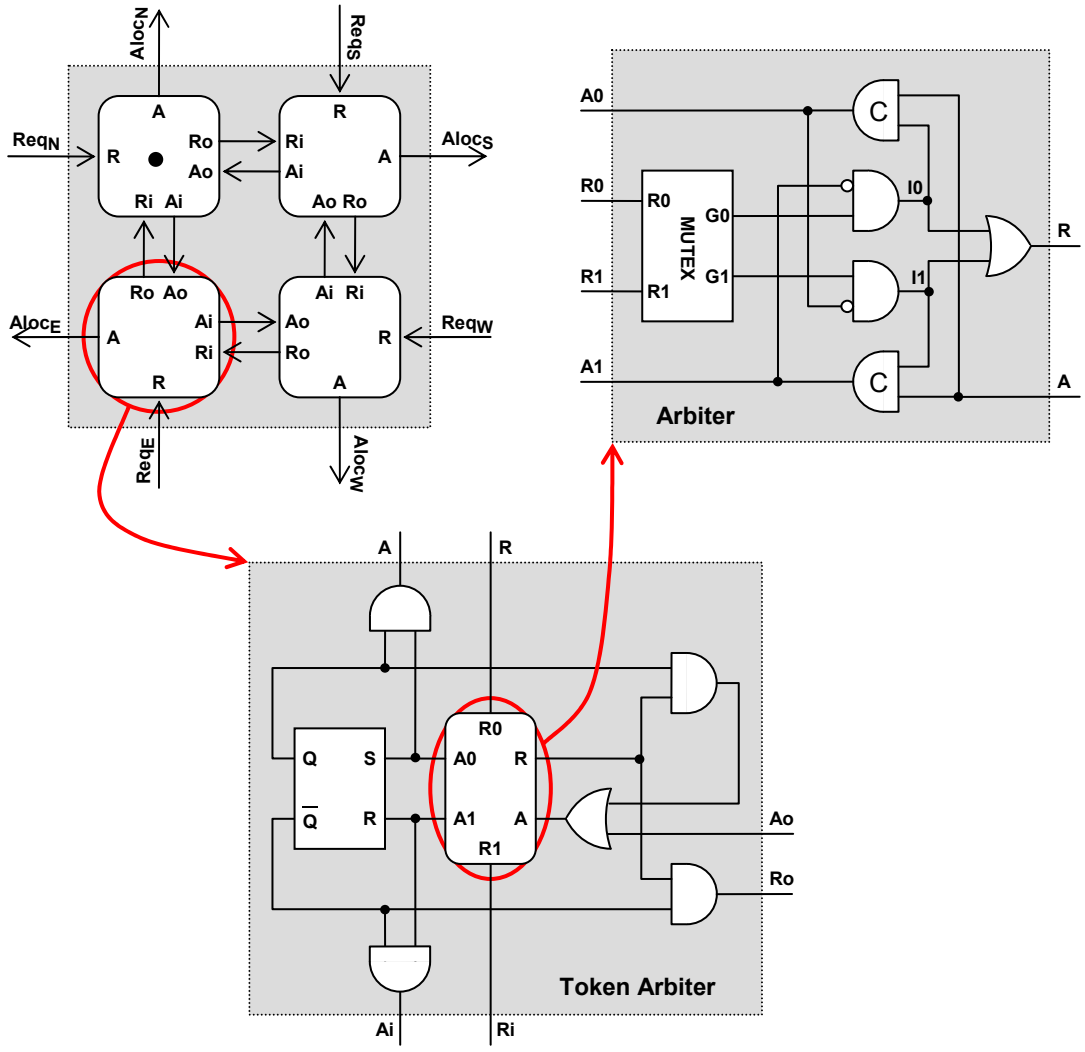


Figure 20. Token-Ring Arbiter as a Round-Robin Scheduler

Requested by a starvation-free scheduling algorithm, a Token-Ring Arbiter has a bounded blocking time. Figure 21 shows a worst-case example. Assume that there are concurrent requests at the inputs of a four-place Token-Ring Arbiter, and imagine that Req_E lose the race with the internal request granted for another request (Req_W). The granted request propagates to the next stage and in the worst case it can be blocked before the two other requests (Req_N and Req_S). Afterwards, when Req_W is acknowledged, the MUTEX immediately grants Req_E, and thereby the Token is not allowed anymore to pass over this place. The granted request for Req_E will propagate to the next stage, where it can fail to keep the possession of the Token at most for two other times.

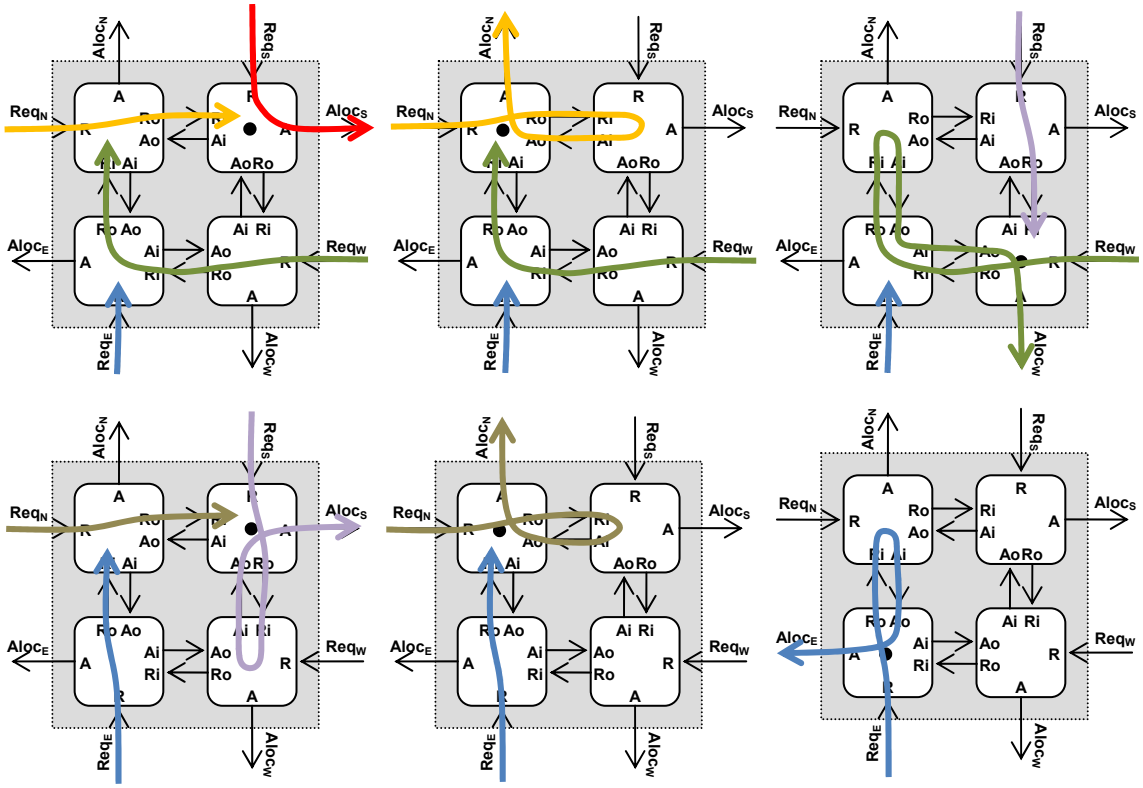


Figure 21. The Worst-Case Blocking Time in a Four-Place Token-Ring Arbiter

3.4 Asynchronous FIFO

Figure 22 displays the block diagram of the asynchronous FIFO (AA_FIFO) used in ASPIN's routers. In the producer side there is one asynchronous demultiplexer and in the consumer side there is one asynchronous multiplexer. The input data is demultiplexed to the FIFO stages, and then they are multiplexed on the FIFO output. In both sides there is an asynchronous controller (named *Domino Controller*) which after finishing the asynchronous event of each side, in a cyclic way selects the next stage to communicate.

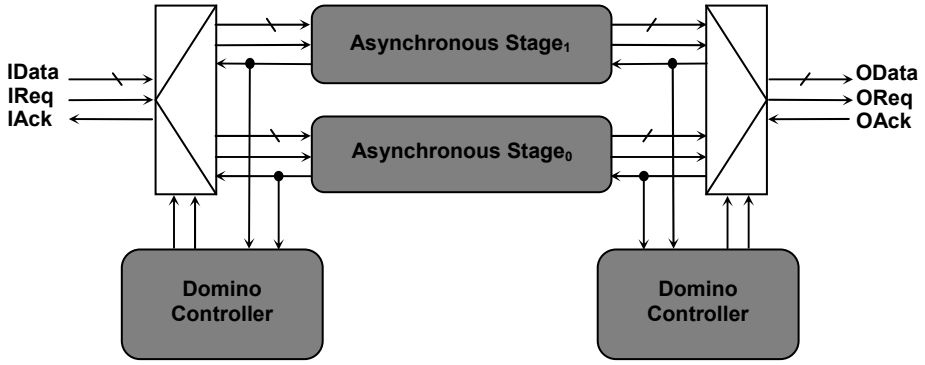


Figure 22. **Asynchronous FIFO (AA_FIFO) with a depth of 2 flits**

The design of the asynchronous multiplexer and demultiplexer using four-phase bundled-data protocol ([62]) are shown in Figure 23.a and Figure 23.b, respectively. These circuits need to do a handshake with their controller module generating the Select signals (S_i). This handshaking brings out with the sequence of S_i+, Ack_i+, S_i-, and Ack_i- . After Ack_i- indicating the end of the current four-phase sequence, the controller can select another set to be multiplexed or demultiplexed.

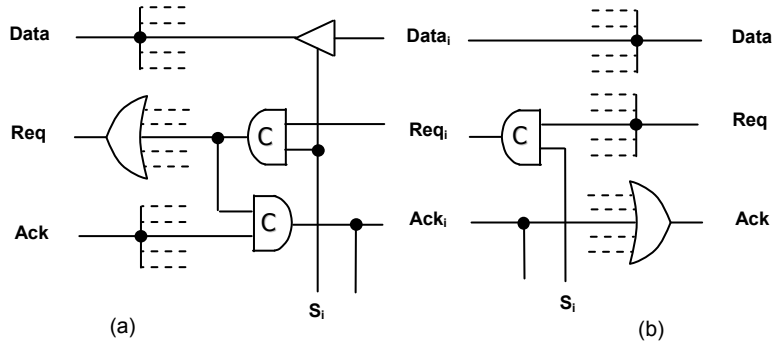


Figure 23. **Asynchronous (a) Multiplexer and (b) Demultiplexer**

The asynchronous storage stage of AA_FIFO could be a simple pipeline respecting the handshake protocol of Four-Phase Bundled-Data. The only constraint in selecting a storage stage is: each side of the stage has to perform its current data communication independent on the other side.

As storage stage, AA_FIFO instantiated in ASPIN's input ports uses an advanced asynchronous pipeline stage whose architecture is presented in [62]. The design is called "*normally opaque fully-decoupled latch controller*", and its circuit implementation is shown in Figure 24.a. The behavior of this pipeline stage, specified as an STG, is depicted in Figure 24.b.

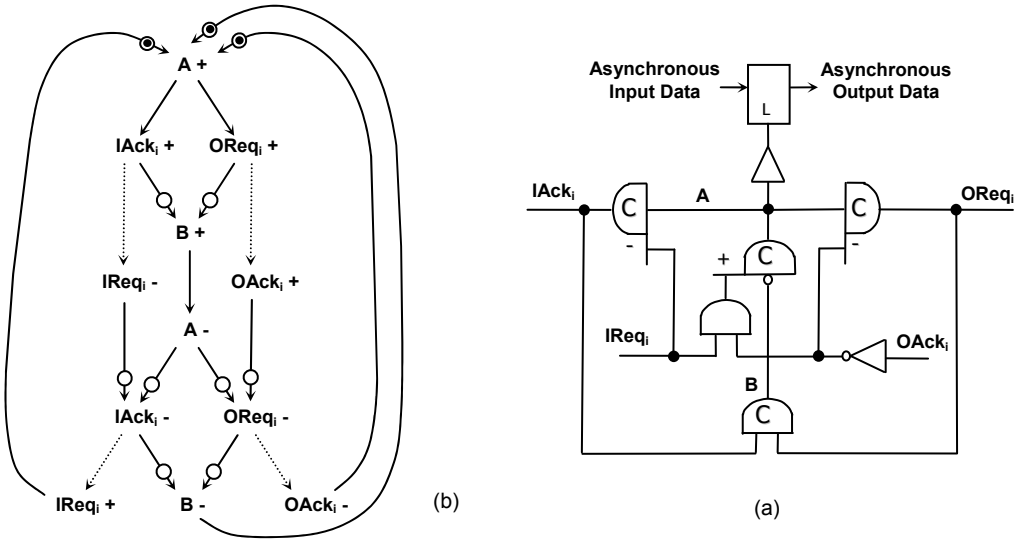


Figure 24. **Asynchronous Pipeline Stage**

3.4.1 Domino Controller

As said, the asynchronous controller used in AA_FIFO is named *Domino Controller*. It is an asynchronous One-Hot counter providing required signals for the handshake protocol of the asynchronous multiplexer and demultiplexer. As an instance, the block diagram of a 3-bit Domino Controller is illustrated in Figure 25.c. Each cell *i* has 2 outputs S_i and A_i (representing ith bit of the counter) and 4 inputs Ack_{i-1}, Ack_i, Ack_{i+1}, and A_{i-1}. The bit one is moved from cell to cell in a ring topology. At the initial state, A_2 and S_0 are high and the other outputs are low. The value high of S_0 means the first asynchronous event will be performed in $stage_0$.

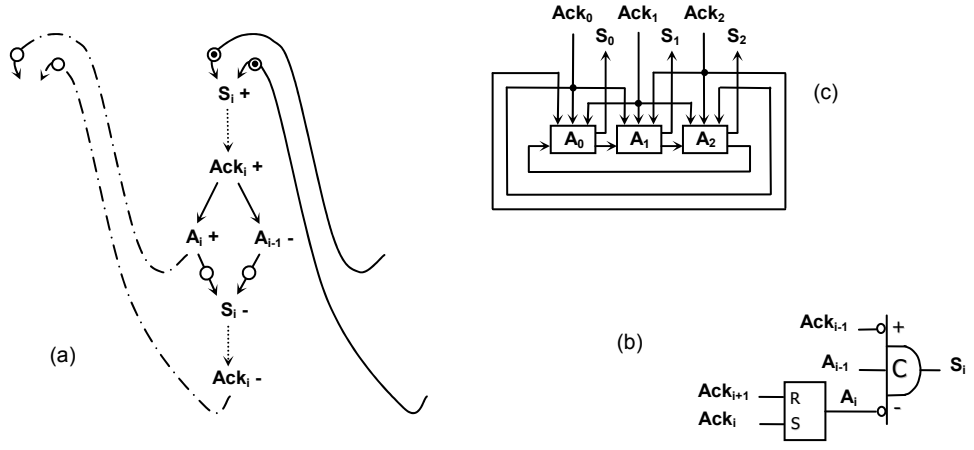


Figure 25. **Asynchronous Domino Controller**

The functionality of *Domino Controller* could be understood by looking the cell Signal Transition Graph demonstrated in Figure 25.a. The circuit implementation of this STG is

presented in Figure 25.b. Ack_i+ means S_i+ is seen, and so the bit one that at this time is in the previous cell (*i-1*) can be transferred to the current cell. The handshake protocol continues by S_i- when the transmission of the one is ended.

3.5 Summary

The detailed implementation of a novel asynchronous NoC architecture has been presented. Arranged in a two-dimensional mesh topology, this architecture (called ASPIN) is a wormhole packet-switching network. To route packets ASPIN uses the distributed, deadlock-free X-first routing algorithm. The starvation-free Round-Robin algorithm is used to schedule the concurrent requests.

ASPIN addresses also the crucial issue of the global long wires which likely have large propagation delays, incompatible with the required throughput. ASPIN's router is not a centralized macro-cell. The router is split in five separated modules (North, South, East, West, and Local) that can be physically distributed over the cluster area. This feature allows balancing the length of the long wires. Guaranteeing a delay-insensitive communication, ASPIN uses the double-rail four-phase handshake protocol for long wires.

Seeing that in a router the storage places (FIFOs) are the most important area occupant, in order to minimize the silicon area overhead the asynchronous FIFO, instantiated in ASPIN's input ports, has been designed as an optimized hard-block using single-rail (bundled-data) encoding.

Chapter 4

Clock Boundary Interfaces

As described in Chapter 1 the main problem of GALS architecture is the synchronization on clock boundaries. To resolve this problem, the use of some special FIFOs as robust interfaces between different timing domains was proposed in Chapter 2. Corresponding to the three possible combinations of mixed-timing domains, this chapter presents three new designs for different FIFO types: Asynchronous-to-Synchronous, Synchronous-to-Asynchronous, and Bi-Synchronous (Synchronous-to-Synchronous). It should be noticed that these designs must allow the designers to select their own hardware synchronizer with regard to the trade-off between latency and robustness. They have also to guaranty a throughput (data transfer rate) of one flit per cycle, and the data must be stored in asynchronous form rather than synchronous. The interest of this last requirement will be explained in Chapter 5.

4.1 General Architecture

Figure 26 shows the general architecture of two FIFOs converting asynchronous Four-Phase Bundled-Data protocol to synchronous FIFO protocol and vice versa. As shown in Figure 27 the Four-Phase, Bundled Data, asynchronous protocol, is a sequence of *REQ*+, *ACK*+, *REQ*- and *ACK*- events, where *REQ* and *ACK* are the asynchronous handshake (flow control) signals. Data is valid when *REQ* is in the high state. The high level of *ACK* indicates that the request of data communication is accepted.

Figure 28 gives examples of the synchronous FIFO protocol, where the producer and the consumer share the same clock signal, and the protocol uses two handshake signals: *ROK* (correspondingly *WOK*) and *READ* (correspondingly *WRITE*). The *ROK* signal (or not empty) is set by the producer at each cycle where there is a valid data to be transferred. The *READ* signal is

set by the consumer at each cycle where the consumer wants to consume a data on the next clock edge. Both *WRITE* and *READ* signals are state signals that can be generated by Moore FSMs.

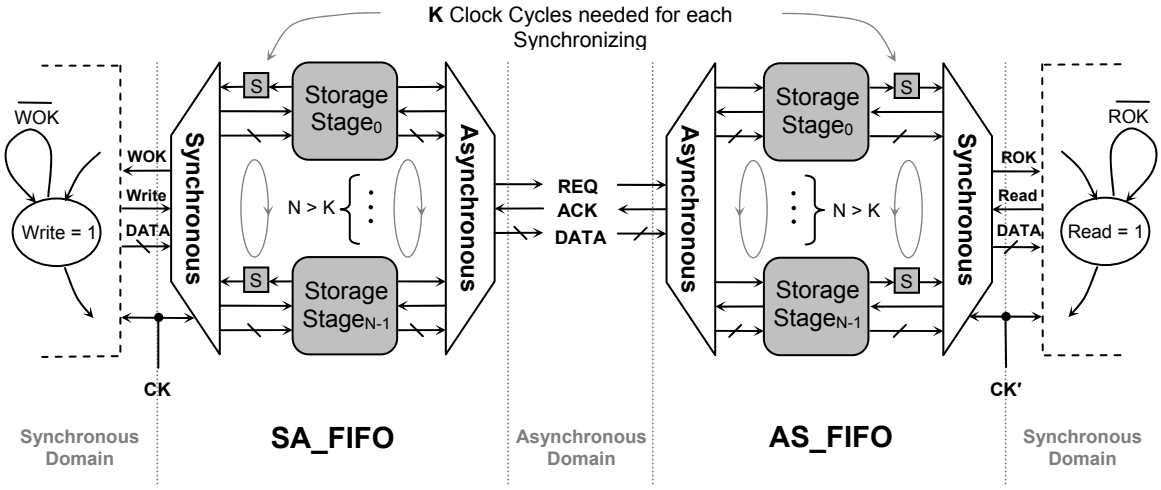


Figure 26. Synchronous ⇔ Asynchronous FIFOs

Asynchronous-to-Synchronous FIFO is called AS_FIFO, and Synchronous-to-Asynchronous, SA_FIFO. The synchronous data incoming to SA_FIFO and asynchronous data entering to AS_FIFO will be written into N pipelined storage stages in a cyclic way. The task of protocol converting is the responsibility of the storage stages.

The signals that have a risk of metastability (and must use a hardware synchronizer) are the handshake signals transmitted from the asynchronous side (precisely from the storage stages) to the synchronous side. As emphasized before, the synchronizer design is a trade-off between robustness (i.e. low probability of metastability) and latency (measured as a number of cycles in the synchronous domain). If the synchronization cost is a latency of K clock cycles, the FIFO must have at least K+1 stages, if we want a throughput of one data transfer per cycle.

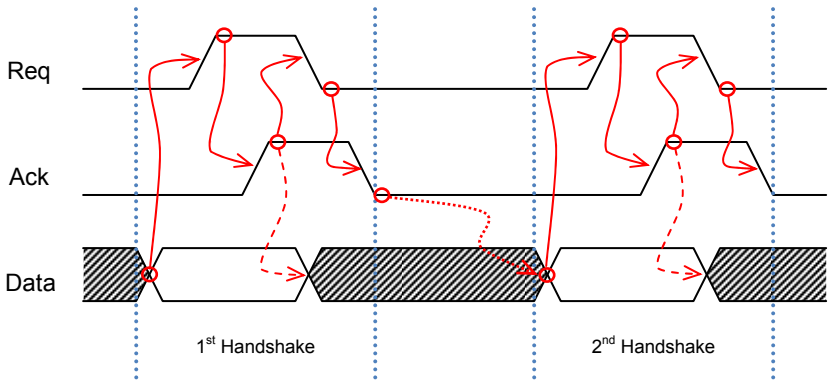


Figure 27. Four-Phase Handshake Protocol in Push Model

In such pipelined design, the effect of synchronization latency is different in the two FIFO types. In the asynchronous-to-synchronous FIFO (AS_FIFO), the synchronizer latency is visible only when the FIFO is empty. In the synchronous-to-asynchronous FIFO (SA_FIFO), it is visible when the FIFO is full. The latency between the arrival of data to an empty AS_FIFO and its availability on the output (typically named FIFO Latency) is about K clock cycles. For a full SA_FIFO, the latency between the consumption of a data and the information of the availability of an empty place on the other side is about K clock cycles.

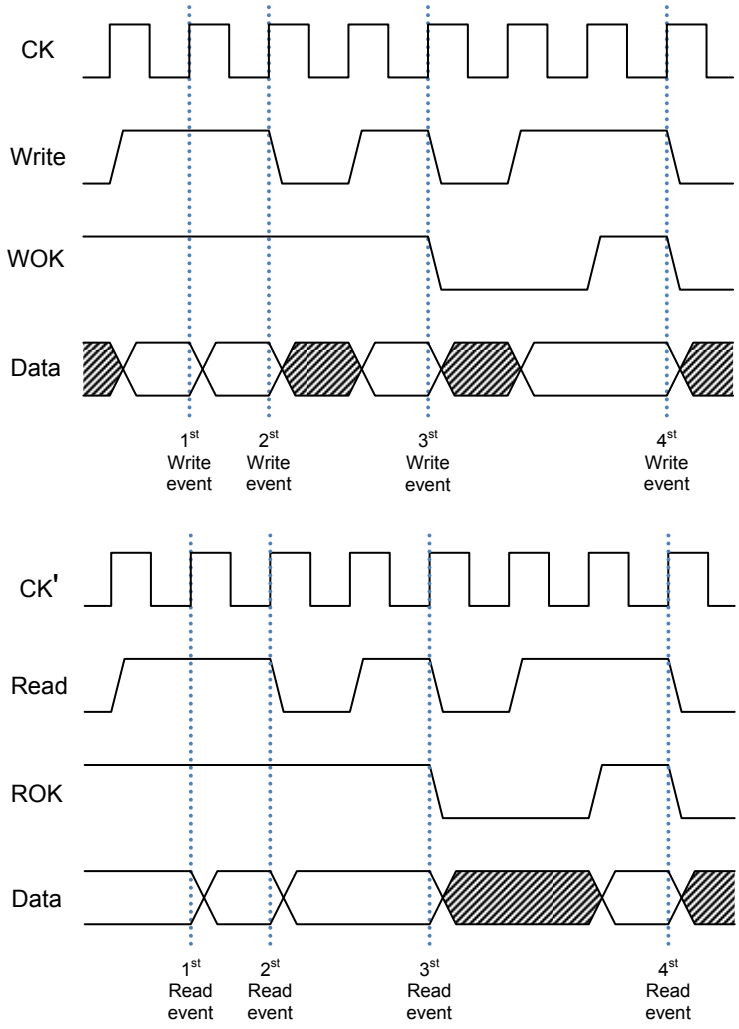


Figure 28. Write and Read event examples of Synchronous FIFO Protocol

4.2 Detailed Architecture

Figure 29.a and Figure 30.a show the internal architecture of SA_FIFO and AS_FIFO, with a depth of two storage stages. Clearly, these two architectures can be generalized to an n-stage FIFO. In SA_FIFO the asynchronous side of the design includes an asynchronous multiplexer and

in AS_FIFO it includes an asynchronous demultiplexer. These circuits need to handshake with a controller module generating their select signals in a cyclic way. The required asynchronous controller is named Domino Controller. The design of the asynchronous multiplexer, demultiplexer, and domino controller was detailed in Chapter 3.

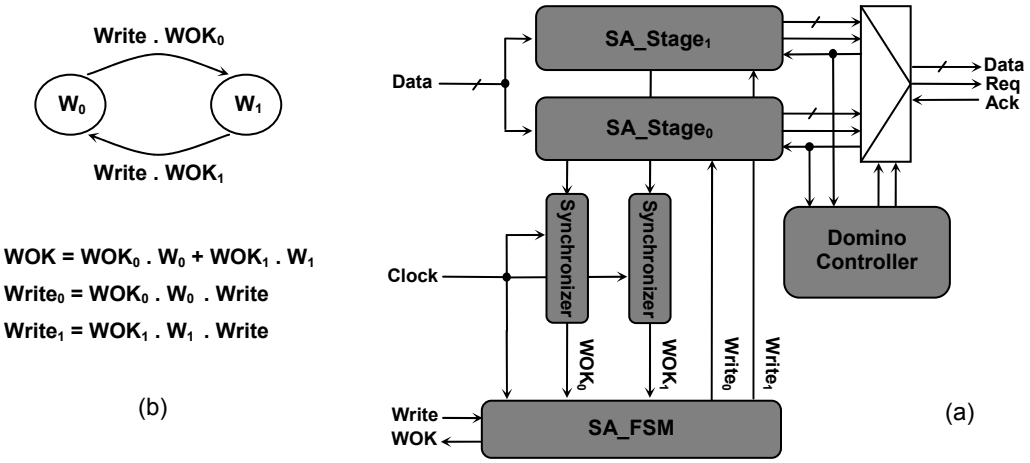


Figure 29. Synchronous-to-Asynchronous FIFO (SA_FIFO)

I present in Figure 29.b and Figure 30.b the Finite State Machines (FSMs) of the synchronous side controllers, i.e. SA_FSM and AS_FSM. These controllers are Mealy Finite State Machines. The state W_i means the next Write event will be done to stage i, and similarly, the state R_i signifies that data will be read from stage i at the next Read event. Consequently, the FIFOs status signals (*WOK* and *ROK*) depend on both the FSMs state and the status of the corresponding storage stage (being signaled by WOK_i and ROK_i). ROK_i means stage i is not empty and WOK_i indicates that stage i is not full.

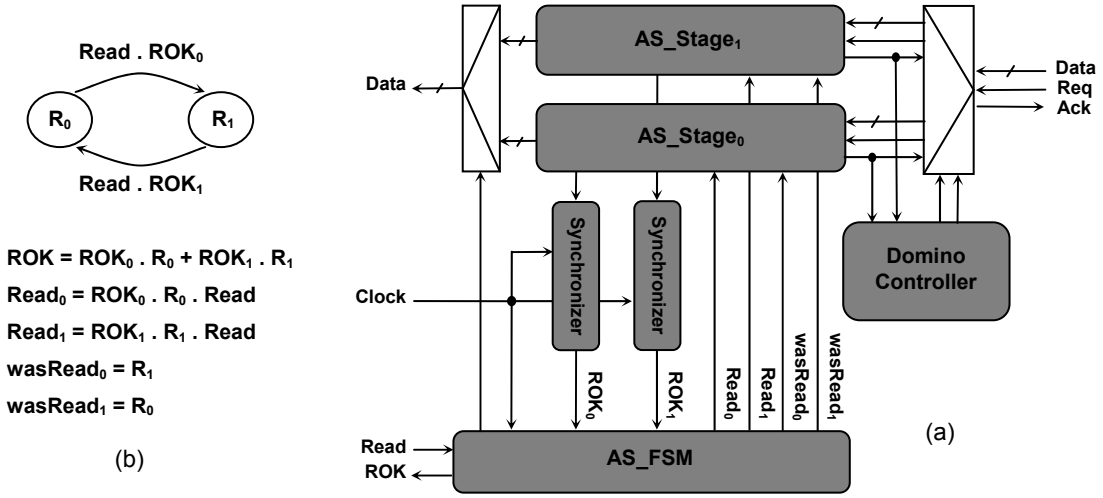


Figure 30. Asynchronous-to-Synchronous FIFO (AS_FIFO)

Synchronous hazard free commands are generated when there is a synchronous request and the current asynchronous stage is ready to accept. The positive edge of $Write_i$ indicates to the i^{th} stage of SA_FIFO that a data has to be written. The positive edge of $Read_i$ informs the i^{th} stage of AS_FIFO that the data will be read and so the stage must be freed. The positive edge of $wasRead_i$ declares that the synchronous consumer has read the data and the i^{th} stage of AS_FIFO can change its value.

As seen in Figure 29, $Write_i$ is a synchronous signal and will be asserted on rising edge of the clock signal, when a new data will be available to be sampled at the next rising edge. On the other side the storage stage (which is an asynchronous circuit) considers $Write_i$ as an asynchronous signal and as a result, immediately on its transition the input data will be written. Consequently while it is intended that the FIFO samples the data at rising edge m, the data is sampled just after rising edge m-1. Here there may be a constraint: a setup-time approximately equal to the whole clock cycle time is required.

In typical synchronous circuits this constraint is not annoying, as usually data changes to a new value on the rising edge on which the previous data is sampled, providing the required setup-time. However respecting the conservator designers, $Write_i$ can be modified to W_{i+1} (similar to $wasRead_i$ in Figure 30). With this modification the stage will be ordered to write on the next clock edge, when data is certainly valid. In this case to have the maximum throughput of one data transfer per cycle, the minimum number of stages should be K+2 where K is the synchronizer latency.

4.2.1 Storage Stages

The pipelined storage stages in AS_FIFO and SA_FIFO have two main functionalities: storing data and converting communication protocol. Figure 31.a and Figure 32.a illustrate the circuit schematics of the storage stages respectively in SA_FIFO and AS_FIFO. As demonstrated, data storage is done by latches sampling on high value of WOK_i (in SA_Stage) and L (in AS_Stage). The transition to 0 of WOK_i indicates SA_Stage contains valid data and no more writing is permitted and so data sampling must be ended. And a new data can be written into AS_Stage only when the value of L on rising edge of $wasRead_i$ (declaring the content of the stage was read) is changed to 1.

The operation of the storage stages in SA_FIFO and AS_FIFO are specified as two Signal Transition Graphs (STGs) respectively in Figure 31.b and Figure 32.b. The dotted lines are the asynchronous side transitions and the dashed lines are that of the synchronous side. According to the synchronous protocol base, the synchronous side transitions should be considered on the edges. Regarding to the two STGs, on rising edge of $Write_i$, $Read_i$, and $wasRead_i$ respectively, A, ROK_i, and C must go to the low position. In the circuit implementations three D-Type Flip-Flops which have a constant value of 0 as input data, generate these signals. The Flip-Flops will asynchronously be set when their S input (Set) signal is 1.

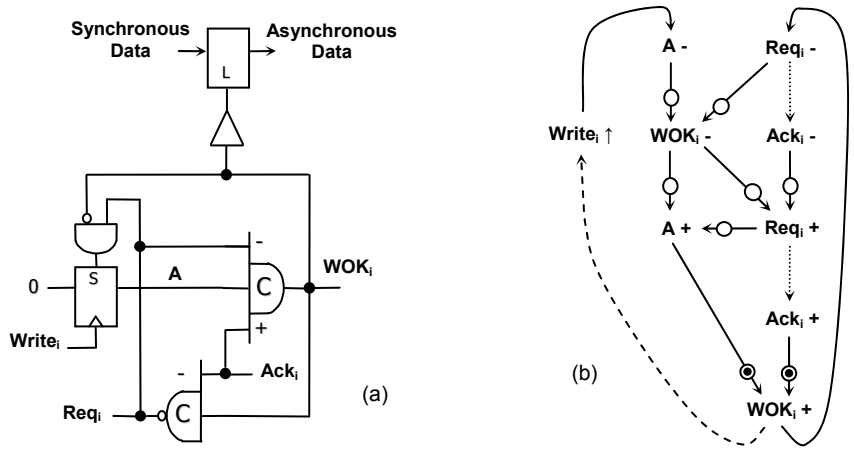


Figure 31. Synchronous-to-Asynchronous Stage (SA_Stage)

The circuit implementation of the AS_Stage shown in Figure 32 has a time constraint: before rising edge of $Read_i$, where ROK_i- must be done, the value of A should return to 0; because while A (as a set signal of the Flip-Flop) has high value, ROK_i (as an output signal of the Flip-Flop) is hold at 1 and cannot go down. The transition of ROK_i+ causes $Read_i$ to rise. Regarding to the architecture of AS_FIFO (see Figure 30), the time between ROK_i+ and rising edge of $Read_i$ ($T2$) is more than K clock cycles where K is the synchronizer latency. On the other side, A- happens after Ack_i+ occurring simultaneous with ROK_i+, by propagation delay of two gates ($T1$). Evidently a two gate propagation delay is less than the latency of a robust synchronizer. The latency of two cascaded Flip-Flops is one clock cycle. However if a designer finds a miraculous synchronizer with a very low latency (!), this time constraint may express a risk on the functionality for the design.

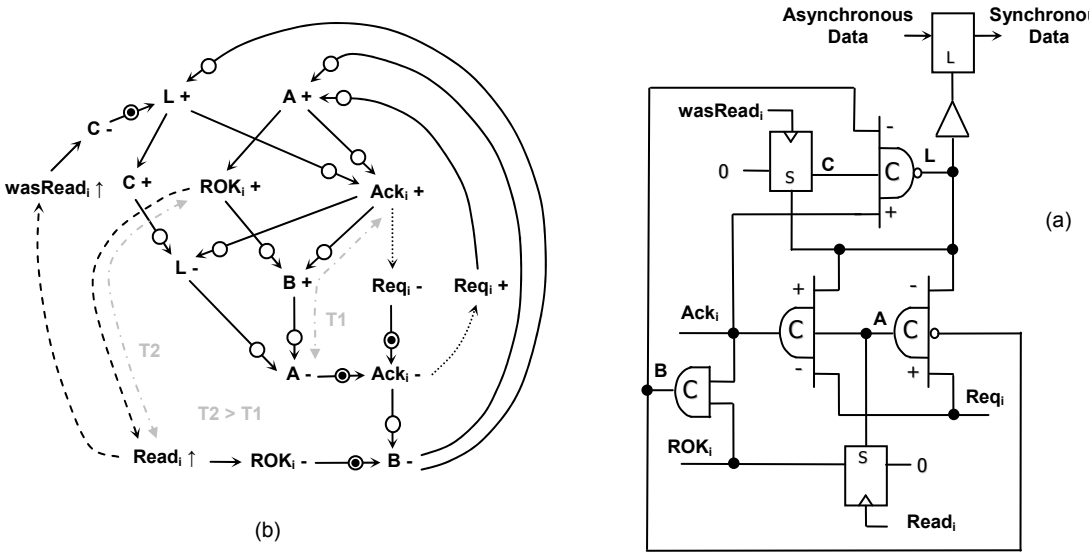


Figure 32. Asynchronous-to-Synchronous Stage (AS_Stage)

4.2.2 Improved Architecture of AS_FIFO

As explained, the architecture of AS_FIFO presented in the previous subsection has a time constraint. To resolve this restriction, here as shown in Figure 33.b, we propose a new STG of AS_Stage. In this new design, the stage will be informed to be freed at the moment it is authorized to accept a new content. $Read_i$ is removed and instead the stage will be freed (ROK_i goes to Low) on the rising edge of $wasRead_i$, where a new data is permitted to be written to the latches (L goes to High). As can be seen, the circuit implementation, demonstrated in Figure 33.a, is simpler than the previous design.

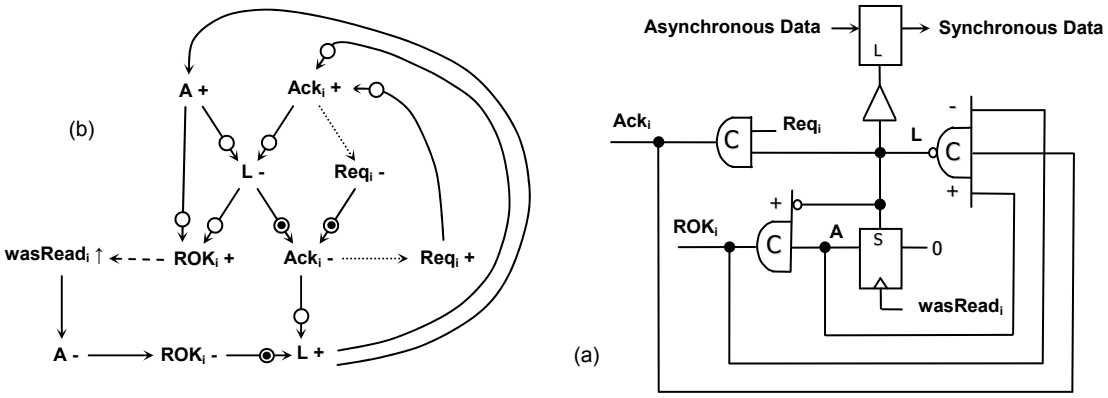


Figure 33. Improved AS_Stage

As a drawback, the stage will be freed after rising edge of the clock, and we need one more clock cycle to perform a complete read event. Consequently in order to have maximum throughput of one data transfer per cycle, this new AS_FIFO must have at least K+2 stages, one stage more than the previous architecture.

