Daniele Ludovici

Technology Aware Network-on-Chip Connectivity and Synchronization Design

Technology Aware Network-on-Chip Connectivity and Synchronization Design

from the NoC concept to actual NoC technology

PROEFSCHRIFT

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

donderdag 30 Juni 2011 om 10:00 uur

door

Daniele LUDOVICI

Master of Science in Computer Systems Engineering
University of Pisa, Italië
geboren te Alatri, Italië

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr.ir H.J. Sips

Copromotor:
Dr.ir. G.N. Gaydadjiev

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr.ir. H.J. Sips	Technische Universiteit Delft, promotor
Dr.ir. G.N. Gaydadjiev	Technische Universiteit Delft, copromotor
Prof. dr.ir. A.J. van der Veen	Technische Universiteit Delft
Prof. dr.ir. G.J.M. Smit	Universiteit Twente
Prof. dr. D.N. Pnevmatikatos	University of Crete & FORTH, Griekland
Prof. dr. L. Carro	Universidade Federal do Rio Grande do Sul, Brazilië
Dr. D. Bertozzi	University of Ferrara, Italië
Prof. dr.ir. C.I.M. Beenakker	Technische Universiteit Delft, reservelid

ISBN 978-90-72298-18-8

Keywords: Network-on-Chip, Globally Asynchronous Locally Synchronous (GALS), Topology Exploration, Synchronizers

Printed in The Netherlands

To Serena, and our dreams together

Technology Aware NoC
Connectivity and Synchronization Design

Daniele Ludovici

Abstract

NoCs have been considered as the new design paradigm for large MP-SoC systems in the past ten years. In the beginning NoCs were radically different compared to the current state of the art mainly due to the unexpected unique challenges that system designers had to solve with the evolving CMOS technology. In fact, various hidden physical level issues may potentially degrade system performance, exceed the available power budgets or even endanger the overall design feasibility. The *connectivity* among different multi-core elements is such an issue that has to be addressed during the design of the overall communication infrastructure. Two different classes of implications related to the aggressive CMOS technology scaling, resulting in growing process variations, reduced power budgets per unit area and worsening signal integrity on chip, have to be considered. On one hand, a good *topology* is required to provide adequate sub-system connectivity while also satisfying the bandwidth and performance requirements. On the other hand, increasing *synchronization issues* make the system design difficult and in some cases even impossible to be realized under a rigid synchronization model. For instance, the topology strongly depends on the physical effects as consequence of the wire delay reverse scaling while the synchronization issues are tightly related to process variation effects. Therefore, in the current and the future CMOS technology nodes, ad-hoc counter measures must be adopted to cope with the above problems. In this thesis we propose a system-level analysis framework and design methodology both considering real layout effects. Our analysis is not only limited to classical layout effects such as the non-regularity of a rectangular tile; the real wire delays of inter-switch links; the number of pipeline stages required to provide the requested link performance; the maximum tolerated skew of a certain synchronization scheme; and more. We also consider the implications of the above physical phenomena while re-designing our architectural blocks. The ultimate result is a framework which is truly technology aware, ready to meet the challenges of the future CMOS technology landscape.

Acknowledgements

I have dreamed of this moment for a long time during my PhD. Now here I am, trying to sort out all the memories related to the last five years. Not easy, for sure, but I will give it a try.

All this started because somebody gave me the opportunity to join the CE-group upon just a couple of exchanges of email. Stamatis was certainly the enabler of my PhD journey and although I did not have much time to enjoy his company, I will be always indebted to him. After having such an opportunity, my PhD trip went through some initial turbulences and I owe deep and sincere gratitude to my co-promotor Georgi Gaydadjiev and to my advisor Davide Bertozzi for helping to work things out. Georgi had me under his wing for the time I spent in Delft; he trusted my choice to move back to Italy for an initial internship with Davide. That internship actually turned out to be a crucial point of my PhD studies and, in fact, it started what ended up being my final research topic. I am indebted to him also for the effort he put into reviewing this manuscript. Davide is the key person I met in my short but exciting research career. He taught and shaped my way of approaching to challenging research problems. Needless to say, he has been also a good friend and a person I could always count on. I hope this thesis is a good way to thank him for his hard and enthusiastic work in establishing a great research group in Ferrara, and for the possibility I had to work on ambitious topics.

Work apart, my PhD journey has been lucky twice because it was split between two countries. In The Netherlands, I had unforgettable moments with my closest friends: Carlo, Christos, Dimitris, Lotfi, Yiannis and Sebastian. All of you guys left me something during a coffee in Kobus, a dinner at Carlo and Niki's or a discussion while biking under the rain. Your friendship is among the best things I could desire out of such a PhD experience. A special thank you goes to Carlo and Dimitris for helping with many thesis practicalities.

I will never forget also the initial Delft period in the company of Giacomo, Gennaro, Mattia, Francesco, Mauro, Maristella, Mario, Corrado, Christian, Alessandro, Gianni, Sergio, Christian, Alessandro, Maria, and the dozens of

other Italians I met during these years in Delft. It has been like home thanks to all of you guys.

A special thank you is due to all the people that helped me with administrative and technical problems during my years at the CE-group: Lidwina Tromp, Monique Tromp, Bert Meijs, Erik de Vries and Eef Hartman. Many thanks also to Arjan van Genderen who contributed to the Dutch translation of this thesis abstract.

Ferrara has been the other half of my PhD life, many friends, exciting moments, hard work and sleepless nights but eventually it worked out. This could happen only because I had the luck to meet amazing friends: Simone and Alessandro in the beginning; we shared hard work and great fun during our *spritz* evenings. After some time, Alberto, Hervé and Luca joined Davide's group. Rapidly, we created a larger and harmonious environment to work together in, but above all we established something more than just work. Sergio and Valeria have been also a large part of my life in Ferrara. I also want to mention and I hope not to forget anybody: Cristian, Daniele, Francesco, Enrico, Raffaele, Andrea, Giorgio, Valerio and Gery. Thanks to all of you my friends. I also owe Federico, Igor, Antonio and Mohammad many thanks for their help with the problems I had during the development of my PhD work. Last, I could not forget the Spanish team: Paco, Samuel and Crispin. I had many enjoyable moments during your internships in Ferrara and I hope you had at least the half of what I got from you.

Finally, needless to say, I owe my parents and my whole family more than I can write on a piece of paper. You are the engine of my dreams and all I am is just because of you.

My last thought is for Serena, I could not desire a more beautiful person for sharing the amazing journey of life, together.

Daniele Ferrara, Italy – Delft, The Netherlands, June 2011

Table of contents

List of Tables

List of Figures

List of Acronyms and Symbols

ASIC	Application-Specific Integrated Circuit
CMP	Chip MultiProcessor
CTS	Clock Tree Synthesis
DVFS	Dynamic Voltage and Frequency Scaling
FF	Flip-Flop
FHD	Full High Definition
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GALS	Globally Asynchronous Locally Synchronous
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HDL	Hardware Description Language
ILM	Interface Logical Model
IP	Intellectual Property
ITRS	International Technology Roadmap for Semiconductors
LEF	Library Exchange Format
LUT	Look-Up Table
MIN	Multi-stage Interconnection Network
MPSoC	Multi Processor System on Chip
MUX	Multiplexer
NI	Network Interface
NoC	Network-on-Chip
OCP	Open Core Protocol
P&R	Place and Route
PDA	Personal Data Assistant
RR	Round Robin
RTL	Register Transfer Level
SOCE	Cadence SoC Encounter
SR	Search&Repair
SoC	System on Chip
TLM	Transaction-Level Modeling
TTM	Time-to-Market
VLSI	Very Large Scale Integration

1 Introduction

THE embedded system market is rapidly growing and features a rich variety of devices that are able to perform a wide multitude of diverse tasks. Nowadays, appliances such as mobile phones, personal data assistants (PDA) and ebook readers became mainstream in our everyday life. These mobile devices are ubiquitous, can be utilized everywhere and their applicability range span from pure computational tasks, through entertainment up to social network connectivity. The tremendous complexity reached by such devices represents a major challenge faced by engineers that have to design systems under a constant and relentless time-to-market (TTM) pressure. In order to shorten such TTM, the design of such devices is traditionally performed by integrating existing components in a plug–and–play fashion into a System-on-Chip (SoC) [1]. Therefore, a major challenge consists of interconnecting many different components with each other in an efficient way. According to ITRS roadmaps [89], thousands of cores will be integrated in a single chip during the next few years. Such scenario opens up many questions regarding scalability issues as all the cores in the single chip will have to be interconnected in a power efficient and scalable way.

Classically, intellectual properties (IPs) (e.g., memory controller, CPUs, GPUs, etc.) designed by different vendors are interconnected by dedicated buses. AMBA, AXI, AHB [46,47] represent well-established industrial examples of such interconnection architectures. Unfortunately, they do suffer from well known scalability problems due to arbitration penalties. This is one of the driver dictating the adoption of a more scalable interconnection scheme: Networks-on-Chip (NoCs). NoC architectures represent a viable, scalable packet-switched micro-network interconnect scheme alternative to classical bus architectures [92]. They are generally believed to be the long term solution to the communication scalability issue.

However, the design of a NoC for a multi-core system comes with its own set of challenges. The *connectivity* problem is certainly the key concern that system designers have to cope with in the early design stages. Yet, this requires thorough knowledge of the underlying technology platform for the sake of realistic (and first-time-right) network architecture planning. For this reason, the connectivity issue is addressed in this thesis and the challenge of bridging the needed abstraction in the early design steps with the intricacy of nanoscale physics is addressed. Network connectivity implies to address two inter-related design concerns. On one hand, a proper *connectivity pattern* must be selected in order to provide the adequate interconnection among the communication pairs of the system meeting at the same time the required bandwidth and performance of the design budget. On the other hand, due to increasing *synchronization issues*, designing a system under a tight synchronization assumption is already difficult and will be even impossible in the near feature because of increasing process variation, limited power budget and signal integrity. Therefore, ad-hoc counter measures must be adopted to relax the synchronization assumption within the system.

Both aforementioned problems are very sensitive and related to the physical design implications of the aggressive scaling of CMOS silicon technologies to the nanoscale era. For instance, the efficiency and sometimes even the feasibility of specific connectivity patterns suffer from the reverse scaling of interconnect delays, thus potentially countering the better abstract properties of the connectivity pattern itself (e.g., bisection bandwidth). Similarly, the successful design of synchronization interfaces needs to safeguard them against wire delay variability and varying layout constraints and operating (e.g., speed ratio, clock phase offset) conditions.

Clearly, in the landscape of nanoscale technologies connectivity design issues cannot be addressed without technology awareness although they come very early in the design process. The major contribution of this thesis consists of a system-level analysis framework which is augmented with technology awareness and therefore aims at advancing NoC design practice and architectures. The thesis fosters technology awareness in two ways:

- constraints posed by the technology platforms with system-level implications are characterized, abstracted through proper modeling frameworks and exposed to the system level design and exploration languages for the sake of technology aware system performance evaluation. The non-regularity of a rectangular tile, the real wire delay of an inter-switch link, the number of link pipeline stages required to provide a specific

link performance, the maximum tolerated skew between mesochronous domains are a few examples of such constraints that this thesis exposes up in the design hierarchy.

- Silicon aware decision making is enforced through the circuit and architecture layers of the NoC design process. Not only physical design effects are taken into account for the sake of architecture/circuit evaluation, but the architectures/circuits themselves are shaped by the need to be effectively implemented in the technology platform.

The ultimate result is a system-level analysis framework where the network-on-chip (providing the global connectivity infrastructure) is designed with technology-awareness from the ground up and those global parameters that are directly determined by the technology platform are accounted for in the system level evaluation. The thesis achieves two specific contributions. On one hand, it deals with the interface design of network building blocks in such a way that a true component-oriented and technology-aware design style can be pursued. On the other hand, it captures the system-wide implications that link synthesis techniques and switch connectivity patterns have on global system performance, thus materializing the paradigm of interconnect-centric design. Finally, the thesis combines these two aspects together thus coming up with an insightful and accurate framework for NoC evaluation in the context of the system-level analysis.

In the next section, we will start by looking at the problem of *connectivity design* (Section 1.1.1) whereas in Section 1.1.2 the *synchronization design* issue will be presented and illustrated. The key approach proposed in this thesis is described in Section 1.2 and finally, Section 1.3 provides an overview of the remaining chapters of the thesis.

1.1 Problem Formulation

When connecting network switching elements and attached IP cores together, two fundamental and inter-related problems immediately arise: determining the connectivity pattern and absorbing the speed differences between the networked entities. Such problems are hereafter detailed, with emphasis on the inability of current NoC literature to effectively tackle them.

1.1.1 Connectivity design with layout-awareness

Many concepts of the on-chip networks are directly borrowed from the off-chip domain, in fact, Networks-on-Chip closely resemble the interconnect architecture of high-performance parallel computing systems and thus the interconnection topologies used in the early NoC prototypes can be traced back to the field of parallel computing. In particular, NoC architectures aiming at low latency communication, performance scalability and flexible routing selected fat-trees as their reference topology. The switch for the butterfly fat-tree network of [8] or the SPIN micronetwork [13] are prominent examples thereof. However, other topologies have found wider application in common NoC design practice so far, namely 2D meshes and even folded tori [139, 140]. Unfortunately, technology scaling to the nanoscale era brings physical design issues to the forefront, such as the reverse scaling of the interconnects. In this context, 2D mesh and torus topologies exhibit a grid-based regular structure which is intuitively considered to be matched to the 2D chip layout. In contrast, the higher wiring irregularity and the larger switch radix of most fat-tree configurations raise some skepticism about their practical feasibility in real SoCs. Moreover, instead of aiming strictly for speed, designers increasingly need to consider energy consumption constraints. Fat-trees are in general, expected to pay for the increased connectivity they provide with a significant area and power costs. In spite of these concerns, constant attention has been devoted to tree-based topologies in the NoC community, proving their superior performance with respect to 2D meshes under different kinds of synthetic traffic patterns [141,142]. However, these analysis frameworks often rely on abstract network simulators which cannot model the behavior of any real architecture and sometimes make unrealistic assumptions, such as packet drop or TCP-compliant network transport protocols for on-chip communication. Furthermore, most of the works really miss an in-depth physical analysis of layout feasibility and efficiency. Even when area synthesis results are provided, the impact of wiring congestion and interconnect delay on network performance is only assessed by means of analytical models. Last, the effectiveness of advanced design techniques such as clock or power gating or link pipelining is usually ignored.

This thesis aims at overcoming the main limitations of previous network topologies evaluations for NoC, by primarily investigating the layout feasibility and the implications of physical mapping efficiency on system-level performance figures. This bottom-up approach to topology evaluation and selection reflects the design paradigm shift pushed by nanoscale technology: silicon-aware decisions are required at each level of the design hierarchy.

1.1.2 The synchronization design issue

Although never quantified so far, a trade-off clearly exists between the abstract theoretical properties of a topology and its silicon mapping efficiency. In extreme cases, the maximum operating speed of the network architecture should not constrain the speed of the networked IP cores. This calls for proper decoupling at the network boundary by means of synchronization interfaces. This is a key requirement also for power management strategies in the embedded computing domain, requiring each core to run at an independent and runtime variable voltage and speed. Nowadays, both application requirements and technology effects call for a disruptive evolution of the synchronization architecture. In fact, distributing a global clock throughout the entire chip with tighly controlled skew is becoming increasingly power inefficient and even infeasible. Indeed, shifting the focus to a pure silicon technology viewpoint, there are some other very important challenges to be faced by both industry and academic research. In fact, as technology advances into aggressive nanometer-level scaling, several design challenges emerge from technology constraints which require a continuous evolution of the interconnection implementation strategy adopted at the circuit and architectural levels. Synchronization of current and future chips with a single clock source and negligible skew is extremely difficult if not close to be impossible [1]. Indeed, synchronization is today definitely among the most critical challenges in the design of a global on-chip communication infrastructure, as emerging technology variability, signal integrity, power dissipation limits are contributing a severe break-down of the global synchronicitiy assumption when logical structures spans more than a couple of *mm* on the die [62]. NoCs typically span the entire chip area and there is now little doubt on the fact that a high-performance and cost-effective NoC in 45nm and beyond can only be designed under relaxed synchronization assumptions [118]. This is raising the need for easily extensible clock trees and for self-assembly clock tree synthesis strategies as that utilized by Intel in the Polaris chip [153]. A solution would be to design such systems using a fully asynchronous global intra-chip communication. Such choice would eliminate the clock distribution concern and would make designs more modular since timing assumptions are explicitly handled in the hand-shaking protocols. Unfortunately, current design tools and IP libraries heavily rely on the synchronous paradigm instead, thus making intermediate solutions more attractive and affordable in the short run. As previously anticipated, synchronizer-based globally asynchronous locally synchronous (GALS) systems represent an appealing solution in the mid term, and is therefore the focus of this thesis. In such systems, the design can be partitioned in different frequency islands and

the interconnection infrastructure can be envisioned as mesochronous domain (isolated by dual-clock FIFOs at the boundary) where a single clock spans the whole communication infrastructure area relying on a loose synchronization assumption. Such mesochronous assumption allows to tolerate an arbitrary amount of space dependent time-invariant phase offset (i.e., skew) among the leaves of the clock signal hence resulting in a lower power clock tree synthesis. Unfortunately, the high cost of traditional synchronizer implementations in terms of area, power and latency is typically the main reason preventing their adoption as intermediate solution. Moreover, this is only one of the possible GALS implementation variants within a large design space where it is difficult to select the best solution for the underlying design. Last, there is a general skepticism of industrial designers to relax the synchronization assumption in their chips due to the usage of unconventional tool capabilities, to the poor predictability of resulting designs and to the threat of process variations.

Aware of these challenging problems, the second goal of this thesis is to develop all the required support for the building process of such GALS systems. In particular, our contribution and emphasis is on the NoC infrastructure that is augmented with all the necessary library components in a sort of *galsification* process. Such GALS blocks are designed to considerably reduce area, power and latency issues. Furthermore, we perform a crossbenchmarking between implementation variants, resulting in actual guidelines for designers that want to migrate from synchronous to GALS solutions. Finally, the development effort of the synchronization library is continuously validated with post-place&route experiments and in variability-dominated scenarios, thus paving the way for predictable and robust synchronizer-based GALS NoCs design.

1.2 Approach

Both the *Connectivity* and the *Synchronization* problems stated above suggest that NoCs are more than a typical interconnection network mapped on a 2D silicon surface. Challenges exposed by the technology scaling dramatically changed what was believed to be a trivial problem. This thesis contributes to the evolution of a pure theoretical NoC concept into a solid, mature and well-established NoC technology.

Specifically, in order to tackle the *synchronization* problem, we design novel globally asynchronous locally synchronous (GALS) interfaces able to meet different layout constraints. Furthermore, we show that it is possible to migrate from the synchronous to the GALS paradigm with a negligible area and

power cost without impacting performance. Moreover, we implement a complete design flow for building GALS NoCs and we utilize such flow to instantiate and cross benchmark two GALS systems: the first implementing a fully synchronous and the second implementing a mesochronous NoC. Both systems leverage dual-clock FIFO interfaces to provide frequency decoupling between the NoC and the computational units. We carry out a thorough comparison of these two GALS systems from the clock tree power, area/wiring, power consumption, skew tolerance and variability robustness viewpoint.

For tackling the *connectivity pattern* issue, we analyze various small scale topologies and assess their quality metrics accounting for physical level effects thus overcoming misleading high-level assumptions. Our analysis considers the impact of NoC link inference techniques such as repeater insertion and link pipelining. We extend our work by proposing a methodology for evaluating large scale systems that is layout aware, prunes time and memory requirements while retaining performance and area accuracy.

In both cases, our work is validated at post-layout level and the gained physical insights are utilized to further optimize the system architecture thus matching the technology challenges. The ultimate result of such combined contributions is a set of most promising connectivity patterns which can work in a mature GALS landscape while taking advantage of the speed decoupling offered by the GALS paradigm. We benchmark such topologies showing that there are several non-intuitive design opportunities depending on the available area and power budget. Overall, the thesis results in a global framework for NoC connectivity evaluation with technology awareness. Physical effects are not considered as an after-thought, but accounted for from the ground up in the circuit- and architecture-level design stages.

1.3 Organization

The contributions of this thesis are organized in 10 chapters. Before presenting the contributions, Chapter 2 first provides the necessary background of the work in this thesis. It surveys topology evaluation frameworks as well as the design of globally-asynchronous locally synchronous interfaces for the building of GALS systems. Finally, it summarizes the shortcoming of the presented work to be addressed in subsequent chapters.
Chapter 3 introduces the synchronization design issue. In a first step, the motivation for adopting synchronization mechanisms in the NoC environment will be discussed. Next, the target GALS platform of this thesis along with the ar-

chitecture of the basic switch block required to build it will be presented. Last, the focus will be on the baseline mesochronous synchronizer and all its improvements that led to a new fully integrated and flexible switch architecture.
Chapter 4 presents a design flow to build GALS systems from the system specification, through synthesis and CTS thus reaching the layout level by performing place&route. In particular, a complete NoC design flow will be illustrated in its front- and back-end part. For both parts, our contribution to make the design flow suitable for building GALS systems will be discussed.
Chapter 5 performs a cross-benchmarking between two different GALS systems. The former leverages a fully synchronous NoC while the latter implements a mesochronous NoC. While the former chapters focused on a switch-level analysis of such GALS system, in the following, a network-level perspective will be taken. Furthermore, both systems will be compared from many viewpoints such as, clock tree power analysis, area and wiring overhead and above all from a variability robustness viewpoint.
Chapter 6 explores the performance and physical feasibility of 16-tile NoCs within several topology configurations. It is the first chapter where our system-to layout-level approach for assessing NoC topologies is presented. Our analysis framework encompasses different abstraction levels as physical key parameters from synthesis and place&route process are calculated and then exposed to our system level simulation infrastructure thus materializing in a layout-aware system-level performance analysis.
Chapter 7 assesses several NoC link inference techniques taking the system-level perspective. In fact, performance speed-ups and power overheads are not evaluated for the links in isolation but for the topology as a whole, thus showing their sensitivity to the link inference strategy.
Chapter 8 extends the work presented in the previous chapters by proposing a methodology for assessing topology implementation cost when scaling to larger systems. Furthermore, such methodology captures the impact of link pipelining on topology area and performance assessing whether and to which extent theoretical benefits are preserved. Overall, such methodology enables the analysis of large scale systems pruning time and memory requirements.
Chapter 9 performs a final comparison leveraging previous efforts: we benchmark several GALS topologies with IP core-network speed decoupling and we carry out a system-level exploration with layout awareness.
Finally, Chapter 10 provides concluding remarks on the work presented. The chapter summarizes the thesis, outlines its contributions and proposes future research directions.

2 Background

THIS chapter starts by surveying several works in the field of topology exploration for significant topology families. Furthermore, contributions concerning NoC link design techniques are also surveyed. Last, recent works in the domain of globally asynchronous locally synchronous (GALS) Networks-on-Chip are reported. The chapter highlights also the contribution of this thesis with respect to the work available in the open literature.

2.1 Topology Exploration

2.1.1 2D and Multi-Dimensional Meshes

Although widely used across a number of Network-on-chip designs [125,130], the 2D mesh NoC topology lacks of scalability and tends to concentrate traffic in the center nodes [129]. This has motivated works in the open literature that come up with optimized NoC topologies while keeping regularity properties as much as possible. A novel interconnect topology called spidergon was proposed in [131], where each core is connected to the clockwise, counterclockwise and diagonal node. A traditional wormhole-routed mesh augmented by a hierarchical ring interconnect for routing global traffic is illustrated in [132]. NOVA is a hybrid interconnect topology targeted at an FPGA, and is compared in [137] with star, torus and hypercube topologies. Gilabert et al. propose in [138] to use high-dimensional topologies, using different metal layers to soft long link delay and trading-off dimensions with the number of cores per router. The work in [129] proposes a concentrated mesh architecture with replicated subnetworks and express channels. [138] trades-off the number of dimensions with the number of cores per router, but lacks of physical insights. Topology exploration is an active research area due to the large scale of on-chip

networks and to the feasibility challenges posed by nanoscale technologies. An effort to compare mesh and torus topologies under different routing algorithms and traffic models with respect to their performance and power consumption is described in [144]. Theoretical uniform traffic based on the request/reply paradigm is used to assess ring, 2D-mesh, spidergon and unbuffered crossbar topologies in [145]. Ring, Octagon and 2D-mesh topologies are also the focus of [100], which emphasizes the communication overhead that has to be expected with wormhole switching, full duplex links and k-port non combining nodes. The work in [39] addresses message-dependent deadlock in the context of NoC topology synthesis. [147] claims that from an energy standpoint, high-dimensional tori should never be selected over hierarchical or express cubes. Different optimal topologies have also been indicated for different traffic patterns. The analysis carried out by [9] shows that the fat-tree topology is a strong candidate to fulfill the latency constraints of many applications. Several interesting topologies emerge by incorporating the third dimension in NoCs. Speed and power consumption of 2D and 3D NoCs are compared in [148].

As technology scales to the nanometer era, topology analysis and exploration needs to be performed with novel methodologies and tools that account for the effects of nanoscale physics, largely impacting final performance and even feasibility of many NoC topologies. A general guideline driving NoC design under severe technology constraints consists of silicon-aware decision-making at each hierarchical level [149]. This is likely to result in less design iterations and in faster timing closure. In this direction, new tools are emerging that guide designers towards a subset of most suitable candidates for on-chip network designs while considering the complex tradeoffs between applications, architectures and technologies [153, 154].

2.1.2 Fat-trees

Fat-trees were first proposed by [4]. In [5] authors extend the notion to trees with a varying number of switches in each level, and links between the levels. In general, fat-trees exhibit variable-sized switches and large switch radix close to the root, which complicates physical implementation. As a workaround, two structures have been proposed featuring constant switch size: k-ary n-trees [127] and butterfly fat-trees [37]. Low latency communication and performance scalability are the main reasons motivating the use of fat-trees also in early network-on-chip prototypes, like in the SPIN micronetwork [10, 13]. A 32-port SPIN network was laid out in less than 5 mm^2 area with a 130 nm process [32]. A butterfly fat-tree is used in [142, 150], with the motivation that

the number of switches converges to a constant independent of the number of levels. More recently, optimized tree-based architectures have been proposed to address the implementation overhead of traditional fat-trees. One approach is to combine the properties of grid-based topologies like mesh and torus with those of tree-based topologies. In this direction, [11] proposes a fat H-tree for on-chip realizations, i.e., a torus structure which is formed by only combining two H-trees. The extension of this solution to 3D NoCs is illustrated in [12]. The work in [120] proposes a deterministic routing algorithm for fat-trees which is able to obtain at least the same performance than adaptive routing. Taking advantage of some particular properties of this algorithm, an unidirectional MIN is proposed in [119], that significantly simplifies fat-tree switches. In that work, authors are concerned with possible wiring intricacies, but no physical analysis is provided.

Comparative studies between fat-trees and 2D meshes have often been the focus of past research. Superior performance of fat-trees has been demonstrated in an on-chip setting and for H.264 communication requirements by [141]. Converging indications come from [14] and [31]. However, the lack of synthesis results, the abstract simulation framework and the sometimes unrealistic architecture assumptions put hardness of experimental evidence in discussion. The work in [142] is perhaps the most accurate and complete comparative analysis of fat-trees performed so far. It proves the high energy cost of fat-trees and octagon, which is the price to pay for the higher throughput and lower latency on the average. Although the network switches are synthesized, place&route effects are captured only through analytical models and technology projections and not from actual layouts.

This thesis significantly extends the past work in the field of topology exploration by taking a layout-aware approach. First, the emphasis is not on capturing the sensitivity of the system performance to physical parameters, but rather on the assessment of such parameters for selected k-ary n-mesh as well as k-ary n-tree topologies when designed for maximum performance. Furthermore, a design space exploration is performed through a transaction-level simulation environment that is able to back-annotate key parameters (frequency, latency, area) from the results of physical synthesis. When extending the analysis to larger 64-tile networks, the unaffordable time and memory requirements for the synthesis of such systems makes a comprehensive exploration based on post-layout figures unfeasible. Therefore, in order to extend the exploration to larger 64-tile networks, we propose a novel modeling methodology based on selective synthesis runs that is able to capture the key post-layout parameters of a large scale topology such as, maximum frequency and switch cell area.

Moreover, such framework is able to capture the impact of link buffering and link pipelining from the timing and area cost viewpoint. Furthermore, by utilizing such physical parameters in our transaction-level simulator, it is possible to perform a layout-aware system-level analysis. This way, overall area and performance figures can be drawn. Last, the simulator is able to model GALS systems where the cores and the network are completely frequency decoupled since dual-clock FIFO interfaces have been implemented.

2.2 Link Design Techniques

When the performance of the NoC link needs to be improved, there are several techniques to be exploited. For instance, *stateless* and *stateful* repeater insertion allow to speedup signal transmission over interconnection wires [15, 28, 29]. Buffer insertion is considered a *stateless* techniques in the sense that, the signal is boosted by breaking the wires and thus reducing RC product because of its quadratic dependency on the link length. On the contrary, link pipelining is a technique that, at the cost of higher latency, enables to shorten an interconnection link by adding one or more *stateful* (i.e., flip-flop) elements along the wire. Power and energy implications of using such techniques have been investigated in depth, e.g., Heo and Asanovic in [30] investigate link optimization by varying repeater spacing, link pipelining and voltage scaling to further reduce transmission energy across the chip. In [16] authors explore the co-design of logic sizing and repeater insertion for improved delay, power and placement. Non-uniform repeater insertion is used to combine cascaded sizing and distributed wire buffering and delivers less power consumption. Xu et al. in [27] address the problem of interconnect pipelining from both power and bit error rate point of view and try to find the optimal solution for a given wire pipelining scheme. Using a graph representation of the interconnects, Youssef et al. [17] introduce a low power multi-pin routing strategy to mitigate the power consumed by the interconnect buffers. Macchiarulo et al. present a study on a modified wire pipelining scheme and they build a floorplanner based on adaptive and nonadaptive wire pipelining which optimizes data rate taking block delay into account [19]. Same authors in [18] illustrate a proposal that allows to use wire pipelining with no architectural changes while increasing performance. Zhou et al. [21] explore retiming to pipeline long wires in SoC design by applying retiming stages to a netlist of macroblocks. Roy et al. present an in-depth analysis of the dependencies of the reliability and power consumption of wire pipelining scheme on the number of inserted

flip-flops and the size of repeaters [20]. Yuchun et al. first consider both block and interconnect pipelining simultaneously by optimizing critical paths in the micro-architecture and inserting pipeline stages in the interconnect wires [26]. Cong et al. extend the regular distributed register micro-architecture to support interconnect pipelining in [22]. Zong et al. [25] propose a methodology to optimize power consuming elements (wires, buffers, clock distribution networks, etc.) of the interconnects during high-level synthesis. Implications have been theme of discussion also at system and architectural level in the NoC domain. In [24], Carloni et al. illustrate an architecture-level technique for latency insensitive design. Marculescu et al. estimate the impact of interconnection links on the overall topology performance in [23] thus pointing out the importance of link design techniques at architectural level.

In contrast to previous work, in this thesis we capture the sensitivity of quality metrics of NoC topologies with a regular connectivity pattern to the specific synthesis technique for their links. This way, the opportunity to invest more in area and/or power to speed up the links of a topology is explored. Some non-trivial implications are therefore demonstrated.

2.3 The GALS Design Style

The choice of the best topology candidate to interconnect a system as well as the link synthesis technique to adopt are challenging tasks that have been addressed by many previous research efforts as pointed out previously. On the same path, when the target system requires the interconnection of several components working at different frequency, the adopted synchronization scheme and its implementation is key for a successful result of the final system design.

A possible solution is to adopt the GALS philosophy which can be seen as an intermediate design style between the fully synchronous and fully asynchronous solutions. The GALS design paradigm was first proposed in [133] and consists basically of partitioning the system architecture in independent synchronous islands while the communication between them is achieved asynchronously. It is therefore a natural enabler for low-power dynamic voltage and frequency scaling (DVFS) and low noise. The GALS paradigm has been frequently experimented by using asynchronous logic [63, 65]. ETH labs developed a complete GALS methodology leveraging the use of pausable clocks [7]. IHP labs designed a 802.11a GALS baseband processor including various IP cores, a viterbi decoder, FFT, IFFT and a Cordic processor. Several asynchronous NoCs have been also proposed: Mango [65], QNoC,

ANoC, CHAIN [124], FAUST [72], Alpin, Magali. A chip dedicated to flexible baseband processing for 3G/4G wireless telecommunication applications and making use of an asynchronous NoC is described in [58, 59]. However, currently the intricacy of asynchronous design and its poor CAD tool support makes the design of hard macros with ad hoc techniques [49] the only viable solution for industrial exploitation [60]. This comes at the cost of large area and penalizes flexibility.

The practical viability of synchronizer-based GALS networks has been demonstrated in [72], where the hierarchical clock tree synthesis technique for such systems is detailed. Since there is no parametric exploration there, it is not possible to validate a common opinion (for instance, reported in [50, 53, 73]) that large on-chip power consumption can be reduced by replacing the balanced clock tree with a GALS clocking scheme which only guarantees minimal clock skew within the local processing elements. Works in [35, 44, 45, 48] claim that the clock tree design and tuning for variability robustness comes at an non-negligible cost. A circuit switched source synchronous GALS link is described in [55], making use of long distance interconnect paths. It is then experimented on a 65nm reconfigurable NoC for an heterogeneous GALS many-core platform. However, there is no comparison whatsoever with alternative clocking styles and/or implementations. In [121] a many-core heterogeneous computational platform employing GALS compatible circuit switching on-chip network has been presented. [103] presents an example of mesh-connected GALS chip multiprocessor. The work shows that the typical performance penalties of GALS systems (mainly due to additional communication latency) can be hidden by using large FIFO buffers. In [72], the physical implementation of the DSPIN network-on-chip in the FAUST architecture has been presented. In [104] a cost effective solution for asynchronous delay-insensitive on-chip communication is proposed. The solution is based on the Berger coding scheme and allows to obtain a very low wire overhead. [107] proposes a new asynchronous NoC architecture aiming at low latency transfers. This architecture is implemented as a GALS system, where chip units are built as synchronous islands, connected together using a delay insensitive asynchronous Network-on-Chip topology.

2.3.1 Synchronization Interfaces

At interfaces level, many GALS wrappers have been proposed in [109], [117], [99], [95], [96] to provide fast and reliable asynchronous communication services. Several works leverage the timing determinism provided by GALS

wrapper to facilitate debug and testing of GALS systems [135], [136]. A mesochronous link is integrated within a multi-processor tiled architecture based on a Network-on-Chip communication backbone on a CMOS 65nm technology in [53]. The work builds on a full-duplex link architecture illustrated in [52] and on integrated flow control [51]. The baseline mesochronous synchronizer is instead proposed in [73]. However, the synchronizer is still an external module to the NoC. The same drawback is exposed by a different synchronization architecture, detailed in [130]. GALS modularity has been used also along with the concept of voltage-frequency islands (VFIs) which has been recently introduced for achieving fine-grain system-level power management. [81] proposes a design methodology for partitioning a NoC architecture into multiple VFIs. [79] and [80] proposes a complete dynamic voltage and frequency scaling architecture for IP units integration within a GALS NoC.

Examples at industrial level are presented in [78], [77], [102], [73]. In [77] authors examine the use of GALS techniques to address on-chip communication between different synchronous modules on a bus. Issues related to validation, module interfaces and tool flows, while looking at advantages in power savings, timing closure and Time-to-Market/Time-to-Money (TTM) are explored. [102], [73] both suggest to implement the boundary interface with a source-synchronous design style and propose some form of ping-pong buffering to counter timing and metastability concerns.

A well established solution for mesochronous synchronization is illustrated in [113] consists of delay-line synchronizers, using a variable delay on the data lines. This delay is computed in such a way to avoid switching in the metastability window of the receiving registers. Variable delay lines make this solution expensive and not always available in standard cell libraries. This is the same problem of the works in [105, 106], which use voltage comparators.

Several periodic synchronizers are illustrated in [94], which avoid metastability by delaying either the data or the clock signal to sample data when the clock is stable. Configurable digital delay lines are again needed and experimented frequency is very low. The same authors in [134] illustrate many ways to "fool" a synchronizer thus showing several weaknesses of the proposed approaches. The works in [70, 83, 105] achieve mesochronous data synchronization by using Muller C-elements and digital delay lines that are typically designed with a full-custom approach. [83] presents a self-tested self-synchronization method for mesochronous communication. The scheme uses two clocks with a phase shift of 180° and a failure detector is used to select which one to use. In [70] a phase detector in place of a metastability detector is used in the same scheme.

Architectures based on FIFO synchronizers are proposed in [82,84,113]. FIFO size in [84] depends on the skew, hence is link-dependent or given in the worst-case. Implementation is also very expensive, as showed in [88]. More recently, an optimized bi-synchronous FIFO has been proposed in [90] featuring low-latency and small footprint. It can be adapted to the mesochronous needs while proving able to tolerate skew only up to 50% of the clock period. An early mesochronous scheme for the SoCBus NoC was proposed in [54], aiming at compact realization while still lacking of a validation on an NoC test case. A significant step forward comes from the OCN system [98], which uses a source synchronous scheme. A matched-delay architecture is used to compensate the strobe skew and enable high-speed mesochronous communication. A FIFO synchronizer is used at the receiver side.

Summing up, mesochronous synchronizers surveyed so far incur several disadvantages: high implementation overhead, use of non-trivial or full-custom components or low skew tolerance. Moreover, very few works are able to assess timing margins with layout awareness. In this thesis, the same approach to synchronization of [73,91] is taken (source synchronous data transmission, safe storage of data at the receiver side, sampling in the receiver domain only when data is stable). However, the baseline architectures and circuits are improved by providing a more compact and equally robust solution. This is used just as the baseline architecture for the sake of comparison with the novel synchronization structure that it is proposed. In general, a tight integration of synchronization interfaces into NoC building blocks to cut down on latency, area and power is advocated. Furthermore, a wide range of architecture alternatives and even port-level configuration capability is provided. Moreover, we identify the distinctive design constraints of the new schemes and perform a design space exploration of mesochronous NoC links and switches to capture how timing margins can be preserved for different combinations of synthesis and layout constraints. Finally, we provide a system-level demonstration of the cost-effectiveness of the proposed GALS NoC. Last, we perform a cross-benchmarking between a fully synchronous NoC and a mesochronous NoC to be utilized in GALS systems where the frequency decoupling between the NoC and the cores is achieved by using dual-clock FIFO interfaces.

3

Relaxing the Synchronization Assumption in Networks-on-Chip

THIS chapter introduces the first challenge we aim to solve in this thesis: the synchronization design issue. In a first step, the motivation for the adoption of synchronization mechanisms in the Network-on-Chip environment will be discussed. Next, the target GALS platform of this thesis along with the architecture of the basic switch block required to build it will be presented. Last, the focus will be on the baseline mesochronous synchronizer and its improvements that led to a new integrated and flexible switch architecture. Furthermore, a design space exploration of the mesochronous link will be carried out highlighting several interesting design points specific to each of the synchronization alternatives.

3.1 Limitations of the Fully Synchronous Approach

Network-on-chip (NoC) communication architectures are being widely adopted in multi-core chip design to ensure scalability and facilitate a component-based approach to large-scale system integration. As technology advances into aggressive nanometer-level scaling, a number of design challenges emerge from technology constraints which require a continuous evolution of NoC implementation strategies at the circuit and architectural level.

Synchronization is today definitely among the most critical challenges in the design of a global on-chip communication infrastructure, as emerging variability, signal integrity, power dissipation limits are contributing to a severe breakdown of the global synchronicity assumption when a logical structure spans more than a couple of millimeters on die [62]. NoCs typically span the entire chip area and there is today little doubt on the fact that a high-performance and

cost-effective NoC can only be designed in 45nm and beyond under a relaxed synchronization assumption [118].

3.2 A Possible Solution: the GALS Design Style

One effective method to address the synchronization issue is through the use of globally asynchronous locally synchronous (GALS) architectures where the chip is partitioned into multiple independent frequency domains. Each domain is clocked synchronously while inter-domain communication is achieved through specific interconnect techniques and circuits. Due to its flexible portability and transparent features regardless of the differences among computational cores, GALS interconnect architecture becomes a top candidate for multi- and many-core chips that wish to get rid of complex global clock distribution networks.

