Universitat Politècnica de València
Departamento de Informática de Sistemas y Computadores

Design Space Exploration for Networks On-chip

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
(Computer Sciences)

Author

Francisco Gilabert Villamón

Advisors

María Engracia Gómez Requena
Davide Bertozzi

Valencia, 2011

Acknowledgments

It is a pleasure to thank the many people who made this thesis possible.

First and foremost, I thank my family for their unending support. I would like to thank Merlyx (my wife) for giving the most important step of my entire life with me: starting our own family. Also, I would like to thank her for her patience during the months I spent out of home in stays and conferences. I would like to thank my parents and my sister for all the help they have provided to me.

In second place, I would like to thank my friends, for their support and company. Mainly, I would like to thank Cris, Blas, Ricardo and Simone. Cris is the closest friend I ever had, and we spent most of our free time together playing computer games, watching movies or just talking. Also, we have traveled around the world together, and we shared several moments that I will never forget. I would like to thank Blas for the great moments spend playing football, as well as for the great meals we shared together with Ricardo in our preferred restaurant. Finally, I would like to thank Simone for his help during my stays on Italy.

I am indebted to my advisors, Davide Bertozzi and María Engracia, for their enthusiasm, their inspiration, and their sound advices, as well as for the encouragement they provided to me. I would have been lost without them. I would like to thank Davide Bertozzi for being such an amazing host during my stays in Italy. Also, I would like to thank Prof. José Duato for including me in his Parallel Architecture Group: a big family composed by good professionals and excellent persons. I would like to thanks Pedro López, I will always remember his help and understanding during my first steps into the research world. Many of the other faculty members in the Parallel Architecture

Group did help me too, either professionally or personally, thanks to you all. Finally, I would like to thanks Sören Sonntag for showing their advices and their guidance during my stay at Lantiq.

I have met many students and professionals from around the world along the years, either in the lab, or during my trips and stays. Although I cannot possibly mention everyone who has enriched my experience or provided moral support, I wish to specifically thank a few individuals: Cris, Simone, Ricardo, Daniele, Gaspar, Carles, Blas, Héctor and Knut. All of them have helped me at any point on my research and on my personal life, and I would like to thanks them for it.

Contents

List of Figures

List of Tables

Abstract

Multi-core designs are becoming a more and more popular solution to most of the single-core designs limitations. A multi-core design follows the System on-Chip (SoC) paradigm, in which several cores are integrated into a single chip. The performance of SoC designs is heavily influenced by the interconnection infrastructure implemented: while small SoCs designs can employ buses as their interconnection, more complex interconnections are required as the number of cores of the design increases.

In this context, the Network on-Chip (NoC) paradigm arises as a solution to the interconnection challenges of new SoC designs. In a NoC, transactions between cores are encapsulated into packets, and delivered through a shared interconnection network. This concept is adopted from the off-chip interconnection networks field, and thus it can inherit most of the techniques of this mature field. Anyway, the on-chip environment presents different constraints than the off-chip one. For example, buffer size is a critical design parameter for any on-chip design, while it is a more relaxed constraint in off-chip ones. For this reason, techniques from the off-chip domain should not be taken for granted when implementing a NoC, but revised and adapted by taking into account the special characteristics of the on-chip domain.

In particular, in the on-chip domain the relentless pace of technology scaling to the nanoscale regime is bringing physical effects into the forefront. For this reason, visibility of the lower layers of the design hierarchy is required to accurately assess the capabilities of any given NoC design. In SoCs, the on-chip interconnect will be implicated in every communication stream. Therefore, the NoC physical characteristics are critical for overall SoC performance. In this context, SoC designs will be viable only when taking silicon aware decision at

each layer of the design hierarchy. However, this does not affect to need for fast design space exploration frameworks by means of abstract tools, hence making the accuracy-exploration speed trade-off increasingly difficult to cope with. For a given design space, the high heterogeneity (architectural and technological) of NoC solutions increases the complexity of selecting the optimal NoC configuration. A common approach is to use high level tools to provide performance estimations that the designer will use to select the most promising candidates at the earlier stages of the design flow. But there is a gap between the performance predictions of high-level tools and the real performance achieved after the system is implemented. In fact, this gap is growing, as the number of libraries available for a given technology node increases, forcing to increase the number of design re-spins due to inaccurate high level predictions, which results in increased design cost. The ultimate implication is that back-annotations from the lower layers of the design hierarchy need to be exposed to design tools operating at the upper layers, finding a new trade-off between simulation and/or exploration speed and layout awareness. Current front-end tools for use in the early design stages suffer from performance mispredictions, since they often ignore the technology platform. The challenge in finding the new trade-off point is that when considering both front-end and back-end issues (e.g., multiple technology libraries, core size), the design space becomes even wider than it is now.

This dissertation focuses on the development of a new set of front-end design, modeling and simulation tools for layout-aware pruning of the design space toward the most promising candidates. As an outcome of this dissertation, NoC designers will be able to take advantage of a new set of design space exploration tools reducing time-to-market and bridging the accuracy gap with the lower layers of the design process. In a first step, we focused on the design and development of an experimental setting for use in the analysis of alternative architecture design techniques. Since the objective is the analysis of a large range of implementation alternatives, it was decided to pursue transaction-level simulations. Hence, the dissertation reports on an extensive abstraction and modeling effort of NoC architecture building blocks while retaining accuracy of functionally equivalent Register Transfer Level (RTL) models. As a result, dramatic improvements in simulation speed were

achieved. At the same time, all abstract models were parameterized with some key parameters from the physical synthesis (e.g., target frequency, link latency), thus being able to evaluate points in the design space quickly and with high accuracy.

In the second step, by using the developed abstract simulation framework layout awareness was brought to the upper layers of NoC design, by reviewing architectural design techniques mutated from the off-chip domain in light of layout constraints, selecting the most promising candidates and in some cases even exploiting the distinctive features of the on-chip setting to come up with radically new (and even counterintuitive) solutions. This activity, which was preliminary to the design of the Design Space Exploration (DSE) tool, was aimed at gaining that layout awareness that the tool itself should have. The activity was split into two stages.

On one hand, starting from a theoretical description of the most popular topologies proposed for NoCs in the open literature, the developed simulation framework was employed in order to demonstrate and address the above mentioned gap. Concrete examples of the abstraction and accuracy gap will be provided for industry-relevant NoC configurations and the methodology will be put at work to close this gap, hence coming up with trustworthy cross-benchmarking indications. To demonstrate the potentials of the new NoC investigation framework, the topology analysis process will be extended to a number of architecture variants, featuring different size, floorplanning constraints, technologies and even HW/SW interaction patterns. Additionally, the most common pitfalls of topology physical design will be highlighted, and most common techniques to address them will be analyzed both in terms of achieved performance and cost overhead.

On the other hand, during the research on the topology exploration topic, several topologies were considered that require the use of the virtual channels flow control mechanism in order to reach their maximum potential. This very popular technique has been used for several years in off-chip networks for a broad number of reasons, and thus, its use has been widely advocated for NoCs. The most common VC architecture for NoCs proposed in the open literature is mutated from off-chip networks. Although fully functional in the NoC context, this architecture is aimed to a different environment and results

in an overly large area and delay overhead. For this reason, we propose and evaluate an implementation of VC better suited for the constraints of NoCs and therefore able to provide a considerable improvement both in performance and in area/power over the commonly proposed implementation, with the aim of demonstrating the mismatch between smart architecture design techniques conceived for off-chip interconnection networks and an on-chip setting.

Finally, we focused on the development of the Design Space Exploration (DSE) tool. Our approach to this challenge is two-folded. On one hand, we capitalize the knowledge gained in previous steps on silicon-aware architecture design choices and simulation tools, to build a CAD tool to aid in the design of NoC-based systems. On the other hand, we introduce some techniques to reduce the time-to-market of new designs via DSE acceleration and fast re-spin techniques.

Resumen

Los diseños multi-núcleo se están convirtiendo en la solución más popular a la mayoría de las limitaciones de los diseños mono-núcleo. Un diseño multi-núcleo sigue el paradigma de diseño conocido como Sistema dentro del Chip (o SoC , del inglés System on-Chip), en el cuál varios núcleos se integran en un mismo chip. Las prestaciones de un diseño SoC dependen en gran medida de la infraestructura de interconexión que implemente: mientras que un SoC pequeño puede utilizar una interconexión de tipo bus, interconexiones más complejas serán necesarias conforme el número de núcleos del diseño aumente.

En este contexto, el paradigma de diseño conocido como red dentro del chip (o NoC, del inglés Network on-Chip) surge como una solución a los desafíos de interconexión presentes en los nuevos diseños de tipo SoC. En una NoC, las transacciones entre núcleos se encapsulan en paquetes, lo cuales serán entregados a su destinatario a través de la red de interconexión. Este concepto está adoptado del campo de las redes de interconexión de altas prestaciones, y como tal, puede heredar la mayoría de las técnicas diseñadas para un campo de investigación tan maduro como este. Pero a pesar de las similitudes existentes, ambos campos presentan diferentes limitaciones. Por ejemplo, el tamaño de los buffers es un factor crítico para el coste de diseños NoC, mientras que es un factor mucho menos relevante en el coste de las redes fuera del chip. Por ello, las técnicas diseñadas en el dominio de las redes fuera del chip no deberían suponerse directamente disponibles en NoCs, sino que deberán ser revisadas y adaptadas a sus particularidades.

En concreto, la reducción continua de las escalas de integración de las nuevas tecnologías hace que los efectos físicos afecten cada vez más a las prestaciones de una NoC. Dado que en la NoC está involucrada en todos los flujos de

comunicaciones de cualquier SoC, es necesario que todas las etapas del flujo de diseño de una NoC tengan en cuenta de las consecuencias a nivel físico de todas las decisiones tomadas. Sin embargo, para un diseño concreto, el alto número de posibles soluciones basadas en NoCs (tanto a nivel de arquitecturas como a nivel de tecnologías) incrementa la complejidad de analizar el espacio de diseño y de elegir la NoC óptima. La solución más común a este problema pasa por la utilización de herramientas de alto nivel para la obtención de estimaciones sobre las prestaciones de cada posible solución NoC, que posteriormente serán utilizadas por el diseñador para cribar el espacio de diseño en las primeras etapas del proceso de diseño. Pero hay una gran diferencia entre las prestaciones estimadas por herramientas de alto nivel y las prestaciones reales obtenidas una vez el sistema se implementa. De hecho, esta diferencia aumenta conforme la cantidad de librerías disponibles para cada nodo tecnológico aumenta, generando un mayor número de errores en el proceso de diseño, lo que se traduce en un mayor coste de diseño. Para evitar estos problemas, es necesario extraer las características físicas de las NoCs de las capas de nivel más bajo del proceso de diseño y anotarlas en las herramientas que operan en los niveles más altos, encontrando nuevos puntos de compromiso entre velocidad de exploración del espacio de diseño y su precisión. Las herramientas actuales para la exploración temprana del espacio de diseño sufren de una pobre precisión, dado que es una práctica habitual el ignorar los efectos de la implementación física en ellas. El principal desafío es que al añadir las implicaciones de la implementación física al espacio de diseño, el tamaño de éste es incluso mayor de lo que era en un principio.

Este trabajo se centra en el desarrollo de nuevas herramientas de alto nivel de diseño, modelado y simulación de NoCs, con el fin de cribar el espacio de diseño de los candidatos menos atractivos mediante simulaciones de alta precisión basadas en la anotación de las características físicas de los candidatos. Como resultado de este trabajo, los diseñadores de NoCs serán capaces de aprovechar un nuevo conjunto de herramientas para la exploración del espacio de diseño que reducen el tiempo de desarrollo y minimizan los problemas de precisión mencionados anteriormente. En un primer paso, nos centraremos en el diseño y desarrollo de una plataforma experimental para analizar arquitecturas alternativas para el diseño de NoCs. Por ello, este trabajo presenta

los esfuerzos realizados en el desarrollo de modelos abstractos de los bloques básicos de arquitecturas NoC con una funcionalidad y precisión similares a las de modelos equivalentes con precisión a nivel de registro (o modelos RTL, del inglés Register Transfer Level). Como resultado de dichos esfuerzos, se obtuvieron dramáticas mejoras en los tiempos de simulación sobre modelos RTL funcionalmente equivalentes. Al mismo tiempo, todos los modelos desarrollados permiten anotar algunos parámetros claves del proceso de síntesis física (como la frecuencia de operación o la latencia de los enlaces), de forma que permiten evaluar cualquier punto del espacio de diseño de forma rápida y precisa.

En el segundo paso, mediante el uso de la plataforma de simulación desarrollada en el primer paso, las capas superiores del proceso de desarrollo de una NoC fueron aumentadas con la capacidad de considerar efectos físicos derivados de la implementación de una NoC sobre sus prestaciones. Mediante esta metodología, se revisaron arquitecturas y técnicas de diseño adoptadas del dominio de las redes de interconexión fuera del chip, seleccionando las más prometedoras y, en algunos casos, explotando las características propias de las redes dentro de chip para obtener nuevas soluciones. Este paso, preliminar al desarrollo de la herramienta para la realización de exploraciones del espacio de diseño (o herramientas DSE, del inglés Design Space Exploration), tiene como objetivo depurar las técnicas para la abstracción de los efectos de la implementación física de las NoCs sobre sus prestaciones. Este paso se dividió en dos etapas.

Por un lado, partiendo de un punto de vista puramente teórico, se analizaron las topologías más populares presentes en la literatura. Posteriormente, la plataforma de simulación desarrollada se utilizó para demostrar y corregir los fallos de precisión anteriormente mencionados. Ejemplo concretos relacionados con dichos problemas de precisión fueron analizados para configuraciones de NoCs relevantes para la industria, utilizando la metodología desarrollada para corregirlos. Para demostrar el potencial de la nueva plataforma para la investigación de NoCs, el análisis de topologías se extendió a un variado número de arquitecturas, con diferentes tamaños, restricciones de mapeado, tecnologías en incluso patrones de interacción entre hardware y software. Adicionalmente, los problemas más comunes que se pueden encontrar a la hora

de analizar el diseño físico de topologías, así como las técnicas para corregirlos, fueron analizados en términos tanto de prestaciones como de coste de implementación.

Por otra parte, varias de las topologías consideradas durante la investigación sobre topologías para NoCs requieren del uso de un mecanismo de control de flujo basado en canales virtuales para poder rendir a su máximo potencial. Esta popular técnica de control de flujo ha sido utilizada en el dominio de las redes de interconexión durante varios años por una gran variedad de razones, y por ello, se ha propuesto en numerosas ocasiones su uso en NoCs. La arquitectura con canales virtuales para NoCs más común en la literatura es importada del dominio de redes off-chip. Aunque es completamente funcional en el contexto de las NoCs, esta arquitectura está diseñada para un entorno diferente, lo que se traduce en penalizaciones en términos de frecuencia de operación y área. Por este motivo, proponemos y evaluamos una implementación de canales virtuales adaptada a las características de las NoCs, y consecuentemente capaz de mejorar considerablemente las prestaciones, el área y el consumo de potencia comparado con la implementación comúnmente aceptada.

Finalmente, nos centramos en el desarrollo de la propia herramienta para la realización de DSE. Nuestra solución a este desafío es doble. Por un lado, aprovechamos las técnicas desarrolladas en los pasos previos relativas al impacto de la implementación física sobre las prestaciones de las NoCs para construir una herramienta CAD para el apoyo al diseño de sistemas basados en NoCs. Por otro lado, introducimos algunas técnicas para reducir el coste de desarrollo de nuevos diseños mediante la aceleración de la exploración y técnicas para la corrección rápida de errores en el proceso de diseño.

Resum

Els dissenys multi-nucli s'estàn convertint en la solució més popular a la majoria de les limitacions dels dissenys mono-nucli. Un disseny multi-nucli segueix el paradigma de disseny conegut com a Sistema dins del Xip (o SoC, de l'anglés System on-Chip), en el qual diversos nuclis s'integren en un mateix xip. Les prestacions d'un disseny SoC depenen en gran mesura de la infraestructura d'interconnexió que implemente: mentres que un SoC petit pot utilitizar una interconnexión de tipus bus, interconnexions més complexes seràn necessàries conforme el nombre del disseny augmente.

En aquest context, el paradigma de disseny conegut com a red dins del xip (o NoC, de l'anglés Network on-Chip) sorgeix com una solución als desafiaments d'interconnexión presents en els nous dissenys de tipus SoC. En una NoC, les transaccions entre els nuclis s'encapsulen en paquets, els quals seràn entregats al seu detsinatari a través de la xarxa d'interconnexió. Aquest concepte està adoptat del camp de les xarxes d'interconnexió d'altes prestacions, i com a tal, pot heretar la majoria de les tècniques dissenyades per a un camp d'investigació tan madur com aquest. Però a pesar de les similituds existents, ambdós camps presenten diferents limitacions. Per exemple, el tamany dels buffers és un factor crític per al cost de dissenys NoC, mentre que és un factor molt menys relevant en el cost de les xarxes fora del xip. Per això, les tècniques dissenyades en el domini de les xarxes fora del xip no haurien de suposar-se directament disponibles en NoCs, més bé haurien de ser revisades i adaptades a les seues particularitats.

Concretament, la reducció contínua de les escales d'integració de les noves tecnologies fa que els efectes físics afecten cada vegada més a les prestacions de una NoC.

Atès que en la NoC està involucrada en tots els fluxs de comunicacions de qualsevol SoC, és necessari que totes les etapes del flux de disseny d'una NoC tinguen en compte les conseqències, a nivell físic, de totes les decisions preses. No obstant això, per a un disseny concret, l'elevat númeo de possibles solucions basades en NoCs (tant d'arquitectures com de tecnologies) incrementa la complexitat d'analitzar l'espai de disseny i d'escollir la NoC òptima. La solució més comú a aquets problema pasa per la utilització d'eines d'alt nivell per a l'obtenció d'estimacions sobre les prestacions de cada possible solució NoC, que posteriorment seran utilitzades pel dissenyador per tal de garbellar l'espai de disseny en les primeres etapes del procés de disseny. Però hi ha una gran diferència entre les prestacions estimades per eines d'alt nivell i les prestacions reals obtingudes una vegada el sistema s'implementa. De fet, aquesta diferència augmenta segons la quantitat de llibreries disponibles per a cada nòdul tecnològic augmenta, generant un major nombre d'errors en el procès de disseny, la qual cosa es tradueix en un major cost de disseny. Per tal d'evitar aquests problemes, és necessari extraure les característiques físiques de les NoCs de les capes del nivell més baix del procés i anotar-les en les eines que operen en els nivells més alts, encontrant nous punts de compromís entre velocitat d'exploración de l'espai de disseny i la seua precisió. Les eines actuals per a l'exploració temprana de l'espai de disseny pateixen una pobre precisió, ja que és una pràctica habitual ignorar els efectes de la implementació física en aquestes. El principal desafiament és que en afegir les implicacions de la implementació física a l'espai de disseny, el tamany d'aquest és fins i tot major del que era al principi.

Aquest treball es centra en el desenvolupament de les noves eines d'alt nivell de disseny, modelat i simulació de NoCs, amb l'objectiu de garbellar l'espai de disseny dels candidats menys atractius mitjanant simulacions d'alta precisió basades en l'anotació de les característiques físiques dels candidats. Com a resultat d'aquest treball, els dissenyadors NoCs seran capaos d'aprofitar un nou conjunt d'eines per a l'exploració de l'espai de disseny que redueixen el temps de desenvolupament i minimitzen els problemes de precisió mencionats anteriorment. En un primer pas, ens centrarem en el disseny i desenvolupament d'una plataforma experimental per tal d'analitzar arquitectures alternatives per al disseny de NoCs. Per això, aquest treball presenta els esforos realitzats

en el desenvolupament de models abstractes dels blocs bàsics d'arquitectures NoC amb una funcionalitat i precisió similars a les dels models equivalents amb precisió de registre (o models RTL, de l'anglès Register Transfer Level). Com a resultat dels esforos mecionats, es van obtindre dramàtiques millores en els temps de simulació sobre models RTL funcionalment equivalents. Al mateix temps, tots els models desenvolupats permeten anotar alguns paràmetres claus del procés de síntesi física (com la freqència d'operació o la latència dels enllaos), de forma que permeten evaluar qualsevol punt de l'espai de disseny de forma ràpida i precisa.

En el segon pas, mitjanant l'ús de la plataforma de simulació desenvolupada en el primer pas, les capes superiors del procés de desenvolupament de una NoC foren augmentades amb la capacitat de considerar efectes físics derivats de la implementació de una NoC sobre les seues prestacions. Mitjanant aquesta metodologia, es revisaren arquitectures i tècniques de disseny adaptades del domini de les xarxes d'interconnexió fora del xip, seleccionant les més prometedores i, en alguns casos, explotant les característiques pròpies de les xarxes dins del xip per tal d'obtindre noves solucions. Aquest pas, preliminar al desenvolupament de l'eina per a la realització d'exploracions de l'espai de disseny (o eines DSE, de l'anglès Design Space Exploration), té com a objectiu depurar les tècniques per a l'abstracció dels efectes de la implementació física de les NoCs sobre les seues prestacions. Aquest pas es va dividir en dues etapes.

D'una banda, partint d'un punt de vista purament teòric, es van analitzar les topologies més populars presents en la literatura. Posteriorment, la plataforma de simulació desenvolupada es va utilitzar per tal de demostrar i corregir les fallades de precisió anteriorment mencionades. Exemples concrets relacionats amb els problemes de precisió foren analitzats per a configuracions de NoCs relevants per a la indústria, utilitzant la metolodologia desenvolupada per a corregir-los. Per tal de demostrar el potencial de la nova plataforma per a la investigació de NoCs, l'anàlisi de topologies es va extendre a un variat nombre d'arquitectures, amb diferents tamanys, restriccions de mapatge, tecnologies en fins i tot patrons d'interacció entre hardware i software. Adicionalment, els problemes més comuns que es poden encontrar a l'hora d'analitzar el disseny físic de topologies, així com les tècniques per a corregir-los, foren

analitzatz en termes tant de prestacions com de cost d'implementació.

D'altra banda, diverses de les topologies considerades durant la investigació sobre topologies per a NoCs requereixen l'ús d'un mecanisme de control de flux basat en canals virtuals per tal de poder rendir al seu màxim potencial. Aquesta popular tècnica de control de flux ha estat utilitzada en el domini de les xarxes d'interconnexió durant diversos anys per una gran varietat de raons, i per això, s'ha proposat en nombroses ocasions el seu ús en NoCs. L'arquitectura amb canals virtuals per NoCs més comú en la literatura és importada del domini de xarxes off-xip.

Encara que és completament funcional en el context de les NoCs, aquesta arquitectura està dissenyada per a un entorn diferent, la qual cosa es tradueix en penalitzacions en termes de freqència d'operació i àrea. Per aquest motiu, proposem i avaluem una implementació de canals virtuals adaptada a les característiques de les NoCs, i conseqentment capa de millorar considerablement les prestacions, l'àrea i el consum de potència comparat amb la implementació comunament acceptada

Finalment, ens centrem en el desenvolupament de la pròpia eina per a la realització de DSE. La nostra solució a aquest desafiament és doble. D'una banda, aprofitem les tècniques desenvolupades en els passos previs relatives a l'impacte de la implementació física sobre les prestacions de les NoCs per construir una eina CAD per al suport al disseny de sistemes basats en NoCs. D'altra banda, introduïm algunes tècniques per reduir el cost de desenvolupament de nous dissenys mitjanant l'acceleració de l'exploració i tècniques per a la correcció ràpida d'errors en el procés de disseny

Chapter 1

Introduction

> *"Motivation is a fire from within. If someone else tries to light that fire under you, chances are it will burn very briefly."*
>
> Stephen R. Covey

In this chapter, we describe the reasons that have motivated this dissertation (Section 1.1). Then, we briefly define the objectives aimed by the dissertation (Section 1.2). Finally, we conclude this chapter by presenting the structure of this thesis (Section 1.3).

1.1 Motivation

The increment on the integration densities will have to be exploited to meet the computational requirements of applications from different domains. Traditionally, scalable performance was requested for high-performance microprocessors, but nowadays this request is also common to embedded computing systems. In the recent past, microprocessors designers followed two main trends in order to address this challenge. First, mono-core architectures with advanced techniques to improve their performance were employed. Second, the operating frequency of new generation designs was significantly increased with respect to previous generations. But those design trends have run out of steam. The resulting architectures present power consumption and heat dissipation levels too high to be attractive for commercial products. Also, their

complexity is increasing up to the point in which the designs are too complex and error prone, which increases the number and cost of design re-spins.

As mono-core architectures are ill-suited to solve the above exposed challenges, multi-core architectures have risen as the most elegant solution to address them. Multi-core architectures are composed by several cores that communicate with each other by means of some on-chip interconnection infrastructure, and are usually referred as Systems-on-Chip (SoC). In order to exploit the computation scalability provided by multi-core architectures, the communication bottleneck will have to be addressed. In this context, performance of gigascale SoCs will be communication dominated. Current on-chip interconnects consist mostly of shared arbitrated buses, based on the serialization of bus access requests. The main drawback of this solution is its poor scalability, which will result in unacceptable performance degradation for SoCs of medium or higher complexity (more than a dozen of integrated cores). Moreover, the connection of new blocks to a shared bus increases its associated load capacitance, increasing the energy consumption of bus transactions associated with the broadcast communication paradigm. Thus, Networks-on-Chip (NoCs) emerged as an scalable communication infrastructure that better supports the trend of the SoC paradigm [29, 49, 154]. The basic idea behind NoCs is borrowed from the interconnection networks domain, and envisions the on-chip interconnection as a network on which packet-switched communication takes place. NoCs are able to inherit design techniques from interconnection networks, which is a nice quality as interconnection networks are a very mature field. A good example of this quality that can be analyzed in the open literature is the virtual channels flow control mechanism. Virtual channels are very popular in interconnection networks. First proposed by [47] as an effective workaround for head-of-line blocking, this design technique was then proposed for a wide range of aims, like implementing fault tolerance mechanisms [38] or guaranteeing deadlock freedom in adaptive routing algorithms [52]. The original virtual channel switch architecture of off-chip networks was further analyzed in [118], and then became the *de facto* solution for NoCs, with minimal changes to adapt it to this new environment. But this practice is not free of risks. NoCs and interconnection networks impose different constraints. As an example, in NoCs links are a cheaper resource than in interconnection

networks, while buffers are much more expensive. This implies that NoC designers should adapt the techniques developed for interconnection networks, not just adopt them. Following the example of the virtual channel flow control mechanism, it is possible that either feasible implementations for off-chip networks become unfeasible in NoCs or unfeasible implementations for interconnection networks become feasible, and even competitive, in NoCs.