4.3 Bi-Synchronous FIFO

In Figure 34 the architecture of a Bi-Synchronous FIFO, called SS_FIFO, is presented. The design is based on the architecture principles of SA_FIFO and AS_FIFO. It can be imagined that SS_FIFO is a SA_FIFO and an AS_FIFO which are merged together: the asynchronous sides of SA_FIFO and AS_FIFO (Asynchronous Multiplexer, Demultiplexer and Domino Controllers) are removed and these two FIFOs are connected together stage to stage.

Each storage stage of SS_FIFO is composed of one SA_Stage and one AS_Stage. Thus each storage stage of SS_FIFO is able to store two data words. As a consequence the number of SS_FIFO places (the storage capacitance) is always even. This feature reduces the number of synchronizers. For each two storage place (each stage) we need one synchronizer on the producer side and one synchronizer on the consumer side. The synchronizer need is divided by two.

In addition, the complexity of the two FSMs and the output multiplexer is reduced, because for each of the two FSMs (SA_FSM and AS_FSM) in an n-place FIFO instead of n states we need n/2 states, and in place of an n-to-1 multiplexer we need an n/2-to-1. As a result, the silicon area of an n-place SS_FIFO is much smaller than an n-place AS_FIFO or SA_FIFO. The cost is a little increase in the FIFO latency, as data must pass through two stages to appear on the output port.

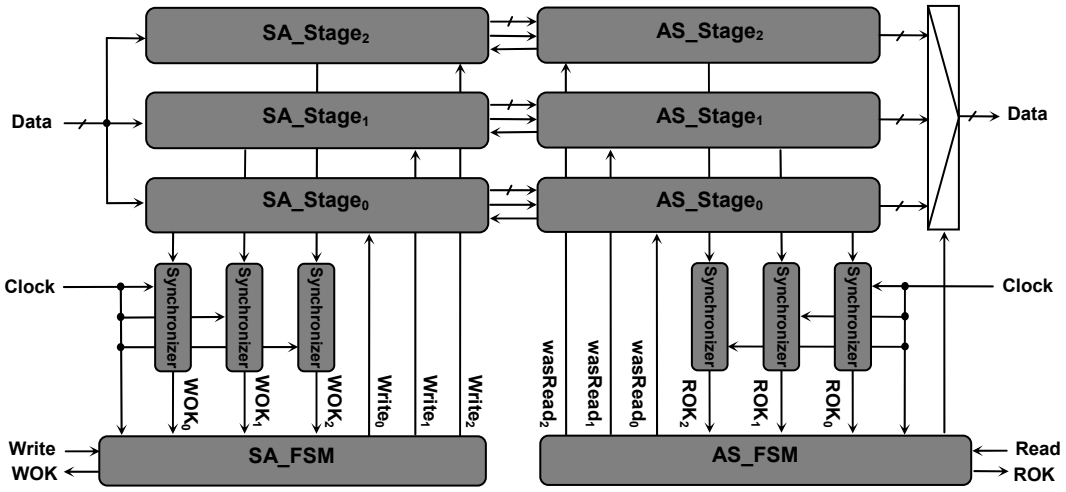


Figure 34. Bi-Synchronous FIFO (SS_FIFO)

4.4 Physical Implementation

We developed a generic FIFO generator, using the *Stratus* hardware description language of the *Coriolis* platform ([63]). This generator creates both a net-list of standard cells and a physical layout. The two parameters are the depth of FIFO (FIFO's places) and the number of data bits per flit. In this implementation the synchronizer uses two cascaded Flip-Flops and its latency is one clock cycle. In order to reach the maximum throughput of one data transfer per cycle, 2-Stage AS_FIFO and 2-Stage SA_FIFO use the constrained architectures.

As a standard cell library, we used the portable *ALLIANCE* CMOS standard cell library ([64]). The physical layouts of some 32-bit FIFOs are presented in Figure 35. The silicon areas of these examples are given in Table 2. These values are normalized to GPLVT STMicroelectronics library surfaces in 90nm fabrication process.

From the physical layout, we extracted SPICE models of the FIFOs, using *ALLIANCE* CAD Tools ([65]). The target fabrication process is the STMicroelectronics 90 nm GPLVT transistors in typical conditions. The electrical simulation results (under *Eldo*) are presented in Table 2. In this Table, T is the consumer clock cycle time. Due to relation between the asynchronous event entrance time and the consumer clock phase, AS_FIFO has various latencies with a difference of

one clock cycle. Caused by skew relation between the consumer and the producer clocks, SS_FIFO has different latencies too.

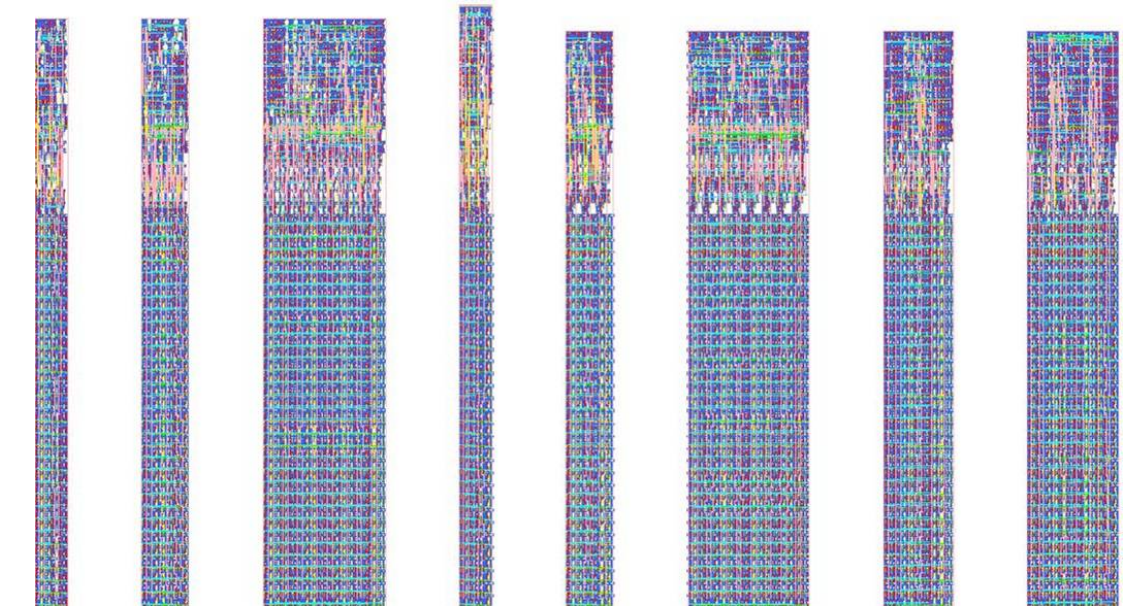


Figure 35. Physical Layouts

The throughput value of SA_FIFOs is limited by the asynchronous side and its handshake protocol. The throughput of AS_FIFOs with more than 3 stages is also limited on the asynchronous side components. But in the case of 2-Place (2-Stage) and 3-Place (3-Stage) AS_FIFO there are other constraints: regarding to Figure 32 in 2-Place AS_FIFO, Ack_i+ and $Req_{i+1}+$ must be happened in the same clock cycle if maximum throughput of one data word transferring per cycle is required. Figure 33 testifies that the throughput of one event per cycle for AS_FIFO with 3 stages is attained if ROK_i- and ROK_i+ are occurred in the same clock. This constraint should also be respected in 6-Place (3-Stage) SS_FIFO.

Due to the inability of 2-Place AS_FIFO to reach to a high throughput (comparing 1.5 GEvents/Sec with 2.61 of 3-Place AS_FIFO), in order to sustain the throughput, one could opt for 3-Place AS_FIFO. Its area (2011 μm^2) is not negligible, but it should not be forgotten that these components in addition of robustly interfacing have another advantage: providing a storage place with a FIFO behavior. As we know, in order to obtain minimum overhead of data communication between two different timing domains, having a FIFO in the interface is not eliminable. So, we suppose using a FIFO with the storage place of more than three may also be reasonable!

Finally, as a quick comparison, the minimum latency of a Mixed-Clock FIFO presented by Tiberiu Chelcea and Steven M. Nowick in [16] is $0.5\ T_P + 2.5\ T_C$ and its maximum value is $0.5\ T_P + 3\ T_C$ where T_P is the producer clock cycle time and T_C is that of the consumer. The maximum

throughput of an 8-bit 4-place Mixed-Clock FIFO is 549 MHz. This evaluation has been given for 0.6 µm HP CMOS technology. The same evaluations for Async-Sync and Sync-Async FIFOs are 421 and 454 MHz respectively.

Another example to compare could be the bi-synchronous FIFO presented by Ivan Miro Panades in [11]. The minimum latency of this FIFO is 2 clock cycles and its maximum delay is 3. If it is intended to have the maximum throughput of one flit per cycle, the number of places must be more than 5. This architecture has been synthesized under *Synopsis* with two different configurations: the output data path is implemented, first, by using tri-state buffers, and second, by using multiplexers. With 32-bit data word using STMicroelectronics CMOS 90 nm standard cells, an 8-place FIFO with tri-state buffers occupies 8032 μm^2 and with multiplexers 6581 μm^2. The maximum frequency of write events in both architectures is 2000 MHz. But the maximum frequency of read events in tri-state buffer model is 1250 MHz and in model using multiplexer is 1000 MHz.

Table 2. Simulation Results

FIFO	Transistors	Surface	Min Latency	Max Latency	Max Throughput
2-Place SA_FIFO	1338	1422 μm^2	177 pS		2.39 GEvents/S
3-Place SA_FIFO	1969	2054 μm^2	207 pS		2.36 GEvents/S
8-Place SA_FIFO	5126	5215 μm^2	219 pS		2.22 GEvents/S
2-Place AS_FIFO	1388	1452 μm^2	271 pS + T	271 pS + 2T	1.50 GEvents/S
3-Place AS_FIFO	1942	2011 μm^2	247 pS + T	247 pS + 2T	2.61 GEvents/S
8-Place AS_FIFO	5054	5107 μm^2	263 pS + T	263 pS + 2T	2.89 GEvents/S
6-Place SS_FIFO	2985	2940 μm^2	362 pS + T	362 pS + 2T	2.61 GEvents/S
8-Place SS_FIFO	3956	3869 μm^2	366 pS + T	366 pS + 2T	4.60 GEvents/S

4.5 Summary

Three new FIFO architectures for interfacing an asynchronous NoC with synchronous subsystems or two adjacent routers in a multi-synchronous NoC have been presented. The design of the FIFO interfacing two asynchronous and synchronous domains can be used as a convertor to convert the asynchronous Four-Phase Bundled-Data protocol to the synchronous FIFO protocol.

The synchronizer used in the architectures can be arbitrarily chosen by the system designer, supporting various trade-off between latency and robustness. The FIFOs can achieve the maximal throughput of one word per cycle, even if the selected synchronizer has a large latency. The designs have been physically implemented with portable ALLIANCE CMOS standard cell

library. Using the STMicroelectronics 90nm GPLVT CMOS fabrication process, the throughputs and latencies have been evaluated by post layout SPICE simulation from the extracted layout.

Chapter 5

Saturation Threshold

The classical interconnects do not scale when the number of components to interconnect increases, and saturate when too many cores generate traffic. The saturation occurs when the load offered by each component reaches a point called saturation threshold, where the average packet latency raises exponentially to an infinite value. In this case the interconnect becomes the system bottleneck. In scalable interconnects the value of saturation threshold is roughly independent on the number of communicating components.

Networks-on-Chip are much more scalable than traditional on-chip interconnects. However the network saturation threshold can still become a problem. A low saturation threshold is an important limiting factor for delay sensitive applications. Recall that the main motivation supporting the NoC paradigm is to improve the saturation threshold. This chapter presents the evaluation of the saturation threshold in DSPIN and ASPIN architectures, and considers some improving techniques.

5.1 Simulation Platform

To evaluate the saturation threshold, a system-level simulation platform which will provides the means to generate packets according to a parameter determining the traffic load, and to measure packet latencies is needed. We have focused on a network in which each cluster contains one Traffic Generator (TG) and one Traffic Analyzer (TA). We used the same cycle accurate SystemC models for Traffic Generator (TG) and Traffic Analyzer (TA) components in both DSPIN and ASPIN simulations.

As said before, we have developed a generic ASPIN generator generating a gate-level net-list of standard cells in structural VHDL. The cell behavioral models are written as *Transport Delay*

Models. As examples, the VHDL behavior models of basic cells of asynchronous circuits are presented in Appendix C. We have used *ModelSim* to perform a VHDL-SystemC co-simulation including ASPIN VHDL model and the cycle accurate TG and TA SystemC models. Refer to Figure 36. The synchronous DSPIN router was described as a cycle accurate SystemC simulation model.

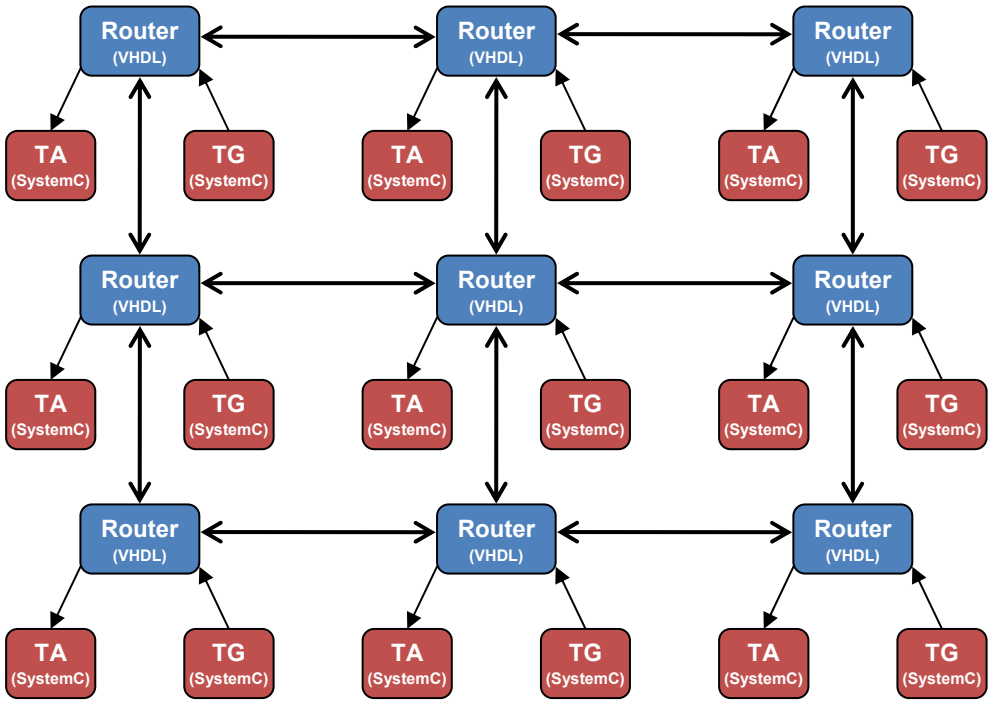


Figure 36. VHDL-SystemC Co-Simulation Platform for ASPIN

The exact value of saturation threshold in a NoC depends on the following parameters:

- **Traffic load:** The total traffic offered by traffic generators during a specified time interval is called traffic load. In a real application the traffic load does not remain constant and has different values at different points of time.
- **Traffic Distribution:** The packet destinations are usually distributed over the network and as a consequence the traffic is not concentrated in some fixed points. However although in a heterogeneous system there are always some hot points and traffic is distributed non-uniformly, the uniform traffic distribution is a model of destination distribution in homogeneous applications.
- **Packet Length:** In a wormhole packet-switching network a packet spread along the path and occupies several network channels. When the packet length is short a lower number of network resources are reserved than when it is long. As a result the packet length has a direct influence on the network saturation threshold.

- **Distributed Storage Capacity:** In a network there is a number of buffer queues distributed in the routers. A larger depth of these FIFOs induces gathering the flits of stalled packets in a lower number of channels, and consequently more network resources are available for other packets, resulting in a higher saturation threshold.

5.1.1 Traffic Generator

The task of the Traffic Generator (TG) is to generate packets and to inject them into the network. The generated traffic has a uniform random distribution. That is to say each TG sends randomly packets to all external Traffic Analyzers (TAs) with a fixed load equivalent in all clusters. When generating a new packet, a destination will be randomly selected and its coordinate in the form of (Y, X) will be included in the header of packet. As a time stamp, the point of time at which the packet begins to be injected (called *START*) will also be included.

If *TOTAL* is the aggregate packet length (total number of flits injected into the network), and *TIME* is the total number of clock cycles, the offered load is defined, for each TG, as the percentage of the maximal bandwidth:

$$Load = \frac{TOTAL}{TIME}$$

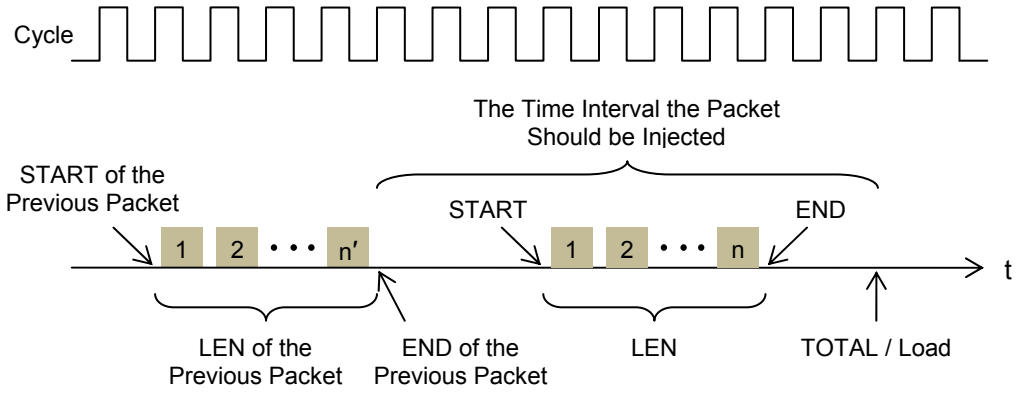
When a new packet is being generated, *START* (the injection time of the packet) will be determined according to the fact that the requested offered load must remain respected. *TIME* (equal to *TOTAL* / *Load*) is the end point of the time interval in which the packet should be injected into the network. The random selection of *START* allows providing a uniform random traffic spectrum.

In order to take into account the network contention and to have a meaningful latency measurement, the packets are posted in an infinite FIFO located in each TG. In other words, it is supposed the generated packets are injected with the maximum rate of one flit per cycle. For example if a packet begun to be transmitted at *START* and its length is *LEN*, the packet injection will be finished at *END* equal to *START*+*LEN*, where a new packet will be generated. For each new packet a new *START* will be determined independent of the real clock cycle number. In fact *START* represents a virtual cycle number at which the packet should be transmitted to the virtual infinite FIFO. If the real clock cycle number has exceeded (or exceeds) from *START*, the FIFO is not empty and thus TG starts to inject the packet into the network. To better understand refer to Figure 37.

5.1.2 Traffic Analyzer

The task of the Traffic Analyzer (TA) is to evaluate the average latency of received packets. The packet latency is measured as the number of clock cycles between the posting time in the source node (included as a time stamp in the header of the packet), and the arrival time in the destination

node. TAs store their results in a common file to be used later. After each simulation we have employed *MatLab* to read the result file and to plot the average packet latency versus the offered load, helping to extract the saturation threshold.




```
void next_packet()
{
   END = START+LEN ;
   LEN = next_length() ;
   TOTAL += LEN ;
   START = (rand() % ((TOTAL*100/Load) - END)) + END + 1 ;
}

void transition()
{
   if (RESETN == true)
   {
      G_FSM = G_IDLE ;
      Cycle = 0 ;
      START = 0 ;
      TOTAL = 0 ;
      LEN   = 0 ;
      next_packet() ;
   }
   else
   {
      Cycle++ ;
      switch (G_FSM)
      {
         case G_IDLE :
            if (Cycle >= START) G_FSM = G_BOP ;
            break ;
         case G_BOP :
            if (WOK == true) { G_FSM = G_DATA ; n = 1 ; }
            break ;
         case G_DATA :
            if (WOK == true) n++ ;
            if (LEN-1 == n) G_FSM = G_EOP ;
            break ;
         case G_EOP :
            if (WOK == true) { G_FSM = G_IDLE ; next_packet() ; }
            break ;
      }
   }
}
```

Figure 37. Uniform Random Traffic Generator

5.2 Multi-Synchronous vs. Asynchronous

Seeing the fact that in a typical shared-memory MP-SoC most of the packets traversing the network are cache miss requests and responses, in our simulations we have chosen a packet length of 16 flits assumed equal to the cache line length. And, we have focused on a network containing 5×5 clusters.

To compare the saturation threshold of the asynchronous (ASPIN) and multi-synchronous (DSPIN) architectures, we have assumed DSPIN router is directly clocked by local subsystems. In our simulations we have chosen a speed ratio of 5 between the asynchronous network and the synchronous subsystems throughputs. For example, if ASPIN works at 1 GFlits/s, clusters inject the flits with a throughput of 200 Mflits/s. Indeed if the multi-synchronous DSPIN network is clocked by a dedicated mesochronous clock signal, with a speed equal to the ASPIN throughput, from the point of view of saturation threshold the behavior of the two networks will be approximately the same. Therefore, from this point forward, in this chapter, DSPIN (*Multi* in the figures) means a network of which the routers are clocked locally and work at the speed of subsystems. Naturally the results of ASPIN (*Async* in the figures) may almost be considered as results of a multi-synchronous network clocked with a fast dedicated signal at a speed five times faster than the speed of the subsystems clock. In the next subsection the influence of the speed ratio and the saturation threshold improvement obtained in a network with a throughput much higher than the flit injection rate will be analyzed.

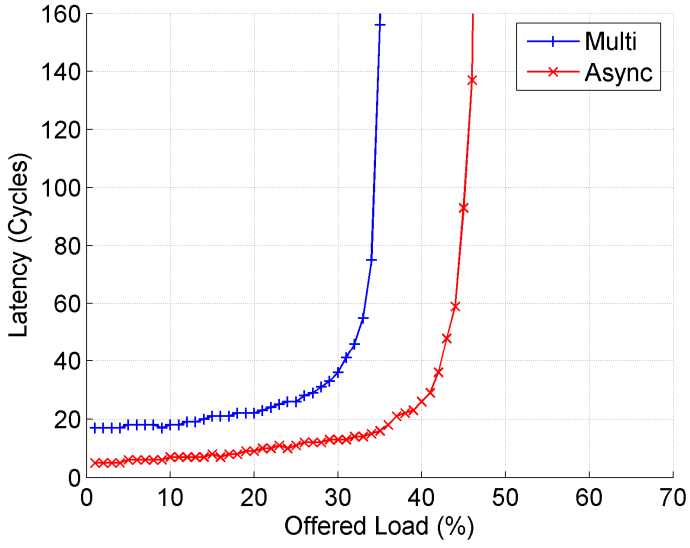


Figure 38. The Network Saturation Threshold in DSPIN and ASPIN

Figure 38 depicts the DSPIN (Multi) and ASPIN (Async) average packet latencies (in cycles) versus the offered load. DSPIN saturates for an offered load of about 34%. ASPIN saturates for an offered load of 44%. In a packet switching network when a packet is traversing the network all

resources in the path between the header (first flit) and the trailer (last flit) are allocated to the packet and no other packet can use those resources. The other packets must wait until the path is released, that is after the packet trailer is passed. In a multi-synchronous NoC the flits injected to the network (as well as the trailer) move through the hops cycle by cycle, with a throughput of one flit per cycle. In contrast, in the asynchronous approach flits propagate as fast as possible. Imagine the case that the speed ratio between network and flit injectors (subsystems) is larger than one. In this case, in the asynchronous network, the trailer releases the path faster than in a multi-synchronous one which works with the speed of subsystems. We believe fast path liberation is why fast asynchronous NoCs have a better saturation threshold.

5.3 FIFO Depth

In case of contention, the storage capacity distributed in the network helps to gather the flits of the stalled packets. Therefore, increasing the depth of the FIFOs in the routers, improves the saturation threshold. In Figure 39, the average packet latencies of DSPIN and ASPIN are shown for two different FIFO depths of 4 and 16 flits. The saturation threshold is improved to about 42% for DSPIN and to about 64% for ASPIN. However, the price to pay is very high. Even for small FIFOs, for example with a depth of 4 flits, most of the router silicon area is due to the FIFOs.

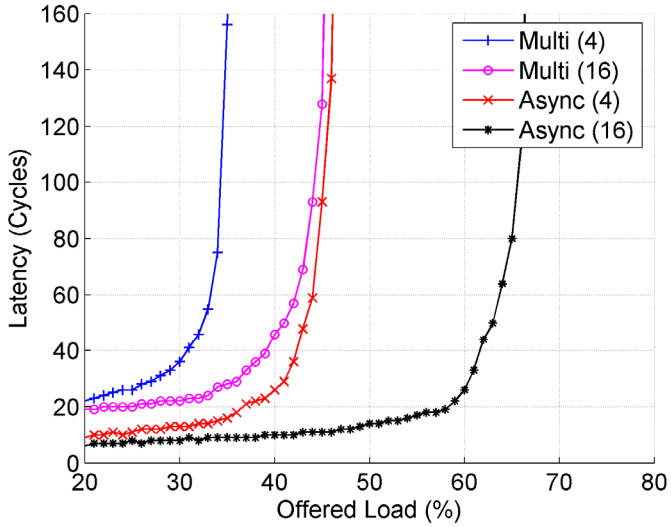


Figure 39. The influence of the distributed storage capacity on the saturation threshold

5.3.1 AS_FIFO

As explained previously, the physical links between the asynchronous network and the synchronous subsystems are two Async-to-Sync and Sync-to-Async FIFOs. SA_FIFO is the first step in each packet path and AS_FIFO is the last. In the path between SA_FIFO and AS_FIFO

there are a variable number of asynchronous FIFOs. The notion of x.y.z in this chapter indicates x as the SA, y as the intermediates, and z as the AS FIFOs depths.

If the clock frequency of the destination subsystem is much lower than the network average throughput, the flits arriving to AS_FIFO cannot exit the network fast, and thus accumulate in the AS_FIFO. As a Back-Pressure, the flit accumulation is retro-propagated trough the network and occupies network resources, causing network saturation.

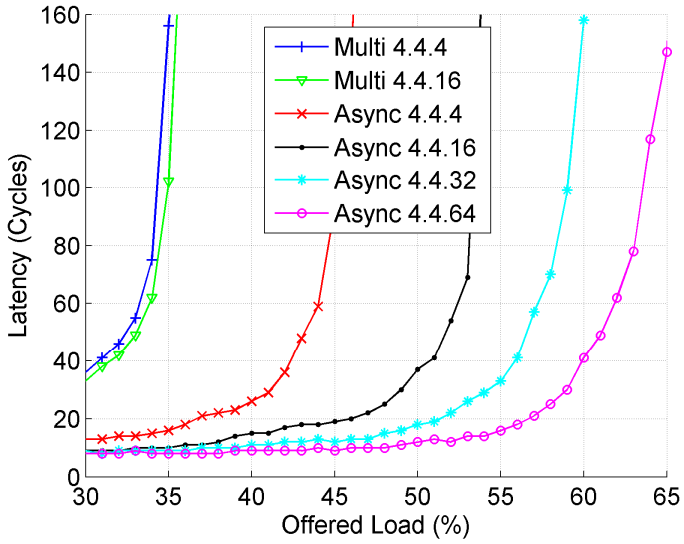


Figure 40. The influence of the last FIFO (AS_FIFO) on the saturation threshold

As illustrated in Figure 40, by paying for an extra silicon area and power consumption, it is possible to strongly improve the saturation threshold. We just need to increase the depth of the last FIFO (AS_FIFO), for example, to 16, 32, or 64 flits.

In a heterogeneous shared-memory MP-SoC application the packet distribution is not uniform. For example, if a system has an external memory controller, a large number of requests will be addressed to the corresponding subsystem. In this case, the system designer will not introduce large AS_FIFO in all clusters. It's enough to instantiate a huge FIFO in the clusters containing the external memory controller.

For a multi-synchronous network ,where the rates of flit injection/consumption and the network throughput are identical (one flit per cycle), the size augmentation of the last FIFO (that of the local output ports) has no significant effect on the saturation threshold.

5.3.2 SA_FIFO

In the case of contention in an asynchronous NoC the flits can be blocked even in the first step (i.e. in SA_FIFO). If SA_FIFO becomes full, NIC is stalled, and the blocked flits remain in

synchronous form. When the network contention is resolved, the blocked flits will be sent at the subsystem speed, one flit per cycle. Clearly, increasing the depth of SA_FIFO will improve the saturation threshold. Recall that thanks to our SA_FIFO, the stored flits have been converted to asynchronous format and will be injected to the network as fast as possible.

According to the simulation results, depicted in Figure 41, the saturation threshold is about 62% with a depth of 4 for SA_FIFO, and is about 70%, when SA_FIFO has a depth of 16 or more. The optimal FIFO depth depends on the packet length, as the SA_FIFO should just have the capability to store a complete packet.

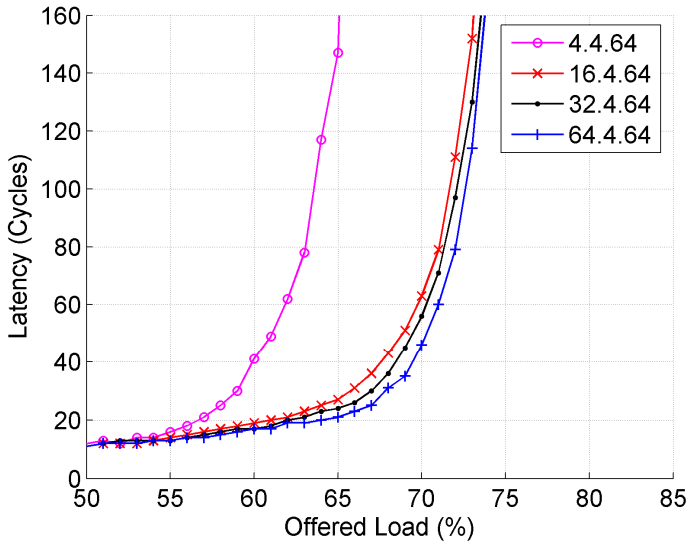


Figure 41. The influence of the first FIFO (SA_FIFO) on the saturation threshold

In an ASPIN router there are one SA_FIFO, one AS_FIFO, and four other FIFOs corresponding to the north, south, east, and west input ports. Thus for a router of 16.16.16, the total storage capacity is 16 + 4×16 + 16 = 96 flits. As presented before, regarding to our simulations the saturation threshold of such an asynchronous network is about 64%. A 16.4.64 network router has the same storage cost (16 + 4×4 + 64 = 96 flits), but the saturation threshold is about 70%. This fact describes the importance of allowing the flits to gather in the last and first steps, instead of distributing over the network.

5.4 Network Throughput

All previous discussions emphasize the effect of network throughput on the saturation threshold. In fast networks, the path liberation is faster than in networks working with the speed of clusters. As a testimonial, we plotted in Figure 42 the saturation thresholds of a 16.4.256 ASPIN network for different values of the ratio between the asynchronous network throughput and the

synchronous subsystems clock frequency. In this simulation, the large depth of AS_FIFO significantly decreases the back-pressure effects and thereby the network throughput can influence the network saturation as a predominant factor.

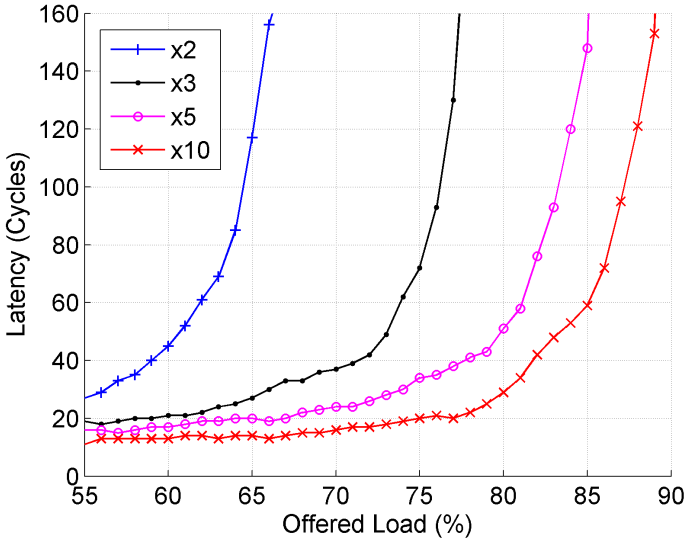


Figure 42. The influence of the speed ratio on the saturation threshold

While a saturation threshold of more than 80% is really an artistic achievement, the speed ratio of 5 or 10 is a surrealistic assumption. As said before, our goal was not to extract the real and feasible value of the saturation threshold. Instead, we wanted to show the importance of the ratio between the network throughput and the flit injection rate.

5.5 The approach of Quasi-Store-and-Forward

As projected by the 2005 ITRS [66], in few years from now, 14×14 mesh topologies will be practically feasible. Figure 43 presents the simulation results for the saturation threshold in such large network. As shown, the saturation threshold of DSPIN degrades to 11%. Such a low saturation threshold could really become a system bottleneck. Likewise, the saturation threshold of ASPIN (an Asynchronous NoC) is reduced to about 30%. In this simulation we used 16.4.16 networks, and a speed ratio of 5 for the case of ASPIN.

We believe that this can be improved by a new switching policy. When the network throughput is faster than the flit injection rate, there is an unusable delay between two consecutive flits of a packet. As a consequence a packet tends to occupy the path more than necessary, reducing the saturation threshold. In order to ameliorate the saturation threshold, we propose to replace the end-to-end wormhole switching, by a new approach, called Quasi-Store-and-Forward (QSF).

In this approach, all flits of a given packet accumulate in the asynchronous form before entering the network. Recall again that all retained flits in a SA_FIFO (presented in this thesis) have been converted to the asynchronous form. Thereby, the packet is not authorized to go until the packet trailer (the end of packet) arrives in the SA_FIFOs. The router-to-router asynchronous communications remain in wormhole approach.

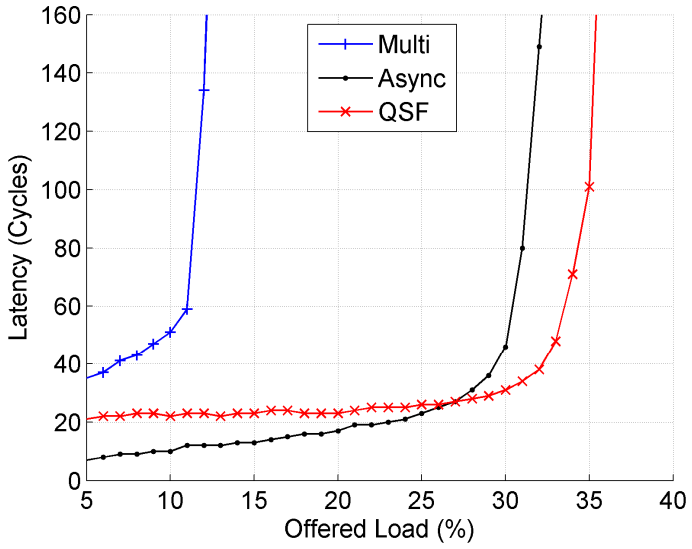


Figure 43. The influence of the QSF approach on the saturation threshold

With this QSF approach, the packet does not occupy any resources unless it is sure that all its flits can traverse the network at full speed. In fact QSF approach attempts to realize a network offered load reduction by a coefficient equal to the speed ratio. The simulation results for the QSF approach are presented in Figure 43 (red curve). The saturation threshold is improved, and the cost is an increased average latency.

5.5.1 QSF Implementation

Figure 44 shows the block diagram of the QSF Implementation. In this figure *Input Controller* represents a synchronous FSM controlling the entrance of packets, and *Output Blocker* symbolizes an asynchronous circuit ruling the packet departure. Firstly, *Output Blocker* blocks the output path, and thereby the incoming packet will be retained in SA_FIFO. When the packet trailer (where the flag of EOP is 1) enters to the FIFO, *Input Controller* set the signal of *Trig*, indicating a whole packet is stored (and thus converted to the asynchronous form). Afterward, *Output Blocker* releases the path, and then when the entire packet exits the FIFO, re-prevents any further output transitions. The signal of *eop*, asserted by *Output Blocker*, indicates that the packet left the FIFO and flits are not authorized anymore to go out. As *eop* is an asynchronous signal toward a synchronous domain, the use of a hardware synchronizer is required.

As the size of SA_FIFO is often limited, in certain applications there are likely some packets which cannot be entirely stored. So another responsibility of *Input Controller* is to count the number of entered flits. When the FIFO becomes full, even if the end of packet is not arrived, *Trig* must be asserted.

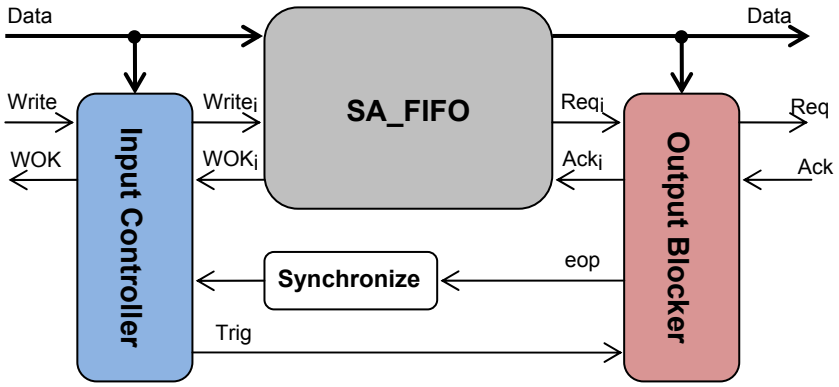


Figure 44. QSF Implementation

The Finite State Machine of *Input Controller* is shown in Figure 45. At the initial time the FSM is at the state T2. It enters to the state N2 when the end of packet is arrived and the FIFO is not full. The state N1 indicates the situation that the FIFO becomes full before the arrival of the packet trailer. In both states the flit input will be halted (*Write$_i$* and *WOK* set to zero), waiting for *eop*. Note that at this time *eop* signifies the exit of the previous packet and there is always one packet interval between *Input Controller* and *Output Blocker*. As a result when a packet is going out, another is allowed to be entered, keeping the maximum bandwidth. *Trig* will be set when the FSM is in the states T1 and T2.