In addition, GALS allows the possibility of fine-grained power reduction through frequency and voltage scaling [36]. Since each core or cluster of cores operates in its own frequency domain, it is possible to reduce the power dissipation, increase energy efficiency and compensate for some circuit variations on a fine-grained level.

Among the advantages of a GALS clocking style, it is worth mentioning [55]:

- GALS clocking design with a simple local ring oscillator for each core eliminates the need for complex and power hungry global clock trees.
- Unused cores can be effectively disconnected by power gating, thus reducing leakage.
- When workloads distributed to cores are not identical or feature different performance requirements, we can allocate different clock frequencies and supply voltages for these cores either statically or dynamically. This allows the total system to consume a lower power than if all active cores had been operated at a single frequency and supply voltage [42].
- We can reduce more power by architecture-driven methods such as parallelizing or pipelining a serial algorithm over multiple cores [43].
- We can also spread computationally intensive workloads around the chip to eliminate hot spots and balance temperature.

The methodology of inter-domain communication is a crucial design point for GALS architectures. One approach is the purely asynchronous clockless handshaking, that uses multiple phases (normally two or four phases) of exchanging control signals (request and ACK) for transferring data words across clock domains [38,40]. Unfortunately, these asynchronous handshaking techniques are complex and use unconventional circuits (such as the Muller C-element [41]) typically unavailable in generic standard cell libraries. Besides that, because the arrival times of events are arbitrary without a reference timing signal, their activities are difficult to verify in traditional digital CAD design flows.

The so-called delay-insensitive interconnection method extends clockless handshaking techniques by using coding techniques such as dual-rail or 1-of-4 to avoid the requirement of delay matching between data bits and control signals [104]. These circuits also require specific cells that do not exist in common ASIC design libraries. Quinton et al. implemented a delay-insensitive asynchronous interconnect network using only digital standard cells; however, the final circuit has large area and energy costs [66].

Another asynchronous interconnect technique uses a *pausible* or *stretchable* clock where the rising edge of the receiving clock is paused following the requirements of the control signals from the sender. This stops the synchronizer at the receiver until the data signals stabilize before sampling [67, 68]. The receiving clock is artificial in the sense that its period can vary cycle by cycle, therefore, it is not suitable for processing elements with synchronous clocking that need a stable signal clock in a long enough time. Besides that, this technique is difficult to manage when applied to a multiport design due to the arbitrary and unpredictable arrival times of multiple input signals.

An alternative for transferring data across clock domains is the source-synchronous communication technique that was originally proposed for off-chip interconnects. In this approach, the source clock signal is sent along with the data to the destination. At the destination, the source clock is used to sample and write the input data into a FIFO queue while the destination clock is used to read the data from the queue for processing. This method achieves high efficiency by obtaining an ideal throughput of one data word per source clock cycle with a very simple design that is also similar to the synchronous design methodology; hence it is easily compatible with common standard cell design flows [69, 109–111]. Unfortunately, correct operation of source-synchronous links is very sensitive to routing delay mismatches between data and the strobe signals, and hence to delay variability. Therefore, it is very challenging in the context of nanoscale silicon technologies.

In general, each GALS paradigm has its own pros and cons. Fully asynchronous design techniques are a more disruptive yet appealing solution, although their widespread industrial adoption might be slowed down by the relevant verification and design automation concerns they raise. Moreover, they tend to be quite area greedy, at least when timing robustness is enforced. In this context, a full custom design style is still the safer strategy for their successful utilization, hence asynchronous NoC building blocks are often instantiated as hard macros.

For this reason, this chapter will not review fully asynchronous NoC architectures, but will rather focus on synchronizer-based GALS architectures, and on source synchronous communication in particular. Using synchronizers for GALS NoC design implies an incremental evolution of mainstream EDA design tools and paves the way for compatible architectures (with careful synchronizer engineering) with a standard cell design flow. This chapter is fully devoted to this kind of GALS architectures.

The remainder of this chapter is organized as follows: Section 3.3 will describe the target GALS architecture under analysis in this thesis. Section 3.4 will present the switch architecture of reference that has been evolved in a sort of *galsification* process till building a GALS NoC switch. In order to achieve this, Section 3.5 presents the baseline loosely coupled synchronizer architecture and its optimizations that led to the novel tightly integrated synchronizer architecture presented in Section 3.6. Section 3.7 presents a theoretical analysis on the synchronization constraints which highlights the distinctive features of each synchronization scheme studied in the chapter. Section 3.8 presents the experimental results for such architectures. In Section 3.9, a design space exploration of the mesochronous link will be presented illustrating several design points an architect can choose from when instantiating a GALS NoC. Finally, Section 3.10 summarizes the contribution of this chapter.

3.3 Target GALS Architecture

There exist several options when implementing a GALS architecture. From a pure implementation viewpoint, one method consists of asynchronous clock-less handshaking, which uses multiple phases of signal exchange to transfer data. Due to the round-trip signal exchange, the transfer latency between two consecutive data words is high. Besides that, the asynchronous clock-less circuits are difficult to verify in traditional CAD flows, and they also demand a comparatively large area and energy requirements [66, 109].

In the system we implemented in this thesis, the on-chip network is seen as an independent clock domain. Therefore, part of the circuitry is devoted to reliably and efficiently move data across asynchronous clock boundaries between NoC switches and connected network interfaces. These latter are assumed to be part of the clock domain of the IP core that they serve. Dual-clock FIFOs are an effective solution to provide asynchronous boundary communication, especially in throughput-critical interfaces. However, many designers are skeptical about their utilization due to the relevant latency, area and power overhead they incur. Beyond urging research activities aiming at the optimization of dual-clock FIFO architectures, this fact emphasizes the need for their conscious use in GALS systems (see our work in [2] which is not in the focus of this thesis though).

Aware of this, we try to minimize their usage as much as possible by instantiating them only at IP core boundaries, after their respective network interfaces (see Figure 3.1. However, this choice moves many chip-wide timing concerns to the on-chip network. In fact, this latter ends up spanning the entire chip and might be difficult to clock due to the growing chip sizes, clock rates, wire delays and parameter variations. As previously anticipated, in 45nm and beyond, a global clock signal spanning the whole chip area will be difficult to control and even to realize with a controllable phase-offset.

In fact, we find that mesochronous synchronization can relieve the burden of chip-wide clock tree distribution while requiring simpler and more compact synchronization interfaces than dual-clock FIFOs. Hierarchical clock tree synthesis is an effective way of exploiting mesochronous links, as already experimented in [72]. During the first step, a clock tree is synthesized for each network switch with a tightly controlled skew (e.g.,5%). Next, each clock tree is characterized with its input delay, skew and input capacitance. This information is used by the clock tree synthesis (CTS) tool to infer a top clock tree balancing the leaves with a much looser skew constraint (e.g., 30/40%). The ultimate result is a global clock tree which consumes less power then the traditional one generated by enforcing chip-wide skew constraints. For future large multiprocessor systems-on-chip, the use of this methodology can be not just an issue of power efficiency but even of CTS feasibility.

However, power savings with this methodology should not be taken for granted, since it involves some overheads: the transmission of the clock signal across mesochronous links, the mesochronous synchronizers themselves (implementing power-hungry buffering resources) and the increased number of buffer slots needed at link end-nodes to cover the larger round-trip time (asso-

ciated with the synchronization latency) for correct flow control management.

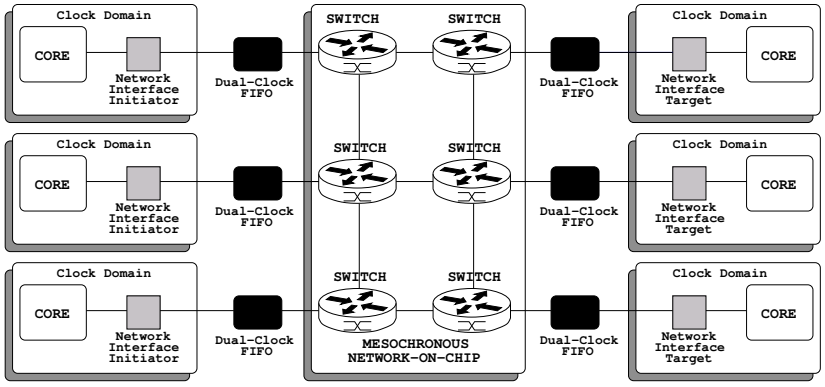


Figure 3.1: Target Design Platform.

Our design platform aims at minimizing such overheads through a novel mesochronous architecture taking advantage of the tight integration of the synchronizer into the switch architecture. However, since these solutions give rise to timing constraints that might not be verified for specific layout conditions, we provide architecture variants for these cases as well, thus coming up with a flexible switch suitable for many design instances.

In this direction, moving from a system-level to a switch block view, next section will present the fully synchronous baseline switch architecture that will be used as starting point for the implementation of our target GALS architecture. In fact, such an architecture will be augmented with various mesochronous synchronizer variants as we will see later on in this chapter.

3.4 ×pipesLite switch architecture

The ×pipesLite network-on-chip architecture has been conceived for the resource-constrained Multi-processor system-on-chip (MPSoC) domain. Therefore, it features a high degree of parameterization and compact implementation [92]. It is unpipelined, fully synthesizable with a standard design flow and achieves frequencies around 1.5GHz for the fastest configurations.

The ×pipesLite switch is conceived as a soft macro from the ground up. The possibility of design-time tuning of parameters such as flit width, number of I/O ports, buffer size and flow control policy makes it suitable to explore several topologies and architectural variants.

The baseline architecture implements an in/out buffered switch implementing wormhole switching and source-based routing, as depicted in Figure 3.2.

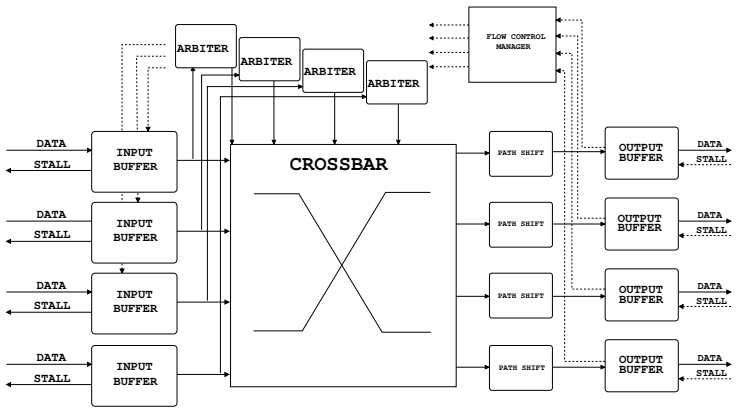


Figure 3.2: Baseline switch architecture.

The crossing latency is therefore equal to 1 cycle in the upstream/downstream link and 1 cycle inside the switch itself. The choice for retiming at input and output ports stems from the need to break the timing path across switch–to–switch links. The first chapters of this thesis show that in 65nm technology the delay of inter-switch links causes a significant performance drop for most regular NoC topologies depending on the intricacy of their connectivity pattern. This is certainly technology-, architecture- and topology-specific and depends on the specific link inference technique too, but a rule of thumb is that with target operating speeds above 700 MHz, even for 2D mesh topologies, the target speed is likely to be achieved by leveraging input and output retiming in the switch. This is due to the increasing role of RC propagation delay across the interconnects and will hold even more in future technology nodes.

The ×pipesLite switch relies on a stall/go flow control protocol [3]. It requires two control wires: one going forward and flagging data availability (“valid”) and one going backward and signaling either a condition of buffer filled (“stall”) or of buffer free (“go”). With this scheme, power is minimized since any congestion issue simply results in no unneeded transitions over the data wires. Moreover, recovery from congestion is instantaneous. The input buffer serves as a retiming stage, while at the same time handling flow control. For this purpose, it needs to be 2 slots deep. During normal operation, only one slot is used. When a “stall” is notified by the downstream stage, output data

of the buffer is frozen. However, it takes one cycle to propagate the "stall" upstream, during which a new input data is driven and needs to be stored in a backup slot. This justifies the need for 2 slots. The architecture can be seen as a synthesizable realization of the elastic buffer concept. The output buffer has a tunable size for performance optimization, and handles flow control as well. Therefore, it shares the underlying architecture design techniques with the input buffer. Arbitration is not centralized, i.e., there is a round-robin arbiter for each output port serializing conflicting access requests for that output port. The critical path of the switch starts from the finite state machine of the switch input buffer, goes through the arbiter, the crossbar selection signals, some header processing logic and finally includes a library setup time for correct sampling at the switch output port (see Figure 3.2). The combinational logic shifts the routing bits in the packet header in such a way that each switch reads the target output port for that packet in the first position.

In order to build our GALS platform, mesochronous synchronization needs to be implemented by means of synchronizers. We envision the design of such synchronizers as a seamless replacement of the switch input buffer of Figure 3.2 thus effectively coming up with the new GALS switch architecture depicted in Figure 3.3.

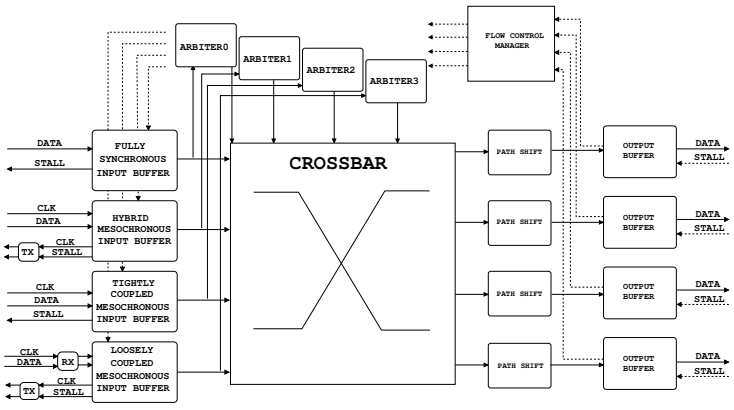


Figure 3.3: GALS switch architecture.

The architecture flexibility, provided by all the synchronizer interfaces variants, enables a fully configurable switch building block that can be tailored depending on different requirements. In fact, as it will be clear later on in this chapter, different synchronizer implemention choices lead to different trade-

off in terms of performance/link-delay toleration. We see this as a very efficient and cost-effective way to implement a GALS Network-on-Chip because only those switch input ports, that have specific skew requirements, can be equipped with a mesochronous synchronizer whereas in the remaining ports, a fully synchronous input buffer can be instantiated.

All the developed mesochronous synchronizer architecture variants will be detailed in the remainder of this chapter starting from the baseline synchronization architecture described in the next section.

3.5 Baseline Synchronization Architecture

This work moves from the synchronizer architecture presented in [91] and illustrated in Figure 3.4. The circuit receives as its inputs a bundle of NoC wires representing a regular NoC link, carrying data and/or flow control commands, and a copy of the clock signal of the sender. Since the latter wire experiences the same propagation delay as the data and flow control wires, it can be used as a strobe signal for them. The circuit is composed by a front-end and a back-end. The front-end is driven by the incoming clock signal, and strobes the incoming data and flow control wires onto a set of parallel latches in a rotating fashion, based on a counter. The back-end of the circuit leverages the local clock, and samples data from one of the latches in the front-end thanks to multiplexing logic which is also based on a counter. The rationale is to temporarily store incoming information in one of the front-end latches, using the incoming clock wire to avoid any timing problem related to the clock phase offset. Once the information stored in the latch is stable, it can be read by the target clock domain and sampled by a regular flip-flop.

Counters in the front-end and back-end are initialized upon reset, after observing the actual clock skew among the sender and receiver with a phase detector, so as to establish a proper offset. The phase detector only operates upon the system reset, but given the mesochronous nature of the link, its findings hold equally well during normal operation.

Since few flow control wires are traveling backwards, another similar but much smaller synchronizer needs to be instantiated at the sender to handle them.

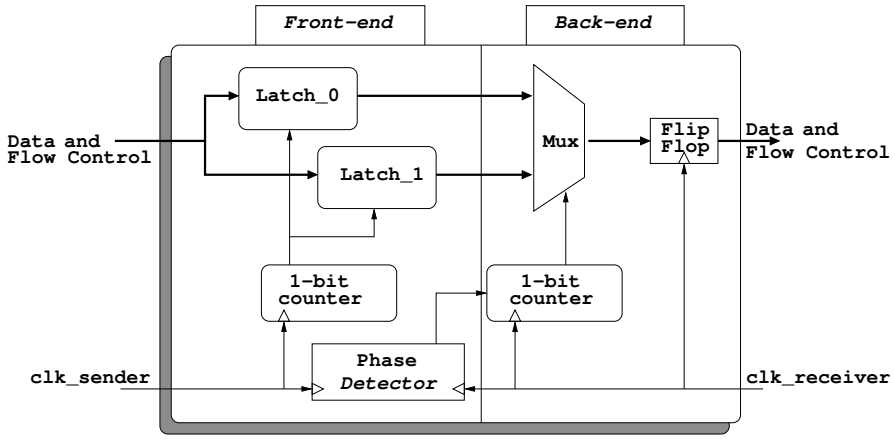


Figure 3.4: Baseline synchronizer architecture of [91].

3.5.1 Optimizations of the baseline architecture: the *loosely* coupled synchronizer

In agreement with [91], it is always possible to choose a counter setup so that the sampling clock edge in the back-end captures the output of the latches in a stable condition, even accounting for timing margin to neutralize jitter. Therefore, no more than 2 latches in parallel are needed in the front-end for short-range (i.e., single cycle) mesochronous communication with this scheme.

However, other considerations suggest that a different choice may be desirable at this point. In particular, by increasing the number of input latches by one more stage, it becomes possible to remove the phase detector (see the new architecture in Figure 3.5). This would be desirable due to the timing uncertainty or the high area footprint or the non-compliance to a standard cell flow that affects many phase detector implementations. A third latch bank allows to keep latched data stable for a longer time window and to even find a unique and safe bootstrap configuration (i.e., counters initialization) that turns out to be robust in any phase skew scenario.

At regime, the output multiplexer always selects the output of the latch bank preceding the bank which is being enabled by the front-end counter. Rotating operation of both front- and back-end counters preserves this order. In contrast to [91], the reset architecture is designed, as Figure 3.5 shows. In most SoCs, the reset signal coming into the chip is an asynchronous input. Therefore, reset de-assertion should be synchronized in the receive clock domain. In fact, if a reset removal to a flip-flop occurs close to the active edge of its clock, flip-flops

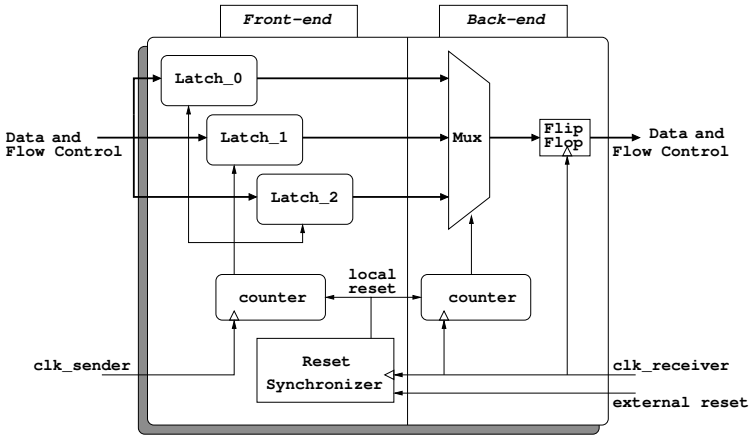


Figure 3.5: The loosely coupled synchronizer of this work.

can enter a metastable state. We use a brute-force synchronizer (available in several new technology libraries as a standard cell, e.g. ST65nm) for reset synchronization with the receiver clock. Now, the challenge is how to reset the front-end. Typically, a reset can be sent by the upstream switch. In our architecture, we prevent metastability in the front-end by delaying the strobe generation in the upstream switch by one clock cycle after reset de-assertion. This way, on the first edge of the strobe signal, the receiver synchronizer is already reset. Such strobe signal generation delay is compliant with network packet injection delay after reset.

The transmitter clock signal is used as the strobe signal in our architecture. Differently than [91], a larger timing margin is enforced for safe input data sampling. In fact, the transmitter clock signal has to be processed at the receiver end in order to drive the latch enable signals. In actual layouts, this processing time adds up to the routing skew between data and strobe and to the delay for driving the latch enable high-fanout nets. As a result, the latch enable signal might be activated too late, and the input data signal might have already changed. In order to make the synchronizer more robust to these events, we ensure that input data sampling occurs in the middle of the clock period. In fact, a switching latch enable signal opens the sampling window of the next latch during the rising edge, and closes the same during the falling one. As a result, the latch enable activation has a margin of half clock cycle to occur. Our post-layout simulations prove that this margin is largely met in practice. Finally, in agreement with [91], we computed the minimum size of the input buffer in the downstream switch to be 4 slots (flits). They are required by

the stall/go flow control protocol in order to cover the round trip latency and not to drop flits in flight when a stall signal has to be propagated backwards. The original input buffer size is 2 slots, reflecting the requirements of stall/go without synchronization. Please refer to [91] for further details about this.

3.6 Tightly Integrated Synchronizer Architecture

The previous synchronizer, similarly to SKIL or to the Polaris one, is a module of the NoC architecture, thus loosely coupled with the downstream switch. The loose coupling stems from the fact that the flip-flop in the synchronizer back-end belongs directly to the switch input buffer. The mux output is therefore sampled like any other input in the fully synchronous scenario. However, our early exploration indicated that the area overhead induced in this input buffer, as an effect of the added synchronization latency, is much larger than the synchronizer area itself.

This hints that a tighter integration of the synchronizer into the switch input buffer is desirable. In particular, the latch enable signals of the synchronizer front-end could be conditioned with backward-propagating flow control signals, thus exploiting input latches as useful buffer stages and not just as an overhead for synchronization. In this case, input data is at first stored in the latches and then synchronized. This allows to completely remove the switch input buffer and to replace it with the synchronizer itself. The synchronizer output is then directly fed to the switch arbitration logic and to the crossbar. The ultimate consequence is that the mesochronous synchronizer becomes the actual switch input stage, with its latching stages acting as both buffering and synchronization stages (see Figure 3.6). As a side benefit, the latency of the synchronization stage in front of the switch is removed, since now the synchronizer and the switch input buffer coincide. The main necessary change to make the new architecture come true is to bring flow control signals to the front-end and back-end counters of the synchronizer. This solution would still require 4 slot buffers, i.e., 4 latching banks. However, a further optimization is feasible. The backward-propagating flow control signal (the stall/go signal) could be directly synchronized with the strobe signal in the synchronizer front-end before being propagated to the upstream switch. This would save also the synchronizer at the transmitter side. In fact, the backward-propagating signal would be already in synch with the strobe, which in turn is in synch with the transmitter clock. The ultimate result is the architecture illustrated in Figure 3.6. We are aware that this latter choice shrinks timing margins for the backward flow con-

trol signal. The reason is that it leaves the downstream switch with a generation delay across its synchronizer and also experiences the link propagation delay. This margin will be assessed post-layout in the experimental section, proving the applicability of the scheme. For this architecture solution, only 3 latching banks are needed in the synchronizer. In practice, only 1 slot buffer more than the fully synchronous input buffer. The tightly coupled synchronizer makes the mesochronous NoC design fully modular like the synchronous one, since no external blocks to the switches have to be instantiated for switch-to-switch communication. Please notice that the reset architecture remains unchanged with respect to Figure 3.5.

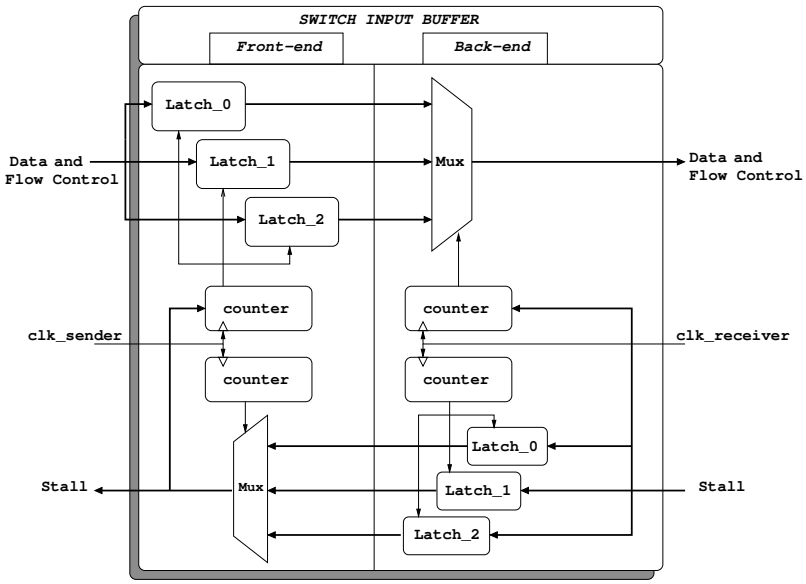


Figure 3.6: Proposed tightly coupled synchronizer.

3.6.1 Operating principle

In case a *go* signal comes from the switch arbiter, at each clock cycle data are latched in the input buffers of the synchronizer, synchronized with the local clock and propagated to the switch arbiter and crossbar. When a *stall* occurs, the output mux keeps driving the same output until communication can be resumed. While the stall signal gets synchronized with the strobe and reaches the front-end, the front-end latches keep sampling input flits in a rotating way. When the stall signal finally leaves the synchronizer, it will stop the transmis-

sion of the upstream switch and the front-end counter operation at the same time. At this point, the situation is frozen. When then a *go* arrives, the output mux becomes operational again. Later, input latches and upstream switch resume their operation again at the same time. Please observe that this mechanism does not waste bandwidth on flow resumption, since the synchronizer backend can immediately start sweeping the output of front-end latches upon receipt of a *go*. Interestingly, flow control logic in the synchronizer is simplified with respect to that of the original switch input buffer. Before, a finite state machine used to generate a stall by monitoring the number of elements in the buffer. When it was equal to one and a stall came from the switch internal logic, than a stall was also generated for the upstream switch. In the new architecture, the synchronizer just synchronizes the stall signal from the switch logic with the transmitter clock and propagates it upstream. This way, a large amount of logic is saved.

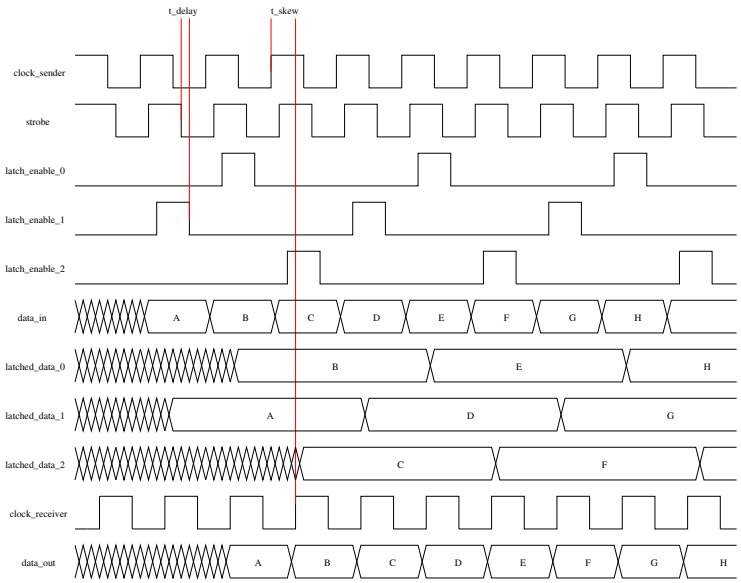


Figure 3.7: Waveforms example of the tightly coupled synchronizer.

Figure 3.7 reports the waveforms showing operation of the tightly coupled synchronizer. A delay from the strobe signal is assumed for the latch enable signals to account for their high-fanout.

3.7 Theoretical Analysis

In this section we analyze the previously presented synchronizer architectures (i.e., *loose* and *tight*) from a formal viewpoint.

The two synchronizer solutions presented so far have a series of common constraints regarding the correct sampling of data and flow control wires. In fact, the basic idea consists of sending *data* (or *stall* hereafter) and clock signal with a null phase offset, latching data at receiver end and performing the sampling only when data is stable. Nonetheless, as discussed in previous sections, there are some architectural peculiarities that differentiate the timing constraints of each solution with respect to another. Let us first analyze the common constraints considering Formula 3.1.

$$T_{setup} \leq \frac{T_{clock}}{2} + T_{latch\_enable} \tag{3.1}$$

For the sake of simplicity, we consider the case where clock and data signals are sent to the link channel without routing skew (perfect wires alignment). At receiver end, when a rising edge of the clock signal occurs, after a further T_{latch_enable} the latch window is transparent and data is captured. When a clock falling edge occurs (after half clock cycle) data is sampled correctly if and only if it was already stable for a T_{setup}. Of course, considering a non-zero routing skew between clock and data wires, a further $T\_routing\_skew$ has to be taken into account at left or right side of the equation depending on the relative position between clock and data signals.

Furthermore, for a correct sampling, data has to be kept stable at least for a T_{hold} before changing. Therefore, within a clock cycle, a $\frac{T_{clock}}{2}$ and a T_{latch_enable} are necessary to sample the data but a further T_{hold} is needed so that it can be considered stable (see Formula 3.2). As before, in a non-zero routing skew exists, it has to be taken into account as it additionally reduces the timing margin for a correct sampling.

$$\frac{T_{clock}}{2} + T_{latch\_enable} \leq T_{clock} - T_{hold} \tag{3.2}$$

So far, we have analyzed timing constraints common to all the proposed solutions. We will now consider, synchronizer by synchronizer, a further timing constraint that is specific for each architecture.

As our previous discussions already indicated, in the loosely coupled architecture, data sent from the upstream switch requires two clock cycles to be

sampled by the downstream switch input buffer. Therefore, within two clock cycles, data has to traverse the link channel in T_{link}, it requires a T_{latch_enable} plus T_{mux} to correctly latch and output data from the external synchronizer towards the switch. A further T_{setup} is necessary to guarantee correct sampling by the downstream switch input buffer (see Formula 3.3). The entire data-path has to be traversed in two clock cycles, 1 cycle to synchronize data by the loosely coupled synchronizer plus a further clock cycle to sample data by switch input buffer. Obviously, a trade-off between clock frequency and link length is necessary for a correct operation of the entire system that would otherwise fail by violating the T_{setup} of the input buffer.

$$2 \cdot T_{Clock} \geq T_{skew} + T_{link} + T_{latch\_enable} + \\ + T_{mux} + T_{setup} \tag{3.3}$$

The same reasoning holds for the tightly coupled architecture with some differences though. In fact, sampling elements in the switch–to–switch data-path are the respective output buffers of the upstream and downstream switches. This path has to be traversed in two clock cycles: a first clock cycle is needed to send data from the upstream switch and to synchronize it by the multi-purpose switch input buffer. A second clock cycle is required to forward data from the switch input buffer to the output buffer of the same switch building block (see Formula 3.4).

$$2 \cdot T_{clock} \geq T_{skew} + T_{link} + T_{latch\_enable} + \\ + T_{mux} + T_{arbiter} + T_{crossbar} + \\ + T_{shift\_header} + T_{setup} \tag{3.4}$$

Differently from the loosely coupled architecture, Formula 3.4 points out the extra timing required to traverse the switch building block (e.g., arbitration time, crossbar traversal, etc.). In this case, a violation of the above would result in a sampling failure of the output buffer. A further timing constraint for the tightly coupled architecture is that illustrated by Formula 3.5.

$$T_{clock} \geq T_{generation} + 2 \cdot T_{link} + T_{counter} + \\ + T_{mux} + T_{setup} \tag{3.5}$$

We name this constraint as the *round-trip* dependency of the tightly coupled synchronizer. In fact, within a single cycle, the clock sent by the upstream

switch triggers the bottom counter of the downstream switch synchronizer front-end (that is in charge of flow control synchronization, see Figure 3.6). Once a multiplexer output has been selected, the just synchronized stall signal can be forwarded to the upstream switch output buffer to stop data transmission. Obviously, for a correct sampling, the output buffer requires the stall signal to be stable for at least T_{setup}. The reason of this constraint is that in a single clock cycle, the stall signal has to be synchronized (at downstream switch end) and forwarded to the upstream switch to stop data transmission. The loosely coupled synchronizer is not affected by this constraint as the flow control synchronization is performed by a 1-bit synchronizer instantiated at upstream switch side. Intuitively, this constraint strongly limits the maximum operating frequency for a given link length of the tightly coupled architecture.

3.7.1 Architecture flexibility: the Hybrid solution

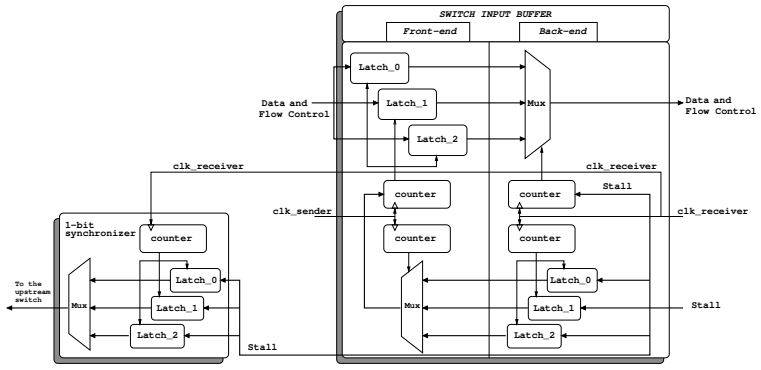


Figure 3.8: The hybrid architecture with a 1-bit synchronizer on the receiver end.

In order to alleviate the limitation of the tightly coupled architecture, resembled by Formula 3.5, the hybrid architecture has been envisioned. In fact, the stall signal generated by the downstream switch arbiter is not synchronized in the backend of the hybrid synchronizer but is directly forwarded to a 1-bit synchronizer instantiated in front of the upstream switch. This way, the *round-trip* dependency can be broken and Formula 3.5 does not hold for such an architecture anymore. Obviously, being the data-path of the hybrid architecture exactly the same as that of the tightly one, Formula 3.4 still remains valid but a new constraint for the stall signal path arises. The new constraint (see Formula 3.6) encompasses the time required to generate the stall signal by the arbiter (the higher the switch radix, the slower is the stall generation as the arbiter size

increases); a T_{link} to traverse the link channel; the time necessary to the 1-bit synchronizer to latch and output the stall signal towards the upstream switch output buffer which requires a further T_{setup} to sample the stall signal.

$$2 \cdot T_{clock} > T_{stall\_generation} + T_{skew} + T_{link} + \\ + T_{latch\_enable} + T_{mux} + T_{setup} \tag{3.6}$$

Results in Section 3.9.1 will give an experimental evidence of the analysis presented above. While describing the specific constraints for each architecture, we omitted commenting T_{skew} parameter. It represents a misalignment between upstream and downstream clock (same clock frequency f, different phase offset). In all the presented architectures, T_{skew} is a decreasing factor for a correct operation as it reduces the timing margin of the analyzed system.

Another degree of flexibility of our architecture is that a switch can be assembled out of a mix of synchronous and mesochronous ports. In fact, the output architecture of the tightly coupled synchronizer resembles that of a synchronous switch input buffer, therefore for the switch data-path and control logic it is irrelevant whether the input port is synchronous or mesochronous. A flexible heterogeneous switch architecture can therefore be built, where input ports are either the conventional 2-slot buffer of synchronous switches or the tightly coupled synchronizer. Finally, an external mesochronous synchronizer can also be instantiated in front of the synchronous switch input ports to infer, whenever needed, the loosely coupled synchronization architecture.

Summing up, the three synchronizer architectures described so far enable the building process of GALS systems in a flexible and scalable way. In fact, depending on the link length (and an associated link delay), one of the different three solutions can be implemented in order to achieve a reliable timing margin for a correct operation of the system.

3.8 Experimental Results

This part of the chapter will characterize the mesochronous interfaces presented so far from the, latency, area footprint and power consumption viewpoint. In order to achieve such goal, both *loose*, *tight* and *hybrid* mesochronous interfaces have been implemented by means of the ×pipesLite NoC library [92]. Synthesisis and place&route have been performed through a backend synthesis flow leveraging industrial tools. The technology library utilized is a low-power low-Vth 65nm STMicroelectronics library (CMP project [146]).

3.8.1 Comparative latency Analysis

Since the tightly coupled synchronizer not only changes the synchronizer implementation but also affects the entire network architecture, we performed basic tests to capture the macroscopic performance differences implied by the different synchronization architectures. We focus on synchronization latency, since the stall/go mechanism implemented in our synchronizer ensures that a stall-to-go transition of the flow control signal can be immediately propagated to the next stage. Hence, there are no wasted cycles at flow resumption, differently than [91]. Since what matters here is not a network-wide performance analysis, but just to investigate the latency of each scheme, this feature can be more conveniently stimulated and analyzed in a simple ad-hoc experimental test case for fine-grain performance analysis. We opted for a simple processor–NoC–memory topology (see Figure 3.9).

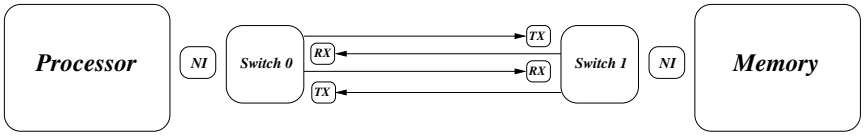


Figure 3.9: Test-case platform under analysis.

The investigated NoC is comprised of a couple of 2x2 switches respectively connected to the processor and the memory. Furthermore, each network switch is connected to its own RX-, TX-mesochronous part meant for synchronizing received data and flow control signals. For the sake of comparison, the SKIL synchronizer is considered as well. This is another loosely coupled module with the switch architecture [73].

The traffic pattern consists of full-bandwidth read and write transactions, i.e., the target memory never stops the access flow. Of course, the only performance differentiation is seen for read transactions, since they are blocking for the processor core, hence they rely on the network ability to keep latency to a minimum. Performance results could be easily interpreted by means of a simple analytical model (Formula 3.7). It relates performance results to the intrinsic design characteristics of each synchronizer.

$$\texttt{cycles} = \texttt{n} + \textit{latency} \times 2 \times \texttt{\#transactions} \tag{3.7}$$

In fact in the best case, SKIL exposes two cycles synchronization overhead plus a further execution cycle for traversing the network switch; whereas our

loosely coupled solution only requires one cycle latency in the mesochronous plus one cycle in the network switch. Even better, the tightly coupled mesochronous synchronizer requires the same computational resources of the vanilla switch (i.e., 1 execution cycle). The reason is that the tightly coupled solution seamlessly replaces the input buffer of the network switch thus providing a fast, reliable and robust mechanism for data synchronization. As for the tightly coupled, the hybrid solution as well only requires a single execution cycle in order to synchronize the data. The reason is that the synchronization circuit of the data path is the same as that of the tightly coupled interface (i.e., 1 cycle). Regarding the flow control, the stall signal in the hybrid architecture is directly forwarded from the arbiter to the TX- module instantiated in front of the upstream switch. This module is the one taking care of synchronizing the flow control signal in the same clock cycle when it has been forwarded.

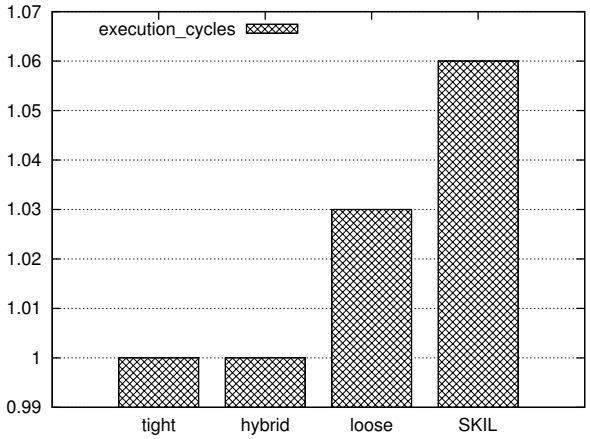


Figure 3.10: Normalized cycle latency of the different synchronization schemes.

Summarizing, whenever the system with the tightly (or hybrid) coupled mesochronous synchronizer performs a computational task in `n` cycles, the alternative schemes, i.e., SKIL and the loosely coupled synchronizer respectively require a number of cycles equal to Formula 3.7, where *latency* is the number of clock cycles of the deployed mesochronous architecture whereas *#transaction* is the number of read operations performed by the processor unit. As depicted in Figure 3.10 there is a direct impact of the adopted synchronization solution on the overall system performance. While the tightly coupled and hybrid solutions keep the same performance as the vanilla network switch, a performance drop up ranging from 3% up to 6% incurs when using a loosely coupled or the SKIL scheme respectively.