The design process of a SoC consists of a chain of design decisions. Among those decisions, a choice of utmost importance for global system performance is topology selection. A topology describes the connectivity pattern of networked cores, defining hard boundaries to the maximum performance achievable by the NoC as well as influencing other critical parameters like area requirements and power consumption. In particular, those boundaries stem from the combination of the theoretical properties of a NoC topology (e.g., average latency, bisection bandwidth) with the quality of its physical synthesis (e.g., link latencies, maximum operating frequency). For this reason it is very important to choose the most suitable topology for the requirements of each design.

The most popular NoC topology proposed in the open literature is the 2D-mesh, in which network cores are distributed in a grid-like structure. This topology provides great modularity and a predictable physical layout, but it is known it presents serious scalability issues as the number of network cores connected to it increases. This fact has motivated several works proposing alternative topologies for NoCs [23, 81], as well as techniques to improve the scalability of 2D-meshes by adapting them to the needs of a given design [110], or methodologies to build ad-hoc topologies for application specific designs [27, 109]. Some works based on abstract level analysis and simulation of topologies [23, 81] lack of accurate performance predictions to support their conclusions. For this reason, other works employ synthesizable Register Transfer Level (RTL) tools, instead of abstract level ones, as the backbone of their performance predictions [27]. This approach solves the accuracy issue, being able to provide signal accurate results that will consider the quality of the physical synthesis of the design. But it presents a serious drawback: although RTL models are usually very accurate, they tend to be slow and/or memory-intensive, at least when compared with more abstract models. This limitation becomes very noticeable when exploring the candidates that com-

pose the design space of any given SoC. When the possible topologies valid for a design are combined with several routing algorithms, network architectures and core placements, the number of combinations starts to be overwhelming. And this number of combinations will grow even higher when considering several technology libraries, with different aims (e.g., low power or high performance libraries). This fact points to the need of using more accurate abstract models able to provide cycle accurate performance estimations while at the same time being able to reflect the impact of the physical synthesis over performance predictions. Also, CAD tools will be needed to guide the designers towards the configurations that are more suitable to the peculiarities of each design.

Just to sum up, this dissertation will make a set of proposals intended for:

- Close the gap between performance predictions based on abstract models and real system performance by enhancing abstract modeling frameworks with layout awareness.
- Evaluate the viability and feasibility of regular NoC topologies other than a 2D-mesh, under different design constraints.
- Demonstrate the benefits of adapting design techniques to the NoC environment, as opposed to simply borrowing them from off-chip networks.
- Implement a CAD tool to assist designers in the process of selecting the most suitable network candidates out of a large design space with layout awareness.

1.2 Objectives

This section presents the objectives of this dissertation. As commented in the previous section, we expect to provide an evaluation framework for a wide range of SoC configurations, as well as demonstrating the benefits of adapting off-chip design techniques to the NoC environment. In order to do this, we pursue the following goals:

- To evaluate the viability and feasibility of several topologies for NoCs under different design constraints and architectural conditions by devel-

oping a methodology to perform fast and accurate topology explorations. Our proposal should consider several key points:

  - The proposed methodology should use a simulation framework faster than existing RTL models while achieving similar performance estimations. Our proposed simulation framework should be able to run several simulations in the same time than existing RTL models take to run a single simulation.
  - Physical synthesis impact over topology performance should be reflected with as much accuracy as possible, helping in this way to close the aforementioned gap in performance estimations.
  - The methodology should be able to support different design techniques unique to NoCs, (e.g., cross-domain clocking mechanisms or link pipelining techniques). In this way it would be possible to analyze the impact of different implementation techniques over the selection of the best candidate topology for a given design.

  In other words, we want to develop a topology exploration framework that will allow us to evaluate a wide range of topologies under different design constraints and with layout awareness.

- To show the benefits that can be achieved when off-chip design techniques are adapted to the NoC environment, even when those techniques can be directly adopted. As virtual channel flow control is a popular design choice that can be used for a wide variety of purposes, we aim to demonstrate how it can be adapted to NoCs, while achieving the following improvements over the adopted solution in interconnection networks.

  - Reduced power consumption and area cost. As the number of virtual channels of a design is increased, the power consumption and area cost of the design typically tends to skyrocket.
  - Improved scalability. The operating frequency of a classical virtual channel switch for NoCs quickly deteriorates as the number of virtual channels grows up. Our solution should have a better

scalability, helping to alleviate one of the main issues of complex switch architectures in NoCs: their maximum operating frequency.

- Lower design cost. A common drawback of most proposals of new switch architectures is the cost of implementation, validation and verification. Although this cost is usually neglected in the open literature, it is one of the main reasons that prevents new proposal to be used in real system implementations. Our aim is to provide a new virtual channel switch architecture with a low design cost, which means to use simpler blocks as the foundation of the architecture, even reusing previously designed blocks when possible.

- To design and implement a tool to automatically explore the whole design space. Starting from the system specification, it should be able to provide a detailed description of the most suitable candidates inside the design space. For this reason it is required that it implements:

  - A layout-aware topology exploration, using the methodology presented in this dissertation.
  - The capability to consider a design space as large as possible. That is, it should be able to support different SoC designs with different applications, architectures, requirements, etc.
  - A highly modular implementation, which should allow to easily change any part of the tool without affecting the rest of the modules. For example, expanding the number of supported topologies should not affect other parts of the tool.
  - A versatile simulation framework, which should be able to explore the design space as fast as possible.
  - Support to fast design re-spins, in such a way that it should be possible for the designer to switch candidates later on the design flow without having to repeat the design space exploration process. For example, this can be achieved by providing a classified list of candidates instead of a single one.

1.3 Dissertation outline

The rest of the dissertation is organized as follows. Chapter 2 describes the fundamentals of NoCs and several works related to the contributions of this dissertation. Chapter 3 presents an analysis of a set of topologies proposed as NoC topologies in the open literature, as well as a topology exploration methodology. Chapter 4 describes a new virtual channel implementation for NoCs as a way of demonstrating the advantages of the *adapt not adopt* philosophy. Chapter 5 presents the Design Space Exploration (DSE) tool developed in collaboration with the company Lantiq Deutschland GmbH. The dissertation ends with Chapter 6, which summarizes our contributions and their advantages.

f

Chapter 2

Background and State of the art

> *"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer."*
>
> Gordon Earle Moore

This chapter describes the basics and terminology for understanding the main aspects of Networks on-Chip. An in-depth view of the whole Networks on-Chip topic is not provided, since it is very complex with a multitude of aspects of no relevance for this dissertation. For this reason, we provide an overview of Networks on-Chip. We refer the reader to the established textbooks on this topic, as well as textbooks for classical off-chip interconnection networks, for further background and introductory material [31, 45, 52, 60].

This chapter is composed by two sections. Section 2.1 is devoted to the aforementioned background on Networks on-Chip, while Section 2.2 provides

the state of the art in topology exploration and design space exploration, as well as for virtual channels architectures, for Networks on-Chip.

2.1 Networks on-Chip

Moore's law states that the number of transistors that can be placed inexpensively on an integrated circuit will double approximately every two years [104]. This increment on the integration densities will have to be exploited to meet the computational requirements of applications from different domains such as multimedia processing, high-end gaming, biomedical signal processing, advanced networking services, automotive or ambient intelligence. Traditionally, scalable performance was requested for high-performance microprocessors, but nowadays this request is also common to embedded computing systems. As an example, systems designed for ambient intelligence use high-speed digital signal processing with computational loads ranging from 10 MOPS for lightweight audio processing, 3 GOPS for video processing, 20 GOPS for multilingual conversation interfaces and up to 1 TOPS for synthetic video generation. Those requirements define a computational challenge that must be addressed at manageable power levels and affordable costs [35].

In the recent past, microprocessors designers followed two main trends in order to address those challenges. First, mono-core architectures were preferred and advanced techniques were employed in order to improve their performance (e.g., executing of instructions out-of-order [135] and/or in an speculative way [111]) together with improvements to some components externals to microprocessors (e.g., placing parts of the cache inside the microprocessor chip, increasing the cache hierarchy or increasing its size and speed [147]). Second, the operating frequency of new generation designs was increased with respect to previous generations. Although it is still possible to design mono-core microprocessors based on the above exposed trends, this design model is exhausted, as it produces designs with a power consumption and heat dissipation too high for attractive commercial products. Also, the techniques required to keep the competitiveness of mono-core architectures are becoming too complex, increasing the design and implementation cost of mono-core architectures. On the other hand, new generation applications require an

increasing amount of parallel capabilities, both in high-performance microprocessors and in embedded systems, like domestic systems (e.g., multi-task systems, game consoles, mobile phones, etc.), scientific applications (e.g., genetic sequentiation, numerical calculus, etc.) and in specialized servers (e.g., digital video encoder/decoder, Internet servers, etc.). Thus, consumers of next generation microprocessors are becoming more demanding, expecting products that achieve performance levels difficult to support in mono-core architectures

In the case of domestic and high-performance systems, the power consumption has been a limiting factor for some time now, as it has a direct impact over the operating cost of any system. But with the increment of popularity of mobile devices (smartphones, PDA, even laptops, etc.) the need for providing designs with a high energy efficiency has increased the pressure of power-wise designs [76], with the aim of maximizing battery life. In addition, the power consumption effect over system cost is two-folded, as the heat dissipation cost of a system is directly related to its power consumption. As new designs become more compact, it becomes more expensive to remove the excess of heat dissipation, an effect that is aggravated by the growing cost of heat dissipation systems.

The most accepted solution to address the above exposed issues consists of simplifying the architecture of microprocessors. But in order to do this and to keep up with the ever increasing demand of performance, the design should be composed of several simple processors integrated into a single chip, in what is commonly referred as *Systems-on-Chip* (SoCs). In this way, a SoC would be composed of several cores with a manageable operating frequency, for instance by implementing simple processors for generic use and specialized ones for the heaviest tasks, instead of a big complex processor able to carry out all the task by itself. The resulting architecture will be capable of achieving similar or better raw performance than an equivalent mono-core microprocessor, with less heat dissipation and power consumption issues.

2.1.1 Systems on-Chip

A typical SoC is composed by one or more processors, several memory blocks (possibly heterogeneous) and several devices and interfaces to provide connections with the external world (e.g., USB, FireWire, Ethernet, etc.), all of them

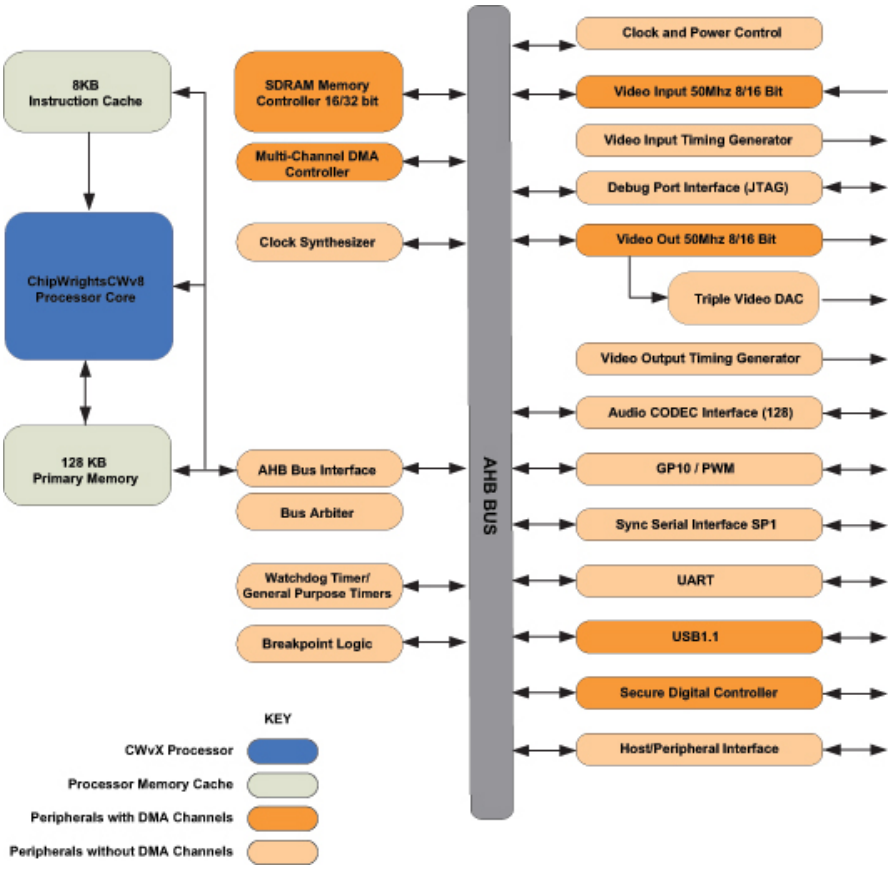


Figure 2.1: Example of a System on-Chip: CW4512

interconnected through a shared infrastructure, like a bus. As an example of a SoC architecture, Figure 2.1 shows the diagram of the CW4512 *System on-Chip* [1], which is composed by a ChipWrightsCWv8 [1] processor attached to a shared bus that interconnects the different components: processor, memories and other peripherals.

By following this paradigm designers are able to tackle the above exposed challenges. First, SoC designs are by definition better suited than classical mono-core designs to work with multi-thread applications and multi-task systems, as they are able to execute several threads and/or applications at the same time in different cores. Second, they offer a great potential to reduce the development cost of new designs due to their modular nature, either by up-

grading old designs by just re-designing a sub-set of the cores or by developing new design that reuse cores developed for previous designs. Third, SoC design are highly scalable as a wide range of characteristics can be adjusted just by altering the number and type of cores of a design, which allows designers to easily derive a whole product family from a single SoC design. Finally, the SoC design paradigm presents high power saving capabilities, as they favor designs with lower operating frequency than an equivalent classical mono-core design. Additionally, it exists the possibility to drastically reduce the power consumption of the system by shutting down idle cores transparently to applications.

Based on the type of cores that compose a SoC, this latter can be grouped into two main classes. On one hand, a SoC is *homogeneous* when composed by cores of the same type. Commonly used in general purpose systems (e.g., multi-core processors for cluster of computers for scientific computation), several examples of commercial homogeneous SoCs exists, like the *Intel Core2 Quad* [2] and the *AMD Phenom X4 Quad-Core* [3] processors, which are composed by four identical processing cores. On the other hand, a SoC is *heterogeneous* if it is composed by cores with a variety of types, like programmable processors of generic use, different kinds of memories, processors optimized for specific applications and/or hardware accelerators. The high degree of heterogeneity of those SoCs makes it possible to use the optimal core for each task. For example, a real-time MPEG4 encoder will use several specialized cores to code the video frames, while another core will code the audio track. A commercial example of heterogeneous SoC is the Nokia N770 Internet Tablet, which implements an OMAP 1710 processor [4] composed by an ARM926TEJ [5] generic processor and by a TMS320C55x digital signal processor [4].

SoCs can also be named after their target market. On one hand, *Multi-Procesor Systems-on-Chip* (MPSoCs) are SoCs aimed to the embedded system market, and therefore are heavily constrained in their power and area cost. Additionally, the extreme commercial competitiveness of this market push designers to reduce the *Time To Market* (TTM) of the new designs to the limit, favoring the use of standard cell libraries and synthesis-based design flows. On the other hand, *Chip Multi-Processors* (CMPs) aim at the high-performance market. Favoring high performance over area and power efficiency over other

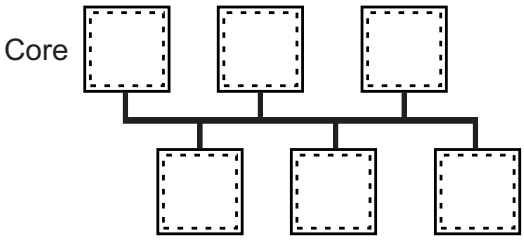


Figure 2.2: Schema of a shared bus.

considerations, full custom designs techniques are largely adopted, rendering TTM as a second rate design constraint.

In this context SoC performance is mainly defined by the cores that compose it, but the impact of the interconnection over SoC performance should not be underestimated; an inadequate interconnection infrastructure may become the system bottleneck, increasing the power consumption of the system and decreasing its performance. For this reason, the choice of the adequate communication infrastructure is a critical step in any SoC design process.

2.1.2 Shared bus

Traditionally, on-chip interconnects consist of low-cost shared arbitrated buses [21,43]. Those networks employ one of the most simple interconnection infrastructures, in which there is a unique communication medium that is shared by all the devices (see Figure 2.2). In a shared medium, transactions are serialized, as usually it is no possible to carry out more than a single simultaneous transaction, thus at any given time only a single device, called *master*, can send data through the bus, while all the other devices can only read data from the bus, acting as *slaves*. In some bus implementations simultaneous transactions are supported, but usually with serious limitations, like a low number of concurrent transactions. This serialization makes the *arbitration* mechanism a key design factor for any bus based system. The arbitration decides which master device will write in the bus when the number of masters trying to write in the bus is higher than the supported number of simultaneous transactions. The information sent through the bus is divided into three kinds: data,

address and control. It is possible to send it through a single multiplexed channel or to send each kind through an independent one. Also, bus networks are usually passive, meaning that the network does not generate messages on its own. Besides, a shared bus is naturally well-suited to work with broadcast communications, in which a single device sends the same data set to a group of devices.

Modern bus-based designs usually implement *backplane* buses, that are designed as a low-overhead communication infrastructure aimed for systems composed by a low number *master* devices and a variable number of *slave* devices. While master devices can initiate transactions slave devices are not, being limited to respond to the transactions initiated by the masters. In modern buses the most common arbitration solution is a centralized mechanism, implemented as a bus arbiter module. In this mechanism a master device must obtain permission from the arbiter module before starting any transaction. Centralized arbitration has a negative impact over the system performance due to the overhead introduced by the requests to the arbiter module, so the arbitration should be as fast and infrequent as possible.

An example of a commercial shared bus is the AMBA (Advanced Microcontroller Bus Architecture) 2.0 standard, designed for the ARM [5] processor family. This simple bus, which has been widely used in SoC designs, uses the AHB protocol (Advanced High-performance Bus) to interconnect on-chip integrated devices, like ARM processors and RAM memories. It supports differentiated transmission of data and address, both in normal and burst modes[1]. This latter is also utilized to alleviate the overhead introduced by the implemented centralized arbiter. However, the 3.0 version of this same standard supports several simultaneous transactions as well as out-of-order transaction resolution.

To summarize, shared buses are the most well-known and the most widely used communications architecture for SoCs, as they are very simple and cost effective. But their scalability is quire limited: it is a viable solution for SoCs with up to five processors and up to ten master devices [31]. For bigger systems it presents unaffordable congestion rates, becoming a bottleneck with a severe

[1]In burst mode, the master device schedules several consecutive data transfers to the same destination with a single round of arbitration

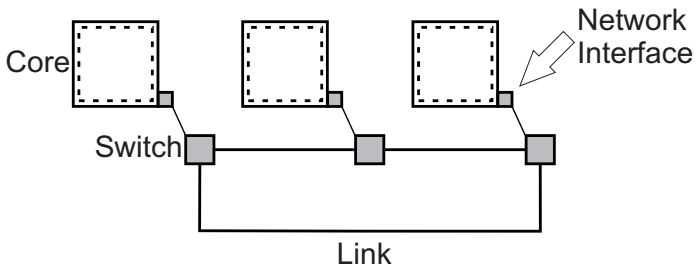


Figure 2.3: Schema of a Network on-Chip.

negative impact over system performance and increasing the system waiting time, therefore wasting a considerable amount of power. For these reasons, the performance achievable by bus-based high-scale SoCs is very limited. As an example, a possible commercial design in which a bus would be ill-suited to satisfy the system communication demands would be the MPSoC TILE64 [6], an MPSoC composed by 64 processors.

Thus, more powerful communication infrastructures are needed in order to fulfill the needs of SoCs with a growing number of components. Although it may seems that this requires to develop new solutions, the truth is that we can adapt solutions developed for similar issues in other research fields. In particular, interconnection networks are the most common high performance communication infrastructures used in other environments, like in supercomputers. The Network on-Chip (NoC) concept is born from the application of interconnection networks to SoC designs [29, 49].

2.1.3 A new design paradigm

Networks-on-Chip (NoCs) were proposed as the solution to shared bus scalability issues, emerging as the new parading for designing scalable communication infrastructures in SoCs. The main advantage of a NoC over a shared bus is its capability to reduce design costs while providing high-performance communications.

In a NoC, each core is connected by a Network Interface (NI) to a network link. Those links are point-to-point data lines connected with other links via switches. A link can connect a switch with another switch or with a NI, in such a way that it exists direct or indirect connection between any two cores.

Due to the similarities between interconnection networks and NoCs, it is possible to borrow concepts and techniques from the former and apply them to NoCs. Even so, most of the techniques developed for interconnection networks cannot be directly applied in NoCs, while other ones would be completely unaffordable using current technologies. The reason for this lies in the restrictions imposed by each environment. For example, while in interconnection networks buffers are a relatively abundant and affordable resource, in NoCs they are scarce and expensive, with high area requirements and power consumption, which hinders even more their use in NoCs. On the contrary, links are an expensive resource in interconnection networks, that must be used with austerity, while in NoCs links are more abundant. Additionally, the total area available and the tolerable power consumption bugdet of a SoC are very limited. An ideal NoC must meet the communications needs of the design while minimizing power and area consumption. This latter fact implies that the switches should be as fast, small and power-efficient as possible, which hinders even more the application of interconnection solutions in NoC designs. The architecture of each NoC component directly or indirectly affects all kinds of design parameters, like maximum operating frequency, buffering requirements, etc. The compilation of architectures of the components that compose the network is referred as *network architecture*.

2.1.4 Networks on-Chip components

Most of the terminology used in NoCs is inherited from the interconnection network domains. Transactions between cores are compacted and divided in packets, which are divided into parts of the size of a flow control unit, known as *flits*. Flits can be classified based on their position inside the packets as *header*, *tail* or *payload*. Header flits are located at the beginning of the packet, and contain all the information required to route the packet towards its destination. The tail flit is the last flit of the packet, and its main role is to act as an end-of-packet marker. Flits located between the header and the payload qualify as payload, and their function is to carry data. On the other hand, a *phit* is the minimal data that can be transmitted through a link in a single cycle. In other words, flits represent logical data units while phits represent physical data units. In the case that the phit size of a link is lower than flit size it will

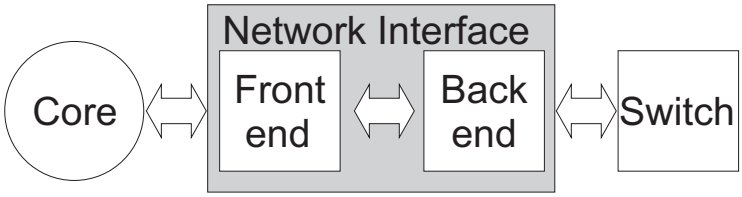


Figure 2.4: Schema of a Network Interface.

take several cycles to transmit a single flit through this link. For this reason, it is a common design technique to build architecures in such a way that flit size and phit size are similar.

As a general rule, a good NoC design should be modular, that is, composed by basic building blocks that can be combined in order to build the network most suitable for the final implementation requirements. The most common components are: *network interfaces*, *switches* and *links* [16]. Figure 2.3 shows the scheme of a generic NoC.

Network Interface

The network interface (NI) is the basic block allowing connectivity of communicating cores to the NoC. It is in charge of providing several services, spanning from session to transport layers of the Open Systems Interconnection (OSI) model (e.g., decoupling computation from communication, definition of QoS requirements of network transactions, packetization) up to the layers closer to the physical implementation (e.g., flow control, clock domain crossing).

The NI provides basic core wrapping services, acting as a layer between cores and the switches. Its role is to adapt the communication protocol of the attached core to the communication protocol of the network. Layering naturally decouples processing elements from the system where they reside, enabling the partition of a design into numerous sub-designs that can be solved concurrently, since they present a minimal degree of inter-dependency. It also facilitates core reutilization across different systems without reusing time by selecting the same industry standard interface in all the cores.

Packetization/depacketization is the very basic service the NI should offer. During the packetization process, incoming signals from the core are trans-

lated into packets compliant with the NoC communication protocol, while in the depacketization process, incoming packets from the network are converted back into signals compliant with the core communication protocol. Additionally, this service is critical in order to design modular NoCs, as it nullifyies the cost of expanding the NoC design when adding cores that use the same communication protocol. Even in the case that an existing NoC design must be modified to work with a different communication protocol, it should be enough to upgrade the part of the NI responsible for this. Notice that in order to achieve high-performance network operation frequency and to reduce implementation complexity of the network building blocks, this service should carefully optimize packet and flit size.

Another critical service provided in the NI is the clock synchronization between cores and the network, known as *clock domain crossing* mechanism. Even when the clock frequency is the same over the entire design, communications will still need for phase adaptation. In a more general case, the system might consist of multiple clock domains which act as islands of synchronicity. Clearly, proper synchronizers are needed at the clock domain boundaries. A simplified case of this scenario occurs when each communicating core coincides with one clock domain, while the network gives rise to an additional clock domain with an operating frequency which is typically much higher that that of the attached cores. A more complex case envisions a true Global Asynchronous Local Synchronous (GALS) paradigm, with a fully asynchronous interconnect fabric, in which each core and the network run at its own frequency. In both cases, clock domain crossing has to be performed at the network interfaces.

The NI is also the right place to enforce latency and/or bandwidth constraints and thus implementing *Quality of Service* (QoS) mechanisms. For instance, this can be achieved by prioritizing the reception of the messages in the attached core. In addition to those services, the NI is expected to provide the traditional transport layer services. Transaction ordering is one of them: whenever the network deliver packets unordered, it may arise memory consistency issues. In this case, NIs should reorder the transaction before forwarding data or control information to the attached core. Another transport layer service involves the reliable delivery of packets from source to destination cores. The on-chip communication medium has been historically considered

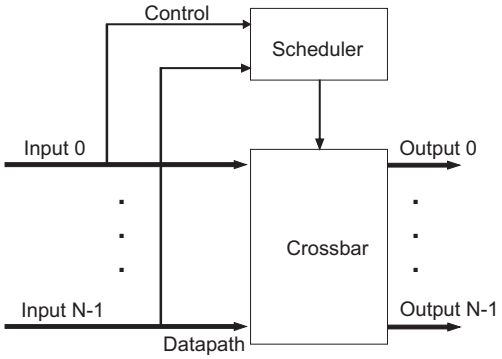


Figure 2.5: Schema of a Network on-Chip switch.

as a reliable medium. However, recent studies expect this to be no longer the case in the context of nanoscale technologies [28, 69, 103, 115, 141]. The NI is the natural place to implement some solutions for those issues, for example by implementing end-to-end error control.