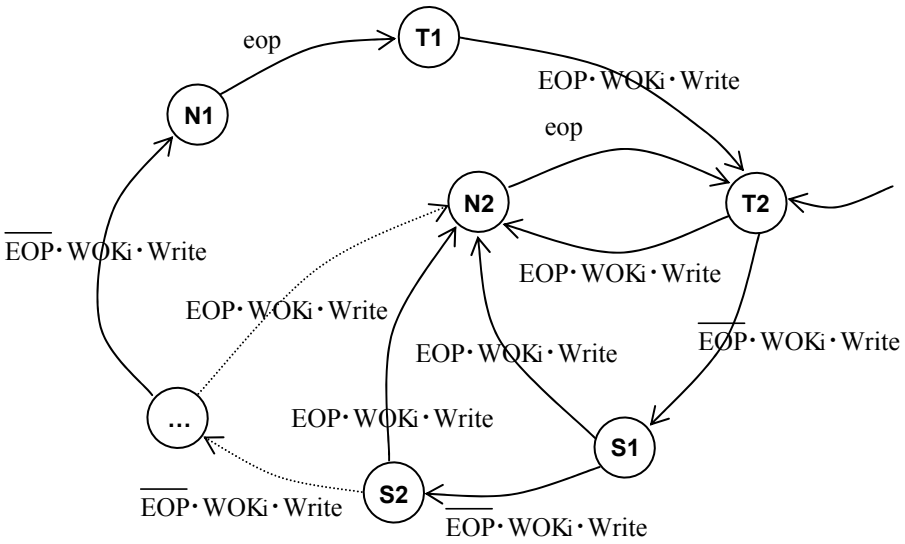


Figure 45. Finite State Machine of Input Controller

The behavior of *Output Blocker* is specified as a Signal Transition Graph (STG), shown in Figure 46.a. In this STG the dotted lines represent the external asynchronous transitions and the dashed line stands for the transition of the synchronous domain. Respecting synchronous protocol base, the transition of the synchronous signal (*Trig*) should be considered on the edge. In the circuit implementation (demonstrated in Figure 46.b) a D-Type Flip-Flop is used to achieve this requirement. On the rising edge of *Trig* the input constant value of 0 appears on the output, generating *B*-. In order to generate *B*+ the Flip-Flop will asynchronously be set when the *S* input (Set Signal) is high. When *B* is high *Req*+ cannot be happened and thereby the handshake does not continue, blocking the path. On the contrary, when *B* is low *Req*+ may be occurred and *Output Blocker* is in the release state.

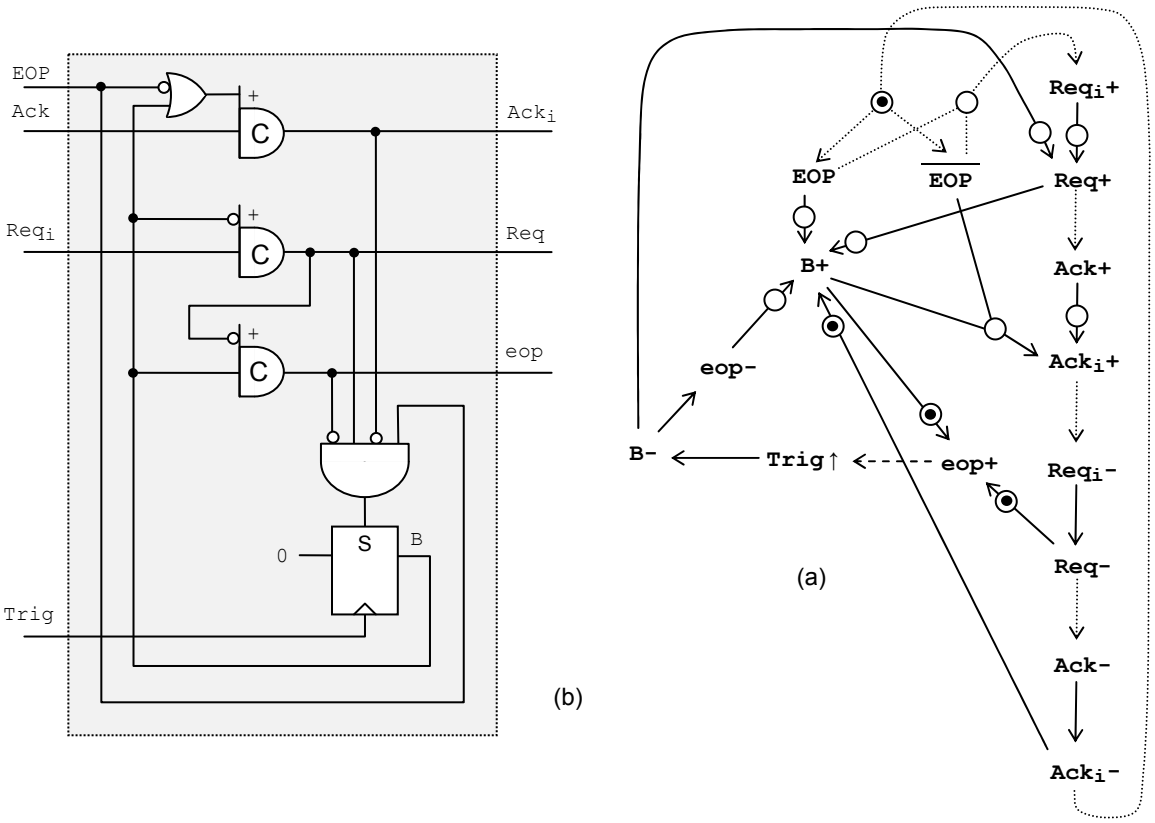


Figure 46. Output Blocker: (a) STG and (b) Circuit Implementation

5.6 Summary

Although NoCs are much more scalable than traditional interconnects, when the number of clusters generating traffic increases the network saturation threshold which is a critical issue reduces and sometimes becomes the system bottleneck. This chapter presented a systematic evaluation of the saturation threshold in ASPIN and DSPIN architecture. ASPIN has fully asynchronous network architecture and DSPIN is a multi-synchronous NoC. In the evaluations the

influence of two parameters has been considered: the flit storage capacity of the network and the network throughput.

We have shown that increasing the storage capacity distributed in the network can improve the network saturation threshold, but the corresponding cost may be much cheaper for fast asynchronous network than for multi-synchronous one which works with subsystems speed. We justified the importance of allowing the flits to gather in the last and first FIFO steps, instead of distributing over the network

Finally, we proposed a new method to improve the saturation threshold in fast and large asynchronous networks: using a Quasi-Store-and-Forward (QSF) algorithm instead of end-to-end wormhole routing. In this approach, all flits of a given packet accumulate in the asynchronous form before entering the network. With this QSF approach, the packet does not occupy any resources unless it is sure that all its flits can traverse the network at full speed. In fact QSF approach attempts to realize a network offered load reduction by a coefficient equal to the speed ratio.

Chapter 6

Physical Performance Comparison

Will future NoCs be synchronous or asynchronous? As the general architectures and the provided services are totally identical in DSPIN and ASPIN implementations, the physical performance comparison between these two architectures may help to answer this question. The trade-off between costs and performances is a major issue of NoC design and determines a NoC is cure or curse. In this chapter we present a systematic comparison in terms of Silicon Area, Power Consumption, Communication Throughput, and Packet Latency which are the most important NoC physical performance parameters.

All results presented in this chapter have been obtained in close cooperation with Ivan Miro Panades, who was in charge of the DSPIN evaluation. We have developed the gate-level structural VHDL models, as well as the layout synthesis, for all ASPIN and DSPIN components. I have developed a first version of Synthesizable VHDL models for all DSPIN components. An improved implementation, including an optional Quality of Services (QoS), has been developed by Ivan MIRO. Regarding the ASPIN components, I developed a generic ASPIN generator, using the *Stratus* procedural hardware description language of the *Coriolis* platform [63]. From a user point of view, *Stratus* allows Python programming flow control, variable use, and specialized functions in order to handle VLSI objects. The *Stratus* language gives the user the ability to describe net-list and layout views. The ASPIN generator generates both a gate-level net-list and the physical layout. Figure 47 demonstrates some layout examples of ASPIN modules. A complete 32-bit ASPN router with FIFO depth of 8-word contains 47672 transistors.

To evaluate the electrical characteristics we have extracted the SPICE models for all DSPIN and ASPIN components. The target fabrication process is the STMicroelectronics 90 nm GPLVT. Electrical simulations have been performed for typical conditions under *Eldo*. For some physical performance parameters such as power consumption, or communication throughput, the length of

the long wires is a key factor. The length of these wires depends on the cluster size as the routers are physically distributed. In 90 nm fabrication process 2×2 mm^2 is a rough surface estimation for a large cluster. Therefore, in the performance evaluations we have considered a simple RC model of wires with a length of 2 mm. The model of an intra-cluster wire connecting one input module to four output modules is shown in Figure 48.

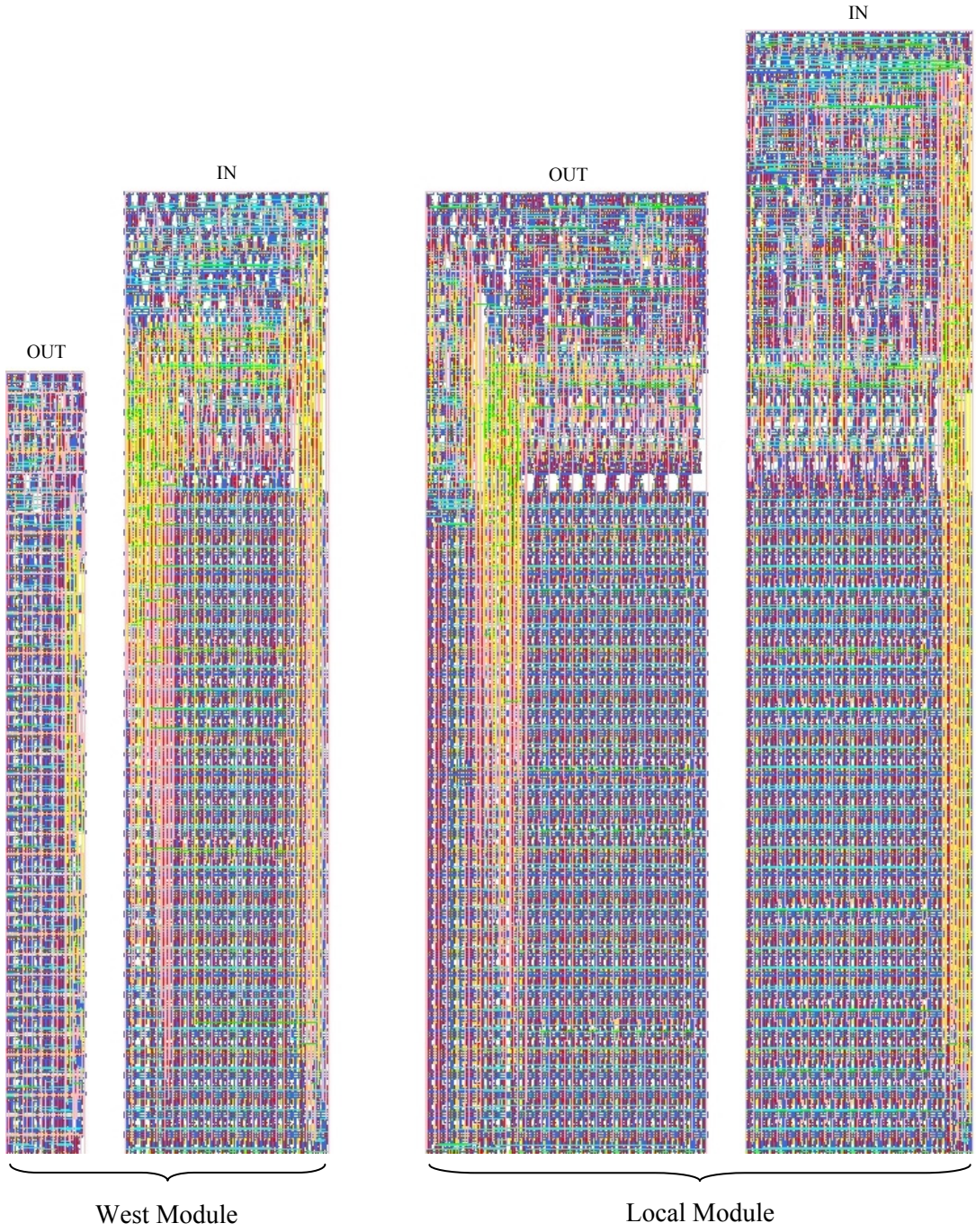


Figure 47. The Physical Layout Examples of ASPIN Modules with 8-Word FIFO Depth

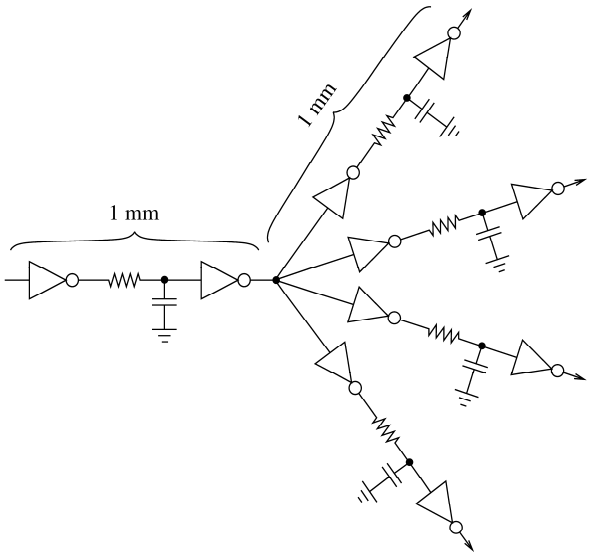


Figure 48. Long Wire RC Model

6.1 Silicon Area

The actual silicon area after physical implementation is the first important parameter. A 32-bit DSPIN router synthesized by *Synopsys* takes 40200 μm^2 for the STMicroelectronics GPLVT standard cell library in 90 nm process, as illustrated in Table 3. For ASPIN, the total silicon area of a 32-bit router is 36199 μm^2, for the same fabrication process. Note that for ASPIN we have used SXLIB, the ALLIANCE portable standard cell library [64], and in order to make this value comparable with DSPIN it is normalized. ALLIANCE is a complete set of free CAD tools and CMOS portable libraries for VLSI design. The layout libraries rely on a symbolic layout approach that provides process independence in order to allow the designers to easily port their designs from one silicon supplier to another.

Table 3. Silicon Area

	DSPIN	ASPIN
Router	40200 μm^2	36199 μm^2
Long Wire Buffers	4276 μm^2	7815 μm^2
Total	44476 μm^2	44014 μm^2

The ASPIN router area is about 10% smaller than the DSPIN area, but another factor must be accounted. In some cases, the long wires (e.g. the intra-cluster wires in DSPIN) need to be bufferized. As for long wires ASPIN uses double-rail data encoding, the area of the long wire buffers is about two times larger in ASPIN than in DSPIN. As a conclusion, ASPIN and DSPIN have nearly similar foot-print if long wires buffers are taken into account.

Perhaps it seems surprising that the silicon area of a synchronous circuit and a similar asynchronous circuit with double-rail wires are almost equal. But it is very important to notice and recall that ASPIN uses double-rail wires only for the long wires, and ASPIN's FIFOs which are the most significant area occupant are designed as optimized hard-blocks and use single-rail protocol (bundled-data encoding).

6.2 Communication Throughput

The communication throughput is the maximum number of flits transmitted per second through a router. In our implementation a flit contains a 32-bit data word. This physical parameter depends on the router micro-architecture and on the long wire effects. The communication throughput strongly impacts the end-to-end system performance and as explained in Chapter 5 it may also be related to the network saturation threshold.

The first row of Table 4 presents the Maximum Throughput for DSPIN and ASPIN routers. In case of DSPIN (the synchronous approach), this indicates the maximum clock frequency that can be used to clock the router. In case of ASPIN (the asynchronous approach), the maximum throughput is equal to the inverse of the time needed to pass a flit through the slowest pipeline stage of the router.

The first row of the table does not take into account the long wire effects. The second row presents the effect of the long wires delays, using a 2 mm wires model. These long wires delays are about four times larger in ASPIN, due to the delay insensitive Four-Phase handshake protocol. In a Four-Phase handshaking for each data transmission there are four signal transitions, consisting of REQ+, ACK+, REQ-, and ACK-.

Table 4. Communication Throughput

	DSPIN	ASPIN
Maximum Throughput	787 MFlits/S	1131 MFlits/S
Long Wire Effect	135 ps	515 ps
Applicable Throughput	711 MFlits/S	714 MFlits/S

The Applicable Throughputs, mentioned in the third row, are the final evaluations. The inverse of the applicable throughput (the time needed by a flit to traverse a router) is equal to the inverse of the maximum throughput plus the effect of the long wire delays. As said before, a 4 mm^2 cluster is a large cluster, so these throughputs are a worst case evaluation which can be considered as an applicable throughput for almost all clusters regardless of their size. Indeed the actual throughput for ASPIN varies depending on the actual cluster size, in contrast with DSPIN where the throughput is exactly equal to the employed clock frequency.

As a summary, the number of flits passing per second through an ASPIN cluster may be higher on average. On the other side in large clusters the applicable communication throughputs for ASPIN and DSPIN are approximately identical.

6.3 Packet Latency

The minimal Packet Latency is the end-to-end delay between the time a packet header enters into the first router and the time it exits the last router, assuming no contention in the network. The path through the network can be decomposed in three parts: the First router, Intermediate routers, and finally the Last router which have different latencies. Table 5 shows the packet latencies in ASPIN and DSPIN architectures.

As DSPIN is a synchronous circuit, its latency depends on the clock cycle time. Precisely, the exact value depends also on the clock skew relation between the network clock, and the subsystems clocks. For example the latency of the first DSPIN router is between the minimum latency of 3 and the maximum latency of 4 clock cycles, including the FIFO stage latencies. Note that here we focused on a DSPIN which uses bi-synchronous and mesochronous FIFOs described by Ivan Miro Panades in [11]. According to the synchronous circuit principles, DSPIN Packet Latency depends only on clock frequency.

The ASPIN Packet Latencies are given in nanosecond. However, in the last router, where an Asynchronous-to-Synchronous FIFO (detailed in Chapter 4) is instantiated, there is a clock frequency dependency between one and two clock cycles, as needed for the hardware synchronization. Packet Latency in ASPIN directly depends on the cluster size and long wire delays. Four-Phase protocol with 2 mm wires causes an extra latency of about 390 ps per cluster.

Assuming 500 MHz as the clock frequency estimation for fast and 200 MHz for slow MP-SoC subsystems in 90 nm technology, the equations below denote some example of the Packet Latencies in large 4 mm^2 clusters, where N is the number of routers in the packet transmission path.

- DSPIN Packet Latency at 500 MHz: $L = (5.00 \times (N-2) + 17.0)$ ns
- ASPIN Packet Latency at 500 MHz: $L = (1.92 \times (N-2) + 6.60)$ ns
- DSPIN Packet Latency at 200 MHz: $L = (12.5 \times (N-2) + 42.5)$ ns
- ASPIN Packet Latency at 200 MHz: $L = (1.92 \times (N-2) + 11.1)$ ns

Depending on the clock frequency the latency of the asynchronous network can be, for example, 2.5 (at 500 MHz) to 6 (at 200 MHz) times smaller. The synchronization delay at each clock boundary crossing explains why the DSPIN Latency is much higher than the ASPIN Latency. In Shared-Memory Multi-processor System-on-Chip (MP-SoC), the packet latency is one of the most important factors that defines the cost of the cache miss and is critical for system

global performance. According to the above equations, the asynchronous approach can really improve the system performance.

Table 5. Packet Latency

	DSPIN	ASPIN
First Router	3~4 T*	1.06 ns
Intermediate Router	2.5 T	1.53 ns
Last Router	4.5~5.5 T	1.76 ns + 1~2 T
Long Wire Effect	0 ns	0.39 ns

* T is the clock cycle time (e.g. 2 ns for 500 MHz clock frequency)

6.4 Power Consumption

Power consumption of the communication structure in deep submicron fabrication processes is a major concern. Although most research has focused on average power consumption or total energy consumption [67], we believe that instantaneous power consumption (or energy consumption) during one short period of time is also important for NoC characterization. In calculating the NoC power consumption, two terms must be taken into account: dissipated energy per transmitted flit and idle power consumption.

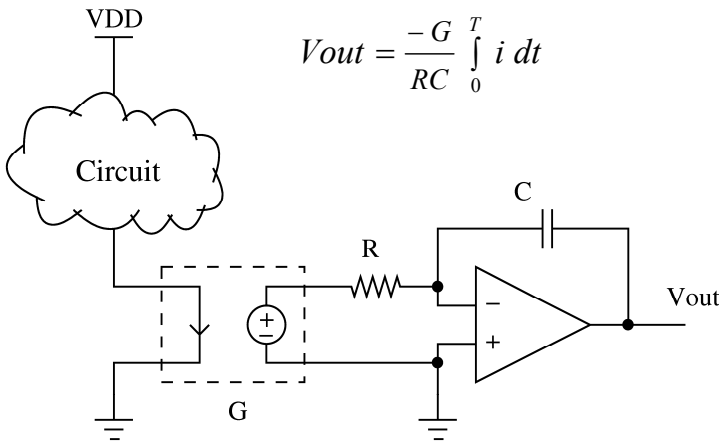


Figure 49. Current Integrator Model

To measure electrical energy consumed by the circuit in a defined period of time, we used a Current Integrator model in electrical simulations. The schematic of the proposed Integrator is shown in Figure 49. The output voltage (*Vout*) is equal to the definite integral of the instantaneous current (*i*) traversing the circuit, from the beginning of the simulation. This Current Integrator can be modeled in SPICE as follows:

```
.param One = 1v

* VDD : the power supply point to be used
vSupply VDD 0 One

*************************
*   Current Integrator   *
* VOUT : Consumed Energy *
*************************
hh1 p1 0 vSupply 1
rr1 p1 p2 1meg
cc1 p2 VOUT 1u ic=0
ee1 VOUT 0 0 p2 999k
*************************
```

As a first step, we have measured, for each router, the idle power consumption. An idle router means there is no packet to route. Table 6 presents the results. The DSPIN power consumption is 2060 µWatt at 500 MHz, using clock gating [68]. With 640 µWatt the ASPIN power consumption is about three times lower. It is well known that the clock power dissipation in synchronous designs is not negligible, even with the use of clock gating techniques.

Table 6. Idle Power Consumption

	DSPIN	ASPIN
Idle Router	2060 µWatt	640 µWatt

In the second step, the energy consumptions of two activated DSPIN and ASPIN routers have been compared. The energy consumptions have been measured for the transmission of a single packet containing five flits. Separated measurements have been done for the First, Intermediate and Last routers. We have executed the measurements with four different hypotheses, depending on two parameters. The first parameter is the packet content. All flits in the packet can have a constant value, or all bit values change between two successive flits. The second parameter is the long wires effect and the corresponding electrical energy dissipation. The energy consumption results for a clock frequency of 500 MHz is summarized in Table 7 when the power dissipation effect of long wires is not taken into account, and in Table 8 when this effect is concerned. In these tables, N is the number of routers in the packet path.

In the asynchronous Double-Rail Four-Phase handshake protocol, one of the two rails of each bit goes to logic One and Return to Zero (RTZ), whatever the bit content (zero or one). Consequently, ASPIN energy consumption is nearly independent on the packet content. In small clusters, where the effect of long wires is insignificant, DSPIN and ASPIN consume approximately the same amount of energy to transfer one packet. When the long wire effect is taken into account, the energy required by DSPIN to transfer a packet with constant content remains almost at the previous value, but if the packet has an alternate content, energy

consumption increases. As expected, the long wire effect on ASPIN energy consumption is much more dissipative.

Table 7. 5-Flit Packet Energy Consumption (pJ) without Long Wire Effect

	With Constant Content		With Alternate Content	
	DSPIN	ASPIN	DSPIN	ASPIN
First Router	37	27	43	34
Intermediate Router	36	33	42	41
Last Router	36	48	42	62
Transmission Path	36×(N-2)+73	33×(N-2)+75	42×(N-2)+85	41×(N-2)+96

In a typical shared-memory multi-processor system using a Best-Effort packet-switching Micro-Network, the average activity of the routers is rather low, and most of the time, the routers are idle. According to a typical average bandwidth utilization of 20% (per input port of the router), the average power consumption is not significantly different between DSPIN and ASPIN routers.

Table 8. 5-Flit Packet Energy Consumption (pJ) with Long Wire Effect

	With Constant Content		With Alternate Content	
	DSPIN	ASPIN	DSPIN	ASPIN
First Router	50	124	83	131
Intermediate Router	45	129	81	137
Last Router	45	147	81	161
Transmission Path	45×(N-2)+95	129×(N-2)+271	81×(N-2)+164	137×(N-2)+292

6.5 Summary

A systematic comparison between physical performance parameters of two different implementations of the same micro-network architecture has been presented. The DSPIN implementation is a multi-synchronous, and the ASPIN implementation is a fully asynchronous NoC. The related parameters are silicon area, packet latency, communication throughput, and power consumption. These electrical characteristics have been evaluated by post layout SPICE simulation for STMicroelectronics 90 nm GPLVT CMOS fabrication process. As a predominant factor, in the evaluations the long wires effects have been taken into account.

- Regarding the silicon area, both implementations have similar foot-prints, if long wire buffers are taken into account.
- In systems containing large clusters, the energy dissipated to transmit a packet is higher in the asynchronous approach than in the synchronous approach, but the idle power consumption is 3 times lower. Consequently, the difference of the average power consumption is expected to be insignificant for typical shared memory MP-SoCs.
- The maximal bandwidths are approximately similar: about 700 MFlits/S for the synchronous approach, against 700 to 1100 MFlits/S (depending on the cluster size) for the asynchronous approach.
- The packet latency is clearly the strong point for the asynchronous approach, as the latency, for example, is 2.5 (at 500 MHz) to 6 (at 200 MHz) times smaller for ASPIN than for DSPIN.

As a general conclusion, silicon area, power consumption and communication bandwidth have approximately similar values, but the asynchronous implementation's average packet latency (which is a crucial parameter) is smaller.

Summary

A novel asynchronous NoC architecture (called ASPIN) to cope with the issues in the Globally Asynchronous Locally Synchronous paradigm and to reduce the possibility of metastability along the packet path has been presented. A large number of locally synchronous islands can communicate together via an asynchronous network and thereby the crucial problem of distributing a global synchronous clock signal over a wide chip area is reduced to a number of smaller sub-problems. An asynchronous NoC limits the synchronization failure only at the network interfaces, where the synchronous data has to enter into the asynchronous network and the asynchronous data into the synchronous subsystems.

Arranged in a two-dimensional mesh topology, ASPIN is a wormhole packet-switching network. To route packets ASPIN uses the distributed, deadlock-free X-first routing algorithm. The starvation-free Round-Robin algorithm is used to schedule the concurrent requests.

ASPIN addresses also the crucial issue of the global long wires which likely have large propagation delays, incompatible with the required throughput. ASPIN's router is not a centralized macro-cell. The router is split in five separated modules (North, South, East, West, and Local) that can be physically distributed over the cluster area. This feature allows balancing the length of the long wires. Guaranteeing a delay-insensitive communication, ASPIN uses the double-rail four-phase handshake protocol for long wires.

Seeing that in a router the storage places (FIFOs) are the most important area occupant, in order to minimize the silicon area overhead the asynchronous FIFO, instantiated in ASPIN's input ports, has been designed as an optimized hard-block using single-rail (bundled-data) encoding.

At clock boundaries ASPIN uses two special FIFOs to connect synchronous IP cores to the asynchronous network. Accordingly, two new concepts for Synchronous-to-Asynchronous FIFO (SA_FIFO) and Asynchronous-to-Synchronous FIFO (AS_FIFO) have been described in details.

All designs have been physically implemented, and the electrical characteristics have been evaluated by post layout SPICE simulation. These architectures are rather generic and allow the system designer to make various trade-offs between latency and robustness, depending on the selected synchronizer. With any choice of synchronizer, these FIFOs can achieve the maximum throughput of one transmitted word per clock cycle.

Furthermore, we illustrated the evaluation of the network saturation threshold in the ASPIN and DSPIN architectures. DSPIN (which has been briefly presented in this thesis manuscript) is a multi-synchronous NoC well-suited to the GALS paradigm. Actually ASPIN is the asynchronous implementation of DSPIN. In the saturation threshold evaluation the influence of two parameters has been considered: the flit storage capacity of the network and the network throughput.

This thesis showed that the ratio of the network throughput to the flit injection rate has a direct influence on the network saturation threshold, and as a consequence the network saturation threshold in a fast asynchronous network is much higher than in a multi-synchronous network which works with the speed of subsystems.

Additionally this PhD thesis justified the fact that in a fast asynchronous network the accumulation of flits in the two ends of a packet path (i.e. in SA_FIFO and AS_FIFO) is more efficient than distributing them over the network (i.e. in the intermediate FIFOs). As a result, the corresponding cost of increasing the storage capacity distributed in the network (which can improve the network saturation threshold) may be much cheaper for a high-speed asynchronous network than for a multi-synchronous one.

We proposed a new method to improve the saturation threshold in fast and large asynchronous networks: using a Quasi-Store-and-Forward (QSF) algorithm instead of end-to-end wormhole routing. In this approach, all flits of a given packet accumulate in the asynchronous form before entering the network. With this QSF approach, the packet does not occupy any resources unless it is sure that all its flits can traverse the network at full speed. In fact QSF approach attempts to realize a network offered load reduction by a coefficient equal to the speed ratio.

Finally, a systematic comparison between physical performance parameters of DSPIN and ASPIN architectures has been presented. The relevant parameters are silicon area, packet latency, communication throughput, and power consumption. The electrical characteristics have been evaluated by post layout SPICE simulation for STMicroelectronics 90 nm GPLVT CMOS fabrication process. As a predominant factor, the long wires effects have been taken into account. As DSPIN and ASPIN are two different implementations of the same micro-network architecture, the performance comparison may help to answer this question that which architecture type could be the best choice to implement, synchronous or asynchronous?

- Regarding the silicon area, both implementations have similar foot-prints, if long wire buffers are taken into account.

- In systems containing large clusters, the energy dissipated to transmit a packet is higher in the asynchronous approach than in the synchronous approach, but the idle power consumption is 3 times lower. Consequently, the difference of the average power consumption is expected to be insignificant for typical shared memory MP-SoCs.
- The maximal bandwidths are approximately similar: about 700 MFlits/S for the synchronous approach, against 700 to 1100 MFlits/S (depending on the cluster size) for the asynchronous approach.
- The packet latency is clearly the strong point for the asynchronous approach, as the latency, for example, is 2.5 (at 500 MHz) to 6 (at 200 MHz) times smaller for ASPIN than for DSPIN.

Appendix A

Networks-on-Chip

"…the first microprocessor only had 22 hundred transistors. We are looking at something a million times that complex in the next generations—a billion transistors. What that gives us in the way of flexibility to design products is phenomenal." said Gordon E. Moore, a co-founder of Intel Corporation. The man whose prediction, now popularly known as Moore's Law, states that the number of transistors on a chip doubles about every two years. It is sometimes quoted as every 18 months. On this basis the chip density increases 10-fold every 5 years. See Figure 50.

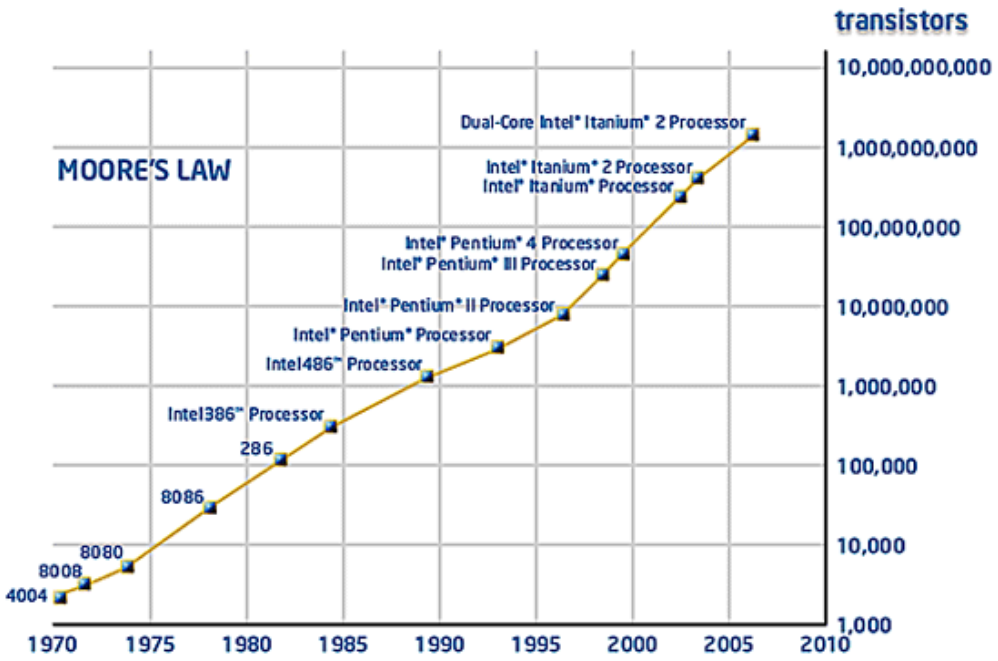


Figure 50. Moore's Law means more Performance (taken from www.intel.com)

Transistor feature size is expected to continue to shrink. A reduction of 70% in linear dimensions of transistors by moving to a new fabrication process (for example from 65 nm to 45 nm) allows a 2-fold increase in the chip density. New technologies such as 3D die stacking may allow even greater increases in total transistor count within a given footprint [69]. 3D die stacking is an exciting new technology that increases transistor density by vertically integrating two or more die with a dense, high-speed interface. The 2005 International Technology Roadmap for Semiconductors (ITRS) [66] projects that multi-billion transistor chips will come to production by the end of this decade. Refer to Figure 51.

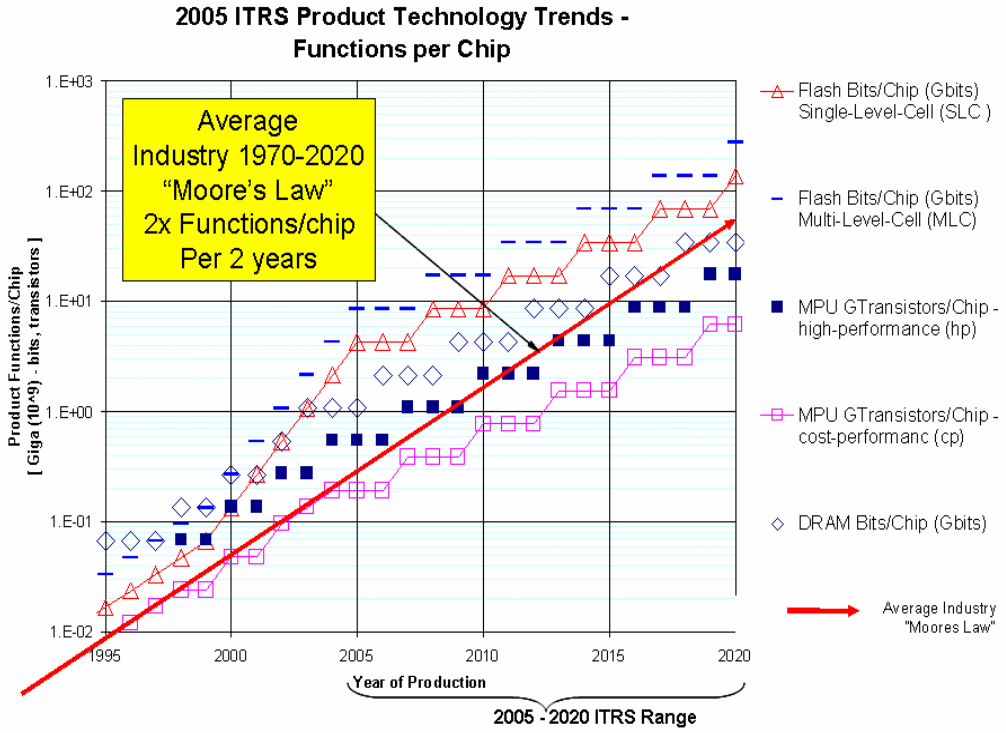
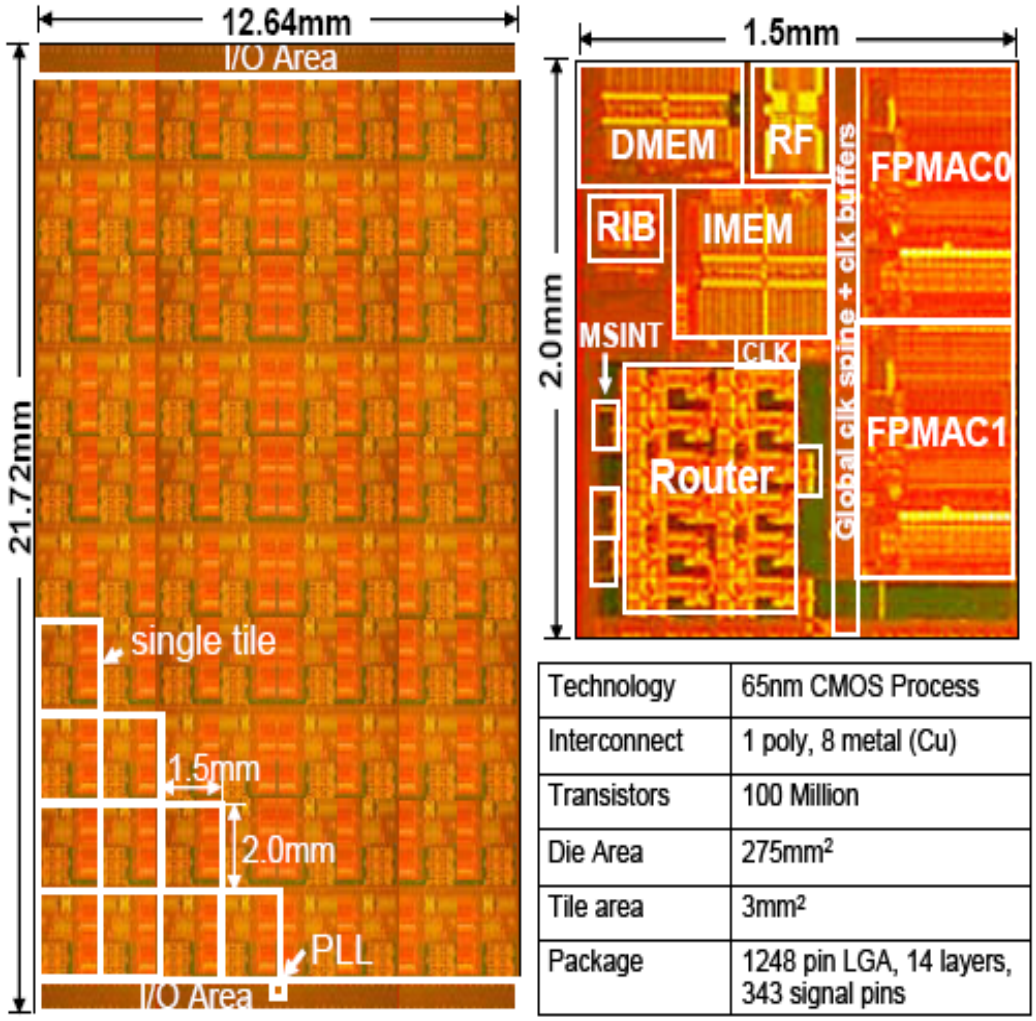


Figure 51. Product Technology Trends (2005 ITRS)

A.1. Multi-Core Processors, Clients for NoCs

As we proceed into deep submicron (DSM) technologies, feature sizes will not only introduce a whole new set of application possibilities but will also aggravate current problems in VLSI/ULSI design, and moreover, introduce several new ones. Recently the Intel Tera-Scale Computing [70] research team in a white paper [71] has mentioned that "power thermal issues, such as dissipating heat from increasingly densely packed transistors, have begun to limit the rate at which processor frequency can also be increased. Although frequency increases have been a design staple for the last 20 years, the next 20 years will require a new approach. Basically, industry needs to develop improved micro-architectures at a faster rate and in coordination with each new silicon manufacturing process, from 45 nm, to 32 nm, and beyond".