Area Overhead

In order to estimate the area savings by using the tight integration design strategy, we went through a commercial synthesis flow and refined RTL description of the mesochronous switches (tightly, hybrid and loosely coupled) down to the physical layout. All the systems were synthesized, placed and routed at the same target frequency of 1GHz. In Figure 3.11, the area footprint of switches is reported along with a breakdown pointing out the contribution of synchronizers and/or input buffers.

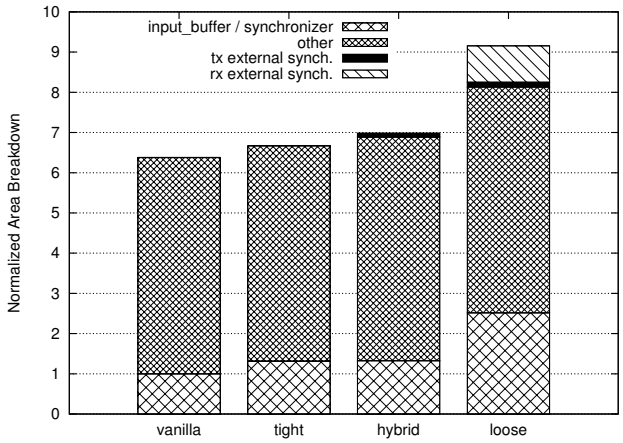


Figure 3.11: Area breakdown of a switch block with its synchronization scheme.

The tightly, hybrid and loosely coupled solutions are compared against a vanilla (i.e., fully synchronous) switch; *input_buffer* area for this switch only refers to the area occupancy of a normal 2 slot input buffer. For the loosely coupled solution, a 4 slot buffer is needed to cover the round trip latency, and this is most of the overhead for this solution. As clearly pointed out by the area breakdown in Figure 3.11, the sum of the transmitter and of the receiver synchronizers is almost equal to that of a 2 slot buffer, i.e., of the input buffer in the vanilla switch. For the tightly coupled solution, *input_buffer/synchronizer* area refers to the multi-purpose switch input buffer (which is also the synchronizer). As for the latter, the hybrid architecture result refers to multi-purpose switch input buffers plus as many instances of TX- synchronizer as input ports. Clearly, there is almost no area overhead when moving from a fully synchronous to a tightly (or hybrid) integrated mesochronous switch as they employ similar buffering resources.

From the performance viewpoint, our post-layout synthesis results confirm that

the critical path of the switch is not impacted by the replacement of the vanilla input buffer with the tightly integrated mesochronous synchronizer. By experimenting with different switch radix, the critical path deviates only marginally in the two cases, therefore no performance penalty should be expected for the mesochronous switch.

Power Consumption

A further step of our exploration was to contrast power consumption of the proposed mesochronous schemes. Our target design is a 5x5 switch in four different variants: the first is a fully synchronous switch block, the second has a tightly integrated mesochronous synchronizer per each input port; the third one is a switch utilizing a hybrid mesochronous synchronizer per input port thus requiring also a tx-synchronizer on the sender side; the last variant has a pair of loosely coupled rx- and tx-synchronizers per input port. Three different traffic patterns have been utilized to carry out an accurate power analysis: *idle*, request for a *random* output port and *parallel* communication flows. Post-layout simulation frequency was 700MHz for all the designs. As showed in Figure 3.12, in all the cases, the highest power consumption is consumed by the most buffer demanding solution, i.e., the loosely coupled design. Power consumption of the vanilla and tightly coupled designs are similar as expected; this is mainly due to the equivalent buffering resources deployed in both switches. The hybrid solution is a bit more expensive compared to the tight and the vanilla mainly because there is a small extra buffering due to the 1-bit synchronizer for each mesochronous port.

Next section will perform a design space exploration of the mesochronous link architecture in order to assess several quality metrics of the synchronizer interface presented so far.

3.9 Mesochronous Link Design Characterization

The above architecture provides degrees of freedom for port-level selection of the most suitable synchronization option based on timing and layout constraints. The following design space exploration of a mesochronous link implemented with our architecture will provide the guidelines for such port-level selection, and is therefore an essential enabler for automatic assembly of the target GALS NoC.

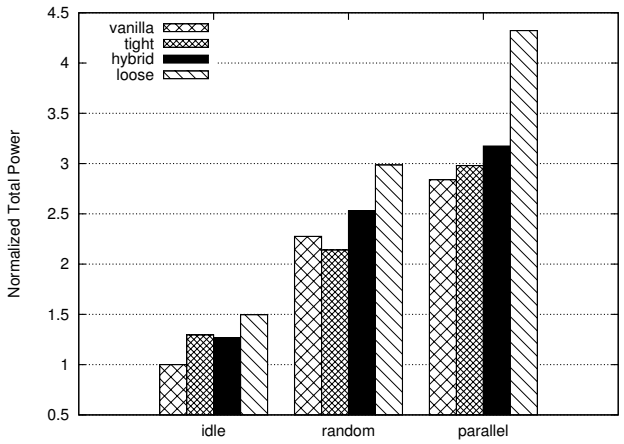


Figure 3.12: Normalized power consumption of different synchronization schemes in different traffic scenarios.

3.9.1 Design tradeoffs

A series of experiments have been carried out in order to characterize, for each synchronizer architecture, the maximum operating frequency for a given link length between two switch building blocks. In practice, a 5x5 switch, ideally extracted from the center of a 2D mesh, has been considered after place&route. The switch has been synthesized with a very tight timing constraints (1 GHz), so that after place&route the critical path for all the architectures will be in the switch–to–switch link. The switch is connected to a tester injecting clock and data with an increasing delay. The utilized testbench assumes an ideal alignment between clock signal and data as well as no routing skew. This way, we can assess for a certain operating frequency, the relative link delay supported by each architecture. Obviously, a certain link delay corresponds to a relative channel length depending on how the link synthesis policy is chosen.

The theoretical analysis discussed in Section 3.7 is here confirmed by experimental results. In fact, Figure 3.13 reports maximum operating frequency of each synchronization scheme for a certain link delay. Obviously, being affected by the round-trip constraint, the tightly coupled architecture turns out to be the less delay tolerant. Indeed, an increment of either frequency or delay would result in packets loss due to the late arrival of the backward propagating signal. However, such supported link delays enable the tightly coupled architecture to sustain a correct communication within a typical range of link length in nanoscale technologies. On the other hand, the hybrid alterna-

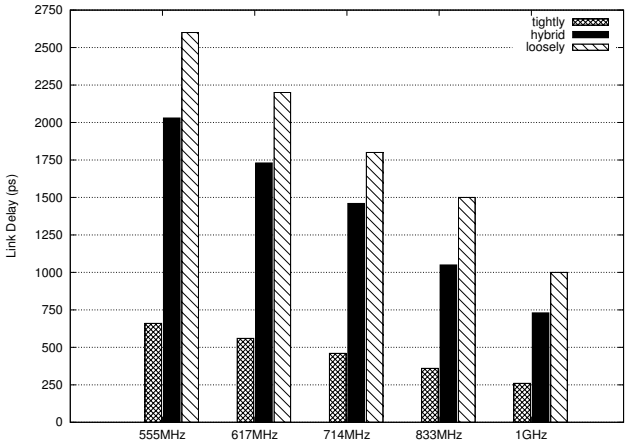


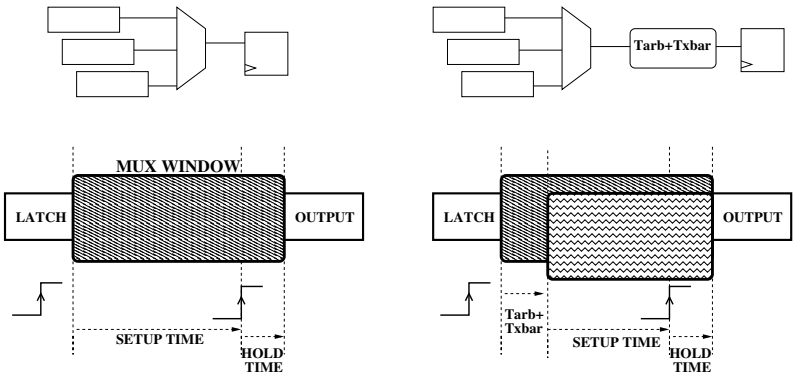
Figure 3.13: Operating frequency and tolerated link delay of different synchronizers.

tive is quite effective as it relieves the round-trip constraint while keeping the area and power cost as low as the tightly coupled architecture. In fact, by using a small single-bit synchronizer at transmitter end for the backward signal, round trip dependency can be removed thus increasing maximum achievable frequency. Best results in terms of link delay toleration for a given frequency are achieved by the loosely coupled architecture. However, this result comes at a high area/power and also latency cost.

3.9.2 Skew Tolerance

Skew tolerance of our architecture schemes depends on the relative alignment of data arrival time at latch outputs, multiplexer selection window and sampling edge in the receiver clock domain. A few basic definitions help to assess the interaction among these parameters in determining skew tolerance. For the loosely coupled synchronizer, such definitions are pictorially illustrated in Figure 3.14(a).

During the *mux window*, data at latch outputs is selected for forwarding to the sampling flip-flop in the switch input port. Its duration closely follows that of the clock period. Sampling occurs *on the next rising edge of the receiver clock inside the mux window*. We denote the time between the starting point of the mux window and such sampling instant as the *Setup time*. Conversely, after an *Hold time* since the rising edge of the clock the mux window terminates. This



(a) Loosely coupled synch. (b) Tightly coupled synch.

Figure 3.14: Basic mechanisms affecting skew tolerance.

is the time required by the counter to switch the multiplexer selection signals.

When we consider the tightly coupled architecture (Figure 3.14(b)), then the same metrics are taken at the switch output port rather than at the multiplexer output of the synchronizer. Therefore, the starting time of the *mux window* is delayed due to the worst case timing path between the synchronizer output and the switch output port, which includes the arbitration time, crossbar selection time and some more combinational logic delay for header processing. At the same time, the sampling rising edge of the receiver clock remains unaltered, therefore the ultimate effect is a shortening of the *Setup time* for the tightly integrated mesochronous switch architecture.

Figure 3.15 quantifies these timing margins for the loosely coupled switch architecture. Results are referred to a 2x2 switch working at 660 MHz after place&route. X-axis reports negative and positive values of the skew, expressed as percentage of the clock period. Setup and hold times have been experimentally measured by driving the switch under test with a clocked testbench, by inducing phase offset with the switch clock and by monitoring waveforms at the switch. The connecting link between the testbench and the switch is assumed to have zero delay. A positive skew means that the clock at the switch is delayed with respect to the one at the testbench. The figure also compares setup and hold times with the minimum values required by the technology library for correct sampling (denoted *FF-Tsetup* and *FF-Thold*).

First of all, we observe that both times are well above the library constraints, thus creating some margin against variability. For the whole range of the skew,

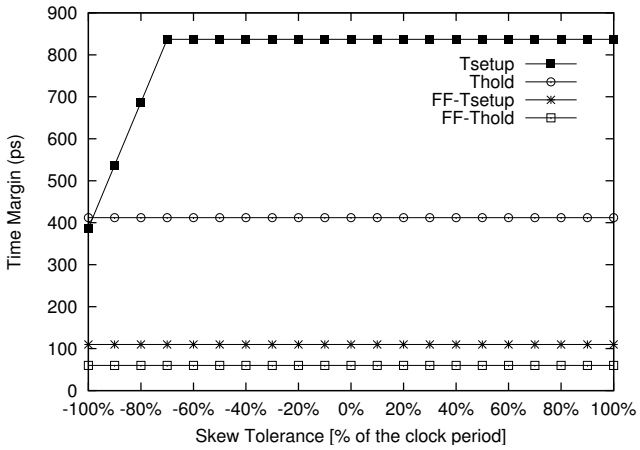


Figure 3.15: T_{setup} and T_{hold} for the loose coupled varying the skew tolerance.

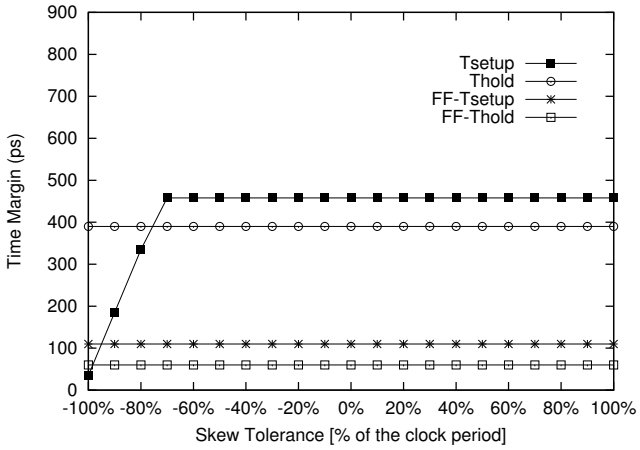


Figure 3.16: T_{setup} and T_{hold} for the tight coupled varying the skew tolerance.

the hold time stays the same. The result is relevant for positive skew, since its effect is to shift the *mux window* to the right, close to the region where latch output data switches. However, the stability window of the latch output data is long enough to always enable correct sampling of stable data before the point in time where it switches.

In contrast, a negative skew causes the *mux window* to shift to the left, therefore as the negative skew grows (in absolute values) the latch output data ends up switching inside the *mux window*, which corresponds to the knee of the setup

time in Figure 3.15. From there on, the switching transient of data becomes closer to the sampling edge of the receiver clock and correct sampling can be guaranteed until the setup time curve equals the *FF-Tsetup* one. However, even for -100% skew synchronizer operation is correct.

Figure 3.16 illustrates the same results for the tightly coupled mesochronous switch. As anticipated above, the setup time is decreased by 370ps, corresponding to the time for arbitration, crossbar selection and shifting of routing bits. Interestingly, the knee of the setup time occurs for the same value of the negative skew, in that the switching instant of the latch output data enters the *mux window* at exactly the same point in time. The ultimate implication is that the tightly coupled synchronizer cannot work properly with -100% skew, since the crossing point with the *FF-Tsetup* occurs at around -95%. In practice, we can conclude that a 2x2 switch with tight coupling of the synchronizer on each port consumes 40% less area and power than its loosely coupled counterpart while incurring a 23% degradation of the maximum skew tolerance.

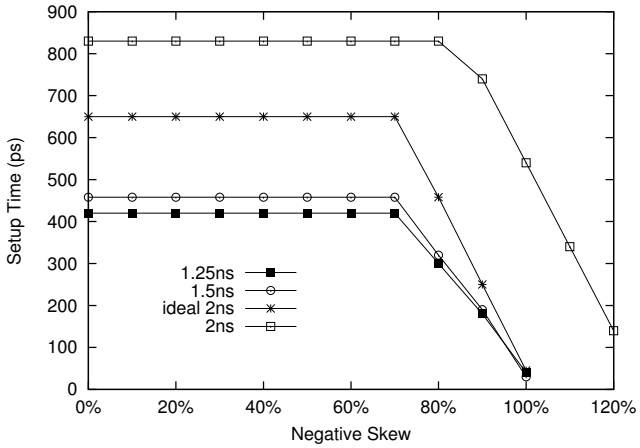


Figure 3.17: Setup time as a function of negative skew.

3.9.3 Target frequency

We now extend the above results to the case where the same RTL design (a 2x2 switch) is synthesized for a higher and lower target frequency and observe implications on the timing margins of a tightly coupled mesochronous NoC architecture. Figure 3.17 shows setup time as a function of negative skew for different target cycle times. The skew is expressed as percentage of the cycle

time, and the assumption behind this plot is that as we relax the cycle time also the maximum skew constraint can be proportionally relaxed.

If we assume that by relaxing the target speed all delays scale proportionally in the design, then we would expect that by relaxing the speed from 1.5ns to 2ns the setup time increases by 1.33x (see starred line). This is not the case, since the real setup time increases much more, as the figure shows. The reason for this is that the arbitration and switching logic inside the switch can be optimized for area as the timing constraint is relaxed only up to a certain point, beyond which no more netlist transformations are feasible. Therefore, the shifting of the *mux window* to the right, illustrated in Figure 3.14(b) for the tightly coupled architectures, scales ideally only up to a certain point, beyond which we observe a more-than-linear increase of the setup time. This is what happens for the 2ns target period.

Another deviation of the ideal curve from the real one regards the knees. In fact, although the target period increases from 1.25 to 2ns, a given skew percentage on the x-axis actually means a different absolute phase offset for the different cases. Therefore, the switching instant of synchronizer latch outputs should enter the *mux window* at the same percentage skew for all target periods (see starred line in Figure 3.17 for a 2ns target).

This is again not the case, indicating that a delay has not scaled proportionally to the clock period. The rationale for this is the time to generate the latch_enable signals in the synchronizer front-end. For a tight 1.25ns constraint, this netlist was already *non-critical*, therefore by relaxing the timing constraint its delay stays more or less the same. Therefore, the knee appears later for large cycle times, as the real curve for a 2ns target proves. The key take-away from this characterization is that by relaxing the target clock frequency for the same RTL design an improvement of the skew tolerance and of the timing margins can be generally achieved for the tightly coupled architecture. In addition, a larger cycle time provides the physical synthesis tool more margin to enforce the feasibility constraint of Formula 3.4.

3.9.4 Switch radix

A last degree of freedom that we explored is the switch radix. We assume that for the same given target frequency, the switch radix is increased from 2 to 5. The effect on the timing margins is similar to what happens when we move from a loosely coupled to a tightly coupled mesochronous architecture. In fact, an increase of arbitration and crossbar selection time takes place, which

results in a decrease of the setup time. Conversely, the knee occurs for the same amount of negative skew, since the modification concerns only the switch internal architecture, not the time at which latch outputs switch. Overall, by combining the two effects we have an additional reduction of the maximum (negative) skew tolerance, which is equal to the increase in delay of the combinational logic described above. In general, for high radix switches it has to be verified that the reduced setup time is still above the minimum value required by the technology library.

In contrast, synthesis constraints help relieve the above limitation. In fact, for both the 2x2 and the 5x5 switch, the synthesis tool tries to meet the same target cycle time and to exploit the available slack to save area. In practice, the syntheses of both the 2x2 and the 5x5 designs converge with almost no slack. Therefore, control logic with 5 and 2 inputs takes almost the same delay with a large difference in area. In this way, the setup time in the two cases is almost unaffected because of the netlist transformations performed by the synthesis tool in the area-performance plane. In our experiments, a 5x5 switch exhibits a setup time which is only 5% lower than the one in the 2x2 switch. For those NoC architectures where the above netlist optimizations are ineffective or for very high radix switches, it is necessary to verify that the reduced setup time is still above the minimum value required by the technology library.

Another implication of the switch radix concerns timing closure of the *hybrid* synchronization architecture. As already noted while commenting Figure 3.8, the critical path starts in the switch arbiter (which generates the *stall* signal) and includes the propagation of the *stall* signal to the upstream switch. As the switch radix increases, the delay for *stall* generation by the arbiter increases, and might make this timing path critical for the entire NoC. This timing path is compared with those of the tightly and loosely coupled architectures in the following section.

3.10 Summary

The motivation behind the adoption of a GALS design paradigm has been presented in this chapter. The fully synchronous switch building block of our NoC has been described and its flexible mesochronous counterpart presented. In particular, to build such mesochronous alternative, a baseline synchronizer interface has been proposed and all the improvements that led to our loose coupled mesochronous synchronizer described. Furthermore, it has been shown that the switch input logic can be utilized for buffering, synchronization and

flow control thus coming up with a novel tight integrated mesochronous interface. A theoretical analysis characterizing such interfaces has been detailed and such considerations led to a further architectural improvement: the hybrid synchronizer, which significantly improves upon the previously described architectures. Post-layout results have been presented demonstrating the benefits of our novel approach from a performance, area and power consumption viewpoint. A design space exploration of the mesochronous link has been provided highlighting several interesting design points.

4

A Design Flow for GALS NoCs

THE previous chapter detailed the architecture of the interfaces required to enable the building process of an actual GALS switch block. This chapter moves a step forward towards the system-level. In fact, the design flow described in this chapter will enable to build an entire GALS system from the system specification, through synthesis and CTS thus reaching the layout level by performing place&route. In particular, a complete Network-on-Chip design flow will be presented in its front- and back-end part. For both parts, our contribution to make the design flow suitable for building GALS systems will be discussed.

4.1 The Front-end

The design flow adopted in this thesis is depicted in Figure 4.1 and it is based on that developed in [85]. As for many typical design flows leveraging industrial toolchain, it can be split in two main branches: front-end and back-end. In the front-end part, the user has to specify the network characteristics that might be hand written or automatically generated by a tool, e.g. Sunfloor [151]. Once the network topology specification has been created, it can be fed to the ×pipescompiler [152] for the automatic generation of the topology instance. The tasks performed by ×pipescompiler involve:

- Performing checks on the input file, verifying the full connectivity of all the system components. (This step is key since the input topology description does not necessarily come from SunFloor; it can also be manually written).
- Configuring the blocks of the ×pipes component library according to the specifications in its input description.

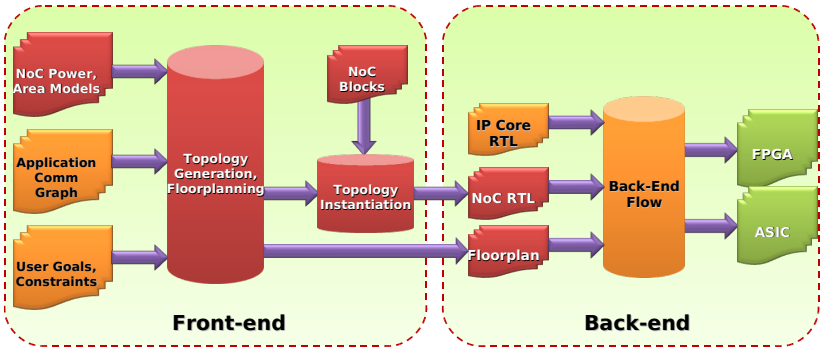


Figure 4.1: The Network-on-Chip Design Flow.

- Creating top-level modules to connect all the blocks together, according to the desired topology.
- Producing suitable routing tables for the NIs, based on the specified communication flows and routes.
- Creating testbenches for the whole topology, capable of stressing all the paths among communicating IP cores in the topology.
- Generating component lists to be imported in the physical implementation scripts.

×pipesCompiler generates code at the RTL level, both in SystemC and Verilog. This code is suitable for simulation, for FPGA emulation and for ASIC implementation flows.

4.1.1 GALS enhancement: the ×pipes compiler

The ×pipes compiler has been augmented with the possibility to partition the system in different frequency islands as well as the network-on-chip itself in different mesochronous partitions as depicted in Figure 3.1.

This new feature required the modification of the specification file format, the addition of the new GALS interfaces to the ×pipes library (see "NoC blocks" in Figure 4.1) and obviously the modification of the compiler code itself. An example of the topology specification file supporting the new GALS features is reported in the Listing 4.1.

Listing 4.1: Example topology description file.

```
// In this topology: 1 processor,
// 1 memory and 2 switches

topology(2switch_1link);

// ------------------------------------------------------
// Definition of the clock domain and subdomain
// A subdomain is a domain that have the same frequency
// of its domain but different phase ( mesochronous )
// ------------------------------------------------------

//NEW FEATURE OF THE COMPILER
domain(clk_1, Domain:2);

// -------------------------------------
// define the cores characteristics and
// the clock domain it belongs to
// -------------------------------------

//NEW FEATURE OF THE COMPILER
core(core_0, switch_0, 1_1, 2, 6, userdef, initiator);
core(shm_1, switch_1, 1_2, 2, 6, userdef, target:0x00);

// ---------------------------------------------
// define the switch characteristics
// and the clock domain/subdomain it belongs to
// ---------------------------------------------

//NEW FEATURE OF THE COMPILER
switch(switch_0, 3, 3, 6, 1_1);
switch(switch_1, 4, 4, 6, 1_2);

// --------------------------------
// define the links here
// link number, source, destination
// --------------------------------

link(link0, switch_0, switch_1);
link(link1, switch_1, switch_0);

// ----------------------------------
// define the routes from source core
// to destination core
// ----------------------------------

route(core_0, shm_1,   switches:0,1);
route(shm_1, core_0,   switches:1,0);
```

As pointed out by the above example, the new GALS features of the ×pipesCompiler allow to associate a core and/or a switch (at a block level selection) to a particular frequency domain (thus calling for the automatic instance of a dual-clock FIFO synchronizer). Furthermore, within the same clock domain, it is possible to specify a sub-domain of operation thus effectively resembling the mesochronous scenario, i.e., same frequency but different phase offset. Once the compiler performed all the previously described steps, GALS interfaces are instantiated if required. The ultimate result is the generation of a SystemC and Verilog netlist at RTL level which describe a GALS system leveraging mesochronous and dual-clock FIFOs interfaces that can be fed to the design automation toolchain of the back-end synthesis flow.

4.2 The Back-end

Due to the quick pace of lithographic miniaturization, it is nowadays well known that a number of physical-level process issues related to deep submicron fabrication (such as wire delays and leakage power) are affecting designs. Understanding these issues is clearly key to tackling them, for example by compensating them at the architectural level.

In the case of NoCs, the relationship among back-end flows and architectural design is even stricter, because of several factors:

- One of the main purposes of NoCs is exactly to help in tackling wire-related physical-level issues.
- NoCs are intended to be large structures, spread across a whole chip. As such, several design issues, such as clock tree distribution, wire delays and variability, play a key role in NoCs.
- NoC are also designed to interconnect a large number of heterogeneous components and devices, each of which could come as a prebuilt, pre-characterized IP macro. Therefore, it is key to be able to leverage standard back-end industrial toolchains for NoC design, otherwise the effort of developing customized infrastructure would be impossible to afford.

As already cited several times throughout this thesis, we focus on standard cell-based physical implementations only. While full custom design does certainly improve final results, it does also greatly decrease flexibility and increase design time.

4.2.1 A Traditional View of the Back-End Design Flow

A traditional back-end design flow based on standard cells is depicted in Figure 4.2. This kind of flow features a streamlined sequence of steps, which are ideally as much decoupled as possible. The entry point of the back-end design flow is the outcome of the front-end part, i.e., an RTL netlist generated by the GALS ×pipesCompiler.

- Starting from a description of the circuit in some RTL language, such as VHDL or Verilog, logic synthesis is initially performed; this translates RTL descriptions into a so-called gate-level netlist, i.e., a connected network of basic gates belonging to a technology library. The technology

library is an abstracted view of the underlying foundry process, and describes the basic gates in terms of function (such as boolean gate, flip-flop, etc., propagation delay, capacitive load, etc. Based on this information and on user constraints, a main task of the logic synthesis is to make sure that the netlist fulfills speed, area and power consumption goals.

- The gates of the netlist are subsequently placed, i.e., mapped onto a canvas representing the geometrical shape of the final device - this is typically a rectangle. Placement involves both a high-level arrangement of the main functional blocks of the chip (a step often called *floorplanning*) and a low-level arrangement of each single gate (*detailed placement*).
- Finally, the routing step takes care of laying metal lines to attach the placed gates to each other, so that the circuit can function. During this stage, some signals (typically, the power supply and the clock) play a special role, since they must be distributed to a large number of gates spread all over the chip. Special attention is paid, for example, to the minimization of the skew in the clock distribution network.

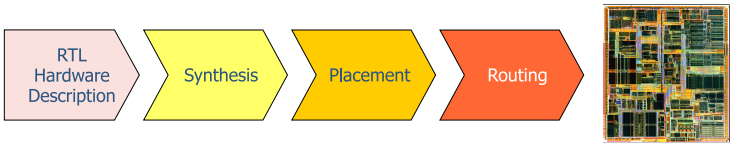


Figure 4.2: A schematic view of a traditional design flow.

Of course, in this reference flow, any constraint violation - such as the impossibility to route the wires to connect the gates of the placed netlist, or an unexpected violation of the required circuit speed - can only be tackled by feedback loops where one or more steps are repeated again, under different assumptions.

However, this basic flow is not sufficient any more to deal with today's technology, for reasons that will become more clear in the following. A crucial point of failure is that it becomes increasingly time-consuming, complex, and potentially even unfeasible, to maintain the strict separation among the steps of the traditional flow sequence; routing issues are nowadays setting an increasing amount of constraints on feasible placements, and this applies, in turn, to all the upstream steps of the flow. Therefore, the number of detected violations and of required feedback loops in physical implementation would become too large for the traditional flow paradigm to hold without changes. In response to

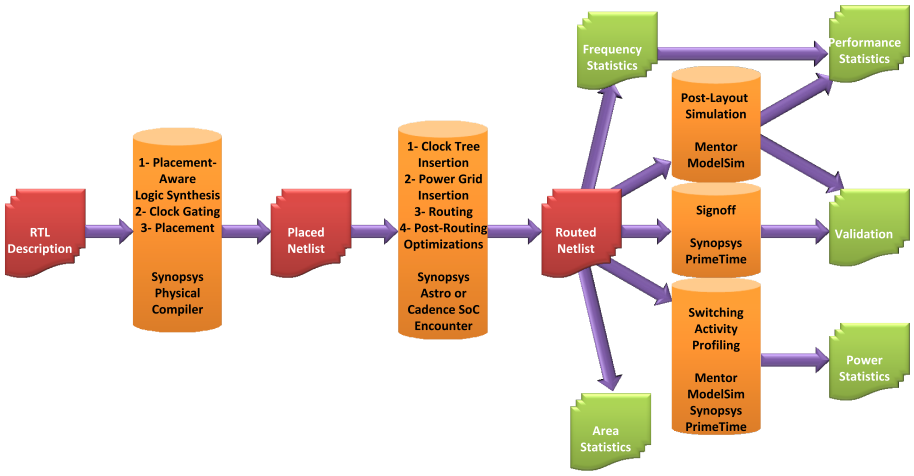


Figure 4.3: The synthesis flow for ×pipes.

this, new solutions must be found, either by actively tackling issues (and NoCs at large are in some sense doing this, e.g., by simplifying routing through an architectural breakthrough), or by simultaneously performing multiple steps at once, with wider constraint visibility.

In the following, we will present an outline of our backend flow, subsequently focusing our attention on specific portions of the flow which have particular relevance.

4.2.2 The ×pipes Back-End Infrastructure

In the proposed GALS NoC design and synthesis framework for ×pipes, we provide a complete back-end flow based on standard cell (Figure 4.3).

First, we perform logic synthesis by utilizing standard Synopsys tools; depending on the underlying technology library, this step may need be augmented with placement awareness, as will be discussed in Section 4.2.5. We support this procedure on 130nm, 90nm and 65nm technology libraries by partner foundries, tuned for different performance/power tradeoffs, with different threshold and supply voltages.

During synthesis, we can optionally instruct the tools to save power when buffers are inactive by applying clock gating to NoC blocks. The gating logic can be instantiated only for sequential cells that feature an *input enable* pin,

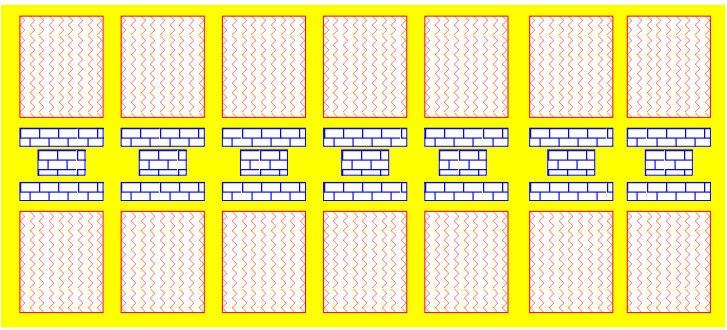


Figure 4.4: Example of the usage of fences in placement. Yellow area (solid pattern): floorplan; red areas (zigzag pattern): hard macros for IP cores; blue areas (brick pattern): soft macros for NoC components.

which are a large majority of the datapath flip-flops of ×pipes.

We subsequently perform the detailed placement&routing step within either Cadence SoC Encounter (SOCE). First, we feed SOCE with a coarse floorplan, generated either manually or by SunFloor. This floorplan contains *hard macros* and *soft macros*, separated by fences (Figure 4.4). The hard macros represent cores and memories, and are modeled as black boxes. Hard macros are defined with a LIBRARY EXCHANGE FORMAT (LEF) file and a Verilog Interface Logical Model (ILM), and obstruct an area of choice. These boxes also obstruct some of the metal layers laying directly above; the exact number of obstructed levels is configurable, depending on how many metal layers the cores are supposed to require and on whether over-the-cell routing should be allowed for the NoC wires *vs.* between-the-cell. Soft macros are also boxes; they enclose the modules of ×pipes, and the placement tool is allowed to operate within them as long as the fences are not trespassed. By constraining the placement tool to operate on a "tile" at a time, the solution space is dramatically pruned, and relatively fast runtimes can be achieved. For proper results, however, it becomes necessary to specify rough timing constraints at the soft macro boundaries; we achieve this by pre-characterization of the links.

4.2.3 GALS enhancement: Hierarchical Clock Tree Synthesis

After each network block has been placed in the layout, the clock tree must be inserted. We adopted a bottom-up clock tree insertion methodology in our GALS design flow. In this respect, we see mesochronous synchronization as a mean to relieve the burden of chip-wide clock tree distribution and this hierar-

chical clock tree synthesis methodology as an effective way of exploiting the mesochronous link features. Specifically, in a first phase each network switch is placed and routed in isolation with a given target frequency. The clock tree of each switch is then synthesized with a tight skew constraint of 5% of the target clock period. Once the local clock tree is characterized with its input delay, skew and input capacitance, a *macromodel* is built in order to be used in the next design step. Furthermore, in order to implement a hierarchical clock tree synthesis (see Figure 4.5), a buffer is inserted to the input clock pin of each switch block.

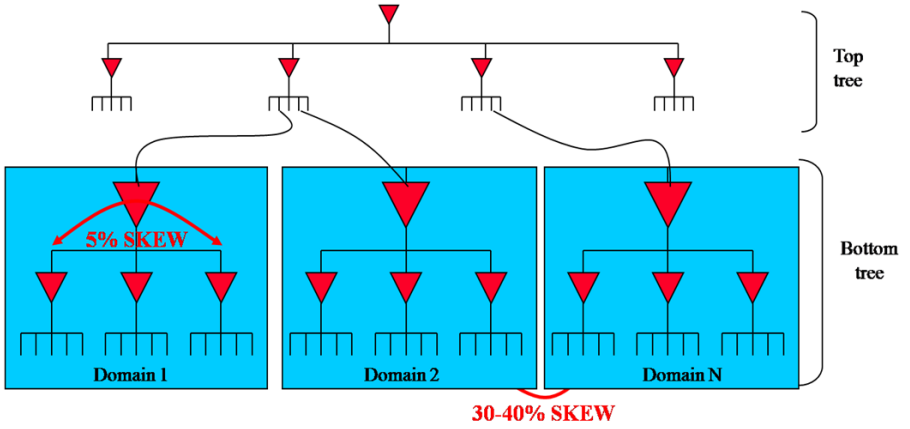


Figure 4.5: Hierarchical clock tree synthesis.

Once the switches have been placed and routed, they are imported as macro blocks in the main network design along with their libraries detailing both timing and physical characteristics. The next step consists of performing a top-level clock tree synthesis by leveraging the switch macromodels previously extracted. In fact, this model can be used to characterized the bottom clock tree given that these local clock trees will not be modified by the place&route tool. Therefore, in order to preserve the clock tree local to the switches, a `PreservePin` tag must be used in the CTS specification file.

Listing 4.2: Example topology description file.

```
#REQUIRED FOR HIERARCHICAL CTS
MacroModel port switch_2x2_6_0/clk 414.7ps 357.2ps 423ps 365.2ps 0fF
MacroModel port switch_2x2_6_1/clk 483.4ps 458ps 440.6ps 418.4ps 0fF

  # Sample Route Type Command
  RouteTypeName           CK1
  #NonDefaultRule         rule1
```

```
  PreferredExtraSpace     1
  TopPreferredLayer       4
  BottomPreferredLayer    3
  # Shielding             vdd gnd
  End

 AutoCTSRootPin    clk
  Period          2ns
  MaxDelay        1ns
  MinDelay        0ns
  SinkMaxTran     300ps
  BufMaxTran      500ps
  MaxSkew         100ps
  NoGating        rising
  MaxDepth        1024
  RouteType       CK1
  DetailReport    YES
  RouteClkNet     NO
  PostOpt         YES
  OptAddBuffer  YES
  Buffer     HS65_LL_BFX18 HS65_LL_BFX27 HS65_LL_BFX35
             HS65_LL_BFX44 HS65_LL_BFX53 HS65_LL_BFX71
  #REQUIRED FOR HIERARCHICAL CTS
  PreservePin
  + xpswitch_0/clk
  + xpswitch_1/clk
End
```

This information is used by the clock tree synthesis (CTS) tool to infer a top clock tree balancing the leaves with a much looser skew constraint (e.g., 30/40%). The ultimate result is a global clock tree which consumes less power with respect to the traditional one generated by enforcing chip-wide skew constraints. An example of a *clock tree synthesis specification file* is reported in the Listing 4.2.

At this point, the power supply nets are added. Two main schemes are available. Traditionally, *power rings* (metal lines carrying the power supply voltages) are laid around the die; as an alternative, a *power grid* can be laid across the chip in the topmost metal layers. The latter choice requires more metal resources, but minimizes *IR drops* (voltage drops and fluctuations due to resistive effects in the supply networks and to the current draw). Therefore, we choose power grids, so as to maximize voltage stability.

4.2.4 Routing

Next, the routing tool begins to route the logic wires. An initial heuristic mapping lays the wires; this initial solution is semi-random and almost certainly violates essential constraints, such as that of not shorting different wires. Therefore, SEARCH&REPAIR (SR) loops are executed to fix any violations, including those regarding excessive propagation delays.

Post-routing optimizations are then performed. This stage includes crosstalk minimization, antenna effect minimization, and insertion of filler cells. Fi-

nally, a *sign-off* procedure can be run by using Synopsys PrimeTime [86] to accurately validate the timing properties of the resulting design.

Post-layout verification and power estimation is achieved as follows. First, the netlist representing the final placed&routed topology, including accurate delay models, is simulated by injecting functional traffic through the OCP ports of the NIs. This simulation is aimed both at verifying the functionality of the placed fabric and at collecting a switching activity report. At this point, accurate wire capacitance and resistance information, as back-annotated from the placed&routed layout, is combined with the switching activity report using Synopsys PrimeTime [86]. The output is a layout-aware power/energy estimation of the simulation.

4.2.5 Placement-Aware Logic Synthesis

As mentioned above, the traditional flow for standard cell design features logic synthesis and placement as two clearly decoupled stages. In [93], authors show that this standard design flow achieves reasonable results for 130nm and 90nm NoC designs, but we have found the situation to be substantially different at the 65nm node.

The origin of the problem lies in the decoupling of the two steps. Synthesis and placement could be considered as independent when wire delays were negligible; this is unfortunately not the case anymore [89]. Since wire delays can be comparable to logic delays, if not larger, it is crucial to be able to estimate wire delays already during synthesis. Since wire delays depend directly on wire length, it is clear that placement algorithms are also unfortunately affecting the solution space of synthesis algorithms.

To alleviate the problem, *wireload models* have been introduced. Wireload models are pre-characterized equations, supplied within technology libraries, that attempt to predict the capacitive load that a gate will have to drive based on its fan-out and on the overall design area. Unfortunately, wireload models remain a statistical representation of the physical reality, and are therefore an inaccurate tool to predict delays on a single net basis, given that each net could exhibit a different behavior. In our 65nm tests, we experience unacceptable performance degradation due to either under- or over-estimations of wire loads. Even when synthesizing single NoC modules (i.e., even without considering long links), the logic synthesis tools generate a netlist with the expectation of some operating frequency; however, after placement, the actually reachable frequency is often up to 30% worse (and even lower after the routing phase).

Furthermore, sometimes placement and routing tools simply do not converge towards any solution at all, trying in vain to match the expectations set by the logic synthesis step.

To address this issue, NoC synthesis in 65nm requires *placement-aware* logic synthesis tools, such as Synopsys Physical Compiler [86]. Therefore, in the proposed NoC back-end flow, after a very quick initial logic synthesis based on wireload models, the tool internally attempts a coarse placement of the current netlist. Next, it iteratively optimizes the netlist and the placement, based on the actual wire loads implied by the current candidate placement. The outcome is a placed netlist that is optimized also accounting for wire delays.

We also observe in our study of NoC synthesis that other issues may arise when placing gates into soft macros. For example, in our test designs, placement tools perform poorly when modules have to be placed within fences which are either too small or too wide. While the former case is clearly understandable, we attribute the unexpected latter effect to the placement heuristics, which are probably performing worse when the solution space becomes very large. The problem must be solved by proper tuning of the spacing among the soft macro fences and, consequently, accurate area models of the NoC modules are required to avoid very time-consuming iterations.