Finally, NIs must be involved in the *flow control* mechanism. This mechanism takes care of regulating the flow of packets through the network: when a given buffering resource in the network is full, there needs to be a mechanism to stall the packet propagation and to propagate the stalling condition upstream. The NI is in charge of the flow control signals exchanged with the attached switch, as well as to manage the flow control signals with the connected cores. In fact, when the network cannot accept more new packets because of congestion, new transactions can still be accepted in the NI from the connected core as long as the necessary amount of buffers is available in the NI. However, when these buffers are also full, the core behaviour is impacted by congestion in the network and it has to be stalled if further communication services are required.

A generic template for the NI architecture is illustrated in Figure 2.4. As can be seen, it is composed by two sub-modules: *front-end* and *back-end*. On one hand, the *front-end* module implements a standardized point-to-point protocol. The interface then assumes the attributes of a socket, that is, an attachment interface which should capture all signalling between the core and the system (such as dataflow signalling, errors, interrupts, flags, software flow control, testing). On the other hand, the *back-end* module implements the

OSI model transport layer, for instance, establishing end-to-end communication connections. It additionally provides transparent transfer of data between cores using the services of the network layer (e.g., data packetization and computing some routing decisions at packet injection time), as well as several data link layer services. First, communication reliability has to be ensured by means of proper error control strategies and effective error recovery techniques. Second, flow control is handled at this module, either by means of upstream/downstream signals for regulating data arrival from/to the attached core and switch, or by using other techniques like buffer/credit flushing. Finally, the physical channel interface to the network has to be properly designed by considering some NoCs distinctive challenges, like clock domain crossing (exposed above), high frequency link operation, low-swing signalling and noise-tolerant communication services.

Switch architecture

Switches carry packets injected into the network to their destination. A switch forwards packets from one of its input ports to one or more of its output ports, using some sort of arbitration to solve conflicts between packets. Figure 2.5 shows a scheme of a typical NoC switch. As can be observed, each port is attached to a physical link from which it receives (input port) or transmits (output port) information. Inside the switch, input and output ports are connected through a dynamically reconfigurable switch matrix or *crossbar*, while the *scheduler* or *allocator* receives the control information from the input ports. The allocator task is to calculate which switch input will be connected to which output at any given time and to configure the crossbar accordingly. An allocator assigns an output port to an input request when the input port is requesting for this particular output port, considering that at most one output can be assigned to an input port and that each output port cannot be assigned to more than one input port at any given time. Any assignment that satisfies this statement is called a *match*. An allocator provides *maximum matching* when is capable of satisfying the highest possible number of requests in a single cycle. In this scheme, flits received at the input ports are evaluated in the allocator, which will configure the crossbar so that each flit reach the right output port. The term *switch radix* defines the number of ports of a

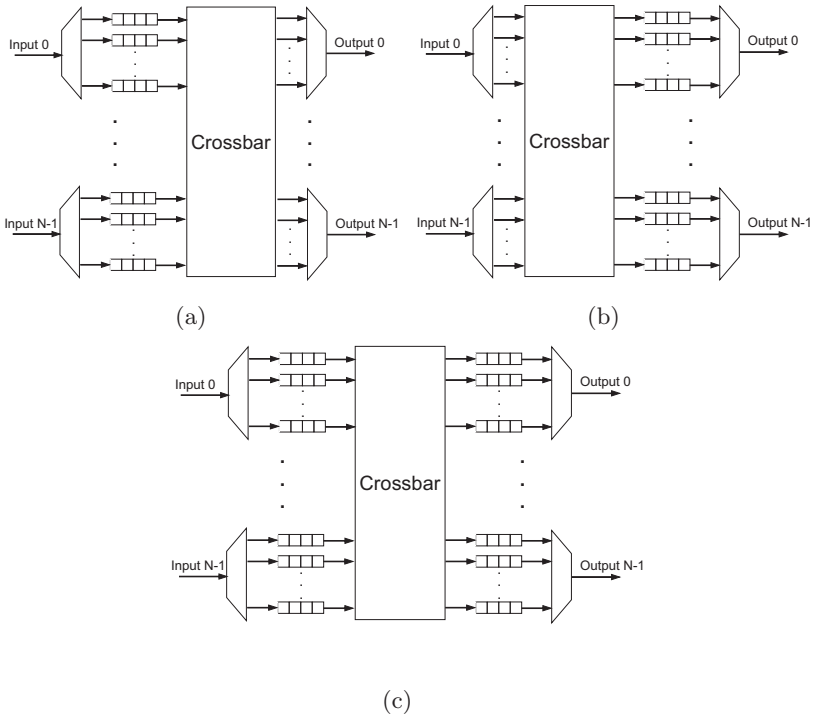


Figure 2.6: Switch architecture by buffer allocation: (a) Input queued (b) Output queued (c) Input/Output queued

switch.

Buffer distribution inside the switch affects several characteristics of both, the switch and the network. In addition to the power and area cost, buffer in NoCs may act as pipeline stages between the links and the switch, thus decoupling their operating frequencies. As depicted in Figure 2.6 a switch may allocate buffers at the input ports, at the output ports or both:

- *Input queuing*: As shown in Figure 2.6(a), buffers are located at the input of the switch, so arriving flits can be stored before the crossbar. In this strategy, output ports are connected directly to the physical link, without the use of buffers, thus the critical path is the sum of the switch critical path and the physical link critical path. In this context, the maximum link length and the maximum operating frequency of the switch are interrelated. Therefore, input queued switches are prone to require

additional buffer resources for retiming reasons, usually in the form of link repeater stages, in order to support high switch operating frequencies and/or increased link lengths.

- *Output queuing*: As shown in Figure 2.6(b), in this case the buffers are located in between the crossbar and the output port, thus there are no buffers between the input ports and the physical link. Output queued architectures present similar issues to input queued ones, with the addition of some issues when contention is present, as flits that suffer from congestion in the input ports have no buffer where to be allocated. The straightforward solution is to drop flits that must wait due to the scheduler, but this solution is hardly acceptable in NoCs. A more elaborated solution is to propagate the scheduler control signal to the previous switch, so the latter send flits only when they can be safely stored in the output buffer, at the cost of an increased control signal delay.

- *Input-output queuing*: As depicted in Figure 2.6(c) this strategy is a merge of the two previous strategies. In this case buffers are located in every port, between the switch and the links. In this way switch operating frequency and link length are completely decoupled, at the cost of an overall higher number of buffer slots.

The *switching* technique determines the way that network resources (e.g., buffer slots or link bandwidth) are allocated to packets traversing the network:

- **Bufferless switching**: As NoC power consumption is heavily driven by buffering resources, this scheme would be the optimal solution from a power consumption view-point but it presents serious implementation issues. In particular, whenever a packet cannot proceed on its way to destination due to a conflict with another packet, it should be either discarded (which is not acceptable for on-chip networks) or misrouted. Anyway, some recent work exists that addresses those issues in the NoC environment [67].

- **Circuit switching**: This is the most basic switching technique. In this case, a path between origin and destination cores is reserved in a

first step; once the path is established, the packet is sent in the second step, guaranteeing that the route is free of contention. In this switching technique, the latency of each transaction is penalized due to the path establishment cost. Additionally, in some cases the network may suffer from low utilization due to long-term path reservation, like when an answer for the packet that reserved the path is expected. Nevertheless, circuit switching requires minimal buffer resources only for two reason: store control packets (e.g., packets used to reserve the path) or as retiming stages to cope with long critical paths. This scheme is usually employed when the traffic pattern is burst intensive or when the connection patterns are relatively static [155], that is, in the cases in which the cost of reserving the path is alleviated by a high path utilization.

- **Packet switching**: In this scheme an entire packet is received and stored in a switch before forwarding it to the next switch. This is the most inefficient switching scheme in terms of latency and memory requirements, except for those systems in which the maximum packet length is small enough to compensate the need to store the whole packet. On the other hand, this scheme does a good work in high network contention situations, as a stalled packet will be stored inside a single buffer of a single switch.

- **Virtual cut-through switching**: This scheme is similar to the packet switching scheme, but in this case a packet can be forwarded as soon as the header is receiver and the next switch guarantees that the whole packet can be allocated. In this way, packet switching latency issues are addressed while keeping contention management capabilities, but it still presents high memory size requirements, since switches must be able to store entire packets.

- **Wormhole switching**: Like in virtual cut-through switching, a packet can also be forwarded as soon as the header is received, but flits are sent as soon as they can be allocated in the next switch, even when the full packet cannot be allocated. Notice that in case the header flit is stalled, subsequent flits have to wait at their current locations and are therefore spread over multiple switches, thus blocking the intermediate

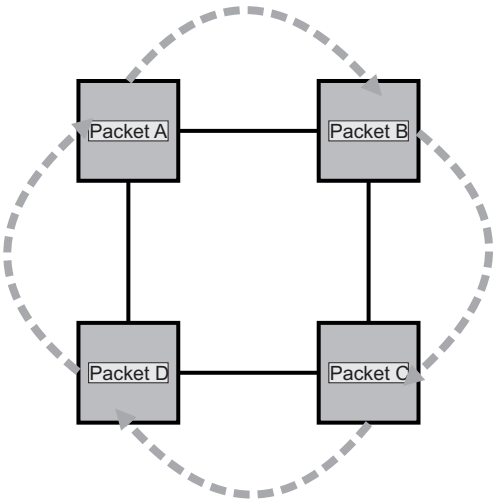


Figure 2.7: Example of a deadlock of 4 packets in a mesh topology.

links and switches. This scheme avoids buffering the full packet at one switch and keeps end-to-end latency low, therefore fixing the main issues of the above mentioned switching schemes, but at the cost of being very sensitive to packet contention due to conflicts in the switches as well as contention induced by cores with a slow packet absorption rate.

Finally, the routing algorithm has a heavy impact over the system performance. The main function of the routing algorithm is to define the paths that packets must follow across the network in order to reach their destinations. Due to deadlock issues, some algorithms are easier to implement than others. Deadlocks may arise when a set of packets cannot advance through the network because the needed network resources are reserved by one or more different packets from the same set. Figure 2.7 shows a classical example of deadlock. The figure represents four packets (denoted as A, B, C and D) that are in deadlock. The next switch for each packet is pointed by a dotted arrow. In the example each packet is allocated in the switch input port, utilizing all of its buffering resources, thus preventing a new packet to be received until some of those buffering resources are freed. As can be observed, it is impossible to recover from this situation unless the cycle is broken by removing a packet. As a general guideline, the higher the number of loops and cycles of a topology

(either introduced by the routing algorithm or intrinsic to the topology wiring pattern) the more it is deadlock prone.

Routing algorithms can be classified as *source* or *distributed*. In *source* routing, the path of the packet across the network is fixed and embedded in the header before packet injection into the network, thus increasing packet size. In this case switch architecture is rather simple, as the only functions it has to perform are reading the header and selecting the right output port, providing minimal routing latency. On the other side, in *distributed* routing the header contains only the destination identifier, and the path is not fixed but dynamically calculated at each switch, which has to decide the optimal output port for each destination. This routing mechanism can be implemented in several ways, being the most common the use of local routing tables at switches or dedicated logic in order to calculate the routing algorithm. Additionally, both routing schemes can employ *deterministic* and *adaptive* routing algorithms. The selection of the routing algorithm also has a heavy impact over the system performace, as it selects the paths that packets follow through the NoC in order to reach their destination. In *deterministic* routing algorithms, the path between any origin-destination pair is always the same regardless the network status, therefore packets cannot avoid neither congestions nor faulty links. On the contrary, in *adaptive* routing paths are not fixed, so packets can avoid potentially problematic spots in the network. While source routing enables simple and fast switches, it presents scalability issues, and makes the implementation of adaptive routing algorithms difficult. On the other hand, distributed routing favors the implementation of adaptive routing algorithms at the cost of producing more complex and slower switches. Finally, deterministic routing algorithms provide worse performance than adaptive ones, although these latter present a higher implementation complexity.

Links

A link is a data line connecting either two switches or a NI and a switch. Links can be considered as *unidirectional* or *bidirectional*, based on data being transmitted in a single or in both directions, respectively. A bidirectional link can be implemented either as a link composed by two independent unidirectional links (one for each way) or as a single data line in which transmission

can be carried out in both directions [31].

The maximum link length of a SoC is a limiting factor of system performance, as in current nanoscale technologies long wires increasingly determine the maximum clock rate, and hence performance, of the entire design. This problem increases as new technologies become available, in fact, it has been estimated that only a fraction of the chip area (between 0.4 and 1.4%) will be reachable in one clock cycle in the foreseen future [19]. A solution to overcome this problem consists of pipelining interconnects [39, 131]. Links will be partitioned into segments (by means of repeaters stages) whose length satisfies pre-defined timing requirements (e.g., desired clock speed of the design). In this way, link length is converted into link latency, but the NoC operating frequency is not bounded by the link length anymore. That is, the latency of a link connecting two modules may turn out to be more than one clock cycle.

In this context, the optimum buffer size required to provide maximum link utilization is directly related to the link latency and to the *flow control mechanism* implemented [52]. The flow control mechanism controls the flit injection rate at the links, and manages stalling conditions by stalling packet propagation and propagating the stalling conditions upstream. Several flow control mechanisms have been proposed in the open literature:

- **XON/XOFF**: In this flow control mechanism the receiver notifies the buffer slots available to allocate new flits to the transmitter. The receiver attached to the link sends a XOFF signal when the buffer utilization is above a given maximum threshold, at this point, the transmitter stops the injection of new flits in the link until it receives a XON signal, which is generated by the receiver when the buffer utilization is under a given minimum threshold. As the information control must travel through the link in the opposite direction, it is also affected by the link latency. In order to guarantee an uninterrupted flow of flits between devices, the XON signal has to be raised only while the receiver buffer has enough flits stored to cover the reception time of the signal at the transmitter side and the fly time of the first flit from the transmitter to the receiver. That implies that the minimum number of slots in the receiver buffer must be twice the link latency. In the case of the XOFF signal, the receiver buffer must allocate all the flits that can be received in the time frame

elapsed between the signal activation and the reception of the signal in the transmitter, that is, a maximum of a flit per each latency cycle of the link. Overall, to avoid the presence of waiting cycles introduced by the XON/XOFF flow control mechanism, that is, to achieve a sustained throughput of one flit per cycle, the minimum number of slots of the receiver buffer is three times the link latency.

- **Credit based**: In this flow control mechanism the transmitter has the number of flits (credits) that can be stored in the receiver. Each time the transmitter sends a flits it spends a credit, and new flits can be injected only if there are available credits. In this way, it is assured that the flits injected in the link can be allocated in the receiver. At the same time, the receiver sends a new credit to the transmitter each time it frees enough buffer space to allocate a new flit. In this mechanism, a sustained throughput of one flit per cycle can be achieved when the receiver has a minimal number of buffers of twice the link latency plus one. In this way, it is guaranteed that the first credit consumed by the transmitter is returned by the receiver before the transmitter runs out of credits.

- **ACK/NACK**: This flow control mechanism manages transmission errors in addition to perform data flow control duties. The transmitter stores a copy of each flit sent, and the receiver sends an ACK signal to the transmitter whenever a flit is successfully received and stored. When this signal is received by the transmitter, the copy of the flit that triggered it can be destroyed. The NACK signal is generated by the receiver in case of transmission error or when its buffer is full, discarding the flit that triggered the signal as well as any following flits. When the transmitter receives the NACK signal, it keeps sending the same copy of the flit that generated the signal until it reads an ACK signal. Due to the fact that the transmitter keeps sending flits in presence of a NACK signal and to the need to temporary store copies of the sent flits, this mechanism is highly inefficient in terms of power consumption. In this mechanism, a sustained throughput of one flit per cycle can be achieved only if the transmitter can allocate a number of flits equal to twice the

	XON/XOFF	Credits	ACK/NACK	STALL/GO
Transmitter	–	–	$4 * Lat$	2
Receiver	$3 * Lat$	$(2 * Lat) + 1$	–	–
Repeater Stage	$Lat - 1$	$Lat - 1$	$Lat - 1$	$2 * (Lat - 1)$
Overall	$(4 * Lat) - 1$	$(3 * Lat)$	$(5 * Lat) - 1$	$2 * Lat$

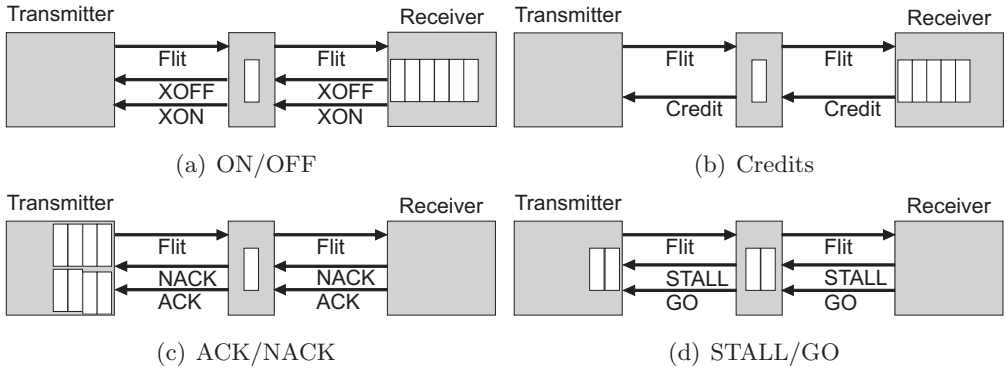
Table 2.1: Minimum buffer requirements of flow control mechanisms by link latency.

link latency.

- **STALL/GO**: In order to implement multi-cycle links, all the above exposed flow control mechanisms employ simple repeater stages, which have enough space to allocate a single flit. The STALL/GO flow control mechanism presented in [122] is an example of a flow control mechanism that requires the use of more complex repeaters stages. This mechanism is an evolution of the XON/XOFF mechanism, in which the control signal is handled at each segment [2] of the link instead of being attached to both sides of the link. The STALL signal notifies a congestion situation that must stop the data flow, while the GO signal indicates the need to resume the data flow. STALL/GO physical implementation involves the use of distributed buffering techniques, as the flow control management is spread along the link instead of being concentrated in the devices attached to it. In particular, each repeater stage and each device attached to the link requires enough buffer space to allocate two flits. In order to guarantee a sustained throughput of one flit per cycle, the flow control mechanism requires a total number of buffer slots of twice the link latency. Notice that in this case, the flow control mechanism buffers include the buffer of the repeater stages, while in the previously exposed flow control mechanism the buffer requirements do not include the repeater stages buffers.

Table 2.1 summarizes the minimal buffer requirements of each flow control mechanism above exposed based on the link latency (Lat) when guaranteeing

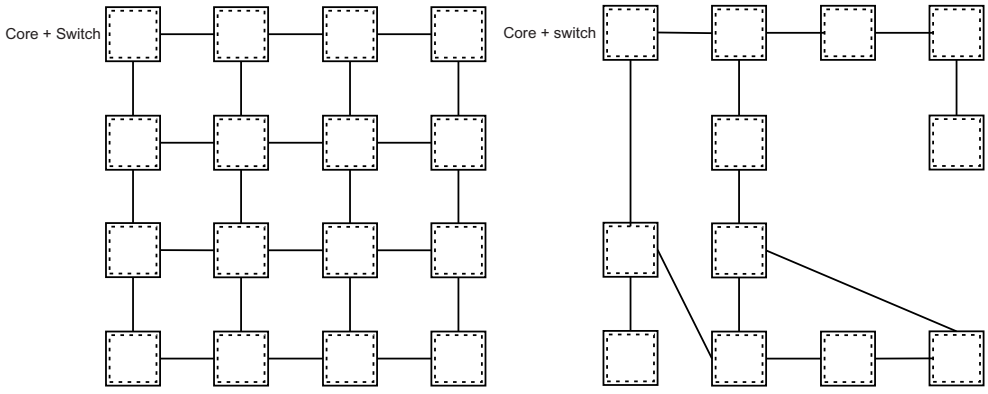
[2] A link segment is part of the link delimited by two pipeline stages (repeaters stages, switch input/ouput buffers or NI network buffers).



(a) ON/OFF (b) Credits

(c) ACK/NACK (d) STALL/GO

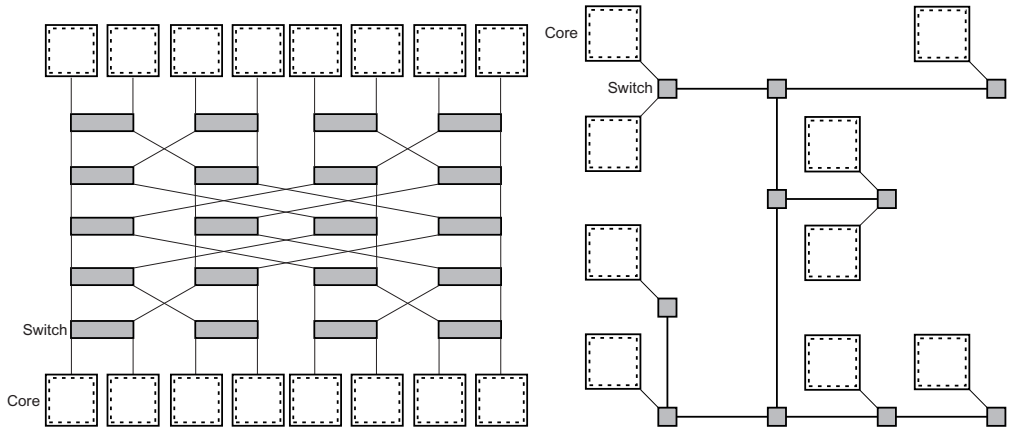
Figure 2.8: Flow control mechanisms for a link with 2 cycles of latency.



(a) Regular (b) Irregular

Figure 2.9: Example of direct topologies

a throughput of one flit per cycle. As can be observed, despite presenting the highest buffering requirements in the repeater stages, STALL/GO requires the lowest overall buffer requirements of all the exposed flow control mechanisms. For example, for an unidirectional link with 2 cycles of latency the XON/XOFF mechanism will require 7 buffer slots, the credits mechanism will require 6 buffer slots, ACK/NACK will increase the number up to 9 buffer slots, while the STALL/GO mechanism requires only 4. This later example is graphically depicted in Figure 2.8.



(a) Regular (b) Irregular

Figure 2.10: Example of indirect topologies

2.1.5 Topology

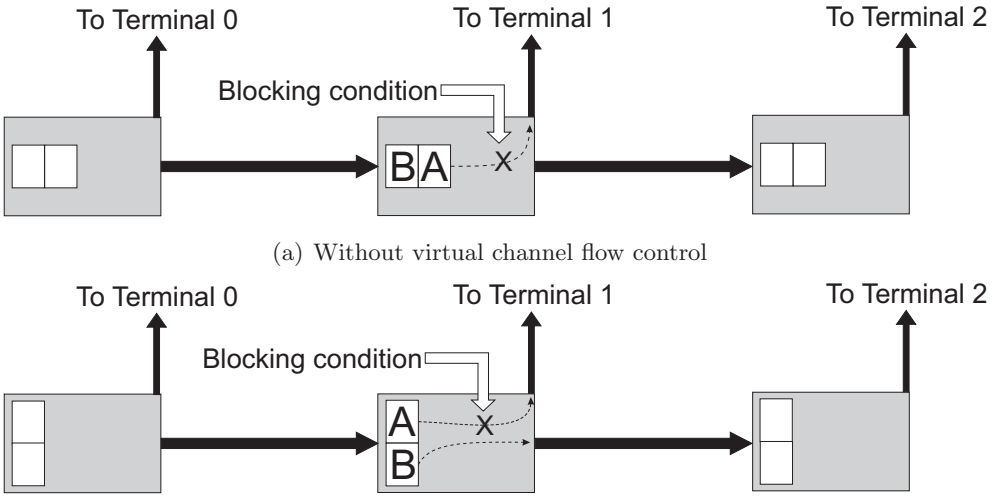
A topology defines the number of links, switches and NIs that will compose the system, as well as their connection pattern. That is, the topology defines which switches are attached to which NIs or other switches by means of network links. The implications of this statement are wide, affecting several key design parameters of any NoC design. For instance, as the topology defines the number of connections of a switch, it will have a direct impact over the switch performance, area and power consumption. Moreover, the topology definition of the connection pattern will surely affect the layout of the design, thus becoming a key factor for maximum system link length, which will directly affect the maximum achievable operating frequency or even the design buffering requirements (in case link pipelining techniques are implemented). In this context, performance and cost of any given NoC-based solution can be strongly improved at design time by selecting a suitable topology for the system at hand. In fact, together with the network architecture, the network topology is one of the most important factors for NoC performance and power/area consumption.

Network topologies can be classified as *direct* or *indirect* based on their implementation. In *direct* topologies (see Figure 2.9), each core is directly connected to the network since it also includes a tightly coupled switching

fabric for connection to its neighbor cores. In *indirect* topologies (see Figure 2.10), the core accesses the network indirectly through a link that is connected to a switch, in turn connected to the network. In indirect topologies, switches can be classified into two general groups: the group of switches that connect to core and other switches, and the group that connects only to other switches. Notice that direct topologies may not need the use of NI modules, as cores and switches are integrated on the same block. On the other hand, in indirect topologies core and switch blocks are clearly decoupled, favoring modular designs.

On the other hand, regardless of their implementation, topologies can also be classified based on their interconnection pattern as *regular* or *irregular* (see Figure 2.9 and Figure 2.10). In *regular* topologies there is a regular predefined pattern that defines the way links connect switches with other switches and/or core, while *irregular* topologies have no predefined interconnection pattern. Usually, regular topologies show better scalability than irregular ones, although their main advantage is topology re-usability and reduced design time. While regular topologies are most suited for general purpose designs [144], irregular topologies are typically used in the embedded computing domain, where NoCs are customized for the communication load of a specific application or set of applications that are known at design time [30]. Although regular topologies are the typical choice for high-performance CMPs, there are external factors to the system design process that may affect the regularity of the topology like power management, and permanent or transient errors. In particular, errors may happen in new systems or along the working life of the system, affecting the network topology. When these errors are not serious enough to render the system unusable, the resulting topology will likely become irregular. Although it is impossible to predict when and where an error will happen at NoC design time, it is possible to design fault tolerant NoCs, with routing algorithms that adapt to changes in the interconnection pattern, alleviating the system performance degradation in presence of errors.