Technology	65nm CMOS Process
Interconnect	1 poly, 8 metal (Cu)
Transistors	100 Million
Die Area	275mm²
Tile area	3mm²
Package	1248 pin LGA, 14 layers, 343 signal pins

Figure 52. Intel's 80-Tile Network-on-Chip in 65nm CMOS

As a consequence, Intel processors with two processing cores (Intel Core Duo, Core 2 Duo, and Xeon x1xx series) have been commercially produced and quad-core processors (Intel Core Quad and Xeon) are bringing out. Intel has explained that "instead of focusing solely on performing individual tasks faster, we will execute many more tasks in parallel at the same time. We will also distribute those tasks across a grouping of cores that work in a coordinated fashion". Intel also said that "the number of cores on a chip will continue to grow, launching an era of vastly more powerful computers. These are the machines that will deliver teraflop performance with the efficient capabilities needed to handle tomorrow's emerging applications." Intel has mentioned three distinct trends that motivate this shift:

- **Performance:** We can no longer simply increase the clock frequency at the same rate as we have in the past in order to increase performance. Power and thermal requirements are beginning to outstrip the benefits that faster clock frequencies offer. However, because the trajectory of Moore's law will continue well into the next decade, we expect to

continue doubling transistors every 18-24 months for the next several years. Parallel execution in multi-core designs will then allow us to take advantage of these greater transistor densities to provide greater performance.

- **Power consumption:** Many simple cores can be built within the same area as a small number of large complex cores. In addition, power consumption can be optimized by using multiple types of cores tuned to match the needs of different usage models. Also, cores that are not busy can be powered down to reduce power consumption during idle times. These advanced power-saving techniques are enabled by multiple cores working in a coordinated fashion.
- **Rapid design cycles:** Building tera-scale processors from standard, repeated tiles that are then integrated into a common infrastructure should allow more reuse of designs between generations of processors. This will also allow us to highly optimize the design of these tiles to further improve power utilization and performance.

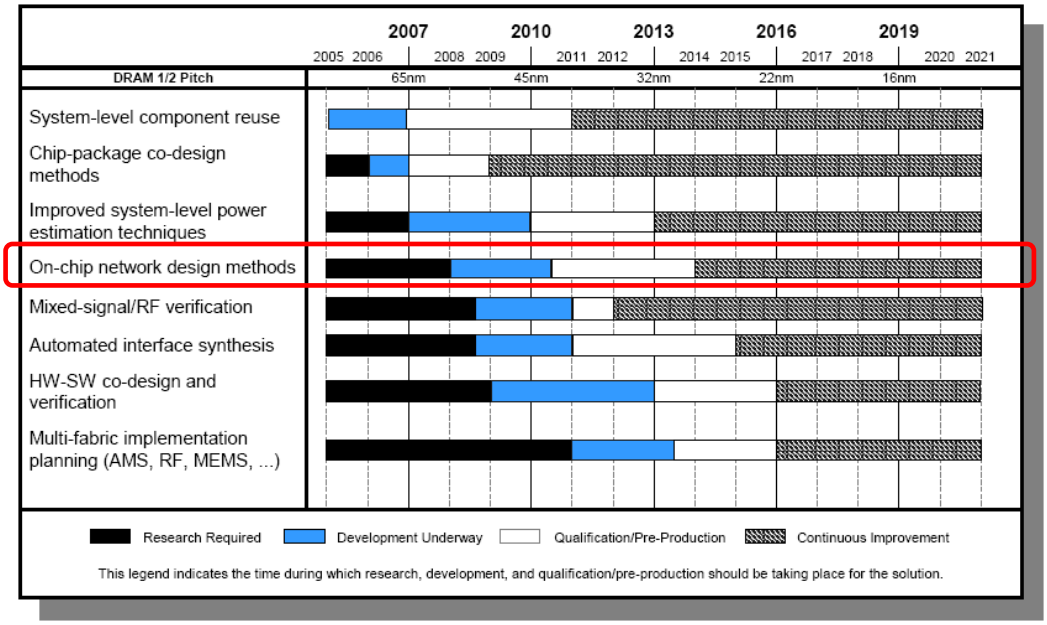


Figure 53. System-Level Design Potential Solutions (2005 ITRS)

Intel's tera-scale research prepares for tens, hundreds of cores [72]. As a testimonial, an 80-Core processor that delivers supercomputer-like performance of 1.28 Teraflops, and use only 181 W, has been introduced in [73]. The chip architecture contains 80 tiles arranged as a 10×8 two-dimensional mesh topology. See Figure 52. Although such tera-scale architectures offer many unique opportunities through their highly integrated multi-core designs, obviously they also present many challenges. For example, consider inter-core communication latencies and bandwidth. Basically as more elements are packed onto a tera-scale chip, there is a correspondingly greater need for each of these units to communicate with each other. The global

interconnect design and micro-network architecture will play a significant role in determining the performance. The basic architecture of Intel's 80-Tile design is based on the Network-on-Chip ideas. Each of these 80 tiles consists of a processing engine connected to a 5-port wormhole packet-switching router for passing data amongst the tiles with a bandwidth up to 256 GB/s.

Even though Intel has indicated that there is no plan to bring this chip to market, without a doubt something like this will be produced in the coming years. Perhaps Multi-Core Processors (Chip-level Multi-Processor or CMP) are one of the first real industrial clients of NoCs. As demonstrated in Figure 53, the 2005 ITRS forecasts that by the end of this decade development and pre-production will be taking place for Network-on-Chip design methods.

A.2. Systems-on-Chip, the Real Clients of NoCs

Monstrous efforts in reducing the transistor feature size, have given the possibility of using a huge number of transistors in a single chip. Today's chips may contain several microprocessors, physical memories, peripheral controllers (such as USB controllers), DSPs, etc. Therefore, a chip could be a set of various subsystems working together as different parts of a large system, hence called System-on-Chip. Systems-on-Chip (SoCs) provide the feasibility of a wide range of applications. Real-time requirements could be achieved and massively parallel processing is a reality. Furthermore, to address high-performance computation Multi-Processor Systems-on-Chip (MP-SoCs) are developed. The 2005 ITRS predicts that future applications, by the end of the next decade, will exploit more than 800 processing engines! See Figure 54.

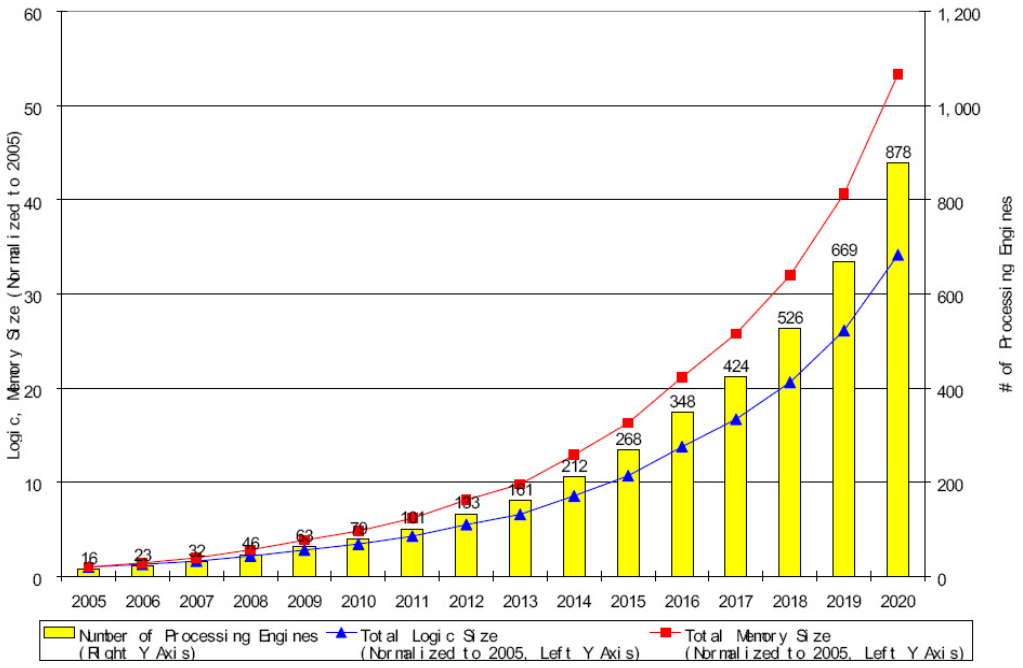


Figure 54. SoC Design Complexity Trends (2005 ITRS)

In such multi-processor (multi-core) applications, inter-core communication efficiency is surely an essential key to the overall system performance. So due to the growth of performance requiring, an efficient on-chip communication seems more indispensable than ever. In the recent years on-chip communication interconnection has become a broad topic of research and development. Communication and computation are now two completely distinct research aspects in SoC design. Networks-on-Chip, as a new SoC paradigm [74], is known as a prominent concept for on-chip communication. In fact, NoCs are the progressive evolution of classical shared busses. As will be clear in the following subsections, traditional communication interconnects do not scale when the transistor feature size reduces and/or number of components to interconnect increases.

A.2.1. Scalability

The opportunity of task-level parallelism between processing units in MP-SoCs has offered the possibility of a wide range of different applications accumulated in a single chip. Mobile Phones are exemplary of such high-performance systems. With the same equipment, we can talk to a friend, watch a movie, explore the Internet, hear a piece of music, play a game, or take a picture.

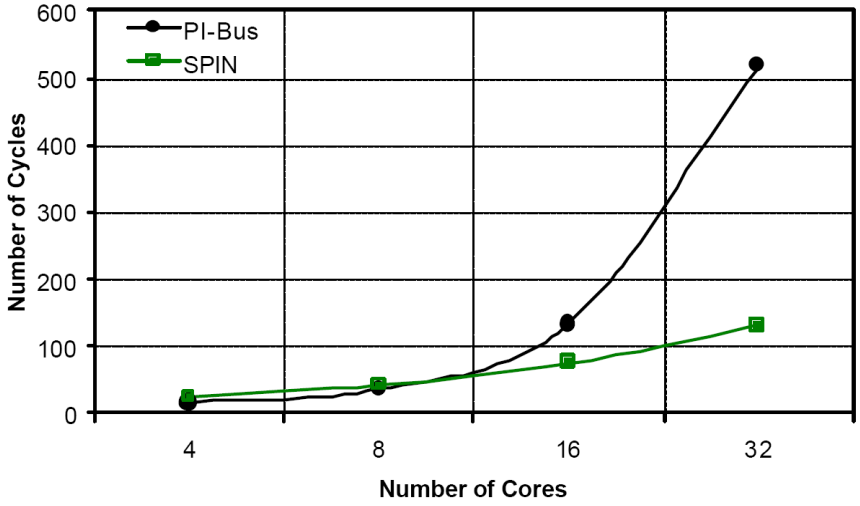


Figure 55. The Average Latency of PI-Bus and SPIN for several numbers of cores

This requirement will continue to grow with a better quality. The need for more number of high-performance hardware Intellectual Property (IP) cores incorporated on a single chip has emerged. However, classical on-chip interconnects cannot support anymore this type of applications. They don't scale because they can only serve a limited number of cores generating traffic. As an example, Figure 55 (taken from [75]) illustrates the average latency of two on-chip interconnects versus number of connected IP cores. The first interconnect is PI-Bus [76], an on-chip shared bus, and the second one is SPIN, LIP6's Network-on-Chip. SPIN outperforms PI-Bus only in systems with more than a dozen of cores.

A.2.1.1. Concurrent Communication

Indeed the communication throughput of large-scale applications like multi-media mobile phones plays a key role in overall performance. The chip may require an aggregate interconnection bandwidth of several hundred gigabits per second. The possibility of a large number of simultaneous inter-core communications in a NoC-based architecture could fulfill this requirement. In classical interconnects such as busses the bandwidth is shared by all attached devices and concurrent communications is not possible.

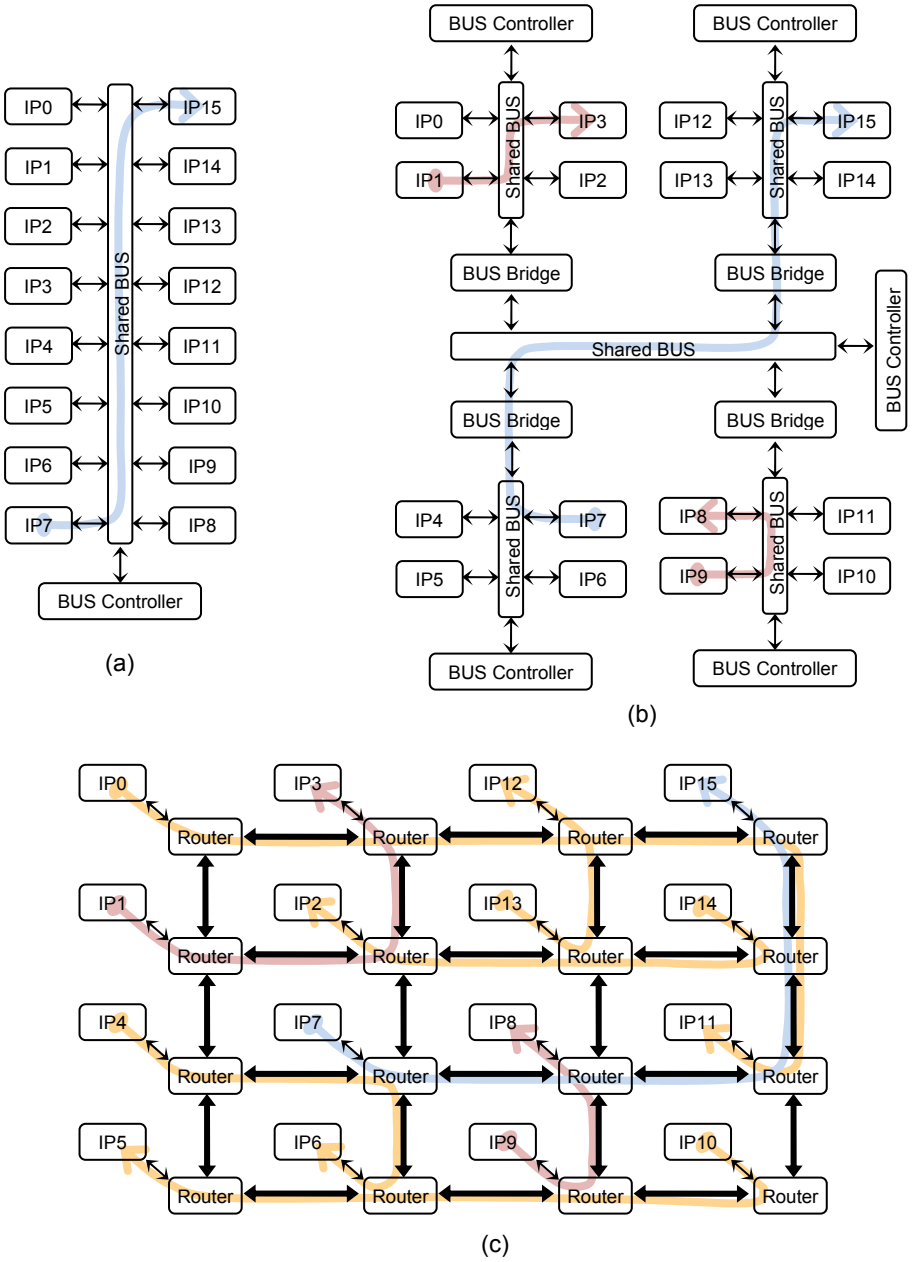


Figure 56. Concurrent Communications in (a) Bus, (b) Hierarchical Bus, and (c) NoC

In order to provide concurrent communications the advanced bus-based approaches recommend multiple and hierarchical on-chip busses. These architectures require case-specific grouping of IPs and the design of traversal bridges with an inherent complexity in arbitration algorithms, which does not make for a truly scalable and reusable interconnection. Due to the tight time-to-market, the success of Systems-on-Chip will rely on the ability to interconnect existing components in a plug-and-play fashion.

In reality Networks-on-Chip are the result of the gradual process of the reusability and scalability improvement in hierarchical interconnect architectures. Wires became pipelines and bridges became routers. Figure 56 shows the growth in number of parallel communication accesses in on-chip interconnecting. In a shared bus (a) when two IP cores are communicating, for example according to the figure, IP7 with IP15, no other IP can begin a new connection. On the contrary, due to the hierarchy of local busses in (b), in addition to the data communication between IP7 and IP15, there is the possibility of some others like IP1 with IP3 and IP9 with IP8. In a NoC (c) the number of concurrent communications vastly increases and in parallel with the above communications, the other IPs like IP0, IP4, IP13, IP14, and IP10 could communicate, for example, with IP11, IP6, IP12, IP2, and IP5, respectively.

A.2.1.2. Saturation Threshold

A well-known behavior of traffic systems with flow control is that when the requested throughput approaches to a certain limit, throughput saturates. It can be explained by looking at the road transportation. When number of moving vehicles is not high and the traffic load is moderate, one can determine how long is required to arrive to the destination. But in a heavy traffic situation, it is not easy to predict the delay. The average speed will be reduced and a large latency may be imposed. In this case due to the high possibility of collision, the use of a flow control system is essential. For example in order to avoid contention in road junctions, traffic lights give the cars the right of way. The largest relative part of the total latency is due to the waiting times at red lights. The absence of simultaneous access ways and limited parallelism is often a cause of congestion.

Data communication in computer systems shows a similar behavior. When number of components requiring access to the communication resources is limited, average communication latency is predictable. But when the offered load increases, the average latency becomes a bottleneck for system performance. The average latency exponentially grows to an infinite value, when the offered traffic load exceeds a point called saturation threshold. Scalability in NoCs means that the value of saturation threshold is roughly independent on the number of communicating units. The main motivation supporting NoC paradigm is the fact that classical interconnects such as shared busses have a low saturation threshold and do not scale with the number of connected units. They can serve a limited number of units, and beyond that the average communication latency strongly increases. Figure 57 (taken from [75]) illustrates the average

latency of PI-Bus and SPIN micro-network versus the offered load. It appears that with 32 connected IP cores, SPIN micro-network has a latency of about 30 cycles and saturates for an offered load of 28%. With the same cores, for very small load, PI-Bus has a lower latency, but the saturation threshold is for an offered load of about 4%.

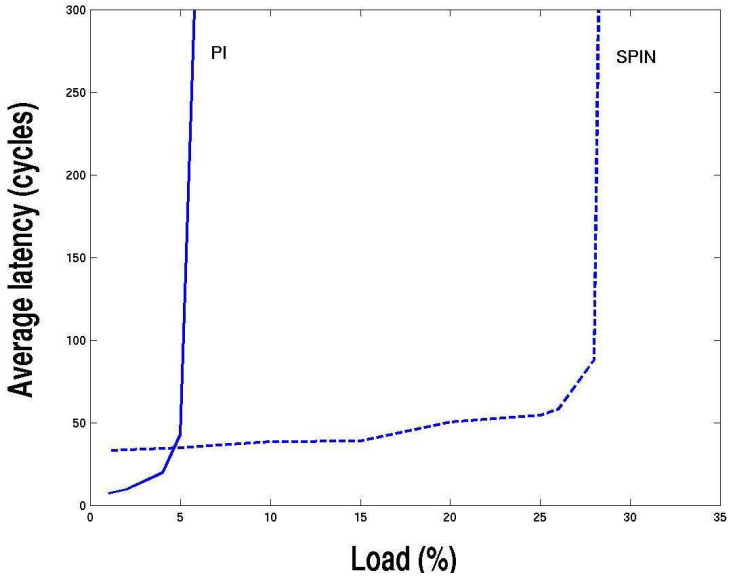


Figure 57. The Saturation Threshold of SPIN vs. PI-Bus

A.2.1.3. Quality of Service

However, even though the saturation threshold in Networks-on-Chip is much higher than in previous interconnect, any interconnect will saturate if a large number of cores generate traffic and the average offered load exceeds saturation threshold. As the network latency becomes unpredictable, the quality of some delay sensitive applications which need a guaranteed latency will be reduced. For example a data stream from a camera to an MPEG encoder in a high quality video application requires high throughput with low, stable, and predictable delay. If the time interval between frames goes beyond a certain limit, the quality of service of the application could not be guaranteed. NoCs providing connection-oriented service (TDM: Time Division Multiplexing, VC: Virtual Channels, etc) give an opportunity for this kind of real-time applications. The ability of a network to provide guaranteed service to specific connections is often denoted as QoS or Quality-of-Service. The primary goal of QoS is to produce guaranteed throughput and latency, even when the network traffic reaches saturation point.

A.2.2. Physical Issues

As SoC complexity scales, it will be more difficult, if not impossible, to capture their functionality with fully deterministic models of operation [77]. In other words, system models

may have multiple implementations. Property abstraction, which is a key to managing complexity in modeling and design, will hide implementation details and designers will have to relinquish control of such details. Architecting at a higher abstraction level is now a key factor for system design. While abstract modeling and automated synthesis enables complex system design, such an approach increases the variability of the physical and electrical parameters.

A.2.2.1. Wire Delay

The shrinking of processing technology in the deep submicron domain aggravates the imbalance between gate delays and wire delays on chip. While gate delays decrease, wire delays, because of the growth of wire resistance, increase [60]. Nevertheless, since the length of local wires usually shrinks with traditional scaling, the impact of their delay on performance is minor. On the contrary, as the die size does not necessarily scale down, global wire lengths do not reduce. Global wires connect different functional units of a system and spread on the entire chip. The largest part of delays now is related to global wires. Figure 58 (taken from the 2005 ITRS) shows the projected relative delay for local wires, global wires, and logic gates of the near future. Whereas the operating frequency and transistor density need to continue to grow, global wires are likely to have propagation delays largely exceeding the required clock period.

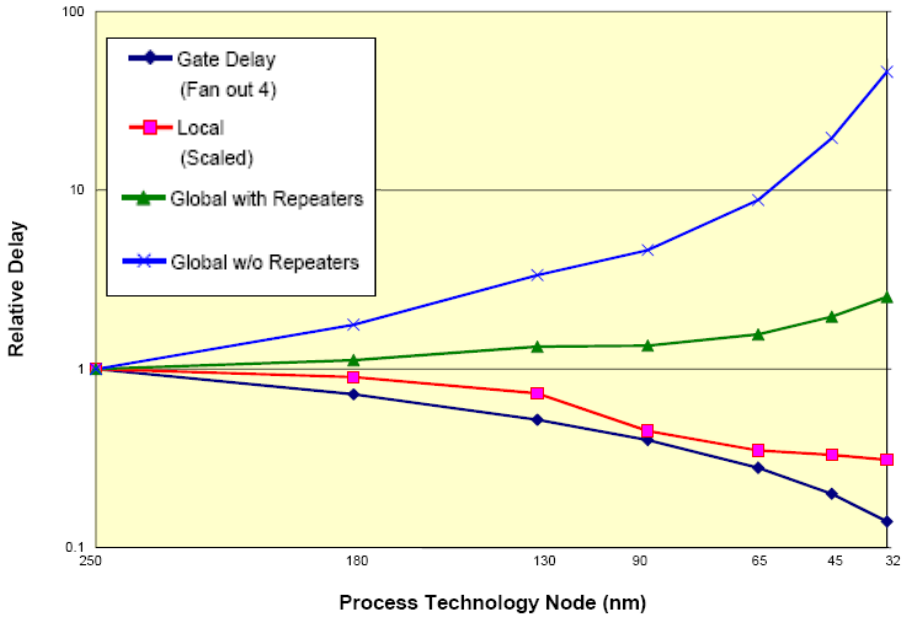


Figure 58. Delay for Local (Metal 1) and Global Wiring versus Feature Size

Furthermore, as the number of components attached to shared wires increases, the load capacitances and consequently signal transition delays grow, degrading the operating frequency. A correct design that safely meets all timing constraints will require knowing the signal delays with a reasonable accuracy. Accurate physical design became the bottleneck for design closure.

All told, due to the physical wire effects of scaling, differentiating between local and global interconnection has become absolutely necessary and the need for global communication architectures supporting Systems-on-Chip requirements has emerged. In NoC design approaches the global wires are replaced with segmented pipelined wires (point-to-point links) connecting network nodes. Each functional unit in the SoC is often connected to a node in the network.

A.2.2.2. Signal Integrity

In addition to traditional applications, such as aircraft control, defense applications, and reliable computing, there are many new fields requiring high-reliable SoCs, ranging from medical applications to automotive control and more generally to embedded systems that are critical for human operation and life. System-level reliability is the probability that the system will operate correctly as a function of time. The expected value of the reliability function is the Mean Time To Failure (MTTF). Increasing MTTF more than the expected useful life of a product is an important design specification.

In DSM technologies, due to problems like fabrication uncertainties, crosstalk, noise sensitivity, temperature variation, etc, the wire models (especially long wires) are unreliable and failures in devices and interconnects are more probable to happen [77]. Therefore reliable SoCs need to be designed with specific abilities to deal with hard (permanent) and soft (transient) malfunctions. System-level solutions for hard errors involve redundancy, and thus require the online connection of a reserved unit and disconnection of the faulty unit. Solutions for soft errors include design techniques for error detection and correction. Networks-on-Chip can provide flexible solutions toward hard errors by supporting reconfigurability and seamless connection/disconnection of units. Also NoCs, due to their layered architectures, can detect and correct soft errors by layered error detection/correction methods.

A.2.2.3. Power Dissipation

Most of today's electronic devices are confronted with the problem of delivering high performance with limited power consumption. Low power consumption is required to achieve acceptable autonomy in battery-powered systems such as PDAs (Personal Digital Assistants) or mobile phones. Moreover low-power circuits reduce the environmental impact (e.g., heat dissipation, cooling-induced noise) and operation cost of stationary systems. In other words, achieving highly energy-efficient computation is a major challenge in electronic design.

Regarding the current projections of future silicon technologies, heat extraction and energy dissipation, where a major contribution is due to leakage, will lead the SoCs to incorporate Dynamic Power Management (DPM) techniques in various forms to satisfy energy consumption bounds [78]. DPM contains a set of techniques that achieve energy-efficient computation by selectively turning off or reducing the performance of system components when they are idle or

partially unexploited. In these methods the need for physical distinction between system power segments has emerged. Modular and reconfigurable NoCs by clusterizing the chip help to independently provide ideal boundaries to power domains [79].

A.3. Design Methodologies

In NoC literature the term of methodology relies on a very broad design aspect, from low-level physical implementation to high-level abstract modeling. Problems get different senses as we move from one design level to another. For example system software issues do not involve physical problems like clock skew or long wire delays, as the physical layer has no idea about handling massive parallelism.

As a consequence, the OSI (Open System Interconnection) model of layered network communication can be adapted for NoC usage [74]. The aim is to shield each level of the system from issues of other levels and thereby to allow communication between independently developed layers. Note that awareness of lower levels can be beneficial as it can lead to higher performance. In a NoC-based system the layers are more closely dependent than in a macro network and have often a physically related characteristic. The design of a system using Network-on-Chip architecture could be considered in four different layers as follows:

- **Physical** (OSI: Physical)**:** This layer is concerned with the lowest-level details of transmitting data on wires. It defines signal timings and all related problems like long wire delays, clock skew, and synchronization failure, which all are aggravated as technology scales down. Power consumption is one of the most important parameters at this level. Globally Asynchronous Locally Synchronous (GALS) design issues are also related to this layer of the network. Implementing the system as a synchronous circuit or an asynchronous one is a design decision that must be made due to the encountered parameters of the physical layer.
- **Network** (OSI: Data Link, Network)**:** The network layer provides a topological view of the communications. The main function of this layer is to determine how messages are routed from a source to a destination. This can be customized by the choice of routing algorithm, switching strategy, and flow control, which all have a great impact on the performance. Dead-lock, live-lock, starvation, and saturation are some critical issues introduced in this layer.
- **Interface** (OSI: Transport)**:** Dealing with the decomposition of messages into packets at the source and their assembly at the destination is the main function of the interface layer of a NoC. As the behavior of most network flow control policies is sensitive to packet size, packetization is a critical design decision. Furthermore this design level offers the independency versus the network implementation and translates the communication protocol of upper levels to the network compliant protocol.

- **Application** (OSI: Session, Presentation, Application)**:** The application layer is concerned as the highest level of abstraction of the underlying communication architecture. It provides application specific functions exploiting the capabilities of lower level components, and thus the system can use these abstract communication functions without any concern for network details. All system software issues are related to the application design level of a Network-on-Chip.

Figure 59 shows the layered-designed components of a NoC-based architecture. Typically in a general shared-memory MP-SoC each subsystem contains one or several processors, one or several physical memory banks, optional dedicated IP cores (Hardware Coprocessors, I/O Controllers …), etc, which may communicate together via a local interconnect. As shown all these components are considered in the application layer of the system.

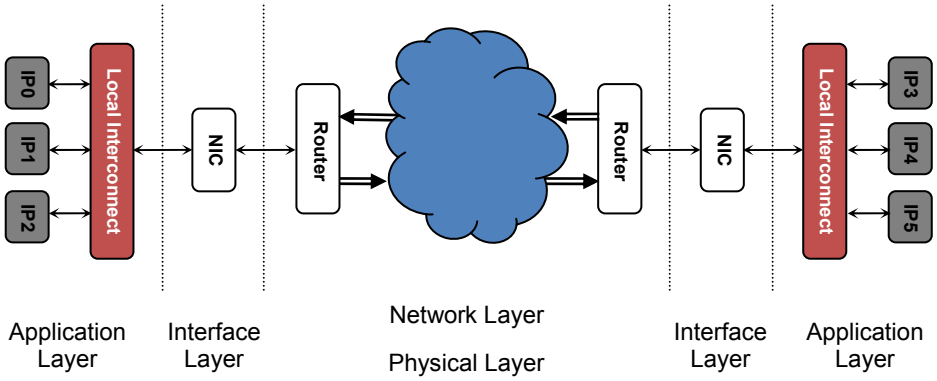


Figure 59. Layered-designed components of a NoC-based architecture

The subsystems are connected to the network by a Network Interface Controller (NIC) which is the only access way. The NIC is involved in the issues of the interface layer. Thanks to this component, even if the architecture is physically clusterized, all processors in all clusters may share the same flat address space and any processor in the system can address any target or peripheral IP core.

A network consists of a number of switches and so the network layer problems include the design of the network switches and their connectivity. Precisely, the network level of design defines the concept of the switches and determines the geometric topology of the system. The switching module of a NoC is often called router and must be implemented according to the physical issues. Thus, a router may belong to both network and physical layers of the system. Distinguishing these two layers as two separated components is not evident.

The focus of this section is mainly on incorporating the ideas of the network architecture implementation. Naturally defining the basics of a NoC-based architecture lies on methodologies of the router design, i.e. physical and network design-levels which are two network architecture

dependent layers. Sometimes the interface layer could also be concerned, as needed to form an interface between the network architecture and the application layer components.

The system performance is fundamentally affected by the network architecture which provides the communication infrastructure for the resources. The possibility of a large number of concurrent communications and a high aggregate throughput, a high value of saturation threshold, a low average latency, absence of dead-lock, live-lock, and starvation, providing in-order-delivery property, etc, all are directly influenced by the architectural issues.

Relating to the network architecture, a crucial problem is the trade-off between generality and performance. Generality provides reusability of hardware, operating systems and development practices, while performance is achieved by using application-specific structures [38]. Nevertheless according to the fact that time-to-market is being very tight and design cost a nightmare, for example, LIP6 has decided to design a high performance, but general-purpose NoC architecture that could be used as a pre-fabricated part of any MP-SoC designs, needed by the future design methodology of high-level abstract modeling.

The NoC bibliography, as a conceptual guideline, leads to update the network architectural ideas. SPIN micro-network presented in [33] was the first published NoC architecture. After that a large number of Networks-on-Chip have been proposed. Some examples are Dally's NoC [34], AEthereal [35], xPipes [37], CLICHÉ [38], Nostrum [36], aSoC [43], ANoC [52], QNoC [50], Octagon [39] (Spidergon [40]), Nexus [51], SoCBUS [41], Chain [48], QoS [49], SoCIN [44], HERMES [42], BFT [45], BONE [46], Proteo [47], and MANGO [53]. To better classify NoC methodologies, the issues could be identified by four key properties: topology, routing algorithm, switching strategy, and flow control.

A.3.1. Topology

The network topology is the study of the arrangement and connectivity of the nodes. The arrangement of the nodes in a geometrical shape and their connectivity determine the physical topology of the network. Likewise the mapping of the flow of data between nodes in the network defines the logical topology that could be dynamically configured. Although usually physical and logical topologies are identical, in any particular network they also may be different. Anyway both physical and logical topologies follow the same classifications.

The choice of a network topology opts for three aspects which are often mutually exclusive: performance, cost, and scalability. Typically a designer would have to make calculated trade-offs between these three aspects in order to find the optimum solution for each individual NoC design. Shared-Bus, Crossbar, Binary Tree, Fat-Tree, Butterfly Fat-Tree, Ring, 2D-Ring (Torus), Chordal Ring, Array, and 2D-Array (Mesh) are the most usual topologies for on-chip interconnecting. As stated in [80] and [81], these different topologies formally can be compared by analyzing some critic properties. According to the requirements of the NoC-based systems and the graph theory,

total number of switches, the degree of each switch, the diameter of the network, total number of channels, and number of bisection channels are the most important parameters for a formal comparison. These parameters are explained as follows:

- **Number of switches:** determines the number of switches which are required to realize the topology. Lower number of switches means lower network overhead.
- **Switch degree:** is the number of input/output ports. Lower degree for switches means reduced design complexity and consequently simpler circuit with lower silicon area and power consumption.
- **Network diameter:** is the length of the maximum shortest path between any two nodes measured in hops. Hop is the basic switching action of the network. Lower diameter for the network means the possibility of lower communication latency.
- **Number of channels:** determines the maximum possibility of the simultaneous communications in the network. Higher number of channels means higher aggregate throughput and lower saturation threshold.
- **Number of bisection channels:** is the minimum number of channels which, if removed, cut the network into two equal parts. Bisection channels are a measure of bottleneck channels that could be used with members of one of the sub-nets communicating with members of the other sub-net. Higher number of bisection channels means higher possibility of concurrent global communications and lower global saturation threshold.

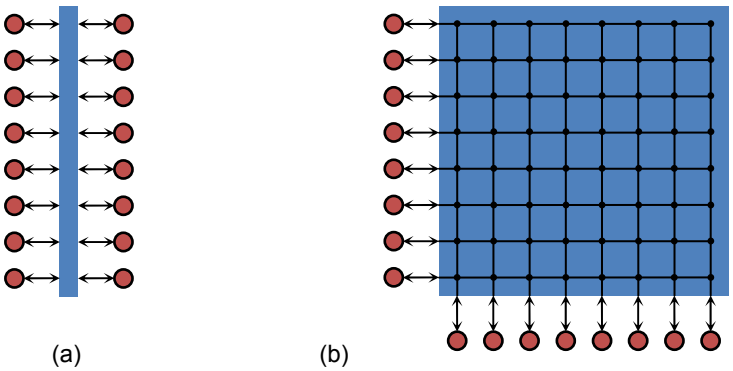


Figure 60. Classical Interconnects of (a) Shared-Bus and (b) Crossbar

The analysis results of some interconnect topologies are presented in Table 9. As shown in this table total number of channels in a shared-bus is 1, introducing a catastrophic feature especially when number of nodes to interconnect is high. On the other hand, although the equality between the number of channels and the number of nodes in a crossbar predicts a better behavior, the switch degree of crossbar is strictly high. The design complexity of a switch with the degree of *N*,

where *N* is number of nodes, reduces the scalability. Refer to Figure 60. Additionally the use of only one switch (one centralized hard macro-core) produces some serious problems, such as incompatibility with GALS paradigm, inflexibility in the design level of core placement, etc. Principally the system designers seek high granularity of system cores.