4.3 Summary

In this chapter, the Network-on-Chip design flow extensions towards the GALS design style has been illustrated. In particular, this chapter described the enhancement required by this existing design flow in order to become suitable for the building process of GALS systems. From an implementation viewpoint, the ×pipescompiler has been augmented with GALS interfaces and its specification file format changed according to the new requirements. Furthermore, a hierarchical clock three synthesis strategy (CTS) has been implemented in order to fully exploit the mesochronous link characteristics. In fact, this methodology turns out to be key when designing a system in 65nm and beyond under a relaxed synchronization assumptions. Overall, the new design flow described in this chapter will be utilized in the next chapter for the comparison of two GALS systems: one implementing a fully synchronous NoC and another implementing a mesochronous one.

5

Contrasting Synchronous vs. Mesochronous Networks-on-Chip

THIS chapter performs a cross-benchmarking between two GALS systems. The first leverages a fully synchronous NoC while the second implements a mesochronous NoC. Both systems leverage dual-clock FIFO interfaces to provide frequency decoupling between the NoC and the computational units. While the former chapters focused on a switch-level analysis of such GALS systems, in the following, a network-level perspective will be taken. Furthermore, both systems will be compared from many viewpoints such as: clock tree power, area/wiring overhead, power consumption and variability robustness viewpoint.

5.1 Introduction

Networks-on-chip (NoCs) are proving capable of easing the communication bottleneck arising in multi-core computing platforms [56, 57, 62, 130], thus overcoming the fundamental performance, power and physical design limitations of shared and multi-layer buses. Furthermore, as we have discussed in the previous chapters, there is today little doubt on the fact that a high-performance and cost-effective NoC can only be designed in 45nm and beyond under a relaxed synchronization assumption [62, 118]. It has been discussed that one effective method to address this issue is through the use of globally asynchronous and locally synchronous (GALS) architectures, where the chip is partitioned into multiple independent voltage and frequency domains. Each domain is clocked synchronously while inter-domain communication is achieved through specific interconnect techniques and circuits [61].

There are several approaches to design GALS architectures, one consists of

using purely asynchronous clock-less handshaking for transferring data words across clock domains [63, 65]. There are a few chip demonstrator proving the viability of such solution such as [58–60], but they have not achieved widespread adoption of asynchronous NoCs in the industrial arena yet. In fact, asynchronous handshaking techniques are complex and use unconventional circuits (e.g., Muller C-elements) usually unavailable in industrial technology libraries. Moreover, asynchronous logic is not well supported by CAD tools. In this context, the most effective solution found so far for actual chip fabrication and industry-relevant designs consists of implementing routers and GALS interfaces as hard macros using ad-hoc design styles [60]. However, the hard macro methodology is more a way of working around the lack of proper design and verification tools and thus guaranteeing performance during physical implementation rather than a way of optimizing area. In fact, area remains consistently larger than fully synchronous NoC counterparts ($1.8\times$ in [60]).

What is currently missing in the open literature, as well as in the industrial prototyping experience, is a mature GALS NoC architecture making use of source synchronous communication techniques. This method achieves high efficiency by obtaining an ideal throughput of one data word per source clock cycle with a design style which is more easily compatible with common standard cell design flows. In spite of a few early works taking this approach [55, 72], such GALS design style has not been consistently brought to maturity over time, thus reducing source synchronous communication to a nice concept with only a limited relevance for real-life designs. For instance, the area and latency overhead associated with the use of synchronizers has never been tackled in a systematic way, and even advanced industrial research prototypes, such as the Intel Polaris chip [130], live with such an overhead. As a consequence, source synchronous communication has never truly evolved from a concept to a mature technology.

The work presented in chapter 3 bridged this gap and developed communication interfaces for enabling a synchronizer-based GALS NoC technology. In particular, design techniques merging synchronizers with network building blocks (named the *tightly coupled design style*) have proved area, power and performance efficient with respect to loosely coupled solutions, where synchronizers are placed as external blocks to NoC switches. All previous work concerns architecture design space exploration and quality metrics assessment of synchronizer-based communications at the switch level.

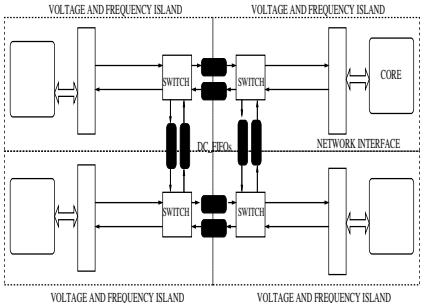
This chapter builds on these milestones and moves a step forward by taking the network-level perspective. While the migration from fully synchronous

parallel systems to GALS systems with voltage/frequency decoupling between IP cores is taken as a matter of fact in this chapter, there are significant GALS NoC architecture variants the designer can still choose from. The first one consists of placing NoC switches in the clock domains of the IP cores they are connected with. In contrast, an alternative solution consists of instantiating the on-chip network as an independent clock domain, disjoint from those of the IP cores. In this scenario, dual-clock FIFOs need to be instantiated only at the network boundary, since the network is synchronized by a single and independent clock signal. The homogeneous performance of NoC switches, the fewer amount of dual-clock FIFOs required and the possibility to have an always on system interconnect fabric make this solution more attractive for our work. However, the feasibility and efficiency of this solution is now mainly on burden of the physical designer. In fact, he has to deal with a large synchronous clock domain (the NoC itself) distributed throughout the entire chip.

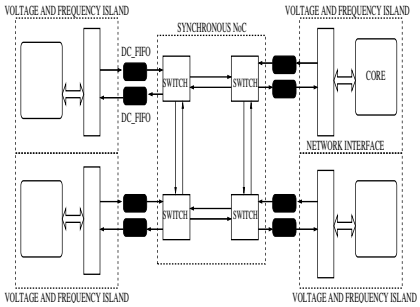
A workaround for this problem consists of designing the network as a set of mesochronous domains, instead of a global synchronous domain, yet retaining a globally synchronous perspective of the network itself. The granularity of a mesochronous domain can be as fine as a NoC switch, which is the case considered in this chapter. The communication between neighboring switches is then mesochronous as the top-level clock tree might not be equilibrated. This brings the additional advantage that mesochronous synchronizers are typically more lightweight than dual-clock FIFOs for use in switch-to-switch links.

The main contribution of this chapter is a comprehensive crossbenchmarking of a mesochronous NoC with a fully synchronous NoC to be used in a GALS system. Both networks share the same baseline MPSoC-oriented NoC architecture for the sake of fair comparison. The tightly coupled design principle is followed for mesochronous links, so that their unique optimization opportunities in the NoC domain are fully exploited. The chapter relies on actual implementations on a 65nm industrial technology library and provides the assessment of a wide range of design quality metrics, some of them of special interest for nanoscale silicon technologies: performance, area, power, robustness to process variations and clock tree synthesis efficiency.

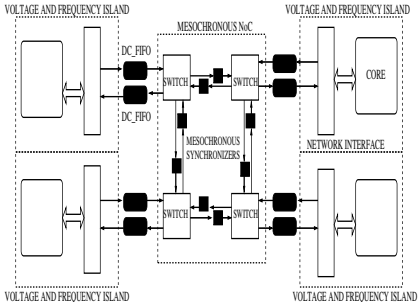
This chapter is structured as follows: the target GALS architecture is described in Section 5.2 whereas the synthesis strategy of the platforms under analysis is detailed in Section 5.3. Experimental results are presented in Section 5.4 while robustness to process variations is assessed in Section 5.5. Finally, Section 5.6 summarizes the contributions of this chapter.



(a) The NoC is split into multiple VFIs.



(b) The NoC is a global clock domain.



(c) Mesochronous NoC for GALS systems.

Figure 5.1: Paradigms for GALS synchronization.

5.2 Target GALS Architectures

A GALS-based design style fits nicely with the concept of voltage and frequency islands (VFIs), which has been introduced to achieve fine-grain system-level power management and is currently driving the transition of most MPSoCs to GALS systems. In these systems, if network components belong to the core's VFIs as in Figure 5.1(a), then performance of communication flows would be determined by the slowest domain crossed on the way to destination. Moreover, in case a VFI is shut down, global connectivity is jeopardized.

An alternative solution is illustrated in Figure 5.1(b), where the NoC lies in its independent VFI. This way, performance of the whole switching fabric would be homogeneous, with only boundary effects to take care of. Moreover, the network would be loosely coupled with the cores' VFIs, and each core/cluster of cores could be shutdown without any impact on global network connectivity.

The main issue with an independent NoC VFI lies in the feasibility of its clock

tree. The reverse scaling of interconnect delays and the growing role of process variations are some of the root causes for this. Even though inferring a global clock tree for the entire network will still be feasible for some time, it will probably come at a significant power cost. Moreover, it is unclear when the difficulty of tightly and globally enforcing the skew constraint, will truly become a roadblock.

However, a workaround for this problem does exist, as illustrated in Figure 5.1(c). The network could be inferred as a collection of mesochronous domains, instead of a global synchronous domain, yet retaining a globally synchronous perspective of the network itself. There are several methods to do this. One simple way is to go through a hierarchical clock tree synthesis process. In practice, a local clock tree is synthesized for each mesochronous domain, where the skew constraint is enforced to be as tight as in traditional synchronous designs. Then, a top-level clock tree is synthesized, connecting the leaf trees with the centralized clock source, with a very loose clock skew constraint. This way, many repeaters and buffers, which are used to keep signals in phase, can be removed, reducing power and thermal dissipation of the top-level clock tree. The granularity of a mesochronous domain can be as fine as a NoC switch block.

The communication between neighboring switches is then mesochronous as the clock tree is not equilibrated, while the communications between switch and IP cores are fully asynchronous because they belong to different clock domains. Bi-synchronous FIFOs are therefore used to connect the network switches to the network interfaces of the cores, as showed in Figure 5.1(c).

This synchronization paradigm comes with additional advantages. First, it makes a conscious use of area/power-hungry dual-clock FIFOs, which end up being instantiated only at network boundaries. Instead, more compact mesochronous synchronizers are used inside the network, thus minimizing the cost for GALS technology.

5.2.1 The dual-clock FIFO overhead

When comparing the schemes in Figure 5.1(a) and Figure 5.1(c), it should be observed that mesochronous NoCs are the reference solution for ultra-low cost synchronizer-based GALS systems. In fact, they make a conscious use of area/power-hungry dual-clock FIFOs, which end up being instantiated only at network boundary. Instead, more compact mesochronous synchronizers are used inside the network, thus minimizing the area and latency overhead.

In order to give a more quantitative estimation of this, let us compare four NoC switch variants engineered to support different synchronization schemes. The baseline switch is from the ×pipesLite architecture [92]. Its input stages have been augmented at first with a mesochronous 3-slot buffer interface and able to tolerate 100% positive and negative skew, and then with a dual-clock FIFO from [2]. This latter needs a 5-slot buffer in order to support frequency decoupling between a transmitter and a receiver while retaining the capability of full throughput operation.

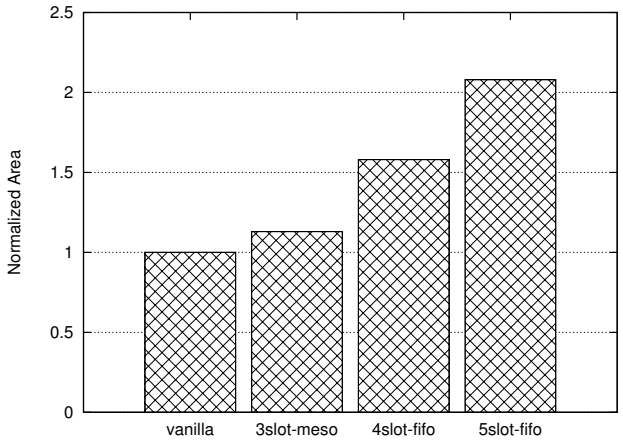


Figure 5.2: Switch area occupancy with different synchronization interfaces.

However, [2] also proves that a specialized 4-slot dual-clock FIFO architecture suffices for operation in a mesochronous environment. Both the baseline dual-clock FIFO architecture and the specialized one have been considered in the area comparison illustrated in Figure 5.2, reflecting synthesis results on 65nm STMicroelectronics technology. Even the specialized dual-clock FIFO is almost 40% more area expensive than the mesochronous solution. Obviously, the reason stems from the fact that the dual-clock FIFO has been natively conceived for a multi-frequency application domain whereas the mesochronous interface is designed ad-hoc for the scenario under investigation (same clock frequency, unknown phase offset) and requires less control logic.

It should also be observed that in future technology nodes the complexity of dual-clock FIFOs is expected to further increase. In fact, based on [64] the resolution time constant of synchronizers does not scale proportionally to the gate delay FO4. In contrast, experimental measurements have proven its degradation. The ultimate implication is that brute-force synchronizers, that are often

employed as a building block of dual-clock FIFOs, will require more cascaded flip flops to resolve metastability in future technology nodes, hence degrading area and synchronization latency. In order to preserve full throughput operation, also the number of buffer slots will need to be increased, thus further increasing the area overhead of the dual-clock FIFO.

Finally, the comparison between the synchronization latency of mesochronous vs dual-clock FIFO synchronizers is cleary in favour of mesochronous ones (see [2] for details), thus making the point for a conscious use of dual-clock FIFOs in GALS NoCs.

The above area and latency considerations suggest that the solutions to be considered for ultra-low cost GALS NoCs are those in Figure 5.1(b) and Figure 5.1(c). In the remainder of this chapter, those architectures will be compared with layout accuracy on the same 65nm STMicroelectronics technology.

5.3 Synthesis of GALS Platforms

Both the synchronous and the mesochronous platforms have been designed to be seamlessly integrated in an industrial design flow using commercial tools for physical synthesis without requiring full custom components.

The reference topology of our experiment is a 4x4 mesh network where each switch is connected to either a core or a memory (of size 1.5mm). As far as the physical synthesis is concerned, the same bottom-up methodology has been utilized for both platforms. Specifically, each network switch has been placed and routed in isolation with a target frequency of 500MHz. The clock tree of each switch has been synthesized with a tight skew constraint of 5% of the target clock period. Once the local clock tree is characterized with its input delay, skew and input capacitance, a *macromodel* is built in order to be used in the next design step. Furthermore, in order to implement a hierarchical clock tree synthesis, a buffer has been inserted to the input clock pin of each switch block. Once the switches have been placed and routed, they are imported as macro blocks in the main network design along with their libraries detailing both timing and physical characteristics. The next step consists of performing a top-level clock tree synthesis by leveraging the switch macromodels previously extracted. In fact, this model can be used to characterized the bottom clock tree given that these local clock trees will not be modified by the place&route tool. Therefore, in order to preserve the clock tree local to the switches, a `PreservePin` tag must be used in the CTS specification file.

Of note, the hierarchical CTS has been used both for the synchronous and the mesochronous platforms, since this is a standard methodology for parallel hardware platforms. *The only difference is the skew constraint in the top level clock tree, which can be loosened for the mesochronous design while should be tightly enforced for the synchronous one.*

Final step of our hierarchical methodology consists of routing the switch-to-switch links and performing parasitics extraction for accurate static-timing analysis and power estimation. Timing closure for both the synchronous and mesochronous NoC has been achieved at 500 MHz by performing exactly the same physical synthesis steps.

5.4 Experimental results

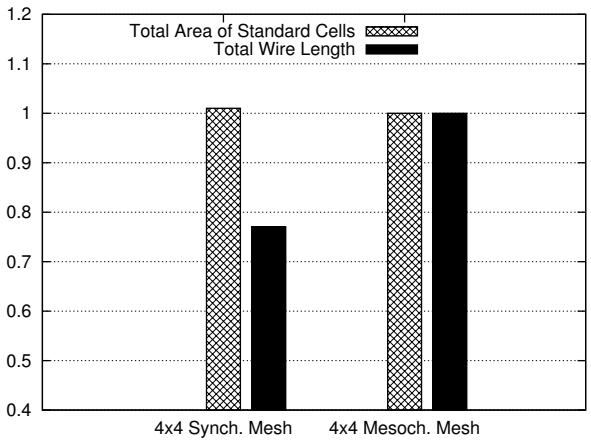


Figure 5.3: Area and wiring overhead for the mesochronous NoC.

5.4.1 Area and Wiring Overhead

Figure 5.3 reports post-place&route area and wiring statistics for the architectures under analysis. From an area viewpoint, both systems exhibit the same footprint. More in detail, the baseline architecture (i.e., the fully synchronous mesh) features a 2-slots input and 6-slots output buffers. On the other hand, its mesochronous counterpart has 3-slots input buffer and exactly the same amount of output buffering. Nonetheless, the area overhead is identical. This

is due to the fact that synchronization mechanisms, tightly coupled in the input buffer, are implemented using latch banks, which require a smaller area footprint compared to the flip-flops adopted in the input buffer of the baseline architecture. The ultimate result is an equal area occupation in both platforms.

From the wiring point of view, a 23% net saving is achieved by the fully synchronous platform. The reason lies in the fact that the mesochronous platform features an additional clock wire per output port utilized as strobe signal for data synchronization and a further external single bit synchronizer for backward flow control synchronization instantiated in each of the 48 switch–to–switch channels of the network. Last but not least, the slightly more complex network topology contributes to a more complex structure of the clock tree.

5.4.2 Power analysis

By leveraging post-layout netlists and back-annotated switching activity, we were able to achieve very accurate power figures with Synopsys PrimeTimePX. Figure 5.4 reports power consumption of both fully synchronous and mesochronous networks. Idle power plays in favor of the fully synchronous network. This is mainly due to the additional switch–to–switch clock wire used as strobe signal for data synchronization. This result calls for further evolution of mesochronous NoC technology, to implement a form of clock gating on these lines.

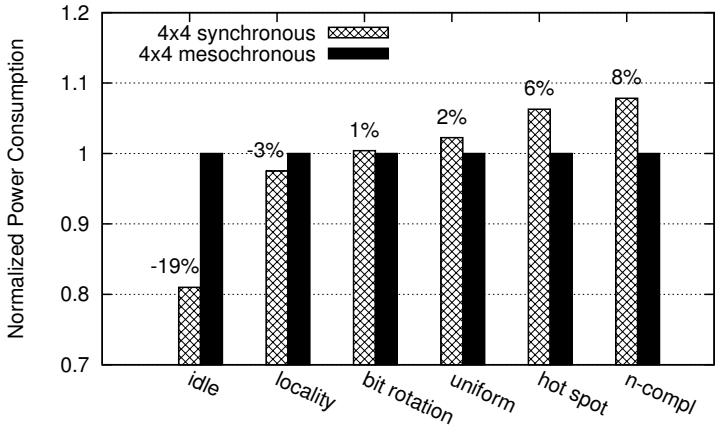


Figure 5.4: Power consumption in idleness and under various traffic patterns..

On the other hand, when stimulating the networks with several traffic patterns,

the mesochronous Network-on-Chip generally exhibits a smaller power consumption with respect to the fully synchronous one. The reason lies in the inherent architectural difference between the input buffer of the mesochronous switch and of the synchronous one. In this latter, both flip-flop banks are triggered at each clock cycle. Conversely, latch banks of the mesochronous input buffer are selectively triggered by an enable signal driven by a counter. The power gap may be bridged by clock gating also the synchronous NoC, however the key take-away is that the mesochronous NoC does not imply any significant dynamic power overhead.

A further demonstration of this comes from Figure 5.5, where power of the NoC architectures under test is presented when carrying the traffic of a full-HD video playback application for mobile devices. Extrapolating the usage scenarios of existing smart phones, one can imagine that in some years from now, larger resolutions and even full HDTV resolution may be expected. In this experiment, communication requirements of an Infineon video playback application have been scaled up to the high-end HDTV resolution (1920x1080 pixel) with 60 frames/second in true colour. The application is partitioned into 16 cores and mapped onto a 4x4 2D mesh.

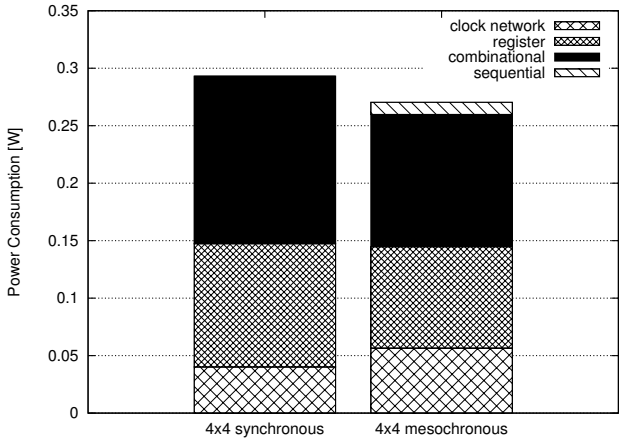


Figure 5.5: Power consumption of an industrial Full-HD Video playback application.

Results in Figure 5.5 show that even the gating-unoptimized mesochronous NoC is slightly more power efficient, proving that under realistic traffic patterns the stand-by power overhead is mitigated and there is no overhead with respect to a synchronous NoC. In the breakdown, the largest contribution of the clock network in the mesochronous NoC accounts for the strobe signals

of source-synchronous links. Overall, this does not increase total power because the lower power of latches and the lower contribution of combinational logic offset this effect. Interestingly, crossbar area (and power) is more relaxed in the mesochronous NoC since latches in the input buffer can perform slack borrowing from the link.

With a mesochronous NoC, an interesting opportunity pointed by [50, 53, 73] is to exploit hierarchical clock tree synthesis to reduce power of the top level clock tree. The tuning knob to materialize power savings is the relaxation of the skew constraint, so that less buffers are instantiated in the top-level tree. However, the effectiveness of this technique has never been validated nor quantified on an actual NoC setting with a parametric sweep of the skew constraint. first time, we experimented this on the 4x4 mesochronous mesh. We experimented this on the 4x4 mesochronous mesh by incrementally relaxing the skew constraint. Given the relatively small system size, we constrained the top level tree to be placed and routed outside IP core area, which captures the challenging requirements of many real-life MPSoC designs

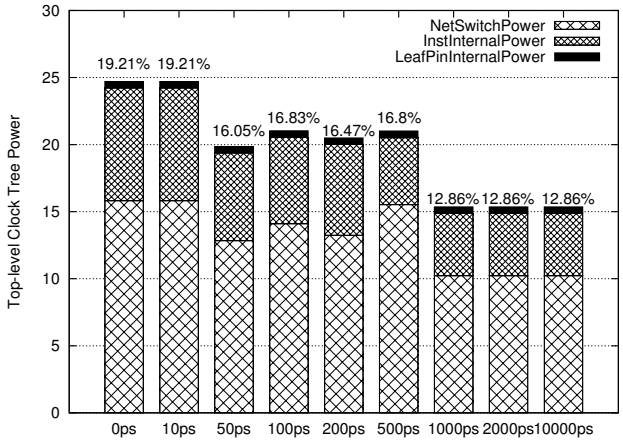


Figure 5.6: Power of the top level clock tree as a function of required skew.

Power of the top-level clock tree is reported in Figure 5.6 as a function of the required clock skew, ranging from 0 to 10000 ps. Transition time constraints are set to be very tight for the CTS tool (Cadence SoCEncounter). The percentage on top of the bars indicates the impact of the top level tree on the total clock tree power. We can see that power of the top level tree can be decreased by up to 40%, from roughly 25mW to 15mW. Also, the impact of the top level tree on total clock tree power can be as large as 20%.

Figure 5.6 may be misleading. In fact, the required skew does not keep up with the actually enforced skew by the CTS tool. When 2000 ps or more were required, the achieved clock skew saturates at about 600ps. Power savings could be even more than those reported if only the CTS tool was able to infer larger skews while saving clock tree area. Unfortunately, the CTS has not been natively conceived for this, but rather for the opposite: minimizing the skew.

Required Skew	Actual Skew	Top tree Power	% of Total clock tree
10ps	320ps	64.817mW	13.93%
1000ps	973ps	61.307mW	13.17%

Table 5.1: Top Clock Tree Power for a 64 cores system.

There is another effect that becomes apparent when the same experiment is performed on an 8x8 2D mesochronous mesh with 64 cores. Results are reported in Table 5.1. The CTS tool was not able to meet the required upper bound on the skew of the top level clock tree. When 10 ps were required, the actual skew resulted 320 ps. The clear message here is that as the system size becomes large and (not showed here) feature sizes shrink, it will become impossible to meet the desired skew constraint in the top level clock tree. This calls for a skew absorbing mechanism in the NoC architecture.

5.5 Variability robustness

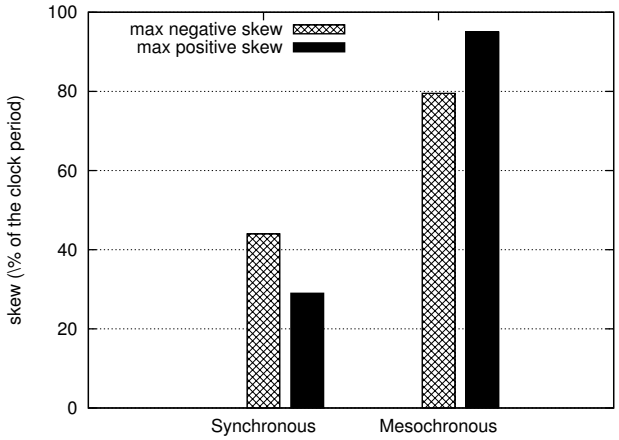


Figure 5.7: Skew tolerance with slack (enforced by the physical synthesis tool).

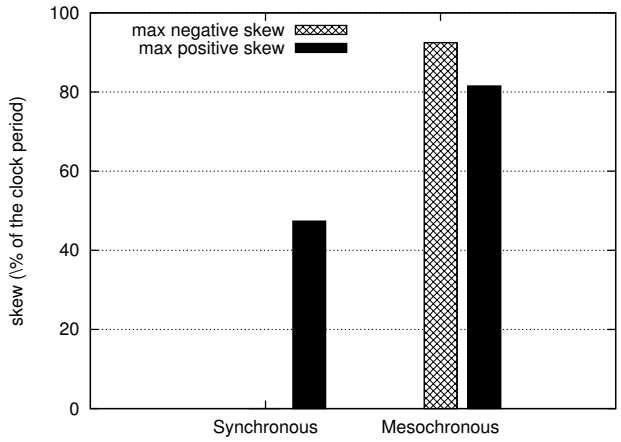


Figure 5.8: Skew tolerance with no slack.

In order to prove robustness to timing uncertainties of the platforms under test, we performed the following experiment. We synthesized, placed and routed two synchronous vs two mesochronous 5x5 switches placed 1.5mm far apart, like in the global platforms. Each switch has its own input pin for the bottom level local clock tree. We manually injected positive vs. negative skew between these two clock pins, thus analyzing the skew tolerance of the two designs to uncertainties in the top-level clock tree. The motivation lies in the fact that the two architectures differ only in the switch interfaces, therefore variability effects affecting internal switch gates and/or nets are likely to have the same impact on the designs under test. Validation of this is left for future work. So, let us focus on uncertainties affecting the top-level tree.

The skew tolerance of the two designs is reported in Figure 5.7. The synchronous design has some margins both for positive and negative skews, before violating the hold time and the setup time constraints respectively. However, the margins of the mesochronous design are much larger and approach 90% of the clock period. The positive skew tolerance is larger since the interconnect delay consumes part of the negative skew margins of the mesochronous link.

The above experiment suggests that the designs have some slack with respect to the target speed of 500 MHz they were synthesized for. Therefore, we repeated the experiment running the synthesized designs at their maximum operating speed, and evaluating skew tolerance in that operating point. In practice, slack was nullified and the actual skew tolerance of the designs was calculated.

The new results are reported in Figure 5.8. Now, the synchronous design has

clearly null negative skew, since the negative skew directly impacts the critical path. This latter goes from the output buffer of the upstream switch to the input buffer of the downstream switch going through the switch-to-switch link. A certain tolerance to the positive skew is however always there. In contrast, the mesochronous design features an excellent robustness to both positive and negative skew, meaning that such robustness is intrinsic of the architecture. The reason is that as the skew becomes increasingly relevant, a different critical path with respect to the one constraining the operating speed shows up and causes failure. The skew does not directly affect the frequency-limiting path delay, but makes a non-critical path become critical. The zero-skew difference between the delay of the two paths is the inherent skew tolerance of the mesochronous design.

The indications of these experiments have a number of implications:
- for a given target frequency of the NoC (dictated by system considerations), a mesochronous NoC can guarantee timing closure at that speed even though the CTS tool is not able to constrain the skew of the top-level clock tree (like in the 64-core system). In contrast, the synchronous NoC is directly exposed to such top level clock tree skew.
- for a given target frequency, the mesochronous NoC will certainly prove more robust to process variations affecting the top-level clock tree. In fact, the ultimate effect of variability will be an unexpected skew deviation with respect to the CTS tool statistics, a deviation that the mesochronous NoC can better absorb compared to the synchronous counterpart.

The above findings on process variation tolerance are now validated by means of a real injection experiment of process variations in the top level clock tree of a 4x4 2D mesh. Recently, a methodology for characterizing variability in NoC links was proposed [6]. The detailed variability model used in that work was later extended to NoC routers in order to analyze how process variation simultaneously affects both components of the network [33]. This new model takes into account, at the same time, systematic and random front-end variations due to, respectively, defects in the photolithographic process and deviations in the threshold voltage due to Random Dopant Fluctuations (RDF). The main use of this model is to generate many instances of a given chip and statistically analyze how process variation affects that particular design.

In our work, this model has been enhanced to consider back-end variability due to resistance variations introduced by the chemical metal planarization process (CMP) in wire dimensions. Concretely, the effect of metal thickness variations

in wire resistance has been introduced. According to the predictions of the ITRS, expected variations of the metal thickness will be lower than 10% (3σ). Note that in agreement with other studies, the effect of capacitance variations in NoC links is not considered [6] [34]. For those wires of a link routed in the same layer the same variation to the metal thickness has been applied. This behavior has been considered to satisfy the strong spatial correlation present in the variations introduced by the CMP process.

Figure 5.9 shows the probability density function (pdf) of the maximum achievable frequency of both synchronous and mesochronous designs when variations are injected to the nominal design. The variations injected are both systematic and random. Concretely, the variability sources considered are: transistors channel length ($3\sigma_{Leff}$ = 12%), threshold voltage $3\sigma_{V_{th}}$ = 33%, 58%, and the metal thickness $3\sigma_t$ = 10%. These values reflect expected variations for a 45nm technology node [89].

Results confirm that for the synchronous design the variability injected in the clock tree has a considerable impact on the maximum achievable frequency. Concretely, we have measured a standard deviation of the maximum frequency

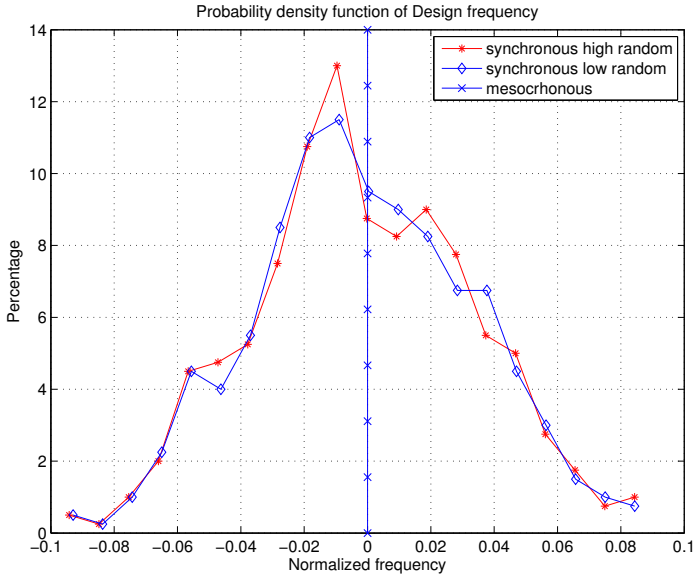


Figure 5.9: Variability-induced percentage deviations of maximum speeds of the designs under test with respect to nominal ones.

equal to 6.8% and 6.2% for the cases of low and high random variations, respectively. Note that differences in the pdf for high and low random variation are minimum. This can be explained by the fact that random variations do not have a significant impact in the top level clock tree, thanks in part to the large size of the clock buffers. On the contrary, the mesochronous design is clearly able to absorb the variations introduced in the clock tree, since it is able to preserve the nominal frequency in all cases. Obviously, since the clock skew directly impacts the critical path of the synchronous design, it is sometimes possible that this latter works at a higher speed of the mesochronous one (depending on the sign of the skew), however this argument is not able to counter the conclusion: the mesochronous NoC proves more robust to process variations in the top-level clock tree.

5.5.1 Performance considerations

As we have seen in Chapter 3, the implemented hybrid coupling mesochronous synchronizer is perfectly integrated with the existing switch architecture. The ultimate purpose of such new input buffer is threefold: buffering, flow control and data synchronization. Therefore, comparing the fully synchronous and the new mesochronous switch, the only notable difference resides in the input buffer architecture. From a latency viewpoint, let us first analyze what happens in a typical synchronous scenario. In the xpipesLite architecture, the fully synchronous switch is single cycle. Therefore, a switch–to–switch data transmission requires 1 clock cycle to cross the link channel and a further clock cycle for the switch traversal. In order to cover the same path, the GALS switch requires from 1 to 3 clock cycles depending on the skew condition (that could be either negative or positive). As we already proved in Section 5.5, both systems are able to tolerate a certain amount of skew. Nonetheless, when the amount of skew is such that the mesochronous switch spends an additional clock cycle for data synchronization, the fully synchronous switch has already failed. The ultimate conclusion is that when both systems are able to work with the same amount of skew, the average cycle latency is exactly the same. When the skew is such that a further cycle has to be devoted to data synchronization, only the mesochronous architecture can keep working without any failure in that operating condition.

5.6 Summary

By capitalizing on mature mesochronous technology, this chapter compares a mesochronous NoC and a fully synchronous NoC for use in a synchronizer-based GALS system. Both NoCs have dual-clock FIFOs at the boundary for frequency decoupling with IP cores. The lesson learned from this experimental work can be summarized as follows: 1) A fully synchronous NoC can be evolved to a mesochronous NoC with no area, latency and dynamic power penalties because of the hybrid coupling design style. In contrast, a 20% higher standby power is incurred because of the transmitted and continuously switching clock signals in source synchronous links. This calls for future work on clock gating techniques for such links. 2) Power savings in mesochronous NoCs can be actually achieved by means of hierarchical clock tree synthesis, although they are not significant yet, for a number of concurrent reasons. First, there is a gap between required maximum skew and the obtained one, since the CTS tool has been conceived for minimizing skew, and not for increasing it. Therefore, to take full advantage of this effect, CTS tools should be customized accordingly. On the other hand, when tight skew constraints are required under challenging physical and timing constraints, the CTS tool is not able to meet the target and therefore clock tree power does not increase a lot. In practice, this means that with current CAD tools it will become rapidly impossible to enforce tight skew constraints in the top level clock tree. Under these operating conditions, it is important to have an underlying architecture with inherent skew robustness. In our experiments, mesochronous NoCs prove capable of meeting this requirement. 3) As technology keeps scaling to the nanoscale era, process variations become increasingly important. In this chapter, we assess their effects when affecting the top level clock tree, thus experimenting the variability robustness of switch interfaces. The mesochronous NoC exhibits an inherent robustness to such delay uncertainties, since the critical path that constraints the operating speed is not directly impacted by the skew of the top level clock tree.

6

Layout-Aware Exploration of 16-tile systems

THIS chapter explores the performance and physical feasibility of 16-tile Networks-on-Chip within several topology configurations. It is the first chapter where our "system- to layout-level" approach for assessing NoC topologies is presented. In fact, our analysis framework encompasses different levels of abstraction as physical key parameters from synthesis and place&route process are first calculated and then exposed to our system level simulation infrastructure thus materializing in a system-level performance analysis with realistic layout-awareness.

The first part of this chapter presents our topology exploration framework along with the general backend synthesis methodology that has been adopted to carry out post-layout analysis of all the investigated topologies. Next, two important topology families (i.e., multi-dimensional and multi-stage topologies) have been analyzed in detail with the aforementioned design flow showing their sensitivity to physical effects and layout constraints.

6.1 Introduction

The execution of many multimedia and signal processing kernels has been historically accelerated by means of specialized processing engines [122]. With the advent of multi-processor system-on-chip (MPSoC) technology, performance of hardware accelerators is becoming accessible by combining multiple programmable processor tiles within a multicore system [123]. In addition, the performance of latest application specific integrated processors (ASIPs) [126] together with the high availability of transistors is making the design of custom hard-wired logic always less convenient. In fact, several time-consuming

design iterations are typically required before converging to a final layout thus increasing the overall time-to-market of the ultimate product. In general, the underlying principle is that efficient computation can be achieved while only marginally impacting programmability and/or configurability, and architectures can be devised that address the computation requirements of an entire application domain [125]. In this context, tile-based architectures provide parallelism through the replication of many identical blocks placed each in a tile of a regular array fabric [75, 76, 125]. This allows to speed-up the design process thus enabling a faster output of the final product thus bridging what is also known as the *productivity gap*. This approach makes performance scalability more a matter of instantiation and connectivity capability rather than architectural complexity.

Perhaps the most daunting challenge to make MPSoC technology mainstream is to realize the enormous bandwidth capacities and stringent latency requirements when interconnecting a large number of processing cores. The global intrachip communication infrastructure is responsible for tackling such problem. Moreover, Networks-on-chip (NoCs) are generally believed to be the long term solution to the communication scalability issue [1].

Topology selection is a NoC design issue which needs to be addressed in the early design stages and which has deep implications both on final system performance and on physical network feasibility. Therefore, drawing the network connectivity pattern is not just a pencil-and-paper exercise, but needs proper insights into nanoscale physics and into the synthesis backend. NoC architectures can be designed with both regular and custom topologies. The primary advantages of a regular NoC architecture are topology reuse, reduced design time, ease of routing, better control of electrical parameters and hence less design respins and a higher degree of performance predictability which facilitates the early design stages. The 2D mesh is by far the most popular regular topology used for on-chip networks in tile-based architectures, because it perfectly matches the 2D silicon surface. Unfortunately, 2D meshes show very poor scalability properties in terms of diameter, average minimal hop count and bisection bandwidth.

In contrast, topologies with more than 2 dimensions are attractive for a number of reasons. First, increasing the number of dimensions in a mesh results in higher bandwidth and reduced latency. Second, the number of dimensions can be traded-off with the number of cores per switch, thus giving rise to concentrated topologies saving network components and trading bandwidth for latency. Third, wiring on a chip comes at a lower cost with respect to off-

chip interconnections. However, wiring is also the challenging aspect of these topologies, since their mapping on a bidimensional plane involves the existence of wires with different lengths. From a layout viewpoint, this translates into links with different latencies and into the use of more metal layers.

The objective of the first part of this chapter is to explore the performance and physical feasibility of multi-dimensional topologies as an optimized alternative to traditional 2D meshes in the context of a 65 nm CMOS technology node. Pursuing our goals involves an analysis framework encompassing different levels of abstraction. On one hand, we developed and leverage a backend synthesis flow based on commercial tools to shed light on physical implementation trade-offs of multi-dimensional topologies. Moreover, we expose key physical parameters (link latency, maximum operating frequency) to system level simulation, thus providing silicon-aware performance figures. Our system-level simulation tool capitalizes on the concept of transaction level modeling, thus combining accuracy with simulation speed.

This tool also allows us to assess network performance with far more accurate traffic patterns than traditional synthetic ones. In particular, we model injected network traffic with "Open Core Protocol" (OCP) transaction accuracy, and assess how recently proposed communication middleware for MPSoCs impacts network traffic and the actual requirements it poses on network topologies.

As technology moves to the nanometer era, topology analysis and exploration needs to be performed with novel methodologies and tools that account for the effects of nanoscale physics, largely impacting final performance and even feasibility of many NoC topologies. A general guideline driving network-on-chip (NoC) design under severe technology constraints consists of silicon-aware decision-making at each hierarchical level [149]. This is likely to result in less design re-spins and in faster timing closure. In this direction, new tools are emerging that guide designers towards a subset of most suitable candidates for on-chip network designs while considering the complex tradeoffs between applications, architectures and technologies [153, 154].