Most popular topologies in CMPS are orthogonal, in which switches are arranged in , so designers can quickly change the candidate without having to repeat the design space exploration processan orthogonal n-dimensional space, in such a way that every link produces a displacement in one dimension. Their



(a) Without virtual channel flow control

(b) With virtual channel flow control

Figure 2.11: Head-of-Line blocking.

main advantage is their natural support for simple routing algorithms and their relatively simple implementation. Figure 2.9(a) shows an example of a direct regular orthogonal topology: the 2D-mesh. Currently, the 2D-mesh is the most popular topology for CMPs, due to its scalable wiring connection pattern, which translates into simple layouts with easily predictable physical parameters. On the contrary, among indirect topologies, multistage networks (non-orthogonal topologies in which cores are interconnected through a number of intermediate switch stages, as depicted in Figure 2.10(a)) are well known for their high performance scalability properties.

2.1.6 Virtual Channels

Virtual channels are an appealing flow control technique that allows to potentially avoid deadlock while improving link utilization and network throughput. However, their use in NoCs is still controversial, due to their significant overhead in terms of area, power and cycle time degradation, particularly in the heavily constrained MPSoC domain.

As exposed, a typical switch architecture associates a single buffer with each link, either at the receiver end, the transmitter end or both. In those ar-

chitectures, Head-of-Line (HoL) blocking is a significant performance limiting factor in NoCs, due to the use of FIFO queues. This issue raises when two packets are allocated in the same input port and one of them is forced to wait due to a congestion that is not directly involved in its path. An example of HoL can be observed in Figure 2.11(a). This example depicts two packets (A and B) with different destinations, which are currently allocated in the same buffer of the same switch. The destination of Packet A is Terminal 1, while packet B is addressed to Terminal 2. As can observed, packet B is waiting for packet A, while packet A is waiting for some blocking condition to disappear in order to be delivered to its destination (Terminal 1). That is, packet B is waiting due to a congestion condition that is not directly in the path to its destination (Terminal 2), but because the packet in the head position of the buffer is blocking the whole buffer. Virtual channel flow control was firstly proposed by [47] as an effective workaround for HoL blocking.

A virtual channel is a buffer that can allocate flits of a packet and store its associated state information [48]. Several virtual channels can be multiplexed over a single physical link, decoupling buffer allocation from link allocation. Figure 2.11(b) shows how virtual channel flow control would help in the previous example. As can be seen, the use of virtual channels allows packet B to overcome packet A, thus avoiding unnecessary delays due the blocking condition affecting packet A. Notice that the use of virtual channels increases the complexity of the switch arbiter, as the arbiter must allocate not only an output port of each packet, but the required virtual channel inside that output port. This additional step in the arbitration phase is usually referred as *virtual channel allocation*.

Nevertheless, the use of virtual channels is not limited to overcome HoL blocking. Their main advantage is their versatility. Along the years, virtual channels have been proposed for a wide range of purposes in the open literature, like as part of Quality of Service (QoS) mechanisms, or to provide deadlock freedom in adaptive routing algorithms. But in NoCs, virtual channels must be used with care, since its implementation requires extra buffers, the switch power consumption may increase with the number of virtual channels [25]. Moreover, virtual channels may arise additional complications that must be addressed, like not delivering packets in injection order, increasing the

critical path to unacceptable levels or introducing race conditions that may affect the coherence of the memory registers.

2.1.7 Networks on-Chip performance evaluation

When comparing the performance of systems with different characteristics (e.g., architecture, topology, ...) there are several parameters to consider. This section summarizes the two main metrics that are usually used in this dissertation when evaluating the performance of two or more systems.

The first metric is the *latency from generation*, that is defined as the elapsed time from the packet injection into the network interface (its generation) until its reception at the destination core. On the other hand, the *network latency* of a packet is defined as the elapsed time from the packet header introduction into the network until its reception in the destination core network interface. Unless otherwise stated, from now on we will refer to latency as the latency from generation, as it is a more accurate metric of the system performance. Additionally, the *average latency* is defined as the average of the latency of all the packets sent and received during the system evaluation. Usually, the lower the average latency the better performance the system achieves.

The other performance metric used in this dissertation is *throughput*. It is defined as the amount of information that the network delivers to the cores per time unit. In order to provide fair comparisons, this metric must be made independent of the system size, so throughput is measured as the total information delivered to all the cores divided by the total number of cores per time unit. Usually, the higher the throughput of a system configuration, the better.

2.1.8 Networks on-Chip physical synthesis evaluation

The evaluation of design techniques for NoCs is coupled with the evaluation of their physical characteristics. At some points, this dissertation presents and evaluates the results of the physical synthesis of different NoC designs. This section clarifies some concepts required to fully understand those results. Please notice that evaluation of core synthesis is outside the scope of

this dissertation. For this reason, when synthesizing NoC designs, cores are represented as black boxes obstructing their corresponding area in the design floorplan.

Timing closure is the process in which the design is modified to meet its timing requirements. Most of the modifications are handled by the backend tools based on directives given by the designer. This term is also used for the goal that is achieved when the design has reached the end of the toolflow and its timing requirements are satisfied. The main steps of this process are: *logic synthesis*, *floorplanning*, and *placement&routing*. The first step is to perform *logic synthesis*, where an RTL description of the design is changed into a gate-level netlist (i.e., logic-gates and specification of their interconnection). At this level, the real wire length is ignored, hence the wire load is simply estimated by means of abstract (and often largely inaccurate) models. Post-synthesis results however provide a rough idea about the critical path delay and the cell area. The next step is *floorplanning*, which defines where design blocks are placed within the chip area (in this case, IP cores, switches, network interfaces and pipeline stages, if any). In our experiments, this task is carried out by defining rigid fences that limit the area where the cells of each module can be placed. From a practical viewpoint, this turns out to be the best approach for extremely regular designs like an on-chip network. Finally, *place&route* performs the physical placement of standard cells on the layout (determining, among the other things, row utilization), followed by the interconnection step between them. At this stage, the final wire routing is performed between the placed cells and the actual wire load capacitance can be quantified. Post-place&route analysis provides accurate information regarding floorplan area and critical path delay of the design and of its sub-blocks, as well as accurate estimations of the power consumption.

2.1.9 Reference architecture

Due to the wide impact of the network architecture over NoC capabilities, NoC design flows should be able to adapt the NoC architecture to the constraints of the design at hand. This requirement introduces a new degree of complexity in the design flow, as the consequences of the physical implementation of each possible architecture should be carefully evaluated. Nevertheless, it is not in

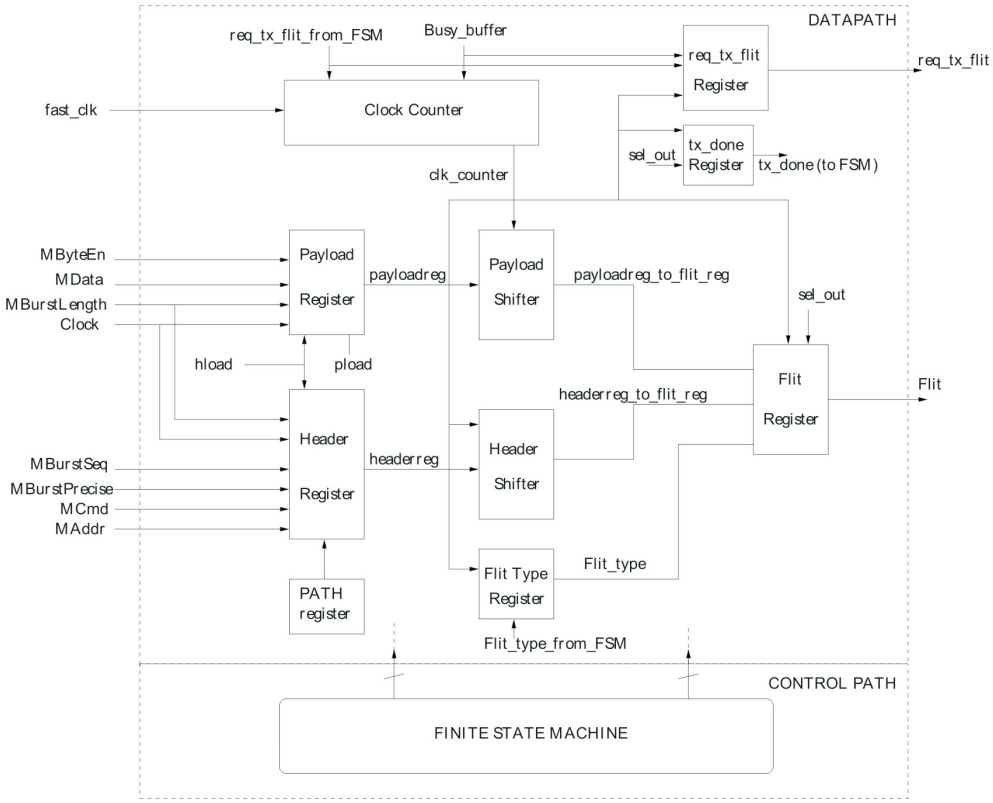


Figure 2.12: Xpipes Lite network interface initiator architecture.

the scope of this dissertation to evaluate the performance of each possible NoC architecture, but to provide a methodology and a framework to build design space exploration tools that can be easily expanded. In this context, it is convenient to resort to a NoC reference architecture when analyzing specific techniques (i.e., analyzing the cons and pros of different network topologies) in order to reduce the complexity of the evaluation process by removing the need for network architecture validation and verification. For this reason, unless otherwise noted, this dissertation uses the xpipes Lite NoC architecture [140] as its reference architecture.

The NI of the xpipes Lite architecture implements a lightweight architecture for best effort NoCs, thus no error control mechanism is considered. Designed as a bridge between an Open Core Protocl (OCP) [7] interface and the NoC switching fabric, its purposes are:

a) Protocol conversion between OCP and network protocol.
b) Packetization.
c) Computation of routing information (stored in a Look-Up Table, LUT).
d) Clock domain crossing.
e) Flit buffering to improve performance and flow control.

The xpipes Lite architecture implements OCP traffic generators. In the OCP protocol, there are two kinds of transactions: request and response. Request transactions are blocking, which means that whenever an OCP core starts one of them, this core will be unable to carry out other tasks until the response for that particular transaction is received. Additionally, for any given transaction, some fields have to be transmitted once, while other fields need to be transmitted repeatedly, for instance, in an OCP burst transaction the address field is transmitted once, while the data field is trasnmitted once per burst beat. There are two kinds of NIs: *initiator* and *target*. While the former are attached to communication initiators, the latter are attached to targets.

Figure 2.12 depicts the network interface initiator of the xpipes Lite architecture. The NI is built around two registers: one stores the transaction header (1 refresh per OCP transaction), while the second one stores the payload (refreshed at each OCP burst beat). During the packetization process, a set of flits encodes the header register, followed by multiple sets of flits encoding the payload register for each new burst beat. The xpipes Lite architecture employs source based deterministic routing, therefore routing information is attached to the header flit of a packet by checking the destination address against a LUT. The length in bits of the routing path depends on the maximum switch degree and the maximum number of hops in the network instance at hand. The header and payload registers represent the boundary between the NI front-end and the attached OCP device. But they also act as clock domain decoupling buffers, as they can be read from the NI back-end at a much higher speed than the writing speed of the OCP side. In practice, network and attached cores can operate at different frequencies. The only constraint posed by this architecture is that the OCP frequency is obtained by applying an integer divider to the network frequency (an approach which is called frequency-ratioed synchronization).

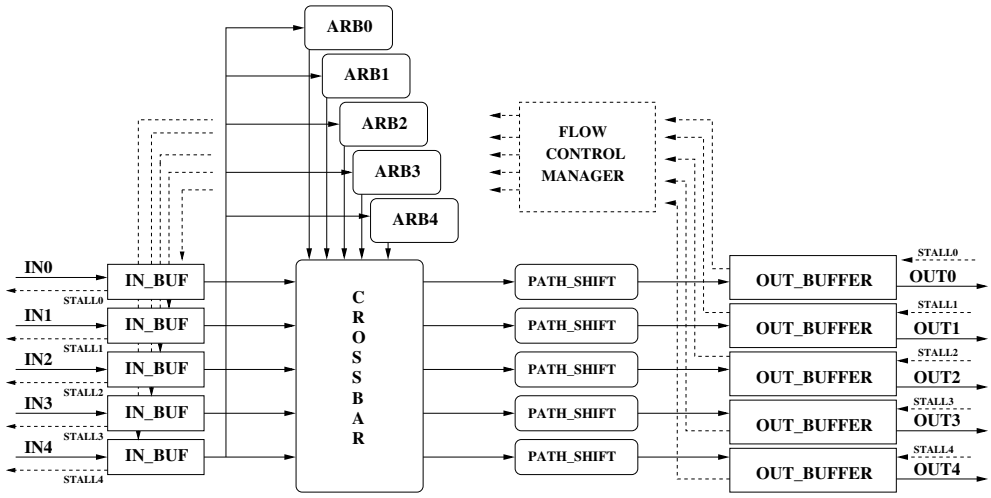


Figure 2.13: Xpipes Lite switch architecture.

The xpipes Lite switch architecture is represented in Figure 2.13. It features a latency of one cycle for traversing from the input ports to the output ports, and another one for traversing the output link, thus traversing the switch fabric overall has a minimum cost of 2 cycles. The implemented buffer allocation scheme is input-output queuing, therefore allocating a buffer in each output port and just a repeater stage in each input port, which are latched to break the timing path. The selected switching scheme is wormhole switching, as it provides low latency communications while requiring minimal buffering resources. The use of a static source routing scheme keeps the switch routing logic minimal: it simply has to read the routing information from the header and then perform the arbitration process. This latter process is handled by an allocator module for each output port based on a round-robin priority algorithm. After a packet wins the arbitration, the routing information concerning the current switch is rotated away in the header flit by the path shifter modules. This allows to keep the next hop at a fixed offset within the header flit, thus simplifying the switch architecture. At this point, access to the output port is granted until the tail flit arrives. *The critical path of the switch starts from the input buffer, goes through the arbiter, the crossbar selection signals, the header path shifters and finally includes a library setup time for correct sampling at the output port.*

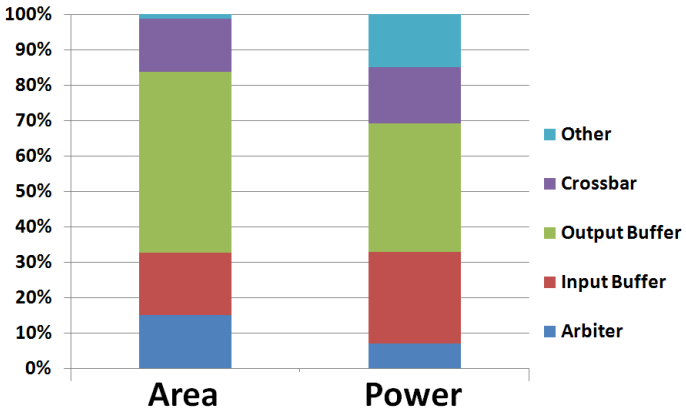


Figure 2.14: Area and power breakdown.

The xpipes Lite architecture is parameterizable in the number of input and output ports, in the link width and in the output buffer size, as well as supporting several flow control mechanisms. Anyway, as demonstrated in [122], the STALL/GO flow control mechanism proves to be the most low-overhead and efficient flow control mechanism in non-error tolerant designs, thus being the most suitable for the needs of this dissertation.

Fig.2.14 illustrates the area and power breakdown of a 5x5 xpipesLite switch synthesized for maximum performance on a 65nm STMicroelectronics technology library. This figure is reproduced for a better understanding of the architecture design techniques that will be exposed in this dissertation. Due to the maximum speed required, combinational circuits are inferred into area-expensive gate level netlists. The crossbar takes about 15% of the total area, while input and output buffers all together take about 68%. Power was measured post-place&route with a 50% switching activity on all input ports. The contribution named *other* coincides almost entirely with the clock tree power. In relative terms, the arbiter's contribution to total power is lower than to total area, while clock tree plus input/output buffers consume approximately 70% of total power. This breakdown reflects typical conditions for wormhole switches in the MPSoC domain (e.g., [24]), while at network level the clock tree typically plays a greater role in determining total power [93].

2.1.10 Design challenges for Networks on-Chip architectures

The evolution of NoC design techniques and tools will be driven by the design challenges imposed by future design objectives and technologies. This section presents a non-exhaustive list of future design challenges. While some challenges will be tackled in this dissertation, some others will not, as they fall outside its scope.

Technology challenges

Link performance is likely going to be the system performance bottleneck in the near future. While gate delays scale down with technology, wire delay typically increases, forcing an increment of link repeaters in order to keep viable operating frequencies, as the maximum operating frequency can be preserved by breaking long timing paths.

Another technology related issue concerns global synchronization of large multi-core chips. The difficulty to control clock skew and the power associated with the clock distribution tree will probably cause a design paradigm shift toward *Globally Asynchronous and Locally Synchronous* (GALS) systems [71]. The basic GALS paradigm is based on a system composed of a number of synchronous blocks designed in a traditional way (and hence exploiting standard synthesis methodologies and tools). However, it is assumed that clocks of such synchronous systems are not necessarily correlated and consequently that those synchronous systems communicate asynchronously. Locally synchronous modules are usually surrounded by asynchronous wrappers that handle inter-block data transfers. Practical GALS implementations may form much more complex structures, such as bus [149] or NoC structures [87], for inter-block communications, and use different data synchronization mechanisms. Additionally, the GALS paradigm does not rely on absolute timing information and therefore favours modularity, by means of local islands of synchronicity that can be arbitrarily combined to build up larger systems [83]. NoC architectures are generally viewed as an ideal target for application of the GALS paradigm [87]. To summarize, GALS NoCs are a promising means of tackling a number of interconnect issues, from power reduction to clock skew management, while preserving design modularity.

Finally, signal integrity issues (cross-talk, power supply noise, soft errors, etc.) will lead to an increment of transient and permanent failures, thus raising the reliability concern for NoCs [33]. In many cases, NoCs can be designed as regular simple structures in which the electrical parameters of wires can be easily optimised and well controlled. But even in those cases, communication failures can occur as an effect of the deviation of technology parameters from nominal design values (such as effective gate length, threshold voltage or metal width). In fact, precise control of chip manufacturing is becoming increasingly difficult and expensive to maintain in nanoscale technologies. This produces performance and power variabilities that result in increasing yield degradation. Providing support for process variation tolerance in NoCs poses unique design challenges. From a physical viewpoint, network circuits should be able to adapt to the technology conditions, for instance by means of the self-calibrating techniques proposed in [133, 156]. From an architectural viewpoint, one or more sections of the network might be unusable, requiring routing and topology solutions able to preserve interconnection of operating nodes, like the ones presented in [61].

Scalability challenges

The most promising approach to deliver massively scalable computation architectures, while effectively managing power and heat, is to split functions into many concurrent operations and distribute them across many parallel processing units. Those processors, known as Chip Multi-Processors (CMPs), achieve a high performance by executing many operations in parallel, while keeping affordable operating frequencies. Evidence of this CMP trend can be found in several current commercial products: AMD's Opteron™, Intel's Core™, IBM's Cell™and Power5™. In the foreseen future, performance gains will come from increases in the number of cores per chip, making the intrachip communication infrastructure a key bottleneck for the system performance.

The scalability issue in NoCs can be tackled in several ways. It is possible to propose architectural solutions aimed at improving the performance of the NoC architectural blocks, for instance by improving switch performance by removing control overheads (routing and arbitration logic) from the critical path [106]. Additionally, the parameters of the NoC can be changed

by modifying the physical interconnection pattern (i.e., different topologies), and/or the communications paths (i.e., different routing strategies), in such a way that the performance of the interconnection is improved. This issue is naturally suited to exploit the similarities between NoC and off-chip high performance interconnection networks by importing to the NoCs domains solutions developed for the off-chip domain that have been widely tested and verified. But this process has to be carefully carried out, as both environments present different enough conditions to have different gains from the application of the same techniques. As an example, the topology topic has been widely analyzed in off-chip interconnection networks, but the differences between both environments make one of the most used topologies in off-chip networks non-competitive in NoCs (as we will see in Chapter 3). And even in those cases in which an off-chip solution that can be directly applied in NoCs it is possible to exploit the distinctive characteristics of NoCs in order improve one or more parameters (performance, power consumption, etc.) of the imported solution. For this reason, techniques from the off-chip domain should not be blindly adopted into NoCs designs, but adapted by taking into account the special characteristics of the NoC domain.

Design productivity challenges

A bug advantage of NoCs is their capability to reduce design costs while providing high-performance communications. But due to the high heterogeneity of NoC solutions, the process to select the optimal solution for a given design is complex, and it is growing more complex as new technologies for NoCs are proposed. Moreover, this complexity makes it very difficult to provide accurate performance estimations of these solutions in the earlier stages of the design process, raising a potential design risk. Hence, new tools are needed to guide the designer to the best candidates inside the design space. As mentioned, the selection of the right topology is a key step of any NoC-based design. Due to the wide range of topologies available to be designed, topology exploration is likely going to be a key factor of these tools. But there are other design decisions which will impact the design viability and that are not fixed by the topology, like the selection of the optimal buffer distribution inside the switch.

In order to address this challenge, early Design Space Exploration (DSE)

is required to find appropriate system architectures out of many candidate architectures. Usually, early exploration is carried out by means of abstract level tools that provide performance estimations that the designer will use to select the most promising candidates at the earlier stages of the design flow. There are several challenges that future DSE tools must overcome.

First, the number of possible solutions for a given design increases as new technologies are introduced, further increasing the size of the design space. In order to keep design time within affordable limits, new techniques to speed up NoC design flows are needed. Second, there is a gap between the performance predictions of those tools and the real performance achieved after the system is implemented. In fact, this gap grows as the scale of new technologies is reduced, forcing to increase the number of design re-spins due to inaccurate high level predictions, which results in increased design cost. To minimize its impact, layout-aware tools must be used in each step of the design process, thus considering the impact of the physical design of the solution at hand. This raises the need for new DSE tools for NoC design, requiring the development of new abstract level models able to provide accurate layout-aware performance estimations, as well as tools to guide the designer towards the most promising candidates inside a given design space within an affordable time cost. Finally, design re-spins may occur at any stage of the design process, increasing the TTM and cost due to inaccurate performance predictions of design candidates, design errors, wrong design constraints, etc. In this context, new design tools for NoC systems should consider this fact, providing support to fast design re-spin techniques. For instance, providing a list of candidates classified by their suitability for different design constraints and/or by reducing the tool computational cost to a minimum.

2.2 State of the Art

In this section, we provide the results of our search along the open literature in the research fields that are treated in this dissertation. Due to the similarities between NoCs and off-chip interconnection networks, the adoption of off-chip solutions, in particular topologies and architectures, into NoCs is a very popular topic.

Due to the relevance of the network topology over interconnection performance and power efficiency, network topologies have been extensively studied in the past for off-chip interconnects [46, 72, 88]. In the NoC field, topology optimization for power efficiency is a very popular research domain. The authors of [151] perform an analysis on the power efficiency of several topologies, concluding that from an energy standpoint, high-dimensional tori should never be selected over hierarchical or express cubes. However, the unpredictability of actual physical parameters and the fact that different optimal topologies were indicated for different traffic patterns prevents to accurately quantify the benefits of the studied topologies. The analysis presented in [82] shows that fat-tree topology is a strong candidate to fulfill tight latency constraints. Unfortunately, the wiring complexity inherent to fat-trees compromises its viability. Ring, spidergon, crossbar and 2D-mesh topologies are compared in [36, 37], pointing out the nice properties of the spidergon topology without analyzing the physical issues and the feasibility concerns of the analyzed topologies. Mesh and torus topologies are compared in [101] under different routing algorithms and traffic models.

Still related to the topology for NoCs topic, a growing number of works propose to exploit the relatively abundant wiring resources of NoCs. In this trend, the work in [116] proposes the use of topologies with a core:switch connectivity of N:M (N cores attached to M switches), instead of traditional 1:1 (a core for each switch) or N:1 (N cores for each switch). The work in [23] proves the limited scalability of 2D-meshes and proposes a concentrated mesh architecture with several cores per switch and express channels, as well as a version with replicated subnetworks. Also, the somehow exceedingly wiring of NoCs favours the use of wiring intensive topologies, like tree-based topologies (i.e., fat-trees). Low latency communication and performance scalability are the main reasons behind the use of fat-trees in early network-on-chip prototypes, like in the SPIN micronetwork [17, 70]. Also, butterfly fat-trees are used in [113, 114] with the motivation that the number of switches converges to a constant independent of the number of cores, at the cost of an increment on the switch radix. More recently, optimized tree-based topologies 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, [95] proposes a new tree-like topology for NoCs: the fat H-tree The extension of this solution to 3D NoCs is illustrated in [96].

Due to the wide variety of topologies available for a given design, a whole research branch is dedicated to development of tools and methodologies to perform topology exploration, that is, to select the optimal topology for a given design. Topology exploration and DSE are two tightly related topics, as most DSE tools include some kind of topology exploration. In this direction, some works present roadmapping tools aimed at restricting the design space to a small subset, to be further investigated by means of more accurate analysis tools [138]. Although roadmapping tools are very useful in the early design stages, their power/area/delay estimations are necessarily based on International Technology Roadmap of Semiconductors (ITRS) projections [15], architecture-level power models [78, 150], pipeline delay models [118], analytical power models [159] or floorplan assumptions that are often referred to generic NoC architectures. Roadmapping tools relay in abstract models of key system parameters (performance, power [40, 41, 99, 112, 146, 150, 158, 159], reliability [18, 79, 145]), which should be eventually validated at lower levels of abstraction. However, the aim of pruning wide design spaces by selecting the most promising candidates justifies their low modeling accuracy. On the contrary, other works propose more accurate analysis by using synthesis tools, with the objective to select the optimal NoC for a given design and generate it [32]. These tools have a more restricted scope and need to capture physical synthesis effects very accurately, thus employing layout-aware techniques to close the aforementioned gap between the architectural and physical design phases. The authors of [77] propose a toolchain for SoCs that, starting from the traffic specifications, instantiates a model of the NoC, synthesizes it and then evaluates it in order to detect timing and performance issues. The work in [20] presents a physical planning tool that optimizes link power consumption during topology design. With a similar aim, the authors of [139] introduce a floorplanning tool based on sliced trees. The work in [108] considers the wiring complexity of each topology in order to detect link length issues early in the design process. A similar approach is used in [74] and [117] to address link delay and power issues.