Table 9. Formal Comparison between Usual On-Chip Interconnect Topologies

	Number of Nodes	Number of Switches	Switch Degree	Network Diameter	Number of Channels	Number of Bisection Channels
Shared-Bus	N	1	N	1	1	1
Crossbar	N	1	N	1	N	N
Binary Tree	$N = 2^n$	N - 1	3	$2 \log_2 N$	4 N - 4	2
Fat-Tree (2-ary)	$N = 2^n$	$\frac{1}{2} N \times \log_2 N$	4	$2 \log_2 N$	$2 N \times \log_2 N$	N
Fat-Tree (4-ary)	$N = 4^n$	$\frac{1}{4} N \times \log_4 N$	8	$2 \log_4 N$	$2 N \times \log_4 N$	N
Butterfly Fat-Tree	$N = 2^n$	$\frac{1}{2} N \times \log_2 N$	2	$\log_2 N$	$N \times \log_2 N$	$\frac{1}{2} N$
Ring	N	N	3	$\frac{1}{2} N$	4 N	4
2D-Ring	$N = n^2$	N	5	$\sqrt{N}$	6 N	$4 \sqrt{N}$
Chordal Ring	N = 2 n	N	4	$\frac{1}{4} N$	5 N	8
Array	N	N	3	N	4 N - 2	2
2D-Array	$N = n^2$	N	5	$2 \sqrt{N}$	$6 N - 4 \sqrt{N}$	$2 \sqrt{N}$

A.3.1.1. Tree-Based Topologies

The binary tree as network topology (Figure 61.a) employs *N-1* switches (routers) with the degree of 3, where *N* is number of nodes. Total number of channels is $4 \times (N-1)$ and the network diameter is $2 \times \text{Log}_2 N$. While all these parameters would show the potential of a satisfactory performance, the limited number of bisection channels (i.e. 2) indicates the limited possibility of simultaneous global communications. Tree-based topologies are often beneficial for exploiting locality of the communications.

The binary fat-tree and butterfly binary fat-tree are two alternative solutions to increase number of bisection channels. As shown in Table 9, the fat-tree topology has twice the number of network channels and bisection channels of a butterfly topology, at the expense of double network diameter and switch degree. The links in a fat-tree are bidirectional, while a butterfly fat-tree uses

unidirectional connections. As can be seen in Figure 62.a, in a butterfly network for each pair of source and destination there is only one route and each route blocks many other routes, but in such a network a simple deterministic routing algorithm could be used. On the other hand, profiting from the diversity of routes in a fat-tree topology (Figure 62.b) that predicts a wonderful performance, demands an adaptive routing algorithm which as will be explained introduces some important issues and increases the complexity of the implementation.

Unlike to the binary tree that may have a regular two-dimensional layout (see Figure 61.b), binary fat-tree (and similarly butterfly binary fat-tree) topologies have more difficulty to regularly lay out on a two-dimensional chip surface and as a consequence wires could not have symmetric lengths and number of long wires vastly grows. Perceptively, a fat-tree could be imagined as a multi-dimensional graph. Figure 62.c shows the layout of a 4-dimensinal binary fat-tree. Laying out a multi-dimensional network on a two-dimensional plane requires multi layers of wires and multi number of vias.

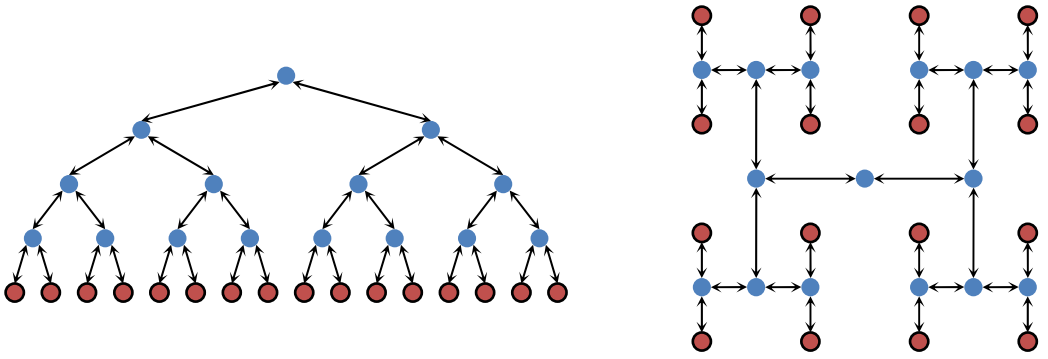


Figure 61. Interconnect topology of (a) binary tree and (b) its layout

The term of 2-ary n-dimensional fat-tree commonly indicates the structure of binary fat-tree topologies. With respect to the terminology of k-ary tree, where k is the number of childes connected to each node, the term of k-ary n-dimensional fat-tree refers to the corresponding topology. SPIN micro-network exploits a 4-ary n-dimensional fat-tree, which is proven in [82] to be the most hardware-efficient supercomputing universal network, compared to any others. As an instance, a 4-ary 3-dimensional fat-tree is depicted in Figure 63. This 3-dimensional graph corresponds to the geometric shape of diametral cube. Refer to Figure 63.b.

Due to the low network diameter with $O(\mathrm{Log}_k N)$, high number of channels with $O(N \times \mathrm{Log}_k N)$, and high number of bisection channels with $O(N)$, the k-ary n-dimensional fat-tree topologies are the best choice from the viewpoint of connectivity and are widely used in parallel computer architectures [80]. However, implementing this kind of highly connected networks on a chip demands a complex wiring, using enormous number of long wires, particularly on various layers. Wiring is a limiting factor and really a major concern in NoC architecture designing.

Moreover, k-ary n-dimensional fat-tree topologies suffer from the fact that number of switches exceeds number of nodes, when number of nodes goes beyond k^k (e.g. 4 in binary fat-trees), as the number of switches is $(N \times \text{Log}_k N)/k$. The growth in k reduces number of switches, but increases the degree of switches and directly affects design and wiring complexity. This event incurs an important network overhead. For the on-chip interconnects the network overhead strictly is more critical than for the off-chip networks, and the design scalability is more essential.

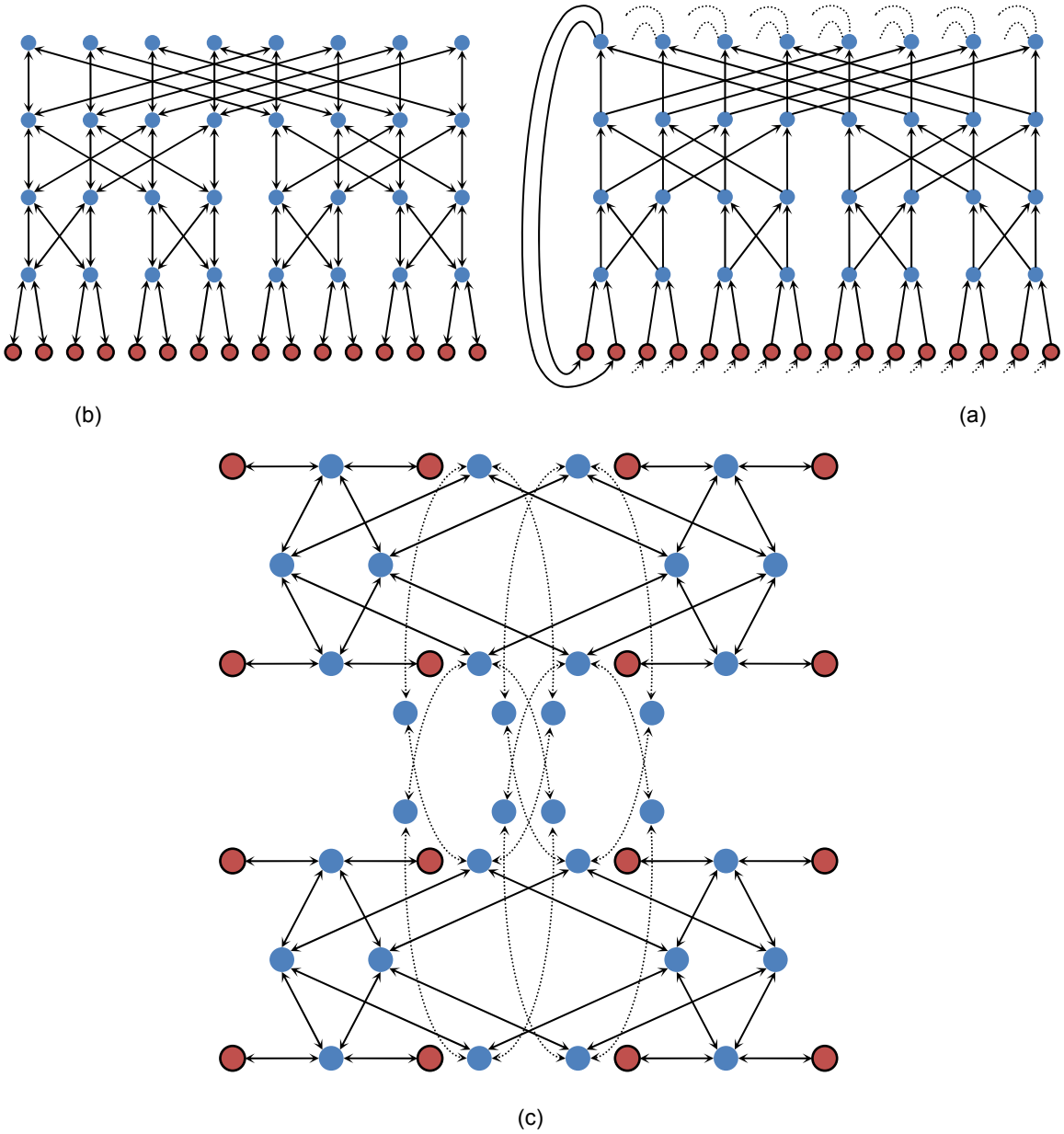


Figure 62. Topologies of (a) butterfly fat-tree and (b) binary fat-tree (2-ary 4-dimensional) with (c) its 2-dimensional layout

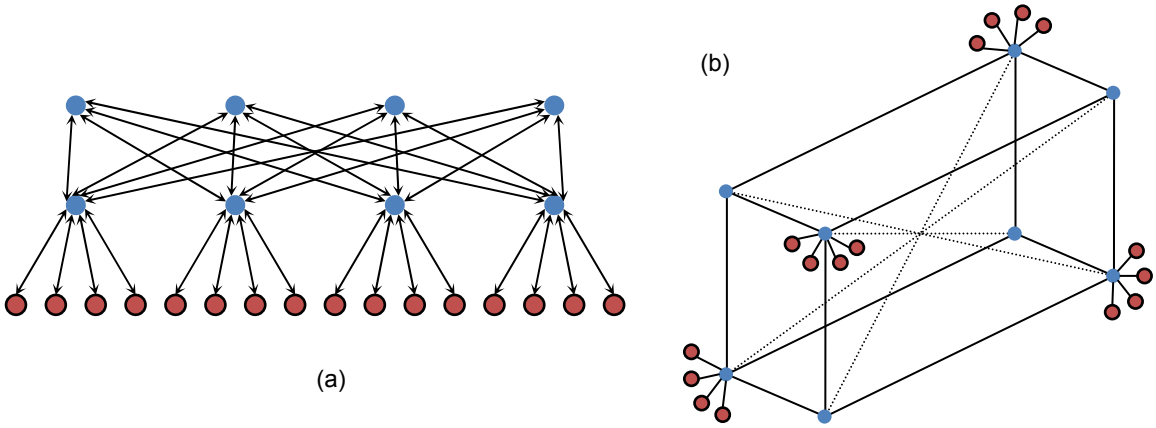


Figure 63. A (a) 4-ary 3-dimensional fat-tree with (b) its 3-dimensional perspective

A.3.1.2. Direct Network Topologies

The use of direct networks helps to simplify the physical implementation. Direct networks are those that have at least one core attached to each switching node of the network, and so switches (routers) may regularly spread between cores, one-to-one. Indirect networks on the other hand have a subset of nodes not connected to any core. All tree-based topologies (including fat-trees) that cores are connected only to the leaf nodes, are indirect networks.

The ring and array are two of the simplest interconnect topologies. As they are direct networks the network overhead increases with $O(N)$ and as demonstrated in Figure 64.a and Figure 64.b these two topologies can easily be laid out on a two-dimensional square plane. But, the low number of bisection channels and high value of network diameter are two limiting factors for the scalability.

Chordal ring is an improved ring topology in which each node of the ring, in addition to two adjacent nodes, is connected to the opposite node by the chord of the ring. See Figure 64.c. the layout of a 16-node chordal ring is displayed in Figure 64.d. The chordal ring topology provides twice shorter routing path, and even for a limited number of cores the shortest compared with any other topologies. For example a chordal ring of 8 nodes guarantees two-hop communication between any pair of nodes, and a ring with 16 nodes four hops.

Anyhow, the chordal ring is also not a scalable topology. Total number of bisection channels is 8 and the network diameter grows with $O(N)$. In order to achieve a scalable network topology the authors of [39], as shown in Figure 64.e, have proposed the hierarchical topology of 8-node chordal rings, called Octagon. This design is based on the use of some special bridge nodes (the red nodes in the figure). Remember that the hierarchy in any kind of bridge-based architectures (such as local busses) profits from the locality of communications. The possibility of concurrent global communications is reduced.

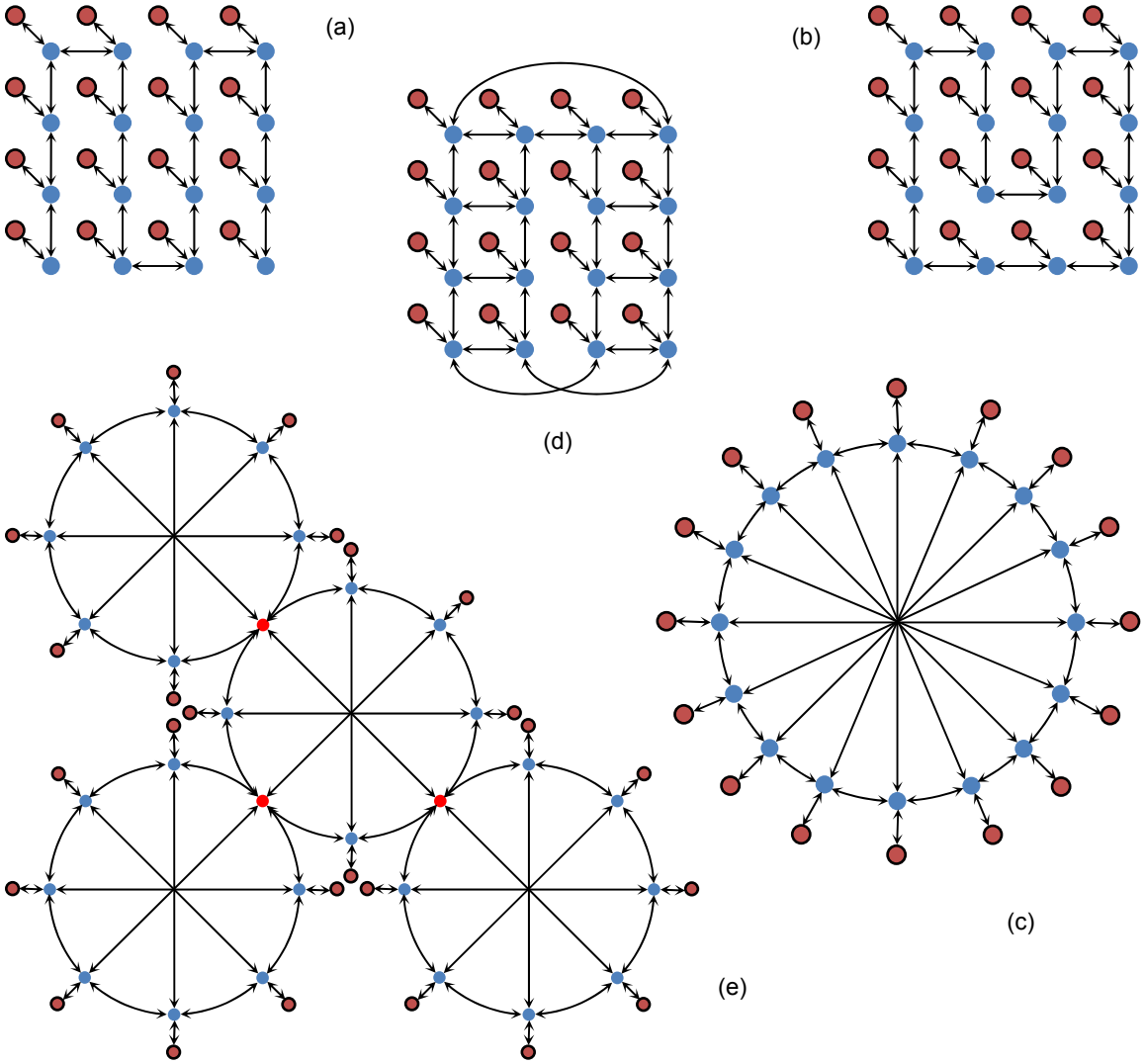


Figure 64. Direct networks of (a) array, (b) ring, and (c) chordal ring with (d) its layout and (e) hierarchical form

A.3.1.2.1. Grid-Based Topologies

To simply distinguish regular topologies which offer acceptable connectivity and utilization of resources in the networks, first in [83], the term of k-ary n-cube was described to provide a convenient notion of multi-dimensional grid-based topologies, where k is the number of nodes in each dimension and n is the number of dimensions. Seeing the fact that because of the two-dimensional square structure, a k-ary 2-cube (a two-dimensional grid-based topology) is straightforward to lay out on a chip, degrading the wiring complexity, two-dimensional ring

(torus) and array (mesh) topologies seem more fulfilling for on-chip interconnecting. Additionally routing in this type of networks is easy and results in potentially small routers.

In these two grid-type networks the length of wires is symmetrically limited to the distance between each two adjacent cores, except for those links of a torus that connect the first and last nodes of each row or column. Refer to Figure 65. To moderate wire lengths a torus could be folded. An example of a 6×6 folded torus network is given in Figure 65.c. Averagely, the length of wires in a mesh is shorter than in a torus or than in a folded torus. On the other hand a torus topology has twice the bisection channels of a mesh network, and half the network diameter, at the expense of a double demand of wiring.

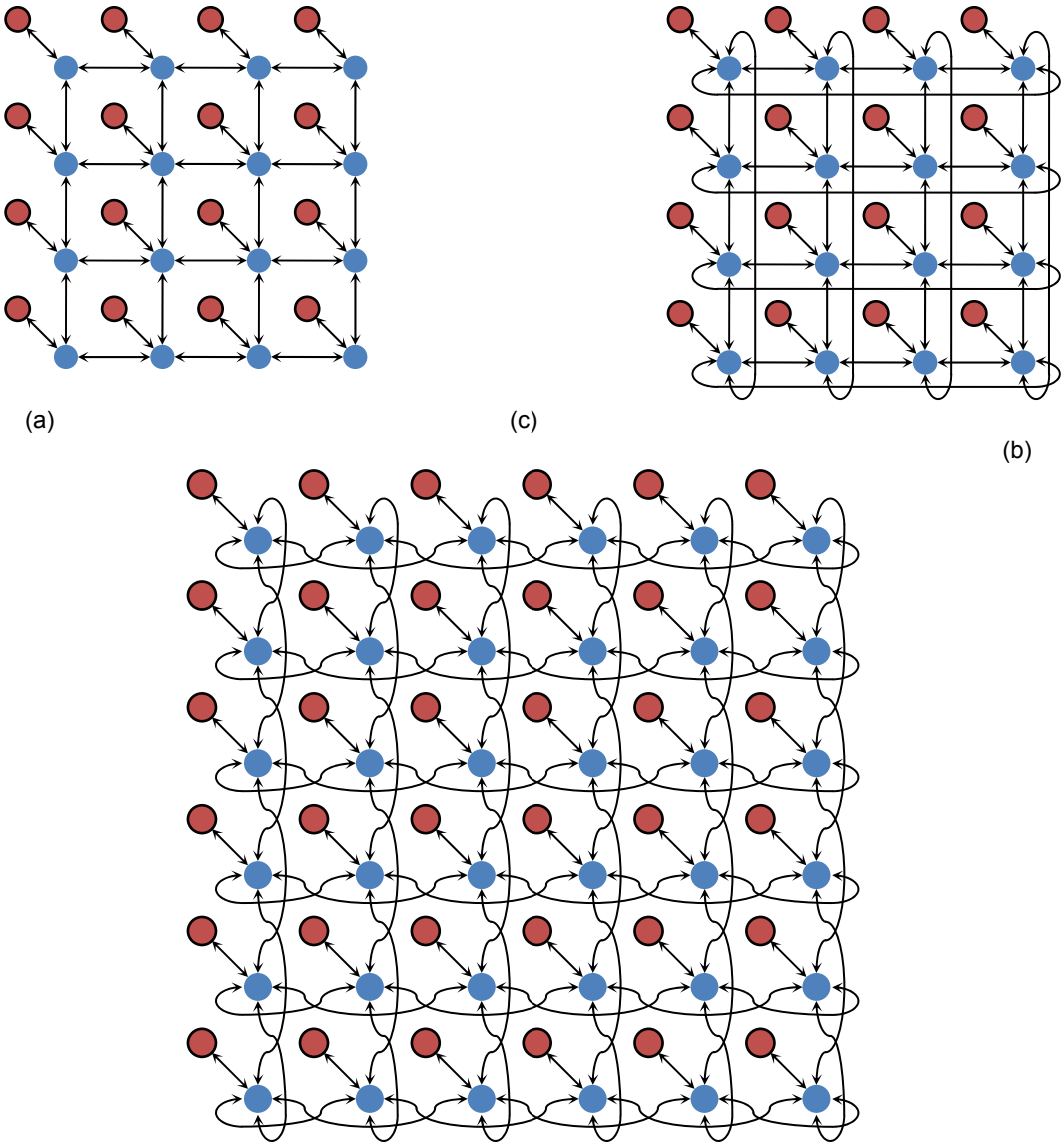


Figure 65. Regular topologies of (a) mesh and (b) torus with (c) a folded-torus example

The choice between torus and mesh as topology of a NoC is an arguable case. Both networks are fully scalable, but while torus provides a better performance, regularity and better use of links and eventually lower network overhead advocate a preference for mesh. For example, as a result of longer wires , in [36] a folded torus is rejected in favor of a mesh with the argument that it has longer delays between routing nodes, and in [34] based on the claim that it has more wire transmission power dissipation. The authors of [44] have also argued that a mesh is a more economic approach as the degree of switches on the borders could be reduced to 4, and on the corners to 3. According to the current and future needs and respecting the new deep submicron technology trends, it seems that to choose a topology for a general-purpose Network-on-Chip, one has to opt for a more regular form with shorter wires that shares the links more efficiently, that is a mesh.

To complete the talk, it is useful to notice that due to probable heterogeneity and irregularity of SoCs using modules with varying size and communication requirements, the best choice of an application-specific topology often is an irregular form of topologies, derived by mixing different forms, in a hierarchical, hybrid or asymmetric fashion. Irregular topologies scale non-linearly with regards to area and power. These are usually based on the concept of localizing communications and exploiting the most efficient solutions for small local-private networks.

A.3.2. Routing Algorithm

Routing is the process of selecting paths in the network between a source and a destination. The regular topologies of general-purpose Networks-on-Chip allow an algorithmic routing, as opposed to static, statistic, or stochastic routings which need often to use a Look-Up Table (LUT). The routing algorithm restricts the set of possible path to a small set of legal paths. Minimal algorithms determine the shortest paths between each two nodes of the network. Depending on the topology, there are many different routing algorithms providing different guarantees and offering different performance trade-offs, such as predictability versus average performance, switch design complexity and speed versus channel utilization, robustness versus aggressiveness [74].

Algorithmic routing can either be determined at the source or with a distributed manner by routers along the path. In source routing the entire route of a packet is decided by the source node which has to be aware of the network's topology. It encapsulates the exact node-to-node itinerary of a packet in the header. As the packet traverses the network this information is used by each node on the path to navigate the packet towards the destination. Seeing that this method results in potentially smaller routers, source routing represents a cheap solution for NoCs. However, there is the problem of the routing information overhead. For a network with a diameter of k, a packet would require k routing information summarized in the header of packet. As the network grows the header overhead may become too high, that is the loose of scalability.

In contrast, the header of a packet in distributed routing has to include only the destination coordinates. Each node examines the destination address and decides along which channel to

forward the packet. Distributed routing can be either deterministic or adaptive. In deterministic routing the path is determined just with regard to the source and destination positions, and often there is only one path between each pair of nodes. An adaptive routing on the other hand tries to adapt the route according to the current traffic conditions and eventual contentions.

While deterministic routings are the best choice for uniform or regular traffic patterns, the adaptive approaches are preferable in presence of irregular traffic or in networks with unreliable nodes and links. As a consequence of involving dynamic arbitration mechanisms an adaptive routing can route packets around blocked nodes and channels. This ability is essential for fault tolerance and brings about a significant improvement of the network performance, but at the expense of a more complex design that possibly works slower. Moreover, since packets may arrive to the destination in different ways and with different latencies an adaptive routing could not guarantee the order of packets. To achieve in-order-delivery property which is an essential for some applications, a network at the network interfaces requires implementing either a hardware reordering module or an end-to-end flow control mechanism. These requirements increase design complexity, and likely decrease communication latency.

A.3.2.1. Routing Blockage

When we talk about network performance in terms of latency and throughput, without being expressed in words we assume that all messages finally arrive to destinations, even with large delays. Nevertheless in a badly-designed network, it is possible that some packets never reach the destinations. Concerning this problem, routing algorithms must avoid two major issues: live-lock and dead-lock, as described as follows.

- **Live-Lock:** As a real-world example, live-lock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they always both move the same way at the same time [84]. Likewise, in network architecture level, live-lock explains the situation that even after passing through an infinite number of nodes the packet will not arrive to the destination. A live-lock free routing algorithm has to guarantee forward progress of each packet, where after each hop the packet in one step closer to its destination. Live-lock is a less common situation, but it may be expected in networks where back-stepping is allowed, e.g. during contention detecting and resolving by non-minimal adaptive algorithms. Figure 66.a shows an example of live-lock in a mesh topology. The red dashed-line presents the desired path of red packet from node 5 to node 10 and the blue dashed-line display the path required by the blue packet to go from node 1 to node 9. Imagine the situation that two packets at the same time arrive to node 7 and want to pass through node 9. In order to remove the contention, the red packet is led to take another path, in which it traverses nodes 3, 2, 6, and again enters node 7.

Similarly, the blue packet comes back to node 7 by passing through the nodes of 8, 4, and 3. As the tragedy could be repeated constantly, two packets never reach their destinations.

- **Dead-Lock:** As a real-world example, dead-lock occurs when two people meet in a narrow corridor, and each waits for the other to move aside first, but they end up waiting without making any progress because they make a paradox. Similarly, in network architecture level, dead-lock defines the situation when permanently two or more packets are each waiting for another to release a shared channel in a circular dependency. An example of routing-dependent dead-lock in mesh topologies is presented in Figure 66.b. The red packet, which desires to go from node 7 to node 2 (red dashed-line), is blocked at node 9 (red solid-line) because the path is already occupied by the green packet that requires arriving to node 3, traversing node 9 (see green lines). The green packet, itself, is blocked at node 5 and waits for the release of the channel between node 5 and 4, allocated to the blue packet. The later cannot make any forward progress due to channel occupation by the brown packet that needs to reach node 10 via node 8. Since the path between node 8 and 9 previously is associated to the red packet, the brown packet also cannot move forward. Unluckily, cyclic dependencies on shared channels here testify the existence of a paradox and cause dead-lock.

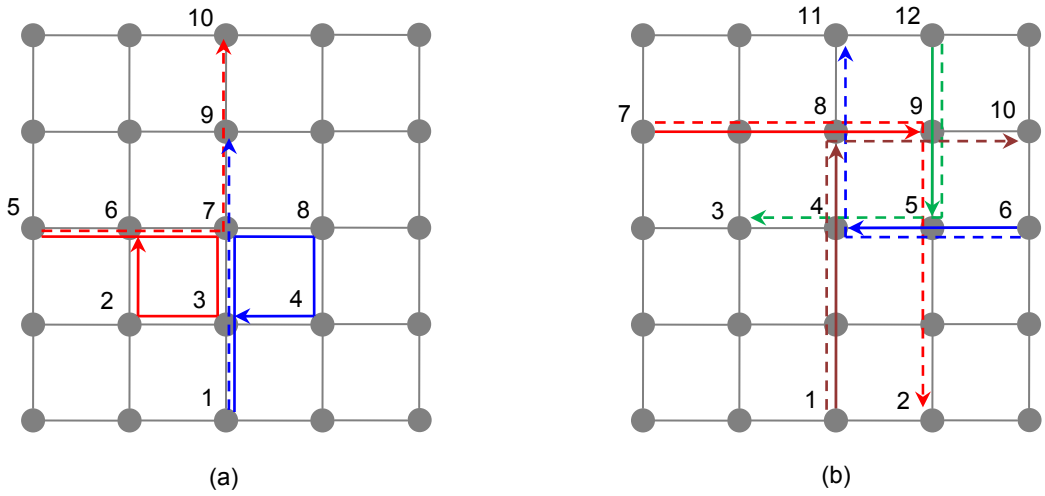


Figure 66. Examples for two routing blockages of (a) Live-Lock and (b) Dead-Lock

Dead-lock can occur in a variety of situations in different layers of a network. For example at the network interfaces a head-on dead-lock may happen, if the system is based on bidirectional communications, constituted by a pair of request and response messages. Each request is correlated to a replied response. Imagine the situation that two nodes try to send a request to each other and each begins to send before either receives. Obviously if they both attempt to complete their own requests (i.e. receiving the related response) before receiving any other request, neither

will make forward progress. In fact as no request will be received, no response will be produced. See Figure 67.a.

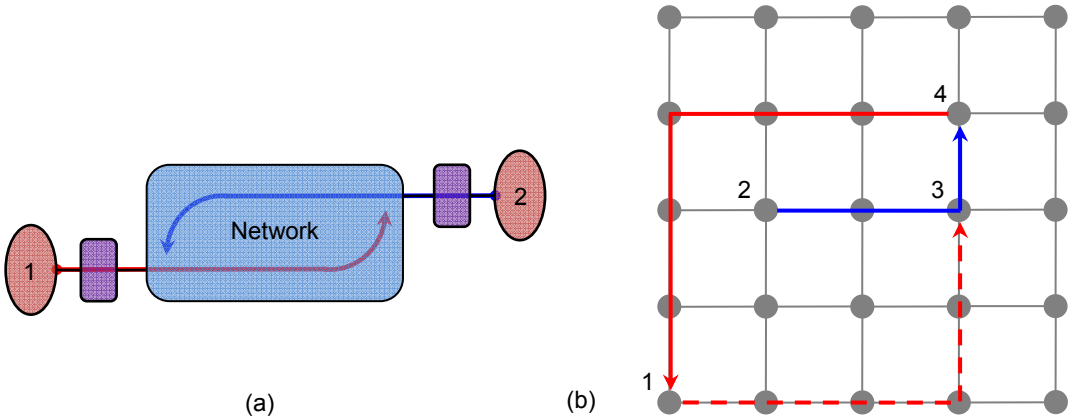


Figure 67. Example of message-dependent dead-locks (a) at network interfaces and (b) within the network

A network may also be the place of a similar message-dependent dead-lock. Supposing there is a limited number of paths between each pair of nodes, to reach an individual destination all kinds of packets (e.g. requests and responses) share same channels and can be blocked one behind the other. To better understand, Figure 67.b shows an example in which there is only one route between each two nodes. Let's assume that node 4 has sent a request to node 1. At the same time node 2 sends another request to node 4. This request could not be received unless node 4 has previously received the response of its request. But the response (red dashed-line) is blocked at node 3, behind the request of node 2 (blue solid-line). It is clear that here there is a dead-lock.

Considering the fact that dead-lock occurs when some processes share a specific type of mutually exclusive resource, a classical method of dead-lock prevention proposes to provide several resources to be associated to each process, if possible. Correspondingly, if request and response packets use two independent groups of channels, the absence of the occurrence of any dead-lock based on request-response dependencies, inside the network will be guaranteed. A simple (but expensive) solution is to double the network, one for requests and other for responses. That is to say using two completely separated networks and allocating one type of packets to each of them, break cyclic message dependencies inside the network. However, from the viewpoint of cost the use of Virtual Channels (VCs) may be a more reasonable approach. In a network with virtual channels packets are assigned to different virtual channels, but with separate physical buffer queues. Therefore, even if a packet is stalled, another is enabled to pass, using its own buffer space.

Seeing that in this kind of message-dependent dead-locks there is no routing involved, they might be considered out of the responsibility of the network architecture and could be concerned

at upper layers, for example, by ensuring that nodes can continue to receive requests even while they have not received the responses yet. Networks typically involve problems imposed by their architecture design, a benefit offered by layered design methodology.

A.3.2.2. Dead-Lock Free Routing Algorithms

Turn-Model routing, presented in [85], is an interesting approach to provide dead-lock freedom in packet routing for a two-dimensional mesh topology. This method claims that a dead-lock free routing algorithm can be constructed by restricting some of eight possible route turns, which form two simple cycles, as shown in Figure 68.a.

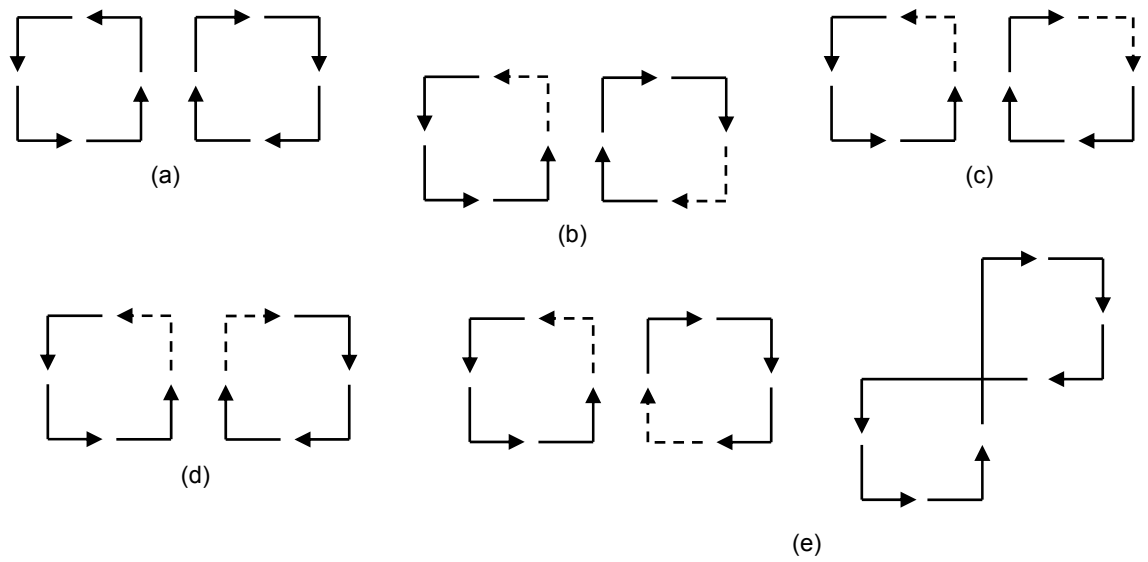


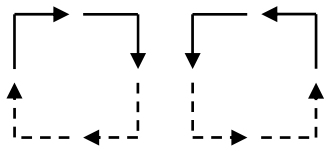
Figure 68. Adaptive Turn-Model Routings: (a) all eight possible route turns, (b) West-First, (c) Negative-First, (d) North-Last, and (e) dead-lock allowable model

A minimal set of restriction gives a maximal adaptiveness to the algorithm. 3 of the 4 different ways to eliminate one turn in each cycle (12 of the 16 if rotations are taken into account) prevent cyclic dependencies. These three dead-lock free adaptive routing algorithms are displayed in Figure 68.b, c, and d. All 12 possible solutions consist of the rotations of these three unique models. Figure 68.e shows the fourth turn-model that a cyclic dependency is allowed and so it is not a dead-lock free solution.

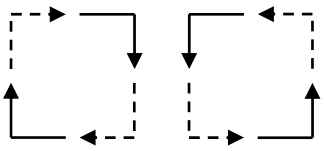
The first dead-lock free algorithm is named West-First because there is no turn allowed into west direction and therefore if a packet needs to go west it must do before making any other turn. Similarly, in Negative-First there is no way to turn from a positive direction to a negative direction, thus a packet must go as negative as its need before heading in either positive direction. Finally, North-Last prohibits turning out of north direction, so the packet must make all its other

adjustment before heading in this direction. Each of these three algorithms allows very complex, even non-minimal routes. But as explained before, an adaptive routing algorithm usually is not a recommended solution for cost-constrained NoC architectures.

Intuitively, constructing other types of routing algorithms should be possible by eliminating more than two turns. 2 of the 12 (6 of the 36 with regard to rotations) different ways of eliminating two turns in each cycle give applicable routing algorithms, which are demonstrated in Figure 69.a and b.



(a)



(b)

```
    if ΔY > 0 then Up
else if ΔX > 0 then Right
else if ΔX < 0 then Left
else if ΔY < 0 then Down
```

(c)

```
    if ΔX > 0 then Right
else if ΔX < 0 then Left
else if ΔY > 0 then Up
else if ΔY < 0 then Down
```

(d)

Figure 69. Turn-Models of (a) Up-First-X-Next and (b) X-First with (c) deterministic algorithm for Up-First-X-Next and (d) deterministic algorithm for X-First

In the first approach, Up-First-X-Next (Figure 69.a), two types of turns are prohibited: turning to up direction and turning out of down direction. Therefore, if a packet needs to go up, it must do before taking any other direction. On the other side, if the packet requires going down, it has to make all its left or right (X direction) adjustments, before heading to this direction. This method is enabled to allow non-minimal partially adaptive routes. Nevertheless, a minimal Up-First-X-Next is a fully deterministic routing algorithm. Refer to Figure 69.c.

X-First routing algorithm routes packets first along X direction and then along Y direction as there is no turn from Y direction to X direction. Potentially, X-First is a minimal routing and can be determined by a simple and deterministic algorithm. See Figure 69.d. This dimension-ordered routing algorithm is sufficient simple to be implemented in low price hardware and its optimal use of channels has led to be the most popular routing mechanism in general-purpose NoC architectures. A general concept of dead-lock free minimal deterministic dimension-ordered routing algorithms is proposed in [83] to be applied in any k-ary n-cube network topology, by travelling the correct distance in the highest order dimension, then the next dimension and so on.