The remainder of this chapter is organized as follows. Section 6.2 presents our topology exploration framework whereas Section 6.3 details the backend synthesis flow that will be utilized throughout this chapter. Next, Section 6.4 presents the analysis of the first topology family (k-ary n-mesh) leveraging our evaluation framework along with the achieved results. Last contribution of this chapter is described in Section 6.5 where k-ary n-tree topologies have been analyzed and ad-hoc floorplanning for their layout has been proposed. Section 6.6 concludes and summarizes the main contributions of this chapter.

6.2 Topology exploration framework

Our realistic topology exploration framework utilizes the the ×pipesLite NoC architecture [92]. The switching fabric implements a 2-cycle-latency (one for switch operation and one for traversing the output link), output-queued wormhole-switched router supporting round-robin arbitration on each output port. The implemented flow-control scheme is stall/go. The switch is parameterizable in the number of its inputs and outputs, its link width as well as in the size of the output buffering. For this work, 6-flit buffers are assumed and the link (and flit) width is set to 32 bits (see Chapter 3.4 for a complete discussion on the switch architecture). The network interface (NI) is designed as a bridge between an OCP [97] interface and the NoC switching fabric. Its purposes are the synchronization between OCP and network timing, (de-) packetization, the computation of routing information (stored in a Look-Up Table - LUT) and flit buffering to improve performance. The NI performs clock domain crossing and in order to keep its architecture simple, the ratio between network and core clock frequencies has to be an integer divider.

Both switches and network interfaces were originally modeled in SystemC as synthesizable RTL-equivalent models. However, conducting system level performance analysis with RTL simulation would imply unaffordable simulation times and resources. For this reason, leveraging the transaction-level (TL) models presented in [143], which abstract all the relevant mechanisms of the ×pipesLite architecture (retiming, buffering, arbitration, flow control, switching, synchronization), including injection and ejection interfaces. We demonstrated an accuracy of TLM simulation within 0.03% of RTL simulation while achieving one order of magnitude faster simulation speeds. A 256 core system with 16 millions of OCP read burst transactions can be simulated in a couple of hours. The simulator is event driven, therefore every time something changes in the network an event is scheduled. Each event contains data relevant to itself and executes the logical functions needed to handle the new network status. Furthermore, events are processed in time order and they can generate as much secondary events as needed (i.e, the reception of a head flit at an input port of the switch generates the event to route it).

Interestingly, the TLM simulator can back-annotate silicon-dependent parameters from the physical synthesis such as link latency and maximum operating frequency of NoC building blocks, thus making silicon-aware decision making viable even at the highest layers of the design hierarchy. Such parameters were extracted from post-layout analysis for 16 node topologies targeting STMicroelectronics 65 nm CMOS technology.

6.3 Backend synthesis flow

The practical feasibility of topologies under test was explored by means of a semi-automated design flow spanning from RTL description to layout-level verification. This enables us to explore and validate topologies down to the placement and routing steps, thus accounting for the effects of nanoscale technologies. The flow has been conceived for the physical synthesis of 2D-meshes and multi-dimension regular topologies. We use industrial tools for placement-aware logic synthesis and for place-&-route on an STMicroelectronics 65*nm* SVT technology optimized for low-power.

The first step of our backend synthesis flow is placement-aware logic synthesis through Synopsys Physical Compiler, applied to switch and network interface modules in isolation. This tool keeps optimizing the gate level netlist based on the expected placement and the wire loads it implies. The final resulting netlist considers placement-related effects and makes performance estimated at this level more trustworthy. Post-synthesis maximum frequency however represents a theoretical upper bound, since the critical path is computed inside the modules and actual routing of switch-to-switch links will then cause a further unpredictable performance drop.

Floorplanning and place&route are performed with the Cadence SoC Encounter tool. Computation tiles are replaced by non-routable hard obstructions of variable size depending on the underlying core/tile assumption. At first, hard black boxes are manually placed on the floorplan. Fences are then defined to limit the area where the cells of each network-on-chip module can be placed. Subsequently, the tool automatically places cells without trespassing the fences. Fence size is devised based on the report of the Physical Compiler on floorplan cell area of each module, while fence position depends on the switch placement strategy. Our choice was to aim at uniform latency across wiring dimensions. As an example, the placement strategy for a 2-ary 4-mesh topology with 1 tile per switch is illustrated in Figure 6.1.

Subsequent steps include clock tree synthesis and power supply network insertion. Each IP core is assumed to be an independent clock domain with its own clock tree. In this work, we assume all IP cores to work at the same frequency, which depends on the network speed and on the divider applied to the frequency-ratioed clock domain crossing mechanism at the network interface. After the power nets have been routed, the tool begins to route the logic wires. After an initial mapping, search and repair loops are executed to fix any violations. As a final step, post-routing optimizations are performed, including

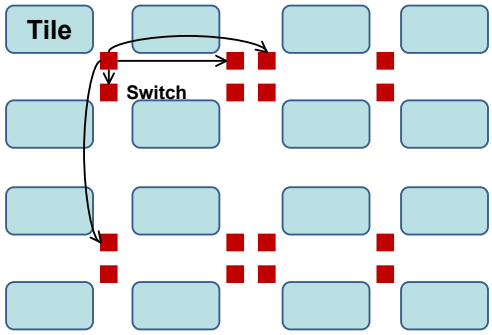


Figure 6.1: Floorplan of a 2-ary 4-mesh with 1 tile per switch.

crosstalk and antenna effect minimization. Finally, a signoff procedure can be run by using Synopsys PrimeTime to accurately validate the timing properties of the design.

6.4 Multi-dimensional Topologies

The target of our analysis is a *tile based architecture* where each tile is assumed to include at least *one processor and one local memory core*. Therefore, the asymmetric tile size needs to be accounted for when laying out the topology: this puts traditional assumptions on mesh and hypercube wiring in discussion.

Meshes can be referred to as k-ary n-meshes. The topology has k^n routers in a regular n-dimensional grid with k switches in each dimension and links between nearest neighbors. Moreover, the n-hypercube topology is a particular case of a mesh where k is always 2. Also, each switch can have one or more tiles attached.

In this chapter, topologies have been analyzed for a single system scale: 16-tile. Therefore, we analyze a 4-ary 2-mesh (referred to as 2D-mesh from now on) with one tile per switch, a 2-ary 4-mesh (4-hypercube) with one tile per switch, and a 2-ary 2-mesh with 4 tiles per switch. The 4-hypercube is a representative topology for those ones featuring a number of dimensions higher than 2, while the 2-ary 2-mesh illustrates the properties of *concentrated* topologies, connecting more nodes to the same switch.

Table 6.1 shows some representative data for the studied topologies. Please note that even with 1 tile per switch, 2 switch input and 2 switch output ports are required to connect the tile, since it includes an initiator and a target NI,

Topology	16 tiles		
	4-ary 2-mesh	2-ary 4-mesh	2-ary 2-mesh
Max Arity	6	6	10
Total Switches	16	16	8
Tiles x Switch	1	1	4
Total Ports	80	96	40
Bisection Cut	4	8	2
Ideal Diameter	6	4	3

Table 6.1: Topologies under test.

each with one input and one output port to the switch for receiving/sending data. The initiator NI uses the output port to send out packets and the input port to receive packets carrying read response data. The target NI uses the input port to receive packets carrying write data or read requests for the connected target. Read responses are packetized and sent out through the output port.

6.4.1 Communication semantics

As previously anticipated, a transaction-level simulator of the ×pipesLite NoC architecture is used for system-level performance analysis. The clock cycle accuracy with respect to RTL simulation is demonstrated in [143], which also demonstrates its superior simulation speed. This section recalls only the details of network traffic generation.

Our approach is to project network traffic based on the latest advances in communication middleware for MPSoCs and to assess its performance with an on-chip network as the communication backbone. We derive from the queue-based library in [128] the guidelines for producer-consumer interaction. That library is suitable for a number of MPSoC architectures, including the tile-based MPSoC scenario addressed in this chapter. Therefore, we built an abstraction layer on top of our TLM simulator, which models the behavior of a processor tile and of its HW/SW communication support.

In essence, the tile architecture consists of a processor core and a local memory core, as illustrated in Figure 6.2(a). Both cores are connected to the network through a network interface initiator and target respectively. We assume that the two network interfaces can be used in parallel. While the processor is read-

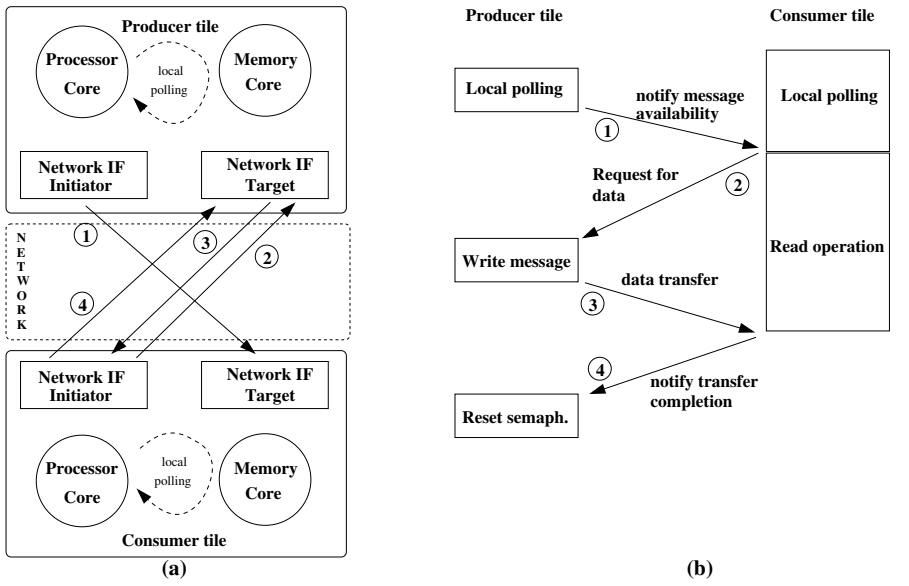


Figure 6.2: Tile abstraction and mapping of producer-consumer communication handshake on network transactions.

ing/writing from/to other tiles, the processor core of other tiles can read/write from/to the tile local memory. We assume producer-consumer communication between tiles based on the handshake shown in Figure 6.2(b). The producer checks local semaphores indicating whether there are previous pending messages for the target destination. If not, it writes communication data to the local tile memory and notifies data availability to the consumer by unblocking a remote semaphore. The consumer was meanwhile performing local polling on that semaphore. The producer is then free to carry out other computation or communication activities to other consumer tiles. Next, the consumer reads computation data from the producer tile, and sends a notification upon completion. This allows the producer to send another message to this specific consumer. The implementation of this communication protocol involves 4 network transactions: notification of data availability, read request, actual data transfer and notification of transfer completion. The producer local polling is performed in order to avoid congesting the network in case the consumer is slow in absorbing its input messages. The consumer local polling for incoming messages allows the consumer to synchronize data transfer operations from multiple producers. This avoids the collision of multiple packets in the network from the producers to the same consumer, since this latter operates all

transfers once at a time. Under these working conditions, concentrated architectures trading bandwidth for latency become attractive. However, physical implementation effects might put this picture in discussion.

6.4.2 Post-layout analysis

Network building blocks have been synthesized in isolation for maximum performance. Post-synthesis achievable frequencies are reported in Table 6.2 - 3rd row. They only account for timing paths in network logic and ignore those going through switch-to-switch links. We always found the critical paths to be in the switches and never in the network interfaces, and this explains why the network speed closely reflects the maximum switch radix of each topology.

	16 tile		
Topology	4-ary 2-mesh	2-ary 4-mesh	2-ary 2-mesh
Max. switch arity	6	6	10
Post-synthesis freq.	1 Ghz	1 Ghz	850 Mhz
Post-layout.	786 MHz	640 Mhz	600 Mhz
Core speed (max. 500)	393 MHz	320 Mhz	300 Mhz
Cell Area	$949k\ \mu m^2$	$1108k\ \mu m^2$	$733k\ \mu m^2$

Table 6.2: Physical parameters of topologies under test.

When post-layout speed is considered, we observe that inter-switch wiring has caused a significant performance drop for all topologies, depending on the wiring intricacy of each of them. As reported in Table 6.2 - 4th row, the more complex connectivity pattern of 2-ary 4-mesh results into a larger frequency drop than the 2D mesh. The 2-ary 2-mesh pays its lower number of switching resources with a larger switch-to-switch separation, and hence with a severe degradation of network performance due to link delay.

Since frequency-ratioed clock domain crossing is implemented in ×pipesLite network interface, network speed affects IP core speed. For this latter, a maximum value of 500 MHz is assumed in the context of multi-core embedded microprocessors. In spite of the post-synthesis speed drop, IP cores cannot sustain the network speed just at the same and therefore a divider of 2 is applied (Table 6.2 - 5th row).

The total larger number of switch I/O ports used by the 4-hypercube well mo-

tivates its larger area footprint than the 2D mesh. Cell (floorplan) area is considered in Table 6.2 - 6th row, while chip floorplan area is not reported since no specific optimizations were applied to it. Although the 2-ary 2-mesh has half the number of switches, its area is not halved as well, due to the fact that those fewer switches have a larger radix.

6.4.3 System Level Analysis

In order to simplify topology analysis, we assumed a workload distribution between the tiles which de-emphasizes the role of the topology mapping algorithm. In fact, we consider a parallel benchmark consisting of one or more producer tasks, a scalable number of worker tasks and 1 or more consumer tasks. Every task is assumed to be mapped on a different hardware tile. The producer task(s) reads in data units from the I/O interface of the chip and distributes it to the worker tasks. There are no constraints on which worker tile has to process a given data unit. Output data from each worker tile is then collected by one or more consumer tiles, which write them back to the I/O interface. All communications follow the queue based semantics illustrated in Subsection 6.4.1. The following assumptions were made on the I/O interface. A maximum of 8 I/O ports is assumed for 64 tile systems, each one used for input or for output. This number was reduced to 2 I/O ports for 16 tile systems. Such ports are accessed through sidewall tiles. The mapping of producer(s) and consumer(s) tasks is therefore constrained to these tiles. This I/O architecture is compliant with that of commercial embedded microprocessors, such as [125]. We set off-chip I/O devices access latency as a function of their frequency. We considered 20 cycles at 500 MHz and 15 cycles at 350 MHz.

While insensitive to worker tile mapping on the topology, our benchmark is still sensitive to I/O tile mapping on the chip periphery. For this reason, two scenarios are considered:
OneSided: all the I/O tiles are placed on the same side of the chip. This mapping has a high probability of I/O streams collision.
FourSided: I/O tiles are spread across the four chip sides. For 64 tile systems, at least one input and one output is placed at each side. For 16 tile systems, I/O tiles are decoupled and each input is placed at the opposite side of its output.

Figure 6.3 shows performance of 16 tile topologies in clock cycles and elapsed time. The left side shows results for *OneSided* mapping, while the right side shows results for *FourSided* mapping. In *OneSided*, the hypercube (2-ary 4-mesh) reduces total number of cycles by 27.4%. In this mapping, inputs and outputs are placed at the top of the chip. Therefore, path length is very irreg-

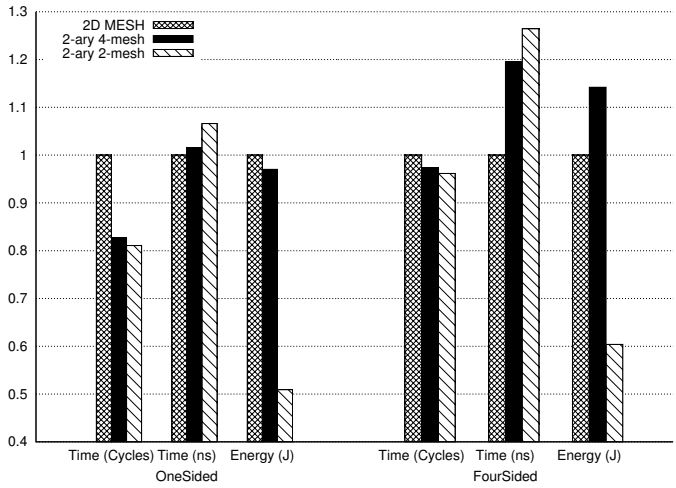


Figure 6.3: Normalized execution time for 16 tile topologies.

ular, as tiles located at the top of the chip can reach input and output through a shorter path than the tiles located at the bottom. This makes topologies with higher network diameter more sensitive to this effect. Moreover, the probability of collision between I/O streams is quite high, thus penalizing topologies with fewer dimensions. The concentrated hypercube (2-ary 2-mesh) reduces cycles only by 1.6% over the hypercube, despite its lower diameter. The main reason for this lies in the chip I/O: as the bottleneck introduced by the topology is alleviated, the external I/O bottleneck arises. So, the maximum improvement that can be achieved in the 16 tile system is bounded by I/O speed. On the other hand, *FourSided* mapping (see Figure 6.3) reduces I/O streams collision probability while providing homogeneous path length, thus decreasing performance differences among topologies.

Unfortunately, these results become irrelevant when considering real operating frequency. While in *OneSided* mapping the reduction of cycles of the 4-hypercube barely compensates for its lower frequency, in *Foursided* mapping it is not enough, and the 2D mesh turns out to be the best topology overall.

Figure 6.3 shows the energy consumed by each topology. Numbers are measured from the post-layout netlists obtained as described in Section 6.3. While the 4-hypercube consumes almost the same or even more energy than the 2D mesh depending on the I/O tile mapping, the concentrated hypercube shows superior energy saving properties (from 40 to 50% less than the 2D mesh). Of note, the reason why the energy trend between the 4-hypercube and the

2D mesh is inverted when moving to a different I/O tile mapping, lies in the trade-off between the energy consumed by the number of available links for a specific topology and the amount of traffic congestion they are able to absorb depending on the chosen mapping.

6.4.4 Discussion

This first part of this chapter presented our analysis framework for the assessment of k-ary n-mesh topologies for regular tile-based architectures. Our work considered a number or real-life issues: physical constraints of nanoscale technologies (post-layout performance, area and power results are given), different physical design techniques, the role of the chip I/O, the communication semantics of middleware for MPSoCs and the role of I/O tile mapping.

We found that the intricate wiring of multi-dimension topologies or the long wires required by concentrated k-ary n-meshes can be changed into 2 different kinds of performance overhead by means of proper design techniques:

- operating frequency reduction. This is likely to be the technique of choice for small scale systems. In this case, in spite of a lower number of execution cycles, multi-dimension topologies loose in terms of real execution time due to lower working frequency. Nonetheless, reducing the number of dimensions and connecting more cores to the same switch represent a way to trade performance for power and area;
- increased latency. An aggressive utilization of retiming stages allows to sustain operating frequency while increasing network latency. The switch delay associated with its radix poses an upper bound to the effectiveness of this technique. Finally, a significant area and power overhead is expected, since retiming stages need to be also flow control stages.

Overall, we found the 2D mesh to still outperform the hypercubes. This is counterintuitive, since hypercubes should scale better in principle. The motivation lies in the mismatch between multi-dimension topologies and the 2D silicon surface (whatever the kind of performance overhead it is changed into) and in the chip I/O bottleneck, which prevents an aggressive performance speed-up at least in clock cycles. Removal of this bottleneck is mandatory to achieve significant performance differentiation between topologies.

Next section will utilize our analysis framework for the performance evaluation of another important topology family, i.e., *multi-stage interconnection*

networks. In this section, the practical feasibility of such topologies will be analyzed and ad-hoc solutions will be proposed for their floorplan. The main goal here was to assess whether or not the theoretical superiority [141, 142] of such multi-stage network still holds when layout considerations are taken into account. Furthermore, as such topologies typically exhibit a large cost in terms of resources (high number of switches with high radix), we considered also a low-area cost implementation named *RUFT* and we evaluated if such topology can be exploited to implement very fast networks regardless of their supposed wire intricacy. For this reason, each switch of the lowest layer has been considered connected to a separate core and memory thus keeping the switch radix of every switch as low as possible. Furthermore, projections for 64-core system have been extrapolated giving useful insights for both the practical feasibility of such topology and their possible final performance.

6.5 Multi-stage Interconnection Networks

Most of the past evaluations of fat-trees for on-chip interconnection networks rely on oversimplifying or even unrealistic architecture and traffic pattern assumptions, and very few layout analyzes are available to relieve practical feasibility concerns in nanoscale technologies. Our work aims at providing an in-depth assessment of physical synthesis efficiency of fat-trees and at extrapolating silicon-aware performance figures to back-annotate in the system-level performance analysis. We utilized a 2D-mesh as reference architecture for comparison. Finally, in order to mitigate the implementation cost of k-ary n-tree topologies, we review an alternative unidirectional multi-stage interconnection network able to simplify the fat-tree architecture and to minimally impact performance, resulting in a more power-aware fat-tree implementation.

Networks-on-chip (NoCs) closely resemble the interconnect architecture of high-performance parallel computing systems [1]. For this reason, the interconnection topologies used in the early NoC prototypes can be traced back to the field of parallel computing. In particular, NoC architectures aiming at low latency communication, performance scalability and flexible routing selected fat-trees as their reference topology. The switch for the butterfly fat-tree network of [150] or the SPIN micronetwork [13] are examples thereof.

However, other topologies have found wider application in common NoC design practice so far, namely 2D-meshes and even folded tori [139, 140]. In fact, technology scaling to the nanoscale era brings physical design issues to the forefront, such as the reverse scaling of interconnects. In this context, 2D-mesh

and torus topologies exhibit a grid-based regular structure which is intuitively considered to be matched to the 2D chip layout. In contrast, the higher wiring irregularity and the larger switch radix of most fat-tree configurations raise some skepticism about their practical feasibility. Moreover, instead of aiming strictly for speed, designers increasingly need to consider energy consumption constraints, and fat-trees are expected to pay the increased connectivity they provide with a significant area and power cost.

In spite of these concerns, constant attention has been devoted to tree-based topologies in the NoC community, proving their superior performance with respect to 2D meshes under different kinds of synthetic traffic patterns [141,142]. However, these analysis frameworks are not able to be fully convincing and to impact NoC design practice in many senses. First, they often rely on abstract network simulators which cannot model the behavior of any real architecture and sometimes make unrealistic assumptions, such as packet drop or TCP-compliant network transport protocols for on-chip communication. Second, most works really miss an in-depth physical analysis of layout feasibility and efficiency. Even when area synthesis results are provided, the impact of wiring congestion and interconnect delay on network performance is only assessed by means of analytical models. Also, the effectiveness of advanced design techniques such as clock or power gating or link pipelining is ignored.

This second part of the chapter aims at overcoming some limitations of previous fat-tree topology evaluations for NoCs, by primarily investigating the layout feasibility and the implications of physical mapping efficiency on system-level performance figures. As we have previously seen applied to multi-dimensional topologies, our system- to layout-level analysis framework has been utilized to evaluate k-ary n-tree topologies with layout awareness. The objective to perform a comprehensive layout-to-system level assessment of a fat-tree and its comparison with a 2D mesh forced us to necessarily restrict our exploration to a reasonable 16 core system. However, scalability insights into the connectivity of 64 core systems are provided as well.

The fat-tree we considered is the commonly used k-ary n-tree, as it is defined in [127]. It allows to infer the topology by structuring multiple switches of constant size in a regular pattern and in a more compact layout. In spite of these properties, k-ary n-trees cannot avoid the trade-off between the increased connectivity they provide and the higher resource cost for it.

As a consequence, we also review an architecture optimization of fat-trees that aims at simplifying the switch architecture and saving network resources, while minimally impacting the high performance of these topologies. In prac-

tice, the proposed unidirectional multi-stage interconnection network (MIN) reduces the complexity of the downward phase, resulting in faster and more compact switches and possibly in power savings with respect to k-ary n-trees.

6.5.1 Topology analysis

Network size	**16 Cores**			
Topology	4-ary 2-mesh	2-ary 4-tree	Unidir 2-ary 4-tree	Unidir 4-ary 2-tree
Switch radix	5	4	2	4
Total switches	16	32	32	8
Total Ports	64	112	64	32
Diameter	6	6	3	1
Bisect. Cut	8	16	8	8

Table 6.3: Network topologies under test.

This section provides a brief description of the topologies under investigation. Deterministic routing is always used with the aim of having in-order delivery of packets. Half of the cores are processor cores, while the remaining half consists of the private memory cores of the processors. Table 6.3 shows some representative data for the studied networks.

Meshes can also be referred as k-ary n-meshes. This topology has k^n routers placed in a regular n-dimensional grid with k switches in each dimension and links between nearest neighbors. The routing algorithm we use for meshes is Dimension-Order Routing (DOR).

Fat-trees (FT) are a particular sub-set of a family of topologies known as multistage interconnection networks (MINs). In MINs, switches are structured in multiple stages. Each switch can only be connected to switches belonging to their previous or to their next stage. Cores are connected to the switches of the lowest stage. For instance, Figure 6.4(a) depicts a MIN with three stages and eight interconnected cores. Fat-trees are based on complete trees, but differ from them for preserving bandwidth near the root. For this purpose, the switch radix grows as we move up to the root, which makes the physical implementation impractical. Therefore, some alternative implementations have been proposed to use switches of fixed arity [127].

In particular, we focus on a specific implementation: the k-ary n-trees (see

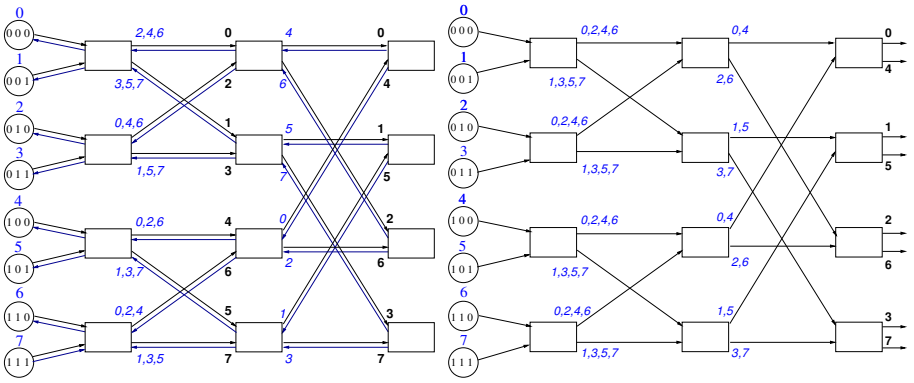


Figure 6.4: (a)A 2-ary 3-tree topology. (b)A RUFT derived from a 2-ary 3-tree. Each switch port shows its reachable destinations.

Figure 6.4(a)), a parametric family of regular multistage topologies. The number of switch stages is n and k is the arity or the number of links of a switch that connect to the previous or to the next stage (i.e., the switch radix is $2k$). Links of a switch can be classified as ascending or descending, depending on whether they connect to switches located in the higher or lower stage, respectively. All the switches have the same number of ascending and descending links. A k-ary n-tree is able to connect $N = k^n$ processing nodes using nk^{n-1} switches and $2nk^n - k$ unidirectional links.

Routing in Fat-trees is performed in two phases: ascending and descending. In the ascending phase, several minimal paths for each source-destination pair are possible. Packets are forwarded upwards in the tree until one of the *nearest common ancestors* between source core and destination core is reached. At this point, the descending phase is started. This latter is deterministic by construction and the path used in this phase depends on the nearest common ancestor that has been reached in the ascending phase. To provide a deterministic routing algorithm for the ascending path, a unique path must be selected for each origin-destination pair among all the possible ones. The deterministic routing algorithm for fat-trees that we utilized has been presented in [120].

In this routing algorithm, during the ascending phase consecutive destinations are shuffled among the different ascending links of the switches. Figure 6.4(a) shows how destinations are distributed in each switch among the different ascending links: each ascending port is labeled in italics with the destination

cores that are reachable through it. Also, Figure 6.4(a) shows how destinations are distributed in the descending phase; in this case, descending ports show in bold their sets of reachable destinations. As can be seen, each descending link is only used by a single destination.

Reduced Unidirectional Fat-Tree (RUFT) is a topology resulting from the simplification of the above mentioned k-ary n-tree when using the deterministic routing algorithm proposed in [120]. RUFT was first presented in [119] as a conceptual topology scheme, without any implementation analysis. When using this deterministic routing algorithm, the whole descending phase can be reduced to a single long link that connects the output ports of the switches of the last stage with the input port of the corresponding destinations. In this way, switches become unidirectional and all packets must reach the last stage of the network (Figure 6.4(b)).

Although the use of long links may compromise the feasibility of this topology, all the hardware resources related to the descending phase are reduced to these long links, simplifying the switch architecture. The resulting topology resembles an unidirectional butterfly, with a permutation of the reachable destinations from the last stage.

An unidirectional reduced k-ary n-tree is able to connect $N = k^n$ processing nodes using nk^{n-1} unidirectional switches and nk^n unidirectional links.

When evaluating RUFT, we consider two different topologies for each network size. The first ones are the unidirectional networks resulting from the simplification of standard k-ary n-tree fat-trees, that is, we analyze an unidirectional 2-ary 4-tree in the 16 cores category, while we analyze an unidirectional 2-ary 6-tree in the 64 cores category. These unidirectional networks have the same number of switches of the original fat-trees but, as can be seen in Table 6.3, the switch radix is reduced to one half.

This leads us to explore an alternative RUFT implementation, denoted as **S(implified)-RUFT** hereafter, trading switch radix for the switch count. In this direction, we define an unidirectional 4-ary 2-tree for the 16-core system and a 4-ary 3-tree for 64 cores. This way, the total number of switches is lower than that in the original fat-tree, as reported in Table 6.3. Please observe that we were not able to perform the same switch count reduction in the original bidirectional fat-tree since this would have led to switches with an overly high radix (i.e., 8x8), resulting in a significantly lower maximum operating frequency [108]. Concisely, the topologies explored in our experiments have a switch radix which is always lower than 5.

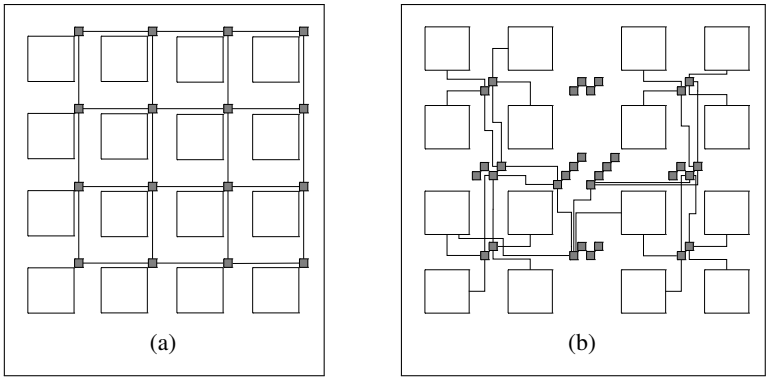


Figure 6.5: Floorplans of topologies under test. a) 4-ary 2-mesh b) 2-ary 4-tree RUFT. Only the main wiring patterns are reported.

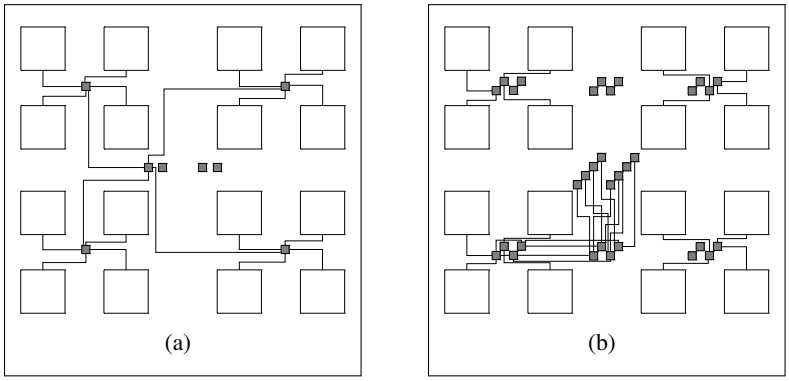


Figure 6.6: Floorplans of topologies under test. c) 4-ary 2-tree S-RUFT d) 2-ary 4-tree FT. Only the main wiring patterns are reported.

6.5.2 Floorplan design

The physical synthesis methodology adopted here is the same as we have seen in previous Section 6.3 with the only exception that the tile size is set to $1mm \times 1mm$. In particular, this section discusses the criteria for floorplan design of the topologies under test. Synthesis time constraints forced us to limit the physical design to 16 core systems. However, projections of 64 core system designs will be extrapolated as well, thus getting useful scalability indications. A more adequate methodology to extrapolate accurately area/performance of 64 core systems will be presented in Chapter 8.

The 2D mesh floorplan is straightforward (see Figure 6.5(a)) due to its regular

grid structure matching the 2D silicon surface. Things are more complex for 2-ary 4-tree FT (Figure 6.6(b)). The topology consists of 4 switch stages with 8 switches each. Our floorplanning strategy was to minimize wirelength between consecutive switch stages. For this reason, cores are clustered in groups of four and the connected switches (of the first and second stage) are placed in the middle of each cluster. The third switch stage is split into 2 subgroups and placed between the upper and lower clusters. Each subgroup serves its relative counterpart from the first and second stage. The last switch stage is located in the center of the chip. The presented layout exhibits equalized wirelengths between the second, the third and the last stage of switches. Another appealing characteristic is the straightforward scalability of this floorplanning strategy to 64-core systems.

The 2-ary 4-tree RUFT (Figure 6.5(b)) is a novel unidirectional fat-tree which has never been laid out before. This time, a switch belonging to the last stage is directly connected to the network interface of a core. This link is viewed in [119] as the intuitive weakpoint of the layout of this topology. To go around this problem, our floorplanning directive in this case is to minimize the wirelength of this critical set of links. Thus, switches from the last stage are positioned in the middle of each 4-core cluster. Obviously, also the first stage has to be close to the appropriate cores. Therefore, it is placed above and below the middle of the chip between two neighboring clusters, so to equalize the link length and keep the delay as homogeneous as possible on the wires of the first stage. As the third stage has to be connected to the last one and to the second one, two groups of switches belonging to the third barrier are placed at the left and at the right of the chip center. This also achieves an easy connection with the second stage, which is positioned in the center of the chip. An interesting property of the presented floorplan is that the link length is kept almost constant on a stage-by-stage basis. Unfortunately, although the aforementioned layout elegantly solves the placement problem for this 16-core RUFT, its scalability to 64 cores is not straightforward and as efficient. In this sense, it can be considered an ad-hoc floorplan for 16 connected nodes.

Finally, the floorplan for a 4-ary 2-tree S-RUFT is illustrated in Figure 6.6(a). Although the number of switch stages is small (just 2), this is a challenging topology from a physical layout viewpoint. The problem stems from the fact that each switch of the first stage is directly connected to all the switches of the second stage. Moreover, a second stage switch is connected to network interfaces of cores, since this is again a unidirectional topology. Following the same floorplanning strategy of the 2-ary 4-tree RUFT, a switch from the last stage has to be placed in the middle of a 4-core cluster.

Unfortunately, in this case the switch has to be interconnected also to all the switches of the first stage, which should be necessarily placed in the center of the chip to equalize link length. This results in very long links going from each cluster to the switches at the center of the layout. As shown later on, this dramatically impacts the achievable post-routing performance of this topology.

6.5.3 Floorplan scalability to 64 cores

The 2D mesh is easily scalable to 64 cores, resulting in an 8-ary 2-mesh. Also the floorplan of the fat-tree (which becomes a 2-ary 6-tree FT) can be easily scaled with our strategy. In fact, a 64 node topology can be viewed as built up by four clusters of 16 cores with two additional switch stages connecting them to each other. The four clusters can be internally placed as previously described. This approach features a high level of modularity along with an adequate link length scalability. Moreover, all the link lengths from the fourth stage (the last stage in the 16-core system) to the two additional ones in the 64-core system can be tuned to be the same in the final layout.

For the 2-ary 6-tree RUFT, again 4 clusters of 16 cores are formed. However, the main issue is that unlike the usual fat-tree, the center of each cluster is now occupied by the second stage of switches and not the fourth one (i.e., the last in the 16-core system). Therefore, the fourth stage being scattered on the edges of the chip, the connection with the additional switch stages is not equalized, leading to links of uneven lengths. The problem stems from the fact that the layout was customized for a 16-core system and afterwards a modular approach was applied in order to re-use previous effort. A better floorplan could be obtained by customizing it for a 64-core system, which is not as intuitive as for a 16-core system and falls outside the scope of this thesis.

Finally, the S-RUFT topology now becomes a 4-ary 3-tree (3 switch stages). Without going into further details, this floorplan turns out be as inefficient as that of the RUFT topology, and results in very long links between the last stage of switches and the network interface of connected cores.

6.5.4 Post-Layout analysis

Timing

In all topologies, the network building blocks have been synthesized for maximum performance. The post-synthesis critical paths (ignoring place&route effects) are reported in Table 6.4, 3rd column. We found the critical path to

Topology	Max Arity	Post-. Synthesis.	Post-place&route	Area footprint	Tot. Wire Length
4-ary 2-mesh	5x5	0.9 ns	1.19 ns	802k μm^2	8081 mm
2-ary 4-tree RUFT	2x2	0.6 ns	1.15 ns	795k μm^2	8370 mm
2-ary 4-tree FT	4x4	0.8 ns	1.29 ns	1280k μm^2	12389 mm
4-ary 2-tree S-RUFT	4x4	0.8 ns	2.1 ns	400k μm^2	7346 mm

Table 6.4: Physical synthesis reports.

be always in the switch and to reflect the maximum switch radix of the topology. Obviously, the lower switch radix of the 2-ary 4-tree RUFT results in a much shorter critical delay. We then iterated place&route starting from the post-synthesis target frequencies.

For a 16 core system, we assessed the integration of a link pipelining technique in the backend synthesis flow overly expensive. For this system size, we aim at analysing whether the delay of switch-to-switch links already impacts overall NoC performance or not. For what follows, the mismatch between wiring of a topology and the 2D silicon layout will result in increased clock cycle time after place&route.

Timing closure was achieved at the post-layout speed reported in the 4th column of Table 6.4. Performance degradation turns out to be very significant, thus pointing out the critical role of interconnects. In fact, for all topologies (and even for the 2D mesh) the critical path goes through the switch-to-switch links. While data/flit wires are sampled at the input and output port of the switch, flow control wires go through the FSM of the flow control stages at switch I/O. As a consequence, these control wires go through logic gates whose delay adds up to the link delay, determining the critical path. The impact of the link delay is evident, in that the critical delays of the topologies are differentiated by the longest link in that topology.

The 2-ary 4-tree RUFT wastes part of its speed with respect to the 2D mesh due to the use of longer links. In practice, the theoretical performance enhancement associated with a lower switch radix does not materialize after place&route, but has served as timing margin against physical degradation effects.

The 2-ary 4-tree FT has incurred a lower degradation than the 2-ary 4-tree RUFT, but its post-synthesis performance was lower, therefore it ends up running even slower than the 2D mesh.

Finally, for the 4-ary 2-tree S-RUFT the above effects are even more apparent due to longer wires that are needed to connect a low number of switching re-

sources sparse all around with each other. The lower area footprint is achieved at the cost of a remarkable 162% speed degradation after place&route.

Overall, in spite of a good post-synthesis frequency, thanks to the lower radix MINs typically suffer from a more significant performance degradation after place&route due to their more intricate wiring compared to a 2D mesh. Hence, final performance cannot be predicted from early post-synthesis results in a straightforward way without accounting for interconnect-related effects.

Area

Since we only performed a coarse grain shrinking of placement blocks on the floorplan without a finer-grain layout area optimization, we only report here total floorplan cell area. See Table 6.4, 5th column. The 2D mesh and the 2-ary 4-tree RUFT exhibit almost the same area, in that they have exactly the same number of I/O ports for a 16-core system, which are the ones that mainly determine switch area. The 2-ary 4-tree FT has a significant 60% area overhead with respect to the 2D mesh, which can be correlated with a 75% increase in the number of switch ports. Interestingly, although featuring the same number of switches, the 2-ary 4-tree RUFT achieves a significant area savings with respect to the FT since it employs lower-radix switches. Obviously, the 4-ary 2-tree S-RUFT exhibits the lowest area footprint with just 8 4x4 switches.

Wirelength

A side effect of using lower switch radix in 2-ary 4-tree RUFT is the large saving in total wire length with respect to the 2-ary 4-tree FT, although they are using the same number of switches. 2-ary 4-tree RUFT and 2D mesh consume almost the same amount of wiring resources in spite of the higher routing complexity of RUFT. This further confirms the high efficiency of the customized layout for this topology. Finally, the 4-ary 2-tree S-RUFT has obviously a lower total wiring length, mitigated by the need to interconnect few switch components that are far apart from each other.