During the design process of a SoC, the selection of the right architecture is a key design point. Therefore, DSE tools must consider the optimal architecture for the needs of each design, providing additional features (e.g., fault tolerance) when required. The maturity of the off-chip interconnection network research provides designers with a lot of opportunities to import solutions into the on-chip domain. For example, providing designers with architectures that implement Quality of Service (QoS) mechanisms or fault tolerance architectures. Among these imported solutions, virtual channels are an iconic example. Virtual channel flow control was first proposed as an effective workaround for head-of-line blocking for interconnection networks in [47]. Virtual channels use is widespread in interconnection networks, not only as a flow control mechanism, but as a foundation for more advanced techniques, like fault tolerance, Quality of Service (QoS) or deadlock avoidance.

The use of virtual channels has been widely proposed in NoCs, but since its implementation requires extra buffers, the switch power consumption will increase with the number of virtual channels [25]. State-of-the-art virtual channel switch architectures for NoCs are illustrated in [118]. This work also develops canonical architectures for classical wormhole switches, the original virtual channel switch architecture presented in [47] and a version of it in which the virtual channel allocation phase is carried out speculatively, thus providing a single-cycle switch architecture that looses some cycles when the speculative step fails.

Further, more efforts were carried out in order to reduce the delay and power consumption of virtual channels switch architectures for NoCs. For example, the work in [75] presents a novel approach for improving the performance of virtual channels for the cases when the target application of the design is known, consisting of the customization of the virtual channel allocation based on the traffic characteristics of the target application. The work in [90] proposes to use virtual channels for flit admission in the network in order to reduce the complexity of the crossbar. Summarizing, the basic techniques to improve switch energy and delay are: speculation [106, 118], bypassing [85, 105], lookahead [68, 84], modified virtual channels allocators [26, 80, 84, 160] and lookahead routing [62]. A brief description of each one follows:

- **Speculative** techniques try to reduce the delay by assuming that the

virtual channels are always free to be allocated, when this is not the case some cycles are required in order to recover from the speculation failure.

- **Bypassing** solutions try to avoid the virtual channel allocation overload by acting as plain wormhole switches when possible, and using virtual channels only when required.

- **Simplified virtual channel allocation** proposals aim at reducing the delay of the virtual channel allocation phase by trading off performance for reduced complexity. For instance, by providing sub-optimal virtual channels allocations with a significant reduction in its implementation complexity.

- **Lookahead** solutions exploit the available wiring density of NoCs in order to increase the control information transfer between adjacent switches. In this way, a given switch is performing the arbitration and virtual channel allocation of a packet when its header is actually still crossing the previous switch of the packet path.

Other works exist with different proposals. For example the work in [157] proposes the use of dynamic virtual channels that can be allocated by any input port when an additional virtual channel is needed, aiming at a global channel resource sharing. Although highly flexible, the proposal has a significant overhead in terms of area and power since tables (whose size depends on the packet length) are needed to track the flits traversing traversing each switch.

To summarize, the actual trend when designing virtual channel architectures for NoC is to to optimize the switch critical path by proposing complex architectures that are often derived from the original virtual channel proposal in [47]. Additionally, in many CMP NoC implementations, custom-designed crossbars are used, like the one in [148]. Custom designs do not employ standard cell libraries, thus allowing designers to produce very optimized implementations at the cost of an increased design time. Those implementations reach ultra-high operating frequencies while providing a significant reduction on power consumption over designs based on standard cells libraries.

Finally, other steps of the DSE process are addressed in the open literature. For example, core mapping is the topic of the work in [124], which aims to address it by a design space exploration methodology based on genetic algorithms. Limited to a single topology (2D-mesh) and to a single step of the DSE process (core mapping), this work proposes to use the trade-off between accuracy and performance inherent to genetic algorithms in order to quickly explore the possible core mappings that form the design space, picking up one of the most promising candidates in terms of both performance and power consumption. The accuracy of the methodology is a parameter that can be adjusted, therefore producing better mappings at the cost of an increased computational cost. Core mapping in a 2D-mesh topology is also the topic of [107]. In this work, an algorithm based on graph theory and mathematical models of 2D-mesh topologies is presented. The algorithm requires to know beforehand the exact traffic pattern of the applications to be executed in the design. These traffic pattern are later on converted into a graph model that is manipulated by the proposed algorithm in order to be mapped into a 2D-mesh while maximizing performance. Although it shows promising results, it has a narrow scope, due to its limitation to work with a single topology and known and fixed traffic patterns.

If we focus on works directly related to the DSE topic, in [102] a whole tool to perform DSE of embedded systems is presented, while the work in [94] presents a process to perform design space exploration of Very Long Instruction Word (VLIW) architectures. Both works require to have some complex estimations of the system traffic pattern, like traces. Although such traffic estimations are usually difficult to find in the earlier steps of the design cycle, their use simplifies the selection of the optimal NoC configuration for the design at hand. Finally, in [42] a full design methodology for automatic generation of heterogeneous NoC-based systems is presented. This work is aimed to a very limited market niche, as the presented methodology is limited to multi-user systems in which the user behavior with similar devices of previous generations is available, thus providing the perfect tool to effectively guide the design process of new generations of similar devices [129].

Chapter 3

Topology exploration for Networks On-Chip

"Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway."

Andrew S. Tanenbaum, Computer Networks, 4th Ed.

This chapter presents the simulation framework developed in order to perform topology explorations. After a brief introduction (Section 3.1), a short description of several topologies proposed for NoCs and their high-level properties is presented in Section 3.2. Next, the simulation framework proposed in this dissertation is introduced and validated in Section 3.3. Finally, Section 3.4 introduces the reader to the most common pitfalls of physical design of topologies for NoCs. Additionally, several cases studies are presented both, as a demonstration of the proposed simulation framework and as a mean of showing the reader the actual gap between abstract and layout-aware topology explorations methodologies.

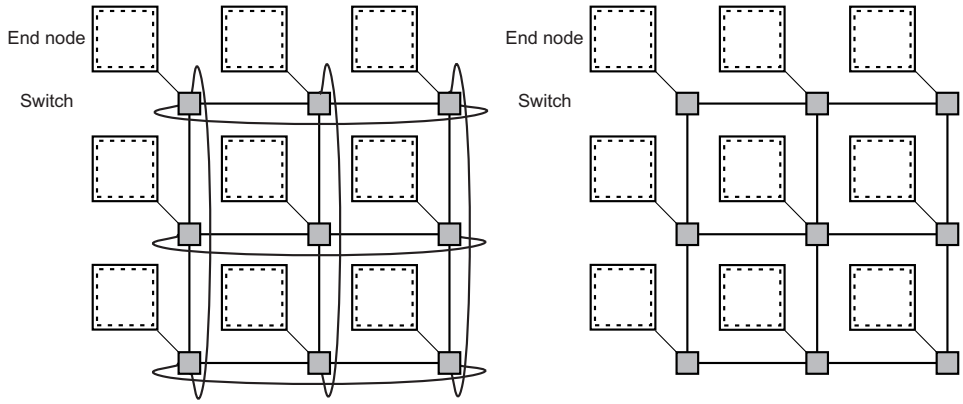
3.1 Introduction

The network topology is a key factor for the performance and cost of any NoC design. As anticipated in Chapter 2, the actual trend in CMPs designs is to

choose a 2D-mesh because of its regular connection pattern. It provides a high suitability for the bidimensional silicon surface as well as highly predictable electrical parameters. However, as the number of cores in future NoCs designs increases, this topology might not be the best choice, since the 2D-mesh bandwidth and latency scalability issues are well known. This raises the need to search for substitute topologies, as well as to assess their suitability for the design at hand by means of a topology exploration framework for use in the early design stages [32].

The topology exploration topic has been extensively analyzed in the domain of off-chip interconnection networks. However, the on-chip environment has radically different constraints than the off-chip one, especially when it comes to link characteristics, wiring cost, signaling techniques and synchronization mechanisms. In the off-chip domain, a significant cost arises from network links, and the available wiring density is quite limited. On the other hand, in the on-chip environment most of the cost is due to silicon area and power, which is typically dominated by storage resources (buffering), while wire density is certainly higher than in the off-chip domain and is not constrained by pin limitations. The gap between the constraints driving the design of on-chip vs off-chip interconnection networks (and hence the gap between the final network topology selected for use in each domain) is widening even more as an effect of technology scaling. New physical effects come into play and may either degrade performance/power in an unpredictable way or even affect feasibility of specific connectivity patterns or architecture design techniques. In particular, the delay of on-chip interconnects increases as an effect of the decreased cross-section area of links, making the performance of designs targeting sub-45nm silicon technologies increasingly dominated by the on-chip interconnection. This effect becomes more and more severe as technology scales down and tends to widen the gap between post-synthesis and post-place&route performance figures, even moving critical path delays from logic blocks to long global wires.

Performance, and even feasibility, of topologies for NoCs is extremely sensitive to on-chip specific design issues, with deep implications on the network architecture. In an on-chip setting, topologies must in fact match the 2D silicon surface, while off-chip realizations are dictated by a board/rack organi-



(a) Symmetric (b) Non-symmetric

Figure 3.1: Example of symmetric and non-symmetric topologies.

zation. The 2D mapping constraint raises implementation issues such as wire crossings, wires of uneven length, in addition to other issues, like the decrease of switch operating frequency with the number of I/O ports. As an ultimate consequence, topologies borrowed from off-chip networks should be reassessed in the on-chip environment and validated against the design constraints of this domain. This is the motivation which is at the core of this chapter. Many regular topologies feature better abstract properties (e.g., diameter, bisection bandwidth) than a 2D-mesh, however their silicon implementation is very challenging. The objective of this chapter is to quantify to which extent their inherently better abstract properties are impacted by the degradation effects of the physical implementation on nanoscale silicon technologies.

3.2 The high level view

3.2.1 High level topology properties

As the number of topologies that can be considered for a NoC increases, the need to predict their capabilities arises. Several properties exist that help to predict topology characteristics from a theoretical point of view. The rest of this section enumerates and defines some of the most common high-level properties utilized in abstract topology analysis and comparison.

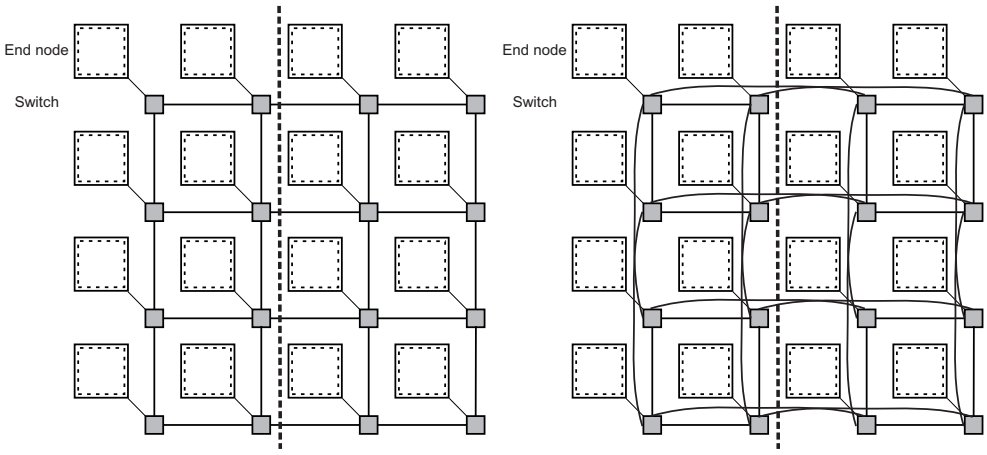
- **Symmetry**: A topology is symmetric when the network looks the same from every switch. Figure 3.1(a) and Figure 3.1(b) show two different topologies, only the first one has the symmetric property. Although symmetric topologies may provide abundant communications paths between any two cores, taking advantage of this effect while providing deadlock free routing is usually a complex task, due to the abundance of cycles.

- **Switch degree**: Defined as the total number of input/output ports of a switch. The operating frequency of a switch and its area requirements are strongly related to its degree: the higher the switch degree, the lower the switch maximum operating frequency and the higher its area cost.

- **Homogeneity**: A topology is homogeneous if all its switches have the same degree, that is, the same number of ports. Figure 3.1(a) and Figure 3.1(b) show two different topologies where only the first one has the homogeneous property. In the NoC environment, non-homogeneous topologies may present switches with different maximum operating frequency, since it strongly depends on the switch degree. This may force faster switches to work at the speed of the slower ones, or to use techniques to support switches with different operating frequencies in the same NoC. Although homogeneous topologies are more modular and in principle easier to implement, homogeneity is a desirable property but not an indispensable one. As an example, the widely used 2D-mesh is not homogeneous. This property is in fact secondary to other important qualities of a topology such as the minimization of communication resources and/or the simplification of the connectivity pattern.

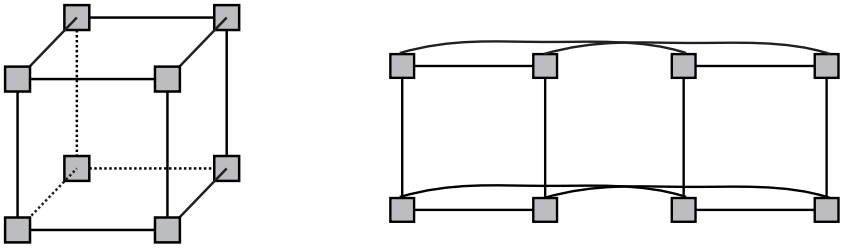
- **Bisection Bandwidth**: It is defined as the smallest aggregated bandwidth of all the pairs obtained by dividing the topology into two equal-size halves. It is a common measure of theoretical network performance: the higher the bisection bandwidth, the better the topology is suited to cope with high traffic loads. Figure 3.2 shows two examples of bisection bandwidth (depicted as a dotted line).

- **Hop count**: It is defined as the maximum number of switches that must be traversed in the topology in order to travel between any two



(a) 4×4 Mesh (b) 4-Hypercube

Figure 3.2: Examples of topology bisection bandwidth.



(a) Conceptual (b) Projected into a 2D plane

Figure 3.3: Representations of a 3-hypercube.

cores by means of minimal routes. The higher the value of this property, the longer messages take to reach their destinations and the higher the probability of collision with other messages. Figure 3.2 shows two regular indirect topologies with the same number of cores. While it takes 7 hops to travel from top-left to bottom-right in the 4×4 mesh, this number is only 5 in the 4-hypercube.

- **Diameter**: Notice that multi-cycle links may distort the hop count metric, as the hop count of a path may differ too much form the real cost in cycles of traversing this path. In order to have a more accurate parameter, the diameter of a topology is defined as the maximum distance

between any two cores in cycles. This property is completely dependent on the physical implementation of the topology. For example, Figure 3.3 shows a conceptual 3-hypercube and its projection into a bi-dimensional plane. Of interest, longer links are the candidates to become multi-cycle, therefore, although both representations of the same topology have the same hop count, they may end up presenting a different diameter. This is a classical example of a theoretical metric that does not hold when the topology is laid-out on silicon because it needs some engineering effort to be effective.

- **Connectivity**: Defined as the minimum number of network links that have to be disconnected in order to prevent a core from sending and/or receiving messages. This property indicates the maximum number of network links failures that a topology can tolerate without isolating any core.

- **Total number of switches**: Defined as the total number of switches required to fully interconnect all the cores with a particular topology. This property is usually related to the number of cores that need to be interconnected.

- **Total number of network links**: It is defined as the number of unidirectional network links required to fully interconnect all cores of a certain topology. As links are not a critical resource in NoCs, this is not an interesting property for NoC designers but the delay of each link is a key factor. In fact, as it will be discussed in section 3.4, a long link may require several pipeline stages to be inserted in order to keep the whole topology frequency above a certain threshold, thus affecting its link delay. Furthermore, since the real link delay is dependent on the topology mapping strategy and the technology library used, it is very difficult to provide an accurate estimation of this parameter in the earlier design stages. On the other hand, this property is still a good implementation cost indicator, as it is directly related to the total number of ports of a topology. The total number of ports of a topology is equal to the total number of unidirectional network links plus the total number of cores. Notice that to perform a comparative analysis of the implementation

cost of topologies with equal number of cores, only the number of unidirectional network (switch-to-switch) links varies between topologies, as the core–to–switch links are a constant in this case. For this reason, from now on this dissertation will refer only to the *total number of networks links*.

The properties listed above allow to perform generic but inaccurate topology comparisons in terms of both, performance and cost, since they are completely agnostic of any physical implementation property of the topologies they describe. While this makes sense for a rough estimate of abstract system level properties, this chapter will demonstrate later on the risks of selecting the topology for a NoC system based only on these considerations.

3.2.2 Topologies for Networks on-Chip

As mentioned, the 2D-mesh is the most common topology for NoCs in the open literature. It is also the common solution for the latest commercial proposals and industrial prototypes, like the Tilera multi-core processor family [6] or the Intel 80-cores Polatis chip [2]. However, this trend is expected to change in the near future due to the 2D-mesh limitations. Alterative topologies are either optimizations or modifications of a 2D-mesh, like the C-Mesh [23], or based on radically different connectivity patterns, like WK-Recursive [126] or Fat-Tree [82]. This section presents some of the most promising regular topologies that have been recently proposed for NoCs in the open literature. Table 3.1 summarizes the properties of all the presented topologies, as well as those of their most common sub-types (if any).

Mesh

Although commonly depicted as a direct topology, a mesh can be implemented as both, direct and indirect. In the former case, every core is directly connected to other cores. In the latter case, every core is connected to the network through a link and a switch. Regardless of their type, meshes are always orthogonal.

A generic mesh is defined by three parameters: the number of dimensions n, the number of cores attached to each switch m, and a n-tuple $\langle k_1, k_2, ...k_n \rangle$,

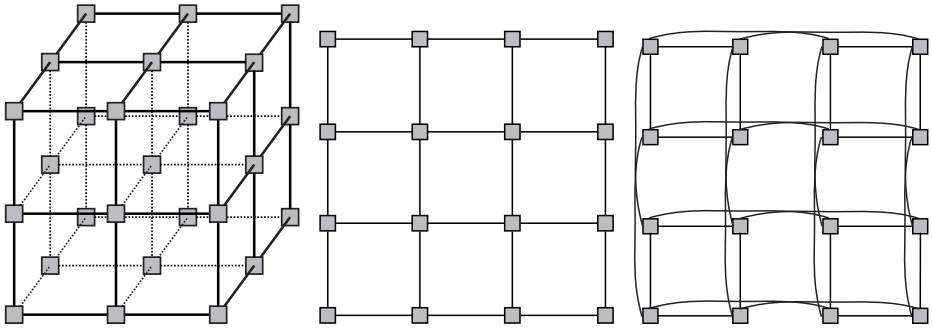
Topology	Switches	Cores/ switch	Total cores	Max. switch degree	Symm.	Homog.
k-ary n-mesh	k^n	m	mk^n	2n+m	No	No
k-ary 2-mesh	k^2	m	mk^2	4+m	No	No
2-ary n-mesh	2^n	m	$m2^n$	n+m	Yes	Yes
k-ary n-cube	k^n	m	mk^n	2n+m	Yes	Yes
k-ary l-rec	k^l	m	mk^l	k+m	No	No
k-ary n-tree	nk^{n-1}	0 or k	k^n	2k	No	No
k-ary n-fly	nk^{n-1}	0 or k	k^n	k	No	Yes
k-cmesh	k^2	m	mk^2	4+m	No	Yes/No
k-ary n-flat	k^{n-1}	k	k^n	(n-1)(k-1)+k	Yes	Yes

Topology	Unidirectional networks links	Bisection bandwidth	Hop count	Connect.
k-ary n-mesh	$2n(k-1)k^{n-1}$	$2k^{n-1}$	n(k-1)+1	n
k-ary 2-mesh	4(k-1)k	2k	2k-1	2
2-ary n-mesh	$n2^n$	2^n	n+1	n
k-ary n-cube	$2nk^n$	$4k^{n-1}$	n(k/2)+1	2n
k-ary l-rec	k^{l+1}-k	$k^2/2$	2^l	k-1
k-ary n-tree	$2k^n(n-1)$	k^n	2n-1	k
k-ary n-fly	$k^n(n-1)$	$k^n/2$	n	1
k-cmesh	$4k^2$	4k	k+1	4
k-ary n-flat	$(n-1)(k-1)k^{n-1}$	$k^n/2$	n	(n-1)(k-1)

Table 3.1: High level properties of analyzed topologies.

in which each k_i defines the number of switches in dimension i. This nomenclature is usually represented as: k_1xk_2x...xk_n. When the value of k of different dimensions differ, the topology become asymmetric, thus worsening the topology hop count and its bisection bandwidth. For this reason it is common to consider a subset of meshes in which the value of k is constant for all the dimensions, known as k-ary n-mesh.

A k-ary n-mesh has k^n switches distributed across n dimensions with k switches in each dimension. Each switch may have up to m cores attached to each switch, thus interconnecting up to mk^n cores. The switch degree increases with the number of dimensions and depends on switch placement inside the topology. The maximum degree is found in switches located at



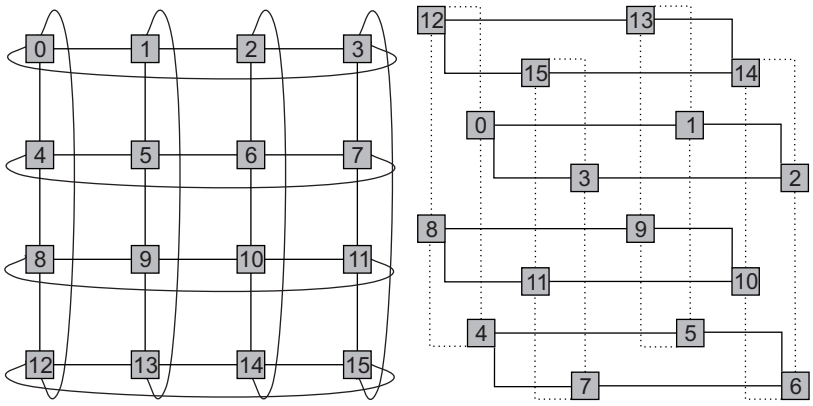
(a) 3-ary 3-mesh (b) 4-ary 2-mesh (c) 2-ary 4-mesh

Figure 3.4: Switch distribution of several mesh configurations.

the middle of the mesh, with $2n + m$ ports. The hop count is $n(k-1)+1$, while the total number of unidirectional network links is: $2n(k-1)k^{n-1}$. Finally, the bisection bandwidth is $2k^{n-1}$ unidirectional links, and the topology connectivity is n links, corresponding with the number of switch–to–switch links of switches located at the corners. As a general rule, a k-ary n-mesh is neither symmetric nor homogeneous, with the exception mentioned below. Figure 3.4 shows several examples of the switch distribution in k-ary n-mesh topologies. Notice that each switch may have several cores attached (although they are not shown).

There are two subsets of meshes that require special attention: *hypercubes* and *2D-meshes*. A 2D-mesh (see Figure 3.4(b)) is a mesh with two dimensions, so it can be represented as k-ary 2-mesh. On the other hand, a hypercube (see Figure 3.4(c)) is a 2-ary n-mesh, that is a mesh in which the number of switches in each dimension is always 2, thus they may also be defined as n-hypercubes. Hypercubes are the only subset of k-ary n-mesh that is both, symmetric and homogeneous, with a switch degree of $n + m$, regardless of switch placement.

The 2D-mesh is the most common topology for NoCs, but meshes with more than two dimensions are less common in the open literature. While a 2D-mesh perfectly matches the layout of the cores, other meshes present a more complex layout, introducing long links and high degree switches. When mapping a mesh with more than two dimensions (n greater than 2) the links



(a) 4-ary 2-cube (b) 4-ary 2-cube folded

Figure 3.5: Switch distribution of several torus configurations.

used to connect the dimensions greater than two are longer. For instance, assuming uniform core size, links of dimensions 2 and 3 will have twice the length of the links of dimensions 1 and 2, the ones of dimensions 4 and 5 four times that length, and so on. As a general rule, the length of a link of the dimension t is $k^{(d-2)/2}$ times the length of a link of dimension 1, where d is equal to t if t is an even number and d is equal to $t+1$ if it is an odd number. But that length may change based on the placement constraints and the cores geometry. Anyway, the work in [81] introduces the flattened butterfly topology, which in some cases is equivalent to a hypercube with several cores attached to each switch, as we will present in this section. Moreover, 3-D technologies are becoming a real possibility for multi-core designs [152], favoring the use of topologies with more complex wiring patterns. For example, a 3-D mesh (see Figure 3.4(a)) may have the optimal wiring pattern for a 3-D environment.

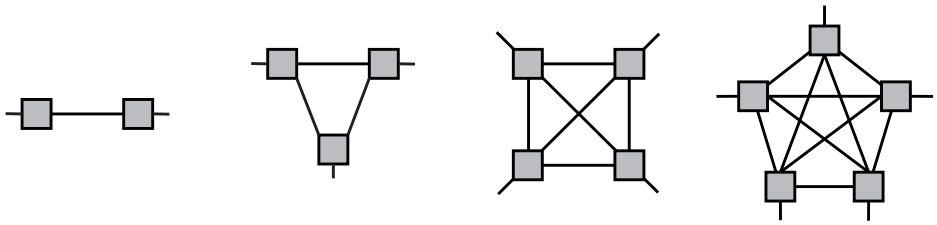
Torus

A generic torus is constructed by adding wrap-around links to the equivalent mesh and thus they can be defined by similar parameters, with the main difference that torus can be either unidirectional or bidirectional topologies, according with the type of links it implements. Tori are orthogonal topologies that are usually implemented as a direct topology. But, like in meshes, it is

possible to get the switch outside the core, converting a torus into an indirect topology. For example, Figure 3.5(a) shows the switch distribution of the torus resulting of adding wrap-around links to the 2D-mesh in Figure 3.4(b). A torus can be defined by the number of dimensions (n), the number of cores attached to each switch (m), and the n-tuple that defines the number of switches per dimension ($\langle k_1, k_2, ...k_n \rangle$). But as in the case of the mesh, a torus with varying number of switches per dimension may not be the best choice.