A.3.3. Switching Strategy

While routing algorithm defines a route between source and destination nodes, switching strategy determines how the data of a message traverses this route. Switching has been defined as just the transport of data, when routing is the intelligence behind [86]. Basically two switching strategies exist: circuit switching and packet switching.

In circuit switching a route from a source to a destination is established prior to data transport and exclusively reserved until the message is completely transferred. As at the phase of path establishment two communicating nodes are being connected point-to-point, circuit switching is always a connection-oriented mechanism requiring a logical or physical connection before data transmission. As an example we can indicate the phone systems that set up a circuit through possibly many routers for each call.

The reduced need for buffers at the intermediate nodes could be an advantage of circuit switching. This policy is beneficial for moving a lot of data through and in particular when communication latency has to be guaranteed. But an advanced operation is required which increases the initial latency. In addition it tends to be unpleasant for any traffic that might cross or share a portion of the reserved route, even when data transferring is not in sight.

As an alternative strategy, in packet switching data is not transmitted on a predefined circuit. The message is broken to a sequence of packets. A packet logically consists of a header (which includes routing and perhaps sequencing information), data payload, and possibly a trailer. The header reserves the routing channel of each router, the data payload will then follow the reserved channel and the trailer will later release the channel reservation. Packets are individually and independently routed through the network, and at the destination the packets are assembled into the original message. In order to ease packet assembling if a message is divided into several packets, the order of packets at arrival must be same as departure. The in-order-delivery property is an essential opportunity that should be provided by any NoC using packet switching.

Packet switching may potentially be the cause of a bother shown in Figure 70. Suppose node 1 is transferring a long message to node 3. The message is divided into 6 different packets (red squares) which are individually sent through the network in a sequence, passing node 4. At the same time node 2 intends to send another message to node 3 too. The series of separate packets of this message (blue squares) are also travelling the network via node 4. Seeing that the path from node 4 to node 3 is shared by two messages, the packets of message blue may arrive at node 3 in between packets of message red. Observe that in packet switching strategy there is no any dedicated path.

The out-of-series arrival of packets can induce a serious problem for the upper layer. Clearly if node 3 accepts only packets of the message being received, i.e. red packets, and does not receive any other packet, the first blue packet (blue square 1) will be stalled at node 3 and will block the route, resulting in a dead-lock. Remember the case of request-response (message-dependent)

dead-locks explained previously. Although the use of Virtual Channels (VCs) could partially reduce the occurrence probability of such dead-locks, it is not a concrete solution because due to a set of limiting factors there can be a limited number of virtual channels, while number of packets which arrives possibly out-of-series is not limited.

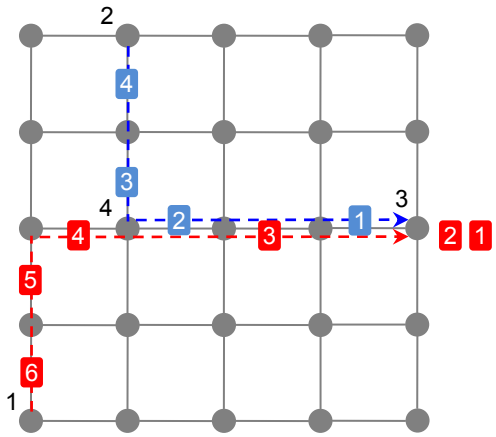


Figure 70. Example of Out-of-Series Arrival of Packets

A way that potentially resolves this problem is that the source sends whole message as a single packet. It is a solution but may have side effects on the network performance. If a packet is too large it may block several channels and prohibit other packets to traverse. Message granularity is crucial to decide. Determining the right packet size to make optimum use of the network resources is an important responsibility of the network interface layer. In fact all packetization related decisions concern this level of design and often are application specific.

Anyway, the wide majority of NoC architectures are based on packet switching, due to the fact that it is fundamentally a cost-effective solution and often allows better utilization of network resource, because buffers and channels are only occupied while a packet is traversing them. Usually packet switching is used to provide best effort service, as opposed to guaranteed service. Best effort service does not give any performance guaranties in terms of latency and effective throughput, but instead it is averagely better. Packet switching is originally a connection-less communication mechanism and there is no any advanced planning and operation.

Furthermore, packetization may offer an effective way to deal with errors in communications. If data is sent on an unreliable channel in packets, error containment and recovery are easier, because the effect of errors is contained by packet boundaries, and error recovery can be carried out on a packet-by-packet basis [77]. Error correction can be achieved by using standard Error Correcting Codes (ECCs) that add redundancy to the transferred information. Typically ECC is included in the trailer of packet. Packet size and number of outstanding packets are some examples of several parameters that can be adjusted depending on the goal of achieving maximum performance at a specified residual error probability.

A.3.4. Flow control

Flow control strategies can be used in different layers of a network-based system. In network architecture level a flow control has been defined as the mechanism that determines how and when each portion of packets, moves from a hop to another along the path. The minimum unit of information that can be transferred across a hop at a time is called flow control unit or, in abbreviation, flit. A flit may be as small as physical unit (phit) or as large as whole packet.

The mechanism of flow control, in particular, is necessary when two or more packets attempt to use the same channel, at the same time. One could be stalled in place, diverted into buffers, or led to an alternate route. Flow control needs specific requirements on the design of routers. It also mainly influences the communication performance. Commonly there are three different flow control strategies, which are:

- **Store-and-Forward:** in store-and-forward policy a flit is a complete packet. So an incoming packet is accepted only when there is enough space available to store it entirely in buffers and furthermore the packet is not forwarded to the next hop until the whole contents have been received. This strategy is an elegant way to achieve optimal utilization of channels resulting in a high saturation threshold, because only one channel is occupied when a packet is stalled. The expense is the increase of network overhead as each router has to be able to store entire packet in physical buffers. Additionally, this mechanism incurs a delay per hop proportional to the packet size, or precisely equal to the time required to receive the packet completely. Whereas communication latency increasingly will be implicated by number of hops on the path, the store-and-forward approach is impractical in large-scale Networks-on-Chip.
- **Virtual Cut-Through:** similar to the store-and-forward strategy, in virtual cut-through policy a flit is also the entire packet. A router cannot forward a packet unless the next node has enough space available to store the packet completely. But to forward the packet, unlike store-and-forward, it is not required the whole packet to be received by the router. As packets may be forwarded as soon as the header of packet is received, the communication latencies decrease. However the virtual cut-through approach is to store the stalled incoming packet into a buffer, so the behavior under contention degrades to that of store-and-forward flow controlling.
- **Wormhole:** in wormhole flow control method a packet itself is split into several flits. A flit is forwarded to the next hop when there is enough space available at the receiving point to accept. The need of buffering is reduced to the least ability of storing a single flit. The wormhole and virtual cut-through strategies benefit from the same policy in which it is not required to complete receiving before sending. However, depending on the flit size, the latency within the routers under contention in the wormhole mechanism may not be that of the virtual cut-through, because forwarding progresses as soon as the next hop has

enough space available to accept a flit and not the whole packet. In wormhole flow control strategy, one packet may occupy several intermediate channels at the same time, because the packet is spanned along a long path. Increasing the storage capacity of the routers allows gathering the packet flits and thereby a lower number of channels will be occupied by a packet. But on the other hand when routers are capable to store several flits, the header of packet that contains packet information may be stalled inside a buffer queue. This probable event disables content-aware packet routing algorithms, such as some adaptive routings which route depending on the type and contents of packets. Nevertheless, thanks to the low communication latency and small required buffer queue, a large number of NoC architecture implementations are based on wormhole packet switching in which a flit consists of a phit.

The implementation of flow control differs depending on the type of the flow control strategy and the design of the router, but the main idea is the same: data is to be transferred from one storage stage to another. The key point is that storage at the destination stage may not be available to accept the transfer, so data must be retained at the source stage until the destination is ready. The organization of data transmission within the router has a significant impact on the router performance. Traditionally this organization is considered as two separated concerns: storage strategy that determines where data is to be retained and output scheduling that defines when retained data can be out.

A.3.4.1. Storage Strategy

In most of NoC architectures, the silicon area of buffers is the main part of the router area. It is a major concern to minimize the need of buffering, under given performance requirements. The issue is not just how effectively the buffer resources are utilized, but how the buffering effects the utilization of other components [80]. For example, sharing a centralized buffer queue allow better utilization of these resources than partitioning the storage among router ports, but if some ports need to access the queue simultaneously, sharing the buffer on demand can hurt channel utilization, as buffer queue can only serve one request at a time. Moreover a single congested output port can block the queue and thereby prevent other traffic from moving through the router.

A.3.4.1.1. Input Buffering

An approach of non-centralized buffering is to provide independent FIFO buffers with each input port. See Figure 71. One problem with the simple input buffered approach is the occurrence of Head-Of-Line (HOL) blocking. Suppose that two ports have packets destined for the same output port. One of them will be scheduled onto the output and the other will be stalled. The packet just behind the blocked packet may be destined for one of the unused outputs, but it will not be able to move forward. Queuing theory analysis shows that the expected utilization in steady state with input buffering is 60% [87].

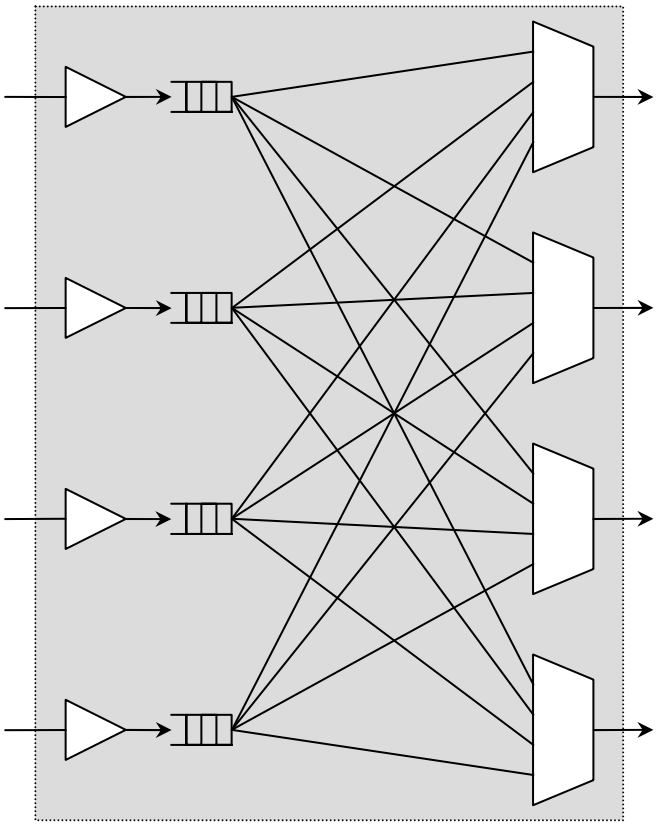


Figure 71. Internal data path of a router using Input Buffering

A.3.4.1.2. Output Buffering

A natural enhancement is to expand the input FIFOs to provide an independent buffer for each output port. See Figure 72. Packets are arranged according to destined output port on arrival. It is a question of perspective whether FIFOs are associated with the input or the output ports. If viewed as output buffering, the key property is that the router has enough internal bandwidth that simultaneously each output port can receive a packet from every input port. As a consequence the router does not introduce additional contention effects internally. With a regular traffic on the inputs, the outputs can be driven at essentially 100%.

However profiting from the advantages of such a design requires a large amount of buffer storage. For a router with degree of N, number of required FIFOs is N^2, while input buffering needs only N buffer queues. Additionally, different from store-and-forward and virtual cut-through, wormhole flow control policy does not benefit sufficiently from output buffering, because this strategy could not be enough beneficial if the depth of each FIFO is not enough to store the whole packet. Remember that wormhole routing tends to be used as a low cost flow control strategy. Indeed the choice between input buffering and output buffering is once again the choice between cost and performance.

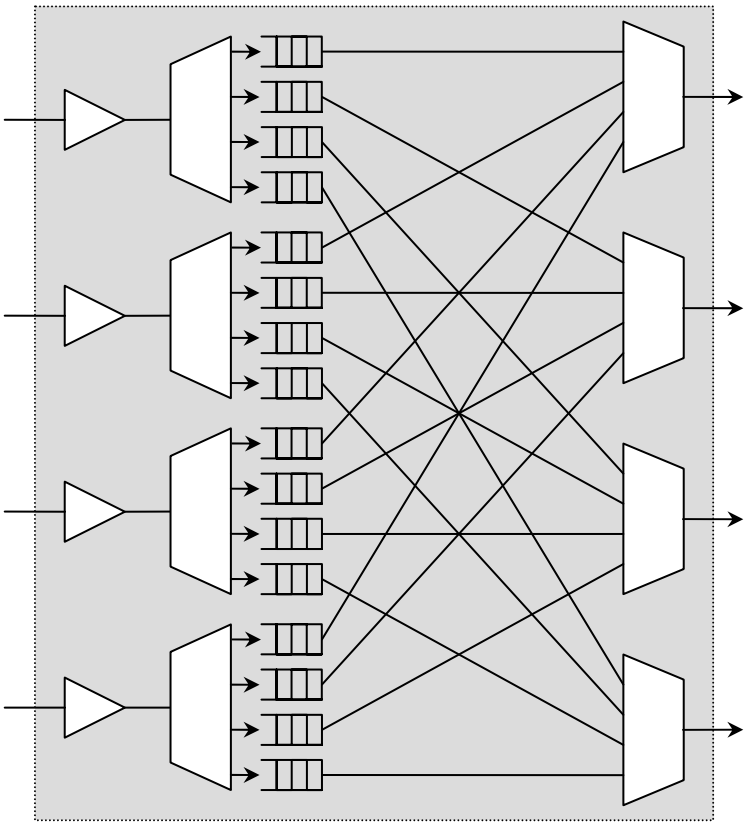


Figure 72. Internal data path of a router using Output Buffering

A.3.4.1.3. Virtual Channels

The use of Virtual Channels (VCs) provides an alternative way of organizing the internal buffers of the router. In virtual channel buffering instead of assigning a buffer queue to each pair of ports, a buffer will be assigned to the incoming packet, when the header flit arrives, and will be reserved until the trailer flit has been transmitted. As displayed on Figure 73, the flow across the router is split on arrival (at an input port) into distinct channel buffers. These are multiplexed together again onto the output ports. If one of these channels is blocked, the others can advance toward the outputs. In fact such a multiple buffer queue virtually provides multiple separate channels. Through virtual channels a network can be divided into multiple disjoint logical networks.

There are a number of advantageous uses of virtual channels. Among them are: avoiding a various kind of deadlocks as virtual channels are not mutually dependent on each other, improving network performance as virtual channels minimize the possibility of stalls, and providing differentiated services as virtual channels can be used to implement such services (e.g. Quality-of-Service) by allowing high priority data streams to overtake those of lower priority or by providing guaranteed latencies on dedicated connections [86].

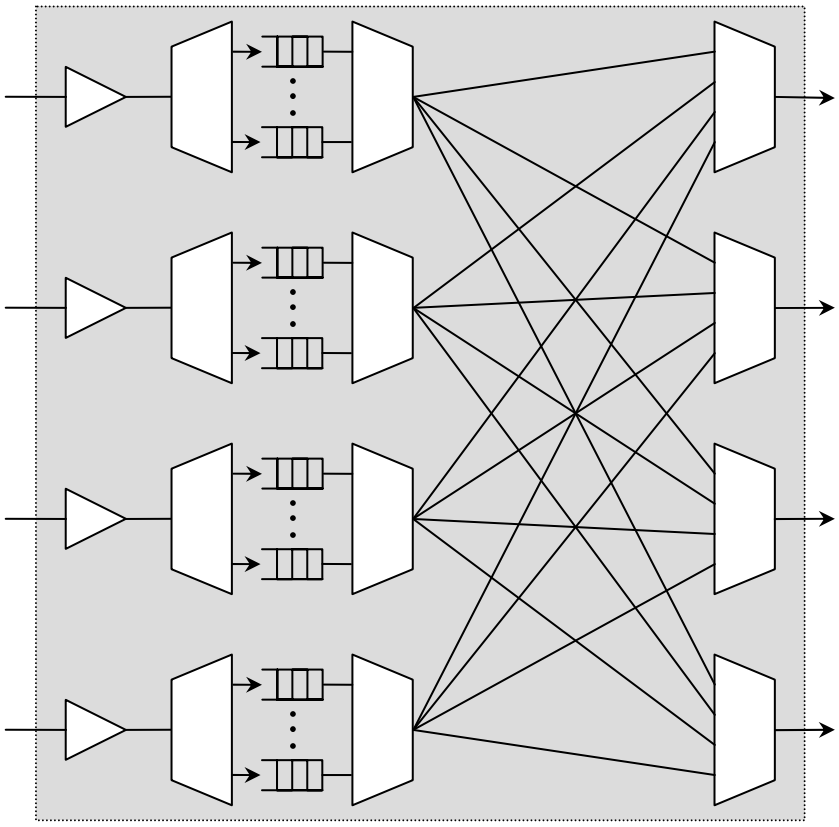


Figure 73. Internal data path of a router using Virtual Channel Buffering

However the implementation of virtual channels results in an area, power, and possibly latency overhead. As number of virtual channels increase, the multiplexing becomes more complicated, requiring additional hardware complexity and potentially increasing latency. Furthermore the sharing of router bandwidth may also increase latency and decrease effective throughput. If one of the physical channels along the path is shared by many packets (virtual channels), that channel becomes a bottleneck and the effective throughput of the entire path is divided by the number of packets traversing the shared channel. The trade-off between increased network aggregate throughput and longer communication latencies should be considered when deciding whether to use virtual channels [88].

A.3.4.2. Output Scheduling

Whereas buffering strategies allow multiple packets per output port to be considered as candidates of forwarding, the need of an output scheduling algorithm that determines the priority order of candidate packets to advance emerges. In fact the scheduling algorithm gives a priority order to each packet, and then the output ports select the highest-ordered packets to be forwarded. Regarding the other aspects of the router design, there is a variety of solutions with different implementation complexity and different performance characteristics.

The scheduling algorithm increases the router latency and thus it should do the job as fast as possible. Clearly static priority (using a simple priority encoder) and random priority are the simplest with excellent average performance results. But they can cause indefinite delays. Basically output scheduling is to avoid defect of blocking a packet so long. The starvation prevention is the main concern that must be considered in a scheduling algorithm. Starvation is produced when a packet can never leave a node because the required output port is always occupied by the other packets with higher proprieties.

Deadline scheduling (e.g. oldest-first) is an efficient starvation free solution that prioritizes the packets according to how long they have been waiting. Oldest-first (first-come first-served) tends to have the same average latency as random assignment, but significantly reduces the variance in latencies [89].

Another simple effective starvation free algorithm is round-robin scheduling, in which packets take the highest priority circularly, one after the other. The round-robin policy is an attempt to treat all packets equally and provide each of them an equal share of the channel capacity. As round-robin scheduling is easy to implement, it can be used as a dominant alternative in NoCs using best-effort traffic. For best-effort traffic the main concern is to provide the property of fairness that has to guarantee if a request is issued it will be granted after that a finite number of other requests have been granted.

A.4. Trade-Off between Cost and Performance

By now, a lot of people have proposed Networks-on-Chip as a solution for large-scale on-chip interconnection. Today, NoCs are one of the most popular academic research fields. As a testimonial, at the 10th international conference on Design, Automation, and Test in Europe (DATE'07) the area of Networks-on-Chip was a particular hot topic with the highest number (56) of received papers. It's a nice step in the right direction, because there are still a lot of problems to be solved. Unfortunately the overhead of proposed NoC-based approaches, generally, has not allowed industry to turn toward NoC technologies, even though it has already been proven that NoCs significantly improve the overall system performance.

Cost-Performance trade-off is a major issue in NoC-based architecture design [90] and determines whether NoCs are blessing or nightmare [91]. While communication throughput vastly improves, NoCs incur increasing communication latency. In a Network-on-Chip, due to the network topology and its architecture, there are a large number of simultaneous communication media and the aggregate network throughput is truly high. But, compared for example with shared busses, it needs more time to route data through the network routers and interfaces. The communication latency is critical for end-to-end system performance. Recall that the main motivation for using Networks-on-Chip is to achieve performance. As said Alain Greiner, we

would not pay for a Ferrari when in a heavy traffic, with a bicycle we arrive to the university, faster and evidently cheaper.

Additional silicon area and power consumption, the result of replacing simple classical communication architectures like shared-busses by Networks-on-Chip which have a more complex circuit, increasingly impact the design cost. Chip design is typically characterized by tight cost constraints, preferentially before high performance demands. As a consequence, routers and network interfaces of a NoC are more resource constrained than those in the computer networks (off-chip). For example, in spite of the fact that Quality-of-Service seems an essential feature for real-time applications, one could fundamentally question the need for hardware implementation of QoS in NoCs [54]. The latency and throughput guaranteed by QoS are the offered opportunities, but they come at a price in terms of area and power overhead and reduced average performance.

A.5. Future Challenges

Although some optimization efforts are still needed, seemingly in the future the research will not anymore focus on NoC architecture design and implementation. For instance two-dimensional mesh topology is being accepted as a general platform, while application-specific architectures will use an irregular topology. Or, deterministic routing algorithms accompanied with a wormhole packet-switching mechanism are being used by all general-purpose high-performance networks.

Apparently the future investigation will be about high-level abstract modeling, application mapping, programming models, reconfiguration algorithms, as well as formally verification methods. As future Systems-on-Chip will consist of hundreds and even thousands of computational subsystems, the simulation-based solutions could not be utilized anymore as functional verification methods.

Time-to-Market is being very tight and as a result, the future SoCs should be highly programmable. A general NoC-based MP-SoC design could be used as hardware framework for various category of multi-core systems. The current programming codes which are based on the sequential processors have to be adapted to the massively parallel processing.

Moreover, FPGAs which represent another embodiment of Networks-on-Chip, are being large and ever increasing sector of the semiconductor market. Within advanced FPGA architectures, both hardware computing elements and interconnects are programmable.

In addition to all these programmable applications, it must be taken into account the exigency of some other kind of systems which need to be managed by runtime techniques such as Dynamic Power Management (DPM) and hard-error detection. As a consequence, the use of different types of reconfiguration algorithms in such architectures appears to be unavoidable. Among them are reconfigurable routing algorithms, reconfigurable threading models, etc.

A.6. Industrial Perspective

Until now, a general use of NoC-based Multi-Processor Systems-on-Chip (MP-SoC) has not been practically feasible. The overhead of using such systems and their inadequate performances has exceeded the advantages. Maybe Cell Processor [92] developed by Sony, Toshiba, and IBM is one of the first multi-processor designed around a NoC that commercially has been produced. It is built as a general-purpose processor for a computer, even though it is primarily targeted for Sony's Playstation 3. Accompanied by memories and I/O controllers, Cell Processor consists of nine processors connected together by a high-throughput, segmented micro-network in a dual ring topology.

As a testimonial for this claim that commercial use of NoC-based solutions will appear in the near future, we can consider the research investment of the most important semiconductor industries on the NoC technology. For instance, Philips research laboratory (NXP) has invested a great deal of resources into their AEthereal NoC architecture [93]. A Philips principle research scientist, in [94] has concluded that "Networks-on-chip have great promise to lift existing platform-based design to communication-centric design. NoCs do so by 1-Addressing deep submicron challenges (global wires, global timing closure, etc.), and 2-Offering a structured view on communication between IPs inspired by protocol stacks".

Recently, STMicroelectronics has announced the on-chip interconnect technology details of which the company has developed to meet the increasingly demanding needs of current and future SoC designs [95]. The new technology, called ST-NoC, is derived from its research in NoC technologies. They have declaimed that the availability of an effective NoC architecture is a crucial factor for cost-effective SoC solutions for next-generation convergence products and, in particular, NoC technology will play a major role in improving design productivity.

Moreover, the appearance of a number of start-up companies seeking to commercialize NoC-based ideas, in the recent years, emphasizes the urge of using NoC technologies in commercial products. Arteris is a French start-up company founded in 2003 by a group of semiconductor industry veterans. The company's focus is on Networks-on-Chip, the next generation of challenges associated with Systems-on-Chip design. Arteris calls itself 'The Network-on-chip Company' and its products comprise a suite of tools for generating and debugging a NoC and a library of configurable NoC components.

Another example is Silistix, a venture-funded start-up spun-out of the University of Manchester. The company focuses on the development and deployment of self-timed CHip-Area INterconnect (CHAIN) [48] technology for on-chip communication. As they said NoC is an example of a CHAIN system. CHAIN was demonstrated in a smart card implementation. Silistix distributes CHAINworks, a suite of software tools for the design and synthesis of customized on-chip interconnect using asynchronous (clock-less / self-timed) circuits.

All told, it is conjectured that Networks-on-Chip will be the backbone of all Systems-on-Chip of significant complexity design with new Deep Submicron technologies [77]. NoCs will be also an integral part of SoC platforms dedicated to specific application domains, and programming platforms with NoCs will be simpler due to regularity and predictability.

Appendix B

Asynchronous Circuits

Historically asynchronous circuits were used at the earliest days of the digital electronics, in the 50s, even before synchronous circuits. In reality all electronic systems are fundamentally asynchronous in nature [2]. By the careful insertion of localized timing relationships and storage elements, an asynchronous system can be adapted to appear to behave synchronously. In other words, the synchronous circuit design method was born to facilitate the asynchronous circuit difficulties by proposing to use a global timing reference signal named Clock. With the arrival of the integrated circuits, the synchronous design became popular and the dominant design style, due to the simple, easy-to-check, and one-sided timing constraint that it provides. Gradually the asynchronous circuit design paradigm returned to the academic research areas. Various details of different aspects of asynchronous design can be found in [96].

B.1. Weakness

The complexity of asynchronous design accounts for the apparent hesitation of industry to adopt such techniques. In synchronous design, every processing stage only needs to complete its activity in less than the duration of the clock cycle time and a designer does not have to worry about the dynamic state of the circuit. But in asynchronous design a great deal of attention has to be given to the dynamic state of the circuit. To exploit data-dependent evaluation, extra completion detection logic is necessary that sometimes is not evident and adds a more difficult design process.

Another issue is the test. The requirement for very high reliability of electronic systems in critical applications (for example in military and nuclear industries) imposes a rigorous test to detect fabrication faults. Testability is an inherent feature of today's electronic devices.

Unfortunately asynchronous systems are difficult to test due to the non-deterministic behavior of some elements, such as arbiters which make a mutual exclusive decision.

As synchronous design techniques are widely used and have been taught in universities for over three decades, most system designers are thus not familiar with asynchronous design techniques. As a result, almost all industrial CAD tools have been developed for synchronous systems and asynchronous design cannot employ them. The lack of suitable CAD tools is probably the main reason that asynchronous logic paradigm still remains mostly in the universities rather than in the industry.

B.2. Strong Points

In asynchronous architectures there is no global signal that needs to be distributed with minimal phase skew across the circuit. The absence of a global clock and consequently its distribution network has several benefits:

- **Power dissipation:** The total amount of energy dissipated within a clock distribution network in synchronous circuits is dramatic. For example it is stated in [2] that the clock distribution network at 600 MHz dissipates more than 44% of the total power consumed by an Alpha Microprocessor of 350nm technology. In addition to the high load of clock lines, one other important cause of this power dissipation is the fact that all parts of a synchronous circuit have to be clocked even if they are not involved in the current operation and are in idle state. Note that in order to reduce clock power dissipation some strategies such as Clock-Gating [68] are highly applicable to the design of effective clock distribution networks. But the implementation of such techniques is not easy to do and needs additional control logic and zero idle-state power consumption could not be achieved (only around 20-30% is often achievable). The power consumption not only affects the cost of energy, but also increases the packaging cost of the system, in order to remove the generated heat. Thanks to the absence of global clock signal, asynchronous circuits promise to have zero idle-state power consumption.
- **Silicon area:** The use of a huge number of clocked gates in high-scale architectures requires a huge number of clock buffers distributed across the clock distribution network, in order to provide sufficient fan-out. The disappearance of the clock distribution network in asynchronous circuits means disappearance of these clock buffers and consequently the release of a large amount of silicon area of the eventual components adjusting skews, such as Phase Locked Loops (PLL) or Delay Locked Loops (DLL).
- **Scalability:** The growth in number of components in a synchronous circuit means an increase in size of the chip and consequently in size of the clock distribution network, which is a limiting factor due to the physical issues explained before. Scaling number of modules in an asynchronous circuit has no impact.

- **Modularity:** The most common way to improve the performance of a synchronous system is to increase the frequency of the global clock (if physical limitations allow), while it will affect all parts of the system and usually requires a redesign for most of the system. In contrast, increasing performance of an asynchronous system can be achieved by locally modifying only the most critic part of the circuit and the rest of the system remains the same.
- **Average-Case performance:** The highest possible clock frequency in a synchronous system is limited by the worst-case combination of some parameters, such as power supply variation, temperature variation, transistor speed variation (e.g. due to the fabrication uncertainty), data-dependent operations, and data propagation delays. Typically, the worst-case combination is encountered very infrequently and the system performance usually is less than what it could be. Asynchronous circuits automatically adjust their speed of operation according to the current conditions and they are not restricted by a fixed clock frequency. Their operating speed is determined by actual local latencies rather than global worst-case and so the overall system performance refers to the average performance of local parts.
- **Reliability:** Even if a synchronous circuit operates in a frequency restricted by worst-case combination of several parameters, an unpredicted variation, for example in supply voltage, temperature, or fabrication process parameters, could cause a grave malfunction of the system. As a serious advantage, asynchronous circuits may offer robustness towards these types of variation. Normally the design of an asynchronous system is based on matched delays and can even be insensitive to circuit and wire delays.
- **Rapid technological migration:** Integrated circuits will often be implemented in several different fabrication technologies during their life-time. The delay insensitivity of certain asynchronous circuits gives a favorable ability to move more quickly from a prior technology to a new one. Asynchronous delay-insensitive IP cores previously designed could be instantiated in a new system using a new technology, by a plug-and-play fashion. The system will properly work, without any new timing constraint. The technological migration in synchronous design often emerges the need of a new timing analysis and requires likely a new design of some new critical path.
- **Electromagnetic compatibility:** This feature refers to the ability of an electrical device to work satisfactorily in its electromagnetic environment without influencing the surrounding devices, or being influenced by the surrounding devices. Electromagnetic Compatibility (EMC) is achieved by addressing the reduction of unintentional generation of electromagnetic energy in order to avoid the propagation of such energy towards the external environment. The global synchronization of a clocked system causes most of the switching activity to occur at the same instant. This effect concentrates the radiated energy emissions of the circuit at the harmonic frequencies of the clock. Asynchronous

circuits produce distributed interference spread across the entire frequency spectrum [97], because there are phase differences between the activities in different parts of the circuit and signals tend to tick at random points in time. In systems which use radio communication this feature can be a significant advantage.

- **Uniform power spectrum:** Hardware implementations of cryptographic algorithms are an essential part of modern digital systems. A cryptographic algorithm transforms plaintext information into ciphertext form with the help of a secret cipher key. The security provided by the algorithm is equal to its ability to protect the cipher key. Any cipher implementation unintentionally leaks information from a variety of runtime sources like power consumption, electromagnetic radiation, or even temperature variation. All these information sources are known as side channels. It was shown that parts of the cipher key can be extracted by processing side channel information [98]. For example side channel attacks that sample the power consumption of a cryptographic device while it operates and later use statistical methods to extract portions of the cipher key have proven to be extremely successful. These side channel attacks are known as Differential Power Analysis (DPA) attacks. Power dissipation highly depends on the switching activities produced by changes of data signals. The power spectrum of an asynchronous circuit is more uniform than that of a synchronous circuit. Lake of a global timing reference induces distributed switching activity spread across the entire operation time. Additionally, asynchronous circuits comprising dual or multi-rail data encoding can be balanced to reduce data dependent emissions. As a result of these abilities, recently asynchronous design methodologies became attractive to designers looking for methods to reduce the susceptibility of cryptographic hardware against DPA attacks [99].

In spite of all above mentioned promising features offered by asynchronous circuits, the lack of suitable CAD tools and design complexity is a strong limitation. A viable compromise is the use of hybrid systems that benefit from appropriate advantages of both synchronous and asynchronous design techniques. A system using an asynchronous NoC to provide GALS architectures exploiting synchronous IPs is the obvious example of such hybrid solutions.

B.3. Design Methodologies

Fundamentally in a synchronous circuit all components share a common and discrete notion of time, as defined by the global clock signal. In contrast in asynchronous circuits there is no common and discrete time. Instead, the circuits use handshaking between their components in order to perform the necessary synchronization, communication, and sequencing of the operations [62]. As an example of synchronous circuits Figure 74.a shows a data pipeline. The propagation of data between the pipeline stages is controlled by the clock signal. Typically a rising edge of the clock designates valid data at the input of a pipeline stage, while for the rest of time the data can propagate the eventual combinational circuits in between.

The equivalent asynchronous design shown in Figure 74.b represents an alternative solution. The clock signal is replaced by a handshake mechanism between each two adjacent pipeline stages. The handshaking is implemented based on simple request-acknowledge protocol. A pipeline stage may store a new data value from its predecessor only if the successor has stored the data value that the stage currently is holding. The sates of the previous and the following stages are signaled by the incoming request (*Req*) and acknowledge (*Ack*) signals respectively. Note that in order to provide full transparency, the combinational circuits between pipeline stages often participate in the handshaking.

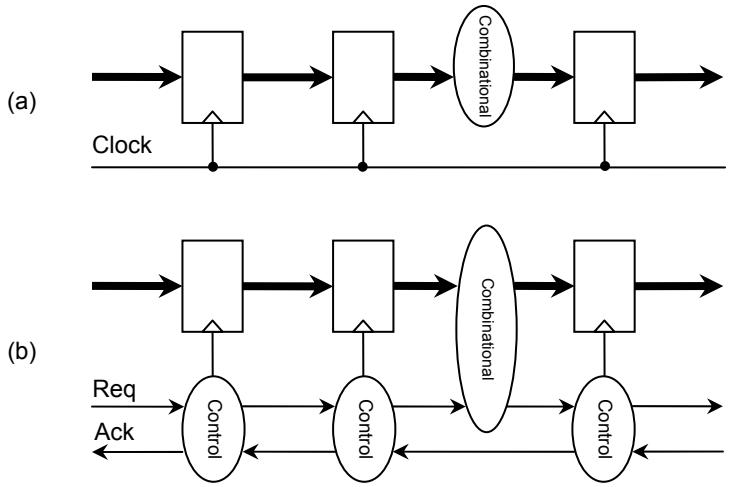


Figure 74. Two equivalent (a) Synchronous and (b) Asynchronous circuits of a data pipeline

B.3.1. Circuit Classification

Principally the asynchronous design methodologies can be classified on the assumptions commonly made with regard to the delays in gates and wires. The bounded delay models assume that the delay in all circuit components and wires is bounded (and even in somewhere known). Since it is the same delay model actually used for synchronous circuits, in this model circuits are designed in a similar way to synchronous circuits. On the other side in the unbounded delay model the delays are generally unknown. As a result, unbounded delay models lead to operate correctly whatever the distribution of delays.

B.3.1.1. Delay-Insensitive Circuits

In delay-insensitive (DI) circuits arbitrary delays are assumed for both gates and wires and the operation of the circuit is independent of the delays. In the design of delay-insensitive circuits the concept of acknowledgment has been introduced where every signal transition should be acknowledged by other signal transitions. This condition stops unseen transitions from occurring.

Such circuits are extremely robust, but unfortunately their class is rather small due to the heavy restrictions [100].

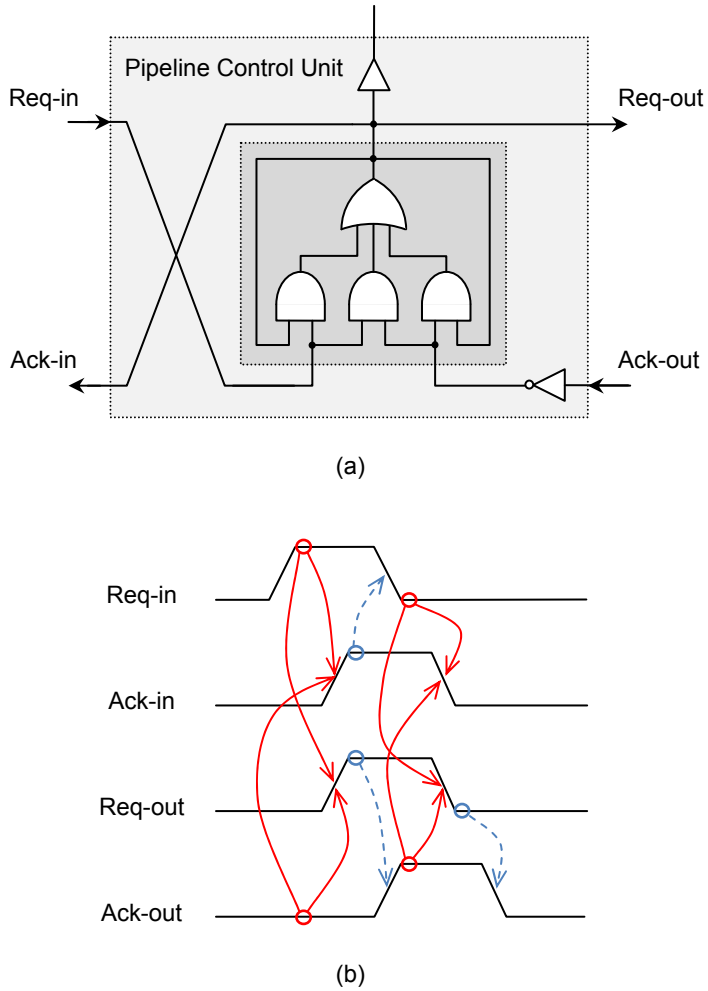


Figure 75. Asynchronous Pipeline Control Unit: (a) Circuit and (b) Signal Transitions

Figure 75.a displays an important example of delay-insensitive circuits that can be used as the control unit of the asynchronous pipelines illustrated in Figure 74.b. The behavior of the design is shown in Figure 75.b in a timing diagram of signal transitions. In the figure the curved arrows indicate the required sequence of transitions (the dashed blue arrows shows the behavior of the circuit's environment, i.e. the previous and next stage control units). There is no implicit assumption about the delay between successive transitions. *Req-Out* and *Ack-in* will be set to 1, when (and only when) *Req-in* is 1 (i.e. the predecessor has a data to be transferred) and *Ack-out* is 0 (i.e. the successor is ready to accept a new data transferring), and will be set to 0 when (and only when) *Req-in* is 0 (i.e. the predecessor has finished the data transferring) and *Ack-out* is 1 (i.e. the successor has accepted the request of data transferring).