Power

Power was computed with the same producer-worker-consumer benchmark of [71], which emulates the average communication bandwidth requirements of a partitioned MPEG-4 application. This traffic pattern models bursty accesses to 2 memories serving as I/O interfaces, and all-to-all accesses associated with a cooperative computation process. Processor and memory cores were mapped on each topology. For the 2D mesh, mapping was done in such a way that each processor core is always 1-hop away from its private memory core. In all fat-trees, a processor core and its memory core were connected to the same switch at the lowest stage of the topology. This mapping may turn out to

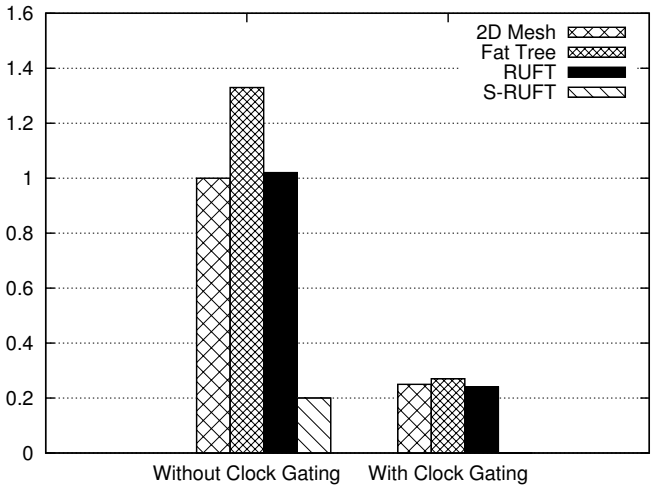


Figure 6.7: Normalized total power.

be highly performance sub-optimal for fat-trees (indeed it is a worst-case for RUFT [119]), however we opted for it since it is very intuitive and avoids the use of high-level mapping tools (which is outside the scope of this work).

Power results are illustrated in Figure 6.7. Power of 2D mesh and 2-ary 4-tree RUFT turns out to be the same due to the equivalent number of I/O ports in both networks. The correlation of power with the number of buffering resources is further emphasized by power reports of the 2-ary 4-tree FT, which is the most power consuming topology: 34% more than the 2D mesh. It is worth noting that all aforementioned topologies work at a comparable frequency (the post-place&route one), whereas the 4-ary 2-tree S-RUFT works at approximately half of the speed. If we combine this feature with its much lower number of I/O ports, we can explain the significantly lower power consumption.

We also went through the physical synthesis process again and enabled the clock gating feature. Results are provided in Figure 6.7 for the most power-consuming topologies. On average, an impressive 75% power saving is materialized by clock gating. This brings power of the most power-hungry topologies close to that of the 4-ary 2-tree S-RUFT. From a timing viewpoint, we observed no major modifications with respect to previous results.

6.5.5 System Level Analysis

Based on the physical insights reported above, we annotated maximum clock speed of each topology into our transaction-level modeling simulator, which was demonstrated in [155] to accurately reproduce performance results of RTL simulation (within 1%) with orders of magnitude lower simulation times. Our TLM simulator abstracts the behaviour of the ×pipesLite NoC architecture and enabled system-level performance analysis of topologies.

Although our tests make use of traditional synthetic traffic patterns such as uniform traffic and hot-spot, we also model a relevant feature of real-life traffic which makes the analyis more accurate.

In particular, we model actual OCP transactions at the network boundary, thus accounting for a more realistic traffic injection/ejection process. In turn, this allows us to capture network behaviour in response to OCP read or write transactions, which give rise to a different traffic pattern on the network. For instance, write transactions are changed into relatively longer network packets, while read transactions generate short request packets to the memory core and long response packets coming back. Moreover, read transactions are blocking for the ×pipesLite network interface, while write transactions are not.

In uniform traffic, OCP transaction destination is randomly chosen among all the available memories in the system. Transaction size is randomly chosen, with a minimum of 4 and a maximum of 16 data burst beats. The idle time between consecutive transactions is adjusted to ensure that all networks are working close to their saturation point.

Figure 6.8 shows performance results for a 16-core system under uniform traffic. Results are normalized to the 2D mesh. The left side of the figure shows total simulation time in clock cycles, while the right one shows the elapsed time in nanoseconds (accounting for the actual post-layout clock period from Table 6.3). Small differences in performance can be seen when execution cycles are evaluated. When only write transactions are considered, the best result is achieved by the fat-tree, that is the topology that provides the larger bisection bandwidth (see Table 6.3). On the contrary, topologies ensuring lower latency (like S-RUFT) can better handle read transactions. RUFT has no locality, therefore its average communication latency equals that of the 2D mesh, thus achieving same performance. Moreover, for this small scale system, fat-tree and 2D mesh have the same diameter and, because of the chosen mapping, the same average distance between every source-destination pair. This explains their performance balancing as well.

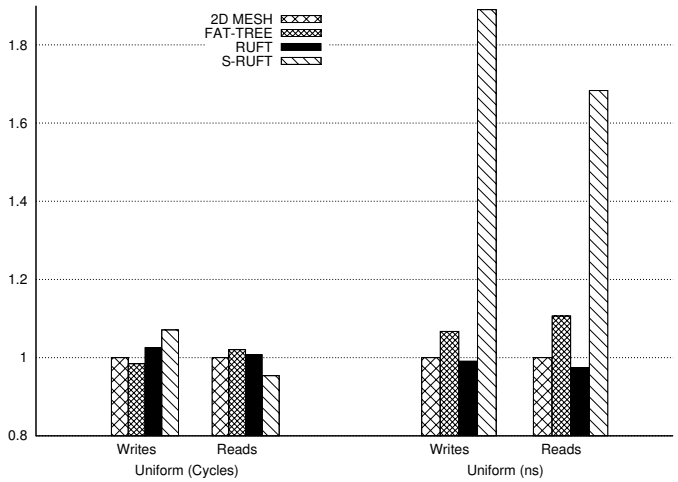


Figure 6.8: 16-core system. Normalized performance.

Unfortunately, when the real achievable speed is considered for each topology, S-RUFT becomes unusable. The fat-tree suffers from around 10% performance penalty with respect to the 2D mesh, in spite of its higher number of network resources, while RUFT proves an equivalent solution to 2D mesh (also power-wise). Given the lower wiring complexity of 2D mesh, this latter might be the reference solution for small scale systems.

6.5.6 Discussion

The final part of this chapter demonstrates that fat-trees are feasible for on-chip networks from a physical design viewpoint. Unfortunately, for small scale systems, they are not able to capitalize on their better performance figure scalability yet. 2D mesh is in contrast a very efficient solution from all viewpoints. Using MINs with less resources (namely S-RUFT) incurs serious physical design issues, mainly associated with long link delay. Moreover, circuit-level power control techniques such as clock gating can significantly cut down on power of the more complex topologies.

As the system size scales up, 2D meshes however suffer from poor performance scalability. Hence, the need for alternative topologies becomes more stringent. k-ary n-trees can provide that performance scalability, but at an impractical power and area cost. In this scenario, unidirectional MINs (like

RUFT and S-RUFT) become attractive for their reduced power and area overhead. Performance-wise, they are effective for latency-sensitive traffic, while they cannot handle bandwidth-intensive traffic as effectively. Unfortunately, their advantages cannot be easily materialized due to a more intricate physical design. No clearly scalable floorplanning strategy is known, and a lot of link retiming and flow control stages would be needed to sustain clock speed. The area and power overhead associated with these stages might turn out to be unacceptable. Due to link pipelining, area and power cost of k-ary n-trees becomes further prohibitive. From a performance viewpoint, the increased link latency of unidirectional MINs causes a lower performance but still better than 2D mesh and also than k-ary n-trees under latency-sensitive traffic.

6.6 Summary

In this chapter, we presented "our system- to layout-level" framework for evaluating network topologies with physical awareness in Section 6.2. Section 6.3 described the physical synthesis methodology adopted throughout this chapter. Such methodology has been utilized to characterize the investigated topologies from the viewpoint of physical design. In Section 6.4, multi-dimensional topologies have been analyzed whereas multi-stage interconnection networks have been studied in Section 6.5. The work presented in this chapter will be further extended in Chapter 8 where an accurate characterization methodology to for analyzing 64-tile networks will be presented. The next chapter explores the design space of the NoC interswitch channel where different optimizations can be utilized to improve the performance of the link itself.

7

Link Design Techniques Evaluation

EACH link implementation solution is not just a specific synthesis optimization technique with local performance and power implications, but gives rise to a well-differentiated point in the architecture design space. This is in an effect of close dependency existing between architecture and physical design layers in nanoscale technologies.

This chapter assesses the impact of NoC link inference techniques (e.g., repeater insertion, link pipelining) by means of commercial backend synthesis tools, taking the system-level perspective. In fact, performance speed-ups and power overhead are not only evaluated for the links in isolation but for the network topology as a whole, thus showing their sensitivity to the link inference strategy. Various k-ary n-mesh topologies are considered during our analysis as they provide a representative range of complex interconnection networks with increasing total wirelength.

7.1 Introduction

Previous studies of wire scaling effects based on ITRS roadmaps return a grim view. [116] extends performance projections of wires out to the 13nm technology node and sees both local and global wires degrading relative to gates over nine generations, by one and three orders of magnitude respectively.

This trend is not just impacting circuit design, but is having heavy architecture-level implications as well. In this direction, network-on-chip (NoC) architectures rely on aggressive path segmentation and regular connectivity patterns, at least for general-purpose tile-based NoCs. The ultimate objective is to come up with a modular architecture removing global wire delays from the critical path. This technique has been successful since the NoC design early days as

the critical path is often confined into the NoC switch arbitration logic [115].

However, the unrelenting pace of technology scaling in the nanoscale era is bringing interconnect-related issues to the forefront even for NoC design. For a 65nm technology, [108] points out the significant gap between post-synthesis and post-place&route performance reports affecting NoC modules when logic synthesis and placement are carried out as two clearly separated stages. Inaccurate wire load models are at the root of this gap, therefore placement-aware logic synthesis tools are envisioned as very important.

Moreover, although global wires are intrinsically segmented, the maximum interswitch link length still plays a key role in topology design. [155] proved that a bufferless implementation of interswitch links already suffices to move critical path back to them. The additional delay contributed to these links by some flow control or buffer control gates along them further stresses their key role for system performance. This trend has to be taken into account especially when evaluating topologies with complex connectivity patterns, e.g. high-dimensional ones or fat-trees. In fact, the delay of wires in the higher dimensions or the intricacy of the routing process may directly and significantly degrade overall network speed. Moreover, such delay is hardly predictable due to layout constraint effects, which makes the indications of abstract pencil-and-paper floorplanning considerations inaccurate if not misleading.

Fortunately, designers can today rely on a number of well-known techniques to engineer delay-optimized wires, thus making system performance less sensitive to the raw numbers of wire performance. As an example, a traditional design technique for long links consists of inserting equally spaced CMOS repeaters to deal with resistive loss along the wire. This makes the delay of repeated wires almost linear with respect to their length rather than quadratic. However, with the increase of the number and the density of the wires at each new technology node, interconnect area and power might be severely impacted. Power overheads in the order of tens of Watts might be expected [101]. Moreover, the use of repeaters in NoC interswitch links is subject to layout constraints, since they have to be inferred in the inter-tile routing channels, thus impacting total floorplan area.

Another way of tackling timing violations on long links is by pipelining it. By providing one or more extra clock periods to traverse long distances, pipeline stages along links solve the link infeasibility issue at a much lower cost than deploying whole NoC switches in place of them. However, the major drawback is that flow control must be extended to account for the fact that feedback signals now return after multiple clock cycles instead of in the same clock

period. This can be handled by deeper buffers at link endpoints or by pipelining the link with flow control-aware elements. In all cases, a significant power overhead might be incurred.

The effectiveness of a link performance boosting technique cannot be assessed on the link in isolation, but has to be captured at system (topology) level. In fact, link performance and power have a number of high-level implications, especially overall network speed. Unfortunately, most previous work on physical link design investigates cost-benefit trade-offs only for the link in isolation, while ignoring the sensitivity of topology metrics to a specific link design technique. This chapter aims at going a step further and at demonstrating topology level implications of high-impact interconnect optimization techniques such as link buffering and pipelining. Baseline topologies with bufferless links will be used for the sake of comparison. This chapter pursues a twofold objective. On one hand, it assesses whether for a given topology an investment on a faster link pays off in terms of total energy. On the other hand, it investigates whether different link synthesis techniques put well-known performance-power trade-offs between various kinds of topologies under investigation in discussion.

The remainder of this chapter is organized as follows. Section 7.2 describes the topologies under analyzing at first a common mismatch in the estimation of the theoretical link length when contrasted with its silicon figure. Furthermore, Section 7.3 illustrates the three fundamental link inference techniques studied in this thesis. Experimental results of the presented analysis are reported in Section 7.4. Section 7.5 discusses and wraps-up the achieved results whereas, Section 7.6 summarizes the main contributions of the chapter.

7.2 Topologies analysis

Link design techniques impact topology quality metrics in two ways. On one hand, it is the longest link in the topology that determines its maximum achievable speed. Therefore, cutting down the delay of that link is beneficial for the whole topology. On the other hand, the cost for boosting the performance of the critical link becomes relevant for the topology only when there are more of such critical links and they account for a significant fraction of the total topology wirelength.

We found representative design points for our link sensitivity analysis to be available in the k-ary n-mesh family of regular NoC topologies. In the standard nomenclature, the topology has k^n routers in a regular n-dimensional grid with k switches in each dimension and links between nearest neighbors. k-ary

Topology	4-ary 2-mesh	2-ary 4-mesh	2-ary 2-mesh
Max. Arity	6	6	10
Link Length	1	1,2	1
Switches	16	16	4
Max. Hops	6	4	2
Bisection bandwidth	4	8	2
Tiles per Switch	1	1	4

Table 7.1: Topologies under test.

2-mesh topologies have been traditionally viewed as the candidate topologies for general-purpose tile-based MPSoC architectures [112], although raising scalability concerns. An optimization of this topology consists of connecting more tiles per switch while keeping the overall number of tiles the same. This way, the total number of switches is reduced, hence resulting in a lower number of hops, but the bisection bandwidth is reduced as well. In this chapter we focus on 16 tile systems, hence consider a 4-ary 2-mesh (referred to as 2D-mesh from now on) with one tile per switch and a 2-ary 2-mesh with 4 tiles per switch. The latter solution is denoted as the *concentrated topology*. Table 7.1 shows some representative data of the studied topologies.

Both the 2D mesh and its concentrated counterpart feature 2 dimensions and homogeneous interswitch wire lengths. However, such wire length will not be the same in the two topologies due to floorplan constraints reflecting a well-known trade-off for these topologies: spread-around topologies on one hand (large number of low-radix switches) as opposed to more concentrated ones with a small number of high-radix switches *placed further apart* to optimize connectivity with a large number of tiles. In fact, these latter have to be placed around the switches they have to be connected to, thus separating the switches in space. In these two topologies, all the links have to be performance-optimized in order to speed-up the entire topology, hence the associated cost will be more relevant in relative terms with respect to the baseline unoptimized topologies.

A different design point is represented by hypercubes. The n-hypercube topology is a particular case of a mesh where k is always 2. For 16 tile systems, a 2-ary 4-mesh (4-hypercube) can be obtained from the 2D mesh by increasing the number of dimensions and by reducing the number of switches in each dimension. Interestingly, the maximum switch radix stays the same, only the

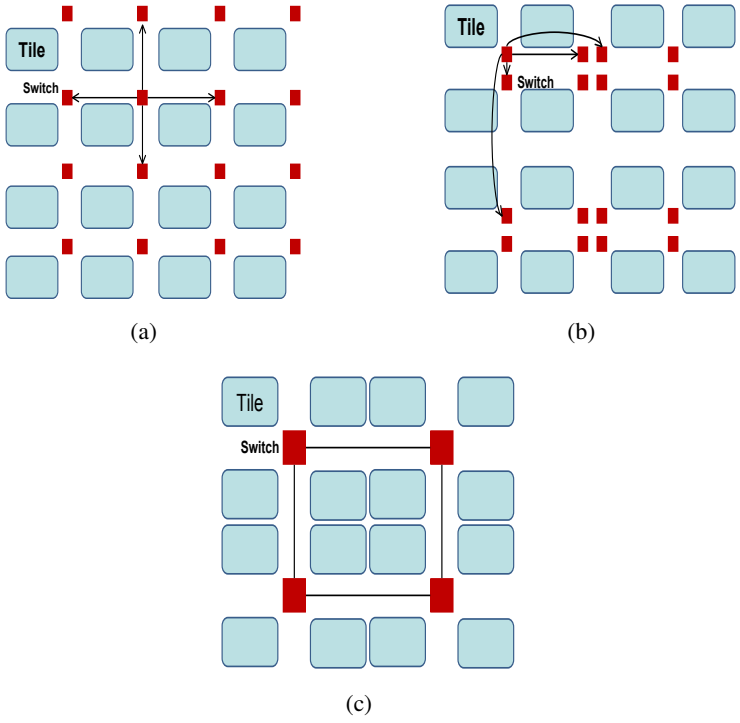


Figure 7.1: Floorplan directives for (a) the 2D mesh, (b) the 4-hypercube and (c) the 2-ary 2-mesh.

4-hypercube has all the switches with the same radix while the 2D mesh has the central switches with a higher radix than the peripheral ones. Of course, the 4-hypercube has larger bisection bandwidth and lower network latency. Unfortunately, this comes at the cost of links with uneven length. In fact, in a mesh with more than two dimensions the links used to connect the dimensions greater than two (often denoted as *express links*) are longer, and this holds for 50% of the 4-hypercube interswitch links.

In theory, the length of a link of the dimension t is generally assumed to be $k^{(d-2)/2}$, where d is equal to t if t is an even number and d is equal to $t + 1$ when t is odd. Unfortunately, placement and layout constraints put this picture in discussion, thus making it very difficult to predict the impact that the cost of a specific link performance boosting technique might have on the cost metrics of the entire topology.

As an example, let us consider the floorplanning directives given for topology

synthesis of the networks reported in Figure 7.1(a), Figure 7.1(b) and Figure 7.1(c). The asymmetric tile size plays in favor of the 4-hypercube wiring, since the length of the horizontal and of the vertical express links turns out to be comparable to that of horizontal wires in the 2D mesh. This latter also features horizontal and vertical links of unequal length, indicating that the layout regularity often assumed in high-level considerations does not materialize in practice. Our guiding principle for floorplan definition consists of *shortening the longest links in each topology* and defining a scalable floorplanning style. For the 2-ary 2-mesh, we placed the computation tiles around the switch they are attached to, which is ideal scenario for pipelining interswitch links. In all cases, network interfaces were placed close to their tile but also to the connected switch, so to move the critical path away from these critical links.

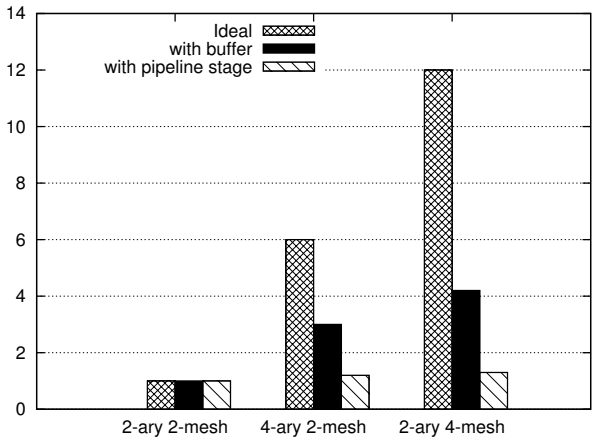


Figure 7.2: Total wire length.

As a result of topology synthesis and place-and-route, Figure 7.2 shows the total wiring length for the three topologies (*Post-Layout* curve), normalized to the least wire-hungry topology. It is compared with the results of traditional pencil-and-paper floorplanning considerations. Curve *Ideal* computes wire length based on the ideal formula given above, which only considers the number of hops crossed by a wire. Curve *Floorplan-aware* updates the previous formula with the knowledge of the asymmetric tile size and of switch placement. The ideal analysis largely overestimates the amount of wiring needed for the 4-hypercube. Floorplan awareness allows to account for specific floorplanning techniques that optimize wiring of a given topology, and therefore leads to more conservative estimations of the wiring overhead. However, this is still far away from real-life, where the post-layout report of total

Topology	4-ary 2-mesh	2-ary 4-mesh	2-ary 2-mesh
Post-Synthesis (WC Switch))	1 ns	1 ns	1.15 ns
Post-P&R (WC Switch)	1.12 ns	1.12 ns	1.4 ns
Post-P&R repeater-less (Topology)	1.27 ns	1.56 ns	1.67 ns
Post-P&R with buffers (Topology)	1.19 ns	1.56 ns	1.5 ns
Post-P&R with pipeline stages (Topology)	–	1.19	1.42

Table 7.2: Timing results.

wire length gives only a 10% overhead of the 4-hypercube wiring with respect to 2D mesh one and a 43% with respect to the 2-ary 2-mesh. This is because switch-to-switch and switch-to-network interface wiring only accounts for a relatively small percentage of total wiring, ranging from 7% for the 2-ary 2-mesh to 26% for the 4-hypercube. This explains the relatively small total wire length difference between the different topologies. This scenario plays in favor of engineering performance-optimized interswitch links with a possibly minor impact on topology cost metrics.

7.3 Link Design Techniques

Three fundamental link inference techniques were studied in this work and an associated topology was laid out for each of them. When moving from one technique to another, the objective was to speed up a topology by speeding up its links. Therefore, our primary design objective was high-performance.

In our first round of topology implementations we forced the inference of unrepeated links. In essence, we prevented the backend tools to instantiate buffers or inverters along the link. Therefore, the only degree of freedom for such tools was to prevent timing violations on the links by inferring a driver of suitable driving strength in the switch output ports.

The first performance boosting strategy was to allow repeater insertion along the link. While an increase of instantiated buffers can be expected, the lower driving strength of switch output gates partially offset the added cost. In any case, a significant cut down on link delay is expected.

Finally, we applied link pipelining to break long timing paths across interconnects. Synthesis tools do not provide a native support for this, thus calling for additional design effort. Given the small scale of the topologies under test,

our approach was not to manually place pipeline stages in the floorplan, but to let the tool handle this based on design constraints and optimization directives. We expect this to be a better approach than blindly placing these stages in the middle of long links, since the tool has better visibility of the floorplan constraints and design rules. When implementing link pipelining, flow control issues need to be considered. In our architecture, a backward propagating *stall* signal has to be retimed as well for each link. As a result, our pipeline stages are not simple registers but true retiming and flow control stages, consisting of 2 slot buffers and a control logic. This prevents from oversizing input buffers of downstream switches to avoid data loss.

7.4 Experimental Results

7.4.1 Timing Closure

For all cases, we had to impose a target performance. Since we are heading for high-performance, our goal was to materialize, after topology place-and-route, the maximum speed achievable by the slowest NoC module (characterized in isolation with post-layout timing analysis). In our architecture, the critical module turns out to be always the switch with the highest radix in the topology. The resulting speed upper bound, which ignores the effect of interswitch links, is reported in the first two rows in Table 7.2. For the slowest switch of each topology in isolation, the gap between post-synthesis and post-P&R speed ranges between 12 to 21%. The 2-ary 2-mesh exhibits a higher post-synthesis value due to the larger max. switch radix and a more severe post-P&R degradation due to the larger switch area compared with non-concentrated topologies.

When we consider timing closure for the entire topologies, then the degradation associated with interswitch link routing and with their synthesis techniques becomes apparent. Let us consider repeater-less links first (Table 7.2, third row). While 2D mesh and 4-hypercube had the same performance upper-bound (since they have switches with the same maximum radix), the more challenging routing of the 4-hypercube gave rise to a 39% degradation of the critical path, while routing of the 2D mesh turn out to be less critical. The 2-ary 2-mesh has some links longer than 4mm, therefore interswitch routing has an even more relevant impact. This also hides the effect of the higher switch radix, which is not the ultimate responsible for the slower operating frequency. For all topologies, the critical path goes through the network links.

Although post-layout effects of interswitch wires paint a dismal picture of

topologies using long wires (because of the use of more dimensions or of the switch separation in the layout), designers can optimize wires to overcome these large delays. Activating repeater-insertion during topology synthesis enables the speedups illustrated by the fourth row of Table 7.2, denoting the best critical delay at which timing closure was achieved. Results are quite heterogeneous. The 2D mesh further benefits from repeaters and achieves a 6% speedup of its critical path, which results 1.19ns. This value is quite close to the performance upper-bound (1.12ns), thus indicating that while links are still critical in this topology, their length is such that their degradation of topology performance can be made irrelevant.

The opposite holds for the 4-hypercube, where repeater insertion did not surprisingly provide any performance improvement. The reason for this lies in the fact that horizontal routing channels (see Figure 7.1(b)) were sized conservatively small, approximately 2.5x the switch side. For this reason, switch-to-switch links sometimes end up finding another switch on their way. This is a placement constraint for the buffers which prevents their insertion with ideal spacing. The result is that link performance does not improve, indicating that buffer insertion should not be taken for granted in NoC design, but should be carefully engineered to materialize its expected advantages. In practice, widening the routing channel ideally to 4x the switch side would solve the problem but would also lead to a large floorplan area overhead. Finally, no such constraints exist for the 2-ary 2-mesh (its connectivity pattern is trivial), which improves its baseline performance by 10%.

Further critical path improvements were expected from link pipelining. When applying this technique, our objective was to materialize the same critical delay of the 2D mesh (with buffers) even for the 4-hypercube and the 2-ary 2-mesh. Interestingly, we noticed that link pipelining was effective even for the 4-hypercube, regardless of its under-sized routing channels. In fact, the target critical delay of 1.19ns was achieved. This result clearly indicates the lower sensitivity of link performance to non-ideal pipeline stage placement compared with non-ideal repeater spacing. Hence, link pipelining proves more robust for area-optimized floorplans with more challenging placements.

Pipelining was effective also for the 2-ary 2-mesh, however the upper bound for its performance is the post-layout critical delay of its high-radix switches (1.4ns), far worse than the 2D mesh delay of 1.19ns.

7.4.2 Implementation Cost

In order to assess the implementation cost for link performance boosting techniques, we illustrate the area reports in Figure 7.3. Results are grouped by topology and normalized to the baseline implementation of each topology.

Buffer insertion is quite cheap for the 2D mesh and the 4-hypercube, while generates a significant area overhead for the 2-ary 2-mesh. This is due to the backend tools, that have dealt with the performance maximization of long links by dramatically increasing the number of buffering gates. As will be explained later on, this implies also a relevant power cost. However, when pipeline stages are inferred, the additional area overhead of the 2-ary 2-mesh is marginal. This due to the fact that by inserting pipeline stages the tool was able to remove an equal amount of buffering area, so that the two contributions offset each other.

This does not hold for the 4-hypercube, which raises its area by 14% when pipelining is used. Not only the area overhead of the pipeline stages is incurred, but many buffers are kept in the links since the frequency boost is significant when moving from buffering to pipelining. The trend of leakage power fully reflects that of area overhead and is therefore omitted for lack of space.

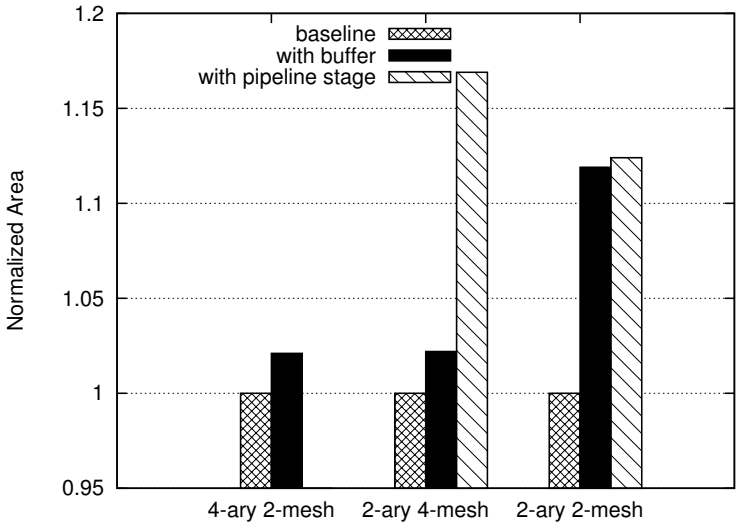


Figure 7.3: Normalized area.

7.4.3 Energy Efficiency

Although there is a price to pay to boost link performance in terms of area and power, the speedup in job completion can be exploited to cut down on total energy of the on-chip network. However, this energy saving materializes only if the gain in execution time outweighs the power overhead. This section sheds light on this aspect. Moreover, it is also investigated whether the different link synthesis techniques can change the relative energy ratios between topologies.

Experiments were carried out with a parallel synthetic benchmark consisting of one producer task, 14 worker tasks and 1 consumer task. Every task is assumed to be mapped on a different computation tile. The producer task reads in data units from the I/O interface of the chip and distributes it to the worker tasks. Output data from each worker tile is then collected by a consumer tile, which writes them back to the I/O interface. During computation, all-to-all accesses are generated to account for a cooperative computation process. Transactions are generated in compliance with the OCP protocol (burst accesses) by programmable OCP traffic generators [114]. We assume loose synchronization between the cores. In fact, we assume that whenever a producer has data available for a consumer, it sends them across the network without any previous check for consumer availability to accept the message. Similarly, we allow different producers to send messages to the same consumer at the same time, thus generating more conflicts in the network. Traffic generation rates were set to have a balanced system operation, i.e. to avoid I/O bottlenecks which would stress other design issues other than network bandwidth and link performance.

Real elapsed time results for the baseline repeater-less topologies along with their respective buffered and pipelined variants are reported in Figure 7.4. For each variant, the maximum achievable post-P&R speed is applied (see Table 7.2). Since our network interfaces use a lightweight frequency-ratioed synchronization mechanism, a clock divider of 2 is applied at all network interfaces, instructing the tiles to run at half the speed of the network and slowing down the tiles as well when the network speed is low. Exploration of this parameter is outside the scope of this thesis.

The following indications come from Figure 7.4. Although the 4-hypercube reports a lower clock cycle count to complete the benchmark, its execution time is always higher in the baseline topologies and in the repeated ones since its running speed is much lower than the 2D mesh. A 16-tile system is too small for an hypercube to take full advantage of its better properties (bisection bandwidth, diameter) and thus to offset the speed degradation associated with its relatively longer links. The situation is even worse for the 2-ary 2-mesh,

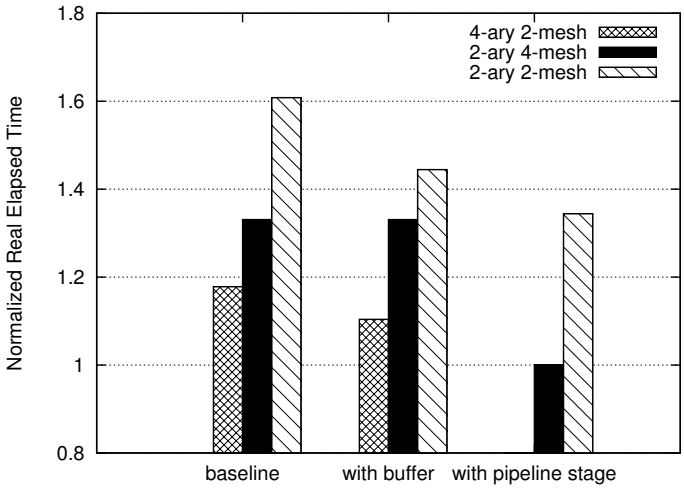


Figure 7.4: Normalized real elapsed time.

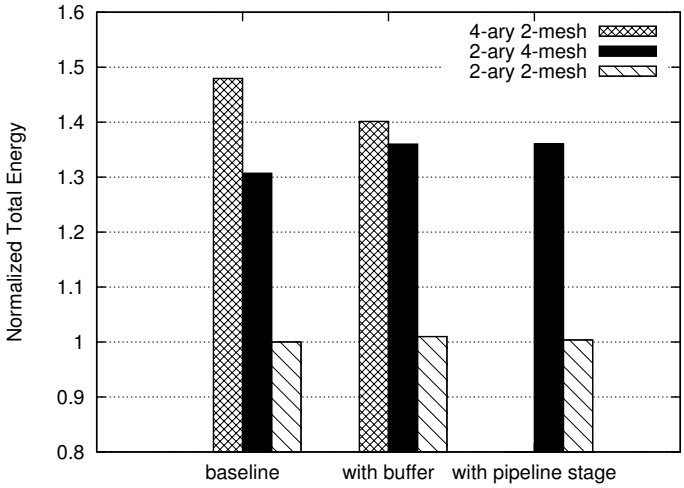


Figure 7.5: Normalized total energy.

which is worse than the other two topologies in the baseline and buffered variants both in execution cycles (because of poor bisection bandwidth) and in running speed (overly long links).

However, when the 4-hypercube can operate at the same speed of the 2D mesh thanks to link pipelining, than it becomes the most performance-efficient solu-

tion. The interesting thing is that link pipelining not necessarily adverses total performance. In this experiment, it enables a significant topology speedup and provides additional buffering to the topology itself, which is effective to handle a bandwidth-sensitive traffic pattern. The high switch radix of the 2-ary 2-mesh causes a saturation effect to the performance gain that this topology can achieve by means of link pipelining.

The power cost incurred to boost link and hence topology performance is illustrated in Figure 7.6. Power is affected by the operating frequency of each topology. If we compare topologies with boosted links with their baseline variants we observe that buffers do not cost a lot in terms of power. This holds for the 2D mesh and for the 4-hypercube, which is in line with the hardly relevant share of interswitch links over total wire length (as explained in Section 7.2). Power overhead of the 2-ary 2-mesh is an exception to this, due to the large power cost of driving a long link effectively.

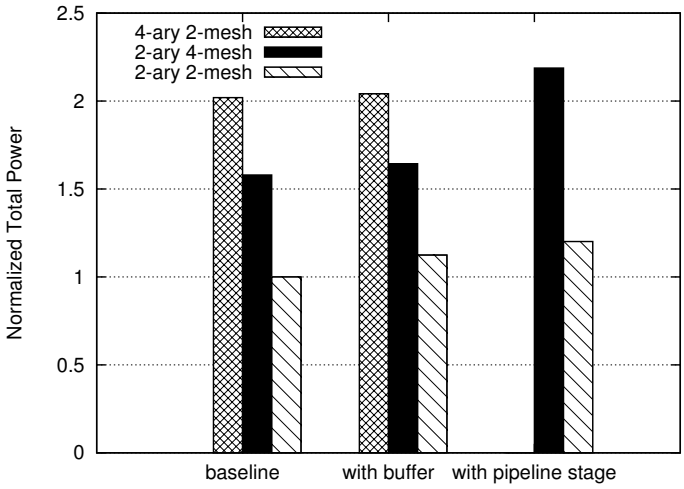


Figure 7.6: Normalized total power consumption.

When it comes to link pipelining, the power cost abruptly increases, especially for the 4-hypercube, while the overhead for the 2-ary 2-mesh increases more smoothly. Fortunately, this is also the scenario where the 4-hypercube gains more in terms of performance.

When we compare topologies with each other, we observe that the topology of choice when low-power is the primary design goal is the 2-ary 2-mesh. The lower power of the 4-hypercube with respect to the 2D mesh mostly derives from its lower speed, although this is not the only explanation. This is con-

firmed also by the marginal power overhead of the 4-hypercube when it can run at the same speed of the 2D mesh (see buffered 2D mesh vs pipelined 4-hypercube in Figure 7.6). This is counterintuitive, since the hypercube has many more buffering resources than the 2D mesh.

The answer has be sought in the power breakdown. This is reported in Figure 7.7 for the baseline version of the two topologies along with the low frequency variant of the 2D mesh. In fact, the 2D mesh was re-synthesized and analyzed both at full speed (rightmost column) and at the same speed of the 4-hypercube for a fair comparison. In all cases, it is evident that the clock tree has a relatively lower impact than register power in the 4-hypercube, while in the 2D mesh it weighs more. As a consequence, when the same speed is inferred, the two topologies have surprisingly the same power. While register power obviously increases for the 4-hypercube, the clock tree is cheaper, and this explains the result. The reason lies in the fact that while the 2D mesh has heterogeneous switches, the 4-hypercube has all switches with exactly the same radix. Hence, the clock tree is inherently more balanced and thus easier to synthesize while meeting skew constraints.

By combining the performance results of Figure 7.4 with the power reports of Figure 7.6, we get the energy results of Figure 7.5. Overall, buffering the links of a 2D mesh is always an energy-efficient strategy. On the contrary, 4-hypercube and 2-ary 2-mesh show minor energy variations when moving from one scenario to the other. This indicates that performance improvements have been achieved at the cost of a proportional power overhead.

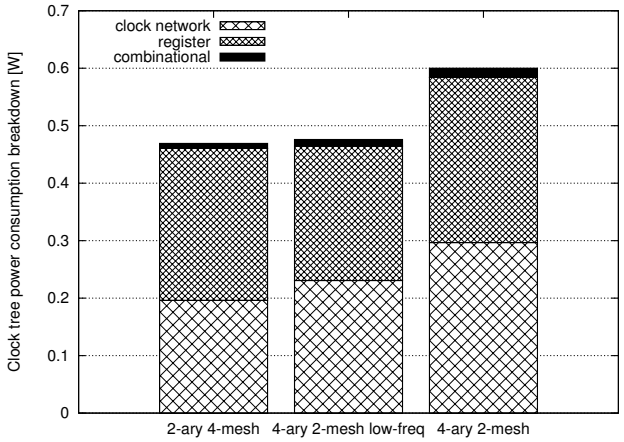


Figure 7.7: Clock tree power impact.

7.5 Discussion

The critical path of NoC architectures is moving once again to the interswitch links, thus making the synthesis technique of these links highly critical for overall topology speed. This chapter explored how repeater-less, repeated and pipelined links affect the critical path and the cost of a family of k-ary n-mesh topologies. As regards the achievable speedup, repeater insertion is very sensitive to floorplanning constraints. Therefore, a non-ideal repeater spacing might result in no performance boost. Widening routing channels is the obvious but costly workaround to overcome such limitation. In contrast, pipeline stage positioning is more flexible and the expected performance gain is always materialized in practice. Moreover, buffers do not cost much when applied to relatively short links (around 2.5mm in 65nm technology), while on longer links the backend tool attempts a performance optimization at a dramatic power (buffering) cost. In all other cases, the steep increase in area arises only when pipeline stages are inferred.

Considering such analysis, we found it always convenient to infer repeated interswitch wires in a 2D mesh, since the performance speedup effect is prevailing. As regards the 4-hypercube, it really needs large scale systems to take advantage of its bisection bandwidth and low diameter. However, already for 16 tile systems, when it can be operated at the same speed of the 2D mesh (by means of link pipelining) it provides much better performance with equal if not lower energy. For the 2-ary 2-mesh, the high switch radix poses a saturation limit to the speedups that link optimization techniques can provide. It is anyway the technique of choice when low-power is the primary design goal. In any case, whenever the topology has already been selected, in general boosting link performance pays off in terms of total topology performance while only minor or even no energy overhead is incurred. In cases when the connectivity pattern becomes intricate, the validity of these considerations becomes very sensitive to layout constraints.

7.6 Summary

In this chapter several NoC link design techniques have been assessed taking the system-level perspective. In this context, classical metrics such as performance, power and area overhead have been evaluated not only for a single link in isolation but considering the entire network topology thus showing their sensitivity to the adopted link design strategy.

8

A Methodology for Assessing Large Scale Systems with Layout-Awareness

THIS chapter extends the work presented in the previous chapters by proposing an evaluation framework for assessing topology implementation cost when scaling to 64-tile systems. Furthermore, we capture the impact of link pipelining on topology area and performance assessing whether and to which extent theoretical benefits are preserved.

The first part of the chapter carries out a high-level analysis that describes high-level properties of the investigated topologies. Therefore, we quantify to which extent such properties are impacted by the degradation effects of the physical synthesis on nanoscale silicon. Next, we propose a NoC physical characterization methodology enabling layout aware analysis of large scale systems pruning time and memory requirements. Moreover, we capture the impact of link pipelining on topology area and performance.