For this reason, it is usual to consider k-ary n -cubes, a subset of torus in which the number of switches per dimension (k) is constant. A k-ary n-cube has k^n switches distributed across n dimensions with k switches in each dimension, with up to m cores attached to each switch, thus interconnecting up to mk^n cores. The switch degree increases with the number of dimensions and is constant for all the switches: $2n+m$ ports. The hop count is $n(k/2)+1$, while the total number of unidirectional network links is: $2nk^n$. The bisection bandwidth is $4k^{n-1}$ unidirectional links, and the topology connectivity is $2n$ links. Finally, a k-ary n-cube is symmetric and homogeneous. Therefore, it provides better performance and fault tolerance, but at the cost of a slightly higher number of links. Also, the wrap-around links of a torus may present length issues. A common way to solve this problem is to use a folded torus, in which the nodes are re-allocated over the floorplan, obtaining an equivalent topology with reduced wrap-around link length at the cost of increasing the length of the other links. Figure 3.5(b) shows the folded version of the 4-ary 2-cube shown in Figure 3.5(a).

Although not as common as 2D-meshes, tori have been proposed in several NoC works. The Proteo NoC [132] employs a ring topology, and rings can be defined as a k-ary 1-cubes. Also, 2D-folded tori have been proposed for their use in NoC [49,86,101,130]. Although rings and 2D-torus have been proposed as NoC topologies, more complex structures than them are very difficult to find even in the open literature due to their complexity and expected layout intricacy. Additionally, 2-ary n-cube and hypercubes (2-ary n-mesh) are considered as equivalent topologies, as wrap-around links are redundant in this case.



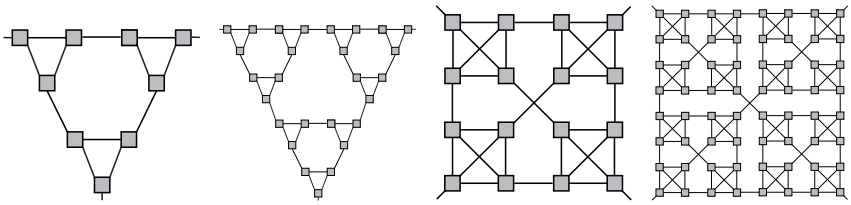
(a) Degree 2 (b) Degree 3 (c) Degree 4 (d) Degree 5

Figure 3.6: Switch distribution in WK-Recursive first level virtual nodes.

WK-Recursive

WK-Recursive topologies were first proposed in [51], and are regular direct non-orthogonal topologies, constructed by recursively replicating a basic structure called *virtual node*. The first virtual node is constructed by connecting k switches with k network ports in a fully connected structure, leaving k links free. Like in meshes and tori, it is possible to take the switch out of the node, thus converting the WK-Recursive topology into an indirect topology. In this case, m cores could be attached to each switch. Hence, a virtual node is virtually similar to a switch of degree k, with mk cores attached to it. Figure 3.6 shows the switch interconnection of several first level virtual nodes. Following the same strategy, k virtual nodes can be used to define a second level virtual node of degree k. In fact, a level l virtual node is constructed by using k virtual nodes of level $(l-1)$. Based on this definition, a WK-Recursive topology is defined by two parameters: the virtual node k and the expansion level l. The most common nomenclature of this topology in the open literature is to use the pair (k,l), but, in order to provide an homogeneous nomenclature among all the topologies described in this chapter, for now on, this topology will be addressed as k-ary l-rec. Figure 3.7 shows several WK-Recursive topologies for different values of k and l.

A k-ary l-rec has k^l switches, with m cores attached to each switch, thus interconnecting mk^l cores. The switch degree is directly defined by virtual node degree and depends on the switch placement inside the topology (for example, switches at the corners of Figure 3.7(c) employ one less port than other switches), being the maximum switch degree $k+m$ ports. The hop count is 2^l,



(a) 3-ary 2-rec (b) 3-ary 3-rec (c) 4-ary 2-rec (d) 4-ary 3-rec

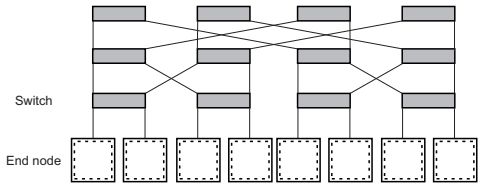
Figure 3.7: Switch distribution of several WK-Recursive configurations.

while the total number of unidirectional network links is: $k^{l+1} - k$. The bisection bandwidth is $k^2/2$ unidirectional links, and the topology connectivity is $k - 1$ links. Finally, a k-ary l-rec is neither symmetric nor homogeneous. Notice that in the case of k-ary 1-rec configurations, the maximum switch degree is $k - 1 + m$ ports and the topology becomes symmetric and homogeneous.

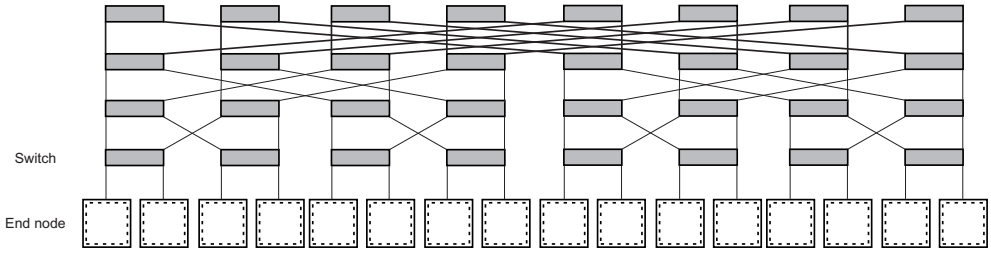
Although it is difficult to provide efficient deadlock-free routing in a k-ary l-rec [126], this topology has attracted the attention of NoC researches [126,142], due to some appealing properties. First, 4-ary l-rec has similar layout than 2D-mesh with the same number of cores and cores per switch. Second, the switch degree is exactly the same in both topologies. Third, the number of switches is exactly the same in both topologies, although the number of links is slightly higher in the WK-Recursive topology. Finally, a WK-Recursive topology has a much lower hop count than its equivalent 2D-mesh. Anyway, the improvements of 4-ary l-rec topologies over 2D-meshes comes at a cost, as its bisection bandwidth scales worse than in a 2D-mesh.

Fat-Tree

Fat-Trees are regular indirect topologies based on complete trees. The difference between a fat-tree and a complete tree is that a fat-tree gets thicker (offers more bandwidth) near the root, thus making switch degree higher the closer the switch is placed to the root. If the switch degree becomes too high, the physical implementation may become infeasible due to an unrealistically low operating frequency and/or cost. For this reason, some alternative implementations have been proposed in order to use switches of fixed degree.



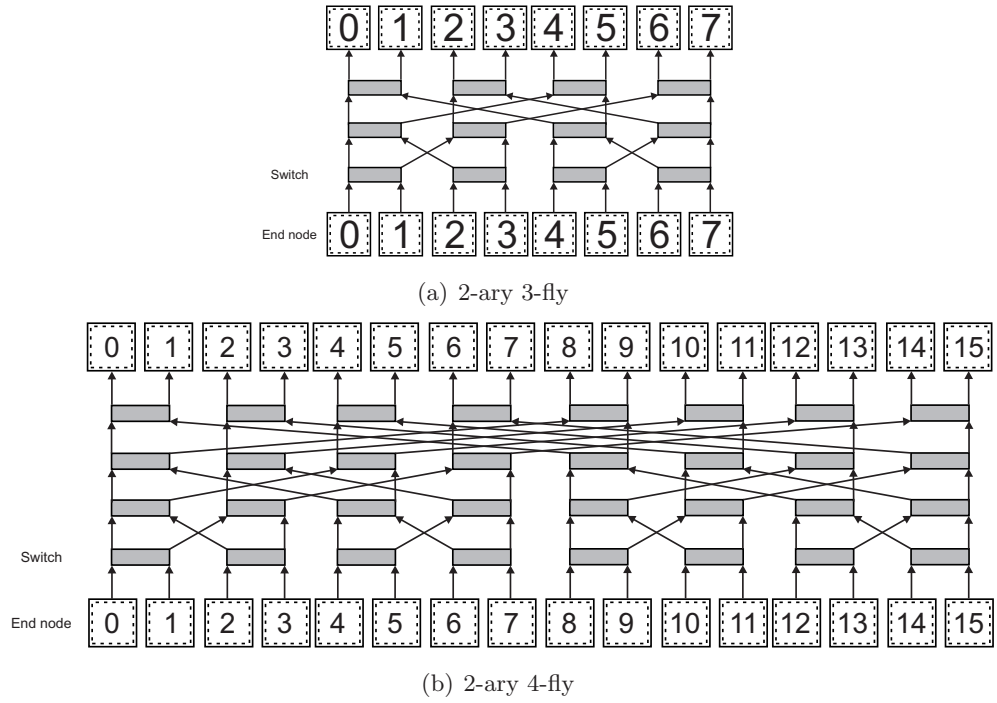
(a) 2-ary 3-tree



(b) 2-ary 4-tree

Figure 3.8: Examples of fat-tree configurations.

Among those alternatives, one of the most common subclasses are butterfly fat-tree or folded butterfly, also known as k-ary n-trees [119]. In this notation, a fat-tree is defined by two parameters: the number of stages n and the switch degree k. A k-ary n-tree belongs to the Multistage Interconnection Network family (MIN), and thus switches are organized in stages, in which a switch only has connections to the next and the previous stage based on a regular pattern, and only switches belonging to the first stage have connections to cores. In particular, a k-ary n-tree is composed composed by nk^{n-1} switches distributed in n stages. Each switch of the first stage has k cores attached, thus the topology provides connection to k^n cores. Each switch employs k bidirectional links to connect with the next stage and k bidirectional links to connect to the previous stage, with the exception of switches belonging to the first and last stages. Switches of the last stage have a switch degree of k, while all the other switches have a degree of $2k$. The hop count is $2n - 1$, while the total number of unidirectional network links is: $2k^n(n - 1)$. The bisection bandwidth is k^n unidirectional links, and the topology connectivity is k links. Finally, a k-ary n-tree is neither symmetric nor homogeneous. Figure 3.8 shows several configurations of k-ary n-trees. Notice that unlike



(a) 2-ary 3-fly

(b) 2-ary 4-fly

Figure 3.9: Examples of butterfly configurations.

the previous topologies, in this case the number of cores connected by the topology is strictly fixed.

Due to their low hop count and high bisection bandwidth, fat-trees have been proposed as topologies for NoC aiming at low latency communication, like the works presented in [17, 114]. Anyway, k-ary n-trees require a much higher number of switches and links that an equivalent mesh with the same number of cores.

Unidirectional topologies

Most topologies proposed for NoC in the open literature employ bidirectional links. But, in resource constrained environments like NoCs, unidirectional topologies may be a better cost-performance trade-off in the design space. An unidirectional topology require unidirectional (one-way) ports and links, so the number of links of the topology can be smaller than in a 2D-mesh

with equal number of cores. This translates in important area and power consumption savings, but there are several issues that question the viability of these topologies in NoCs. For example, simple unidirectional topologies only offer a unique path between any two cores, making them highly vulnerable to link failure, as there is no way to avoid those faulty links. Although it is possible to use more complex unidirectional topologies to overcome this limitation, this is likely to increase the cost of the topology, thus offsetting the low cost advantage of unidirectional topologies.

There are two kind of unidirectional topologies that are well known in the open literature: rings and multistage. Unidirectional rings are just a sub-case of the torus family (see Section 3.2.2). Regarding multistage topologies, unidirectional multistage interconnection networks (uMINs) have some properties that make them appealing as NoC topologies. For example, the work in [22] proposes the use of the butterfly topology as mean to provide high performance communications with an affordable implementation cost.

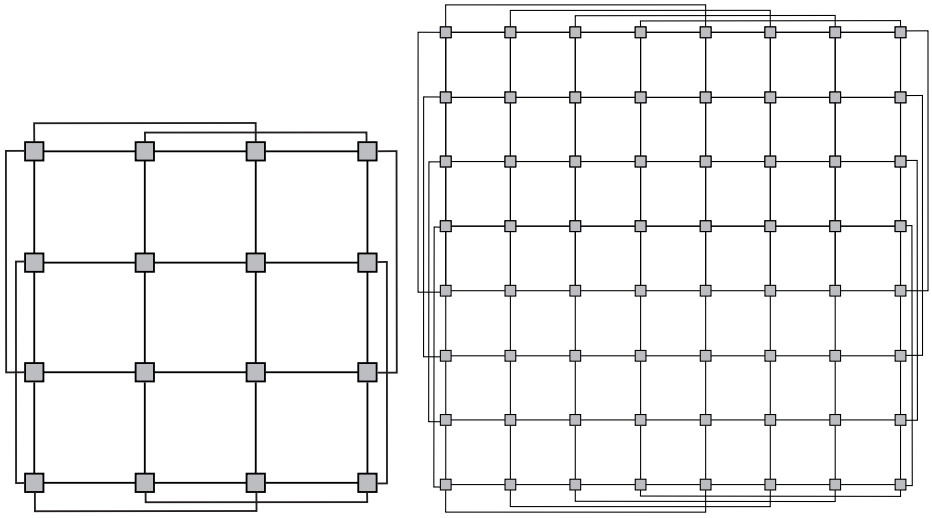
In an uMIN, switches are organized in stages, and each stage only connects to the previous and next stage following a regular pattern, with the exception of the first and the last stage, that have connections to the cores. In those topologies, communications are one-way: messages are always delivered from the first stage to the last stage. An uMIN is characterized by the pattern defining how stages interconnect with each other. There are several common patterns, and one of the most common patterns is the butterfly, which is also used in the building of the k-ary n-tree topology. The butterfly topology, also known as k-ary n-fly, is a regular indirect topology, composed by nk^{n-1} switches distributed in n stages. The topology provides connection to k^n cores. Switch degree is constant and equal to k, while the hop count is n, and the total number of unidirectional network links is: $k^n(n-1)$. The bisection bandwidth is $k^n/2$ unidirectional links. Due to the existence of a single path between any pair of cores, the topology does not tolerate any link failure, thus its connectivity is 1 unidirectional link. Finally, a k-ary n-fly is homogeneous, but it is not symmetric.

Figure 3.9 shows several configurations of k-ary n-fly. As can be observed, at the first stage, each switch has k cores attached only to send messages through the topology, while at the last stage, each switch has k cores attached

only to receive messages from the topology.

Concentrated Mesh

The Concentrated Mesh (or C-Mesh) topology was first proposed in [23]. This topology is an evolution of a 2D-mesh, and aims at solving 2D-mesh scalability issues while keeping its advantages.



(a) 4-ary 2-cmesh (b) 8-ary 2-cmesh

Figure 3.10: Switch distribution of several C-Mesh configurations.

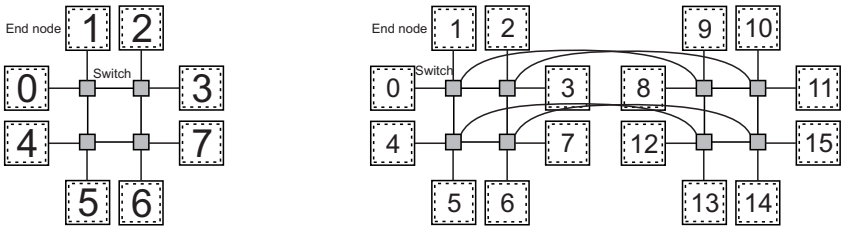
In [23] a C-Mesh is built around a 4×4 2D-mesh with four cores attached to each switch. This way, a lower hop count with respect to that of a classical 2D-mesh, with one core per switch, can be achieved. However, this comes at the cost of a diminished bisection bandwidth. For this reason, the adoption of express links is used to improve C-Mesh bisection bandwidth. Express links are added only along the perimeter of the network, thus taking advantage of the switches that have a lower degree than the maximum one. In this way, switch degree remains constant across all the switches of the topology, and equal to the 2D-mesh maximum one. Although the main goal of using express links is to increase the bisection bandwidth, they may be also used to greatly reduce the hop count as they effectively represent alternative routes. Unfortunately,

this opportunity comes with a cost, as deadlock may more easily become an issue. Therefore, the provision of deadlock-free routing algorithms is a more complex task with respect to the original mesh. For this reason, in the original proposal, express links are used only to advance messages that travel along the perimeter of the topology a distance equal or higher than the switches that this express link overtakes. Albeit the use of the express links considered in [23] is limited, those express links can be used in a similar way to the wrap-around links of a torus, regardless of whether the message is traveling through the perimeter of not. From a pure topology point of view, a C-Mesh can be seen as a classical 2D-mesh with express links, regardless of the number of cores attached to each switch. Figure 3.10 shows several examples of switch distribution of generalized C-Meshes, in which each switch can connect to a parameterizable number cores.

A C-Mesh is defined by the same parameters that a conventional 2D-mesh, so it can be defined as a k-cmesh, with k^2 switches distributed across 2 dimensions and k switches in each dimension, with m cores attached to each switch, interconnecting a total of mk^2 cores. Express links connect switches from the same dimension that are located at the perimeter of the network and are $k/2$ hops distant from each other, thus configurations with a number of switches per dimensions (k) of less than three, are infeasible in this topology. The maximum switch degree is equal to $4+m$ ports, and is constant only when k is an even number. The hop count is $k+1$, while the total number of unidirectional network links is: $4k^2$. The bisection bandwidth is $4k$ unidirectional links, and the topology connectivity is 4 links. Finally, a k-cmesh is not symmetric, and it is is homogeneous only for even values of k.

Flattened Butterfly

The flattened butterfly topology was first proposed as a NoC topology in [81], as a solution for cost-efficient high-degree networks. This topology is an evolution of the butterfly topology, and it is formed by flattening the switches in each column of a butterfly while keeping the same inter-switch connections. The resulting topology is a regular orthogonal topology that provides the same bisection bandwidth that a butterfly and a lower hop count, at the cost of a greatly increased switch degree. Notice than the switch degree greatly



(a) 2-ary 3-flat (b) 2-ary 4-flat

Figure 3.11: Examples of flattened butterfly configurations.

increases with both original butterfly parameters: switch degree (k) and number of stages (n), thus the real performance achievable is extremely sensitive to the network architecture employed in the final implementation.

A flattened butterfly may be defined as a k-ary n-flat, with k cores attached to each switch. In this way, a flattened butterfly can be designed as a k-ary n-flat, with k^{n-1} switches distributed across $n-1$ dimensions with k switches in each dimension, interconnecting a total of k^n cores. The switch degree is constant and increases with the number of dimensions, being equal to $(n-1)(k-1)+k$ ports. The hop count is n, while the total number of unidirectional network links is: $(n-1)(k-1)k^{n-1}$. The bisection bandwidth is $k^n/2$ unidirectional links, and the topology connectivity is $(n-1)(k-1)$ links. Finally, a k-ary n-flat is both symmetric and homogeneous. Figure 3.11 shows the flattened butterfly configurations obtained from flattening the butterfly configurations presented in Figure 3.9.

Notice that this topology may overlap with previous topologies in several cases. First, as can be observed in Figure 3.11, when flattening a 2-ary n-fly, the resulting topology will be identical to a 2-ary $(n-1)$-mesh with two cores attached to each switch. Second, when flattening a butterfly of two stages, the resulting topology is equivalent to the basic virtual node of the WK-Recursive of degree k with k cores attached to each switch. In particular, a k-ary 2-flat will be equivalent to a k-ary 1-rec with k cores attached to each switch, for any value of k. Finally, a k-ary n-flat is completely equivalent to a generalized hypercube [34] of degree k and $n-1$ dimensions, with k cores attached to each switch.

Topology	Switches	Cores/ switch	Max. degree	Unidir. links	Bisection bandwidth	Hop count	Connect.
4-ary 2-mesh	16	1	5	48	8	7	2
2-ary 4-mesh	16	1	5	64	16	5	4
2-ary 3-mesh	8	2	5	24	8	4	3
2-ary 2-mesh	4	4	6	4	4	3	2
4-ary 2-cube	16	1	5	64	16	5	4
4-ary 2-rec	16	1	5	60	8	4	3
8-ary 1-rec	8	2	9	56	32	2	7
2-ary 4-tree	32	0 or 2	4	96	16	7	2
4-ary 2-tree	8	0 or 4	8	32	16	3	4
2-ary 4-fly	32	0 or 2	2	48	8	4	1
4-ary 2-fly	8	0 or 4	4	16	8	2	1
4-ary 2-flat	4	4	7	12	8	2	3
4-cmesh	16	1	5	64	16	5	4

Table 3.2: High level parameters of solutions for 16 cores.

3.2.3 High level topology exploration

It is possible to utilize the above exposed properties to perform a high-level comparison of topology capabilities. This analysis, may be employed in the early stages of a system design in order to select the subset of the most promising topology candidates. This section presents an example of high-level topology exploration for two system with different sizes: 16 and 64 cores. The number of cores attached to each switch in those topologies has been limited to four not to limit too much the maximum operating speed of the switches. Also, the placement of cores around the switches in those cases would not be a trivial task, since as the length of the injection/ejection links grows, the NoC would suffer from a significant drop in performance.

Table 3.2 summarizes the values of the parameters of all 16 cores configurations considered for each topology. For such small systems, there are only a small number of different configurations to consider. As can be seen, apart from the classical 2D-mesh, several hypercubes (that is 2-ary n-mesh topologies) with different concentration degrees of cores at each switch are considered. On the other hand, only one torus configurations is considered.

Although it is possible to concentrate cores in a torus, for this particular system size, the resulting torus topologies will have a k value of 2, resulting in the same hypercubes already considered in the table. Also, two WK-Recursive configurations with different concentration degree of cores are considered. In the case of the C-Mesh topologies, there is only a single valid configuration, as configurations with a k value lower than three are not valid in this topology. Regarding MINs, two kinds of fat-trees are considered, with switch degrees (k) of four and eight. Also, the equivalent unidirectional butterflies of the analyzed fat-trees are considered, together with their flattened versions, with the exception of the 2-ary 4-flat configuration, as it is completely equivalent to the 2-ary 3-mesh with two cores attached to each switch. Finally, when flattening the 4-ary 2-fly solution, the resulting topology overlaps with a 4-ary 1-rec solutions with four cores attached to each switch.

As can be observed, the 8-ary 1-rec solution shows the highest bisection bandwidth, making this configuration the best proposal for high load configurations. It also provides the minimum possible number of hops, while requiring the highest switch degree of all the considered solutions. This fact questions the achievable maximum operating frequency of its physical implementation. On the other hand, from a low-latency (low number of hops) view-point there is not a clear winner. There are three solutions that can reach any destination in only two hops: 4-ary 2-fly, 8-ary 1-rec and 4-ary 2-flat. The 4-ary 2-fly solution seems the best choice: it requires a similar number of switches of lower degree than the 8-ary 1-rec solution, with a lower number of links as well. Unfortunately, it has an important drawback associated with its connectivity. In fact, with a connectivity of only a single unidirectional link, this solution is highly sensitive to faulty links. Regarding the 4-ary 2-flat, although it offers a smaller bisection bandwidth than the other two solutions, it has the lowest number of switches and links with respect to the other two low-latency solutions. However, there is an issue that further complicates the comparison in this case: the switch degree. While the 8-ary 1-rec and the 4-ary 2-flat solutions require high-degree switches, the 4-ary 2-fly solution has a maximum degree of four. Finally, the 2-ary 4-mesh, 4-ary 2-cube, 4-ary 2-fly, and 4-cmesh solutions appear to be equivalent topologies providing the second highest bisection bandwidth with an acceptable number of hops. Among

them, the only topology that shows a clear inconvenient is the 4-ary 2-tree, that has a switch degree almost as big as the 8-ary 1-rec.

In overall, the high-level analysis for 16 cores systems shows that the best topology is the 8-ary 1-rec solution. This topology has a low number of switches and links, while providing a high connectivity, the highest bisection bandwidth and the lowest number of hops. As anticipated, the only parameter that may arise some concerns is its switch degree, which maybe is too high for such a small system. In the case that the 8-ary 1-rec topology proves inefficient for physical implementation, the best solution might be one of the following solutions: 2-ary 4-mesh, 4-ary 2-cube, or 4-cmesh. These latter solutions provide the second best bisection bandwidth and an acceptable number of hops.

Regarding the 64 cores analysis, Table 3.3 summarizes the properties of all 64 cores configurations considered for each family of topologies. In this case, due to the increased system size, there are more topology configurations to be considered, even when excluding overlapped topologies. Notice that there are two configurations that are excluded from the analysis due to an extremely high switch degree. Those configurations are the 8-ary 2-flat, 8-ary 2-tree, and the 16-ary 1-rec with 4 cores concentrated in each switch, thus resulting in a switch degree of 15, 16 and 19 respectively. Unless full custom solutions are conceived, the implementation of such high-degree switches will present unaffordable low operating frequencies.

For this system size, the solutions best suited for high traffic loads are the 2-ary 6-mesh, 4-ary 3-cube, 2-ary 6-tree, 4-ary 3-tree. Among those, the 2-ary 6-tree solution has the highest hop count (11 hops), making it the worst candidate for latency sensitive systems and/or applications. Also, it requires the highest amount of resources: 192 switches of degree 4 and 640 unidirectional links. Regarding the 4-ary 3-tree, it has an acceptable number of hops (5 hops), while requiring a moderate amount of resources: 48 switches of degree 8 and 256 unidirectional links. On the other hand, the 2-ary 6-mesh and 4-ary 3-cube solutions are similar from a high-level viewpoint, as both of them require 64 switches of degree 7 and 384 unidirectional links, that is more switches and links than the 4-ary 3-tree solution, with only one port less in each switch. They also have a higher number of hops than the 4-ary

Topology	Switches	Cores/ switch	Max. degree	Unidir. links	Bisection bandwidth	Hop count	Connect.
8-ary 2-mesh	64	1	5	224	16	15	2
4-ary 3-mesh	64	1	7	288	32	10	3
4-ary 2-mesh	16	4	8	48	8	7	2
2-ary 6-mesh	64	1	7	384	64	7	6
2-ary 5-mesh	32	2	7	160	32	6	5
2-ary 4-mesh	16	4	8	64	16	5	4
8-ary 2-cube	64	1	5	256	32	9	4
4-ary 3-cube	64	1	7	384	64	7	6
4-ary 2-cube	16	4	8	64	16	5	4
4-ary 3-rec	64	1	5	252	8	8	3
4-ary 2-rec	16	4	8	60	8	4	3
8-ary 2-rec	64	1	9	504	32	4	7
2-ary 6-tree	192	0 or 2	4	640	64	11	2
4-ary 3-tree	48	0 or 4	8	256	64	5	4
2-ary 6-fly	192	0 or 2	2	320	32	6	1
4-ary 3-fly	48	0 or 4	4	128	32	3	1
8-ary 2-fly	16	0 or 8	8	64	32	2	1
4-ary 3-flat	16	4	10	96	32	3	6
8-cmesh	64	1	5	256	32	9	4
4-cmesh	16	4	8	64	16	5	4

Table 3.3: High level parameters of solutions for 64 cores.