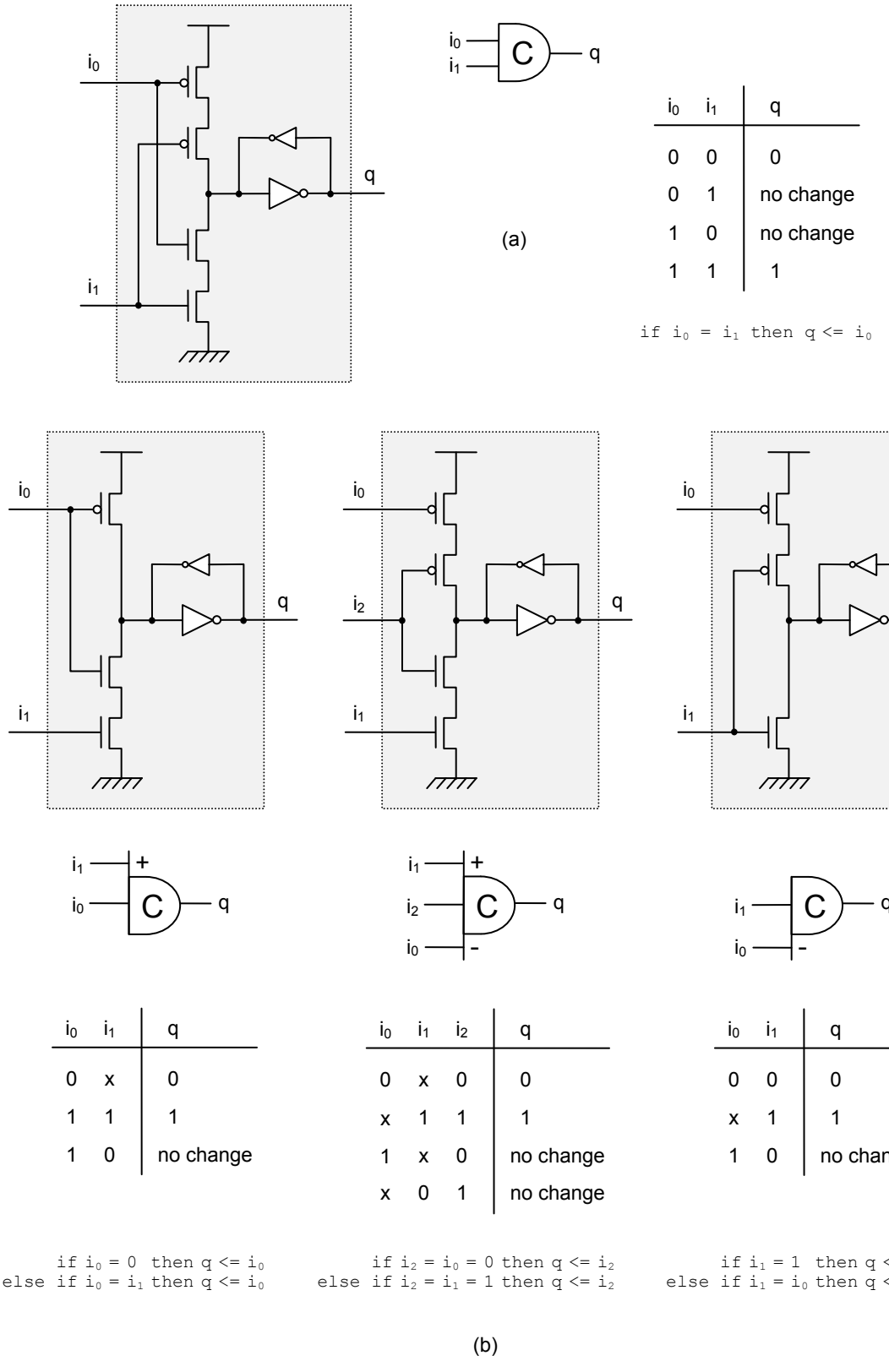


Figure 76. Muller C-elements: Implementations, Symbols, and Specifications

The block indicated by a dark gray square in the circuit of Figure 75.a plays an important role in the design of such circuits. In reality this part of the circuit is used for both synchronizing events and holding states. Due to this state-holding block the circuit remains at the previous state until required transitions of both inputs occur, independent of their delays.

Figure 76.a shows an atomic implementation of this module, called Muller C-element. When both inputs are 0 the output is set to 0, and when both inputs are 1 the output is set to 1. For other input combinations the output does not change. Consequently, when the output changes its state from 0 to 1 or from 1 to 0, one may conclude that both inputs are now at 1 or 0, respectively. In contrast, in the asymmetric variants of the C-element, illustrated in Figure 76.b, some inputs may only affect either the rising or falling output transition, not both. The Muller C-elements are fundamental components that are extensively used in asynchronous designs.

B.3.1.2. Quasi-Delay-Insensitive Circuits

In order to make wider the range of the asynchronous circuits that correctly can be designed and implemented in a CMOS technology, Quasi-Delay-Insensitive (QDI) circuits were introduced. In this delay model the delay of gates and wires is still considered to be arbitrary, but it is assumed that the delays of all branches of an interconnecting wire are identical. For example if d_2 equals d_3, the circuit fragment illustrated in Figure 77 is quasi-delay-insensitive and all other delays (d_A, d_B, d_C, and d_1) can be arbitrary. Most circuits that are referred to in the literature (incorrectly) as delay-insensitive are only quasi-delay-insensitive [62].

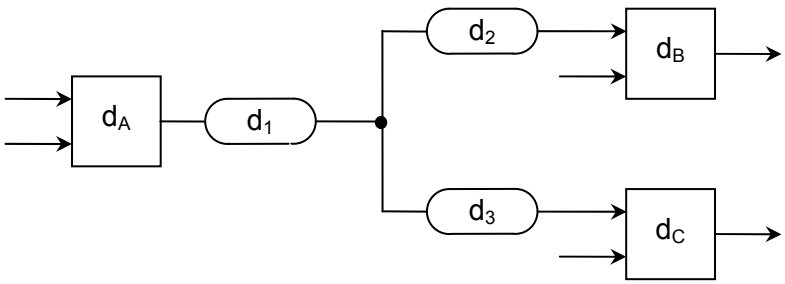


Figure 77. Gate and Wire Delays of a circuit fragment

B.3.1.3. Speed-Independent Circuits

In Speed-Independent (SI) circuits, while gate delays still exhibit arbitrary values, wire delay is considered to be zero, or in practice negligible compared to gate delays. The speed-independent model allows more implementation alternatives than quasi-delay-insensitive circuits, but it requires delay assumptions that can be difficult to realize in practice. However, although in large

circuits that wire delays are significant and thus technology mapping is almost impossible, for small circuits where all the wires are very short the speed-independent assumption is valid.

B.3.2. Handshake Protocols

Most information signaling mechanisms in asynchronous circuits are based on handshake protocols involving requests, which are used to initiate an action, and corresponding acknowledgments, used to signal completion of that action. In a Push model the sender initiates the action and pushes the information in the same direction as the request signal. A Pull model, on the other hand, implies that the receiver initiates the action and pulls the information in the same direction as the acknowledge signal. The flow direction of handshake signals in Push and Pull models is illustrated in Figure 78.

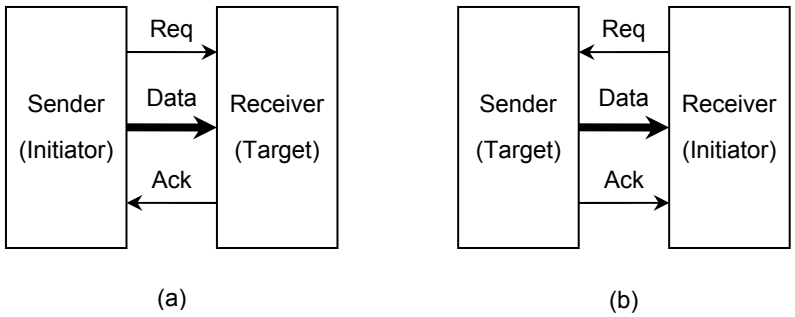


Figure 78. Flow Direction of Handshake Signals in (a) Push and (b) Pull Model

The strict alternation of the request and acknowledge handshake signals provides all the necessary sequence controls for events in the system. There are several choices of how these alternating events are dealt with. Two choices have been predominant that will be described here.

B.3.2.1. Four-Phase Protocol

Four-phase handshake protocol uses the level of signals to indicate events, hence called also level signaling. A high level of the request signal signifies the need for a new data transferring, and a high level of the acknowledge signal indicates the acceptance of the request. The handshake signals can easily be used to drive level-controlled latches, and as a consequence the four-phase handshake protocol is very simple in the implementation of control logic.

Assuming both the request (*Req*) and acknowledge (*Ack*) signals are initially low, as illustrated in Figure 79, in both push and pull variants of four-phase protocol the sequence of events exhibits a Return-To-Zero (RTZ) waveform. There are four transitions (two on *Req* and two on *Ack*) involved in the completion of each handshaking, and it is necessary that the signals return to the same level (zero) as they were in before the handshaking. While the required transitions for

returning to the original state (falling to zero) typically incur the loss of power, the advocates of this protocol argue that these transitions are often hidden by overlapping them with other actions in the circuits and thereby they do not usually cause the performance degradation.

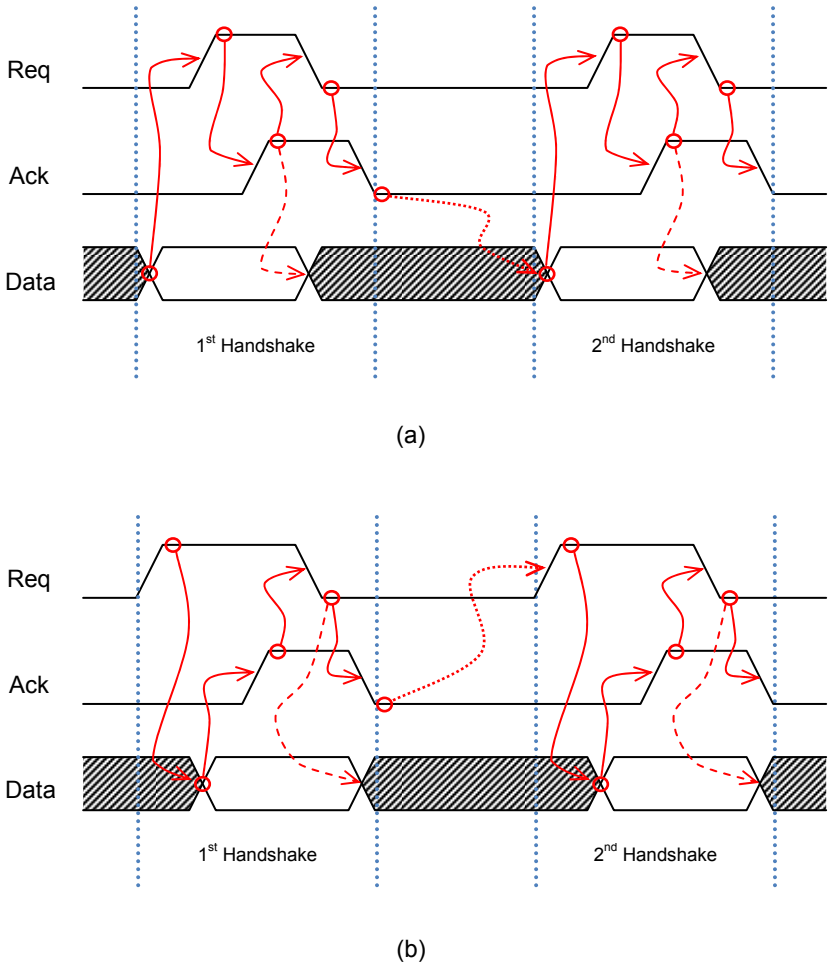


Figure 79. Four-Phase Handshake Protocol in (a) Push and (b) Pull Model

B.3.2.2. Two-Phase Protocol

Two-phase handshake protocol is a transition signaling and therefore events are indicated by transitions of the handshake signals, i.e. the rising and equivalently falling edges of the request signal represents the need for a new data transferring, and the rising and falling edges of the acknowledge signal implies the acceptance of the request. The level of these signals is not important.

The sequence of events using two-phase handshake protocol in both of the push and pull models, as shown in Figure 80, illustrates a Non-Return-to-Zero (NRZ) waveform. As a

consequence this handshake protocol results in potentially higher performance and lower power dissipation. But, the implementation of components that respond to signal transitions often is more complex and likely requires more logic.

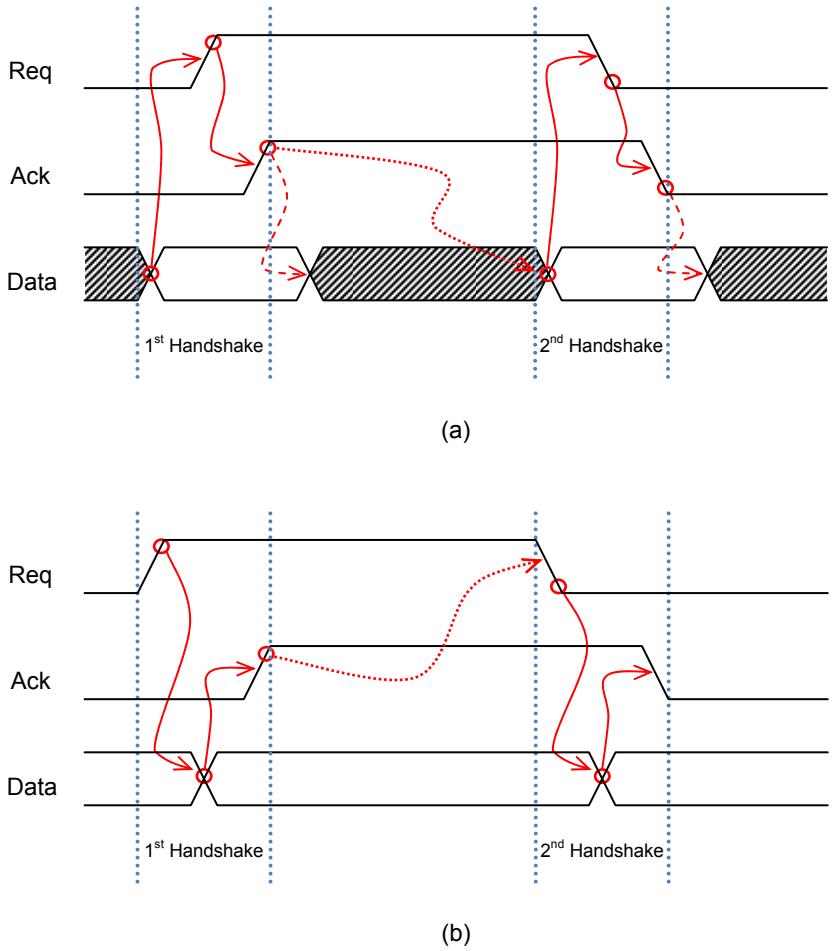


Figure 80. Two-Phase Handshake Protocol in (a) Push and (b) Pull Model

B.3.3. Data Encoding

In synchronous circuits data is typically represented as simple binary encoded signals, where each wire represents a single bit of information. The opposite, a further dimension in asynchronous design is the choice of data encoding schemes.

B.3.3.1. Bundled-Data Encoding

The term of bundled-data refers to a situation where the data signals use normal binary encoding (the same way as in a conventional synchronous system), and where separate request and

acknowledge wires are bundled with the data signals. The bundled-data encoding technique relies on an inherent timing assumption, in which the delay in data path must be less than the delay in wires signaling the data readiness, i.e. request in push model, and acknowledge in pull model.

Bundled-data encoding is very popular due to the similar area requirements to those of synchronous homologues. Additionally the synthesis of a data path can be done using standard synchronous tools. However the main difficulty with this data encoding in large circuits with long wires is that extra design effort is required in order to verify that the delay of the request (or acknowledge) signal matches the delays of data signals under all possible conditions.

B.3.3.2. Dual-Rail Encoding

Dual-rail encoding uses two wires to represent a single bit of information. The transfer of each bit will involve activity on only one of the two wires. Due to the fact that in dual-rail encoding it is possible to determine when a new data is valid (by detecting a level for 4-phase protocol, or an edge for 2-phase protocol, on one of the two rails), a separate handshake signal wire to convey data readiness is thus not necessary. Refer to Figure 81.

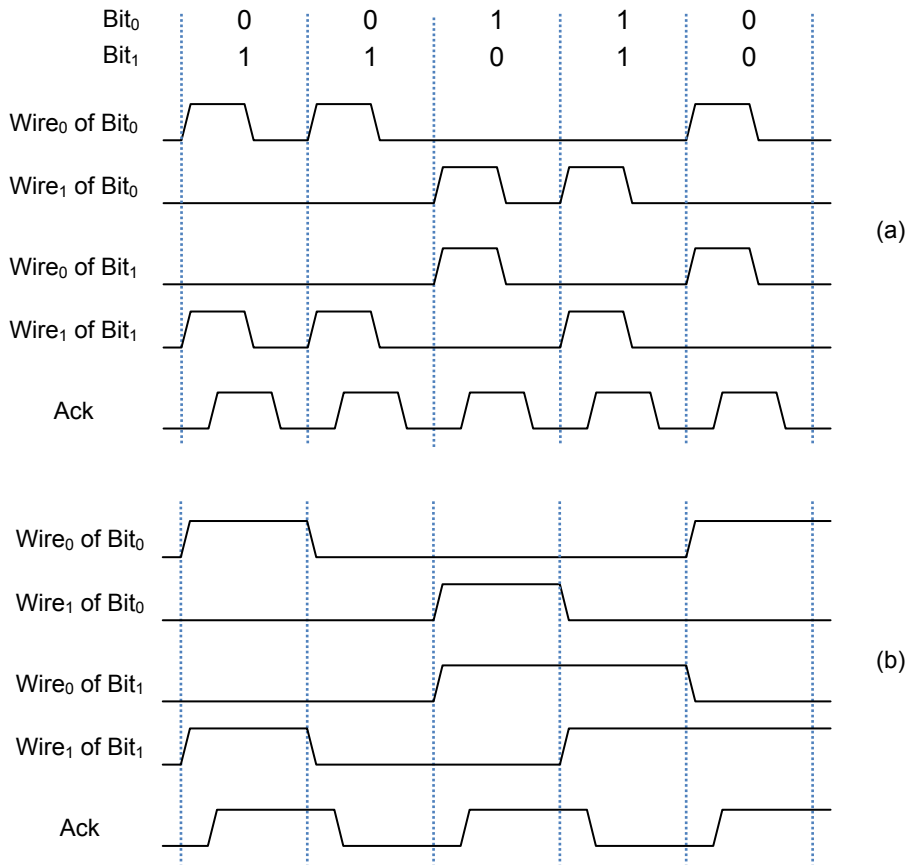


Figure 81. Dual-rail encoded data sequence example in (a) 4-phase and (b) 2-phase protocols

Dual-rail data encoding is a delay-insensitive code because, as said, the validity information is carried along every bit in the data word, and thereby the receiver is enable to unambiguously detect the word completion, regardless of delays. As an example, Figure 82 shows a circuit suitable for detecting the presence of a valid 4-bit data word using 4-phase handshake protocol (level signaling).

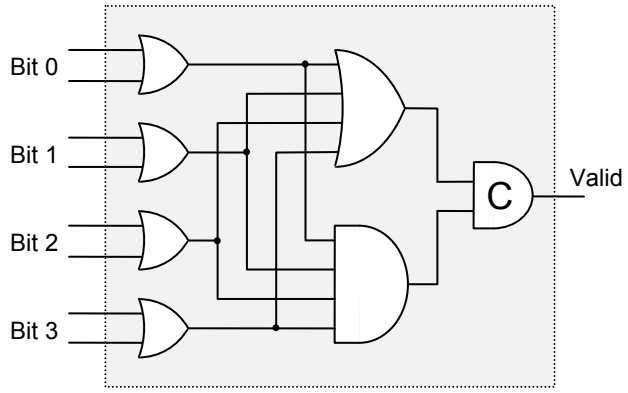


Figure 82. Dual-rail data word completion detector in four-phase handshake protocol

The main issue with dual-rail encoding is that it results in a significant overhead in the excess wiring, extra power dissipation, and large area, due to the duplication of logic functions and the supplementary circuit of completion detection.

B.3.3.3. n-of-m Encoding

The general term of n-of-m represents a delay-insensitive data encoding which use actions on n of the m wires in a group to indicate one of a set of possible codes. Dual-rail data encoding, described above, is an example of n-of-m encoding, where m and n equal 2 and 1 respectively.

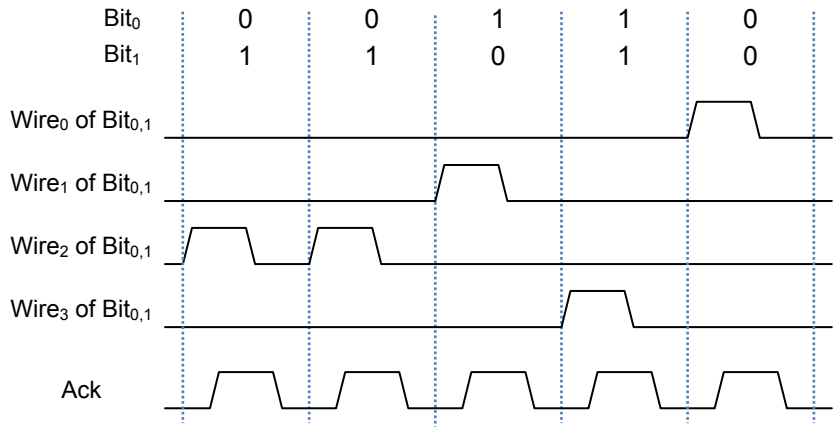


Figure 83. One-of-four encoded data sequence example in 4-phase handshake protocol

One-of-four data encoding is another popular example. Compared with dual-rail, one-of-four encoding scheme reduces the switching activity of the system. In this case four wires are used to transmit two bits of data in parallel, using only one active signal to designate a valid code symbol. Figure 83 illustrates the one-of-four encoded form of the dual-rail encoded data sequence shown in Figure 81.a. It can be observed that one-of-four encoding generates roughly 50% fewer signal transitions.

More complex codes which use more than one active wire in each group (i.e. n > 1) may offer better utilization of the available wires. For example a 2-of-7 encoding employs seven wires to transmit four bits using only two active signals per symbol, while dual-rail and one-of-four data encoding requires eight wires to transmit the same number of bits. In general, k bits could be designated by 2^k codes, where total number of codes that could be transmitted with m wires using n-of-m encoding is:

$$Total\ Number\ of\ Codes = \frac{m!}{n!(m-n)!}$$

However, n-of-m encoding mechanisms, where n>1, results in larger arithmetic and completion detection circuits and conversion between encoded from a single wire scheme is more expensive that make them unattractive for implementation on silicon.

B.3.4. Metastability

The metastability is likely encountered when a bistable system with hysteresis must determine an ordering of two asynchronous inputs that occur almost simultaneously. For example a C-element can be in metastable state if its two inputs change at the same time but to the opposite value. See Figure 84.a. In asynchronous circuits the choice of a correct handshake protocol prevents transitions probably causing the metastability.

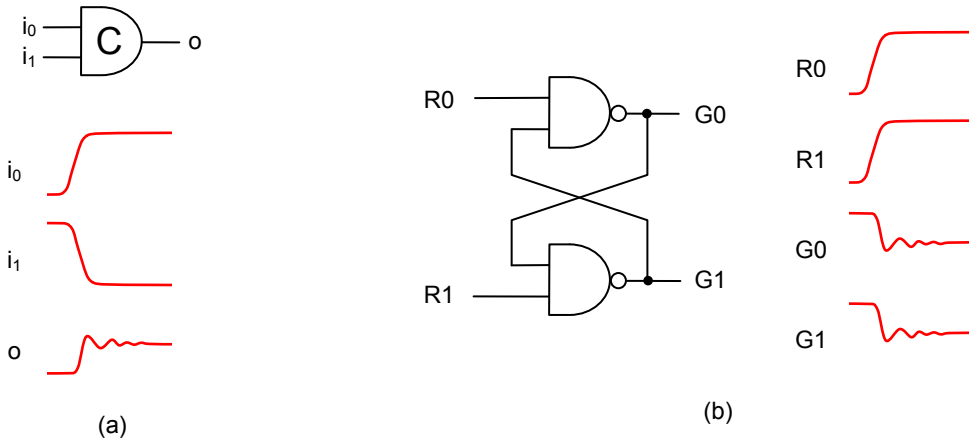


Figure 84. Metastability in (a) C-element and (b) circuit fragment arbitrating between two mutually exclusive request signals

However, in any kind of the asynchronous handshake protocols when two request signals are contending to be the first, there is not a way to avoid the metastability. The circuit fragment shown in Figure 84.b is popularly used to arbitrate between two mutually exclusive request signals. These cross-coupled NAND gates enable one input to block the other. But if both requests arrive approximately at the same time the circuit likely becomes metastable with both outputs.

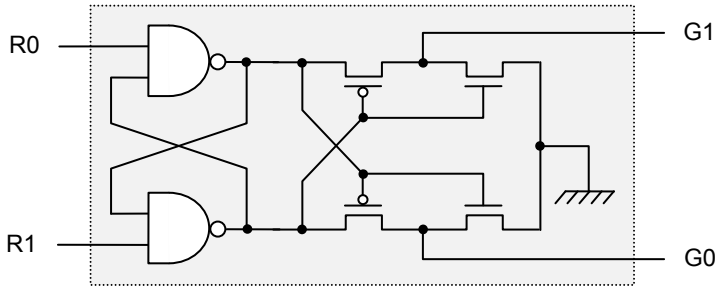


Figure 85. Mutual Exclusion (MUTEX) Component

Recall that the probability that the metastable state continue decreases exponentially with time. This is accommodated in a synchronous system by waiting for a predefined period (a clock cycle for example) before using outputs susceptible to be in the metastable state. Thereby, the possibility of failure is reduced to an acceptable value. Asynchronous design uses the alternative approach of waiting until the metastability has been totally resolved. As shown in Figure 85, it can be achieved by using a metastability filter attached to the mutual exclusion (MUTEX) circuit, preventing the undefined (undesirable) value of the outputs to pass into the rest of the system. If the rare occasional long delay can be tolerated then arbitration between contending signals can be failure free.

B.3.5. Signal Transition Graph

To specify an asynchronous circuit there is a number of techniques available. But, for most of them, sadly, there is not a synthesis route. As a commonly used technique for which a synthesis way currently exists, the behavior of the intended circuit and its environment can be described as signal transition graphs. A Signal Transition Graph (STG) [101] is a Petri Net [102] and it can be seen as a formalization of a timing diagram. Unfortunately, defining an STG is not easy to make for large systems. One may use this technique for the construction of small modules. Larger designs can then be created from compositions of these small components.

An STG is a graph composed of directed arcs and two types of nodes: transitions and places. Some places can be marked with tokens and the STG can be executed by firing transitions. A transition is enabled to fire if there are tokens on all of its input places, and an enabled transition must eventually fire. When a transition fires, a token is removed from each input place and a

token is added to each output place. A transition is labeled with either a '+' (representing a rising signal) or a '-' (representing a falling signal). Dependencies and causalities are represented in the STG using the notations shown in Figure 86.

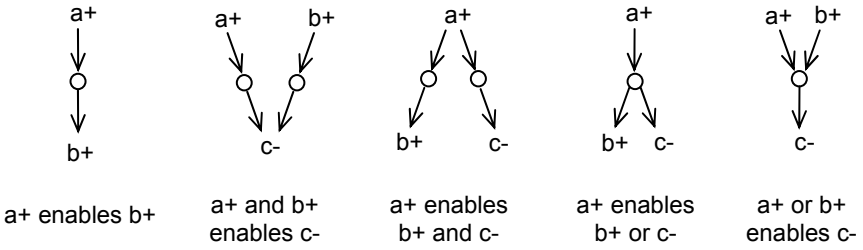


Figure 86. STG Notations

B.3.5.1. Implementations using State-Holding Elements

During operation each variable (internal states and output signals) in the circuit will go through a sequence of 0, followed by one or more states where it is excited to be reset, followed by a sequence of 1, followed by one or more states where it is excited to be set. The starting point for the implementation of an STG using state-holding elements is to determine the Boolean equations for the set and reset signals of the circuit variables. Figure 87 shows the implementation template using a standard Muller C-element (as a state-holding element). Note that there are a number of alternative solutions for implementing variables using a state-holding device. It is obvious that for all reachable states the set and reset functions for a signal z must never be active at the same time.

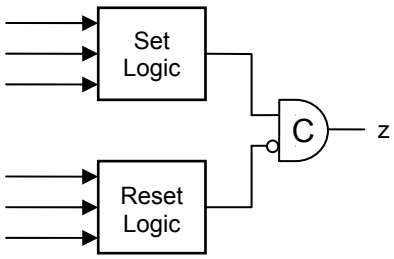


Figure 87. Implementation Template using standard Muller C-element

The idea of using a state-holding device for each variable (each non-input signal) can be formalized as follows [62]:

- An excitation region, ER, for a variable z is a maximally-connected set of states in which the variable is excited:
    - ER(z+) denotes a region of states where z is excited to be 1

    - ER(z-) denotes a region of states where z is excited to be 0
- A quiescent region, QR, for a variable z is a maximally-connected set of states in which the variable is not excited:
    - QR(z+) denotes a region of states where z is 1
    - QR(z-) denotes a region of states where z is 0

For a given circuit the state space can be divided into one or more regions of each type.

- The set function for a variable z:
    - Must contain all states in the ER(z+) regions
    - May contain states from the QR(z+) regions
    - May contain states not reachable by the circuit
- The reset function for a variable z:
    - Must contain all states in the ER(z-) regions
    - May contain states from the QR(z-) regions
    - May contain states not reachable by the circuit
- In order to avoid hazards, a product term in the set or reset function of a variable must only be entered through a state where the variable is excited

B.3.5.2. Initialization

Initialization is an important aspect of circuit design. Since the circuits generally use state-holding elements or circuitry with feedback loops, it is necessary to actively force the circuit into a well defined initial state. Consequently the designer has to extend the circuit with an extra signal which, when active, sets (or causes) all state-holding constructs into the desired state. It is assumed that initialization signal is asserted for long enough to cause the desired actions before it is de-asserted.

The initialization of an asynchronous circuit based on handshake components, can be achieved by an implicit approach that exploits the function of the circuit to propagate initial signal values into the circuit. Sometimes we have to add the initialization signal to the relevant Boolean equations explicitly, and occasionally the modified logic equations need further logic decomposition.

Appendix C

Basic Cells of Asynchronous Circuits for SXLIB

SXLIB is the design rule independent CMOS standard cell library of the ALLIANCE package developed in LIP6 [65]. It is supposed to be used with a usual standard cells place and route tool. Each object in the library has, for static ones, or produces, for dynamics ones, three views:

- The behavior, specified in VHDL data flow form
- The net-list, in terms of transistor interconnections
- The symbolic layout, that describes the cell topology

To include the specific basic cells of asynchronous circuits, used in the designs described in this thesis, SXLIB has been completed. The new cells will be presented as follows:

C.1. Muller C-element

– **Behavior Model in VHDL**

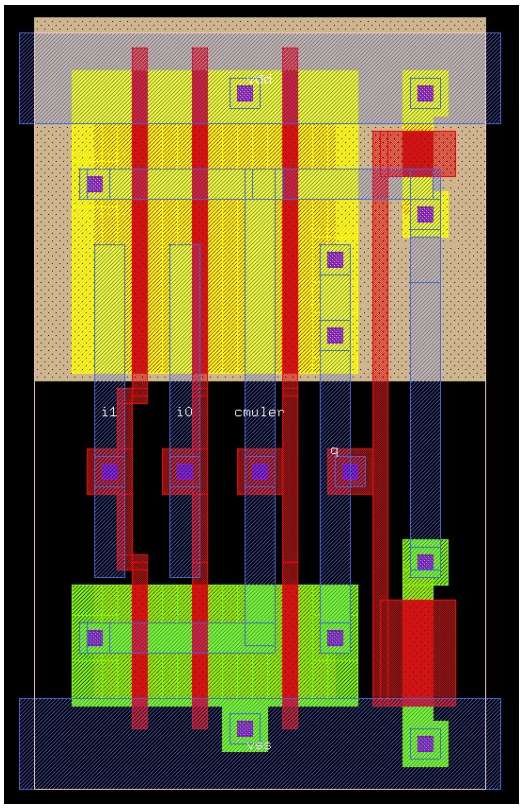
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY cmuler IS
PORT(
  i0    : IN STD_LOGIC;
  i1    : IN STD_LOGIC;
  q     : OUT STD_LOGIC;
  vdd   : IN STD_LOGIC;
  vss   : IN STD_LOGIC
);
END cmuler;

ARCHITECTURE RTL OF cmuler IS
  SIGNAL mem    : STD_LOGIC;
BEGIN
  q <= transport mem after 10 ps;
  PROCESS ( i1, i0 )
```

```
  BEGIN
    IF ((i0 AND i1) = '1')
    THEN mem <= '1';
    ELSIF ((NOT(i0) AND NOT(i1)) = '1')
    THEN mem <= '0';
    END IF;
  END PROCESS;
END RTL;
```

– Transistor Net-List in Spice

```
* INTERF i0 i1 q vdd vss

.subckt cmuler 1 2 7 4 8

* NET 1 = i0
* NET 2 = i1
* NET 4 = vdd
* NET 7 = q
* NET 8 = vss

xtr_00008 3 2 5 4 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00007 4 1 3 4 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00006 7 5 4 4 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00005 5 7 4 4 plvt tometer=1 L=0.28U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00004 6 2 5 8 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00003 8 1 6 8 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00002 7 5 8 8 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00001 8 7 5 8 nlvt tometer=1 L=0.64U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U

.ends cmuler
```

– Physical Layout



C.2. Muller Upper Asymmetric C-element

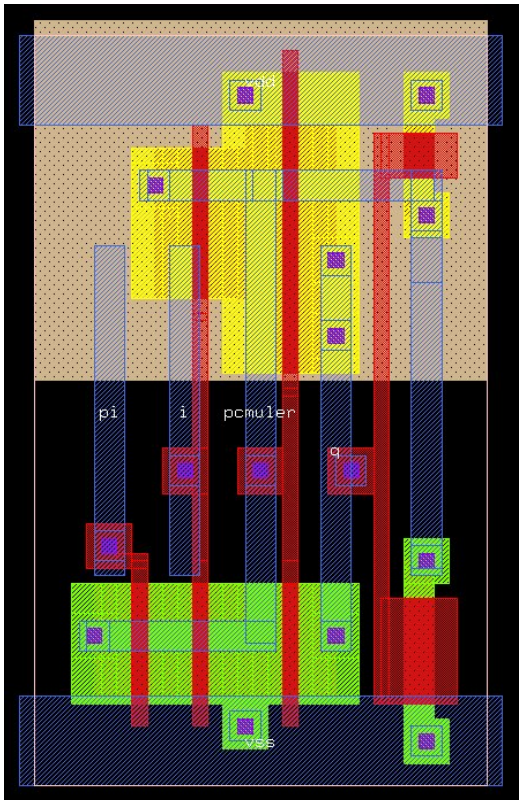
– Behavior Model in VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY pcmuler IS
PORT(
  i     : IN STD_LOGIC;
  pi    : IN STD_LOGIC;
  q     : OUT STD_LOGIC;
  vdd   : IN STD_LOGIC;
  vss   : IN STD_LOGIC
);
END pcmuler;

ARCHITECTURE RTL OF pcmuler IS
  SIGNAL mem    : STD_LOGIC;
BEGIN
  q <= transport mem after 10 ps;
  PROCESS ( pi, i )
  BEGIN
    IF ((i AND pi) = '1')
    THEN mem <= '1';
    ELSIF (NOT(i) = '1')
    THEN mem <= '0';
    END IF;
  END PROCESS;
END RTL;
```

– Transistor Net-List in Spice

```
* INTERF i pi q vdd vss

.subckt pcmuler 2 1 4 3 6

* NET 1 = pi
* NET 2 = i
* NET 3 = vdd
* NET 4 = q
* NET 6 = vss

xtr_00007 7 4 3 3 plvt tometer=1 L=0.28U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00006 4 7 3 3 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00005 3 2 7 3 plvt tometer=1 L=0.1U W=0.87U AS=0.1305P AD=0.1305P PS=2.05U PD=2.05U
xtr_00004 6 4 7 6 nlvt tometer=1 L=0.64U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00003 4 7 6 6 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00002 6 2 5 6 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00001 5 1 7 6 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U

.ends pcmuler
```

– **Physical Layout**



C.3. Muller Lower Asymmetric C-element

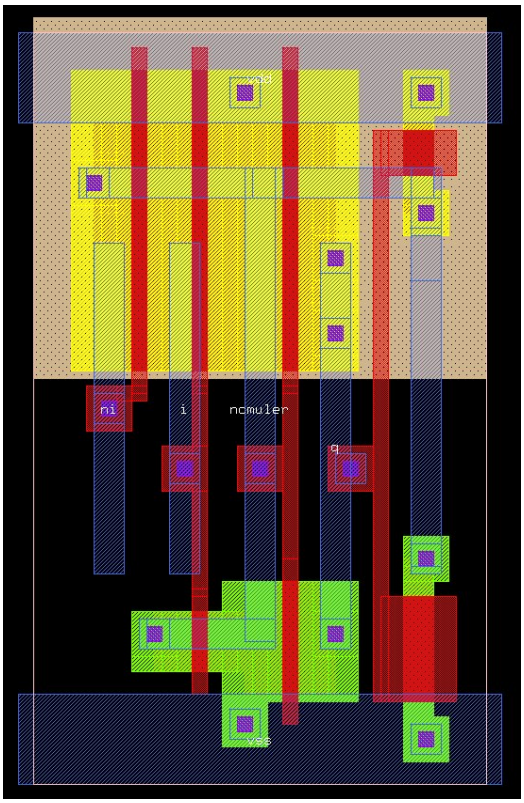
– Behavior Model in VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY ncmuler IS
PORT(
  i     : IN STD_LOGIC;
  ni    : IN STD_LOGIC;
  q     : OUT STD_LOGIC;
  vdd   : IN STD_LOGIC;
  vss   : IN STD_LOGIC
);
END ncmuler;