8.1 Introduction

Nowadays, system designers have adopted Networks-on-Chip as communication infrastructure of general-purpose tile-based Multi-Processor System-on-Chip (MPSoC). Such decision implies that a certain topology has to be selected to efficiently interconnect many cores on the chip. To ease such a choice, the networking literature offers a plethora of works about topology analysis and characterization for the off-chip domain. However, theoretical parameters and many intuitive assumptions of such off-chip networks do not necessarily hold when a topology is laid out on a 2D silicon surface. This is due to the distinctive features of silicon technology design pitfalls. The open literature features frameworks able to evaluate network topologies only from a pure theoretical

viewpoint thus neglecting all the physical effects of nanoscale technologies. On the other hand, other works focused on the physical modeling of interconnection networks but only limited to small scale systems mainly due to the unaffordable time and memory requirements for the synthesis of such systems. This is the motivation that pushed our work presented in the previous chapters.

Unfortunately, considering a system scale larger than 16 tiles, the feasibility of such analysis is jeopardized. In fact, due to the increasing memory requirements as well as the to the very long synthesis runs, the assessment of a large scale network topology with layout accuracy becomes an overwhelmingly difficult if not impossible task.

Therefore, we extend our work framework by proposing a methodology for analyzing large scale systems. With this methodology, only a few selected synthesis runs for each topology are required to characterize its delay and area.

Summarizing, the first contribution of this chapter consists of:

- an area and network critical path modeling framework able to accurately analyze performance of k-ary n-mesh and C-mesh topologies with layout awareness. Our proposed methodology scales easily to large size systems as only a few sub-systems of the whole network need to be analyze to draw comprehensive area and performance figures.

When accounting for layout effects, conclusions drawn from high-level theoretical analysis can be highly misleading. Moreover, k-ary n-mesh and C-mesh topologies suffer of a considerable slow down when laid out on silicon. This is mainly due to their long links which represent the speed bottleneck of the whole network. To tackle this problem, pipeline stages are typically implemented in links of the top dimensions. To the best of our knowledge, all the previously published analysis frameworks do not account for the implications of using such techniques from an area/timing viewpoint with layout awareness. Therefore, the second contribution is:

- the enhancement of our modeling framework with the capability of accurately capture the impact of using link pipelining from both the area and timing viewpoint accounting for physical effects. Interestingly, when considering such layout implications of using link pipelining, some topologies previously considered low speed turn out to be competitive.

The chapter is organized as follows. Section 8.2 presents the topologies under analysis from a high-level viewpoint discussing their abstract properties. Sec-

tion 8.3 describes the modeling methodology utilized to characterize the investigated topologies and reports performance and area results with and without pipeline stages.

8.2 High-level Topology Exploration

Topology	Switches	Cores/ switch	Max. degree	Unidir. links	Bisection bandwidth	Hop count	Connect.
8-ary 2-mesh	64	1	5	224	16	14	2
4-ary 3-mesh	64	1	7	288	32	9	3
4-ary 2-mesh	16	4	8	48	8	6	2
2-ary 6-mesh	64	1	7	384	64	6	6
2-ary 5-mesh	32	2	7	160	32	5	5
2-ary 4-mesh	16	4	8	64	16	4	4
8-cmesh	64	1	5	256	32	8	4
4-cmesh	16	4	8	64	16	4	4

Table 8.1: High level parameters of topologies with 64-tile.

In this section a high-level comparison of topology performance is provided. However, this analysis will only give the high-level perspective and is agnostic of physical implementation effects. Nonetheless, it may be used in early system design stages to select the most promising subset of topology candidates.

We restrict our focus to large 64-tile systems. The number of cores attached to each switch has been limited to four as a higher number of connected cores would introduce serious performance and feasibility issues. In fact, the topology would have a very low bisection bandwidth. Moreover, the placement of cores around the switches would not be a trivial task since the length of the injection/ejection links would increase. This would significantly limit overall NoC performance [155].

Table 8.1 summarizes the values of the properties of all 64 cores configurations considered for each topology. The analysis includes two different configurations of the CMesh network. From a pure topology viewpoint, a CMesh can be seen as a classical 2-D mesh with express links, regardless of the number of cores attached to each switch. As the investigated systems sizes are quite large, several topology configurations are possible and need to be taken into account. The best solution for high traffic loads is represented by the 2-ary 6-mesh. Moreover, this topology has one of the lowest hop counts (6), thus making it well suited for latency sensitive systems and applications. However, it requires the highest amount of resources: 64 switches of degree 7 and 384

unidirectional links. On the other hand, from a low-latency viewpoint, the best solution is either the 2-ary 4-mesh or the 4-cmesh, which again are completely equivalent from a high-level view-point.

Overall, the best topology would be the 2-ary 6-mesh, as it provides four time more bisection bandwidth than the low-latency solutions, while requiring only two hops more (6 hops in the 2-ary 6-mesh versus 4 hops in both low-latency solutions). The only drawback of such topology lies in the high number of required resources. Finally, when system specifications do not require such a high bisection bandwidth, the 2-ary 5-mesh solution becomes a good trade-off that provides twice the bisection bandwidth of the low-latency solutions (while increasing the number of hops by one). Clearly, by blindly relying on this table and upon the underlying theoretical analysis, a designer would easily discard the 2D mesh (8-ary 2-mesh) as candidate topology.

The remainder of this chapter will demonstrate that theoretical properties of such topologies are put in discussions when layout considerations are taken into account and may even lead to counterintuitive final results.

Next section will present the characterization methodology that is at the core of our modeling framework. Such methodology will be used to extrapolate key physical parameters to be back-annotated in the transaction-level simulator, thus enabling a layout-aware system-level exploration.

8.3 Physical Modeling Framework

The ×pipesLite [92] switch was used as the basic building block to construct the 64-tile topologies under test. However, exploring the design space of topologies with such a large number of cores with full physical synthesis proved impractical due to synthesis time and memory capacity requirements. Therefore, next section will present a way to prune the number of physical synthesis tests while still characterizing the full topology with high accuracy.

All the analyzed topologies of this work have been laid out by means of a backend synthesis flow leveraging industrial tools. The topology specification is fed to the *xpipescompiler* tool [152], resulting in the generation of self-contained SystemC code for RTL-equivalent simulation and for synthesis. Synopsys Physical Compiler is used for placement-aware logic synthesis. The technology library is a low-power low-Vth 65nm STMicroelectronics library available through the CMP project [146]. Placement and routing have been performed with Cadence SoC Encounter.

8.3.1 Characterization Methodology

In order to accurately characterize the switch and link buffering cell area of the topology under analysis, we propose to utilize the methodology depicted in Figure 8.1. In fact, as already reported in Chapter 6, the performance bottleneck of a topology lies in its longest switch–to–switch communication channel. Aware of this, from a high-level topology specification we build a subsystem composed of two communicating switches at the maximum possible distance in the topology. This way, the critical link delay can be extracted. Such delay (which is the critical path delay of the network) is then used as the *target delay* to re-synthesize, place and route all the possible switch–to–switch sub-systems for each different inter-switch link length. The reason for this is that our goal is to accurately capture the switch cell area at a certain distance and at a certain target speed. It is well known from logic synthesis theory that as the target speed is decreased, large area can be saved. In this direction, it would make no sense to synthesize switches for maximum performance when a long link limits overall network speed (unless decoupling techniques like link pipelining are used, as we will see later on). Please note that each switch of the built sub-system has been pre-characterized standalone with the input/output delay that is able to tolerate from its neighbor communicating block. These parameters were set in order to optimize the link delay as much as possible thus shortening the critical path of the switch–to–switch modeling architecture.

With this methodology, only a few selected synthesis runs for each topology need to be performed to characterize its delay and area as a whole. The approximation lies in the availability of enough routing channels for regular routing of NoC links and in the balance preservation of relative wire delays in links that undergo bending in the actual layout.

Moreover, with this method we are also able to capture the link buffering cost, in fact, by leveraging the report of the utilized physical synthesis tool, we are able to trace the inferred buffers of the switch–to–switch channel. In order to be as accurate as possible when characterizing a topology, two communicating ports of both switches in our subsystem were left unconnected. They are the ports connecting to the processing cores, which are typically placed close to their switch and therefore feature minimum capacitive load. Should we fail to model this (even by simply leaving an output port unconnected), the input and output buffer of the switch would be incorrectly sized by the synthesis tools by using a larger driving strength instead of that actually needed for the switch–to–core links.

A further step of our work is the estimation of the number of required pipeline

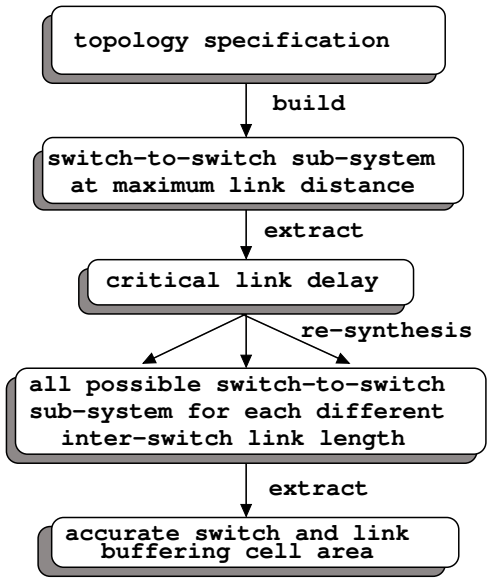


Figure 8.1: Characterization methodology flow.

stages for each link to speed up a topology. For this purpose, such retiming stages are instantiated along the communication link thus breaking the switch–to–switch critical path. By incrementing the number of pipeline stages, we were able to achieve timing closure bringing back the critical path to the second link dimension. In fact, as mentioned later, in order to limit area overhead, our pipeline stage insertion criteria consisted of adding such stages only from the third link dimension onwards.

The next section starts by commenting physical synthesis results achieved for 64-tile topologies without link pipelining. Consequently, the analysis is shifted to pipelined systems. Chapter 9 will utilize the obtained physical results to carry out a system-level exploration with layout-awareness.

8.3.2 64-tile topologies

As reported in Table 8.2, the range of possible switch radix per topology spans from a reasonable 6 to a large 12 that is even more difficult to place and route as a stand-alone block without DRC (design rules check) violations [108]. Post-synthesis frequency results reflect the increasing trend with the switch radix, in agreement with the analysis of [108]. After placement and routing, the ef-

fect of the long links comes noticeably into play. Most of the topologies suffer from long switch–to–switch channels that need to be routed along the chip. For the sake of the analysis, only the longest link per topology is reported in the 5th column. By comparing such column with the 4th one, it is possible to recognize a clear correlation between the increasing link length and the decreasing operating speed of the topology under analysis. In fact, the critical role of the interconnect is a major factor limiting the performance of a topology. It should also be observed that also some logic gates end up in series to the critical links close to the far-ends. They are associated with flow control management and further contribute to the critical path delay. The trend above is even more apparent when we consider larger topologies. In fact, only topologies with short links (e.g., 8-ary 2-mesh and 4-ary 2-mesh) can work at a reasonable frequency for realistic application scenarios.

TOPOLOGY	Radix	post-synthesis frequency	post-P&R frequency	longest link
8-ary 2-mesh	6	1.08GHz	890MHz	1.5mm
8-cmesh	6	1.08GHz	250MHz	6.75mm
4-ary 3-mesh	8	950Mhz	220MHz	6.9mm
2-ary 6-mesh	8	950Mhz	220MHz	6.9mm
2-ary 5-mesh	9	810MHz	230MHz	6.96mm
4-ary 2-mesh	12	720MHz	530MHz	3.0mm
2-ary 4-mesh	12	720MHz	260MHz	6.4mm
4-cmesh	12	720MHz	260MHz	6.4mm

Table 8.2: Post-place&route results of the 64-tile topologies under test.

From the area viewpoint (see Figure 8.2), it is interesting to note that this result is influenced by the combination of many parameters such as: number of switches in the topology, their radix and consequently their final working frequency. In fact, as explained above, in order to be accurate, all representative switches in every topology have been re-synthesized at the final working speed of the whole network.

As an example, let us consider a very slow topology like 2-ary 6-mesh that features a larger area footprint with respect to the 8-ary 2-mesh. Such a network is operating at a frequency much slower than the 8-ary 2-mesh, but since it has an equal number of switches (64) with a higher radix (8 vs. 4, 5 or 6), the overall area figures plays in favor of the 8-ary 2-mesh with a 10% saving.

Another interesting result concerns the 4-ary 2-mesh. This topology has a relatively short link (3mm), thus it does not suffer from a large speed degradation

after place-and-route. As reported in Table 8.2, this topology is the only one (along with 8-ary 2-mesh) to have a final working speed above 500MHz. Interestingly, the area footprint of such topology has a 20% saving with respect to the 8-ary 2-mesh as it only has 16 switches. Although their radix is 10, 11 and 12, their final working speed along with the number of their instances results to be more area effective than the 8-ary 2-mesh counterpart.

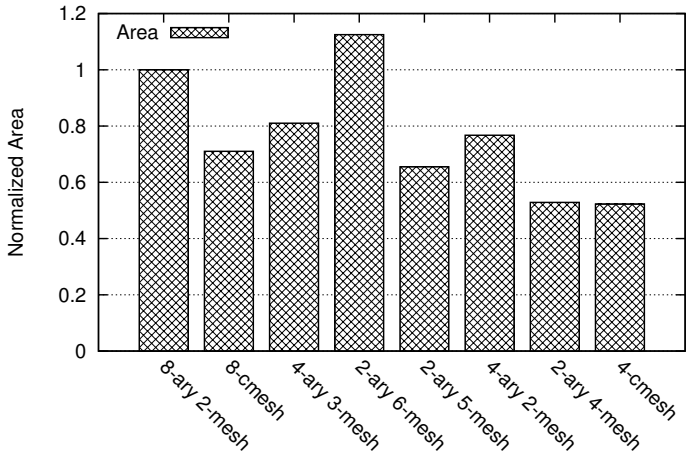


Figure 8.2: Normalized area for 64-tile topologies.

The overall conclusion is that most of the topologies are not competitive with the 8-ary 2-mesh because of their long links that influence the final working speed. A natural way to tackle this problem is to implement link pipelining on such long links but the policy of insertion has to be carefully engineered. In fact, the studied 64-tile topologies feature a high number of long links that could rapidly bring the area cost to an unaffordable budget for a SoC.

8.3.3 Pipeline stage insertion for 64-tile systems

In order to cope with the high speed degradation of most topologies analyzed in the previous section, pipeline stages need to be inserted especially in the top dimensions. By adding pipeline stages, it is possible to partially (if not completely) recover the initial operating frequency of the basic switch block. The criteria that has been adopted for the insertion of pipeline stages is to use them only from the third link dimension onwards. Therefore, topologies such has 8-ary 2-mesh and 4-ary 2-mesh have not been modified. Table 8.3 collects the results of this experiment. As clearly reported in the 3rd and 4th column,

the insertion of pipeline stages is a very effective way to reduce post-place and route frequency degradation. Column 5 reports the number of pipeline stages inferred in each link dimension whereas the 6th column points out the number of links of each topology. The area weight comes from the combination of these two factors and it is reported in the 7th column. Total cell area of the topologies along with the contribution of such retiming stages insertion is reported in Figure 8.3.

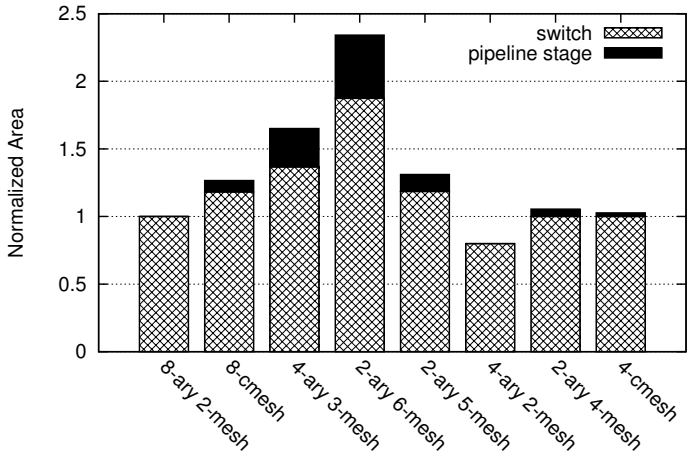


Figure 8.3: Normalized area for 64-tile topologies with pipeline stages.

Please note that the number of pipeline stages per link depends on the maximum achievable frequency (dictated by the maximum switch radix) along with the link length which is an intrinsic characteristic of each topology. As reported in Figure 8.3, the 2-ary 6-mesh is the most area greedy topology because it has the highest number of switches (64) and they were placed and routed at the high frequency of 855MHz. Moreover, this topology features 192 links with up to 5 pipeline stages on the longest interconnection channel. The key take away is that, for each topology, there is a different price to pay to restore the possible working frequency allowed by the elementary switch block. For this reason, Chapter 9 will introduce the throughput/area metric (or area efficiency) that provides a fair assessment of the cost of the achievable bandwidth in each topology (see Figure 9.5).

To conclude the physical implementation part, it is interesting to observe the result depicted in Figure 8.4. For each topology, this graph reports area results before and after inserting pipeline stages. Interestingly, the substantial cell area increment in all cases (except topologies where pipeline stages were

topology	radix	post-synthesis frequency	post-P&R frequency	# of pipe-stage per dimension	num. links	tot. pipe-stage area area (um2)	to.t switch area (um2)	impact of pipe-stage insertion on tot. switch area
8-ary 2-mesh	6	1.08GHz	893MHz	0	112	0	2327712.8	0%
8-cmesh	6	1.08GHz	893MHz	express link⇒4	128	193425.9	2752108.8	7.03%
4-ary 3-mesh	8	950MHz	855MHz	dim.3⇒4	144	660216.3	3182953.2	20.74%
2-ary 6-mesh	8	950MHz	855MHz	dim.3,4⇒1, dim.5,6⇒5	192	1087918.1	4362092.8	24.94%
2-ary 5-mesh	9	810MHz	562MHz	dim.3⇒1, dim.4,5⇒3	80	293081.6	2758480.4	10.62%
4-ary 2-mesh	12	720MHz	532MHz	0	24	0	1860718.3	0%
2-ary 4-mesh	12	720MHz	532MHz	dim.3,4⇒3	32	125574.7	2328426.4	5.39%
4-cmesh	12	720MHz	532MHz	express link⇒3	32	62787.4	2328426.4	2.69%

Table 8.3: Post-place&route results of 64-tile topologies with pipeline stage insertion.

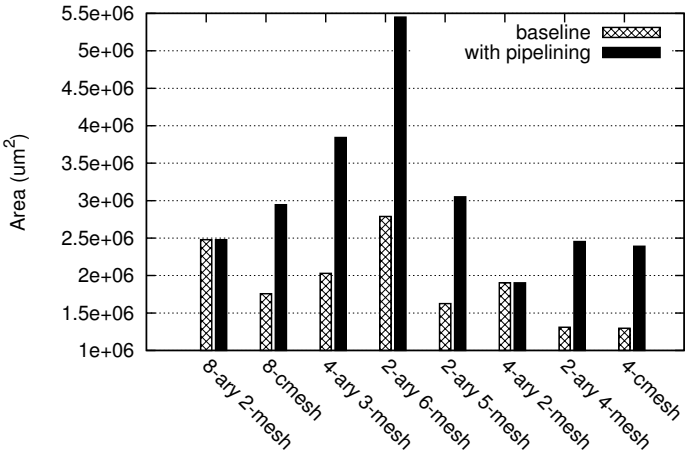


Figure 8.4: 64-tile topologies area overhead for pipeline stage insertion.

not inserted) comes from a twofold contribution: pipeline stages insertion (as discussed so far) and the restored higher frequency allowed by such insertion. In fact, the largest contribution in terms of area comes from the switch cell area devoted to achieve the new working frequency of the switch block. This relevant effect is typically overlooked by vast majority of topology exploration frameworks in the open literature.

8.4 Summary

In this chapter we presented a comprehensive analysis framework to assess k-ary n-mesh and C-mesh topologies at different levels of abstraction, from system to layout level. Our framework leverages an accurate physical characterization methodology that allows to characterize various topologies from the area and timing viewpoint while pruning the implementation time as well as memory requirements. All the topologies have been evaluated at physical level and their key parameters have been back-annotated for use in a transaction-level simulator that performs layout-aware system-level exploration. The latter has been enhanced with dual-clock FIFO interfaces to allow fully decoupled working frequencies between cores and the NoC.

This chapter demonstrated that is possible to evaluate large scale topologies with physical level accuracy while cutting down the analysis time by utilizing only a few selected synthesis and place&route runs. Furthermore, we demon-

strated that conclusions drawn by a pure high-level analysis of topology performance can be highly misleading if not enriched by the information provided by the physical synthesis. As an example, let us consider k-ary n-mesh, these are very difficult topologies to be realized without link pipelining and the implementation cost of using such technique is typically overlooked. To tackle this problem, our modeling framework has been devised in such a way that is able to capture the utilization of pipeline insertion from an area and timing viewpoint. Leveraging this capability, analyzed topologies, typically considered too slow, turn out to be cost-effective and may represent a valid alternative for a given implementation budget.

9

Large Scale GALS Systems Analysis

THIS chapter leverages the effort of the previous chapters and extends our topology analysis framework to consider GALS systems where the core and the network speed is fully decoupled. This last brick finally bridges the gap between the physical and the system level thus enabling a truly *technology-aware system-level assessment*. The chapter starts by detailing the experimental setup. Next, a first set of system-level experiments is carried out differentiating the analysis with pure high-level estimations only, layout-aware with and without pipelining. Last, in order to assess the real effectiveness of link pipelining techniques, an area efficiency metric is also considered.

9.1 Introduction

As described in the previous chapters of this thesis, GALS systems are gaining momentum for several reasons. One of them is the necessity of providing voltage and frequency decoupling between different islands of the same chip. Such capability is key in an application domain, such as the MPSoC one, where power dissipation plays a critical role. On the other hand, GALS systems are also very important as they represent an appealing solution to tackle and soften all the problems due to the upcoming synchronization challenge.

In the previous work presented in this thesis, we started by detailing all the architectural modifications required to build up a cost-effective GALS system along with the developed design flow to instantiate a GALS platform. Furthermore, we performed a cross benchmarking between two GALS systems. In the second part of this thesis, we took a system level perspective and we started looking at an accurate way of modeling and characterizing small scale and large scale network topologies where cores and network speed was constrained by an integer divider thus limiting the overall performance of the system.

In this chapter, we put everything together extending our analysis to systems implemented with the globally asynchronous locally synchronous (GALS) design style where cores and network speed ratio can be any. Interestingly, the adoption of such GALS approach has a considerable consequence on the performance and area figures of various topologies that were not competitive at all in the previously investigated scenarios. In order to achieve this objective, our transaction level simulator has been enhanced with dual-clock FIFO interfaces for cores and network frequency decoupling.

The chapter is a direct extension of the previous one, in fact, by leveraging the physical layout results of Section 8.3, we carry out a system-level exploration with layout awareness for 64-tile systems in Section 9.2.

9.2 System-level Exploration

This section will discuss the gap between high-level and realistic performance predictions, by comparing the former with layout-aware ones.

9.2.1 Experimental setup

In order to obtain accurate performance estimations, our TLM simulator, which is cycle accurate with the assumed RTL architecture, has been utilized. The clock domain crossing mechanism implemented in the original simulator was ratio based. In particular, tile frequency was forced to be an integer divider of NoC frequency. However, as discussed in the Chapter 8, when link pipelining is not considered, some topologies present severe critical path degradations. These low frequency topologies cannot remain competitive with ratio-based clock domain crossing mechanism, as it will have a direct impact over the speed of the processing cores (as discussed in Chapter 6). In order to allow a fair performance comparison between topologies with extremely different operating frequencies, the simulator was augmented with the implementation of a dual-clock FIFO interface thus enhancing the whole modeling framework. Last but not least, a dual-clock FIFO allows frequency decoupling between a core and the network node it is connected to. In all the cases, tiles are assumed to work at a frequency of 750 MHz.

9.2.2 Experimental results

Figure 9.1 depicts accepted traffic vs. average message latency for a uniform distribution of message destination for different topologies when considering high-level estimations. Obtained results reflect the conclusions drawn in Section 8.2 where the 2-ary 6-mesh proved to be the best solution when neglecting layout implications. Figure 9.2 shows the same analysis where each topology works at the operating frequency reported in the Table 8.2, without pipeline stages. By comparing Figure 9.1 against Figure 9.2, when link pipelining is not considered, there is a misleading gap between the performance predictions of the high-level analysis and the layout-aware one. In fact, while the theoretical results reported in Figure 9.1 claim that several topologies outperform the 8-ary 2-mesh, this latter topology is proved to be the best solution in the layout-aware results of Figure 9.2. In fact, there is a direct correlation between the operating frequency and the achieved system-level performance: the lower the operating frequency, the higher the average latency and the lower the maximum achievable throughput, regardless of the results obtained in the high level analysis. In practice, poor matching with silicon technology completely offsets the better theoretical properties of the topologies. However, when the impact of wiring complexity over the critical path is alleviated by using link pipelining techniques, different conclusions can be drawn.

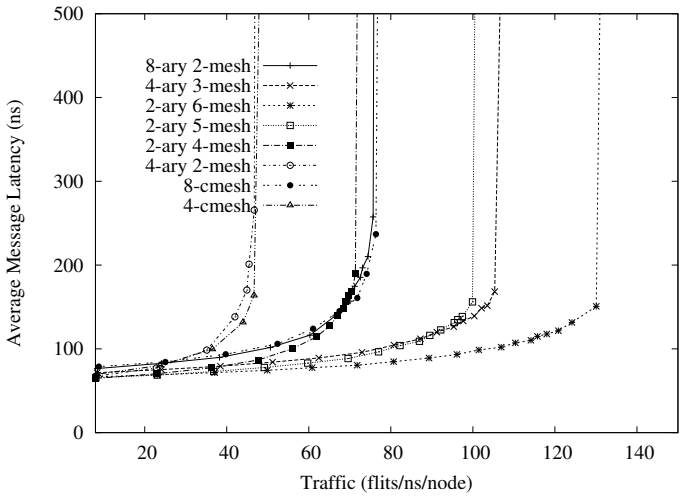


Figure 9.1: High-level estimation.

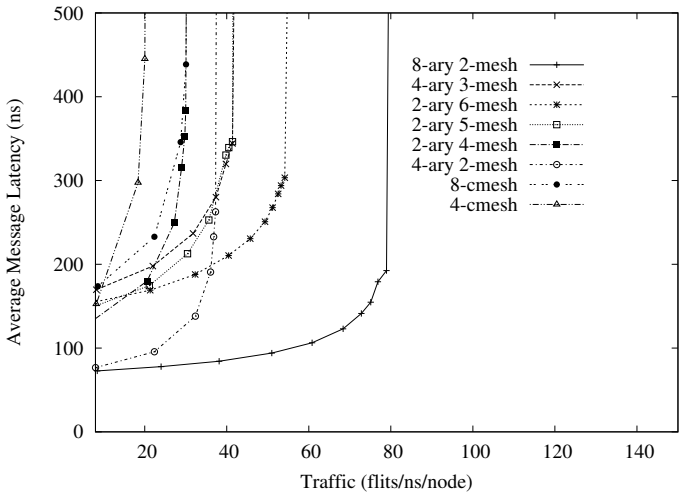


Figure 9.2: Layout-aware, no pipelining.

Figure 9.3 reports the same analysis results when each topology works at the operating frequency (see Table 8.3) enabled by the usage of link pipelining. In this case, there are three network topologies that clearly outperform the 8-ary 2-mesh: 2-ary 6-mesh, 2-ary 5-mesh and 4-ary 3-mesh.

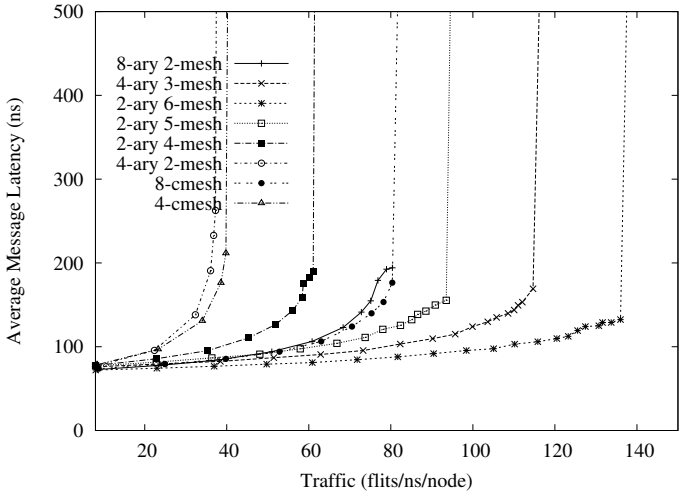


Figure 9.3: Layout-aware, with pipelining.

Similar curves have been drawn for several traffic patterns for each topology.

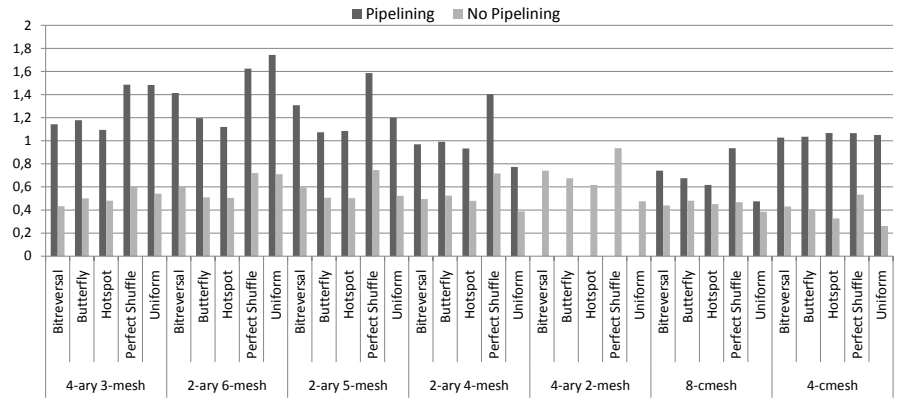


Figure 9.4: Normalized performance of 64-tile systems.

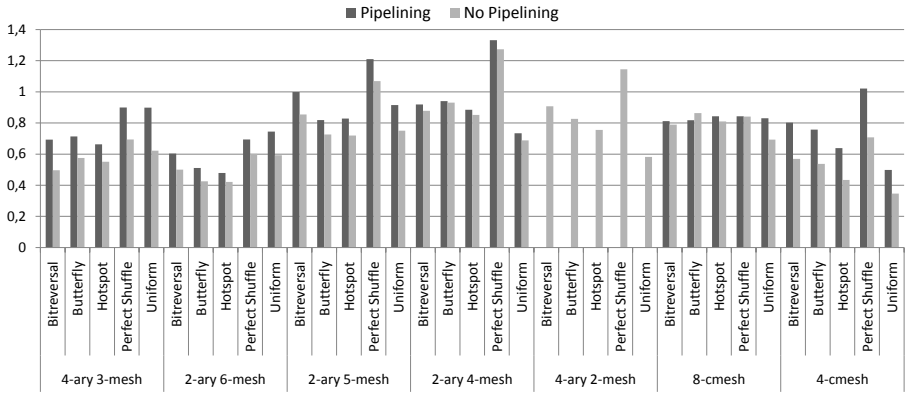


Figure 9.5: Normalized area efficiency of 64-tile systems.

Those results are summarized in Figure 9.4. This figure shows the normalized maximum throughput of each topology with respect to the 8-ary 2-mesh solution. In this plot, a bar higher than 1 implies an improvement of the maximum throughput over the 8-ary 2-mesh solution. Interestingly, those results follow the same trend as discussed for the uniform traffic pattern. All non-pipelined solutions are clearly worse than the 8-ary 2-mesh, while pipelined solutions follow the same trend reported in the high level analysis: most of the solutions outperforms the 8-ary 2-mesh, with the 2-ary 6-mesh being the best solutions for all the traffic patterns. Although in this case the obtained performance is closer to the high-level estimations, link pipelining techniques may have a

great impact over the implementation cost, thus requiring a new metric to asses the real effectiveness of link pipelining techniques.

In particular, we have considered the *area efficiency metric*, defined as *throughput/area*, which correlates the throughput improvement with the area cost that has been paid to achieve that. Results are shown in Figure 9.5, which depicts the area efficiency of each topology normalized with respect to that of the 8-ary 2-mesh. Results are reported with and without pipelining for several traffic patterns. In most of the cases, the area efficiency of both pipelined and non-pipelined solutions is clearly lower than the 8-ary 2-mesh solution.

The key take away is that the performance improvements achieved by complex topologies with pipelined links are not cost-effective but they are possible. In fact, it is up to the system designer, based on the target system requirements and on the available silicon budget, to opt for a more costly and high performance solution or to go for a slower but cost-effective alternative.

9.3 Summary

In this chapter, we extended our analysis framework for assessing large k-ary n-mesh and C-mesh topologies to consider also GALS features. In fact, key parameters from the topology exploration results have been back-annotated for use in a transaction-level simulator that performs layout-aware system-level exploration. The simulator has been enhanced with dual-clock FIFO interfaces to allow fully decoupled working frequencies between cores and the NoC which are typical of GALS systems. Utilizing such systems brings back momentum to topologies that were strongly limited by the constraint of using an integer clock divider between cores and network. This last brick of our simulation infrastructure bridges the gap between the physical and the system level thus enabling a truly *technology-aware system-level assessment*.

10 Conclusions

DIFFERENTLY from the first position papers, today NoC designers are facing the severe constraints imposed by the aggressive scaling of CMOS technology in the nanoscale era. On one hand, power consumption and synchronization issues are reaching hard limits. On the other hand, the complexity of large MPSoC along with an increasing time–to–market pressure demands for a communication infrastructure that is able to provide the required bandwidth, to solve the scalability issue and to furnish the necessary connectivity. NoC is generally believed to be the answer to such challenges. However, the low maturity of the NoC technology makes the designer's task still a daunting challenge. In fact, on one hand, the selection of the best connectivity pattern to interconnect all the system components is not obvious in spite of a plethora of work in the open literature. On the other hand, synchronization is still an open issue and will certainly worsen in the coming technology nodes thus calling for adequate architectural developments to counter its consequences. Overall, the common factor is the strong dependency of both issues from the implications of the aggressive technology scaling that must be accounted at each layer of the design hierarchy.

This thesis contributes to the evolution of the NoC concept into a mature technology by proposing a system-level analysis framework with layout awareness where technology insights, gained through different design layers, have been collected and exploited to conceive and customize all the design methodologies and architectural developments required to tackle the presented challenges.

This chapter is structured in three sections. Section 10.1 summarizes the work presented in this thesis. In Section 10.2, we present the major contributions of the thesis. Finally, Section 10.3 lists some future directions and open issues worth of further research and investigation in the context of NoCs.

10.1 Summary

This dissertation began by providing the necessary background of the work in Chapter 2. It surveyed topology evaluation frameworks as well as the design of globally-asynchronous locally synchronous interfaces for the building of GALS systems. Finally, it summarized the shortcoming of the presented work to be addressed in subsequent chapters.

Chapter 3 introduced the synchronization design issue. In a first step, the motivation for the adoption of synchronization mechanisms in the Network-on-Chip environment was discussed. Next, it was presented the target GALS platform of this thesis along with the architecture of the basic switch block required to build it. Last, the focus was on the baseline mesochronous synchronizer and all its improvements that led to a new fully integrated and flexible GALS switch architecture.

Chapter 4 moved a step forward towards the system-level. In fact, the design flow described in this chapter enabled the building of an entire GALS system from the system specification, through synthesis and CTS thus reaching the layout level by performing place&route. In particular, a complete Network-on-Chip design flow was presented in its front- and back-end parts. For both parts, our contribution to make the design flow suitable for building GALS systems was discussed.

Chapter 5 performed a cross-benchmarking between two GALS systems. The first leveraged a fully synchronous NoC while the second was implemented through a mesochronous NoC. While the former chapters focused on a switch-level analysis of such GALS systems, in this chapter, a network-level perspective was taken. Furthermore, both systems were compared from many viewpoints such as, clock tree power analysis, area and wiring overhead and above all from a skew tolerance and variability robustness viewpoint.

Chapter 6 explored the performance and physical feasibility of 16-tile Networks-on-Chip within several topology configurations. It was the first chapter where our "system- to layout-level" approach for assessing NoC topologies was presented. Our analysis framework encompassed different levels of abstraction as physical key parameters from synthesis and place&route process were calculated and then exposed to our system-level simulation infrastructure thus materializing in a system-level performance analysis with layout-awareness.

Chapter 7 assessed several NoC link inference techniques (e.g., repeater insertion, link pipelining) by means of commercial backend synthesis tools, taking the system-level perspective. Performance speed-ups and power overhead

were not only evaluated for the links in isolation but for the network topology as a whole, thus showing their sensitivity to the link inference strategy. Various k-ary n-mesh topologies were considered during our analysis as they provide a representative range of complex interconnection networks with increasing total wirelength.
Chapter 8 extended the work presented in the previous chapters by proposing an accurate characterization methodology for the evaluation of the topology implementation cost when scaling the system size to 64 tiles . The first part of the chapter carried out a high-level analysis describing the high-level properties of the investigated topologies. Therefore, we quantified to which extent such properties are impacted by the degradation effects of the physical synthesis on nanoscale silicon. Next, we proposed a NoC physical characterization methodology enabling layout-aware analysis of large scale systems pruning time and memory requirements. Moreover, we captured the impact of link pipelining on topology area and performance.
Chapter 9, leveraging the effort of previous chapters, considered IP core-network speed decoupling typical of GALS systems in the topology evaluation framework. We carried out a system-level exploration with layout awareness pointing out that there are several non-intuitive design opportunities depending on the available area and power budget.

10.2 Major Contributions

This section discusses the major contributions of our study.

- **A layout-aware topology exploration framework:** Neglecting layout implications and the distinctive features of silicon technologies would certainly lead to misleading conclusions when analyzing a network topology. Therefore, we proposed an analysis framework to assess topologies with layout awareness thus capturing the implications of physical effects at system-level. Moreover, we extended our framework with an analysis of the implications of utilizing link boosting techniques such as repeater insertion and link pipelining. Furthermore, we proposed a characterization methodology that enables the analysis of large scale systems pruning time and memory requirements. Our methodology captures also the impact of link pipelining on topology area and performance.

- **GALS design methods:** Among several implementation variants, we selected a source synchronous design style as the choice for implementing our target GALS platform. Moreover, in order to migrate from a synchronous to a GALS design style at a negligible area and power cost, we opted for mesochronous synchronization within the network domain. In light with this, we designed a novel mesochronous synchronizer that is fully merged with the switch input buffer. We explored the design space of the mesochronous link by characterizing the skew tolerance of such architecture as a function of several NoC parameters such as switch radix, interswitch link length, switch operating frequency. We integrated our developed GALS components into an industrial design flow thus easing the generation of GALS systems. We developed a clock tree synthesis methodology able to exploit power optimization opportunities in the clock distribution network. A final cross-benchmarking between two GALS systems: the first implementing a fully synchronous and the second implementing a mesochronous NoC has been carried out. Both systems leverage dual-clock FIFO interfaces to provide frequency decoupling between the NoC and the computational units. Such comparison characterized several important metrics such as skew tolerance, area and wiring overhead, system and clock tree power as well as robustness to process variation.

- **Technology-Aware System-level Assessment:** By combining the previous two major contributions, we extended our topology analysis framework to consider GALS systems where the core and the network speed is fully decoupled. This last brick finally bridges the gap between the physical and the system level thus enabling a truly *technology-aware system-level assessment*. In fact, technology information have been not only accounted across all the design layers for carrying out an accurate design space exploration, but, such cross-layer considerations of physical phenomena have been also utilized to conceive and customize all the design methodologies, circuits and the architectural developments presented in this thesis.

10.3 Open Issues and Future Directions

- **Architectural power optimizations:** we are aware that the adoption of a source synchronous design style has the drawback of sending an ever switching clock signal between two domains. This calls for an archi-

tectural optimization in the mesochronous interface where clock gating techniques could be implemented. Unfortunately, with the current synchronizer architecture, the number of latch banks does not suffice to support such clock gating techniques. Therefore, the interface architecture should be changed by increasing the number of slot buffers and a design space exploration to assess whether or not such modification pays off in terms of the overall system power saving should be also carried out.

- **Fault tolerant GALS NoC:** In our recent work on fault tolerant systems [156], we developed a self-testing infrastructure for Networks-on-Chip. The underlying assumption is that the system has a unique clock signal, thus, no GALS systems are supported. Since this testing strategy fully exploits a cooperative approach of all the network components, translating such methodology to GALS environment is not a trivial task but it is certainly worth investigating.