3-tree solution: 7 hops against 5. So, the best configuration for high traffic loads in a 64 cores system is the 4-ary 3-tree, as it has the lower hardware resources requirement and number of hops among the best solutions for high communications loads.

On the contrary, from a low-latency view-point, the best solution is the 8-ary 2-fly, which needs only two hops to reach any core. This solution employs 16 switches of degree 8 and only 64 links, providing a bisection bandwidth of 32 unidirectional links. Its only drawback is its low connectivity, common to all butterfly topology.

Overall, for bandwidth-intensive systems the 4-ary 3-tree solution seems to be the best option, while for latency-sensitive systems the 8-ary 2-fly topology seems the most promising one. However, the 8-ary 2-fly solution offers the

second best bisection bandwidth, becoming the overall best solution unless a higher bisection bandwidth is required. Unfortunately, it is not suited to systems that require some degree of fault tolerance and/or a variety of paths between any pair of cores. In this case, the 4-ary 3-flat solution provides the same bisection bandwidth than the 8-ary 2-fly, while increasing the number of hops by one. The specific requirements of the system designer and the features of the network architecture can push one solution instead of another, as there is in general no clearly winning topology but a trade-off between many factors.

3.3 Networks on-Chip modelling

High level topology properties provide fast early topology explorations. However, explorations based in high-level methodologies are oblivious to the physical synthesis of the different topologies, so their accuracy is questionable. This accuracy problem is aggravated by the effects of nanoscale technologies over NoC physical design. Issues such as interconnect delay and maximum achievable frequency cannot be ignored by NoC synthesis flows. Moreover, a single technology library no longer exists for standard cell design. In fact, manufacturing technologies are spreading across a variety of libraries optimized for specific uses, such as low power or high performance, with several intermediate levels featuring for example different threshold voltages. Using different libraries at the same design generates large differences in synthesis results, and their spread is increasing as technology scales down [121]. Therefore, it is of utmost importance to reduce the gap between both design layers. This objective can be achieved in two ways. On one hand, layout information can be annotated into abstract exploration tools in order to increase the accuracy of their predictions. On the other hand, high-level exploration tools can provide feedback to the backend flow in order to guide the synthesis process to the most suitable candidates in the implementation space [108]. The way these issues are addressed in the early design phases not only results in more or less efficient NoC designs, but can even limit their practical feasibility. As a general guideline, driving NoC designs under severe technology constraints consists of making silicon-aware decisions at each hierarchical level of the design flow [54]. This is likely going to result in less design re-spins and in faster

timing closure.

This chapter capitalizes on the concept of Transaction Level Modelling (TLM) to come up with a fast and accurate simulation environment that abstracts all relevant NoC architecture-level mechanisms while maintaining execution time accuracy of Register Transfer Level (RTL) simulations. This simulation environment can be used to explore the implementation space either by performing implementation space parametrical explorations or by back-annotating physical design information.

In *parametrical explorations* the ranges of network operating frequencies and link latencies in which performance of one topology is consistently better than another one are identified. The results of the implementation space exploration are then changed into directives for the backend synthesis flow. The degrees of freedom in the synthesis flow (e.g., mix of technology libraries, floorplanning decisions, pipelined link insertion) should then be exploited to meet those constraints. In this way, the physical constraints that, if met, would allow not to waste the better theoretical properties of a particular topology as a result of physical implementation are derived. In the case of *back-annotation*, the physical characteristics of a particular topology are first obtained (e.g., by means of a physical synthesis or technology characterization or a predictive model). This physical design information is then distributed back to the simulation framework, that will use this information in such a way that performance predictions reflect the impact of the topology physical characteristics.

3.3.1 Networks on-Chip behavior abstraction

Our simulation framework takes its steps from the synthesizable RTL model of the reference architecture exposed in Section 2.1.9. The starting point of the simulation framework was first presented in [89]. This work presents a simulator for interconnection networks developed in MODULA-2. However, that simulator just models a concept architecture, does not carry any information about physical implementation and models network interfaces only approximately. Deep modifications of the native simulator were carried out in order to abstract the behavior of the reference architecture as well as to add the capability of back-annotate physical implementation information.

The main difference between the RTL and TLM models is that while the

former (developed in SystemC [8]) is cycle and network signal accurate, the latter is only cycle accurate. The simulation framework includes abstract models of each NoC component that is relevant to the topology performance: switches, links (including repeater stages) and network interfaces. A great effort was devoted to modeling the Network Interfaces (NIs), as the native simulator did not model design challenges specific to on-chip environments that define the flit injection rate into the network (e.g, clock domain crossing mechanism and OCP protocol conversion). The RTL structure of the components is abstracted as a set of counters and logical functions: while buffers are represented as counters, component behavior is defined as a set of logical functions. The simulator framework is event driven, and events are scheduled only when there is a change in the network status.

3.3.2 Network traffic generation

In a relatively immature field such as NoCs, realistically capturing traffic behavior is a challenging task. In literature, three major traffic generation approaches exist. First, synthetic traffic patterns are used to send generic messages to predefined or random destinations at a given rate. Second, benchmarks and traces can be used, but hardware architectures and application constraints must match the target design. Third, mathematical models could be used, however it is difficult to ensure that they are representative for given application constraints.

Our approach to address this issue is two folded. On one hand, OCP transactions can be generated based on traffic traces specified in an external file. This mechanism can be used when the traffic pattern is known at design time or for validating TL simulator behavior with respect to the RTL-equivalent SystemC simulator, as the same input traffic specification can be reused for both RTL and TL simulation. On the other hand, the TL simulator implements synthetic traffic generators, allowing designers to evaluate NoC performance when the traffic pattern is not available at design time.

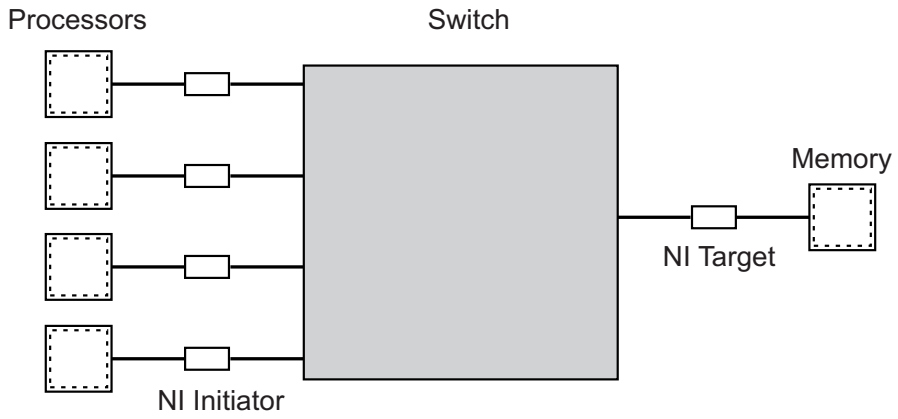
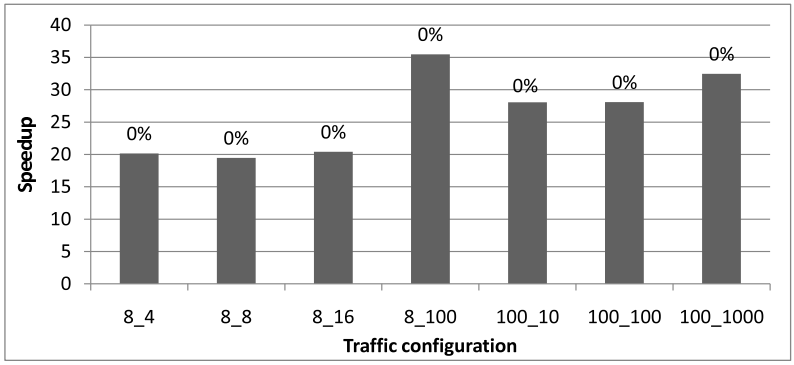


Figure 3.12: Experimental setup

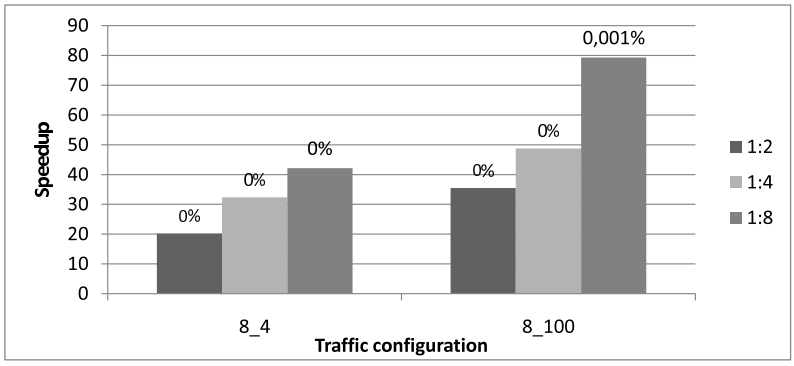
3.3.3 Validation of Transaction-Level simulator accuracy

The accuracy of the abstract models is proved through experimental evidence. Individual OCP read and write transactions need to be validated in both simulation environments. Main parameters include the OCP burst length and the inter-burst idle time, in addition to OCP parameters for each transaction. Several test runs were performed and the cycles that the simulated design would require to process the injected traffic pattern estimated by the TL simulator were compared with the results of the RTL simulator.

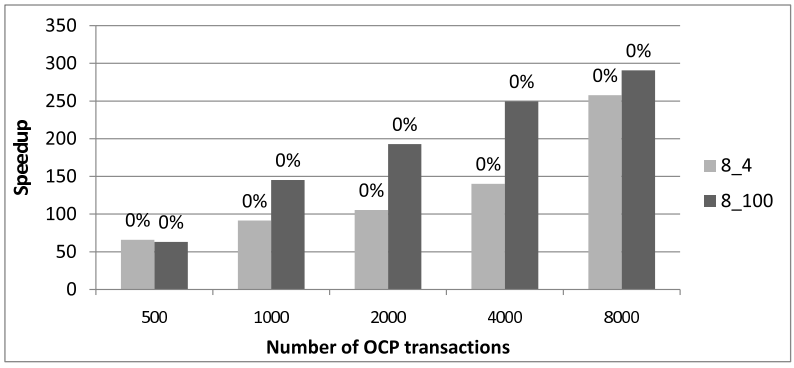
The first test runs aim at capturing the accuracy with which the *injection and ejection interfaces* were modeled, as well as the *flow control mechanism* inside the switches. This experimental setup is shown in Figure 3.12. As depicted, it is composed of a single switch to which 4 processor cores are attached via their network interfaces. Packets from the cores are directed to a single shared OCP memory connected to one output port of the switch via its network interface. The operating frequency of the cores was assumed to reflect the trend for the speed of industrial ARM cores. 470 MHz is a worst case operating frequency for an ARM926EJ-S in 90 nm [9], while CPUs in the ARM11 MPCore can be operated up to 620 MHz in the worst case [10]. In this context, 500 MHz seems to be a reasonable assumption for core speed in current designs, while the network is assumed to work at 1GHz. Figure 3.13(a) show the results obtained for the first series of tests when



(a) First series - Network interfaces & Flow control



(b) Second series - Clock crossing mechanism



(c) Third series - Simulation length & Inter-switch communication

Figure 3.13: Experimental setup results

100 random OCP transactions (both read and write) are generated in each processor. In the figure, the horizontal axis shows the configuration of the OCP traffic generators, which generate bursts of given length L (number of beats) with inter-burst idle wait M (represented as L_M pairs in Figure 3.13). The vertical axis shows the times that TL simulator is faster than the RTL one. Finally, the deviation of the TL estimation over the one of the RTL simulator for each configuration of the OCP traffic generators is shown as a number over each bar. As can be observed, the TL simulator presents an excellent accuracy, while the simulation speedup varied from 20x to 35x. The justification for speedup numbers is in the way that the abstraction was done. The TL simulator only simulates those cycles in which some event is scheduled and only pays attention to the signals and components that are affected by that event, while the RTL simulator re-evaluates every signal at every cycle. This implies that the speedup factor heavily depends on the number of idle cycles in the simulation.

The second series of experiments assesses the accuracy with which the *clock domain crossing* mechanism is abstracted. The experimental setup is the same as in the first case, with the exception of the core operating frequencies, that changes between tests in order to test different clock ratios between core and network. In particular clock ratios of 1:2, 1:4 and 1:8 where tested, that is, core frequency is half of the network one, four times less or eight times less, respectively. In this case, OCP traffic generators were configured with two traffic patterns: 8_4 and 8_100. Both traffic patterns test different behaviours, the 8_100 pattern presents a low-congestion environment, in which all processors generate a burst of length 8 and then wait 100 cycles before generating the next transaction, thus providing the memory with some time to absorb all the remaining transactions before generating new ones. On the contrary, the 8_4 traffic configuration presents a high-contention environment, in which each processors generates transactions after just 4 cycles of idle time. Results are reported in Figure 3.13(b). The legend explicits ratio between OCP and network clock frequencies. As can be observed, the TL simulator accuracy remains excellent. Regarding speedup, it increases with the clock ratio. As slowing down cores implies an increment of the network idle time, those results confirm the above exposed trend about the speedup behaviour.

Traffic configuration	8_100	8_4	8_4
Total Bursts	1764	1764	4900
Speedup	100x	85x	114x
Deviation	0.01%	0%	0%

Table 3.4: Simulation accuracy for a 4-ary 2-mesh.

The third series aims at testing the across-switch communication model, as well as stressing the accuracy of the TL simulator by increasing the number of transactions of the simulation. This involves accurate abstraction of the *switching* and of the *flow control mechanisms*, as well as of buffer control and congestion resolution mechanisms. For this purpose, the number of switches of the previous experimental setting was increased to 5, in such a way that the switches form a line in which all processors are attached to the first switch and the memory to the last one. Figure 3.13(c) shows the results obtained for this test. As in the previous case, the figure shows the speedup of the TL simulator over the RTL one, as well as the deviation in the performance estimations, but in this case the horizontal axis shows the number of transactions simulated on each test. The results show that regardless of the congestion degree of the test and the number of transactions, the TL simulator performance estimations remain highly accurate.

Finally, a more complex configuration was tested. This topology is a 4-ary 2-mesh, with one processor attached to each switch. Only one switch located in the middle has attached a shared memory instead of a processor core. This tests simulates a heavy contention traffic scenario, where each processor performs write accesses to the centralized shared memory. As it can be observed from Table 3.4, the results show that the model is still very accurate.

The following section present a case study as an example of the potential of the developed simulation framework when performing a *parametrical exploration* of mesh topologies. Several examples of *back-annotation* of physical parameters will be presented after section 3.4.

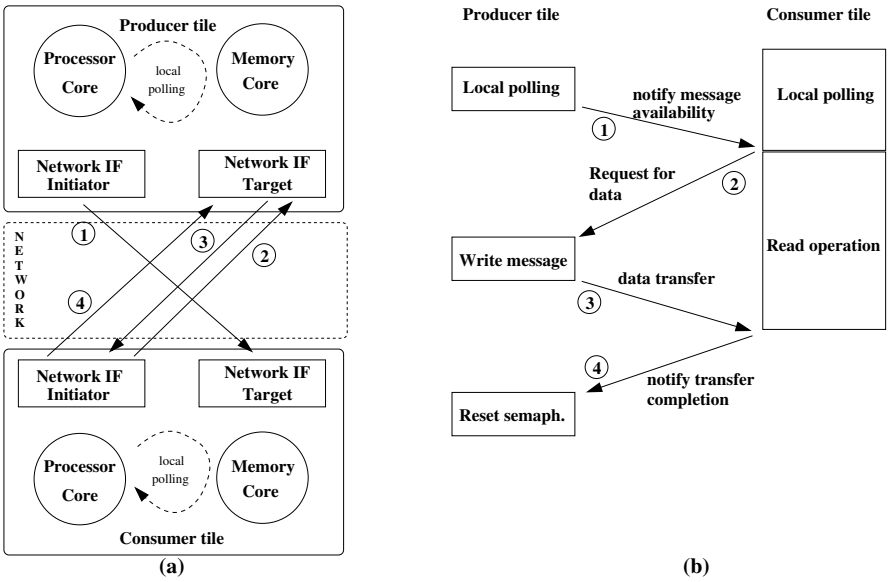


Figure 3.14: Tile abstraction and mapping of producer-consumer communication handshake on network transactions.

3.3.4 Case study: Parametrical exploration

This case study explores the feasibility and the efficiency of topologies of the k-ary n-mesh family by means of a parametrical exploration. For the same number of communicating cores, the performance of a 2D-mesh is compared against that of an equivalent hypercube and of concentrated mesh topologies (see Section 3.2.2). This case study focuses on CMP tiled-based design, composed of 16 tiles system with access to external I/O devices. The I/O devices are in charge of providing new raw data to the system and retrieving the processed data, while the tiles are responsible of processing raw data. The TL simulator above exposed will be combined with the computation tile simulation model described in this case study in order to explore regular NoC topologies. At first, performance of topologies was derived from TL simulation in terms of clock cycles. Next, the possible variants in the implementation space (clock frequency, link latency) of topologies were explored.

Tile architecture

In essence, the tile architecture consists of a processor core and a local memory core, as illustrated in Figure 3.14(a). Both cores are connected to the network through a network interface initiator and target respectively. It is assumed that the two network interfaces can be used in parallel. While the processor is reading/writing from/to other tiles, the processor core of other tiles can read/write from/to the tile local memory. This can be achieved through a dual-port memory and a proper tile architecture, which however falls outside the scope of this dissertation.

Communication protocol

The optimal topology for a given design is highly dependant on the traffic pattern it is going to accommodate and which is used during topology exploration. This case study aims at projecting network traffic based on moderns communication middleware for MPSoCs and to assess its performance with an NoC as the communication backbone. The guidelines for producer-consumer interaction are derived from the queue-based library in [31]. That library is suitable for a number of MPSoC architectures, including distributed architectures with local and tightly coupled memories for each processor core. This matches the tiled CMP scenario addressed in this study.

An abstraction layer was built on top of the TL simulator, which models the behavior of a processor tile and of its HW/SW communication support. While read and write transactions to the network are modelled with almost cycle-true accuracy, timings for communication control and for computation in the tile were estimated based on the work presented in [120].

A producer-consumer communication scheme between tiles based on the handshake in Figure 3.14(b) is assumed. 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. The consumer then reads computation data

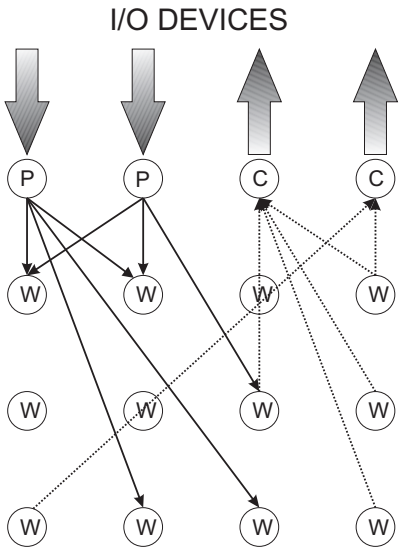


Figure 3.15: System organization and workload distribution.

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 to check pending messages for the target consumer is performed in order to avoid congesting the network in case the consumer is slow in absorbing its input messages. This behavior may result in low network bandwidth utilization, thus making global performance mainly sensitive to network latency.

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

Workload distribution

In order to simplify the analysis, a workload distribution between the tiles which de-emphasizes the role of the topology mapping algorithm with respect to overall performance is assumed. In fact, a parallel benchmark consisting of one or more *producer* tasks, a scalable number of *worker* tasks and 1 or more *consumer* tasks is considered (see Figure 3.15). Every task is assumed to be mapped on a different computation tile. The *producer* task(s) reads 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. The higher the number of worker tiles, the higher the data processing rate, assuming that the I/O interfaces can keep up with it. Output data from each worker tile is then collected by a *consumer* tile, which writes them back to the I/O interface.

The following assumptions were made regarding the I/O interface. It models input (output) data streams that are read from (written to) I/O devices, which are accessed through read/write operations like in SDRAM accesses. We assume that input and output streams do not interfere with each other, but are handled through different I/O ports. A maximum of 5 I/O ports is assumed, that can be used for input or for output. 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 network-based embedded multi-core products, such as [6]. In order to stress the topology differentiation, it is assumed that the I/O tiles are located on the same side of the chip, due for instance to floorplanning constraints. This constraint assures that collisions between input and output streams are generated. The case where input and output tiles are located on opposite sides of the chip did not show significant performance differentiation between alternative topologies, due to the effects of the communication protocol above exposed.

Topologies under test

The topologies to evaluate are: a 4-ary 2-mesh (a classical 2D-mesh) with one tile per switch, a 2-ary 4-mesh (also known as 4-hypercube) with one tile per

Topology	4-ary 2-mesh	2-ary 4-mesh	2-ary 3-mesh	2-ary 2-mesh
Switches	16	16	8	4
Tiles per Switch	1	1	2	4
Maximum Switch Degree	6	6	7	10
Unidirectional Links	48	64	24	8
Bisection Bandwidth	8	16	8	4
Hops Count	7	5	4	3
Connectiviy	2	4	3	2

Table 3.5: Topologies under test.

switch, a 2-ary 3-mesh with 2 tiles per switch, and a 2-ary 2-mesh with 4 tiles per switch. These two latter solutions are denoted as the concentrated topologies. Table 3.5 shows the high-level properties exposed in Section 3.2 of the four studied topologies. Please note that switch degree numbers do not follow the guidelines exposed in this section. The reason is that for each tile 2 switch input and 2 switch output ports are required, since the tile includes an initiator and a target NI, each with one input and one output port to the switch for receiving/sending data.

Regarding the routing algorithm, the reference architecture forces the use of deterministic routing algorithms. The implemented routing algorithm was the commonly used Dimension Order Routing (DOR). In this deterministic routing algorithm, the dimensions of the topology are first ordered, then packets move in the lowest dimension in which current and destination switches are not aligned. This latter step is repeated until the destination switch is reached. As an example, for a 2D-mesh, dimensions are usually labelled as X and Y. Each switch has an identifier based on its X and Y coordinates, forming a tuple (X, Y). Assuming that X is the lowest dimension and Y is the highest one, a packet actually allocated in switch $(0, 0)$ that must reach switch $(2, 2)$ will first move in the X dimension, thus reaching switch $(1, 0)$ and then switch $(2, 0)$. After, it will move in the Y dimension, thus crossing through switch $(2, 1)$ to finalize its travel through the network at switch $(2, 2)$. More

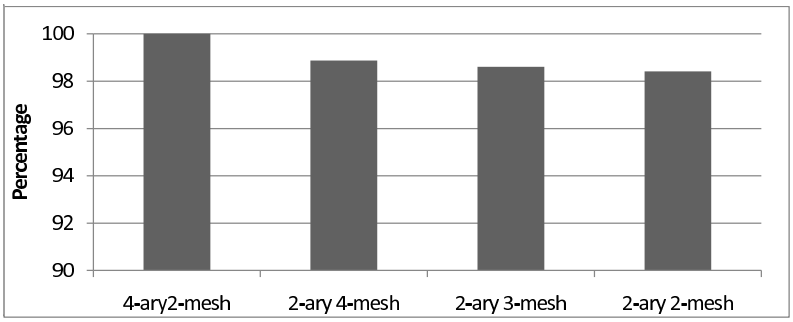
details about DOR can be found in [52].

Topology performance results

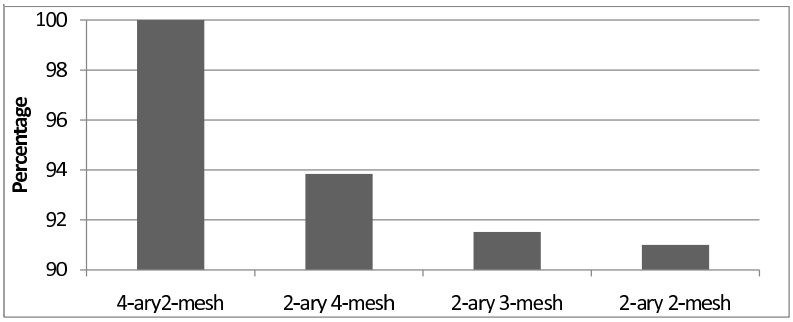
The reference architecture requires the ratio between core and network clock cycles, which is constrained to be an integer divider. Previous works of the reference architecture suggest that for the systems under test, a target clock frequency of 1 GHz is to be expected for the network switches, while the links are assumed to be able to cope with this frequency. Regarding the tiles frequency, as above exposed, a reasonable frequency is 500 MHz, while I/O devices were assumed to work at that same frequency. As a consequence, ratio of 1:2 for all clock domain crossings is set. Should such network frequencies prove infeasible, the implications would be assessed during the implementation space exploration. An initial latency of 20 cycles for external I/O device access was set. By varying the number of producer and consumer (I/O) tiles and the computation time of the worker tiles for each data unit, 4 different performance scenarios were defined. Figure 3.16 shows the estimated cost in cycles that each topology requires to process 64000 data units, normalized to the results obtained for the 2D-mesh (4-ary 2-mesh).

A. Bottleneck in the workers: In this scenario, the I/O interfaces are fast enough to feed workers with raw data and to absorb processed data from them. As a consequence, producer and consumer tiles experience idleness, since they are waiting for workers to complete. As shown in Figure 3.16(a), in this case performance does not depend on the topology, since the network is not the bottleneck. Even though some topologies provide lower latency, the time a worker takes to read input/write output data is masked by other workers processing their data in parallel. Overall, the performance advantage of concentrated topologies is almost negligible in this scenario. Interestingly, topology selection is just a physical issue here, since the concentrated solutions certainly involve less resources and probably less power, but are likely to run at lower frequencies.