ARCHITECTURE RTL OF ncmuler IS
  SIGNAL mem    : STD_LOGIC;
BEGIN
  q <= transport mem after 10 ps;
  PROCESS ( ni, i )
  BEGIN
    IF (i = '1')
    THEN mem <= '1';
    ELSIF ((NOT(ni) AND NOT(i)) = '1')
    THEN mem <= '0';
    END IF;
  END PROCESS;
END RTL;
```

– Transistor Net-List in Spice

```
* INTERF i ni q vdd vss

.subckt ncmuler 2 1 5 4 7

* NET 1 = ni
* NET 2 = i
* NET 4 = vdd
* NET 5 = q
* NET 7 = vss

xtr_00007 3 1 6 4 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00006 4 2 3 4 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00005 5 6 4 4 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00004 6 5 4 4 plvt tometer=1 L=0.28U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00003 5 6 7 7 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00002 7 5 6 7 nlvt tometer=1 L=0.64U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00001 7 2 6 7 nlvt tometer=1 L=0.1U W=0.33U AS=0.0495P AD=0.0495P PS=0.97U PD=0.97U

.ends ncmuler
```

– **Physical Layout**



C.4. Muller Generalized C-element

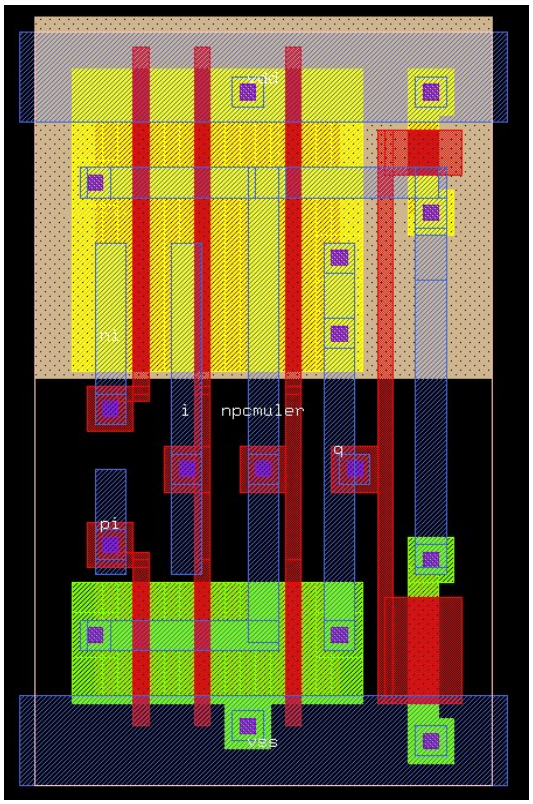
– Behavir Model in VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY npcmuler IS
PORT(
  i     : IN STD_LOGIC;
  pi    : IN STD_LOGIC;
  ni    : IN STD_LOGIC;
  q     : OUT STD_LOGIC;
  vdd   : IN STD_LOGIC;
  vss   : IN STD_LOGIC
);
END npcmuler;

ARCHITECTURE RTL OF npcmuler IS
  SIGNAL mem    : STD_LOGIC;
BEGIN
  q <= transport mem after 10 ps;
  PROCESS ( ni, pi, i )
  BEGIN
    IF ((i AND pi) = '1')
    THEN mem <= '1';
    ELSIF ((NOT(i) AND NOT(ni)) = '1')
    THEN mem <= '0';
    END IF;
  END PROCESS;
END RTL;
```

– Transistor Net-List in Spice

```
* INTERF i ni pi q vdd vss

.subckt npcmuler 2 3 1 6 5 8

* NET 1 = pi
* NET 2 = i
* NET 3 = ni
* NET 5 = vdd
* NET 6 = q
* NET 8 = vss

xtr_00008 9 6 5 5 plvt tometer=1 L=0.28U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00007 6 9 5 5 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00006 5 2 4 5 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00005 4 3 9 5 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00004 8 6 9 8 nlvt tometer=1 L=0.64U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00003 6 9 8 8 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00002 8 2 7 8 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00001 7 1 9 8 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U

.ends npcmuler
```

– **Physical Layout**



C.5. MUTEX

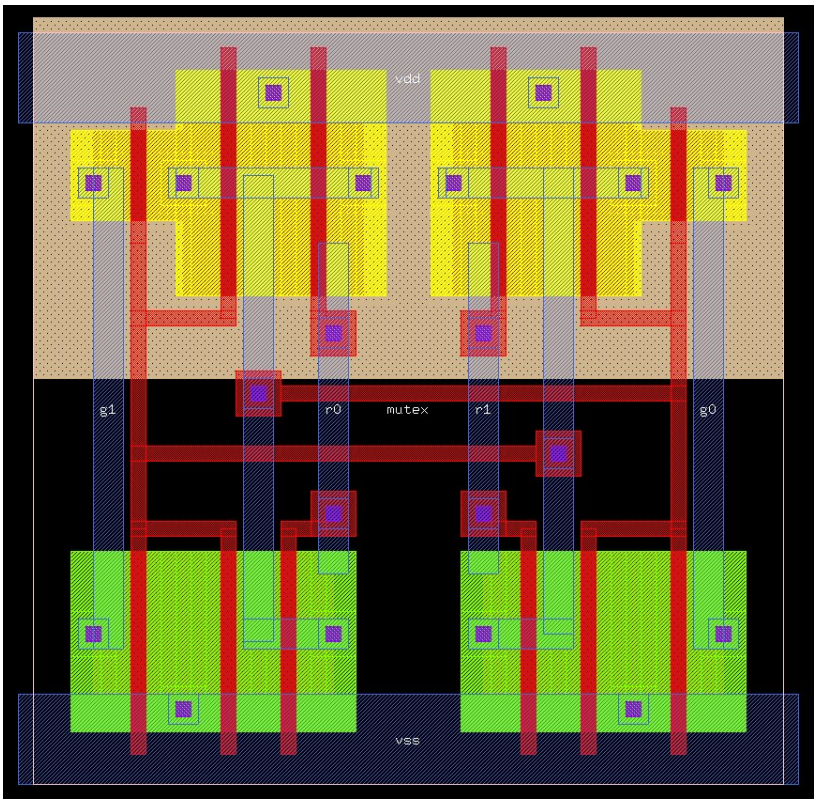
– Behavior Model in VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY mutex IS
PORT(
  r0    : IN STD_LOGIC;
  r1    : IN STD_LOGIC;
  g0    : OUT STD_LOGIC;
  g1    : OUT STD_LOGIC;
  vdd   : IN STD_LOGIC;
  vss   : IN STD_LOGIC
);
END mutex;

ARCHITECTURE RTL OF mutex IS
  SIGNAL x1     : STD_LOGIC;
  SIGNAL x0     : STD_LOGIC;
BEGIN
  g0 <= x0;
  g1 <= x1;
  x0 <= transport (NOT x1 and r0) or (not r1 and r0) after 10 ps;
  x1 <= transport (    x1 AND r1) or (not r0 and r1) after 10 ps;
END RTL;
```

– Transistor Net-List in Spice

```
* INTERF g0 g1 r0 r1 vdd vss


.subckt mutex 2 8 5 4 1 9

* NET 1 = vdd
* NET 2 = g0
* NET 4 = r1
* NET 5 = r0
* NET 8 = g1
* NET 9 = vss

xtr_00012 7 6 1 1 plvt tometer=1 L=0.1U W=1.32U AS=0.198P AD=0.198P PS=2.95U PD=2.95U
xtr_00011 1 4 7 1 plvt tometer=1 L=0.1U W=1.32U AS=0.198P AD=0.198P PS=2.95U PD=2.95U
xtr_00010 2 6 7 1 plvt tometer=1 L=0.1U W=0.51U AS=0.0765P AD=0.0765P PS=1.33U PD=1.33U
xtr_00009 6 7 8 1 plvt tometer=1 L=0.1U W=0.51U AS=0.0765P AD=0.0765P PS=1.33U PD=1.33U
xtr_00008 1 7 6 1 plvt tometer=1 L=0.1U W=1.32U AS=0.198P AD=0.198P PS=2.95U PD=2.95U
xtr_00007 6 5 1 1 plvt tometer=1 L=0.1U W=1.32U AS=0.198P AD=0.198P PS=2.95U PD=2.95U
xtr_00006 7 4 3 9 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U
xtr_00005 3 6 9 9 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U
xtr_00004 9 6 2 9 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U
xtr_00003 8 7 9 9 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U
xtr_00002 9 7 10 9 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U
xtr_00001 10 5 6 9 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U

.ends mutex
```

– **Physical Layout**



C.6. RS Flip-Flop

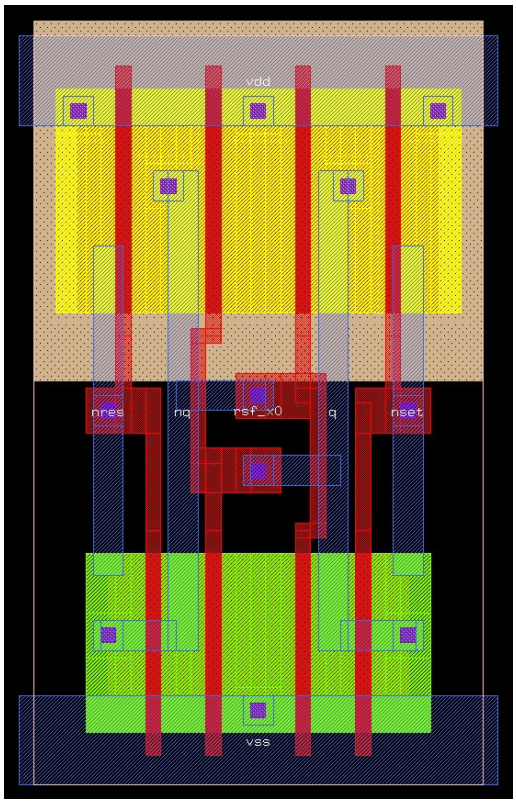
– Behavior Model in VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY rsf_x0 IS
PORT(
  nres  : IN STD_LOGIC;
  nset  : IN STD_LOGIC;
  nq    : OUT STD_LOGIC;
  q     : OUT STD_LOGIC;
  vdd   : IN STD_LOGIC;
  vss   : IN STD_LOGIC
);
END rsf_x0;

ARCHITECTURE RTL OF rsf_x0 IS
  SIGNAL me_m   : STD_LOGIC;
BEGIN
  nq <= transport NOT(me_m) after 10 ps;
  q <= transport me_m after 10 ps;
  PROCESS ( nres, nset )
  BEGIN
    IF (NOT(nset) = '1')
    THEN me_m <= '1';
    ELSIF ((NOT(nres)  ) = '1')
    THEN me_m <= '0';
    END IF;
  END PROCESS;
END RTL;
```

– Transistor Net-List in Spice

```
* INTERF nq nres nset q vdd vss

.subckt rsf_x0 5 2 3 6 1 8

* NET 1 = vdd
* NET 2 = nres
* NET 3 = nset
* NET 5 = nq
* NET 6 = q
* NET 8 = vss

xtr_00008 1 3 6 1 plvt tometer=1 L=0.1U W=1.32U AS=0.198P AD=0.198P PS=2.95U PD=2.95U
xtr_00007 6 5 1 1 plvt tometer=1 L=0.1U W=1.32U AS=0.198P AD=0.198P PS=2.95U PD=2.95U
xtr_00006 1 6 5 1 plvt tometer=1 L=0.1U W=1.32U AS=0.198P AD=0.198P PS=2.95U PD=2.95U
xtr_00005 5 2 1 1 plvt tometer=1 L=0.1U W=1.32U AS=0.198P AD=0.198P PS=2.95U PD=2.95U
xtr_00004 7 3 6 8 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U
xtr_00003 8 5 7 8 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U
xtr_00002 4 6 8 8 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U
xtr_00001 5 2 4 8 nlvt tometer=1 L=0.1U W=1.05U AS=0.1575P AD=0.1575P PS=2.41U PD=2.41U

.ends rsf_x0
```

– **Physical Layout**



C.7. Latch

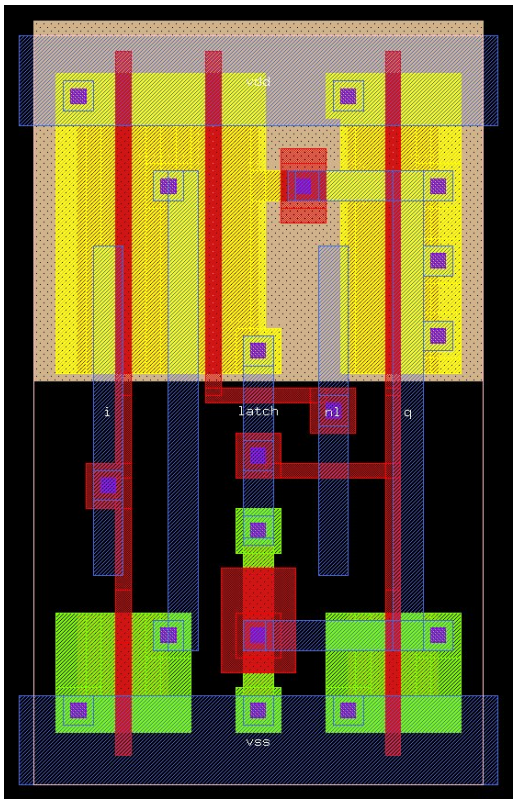
– Behavior Model in VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY latch IS
PORT(
  nl    : IN STD_LOGIC;
  i     : IN STD_LOGIC;
  q     : OUT STD_LOGIC;
  vdd   : IN STD_LOGIC;
  vss   : IN STD_LOGIC
);
END latch;

ARCHITECTURE RTL OF latch IS
  SIGNAL latch_m        : STD_LOGIC;
BEGIN
  q <= transport latch_m after 10 ps;
  PROCESS ( i, nl )
  BEGIN
    IF (NOT(nl) = '1')
    THEN latch_m <= i;
    END IF;
  END PROCESS;
END RTL;
```

– Transistor Net-List in Spice

```
* INTERF i nl q vdd vss

.subckt latch 2 1 6 3 7

* NET 1 = nl
* NET 2 = i
* NET 3 = vdd
* NET 6 = q
* NET 7 = vss

xtr_00007 6 5 3 3 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00006 5 6 3 3 plvt tometer=1 L=0.28U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00005 5 1 4 3 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00004 4 2 3 3 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00003 7 6 5 7 nlvt tometer=1 L=0.64U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00002 6 5 7 7 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00001 4 2 7 7 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U

.ends latch
```

– **Physical Layout**



C.8. Magic

– Behavior Model in VHDL

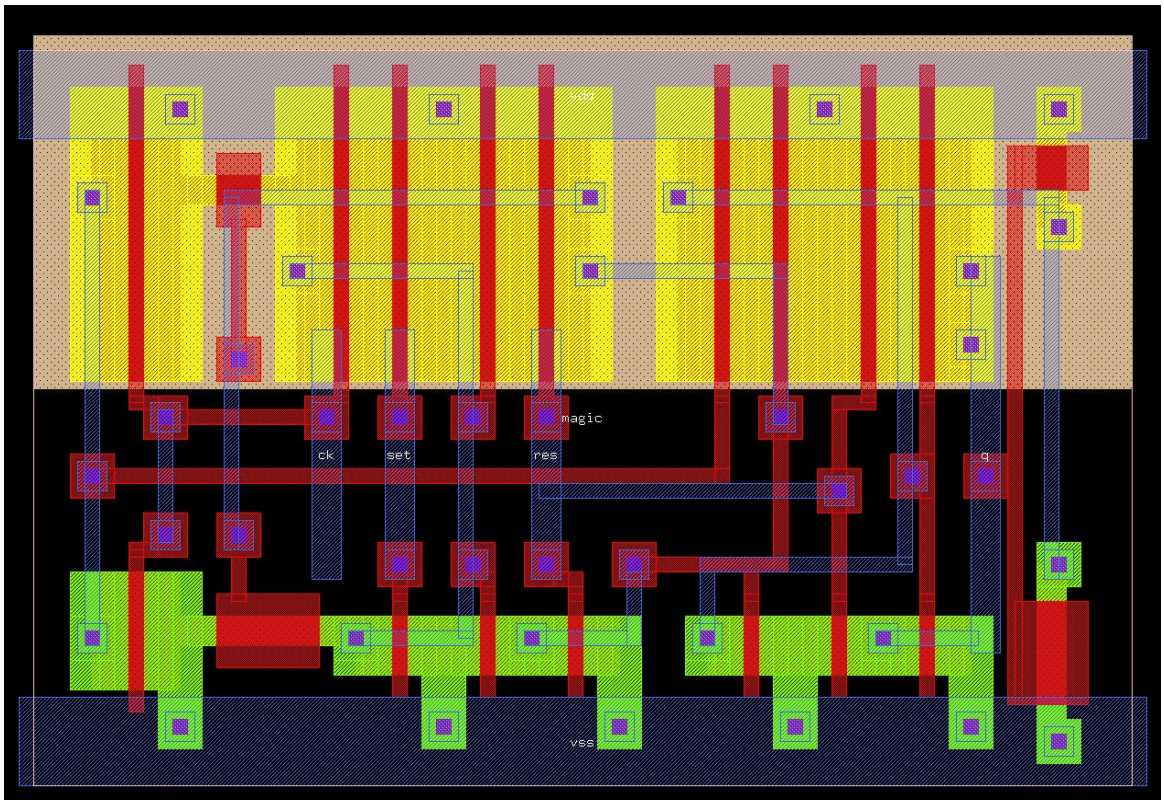
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY magic IS
PORT(
  set   : IN STD_LOGIC;
  res   : IN STD_LOGIC;
  ck    : IN STD_LOGIC;
  q     : OUT STD_LOGIC;
  vdd   : IN STD_LOGIC;
  vss   : IN STD_LOGIC
);
END magic;

ARCHITECTURE RTL OF magic IS
  SIGNAL mem    : STD_LOGIC;
BEGIN
  q <= transport mem after 10 ps;
  PROCESS ( ck, res, set )
  BEGIN
    IF ((set AND NOT(res)) = '1')
    THEN mem <= '1';
    ELSIF (res = '1')
    THEN mem <= '0';
    ELSIF  ((ck = '1') AND ck'EVENT)
    THEN mem <= '0';
    END IF;
  END PROCESS;
END RTL;
```

– Transistor Net-List in Spice

```
* INTERF ck q res set vdd vss

.subckt magic 7 6 1 8 10 13

* NET 1 = res
* NET 6 = q
* NET 7 = ck
* NET 8 = set
* NET 10 = vdd
* NET 13 = vss

xtr_00020 5 6 10 10 plvt tometer=1 L=0.28U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00019 6 5 3 10 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00018 3 1 10 10 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00017 10 9 4 10 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00016 4 12 5 10 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00015 9 1 2 10 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00014 2 14 10 10 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00013 10 8 11 10 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00012 11 7 14 10 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00011 10 9 14 10 plvt tometer=1 L=0.28U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
xtr_00010 10 7 12 10 plvt tometer=1 L=0.1U W=1.77U AS=0.2655P AD=0.2655P PS=3.85U PD=3.85U
xtr_00009 13 5 6 13 nlvt tometer=1 L=0.1U W=0.33U AS=0.0495P AD=0.0495P PS=0.97U PD=0.97U
xtr_00008 6 1 13 13 nlvt tometer=1 L=0.1U W=0.33U AS=0.0495P AD=0.0495P PS=0.97U PD=0.97U
xtr_00007 13 9 5 13 nlvt tometer=1 L=0.1U W=0.33U AS=0.0495P AD=0.0495P PS=0.97U PD=0.97U
xtr_00006 13 1 9 13 nlvt tometer=1 L=0.1U W=0.33U AS=0.0495P AD=0.0495P PS=0.97U PD=0.97U
xtr_00005 9 14 13 13 nlvt tometer=1 L=0.1U W=0.33U AS=0.0495P AD=0.0495P PS=0.97U PD=0.97U
xtr_00004 13 8 14 13 nlvt tometer=1 L=0.1U W=0.33U AS=0.0495P AD=0.0495P PS=0.97U PD=0.97U
xtr_00003 13 9 14 13 nlvt tometer=1 L=0.64U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U
```

```
xtr_00002 13 7 12 13 nlvt tometer=1 L=0.1U W=0.69U AS=0.1035P AD=0.1035P PS=1.69U PD=1.69U
xtr_00001 13 6 5 13 nlvt tometer=1 L=0.64U W=0.15U AS=0.0225P AD=0.0225P PS=0.61U PD=0.61U

.ends magic
```

– **Physical Layout**



Publications

- Multi-Synchronous & Fully Asynchronous Networks-on-Chip for GALS Architectures
  - Sheibanyrad Abbas, Greiner Alain, Miro Panades Ivan
  - accepted to appear in IEEE Design & Test of Computer magazine

- Two Efficient Synchronous ⇔ Asynchronous Converters well-suited for Networks-on-Chip in GALS Architectures
  - Sheibanyrad Abbas, Greiner Alain
  - INTEGRATION, the VLSI journal, Jan. 2008, vol. 41, num. 1, pp. 17-26, ELSEVIER

- Systematic Comparison between the Asynchronous and the Multi-Synchronous Implementations of a Network on Chip Architecture
  - Sheibanyrad Abbas, Miro Panades Ivan, Greiner Alain
  - 10th Design Automation and Test in Europe Conference (DATE'2007), 2007, Nice, France, April 2007, pp. 1090-1095

- Hybrid-Timing FIFOs to use on Networks-on-Chip in GALS Architectures
  - Sheibanyrad Abbas, Greiner Alain
  - International Conference on Embedded Systems and Applications (ESA'2007), Las Vegas, Nevada, USA, June 2007

- A Low Cost Network-on-Chip with Guaranteed Service Well Suited to the GALS Approach
  - Miro Panades Ivan, Greiner Alain, Sheibanyrad Abbas
  - 1st International Conference on Nano-Networks (NanoNet'2006), Lausanne, Switzerland, September 2006, pp. 1-5

- Two Efficient Synchronous ⇔ Asynchronous Converters Well-Suited for Network on Chip in GALS Architectures
  - Sheibanyrad Abbas, Greiner Alain
  - Power and Timing Modeling Optimization and Simulation (PATMOS'2006), Montpellier, France, September 2006, pp. 192-202

- Micro-réseau sur puce compatible avec l'approche GALS
  - Miro Panades Ivan, Greiner Alain, Sheibanyrad Abbas
  - Journées Nationales du Réseau Doctoral en Microélectronique (JNRDM'2006), Rennes, France, Mai 2006

References

[1] J. Oberg, "Clocking strategies for networks-on-chip," in *Networks on chip*: Kluwer Academic Publishers, 2003, pp. 153-172.

[2] E. G. Friedman, "Clock Distribution Networks in Synchronous Digital Integrated Circuits," in *Proceedings of the IEEE*, 2001, pp. 665-692.

[3] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems." vol. Ph.D. Thesis: Stanford Univ., CA. Dept. of Computer Science, 1984.

[4] J. Sparsø, "Future networks-on-chip; will they be synchronous or asynchronous? ," in *Swedish System on Chip Conference (Invited talk)*, 2004.

[5] L. R. Dennison, W. J. Dally, and D. Xanthopoulos, "Low-latency plesiochronous data retiming," in *Proceedings of the 16th Conference on Advanced Research in VLSI (ARVLSI'95)*: IEEE Computer Society, 1995.

[6] W. K. Stewart and S. A. Ward, "A Solution to a Special Case of the Synchronization Problem," *IEEE Trans. Comput.,* vol. 37, pp. 123-125, 1988.

[7] A. Chakraborty and M. R. Greenstreet, "Efficient Self-Timed Interfaces for Crossing Clock Domains," in *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*: IEEE Computer Society, 2003.

[8] E. Nilsson and J. Oberg, "Reducing power and latency in 2-D mesh NoCs using globally pseudochronous locally synchronous clocking," in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* Stockholm, Sweden: ACM Press, 2004.

[9] Y. Semiat and R. Ginosar, "Timing Measurements of Synchronization Circuits," in *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*: IEEE Computer Society, 2003.

[10] R. Ginosar and R. Kol, "Adaptive Synchronization," in *IEEE International Conference on Computer Design*, 1998.

[11] I. Miro Panades and A. Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures," in *Proceedings of the First International Symposium on Networks-on-Chip*: IEEE Computer Society, 2007.

[12] J. Mekie, S. Chakraborty, D. K. Sharma, G. Venkataramani, and P. S. Thiagarajan, "Interface Design for Rationally Clocked GALS Systems," in *Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems*: IEEE Computer Society, 2006.

[13] U. Frank, T. Kapshitz, and R. Ginosar, "A predictive synchronizer for periodic clock domains," *Form. Methods Syst. Des.,* vol. 28, pp. 171-186, 2006.

[14] L. F. G. Sarmenta, G. A. Pratt, and S. A. Ward, "Rational clocking [digital systems design]," in *Proceedings of the 1995 International Conference on Computer Design: VLSI in Computers and Processors*: IEEE Computer Society, 1995.

[15] Ateris, "A comparison of Network-on-Chip and Buses," White Paper.

[16] T. Chelcea and S. M. Nowick, "Robust interfaces for mixed-timing systems with application to latency-insensitive protocols," in *Proceedings of the 38th conference on Design automation* Las Vegas, Nevada, United States: ACM Press, 2001.

[17] J. Jex, C. Dike, and K. Self, "Fully asynchronous interface with programmable metastability settling time synchronizer," US Patent 5 598 113, 1997.

[18] J. N. Seizovic, "Pipeline Synchronization," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1994, pp. 87--96.

[19] S. Moore, G. Taylor, R. Mullins, and P. Robinson, "Point to point GALS interconnect," *in Proc. of Int. Symp. on Asynchronous Circuits and Systems,* pp. 769-775, 2002.

[20] D. S. Bormann, "Asynchronous Wrapper for Heterogeneous Systems," in *Proceedings of the 1997 International Conference on Computer Design (ICCD '97)*: IEEE Computer Society, 1997.

[21] A. E. Sjogren and C. J. Myers, "Interfacing Synchronous and Asynchronous Modules Within a High-Speed Pipeline," in *Proceedings of the 17th Conference on Advanced Research in VLSI (ARVLSI '97)*: IEEE Computer Society, 1997.

[22] http://iangclark.net/metastability.html.

[23] K. Y. Yun and R. P. Donohue, "Pausible Clocking: A First Step Toward Heterogeneous Systems," in *Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors*: IEEE Computer Society, 1996.

[24] R. R. Dobkin, R. Ginosar, and C. P. Sotiriou, "Data Synchronization Issues in GALS SoCs," in *IEEE International Symposium on Asynchronous Circuits and Systems*, 2004, pp. 170-180.

[25] E. G. Wormald, "A Note on Synchronizer or Interlock Maloperation," *IEEE Trans. Comp.,* pp. 317 - 318, 1977.

[26] P. Varma, B. S. Panwar, and K. N. Ramganesh, "Cutting Metastability Using Aperture Transformation," *IEEE Trans. Comput.,* vol. 53, pp. 1200-1204, 2004.

[27] S. Ghahremani, "Metastable protected latch ": US Patent 6 072 346, 2000.

[28] T. J. Chaney, "Comments on `A Note on Synchronizer and Interlock Maloperation'," *IEEE Trans. Computers,* vol. 28, pp. 802-804, 1979.

[29] R. Ginosar, "Fourteen Ways to Fool Your Synchronizer," in *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*: IEEE Computer Society, 2003.

[30] http://www.fpga-faq.com/FAQ_Pages/0017_Tell_me_about_metastables.htm.

[31] C. Dike and E. Burton, "Miller and noise effects in a synchronizing flip-flop," *IEEE Journal of Solid-State Circuits.,* vol. 34, pp. 849-855, 1999.

[32] D. J. Kinniment, A. Bystrov, and A. V. Yakovlev, "Synchronization circuit performance," *IEEE Journal of Solid-State Circuits,* vol. 37, pp. 202-209, 2002.

[33] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proceedings of the conference on Design, automation and test in Europe* Paris, France: ACM Press, 2000.

[34] J. W. Dally and B. Towles, "Route packets, not wires: on-chip inteconnection networks," in *Proceedings of the 38th conference on Design automation* Las Vegas, Nevada, United States: ACM Press, 2001.

[35] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Des. Test,* vol. 22, pp. 414-421, 2005.

[36] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip," in *Proceedings of the conference on Design, automation and test in Europe - Volume 2*: IEEE Computer Society, 2004.

[37] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "xpipes: a Latency Insensitive Parameterized Network-on-chip Architecture For Multi-Processor SoCs," in *Proceedings of the 21st International Conference on Computer Design*: IEEE Computer Society, 2003.

[38] S. Kumar, A. Jantsch, J. P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A Network on Chip Architecture and Design Methodology," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*: IEEE Computer Society, 2002.

[39] F. Karim, A. Nguyen, and S. Dey, "An Interconnect Architecture for Networking Systems on Chips," *IEEE Micro,* vol. 22, pp. 36-45, 2002.

[40] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: a novel on-chip communication network," *Proceedings of International Symposium on System-on-Chip,* 2004.

[41] D. Wiklund and D. Liu, "SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*: IEEE Computer Society, 2003.

[42] F. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integr. VLSI J.,* vol. 38, pp. 69-93, 2004.

[43] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A Scalable, Single-Chip Communications Architecture," in *Proceedings of the 2000 International Conference on Parallel Architectures and Compilation Techniques*: IEEE Computer Society, 2000.

[44] C. Albenes Zeferino and A. Amadeu Susin, "SoCIN: A Parametric and Scalable Network-on-Chip," in *Proceedings of the 16th symposium on Integrated circuits and systems design*: IEEE Computer Society, 2003.

[45] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Design of a switch for network on chip applications," *Proceedings of the International Symposium on Circuits and Systems,* vol. 5, pp. 217-220, 2003.

[46] S.-J. Lee, S.-J. Song, K. Lee, J.-H. Woo, S.-E. Kim, B.-G. Nam, and H.-J. Yoo, "An 800MHz star-connected on-chip network for application to systems on a chip," *Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC,* vol. 1, pp. 468- 469, 2003.

[47] I. Saastamoinen, D. Siguenza-Tortosa, and J. Nurmi, "Interconnect IP Node for Future System-on-Chip Designs," in *Proceedings of the The First IEEE International Workshop on Electronic Design, Test and Applications (DELTA '02)*: IEEE Computer Society, 2002.

[48] J. Bainbridge and S. Furber, "Chain: A Delay-Insensitive Chip Area Interconnect," *IEEE Micro,* vol. 22, pp. 16-23, 2002.

[49] F. Feliciian and S. B. Furber, "An asynchronous on-chip network router with quality-of-service (QoS) support," in *Proceedings of International IEEE SOC Conference*, 2004, pp. 274- 277.

[50] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *J. Syst. Archit.,* vol. 50, pp. 105-128, 2004.

[51] A. Lines, "Asynchronous Interconnect for Synchronous SoC Design," *IEEE Micro,* vol. 24, pp. 32-41, 2004.

[52] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An Asynchronous NOC Architecture Providing Low Latency Service and Its Multi-Level Design Framework," in *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*: IEEE Computer Society, 2005.

[53] T. Bjerregaard and J. Sparso, "A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip," in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 2*: IEEE Computer Society, 2005.

[54] T. Bjerregaard, "The MANGO Clockless Network-on-Chip: Concepts and Implementation." vol. PhD thesis: Technical University of Denmark, 2005.

[55] G. M. Chiu, "The Odd-Even Turn Model for Adaptive Routing," *IEEE Trans. Parallel Distrib. Syst.,* vol. 11, pp. 729-738, 2000.

[56] D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, Y. Durand, J. Durupt, D. Varreau, P. Vivet, P. Penard, A. Bouttier, and F. Berens, " A Telecom Baseband Circuit based on an Asynchronous Network-on-Chip," in *IEEE Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers*, 2007, pp. 258-601.

[57] E. Beigne and P. Vivet, "Design of On-chip and Off-chip Interfaces for a GALS NoC Architecture," in *Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems*: IEEE Computer Society, 2006.

[58] E. C. Clifford, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," in *Synopsys Users Group Conference*, 2002.

[59] A. Andriahantenaina and A. Greiner, "Micro-Network for SoC: Implementation of a 32-Port SPIN network," in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*: IEEE Computer Society, 2003.

[60] R. Ho, K. W. Mai, and M. A. Horowitz, "The Future of Wires," in *Proceedings of the IEEE* vol. 89, 2001, pp. 490-504.

[61] A. J. Martin, "The Design of a Self-timed Circuit for Distributed Mutual Exclusion. ," Technical Report. California Institute of Technology. 1983.

[62] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*: Kluwer Academic Publishers 2002.

[63] http://www-asim.lip6.fr/recherche/coriolis/.

[64] http://www-asim.lip6.fr/recherche/alliance/.

[65] A. Greiner and F. Pêcheux, "ALLIANCE: A complete Set of CAD Tools for teaching VLSI Design," proc. Third EuroChip Workshop, 1992.

[66] http://www.itrs.net/Links/2005ITRS/Home2005.htm.

[67] T. Tao Ye, G. De Micheli, and L. Benini, "Analysis of power consumption on switch fabrics in network routers," in *Proceedings of the 39th conference on Design automation* New Orleans, Louisiana, USA: ACM, 2002.

[68] L. Benini, P. Siegel, and G. De Micheli, "Saving Power by Synthesizing Gated Clocks for Sequential Circuits," *IEEE Des. Test,* vol. 11, pp. 32-41, 1994.

[69] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, H. L. Gabriel, D. McCaule, P. Morrow, W. N. Donald, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*: IEEE Computer Society, 2006.

[70] http://www.intel.com/research/platform/terascale/.

[71] J. Held, J. Bautista, and S. Koehl, "From a Few Cores to Many: A Tera-scale Computing Research Overview," Intel White Paper, 2006.

[72] Intel, "Intel's Tera-Scale Research Prepares for Tens, Hundreds of Cores," Technology@Intel Magazine, 2006.

[73] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," in *IEEE International Solid-State Circuits Conference* San Francisco, CA, USA: Digest of Technical Papers, 2007, pp. 5-7.

[74] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *Computer,* vol. 35, pp. 70-78, 2002.

[75] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. Albenes Zeferino, "SPIN: A Scalable, Packet Switched, On-Chip Micro-Network," in *Proceedings of the conference on Design, Automation and Test in Europe: Designers' Forum - Volume 2*: IEEE Computer Society, 2003.

[76] The Open Microprocessor systems Initiative, "PI-Bus Draft Standard Specification (OMI 324)", 1994.

[77] G. De Micheli and L. Benini, *Networks on Chips: Technology and Tools*: Morgan Kaufmann, 2006.

[78] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. Very Large Scale Integr. Syst.,* vol. 8, pp. 299-316, 2000.

[79] Arteris, "Network-on-Chip: The future of SoC power management," in *CDNLive! EMEA*, 2006.

[80] D. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach* The Morgan Kaufmann, 1999.

[81] A. Jantsch, "Communication performance in Network-on-Chips," in *Presentation at the Swedish INTELECT Summer School on Multiprocessor Systems on Chip*, 2003.

[82] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.,* vol. 34, pp. 892-901, 1985.

[83] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.,* vol. 36, pp. 547-553, 1987.

[84] http://en.wikipedia.org/wiki/Deadlock.

[85] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *SIGARCH Comput. Archit. News,* vol. 20, pp. 278-287, 1992.

[86] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of Network-on-chip," *ACM Comput. Surv.,* vol. 38, p. 1, 2006.

[87] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "input versus output queuing on a space-division packet switch," *IEEE Trans. Commun.,* vol. 35, pp. 1347-1356, 1987.

[88] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer,* vol. 26, pp. 62-76, 1993.

[89] W. J. Dally, "Virtual-Channel Flow Control," *IEEE Trans. Parallel Distrib. Syst.,* vol. 3, pp. 194-205, 1992.

[90] S. G. Pestana, E. Rijpkema, A. Radulescu, K. Goossens, and O. P. Gangwal, "Cost-Performance Trade-Offs in Networks on Chip: A Simulation-Based Approach," in *Proceedings of the conference on Design, automation and test in Europe - Volume 2*: IEEE Computer Society, 2004.

[91] P. Wielage and K. Goossens, "Networks on Silicon: Blessing or Nightmare?," in *Proceedings of the Euromicro Symposium on Digital Systems Design*: IEEE Computer Society, 2002.

[92] J. A. Kahl, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the Cell multiprocessor," *IBM Journal of Research and Development,* 2005.

[93] J. Dielissen, A. Radulescu, K. Goossens, and E. Rijpkema, "Concepts and Implementation of the Philips Network-on-Chip," in *IP-Based SOC Design*, 2003.

[94] K. Goossens, "Networks on Chips for Consumer Electronics," in *the Proceedings of the International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems* 2005.

[95] http://www.st.com/stonline/press/news/year2005/t1741t.htm.

[96] http://intranet.cs.man.ac.uk/apt/async/.

[97] S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, and N. C. Paver, "AMULET2e: An Asynchronous Embedded Controller," in *Proceedings of the 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems*: IEEE Computer Society, 1997.

[98] C. K. Paul, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*: Springer-Verlag, 1996.

[99] F. Gürkaynak, S. Oetiker, H. Kaeslin, N. Felber, and W. Fichtner, "Improving DPA security by using globally-asynchronous locally-synchronous systems," in *Proceedings of Solid-State Circuits Conference, 2005. ESSCIRC 2005. of the 31st European*, 2005, pp. 407-410.

[100] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Proceedings of the sixth MIT conference on Advanced research in VLSI* Boston, Massachusetts, United States: MIT Press, 1990.

[101] T. A. Chu, "On the models for designing VLSI asynchronous digital systems," *Integr. VLSI J.,* vol. 4, pp. 99-113, 1986.

[102] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*: Prentice Hall PTR, 1981.