- **A power aware CTS methodology:** We have shown in our work that current EDA tools are able to relax the skew constraint only up to a certain point. The rationale is that such parameter is typically required to be as small as possible and thus the objective function of any design automation tool is to minimize it. Nonetheless, we are aware that a better controllability of the skew constraint might increase the power optimization in the clock tree while retaining the overall system functionality since mesochronous interfaces in the switch blocks would absorb an even larger skew offset. Implementing such CTS tool and investigating the consequent clock tree power saving opportunities would be of great interest for future multi-core chips where the clock distribution network is expected to be more and more power greedy and the overall synchronization assumption will have to be relaxed anyway in order to have a reasonable system feasibility range.

Bibliography

[1] L. Benini, G. De Micheli, "Networks on Chips: a New SoC Paradigm", IEEE Computer 35(1), pp. 70–78, 2002.

[2] A. Strano, D. Ludovici, D. Bertozzi, "A Library of Dual-Clock FIFOs for Cost-Effective and Flexible MPSoCs Design", Proceedings of the International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS), 2010.

[3] A. Pullini, F. Angiolini, D. Bertozzi, L. Benini, "Fault Tolerance Overhead in Network-on-Chip Flow Control Schemes". Proceedings of 18th Annual Symposium on Integrated Circuits and System Design (SBCCI), pp. 224–229, 2005.

[4] C. Leiserson, "Fat-trees: Universal Networks for Hardware Efficient supercomputing", IEEE Transactions on Computer, vol. 34, no. 10, 1985.

[5] S.R. Ohring, M. Ibel, S.K. Das, and M.J. Kumar, "On Generalized Fat Trees", Proceedings of the International Parallel Processing Symposium, pp. 37–44, 1995.

[6] C. Hernandez, F. Silla, J. Duato, "A Methodology for the Characterization of Process Variation in NoC Links", Proceedings of Design, Automation and Test in Europe (DATE), 2010.

[7] F. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, W. Fichtner, "GALS at ETH Zurich: Success or Failure?", Proceedings of Asynch'06, pp. 150–159, 2006.

[8] C. Grecu, P. P. Panda, A. Ivanov, R. Saleh, "Structured Interconnect Architecture: A Solution for the non-scalability of bus-based SoCs", Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI), pp. 102–195, 2004.

[9] M. Kreutz, C. Marcon, L. Carro, N. Calazans, A. A. Susin, "Energy and Latency Evaluation of NoC Topologies", Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 5866–5869, Vol. 6, 2005

[10] P.Guerrier, A.Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections", Proceedings of Design, Automation and Test in Europe (DATE), pp. 250–156, 2000.

[11] H. Matsutani, M. Koibuchi, H. Amano, "Performance, Cost and Energy Evaluation of Fat H-Tree: a Cost-Efficient Tree-Based On-Chip Network", Proceedings of IPDPS, pp. 1–10, 2007.

[12] H. Matsutani, M. Koibuchi, D. F. Hsu, H. Amano, "Three-Dimensional Layout of On-Chip Tree-Based Networks", Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks, pp. 281–288, 2008.

[13] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, C. A. Zeferino, "SPIN: a Scalable, Packet Switched, On-chip Micro-Network", Proceedings of Design, Automation and Test in Europe (DATE), pp. 1128–1129, 2003.

[14] S. Suboh, M. Bakhouya, S. L. Buedo, T. E. Ghazawi; "Simulation-Based Approach for Evaluating On-Chip Interconnect Architectures", Proceedings of the Southern Conference on Programmable Logic, pp. 75–80, 2008.

[15] A. Nalamalpu, W. Burleson, "Repeater Insertion in deep sub-micron CMOS: Ramp-based Analytical Model and Placement Sensitivity Analysis". Proceedings of the International Symposium on Circuits and Systems, pp. 766–769, 2000.

[16] S. Srinivasaraghavan, W. Burleson, "Interconnect Effort - A Unification of Repeater Insertion and Logical Effort". Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03), pp. 55, 2003.

[17] A. Youssef, T. Myklebust, M. Anis, M. Elmasry, "A Low-Power Multi-Pin Maze Routing Methodology". Proceedings of the 8th International Symposium on Quality Electronic Design (ISQED'07), pp. 153–158, 2007.

[18] M.R. Casu, L. Macchiarulo, "A new system design methodology for wire pipelined SoC". Proceedings of Design, Automation and Test in Europe (DATE), pp. 944–945, 2005.

[19] M.R. Casu, L. Macchiarulo, "Floorplanning With Wire Pipelining in Adaptive Communication Channels". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25(12):2996–3004, 2006.

[20] A. Jingye Xu Roy, M.H. Chowdhury, "Optimization technique for flip-flop inserted global interconnect". Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), pp. 3386–3389, 2008.

[21] H. Zhou, C. Lin, "Retiming for wire pipelining in system-on-chip". IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, 23(9):1338–1345, 2004.

[22] J. Cong, Y. Fan, Z. Zhang, "Architecture-Level Synthesis for Automatic Interconnect Pipelining". Proceedings of 41st Conference on Design Automation (DAC'04),

[23] U. Y. Ogras, R. Marculescu, H. G. Lee, C. Puru, D. Marculescu, M. Kaufman, N. Peter, "Challenges and Promising Results in NoC Prototyping Using FPGAs". IEEE Micro Special Issue on Interconnects for Multi-Core Chips, 27(5):86–95, 2007.

[24] L. P. Carloni, K. L. Mcmillan, A. L. Sangiovanni-Vincentelli, "Latency insensitive protocols". Proceedings of the 11th International Conference on Computer-Aided Verification, pp.123–133, 1999.

[25] L. Zhong, N.K. Jha, "Interconnect-aware low-power high-level synthesis". IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, 24(3):336–351, 2005. pp.602–607, 2004.

[26] Y. Ma, Z. Li, J. Cong, X. Hong, G. Reinman, S. Dong, Q. Zhou, "Micro-architecture Pipelining Optimization with Throughput-Aware Floorplanning". Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 920–925, 2007.

[27] A. Jingye Xu Roy, M.H. Chowdhury, "Analysis of Power Consumption and BER of Flip-flop Based Interconnect Pipelining". Proceedings of Design, Automation and Test in Europe (DATE), pp. 1–6, 2007

[28] P. Cocchini, "Concurrent flip-flop and repeater insertion for high performance integrated circuits". Proceedings of the International Conference on Computer Aided Design, pp. 268–273, 2002.

[29] L.P. Carloni, A.L. Sangiovanni-Vincentelli, "On-chip communication design: roadblocks and avenues". Proceedings of the IEEE International Conference on Hardware-software Codesign and System Synthesis, pp. 75–76, 2003.

[30] S. Heo, K. Asanovic, "Replacing global wires with an on-chip network: A power analysis". Proceedings of the International Symposium on Low Power Electonic and Design (ISLPED), pp. 369–374, 2005.

[31] N. Wu, Ge Fen, Wang Qi, "Simulation and Performance Analysis of Network on Chip Architectures using OPNET", Proceedings of the International Conference on ASIC, pp. 1285–1288, 2007.

[32] A. Adriahantenaina, A. Greiner; "Micro-Network for SoC: Implementation of a 32-port SPIN Network", Proceedings of Design, Automation and Test in Europe (DATE), pp. 1128–1129, 2003.

[33] C. Hernandez, A. Roca, F. Silla, J. Flich, J. Duato, "Improving the Performance of GALS-based NoCs in the Presence of Process Variation", Proceedings of the International Conference on Networks-on-Chip (NOCS), 2010.

[34] M. Mosin, R. Tamer, W. Xiang, A. Adnanm, M. Yehia, "Provisioning On-Chip Networks under Buffered RC Interconnect Delay Variations", Proceedings of International Symposium on Quality Electronic Design, 2007.

[35] M. R. Guthaus, D. Sylvester, R. B. Brown, "Clock Tree Synthesis with Data-Path Sensitivity Matching", Proceedings of the 2008 Asia and South Pacific Design Automation Conference (ASP-DAC), 2008.

[36] S. Borkar, "Thousand Core Chips: a Technology Perspective", Proceedings of the Design Automation Conference (DAC), pp. 746–749. 2007.

[37] C. Grecu, P. P. Pande, A. Ivanov, R. Saleh, "Structured interconnect architecture: A solution for the non-scalability of bus-based SoCs", Proceedings of the Great Lakes Symposium on VLSI, pp. 192–195, 2004.

[38] C. J. Myers et al., "Asynchronous Circuit Design", Wiley, 2001.

[39] S. Murali et al., "Designing Message-Dependent Deadlock Free Networks on Chips for Application-Specific Systems on Chips", Proceedings of the International Conference on Very Large Scale Integration, pp. 158–163, 2006.

[40] J. Sparso, S. Furber, "Principles of Asynchronous Circuit Design: A System Perspective", Kluwer, 2001.

[41] J. M. Rabaey, "Digital Integrated Circuits: a Design Perspective", Prentice-Hall, 2003.

[42] S. Herbert, D. Marculescu, "Analysis of Dynarmic Voltage/Frequency Scaling in Chip Multiprocessors", Proceedings of the International Symposium on Low Power Electronics and Design, pp. 38–43. 2007.

[43] A. P. Chandrakasan et al, "Low Power CMOS Digital Design", IEEE Journal of Solid State Circuits, Vol. 27, pp. 437–484, 2007.

[44] V. Khandelwal, A. Srivastava "Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 27, Issue 4, April 2008.

[45] K. Nagaraj, S. Kundu, "A Study on Placement of Post Silicon Clock Tuning Buffers for Mitigating Impact of Process Variation", Procedings of Design, Automation and Test in Europe (DATE), 2009.

[46] ARM Ltd., "AMBA AHB Overview", `http://www.arm.com/products/solutions/AMBA_Spec.html`, 2005.

[47] ARM Ltd., "AMBA 3 AXI Overview", `http://www.arm.com/products/solutions/AMBA3AXI.html`, 2005.

[48] J. Tsai, L. Zhang, and C. Chen, "Statistical Timing Analysis Driven Post-Silicon-Tunable Clock-Tree Synthesis", Proceedings of ICCAD, 2005.

[49] A. J. Martin, M. Nystrom, "Asynchronous Techniques for System-on-Chip Design", Proceedings of the IEEE, vol.94, no.6, pp. 1089–1120, 2006.

[50] T. N. K. Jain, "Asynchronous Bypass Channels: Improving Performance for Multi-Synchronous NoCs", Proceedings of the International Symposium on Networks-on-Chip (NOCS), pp. 51–58, 2010.

[51] S. Saponara, F. Vitullo, R. Locatelli, P. Teninge, M. Coppola, L. Fanucci, "LIME: a Low-Latency and Low-Complexity On-Chip Mesochronous Link with Integrated Flow Control", Proceedings of Euromicro Conference on Digital System Design (DSD), pp. 32–35, 2008.

[52] D. Mangano, G. Falconeri, C. Pistritto, A. Scandurra, "Effective Full-Duplex Mesochronous Link Architecture for Network-on-Chip Data-Link Layer", Proceedings of Euromicro Conference on Digital System Design (DSD), pp. 519–526, 2007.

[53] F. Vitullo et al. "Low-Complexity Link Microarchitecture for Mesochronous Communication in Networks-on-Chip", IEEE Trans. on Computers, Vol.57, issue 9, pp. 1196–1201, 2008.

[54] D. Wiklund, "Mesochronous Clocking and Communication in On-Chip Networks", Proceedings of the Swedish System-on-Chip Conference, 2003.

[55] A.T.Tran, D.N.Truong, B.Baas, "A Reconfigurable Source-Synchronous On-Chip Network for GALS Many-Core Platforms", IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Vol.29, no.6, 2010.

[56] D.Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor", IEEE Micro, vol.27, no.5, pp. 15–31, 2007.

[57] D. A. IIitzky, J. D. Hoffman, A. Chun, B. P. Esparza, Architecture of the Scalable Communications Core's Network on Chip", IEEE Micro, vol.27, Issue 5, pp. 62 - 74,2007.

[58] F. Clermidy, R. Lemaire, X. Popon, D. Ktenas, Y. Thonnart, "An Open and Reconfigurable Platform for 4G Telecommunication: Concepts and Application", Proceedings of Euromicro Conference on Digital System Design (DSD), pp. 62–74, 2009.

[59] F. Clermidy, C. Bernard, R. Lemaire, J. Martin, I. Miro-Panades, Y. Thonnart, P. Vivet, N. Wehn, "A 477mW NoC-based Digital Baseband for MIMO 4G SDR", ISSCC'2010, pp. 278–279, 2010.

[60] Y. Thonnart, P. Vivet, F. Clermidy, "A Fully-Asynchronous Low-Power Framework for GALS NoC Integration", Proceedings of Design, Automation and Test in Europe (DATE'10), pp. 33–38, 2010.

[61] M. Krstic et al., "Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook", IEEE Design and Test of Computers, vol. 24, no. 5, pp. 430–441, Sept. 2007.

[62] E. Flamand, "Strategic Directions Towards Multicore Application Specific Computing", Proceedings of Design, Automation and Test in Europe (DATE'09), pp. 1266, 2009.

[63] R. Dobkin, V. Vishnyakov, E. Friedman, R. Ginosar, "An Asynchronous Router for Multiple Service Levels Networks on Chip", Proceedings of ASYNC'05, pp. 44–53, 2005.

[64] S. Beer, R. Ginosar, M. Priel, R. R. Dobkin, A. Kolodny, "The Devolution of Synchronizers", Proceedings of ASYNC, pp. 94–103, 2010.

[65] T. Bjerregaard, J. Sparso, "A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip", Proceedings of Design, Automation and Test in Europe (DATE), pp. 1226–1231, 2005.

[66] B. R. Quinton, M. R. Greenstreet, S. J.E. Wilton, "Asynchronous IC Interconnect Network Design and Implementation Using a Standard ASIC", Proceedings of the International Conference of Computer Design (ICCD), pp. 267–274, 2005.

[67] K. Y. Yun, R. P. Donohue, "Pausible Clocking: a First Step Toward Heterogeneous Systems", Proceedings of the International Conference of Computer Design (ICCD), pp. 118–123, 1996.

[68] R. Mullins, S. Moore, "Demystifying Data-Driven and Pausible Clocking Schemes", Proceedings of the International Symposium on Asynchronous Circuits and Systems, pp. 175–185, 2007.

[69] Z. Yu, B. M. Baas, "Implementing Tile-Based Chip Multiprocessors with GALS Clocking Styles", Proceedings of the International Conference on Computer Design, pp. 174–179, 2006.

[70] B. Mesgarzadeh, C. Svensson, A. Alvandpour, "A New Mesochronous Clocking Scheme for Synchronization in SoC", ISCAS, pp.605–609, 2002.

[71] A. Ferrante, S. Medardoni, D. Bertozzi, "Network Interface Sharing Techniques for Area Optimized NoC Architectures", Proceedings of the 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD), 2008, Italy.

[72] I. M. Panades, F. Clermidy, P. Vivet, A. Greiner, "Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture", Proceedings of International Symposium on Networks-on-Chip (NOCS), pp. 139–148, 2008.

[73] F. Vitullo, N. E. L'Insalata, E. Petri, L. Fanucci, M. Casula, R. Locatelli, M. Coppola, "Low-Complexity Link Microarchitecture for Mesochronous Communication in Networks-on-Chip", IEEE Trans. on Computers, Vol.57, no.9, pp. 1196–1201, 2008.

[74] S. Vangal et al.; "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS", IEEE Journal of Solid-State Circuits, Vol.43, Issue 1, pp. 29–41, 2008.

[75] S. W. Keckler et al., "A wire-delay scalable microprocessor architecture for high performance systems", ISSCC'03, Feb. 2003, pp. 168–169.

[76] Z. Yu et al., "An asynchronous array of simple processors for DSP applications", in ISSCC'06, Feb. 2006, pp. 428–429.

[77] A. M. Scott, M. E. Schuelein, M. Roncken, J. Hwan, J. Bainbridge, J. R. Mawer, D. L. Jackson, A. Bardsley, "Asynchronous on-Chip Communication: Explorations on the Intel PXA27x Processor Peripheral Bus", Proceedings of the 13th International Symposium on Asynchronous Circuits and Systems, pp. 60–72, 2007.

[78] M. B. Stensgaard, T. Bjerregaard, J. Sparso, J. H. Pedersen, "A Simple Clockless Network-on-Chip for a Commercial Audio DSP Chip", Proceedings of the 9th EUROMICRO Conference on Digital System Design, pp. 641–648, 2006.

[79] E. Beigne, F. Clermidy, S. Miermont, P. Vivet, "Dynamic Voltage and Frequency Scaling Architecture for Units Integration within a GALS NoC", Proceedings of the International Symposium on Networks-on-Chip (NOCS), pp. 129–138, 2008.

[80] E. Beigne, F. Clermidy, S. Miermont, Y. Thonnart, A. Valentian, P. Vivet, "A Localized Power Control mixing hopping and Super Cut-Off techniques within a GALS NoC", Int. Conf. on Integrated Circuit Design and Technology and Tutorial, pp. 37–42, 2008.

[81] U. Y. Ogras, R. Marculescu, P. Choudhary, D. Marculescu, "Voltage-Frequency Island Partitioning for GALS-based Networks-on-Chip",

Proceedings of the Design Automation Conference (DAC), pp. 110–115, 2007.

[82] T. Ono, M. Greenstreet, "A Modular Synchronizing FIFO for NOCs", Proceedings of International Symposium on Networks-on-Chip (NOCS), 2009

[83] F. Mu, C. Svensson; "Self-Tested Self-Synchronization Circuit for Mesochronous Clocking", IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, Vol.48, no.2, pp.129–141, 2001.

[84] A. Edmanand, C. Svensson, "Timing Closure through Globally Synchronous, Timing Portioned Design Methodology", Proceedings of Design Automation Conference (DAC), pp. 71–74, 2004.

[85] Federico Angiolini, "Interconnection Systems For Highly Integrated Computation Devices", Phd Thesis, University of Bologna, 2008.

[86] "Synopsis Physical Compiler", `http://www.synopsys.com`

[87] "Cadence SoC Encounter", `http://www.cadence.com`

[88] P. Caputa, C. Svensson, "An On-Chip Delay- and Skew-Insensitive Multicycle Communication Scheme", IEEE Solid-State Circuits Conference (ISSCC), pp. 1765–1774, 2006.

[89] SIA Semiconductor Industry Association "The International Technology Roadmap for Semiconductors", `http://public.itrs.net/`

[90] I. M. Panades, A. Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures", Proceedings of International Symposium on Networks-on-Chip (NOCS), pp. 83–94, 2007.

[91] I. Loi, F. Angiolini, L. Benini, "Developing Mesochronous Synchronizers to Enable 3D NoCs", Proceedings of International Conference on VLSI Design, 2007.

[92] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, G. De Micheli, "xpipesLite: a Synthesis Oriented Design Library for Networks on Chips", Proceedings of the Design Automation and Test in Europe Conference (DATE), pp. 1188–1193, 2005.

[93] F. Angiolini, L. Benini, P. Meloni, L. Raffo, S. Carta, "Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness", Proceedings of Design, Automation and Test in Europe (DATE), 2006.

[94] Y. Semiat, R. Ginosar, "Timing Measurements of Synchronization Circuits", Proceedings of the International Symposium on Advanced Research in Asynch. Circuits and Systems, pp. 68–77, 2003.

[95] W. Ning, G. Fen, W. Fei, "Design of a GALS Wrapper for Network on Chip", World Congress on Computer Science and Information Engineering, pp. 592–595, 2009.

[96] D. Mangano, G. Falconeri, C. Pistritto, A. Scandurra, "Effective full-duplex Mesochronous Link Architecture for Network-on-Chip Data-Link layer", Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, pp. 519–526, 2007.

[97] OCP International Partnership, "OCP Specification 2.2", 2007.

[98] D. Kim, K. Kim, J. Y. Kim, S. Lee, H. J. Yoo, "Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC", Proceedings of the International Symposium on Networks-on-Chips (NOCS), 2007.

[99] Y. Thonnart, E. Beigne, P. Vivet, "Design and Implementation of a GALS Adapter for ANoC Based Architectures", Proceedings of the 2009 15th IEEE Symposium on Asynchronous Circuits and Systems, pp. 13–22, 2009.

[100] J. Jaros, M. Ohlidal, V. Dvorak, "Complexity of Collective Communications on NoCs", Proceedings of the International Symposium on Parallel Computing in Electrical Engineering, pp. 127–133, 2006.

[101] D. Sylvester and K. Keutzer, "Getting to the bottom of deep sub-micron II: A global paradigm", Proceedings of the IEEE International Symposium on Physical Design, pp. 193–200, 1999.

[102] M. Ghoneima, Y. Ismail, M. Khellah, V. De, "Variation-Tolerant and Low-Power Source-Synchronous Multi-Cycle On-Chip Interconnection Scheme", VLSI Design, 2007.

[103] Zhiyi Yu, Bevan M. Baas, "High Performance, Energy Efficiency, and Scalability with GALS Chip Multiprocessors", IEEE Trans. VLSI, vol.17, no.1, pp. 66–79, 2009.

[104] G. Campobello, M. Castano, C. Ciofi, D. Mangano, "GALS Networks on Chip: a new solution for asynchronous delay-insensitive links", Proceedings of Design, Automation and Test in Europe (DATE), pp. 160–165, 2006.

[105] S. Kim, R. Sridhar, "Self-Timed Mesochronous Interconnections for High-Speed VLSI Systems", Proceedings of GLSVLSI, pp. 122–128, 1996.

[106] M. R. Greenstreet, "Implementing a STARI chip", Proceedings of ICCD, pp.3, 1995.

[107] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, M. Renaudin, "An Asynchronous NOC Architecture Providing Low Latency Service and Its Multi-Level Design Framework", Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 54–63, 2005.

[108] A. Pullini et al., "Bringing NoCs to 65 nm", IEEE Micro 27(5): pp. 75–85, 2007.

[109] E. Beigne, P. Vivet, "Design of on-chip and off-chip interfaces for a GALS NoC architecture", Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 172, 2006.

[110] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor", IEEE Micro, Volume 27:5, 2007.

[111] B. Stackhous et al., "A 65nm 2-Billion Transistor Quad-Core Itanium Processor", IEEE Journal of Solid State Circuits, Volume 44, pp. 18–31, 2009.

[112] M. Millberg, E. Nilsson, R. Thid, S. Kumar, A. Jantsch, "The Nostrum backbone - a Communication Protocol Stack for Networks on Chip", Proceedings of the VLSI Design Conference, 2004.

[113] W. J. Dally, J. W. Poulton, "Digital Systems Engineering", Cambridge University Press, 1998

[114] S. Mahadevan, F. Angiolini, M. Storoaard, R.G. Olsen, J. Sparso, J. Madsen, "Network traffic generator model for fast network-on-chip simulation". Proceedings of Design, Automation and Test in Europe (DATE), pp.780–785, 2005.

[115] S. Medardoni, D. Bertozzi, L. Benini, E. Macii, "Control and datapath decoupling in the design of a NoC switch: area, power and performance implications". Proceedings of the International Symposium on System-on-Chip, pp.1–4, 2007.

[116] R. Ho, K.W. Mai, M.A Horowitz, "Managing wire scaling: a circuit perspective". Proceedings of the IEEE 2003 International Interconnect Technology Conference, pp.177–179, 2003.

[117] M. Krstic, E. Grass, C. Stahl, "Request-driven GALS technique for wireless communication system", Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 76–85, 2005.

[118] S. Borkar, "Design Perspectives on 22nm CMOS and Beyond", Proceedings of the Design Automation Conference (DAC), 2009.

[119] C. Gomez et al.; "Beyond Fat-Tree: Unidirectional Load-Balanced Multistage Interconnection Network", Computer Architecture Letters, June 2008.

[120] C. Gomez et al., "Deterministic versus Adaptive Routing in Fat-Trees", Proceedings of CAC'07, as part of IPDPS'07, 2007.

[121] A. Tran, D. Truong, B. Baas, "A GALS Many-Core Heterogeneous DSP Platform with source-Synchronous On-Chip Interconnection Network", Proceedings of the International Symposium on Networks-on-Chip, 2009.

[122] STn8811A12 Mobile Multimedia Application Processor, available online: http://www.st.com

[123] SPEAr Plus600 dual processor cores, available online: http://www.st.com

[124] J. Bainbridge, "CHAINWorks", Silistix, http://www.silistix.com

[125] TILE64 PROCESSOR FAMILY, available online: http://www.tilera.com/pdf/ProBrief_Tile64_Web.pdf

[126] Tensilica LX configurable processor, Tensilica Inc., http://www.tensilica.com

[127] F. Petrini, M. Vanneschi, "k-ary n-trees: High Performance Networks for Massively Parallel Architectures", International Parallel Processing Symposium, pp. 87–93, 1997.

[128] A. Dalla Torre, M. Ruggiero, A. Acquaviva, L. Benini, "MP-Queue: an Efficient Communication Library for Embedded Streaming Multimedia Platform", IEEE Workshop on Embedded Systems for Real-Time Multimedia, 2007.

[129] J. Balfour, W. J. Dally, "Design Tradeoffs for Tiled CMP On-Chip Networks", Proceedings of the ACM International Conference on Supercomputing, 2006.

[130] S. Vangal et al., "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS", Proceedings of ISSCC, pp. 98–589, 2007.

[131] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, A. Scandurra, "Spidergon: a novel on-chip communication network", Proceedings of the International Symposium on System-on-Chip, 2004, pp. 16–18, 2004.

[132] S. Bourduas, Z. Zilic, "Latency Reduction of Global Traffic in Wormhole-Routed Meshes Using Hierarchical Rings for Global Routing", Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 302–307, 2007.

[133] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems". PhD Dissertation, Stanford University, October 1984.

[134] R. Ginosar, "Fourteen Ways to Fool Your Synchronizer", Proceedings of International Symposium on Asynchronous Circuits and Systems, pp. 89–97, 2003.

[135] Xuan-Tu Tran, J. Durupt, F. Bertrand, V. Beroulle, C. Robach, "A DFT Architecture for Asynchronous Networks-on-Chip", Proceedings of the IEEE European Test Symposium, pp. 219–224, 2006.

[136] Xuan-Tu Tran, Y.Thonnart, J.Durupt, V.Beroulle, C.Robach, "A Design-for-Test Implementation of an Asynchronous Network-on-Chip Architecture and its Associated Test Pattern Generation and Application", Proceedings of the International Symposium on Networks-on-Chip, pp. 149–158, 2008.

[137] F. Martinez Vallina, N. Jachimiec, J. Saniie, "NOVA interconnect for dynamically reconfigurable NoC systems", Proceedings of the IEEE International Conference on Electro/Information Technology, 2007, pp. 546–550, 2007.

[138] F. Gilabert, M. E. Gomez, P. J. Lopez, "Performance Analysis of Multi-dimensional Topologies for NoC", ACACES 2007, poster session with proceedings at the HiPEAC Summer School.

[139] E. Rijpkema, K. Goossens, A. Radulescu, "Trade Offs in the Design of a Router with both Guaranteed and Best-Effort Services for Networks on Chip", Design, Automation and Test in Europe (DATE), pp. 350–355, 2003.

[140] S. Kumar et al., "A Network on Chip Architecture and Design Methodology", IEEE Computer Society Annual Symposium on VLSI, pp. 105–112, 2002.

[141] V.D. Ngo, H.N. Nguyen, H.W. Choi, "Analyzing the Performance of Mesh and Fat-Tree Topologies for Network on Chip Design", L.T.Yang et al. (Eds): EUC 2005, LNCS 3824, pp. 300–310, 2005.

[142] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, R. Saleh, "Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures", IEEE Transaction on Computers, Vol.54, no. 8, 2005.

[143] S. Medardoni, F. Gilabert, D. Bertozzi, D., M. E. Gomez, P. J. Lopez, "Towards an Implementation-Aware Transaction-Level Modeling of On-Chip Networks for Fast and Accurate Topology Exploration", INA-OCMC Workshop, co-located with the HiPEAC Conference, 2008.

[144] M. Mirza-Aghatabar, S. Koohi, S. Hessabi, M. Pedram, "An Empirical Investigation of Mesh and Torus NoC Topologies Under Different Routing Algorithms and Traffic Models", Proceedings of the Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD), pp. 19–26, 2007.

[145] L. Bononi, N. Concer, M. Grammatikakis, M. Coppola, R. Locatelli, "NoC Topologies Exploration based on Mapping and Simulation Models", Proceedings of the Euromicro Conference on Digital System Design Architectures (DSD), pp. 543–546, 2007.

[146] Circuits Multi-Projects, Multi-Project Circuits; http://cmp.imag.fr

[147] H. Wang, L. S. Peh, S. Malik, "A Technology-Aware and Energy Oriented Topology Exploration for On-Chip Networks", Proceedings of the Design, Automation and Test in Europe (DATE), pp. 1238–1243, 2005.

[148] V. F. Pavlidis, E. G. Friedman, "3-D Topologies for Networks-on-Chip", Proceedings of the International System-on-Chip Conference (SOC), pp. 285–288, 2006.

[149] I. Hatirnaz, S. Badel, N. Pazos, Y. Leblebici, S. Murali, D. Atienza, G. De Micheli, "Early Wire Characterization for Predictable Network-on-Chip Global Interconnects", Proceedings of SLIP'07, pp. 57–64, 2007.

[150] P. P. Pande, C. Grecu, A. Ivanov, R. Saleh "Design of a Switch for Network on Chip Applications", ISCAS'03, pp. 217–220, Vol.5, 2003.

[151] S. Murali, G. De Micheli, "SUNMAP: a Tool for Automatic Topology Selection and Generation for NoCs", Proceedings of the Design Automation Conference, 2004, pp. 914–914.

[152] A. Jalabert et al., "xpipesCompiler: a Tool for Instantiating Application Specific Networks on Chip", Proceedings of the Design, Automation and Test in Europe (DATE), pp. 884–889, 2004.

[153] V. Soteriou, N. Eisley, H. Wang, B. Li, L. S. Peh, "Polaris: a System-Level Roadmapping Toolchain for On-Chip Interconnection Networks", IEEE Trans. on VLSI 15(8), pp. 855–868, 2007.

[154] S. Murali, G. De Micheli, "SUNMAP: a Tool for Automatic Topology Selection and Generation for NoCs", Proceedings of the Design Automation Conference, pp. 914–914, 2004.

[155] F. Gilabert, S. Medardoni, D. Bertozzi, L. Benini, M. E. Gomez, P. J. Lopez, J. Duato, "Exploring High-Dimensional Topologies for NoC Design Through an Integrated Analysis and Synthesis Framework", Proceedings of the International Symposium on Networks-on-Chip, 2008.

[156] A. Strano, C. G. Requena, D. Ludovici, M. E. Gomez, M. Favalli, D. Bertozzi, "Exploiting Network-on-Chip Structural Redundancy for A Cooperative and Scalable Built-In Self-Test Architecture", Proceedings of Design, Automation and Test in Europe 2011 (DATE), Grenoble, Franch, March 2011.

List of Publications

Book Chapters

1. F. Gilabert, **D. Ludovici**, M. E. Gómez, D. Bertozzi, **Topology Exploration**, Chapter 4 in *"Designing Network-on-Chip Architectures in the Nanoscale Era"*, pp. 89-134, December 2010, CRC Book, ISBN: 978-1-4398-3710-8.

2. D. Bertozzi, A. Strano, **D. Ludovici**, V. Pavlidis, F. Angiolini, M. Krstic, **The Synchronization Challenge**, Chapter 6 in *"Designing Network-on-Chip Architectures in the Nanoscale Era"*, pp. 177-235, December 2010, CRC Book, ISBN: 978-1-4398-3710-8.

3. D. Bertozzi, A. Strano, F. Gilabert, **D. Ludovici**, **Technology-Aware Communication Architecture Design for Parallel Hardware Platforms**, Chapter in *"Advanced Circuits for Emerging Technology"*, CRC Book, 2011, *in press*.

International Journals

1. A. Strano, **D. Ludovici**, D. Bertozzi, **A Library of GALS Interfaces for Cost-Effective and Flexible MPSoC Design**, *Transaction on HiPEAC*, *accepted for publication*.

International Conferences (with proceedings)

1. A. Ghiribaldi, **D. Ludovici**, M. Favalli, D. Bertozzi, **System-Level Infrastructure for Boot-time Testing and Configuration of Networks-on-Chip with Programmable Routing Logic**, *Proceedings of the IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, *accepted for publication*, Hong Kong, China, 2011.

2. A. Strano, C. G. Requena, **D. Ludovici**, M. E. Gómez, M. Favalli, D. Bertozzi, **Exploiting Network-on-Chip Structural Redundancy for A Cooperative and Scalable Built-In Self-Test Architecture**, *Proceedings of Design, Automation and Test in Europe 2011 (DATE)*, pp. 661–666, Grenoble, France, 2011.

3. **D. Ludovici**, A. Strano, G. Gaydadjiev, D. Bertozzi, **Mesochronous NoC Technology for Power-Efficient GALS MPSoC**, *Proceedings of the Fifth ACM Interconnection Network Architecture, On-Chip Multi-Chip Workshop (INA-OCMC)*, pp. 27–30, Heraklion, Greece, 2011.

4. **D. Ludovici**, F. Gilabert, M. E. Gómez, G. N. Gaydadjiev, D. Bertozzi, **Contrasting Topologies for Regular Interconnection Networks under the Constraints of Nanoscale Silicon Technology**, *Proceedings of the 3rd ACM/IEEE International Workshop on Network-on-Chip Architectures (NoCArc)*, pp. 37–42, Atlanta, USA, 2010.

5. A. Strano, **D. Ludovici**, D. Bertozzi, **A Library of Dual-Clock FIFOs for Cost-Effective and Flexible MPSoCs Design**, *Proceedings of the International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS)*, pp. 20–27, Samos, Greece, 2010.

6. **D. Ludovici**, A. Strano, G. N. Gaydadjiev, L. Benini, D. Bertozzi, **Design Space Exploration of a Mesochronous Link for Cost-Effective and Flexible GALS NOCs**, *Proceedings of Design, Automation and Test in Europe 2010 (DATE)*, pp. 679–684, Dresden, Germany, 2010.

7. **D. Ludovici**, A. Strano, D. Bertozzi, **Architecture Design Principles for the Integration of Synchronization Interfaces into Network-on-Chip Switches**, *Proceedings of the 2nd ACM/IEEE International Workshop on Network-on-Chip Architectures (NoCArc)*, pp. 31–36, New York, USA, 2009.

8. **D. Ludovici**, A. Strano, D. Bertozzi, L. Benini, G. N. Gaydadjiev, **Comparing Tightly and Loosely Coupled Mesochronous Synchronizers in a NoC Switch Architecture**, *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pp. 244–249, San Diego, USA, 2009.

9. **D. Ludovici**, D. Bertozzi, L. Benini, G. N. Gaydadjiev, **Capturing Topology-Level Implications of Link Synthesis Techniques for Nanoscale Networks-on-Chip**, *Proceedings of the 19th ACM/IEEE Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 125–128, Boston, USA, 2009.

10. **D. Ludovici**, F. Gilabert, S. Medardoni, C. G. Requena, M. E. Gómez, P. López, D. Bertozzi, G. N. Gaydadjiev, **Assessing Fat-Tree Topologies for Regular Network-on-Chip Design under Nanoscale Technology Constraints**, *Proceedings of Design, Automation and Test in Europe 2009 (DATE)*, pp. 562–565, Nice, France, 2009.

11. F. Gilabert, **D. Ludovici**, S. Medardoni, D. Bertozzi, L. Benini, G. N. Gaydadjiev, **Designing Regular Network-on-Chip Topologies under Technology, Architecture and Software Constraints**, *Proceedings of IEEE International Workshop on Multi-Core Computing Systems (MuCoCoS)*, pp. 681–687, Fukuoka, Japan, 2009.

International and Local Conferences (without proceedings)

1. **D. Ludovici**, G. Keramidas, G. N. Gaydadjiev, S Kaxiras, **Integration of Power Saving Techniques in the UNISIM Simulation Framework through the Shadow Module design paradigm**, *1st Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO), in conjunction with HiPEAC Conference*, Paphos, Cyprus, 2009.

2. **D. Ludovici**, F. Gilabert, C. G. Requena, M. E. Gómez, P. López, G. N. Gaydadjiev, J. Duato, **Butterfly vs. Unidirectional Fat-Trees for Networks-on-Chip: not a Mere Permutation of Outputs**, *3rd Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip (INA-OCMC), in conjunction with HiPEAC Conference*, Paphos, Cyprus, 2009.

3. **D. Ludovici**, G. N. Gaydadjiev, **SARC Power Estimation Methodology**, *Proceedings of the 18th Annual Workshop on Circuits, Systems and Signal Processing (ProRisc)*, Veldhoven, The Netherlands, 2007.

4. **D. Ludovici**, S. Wong, **Performance Analysis of RR and FQ Algorithms in Reconfigurable Routers**, *Proceedings of the 17th Annual Workshop on Circuits, Systems and Signal Processing (ProRisc)*, Veldhoven, The Netherlands, 2006.

Samenvatting

On-chip netwerken (NoCs) worden de laatste tien jaar gezien als de nieuwe ontwerpmethode voor grote multiprocessor systemen (MPSoC). In het begin vershilden NoCs sterk van de huidige oplossingen, met name door de vele onverwachte ontwerpuitdagingen van de evoluerende CMOS technologie. Echter, verschillende verborgen problemen op laag niveau kunnen leiden tot verslechterde systeemprestaties, niet acceptabel vermogensverbruik en zelfs tot een onmogelijk ontwerp. De connectiviteit tussen de verschillende multiprocessor elementen is zo'n probleem dat aangepakt moet worden tijdens het ontwerp van de communicatie infrastructuur. Twee implicaties verbonden aan de agressieve CMOS technologie downscaling, als resultaat van groeiende procesvariaties, verminderde vermogensbudgetten en verslechterende signaalkwaliteit, moeten worden beschouwd. Aan de ene kant wordt een goede topologie vereist voor een efficiënte subsysteem connectiviteit, terwijl ook aan de bandbreedte en de prestatievereisten wordt voldaan. Aan de andere kant maken synchronisatiekwesties het systeemontwerp moeilijk en in sommige gevallen zelfs onmogelijk onder een strak synchronisatiemodel. Zo hangt de topologie bijvoorbeeld sterk af van de laagniveau effecten als gevolg van de verslechterende vertragingstijden, terwijl de synchronisatiekwesties nauw aan procesvariaties verbonden zijn. Daarom moeten in de huidige en de toekomstige CMOS technologieën, tegenstrijdige ad-hoc maatregelen genomen worden om de bovenstaande problemen aan te kunnen. In dit proefschrift stellen wij een systeemniveau analyse methode en ontwerpmethodologie voor, beiden rekening houdend met echte layout effecten. Onze analyse is niet alleen beperkt tot klassieke layout effecten zoals de niet-regelmatigheid van de rechthoekige modules; de werkelijke vertragingstijden binnen inter-schakelaar connecties; het aantal van pipeline niveaus dat vereist is om de gewenste prestaties te krijgen; de maximum toelaatbare signaalskew van de synchronisatie methode; en meer. Wij beschouwen ook de implicaties van de bovengenoemde laag-niveau effecten tijdens herontwerp van onze architectuur blokken. Het ultieme resultaat is een ontwerpmethode welke werkelijk rekening houdt met de technologie, en klaar is om de uitdagingen van het toekomstige CMOS technologielandschap aan te kunnen.

Curriculum Vitae

Daniele Ludovici was born on the 12^{th} of August 1981 in Alatri (FR), Italy. He received the Bachelor of Science (BSc.) degree in Computer Systems Engineering from University of Pisa, Italy, in 2003. In the same year, he was enrolled as postgraduate student, within the same university, in a MSc. program. In December 2006, he obtained the Master of Science (MSc.) degree in Computer Systems Engineering from University of Pisa. Daniele developed his MSc. thesis at the Computer Engineering (CE) laboratory of Delft University of Technology (TU Delft), The Netherlands. In 2007, he joined the same laboratory as PhD. candidate under the guidance of dr. Georgi N. Gaydadjiev. Part of his PhD. work has been developed in a tight cooperation with the MPSoC research group of the University of Ferrara, Italy, under the supervision of dr. Davide Bertozzi. Daniele's work spans the area of on-chip networks including technology aware topology exploration; physical link design, characterization and modeling; design and integration of synchronizers for the Network-on-Chip domain. He served as reviewer in various conferences and journals including the ACM Transactions on Embedded Computing Systems (TECS), the Design Automation Conference (DAC), the Design Automation and Test in Europe (DATE) conference and the ACM/IEEE International Symposium on Networks on Chip (NOCs). He served as program chair of the ACM Interconnection Network Architecture On-Chip Multi-Chip (INA-OCMC) workshop colocated with the HiPEAC conference in 2011.