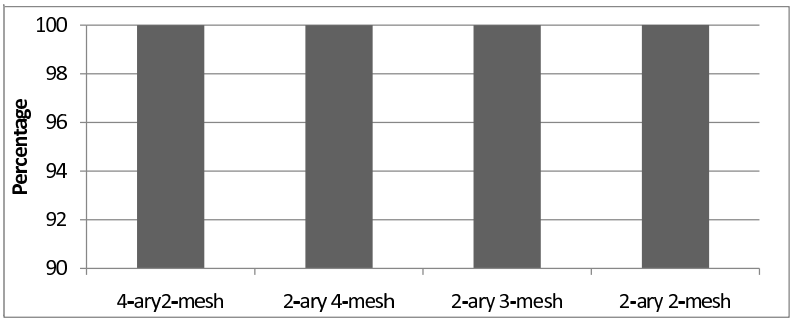
B. Bottleneck in the consumers: Figure 3.16(b) shows that this scenario provides the largest performance differentiation among topologies. The input interface is fast enough to minimize the time that worker tiles expend waiting for new computation data. On the contrary, each worker waits almost



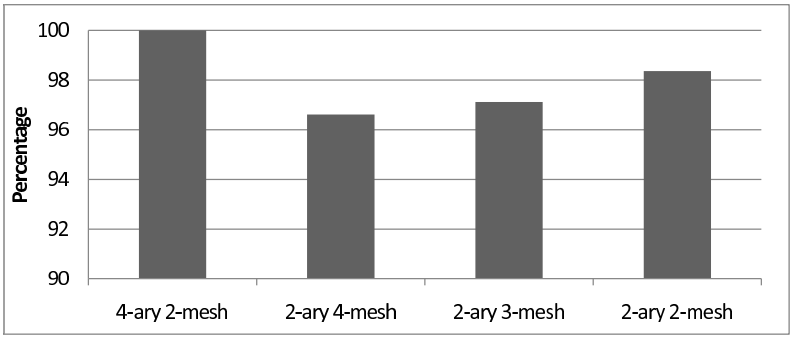
(a) Bottleneck in the workers



(b) Bottleneck in the consumers



(c) Bottleneck in the producers



(d) Balanced

Figure 3.16: Performance comparison (in execution cycles) of topologies.

50% of its execution time in the simulation to forward its output data to the consumer tile, which cannot keep up with the needed consumption rate. In this case, the shorter the transfer time to the consumer, which is impacted by network latency, the sooner a new data unit can be processed. This is in essence a network latency-sensitive scenario, where concentrated topologies outperform the others, due to a lower hop count.

C. Bottleneck in the producers: This is the case where a lower network transaction latency does not help. In fact, the producer tile is not able to keep up with data requirements from the workers. The use of low-latency topologies allows the workers to read input data more rapidly, but then, once the computation of those data is complete, the worker becomes idle again waiting for new data units to process. A longer data unit processing time would not have impacted performance. In summary, workers and consumers suffer from starvations, so all topologies perform the same in this scenario (see Figure 3.16(c)). Again, the choice among topologies will be based on physical implementation considerations (e.g., power vs performance trade-off).

D. Balanced scenario: In this scenario, the idle time of all tiles in the system (producer, consumer and worker) for mutual synchronization was minimized. This system configuration puts the highest bandwidth pressure to the network. This explains the performance results in Figure 3.16(d). The 2-ary 4-mesh provides more bandwidth and therefore achieves the shortest execution time. In contrast, concentrated topologies trade bandwidth for low latency, and therefore provide worst performance than the 2-ary 4-mesh in this bandwidth-sensitive scenario, but still outperform the 2D-mesh.

Implementation space exploration

The next step consist of an analysis about how performance ratios between alternative topologies are impacted by possible implementation degradation effects. These considerations will drive the synthesis process by posing optimization directives to specific logic modules and/or links in order to preserve theoretical performance benefits. As an example, the two most relevant scenarios from the previous step are explored.

In particular, scenarios B and D were selected. B is a latency-sensitive scenario, while D is a bandwidth-sensitive one. In D, the implementation space

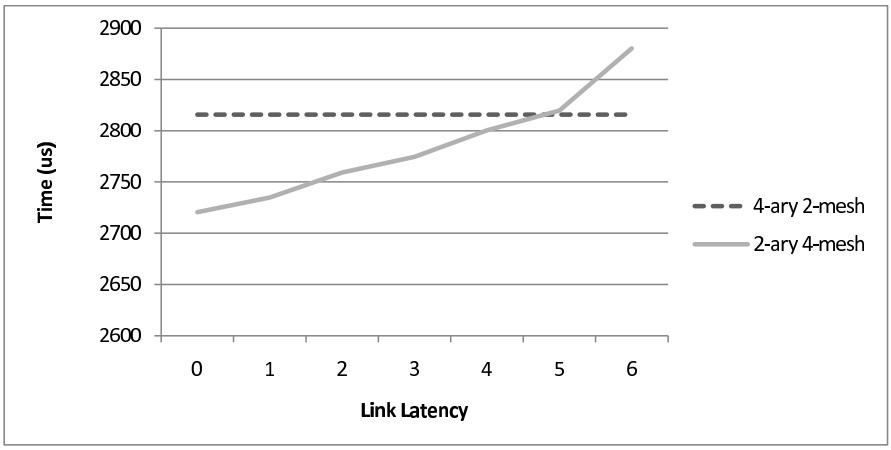
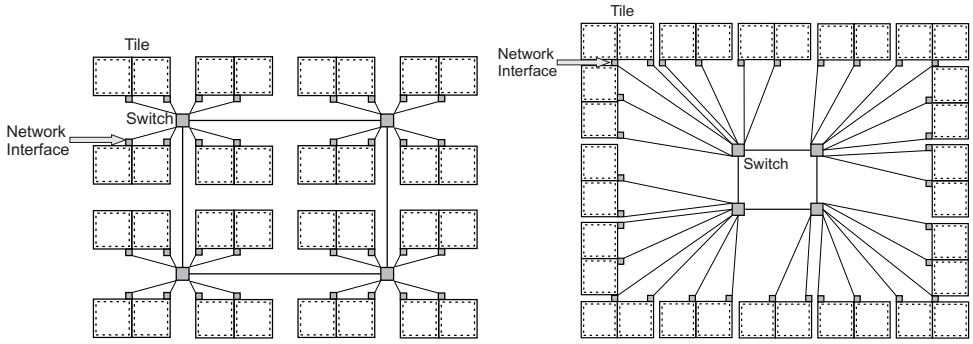


Figure 3.17: Implementation space: 4-ary 2-mesh vs 2-ary 4-mesh.

exploration is restricted to the 4-ary 2-mesh (reference topology for many NoC designers) and to the promising 2-ary 4-mesh solution. Both topologies require the same switch degree. Therefore, the operating frequencies for the two topologies might differ only as an effect of the different wiring patterns and of the critical path being in the links. Since all NoC modules in the reference architecture feature input and output latching, the critical path will be determined either by the switch or by the network interface. If link delay exceeds this maximum delay, it is assumed that such critical links will be broken via retiming and repeating stages, following the link pipelining technique exposed in Chapter 2.

While links of the first and the second dimension in both topologies can be retained of comparable length in many floorplan variants, the links from the third and the fourth dimension in the 2-ary 4-mesh are the most likely candidates for retiming (see Section 3.2.2). Figure 3.17 reports how the performance of the 2-ary 4-mesh is affected by the latency on those links. In the figure, the horizontal line represents the performance of the 4-ary 2-mesh, in which all links have a latency of 0 cycles. As it can be observed the 2-ary 4-mesh remains a competitive solution with up to 5 cycles of latency on the links of the higher dimensions. Higher latencies in those links make the 2D-mesh the most competitive solution. This leaves ample margin to the physical synthesis tool to work with.

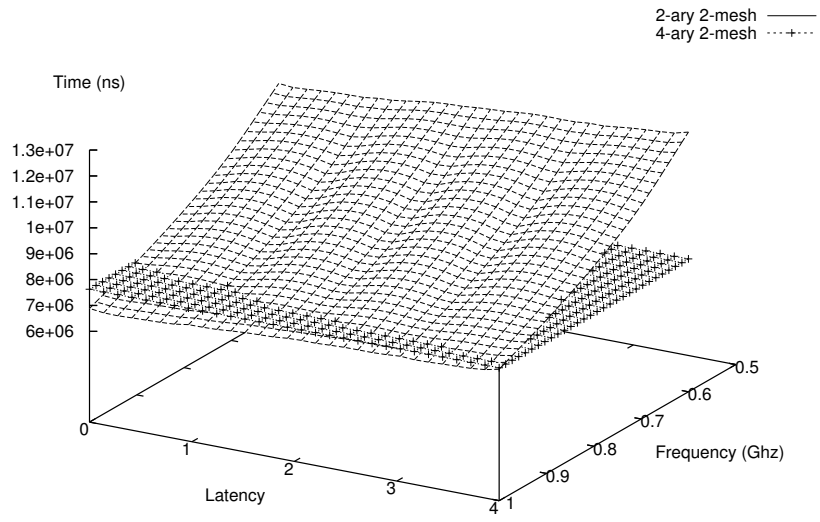


(a) Latency in the network links (b) Latency in the injection/ejection links.

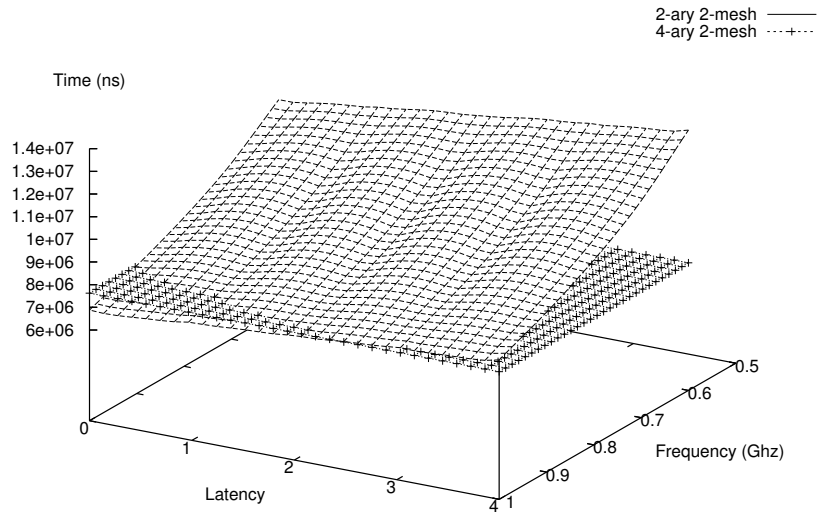
Figure 3.18: Floorplanning directives for a 2-ary 2-mesh with 4 tiles per switch.

Regarding scenario B, the implementation space complexity increases due to another physical parameter that the exploration has to account for: the maximum achievable frequency. In this scenario, the best two solutions are compared: 4-ary 2-mesh vs. 2-ary 2-mesh concentrated. This latter requires a switch degree of 10 (it is 6 in the 4-ary 2-mesh), which might lead to severe cycle time degradations. Moreover, concentrated topologies present some floorplanning challenges. In particular, a large number of cores (4 in this case) need to be placed close to the same switch where they are attached to. This might be achieved in many ways, posing different layout constraints. As an example, a possible floorplan is to place all 4 cores as close as possible to the switch, so the switch–to–switch links might be as long as the side of two tiles in the best case, as depicted in Figure 3.18(a). Such links might be candidate for retiming stage insertion. Alternatively, the floorplan might consist of a central network around which all cores are placed (see Figure 3.18(b)). In this variant, the critical links are those connecting the network interfaces (attached to the tiles) to the switches, also known as injection/ejection links. In the two floorplan variants, the latency incurred by different links might impact performance in a radically different way, since different communication flows are affected.

We kept the 4-ary 2-mesh topology at the reference frequency of 1 GHz, while scaling clock frequency of the concentrated topology and the latency in its switch–to–switch or injection/ejection links. Execution time results are



(a) Latency in the network links



(b) Latency in the injection/ejection links.

Figure 3.19: Implementation space: 4-ary 2-mesh vs concentrated 2-ary 2-mesh.

illustrated in Figure 3.19. As network frequencies are scaled, the computation tile and external I/O device frequencies must be scaled accordingly, since the reference architecture forces an integer divider between the frequencies in the two clock domains. So, as network frequency scales down from 1 GHz to 600 MHz, tiles and I/O devices were forced to work from 500 MHz (their assumed maximum speed) to 300 MHz, keeping a frequency divider of 2. If the 500 MHz case for the network were considered, the divider will have been changed to 1 instead of further scaling down I/O frequency, but this would have led to a different performance scenario than B.

The horizontal plane in Figure 3.19 represents 4-ary 2-mesh performance. Figure 3.19(b) indicates that as soon as the frequency of the concentrated topology falls below 900 MHz, the performance improvements of the concentrated topology are at first balanced and then progressively vanish. If the physical synthesis is able to limit switch operating frequency degradation in the concentrated topology to 100 MHz, then its performance benefits can be retained even placing up to 2 retiming stages in the injection/ejection links. In contrast, Figure 3.19(a) indicates that performance of the concentrated topology is much less sensitive to latency in the switch–to–switch links. In this case, not all communication flows are affected, since communications between tiles attached to the same switch do not go through the longer links. On the contrary, when latency of the injection/ejection links is increased, each communication flow incurs always twice extra cycles latency, one time in the injection and another one in the ejection links.

Once the parametrical exploration is performed, it outputs the directives that will guide their physical synthesis. In the case where those constraints became too tight, the topology can be safely discarded as non-competitive.

3.4 Physical design pitfalls

When exploring the design space for a given NoC design, it is necessary to evaluate if the high-level properties of a given topology will hold when it is mapped on a 2D-layout. There are many considerations putting the aforementioned claim in discussion. This section will point out several physical design pitfalls that are not always easily recognizable at first glance. While the previ-

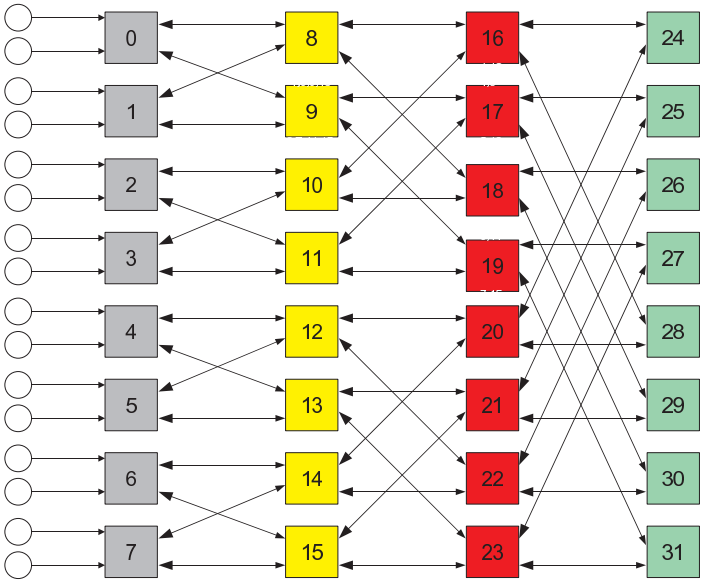


Figure 3.20: High-level sketch of a 2-*ary* 4-*tree*.

ous section presented a case study in which the simulation framework guided the synthesis process by means of a parametrical exploration of the candidate topologies, this section will show how to use the simulation framework in order to back-annotate physical synthesis results in order to provide accurate performance estimations.

A common approach to capture physical design implications of a topology in the early design stages is by means of pencil–and–paper floorplanning considerations. Unfortunately, a number of assumptions are often made that are not verified on actual layouts.

For instance, with this approach, the same length is typically assumed for all the links in the topology, thus neglecting the constraints imposed by the 2D silicon surface. The example in Figure 3.20 depicts the high-level connectivity pattern of a 2-ary 4-tree topology, while a possible layout realization of that connectivity pattern is shown in Figure 3.21 (where only some links are illustrated). When comparing both figures it is clear that the physical view is radically different from the abstract representation and it may not be intu-

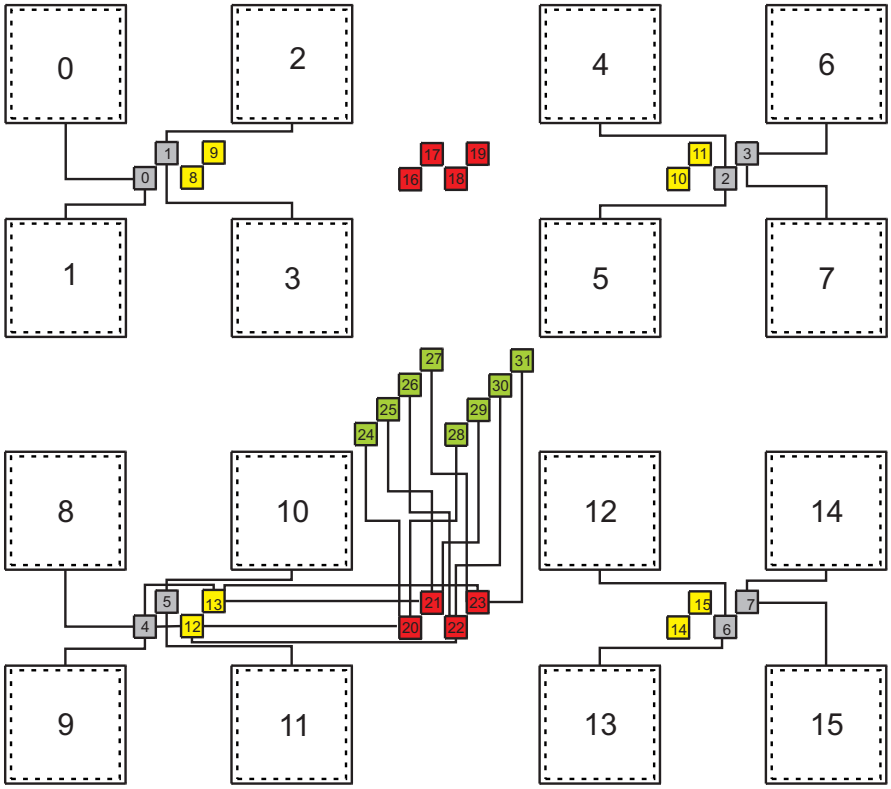


Figure 3.21: Floorplan of a 2-*ary* 4-*tree*. Only the main wiring patterns are reported.

itive at all. Notice that the width/height of a core is much larger than that of a simple network switch. This is a simple layout constraint that is often neglected when devising a possible topology floorplan.

In other topologies, the use of long links may arise some issues. That is the case of the C-Mesh, where the utilization of express links comes at a considerable price. In fact, as an express link turns out to be a very long link to be routed, it is likely to limit the overall operating frequency of the topology. Obviously, there are well known methods to solve this problem, namely repeater insertion and/or link pipelining. However, they come with an area cost and in some cases they induce an ever higher overhead in the architecture.

Another design pitfall concerns switch homogeneity. Some topologies may

require switches of different degree. For instance, in a 2D-mesh, there are typically three different degrees. Assuming that a single core attached to each switch and that a core requires a single I/O port towards the switch it is connected to, switches located at the corners of the grid will have a degree of 3, the ones placed in the border of the grid (but not in the corners) a degree of 4, while any other switch will present a degree of 5. So, a switch in the corner is potentially the fastest of the entire network. However, the final operating frequency of the topology is set by the switch with the highest degree in combination with the longest link in the topology. Therefore, these border switches will end up being synthesized at a lower speed than the maximum one they could achieve, resulting in lower area. An early area estimation framework neglecting these effects in non-homogeneous topologies would be highly misleading.

Finally, physical routing may become a design pitfall that is usually ignored. Some designs employ a NoC to interconnect different IP blocks provided by various vendors. Those IP blocks could be provided either as non-routable hard-macro, or as soft-macros. The formers prevent routing NoC links through the IP blocks (also known as *over-the-cell* routing), forcing the physical designer to reserve physical routing channels for the routing of links. Instead, soft-macros made over-the-cell routing feasible, thus giving rise to more compact layouts. However, even in this case, routing on top of IP blocks may not be convenient due to the effects of inferring link repeaters in the same fence of the IP block.

The next sections will provide more solid evidence of the design pitfalls illustrated above by means of several case studies where the physical design of several topologies turns out to be extremely challenging. In the first case study, topologies for a 16-tiles system will be laid out on silicon. In the second one, topologies for 64-tiles systems will be considered, although they will be evaluated by means of a modeling framework of physical effects inspired by the first case study. In those two case studies, MIN topologies will be left out on purpose. Their intricate wiring pattern justifies a detailed case study dedicated to the evaluation of their feasibility and viability. In each case study, the proposed simulation framework will be used to show the gap between high-level performance estimations and layout-aware ones. All physical synthesis

Topology	4-ary 2-mesh	2-ary 4-mesh	2-ary 2-mesh
Max. Degree	6	6	10
Switches	16	16	4
Max. Hops	7	5	3
Bisection bandwidth	4	8	2
Tiles per Switch	1	1	4

Table 3.6: Topologies under test.

experiments of this chapter have been carried out by means of a commercial synthesis toolflow on an industrial 65nm technology library.

3.5 Case study: 16-tiles systems

This analysis focuses on the k-ary n-mesh family of topologies (see Section 3.2.2), as it offers representative design points for our analysis. For example, it may present links of uneven lengths or heterogeneous switch degrees.

The following topologies are considered. First, the baseline solution is a 4-ary 2-mesh (denoted as 2D-mesh for now on) with one tile per switch. Second, a 2-ary 4-mesh with one tile per switch as the high bandwidth solution. Third and last, a 2-ary 2-mesh with 4 tiles per switch is considered as the low latency solution (denoted as concentrated 2D-mesh). Table 3.6 shows some representative data of the studied topologies. Please notice that the maximum degree numbers does not follow the equations presented in Section 3.2.2. This is due to the tile architecture considered in this case study, which is similar to the one presented in the case study of Section 3.3.4. Regarding tile size, assuming a core size of 1 mm x 1mm, a full tile size will be 1mm x 2mm. Such an asymmetric size will heavily impact the physical synthesis of the studied topologies.

Both the 2D-mesh and its concentrated counterpart feature 2 dimensions and homogeneous inter-switch wire lengths. However, such wire length will not be the same in the two topologies due to floorplan constraints. In particular, the concentrated solution requires to place cores around the switches

they have to be connected to, thus separating the switches in space. Also, the concentrated solution presents a switch degree higher than the baseline solution (the 2D-mesh).

A different design point is represented by hypercubes (represented by the 2-ary 4-mesh in this analysis). This topology keeps the same maximum switch degree than the 2D-mesh. Unfortunately, this comes at the cost of links with uneven length. In fact, as state in Section 3.2.2, in a mesh with more than two dimensions the links used to connect the dimensions greater than two are longer, and this holds for 50% of the 2-ary 4-mesh switch–to–switch links.

Finally, the implemented routing algorithm was the commonly used Dimension Order Routing (DOR), exposed in the case study of Section 3.3.4.

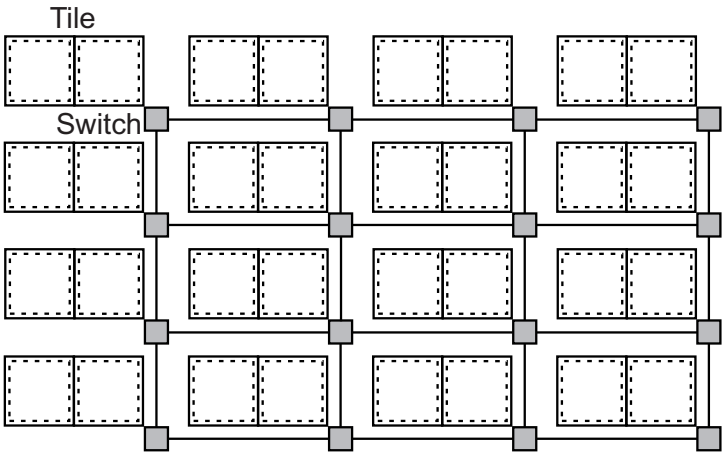
3.5.1 Total wire length

The way link design techniques impact topology quality is twofold. On one hand, when the critical path is not in the switches, it is the longest link in the topology that determines the maximum achievable speed. Therefore, cutting down on the delay of that link is beneficial for the whole topology. On the other hand, it is possible to use link performance boosting techniques in order to return the critical path to the switches or just alleviate it. But the cost of such techniques becomes relevant for the topology, mainly when there are more such critical links and they account for a significant fraction of the total topology wirelength.

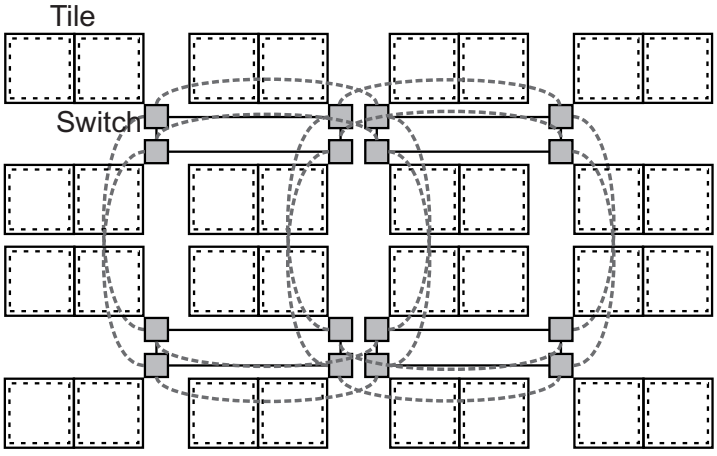
In the experiment that follows, the impact of switch–to–switch link on total topology wirelength will be assessed. Moreover, it will be demonstrated that drawing a conclusion on this based on abstract considerations on the topology connectivity pattern is highly misleading.

Let us consider the floorplanning directives given for topology synthesis reported in Figure 3.22.

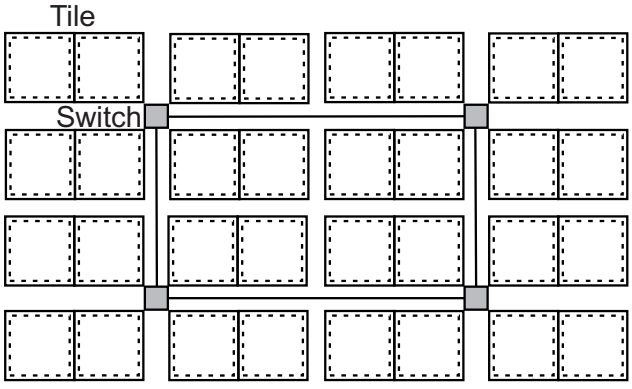
The asymmetric tile size plays in favor of the 2-ary 4-mesh wiring, since the length of the horizontal and of the vertical long links (3rd and 4th dimensions) turns out to be comparable to that of horizontal wires in the 2D-mesh. This latter also features horizontal and vertical links of unequal length, indicating that the layout regularity often assumed in high-level considerations does not materialize in practice. The floorplan definition aims at: shortening



(a) 2D-mesh (4-ary 2-mesh)



(b) Hypercube (2-ary 4-mesh)



(c) Concentrated 2D-mesh (2-ary 2-mesh, 4 tiles per switch)

Figure 3.22: Floorplan directives.

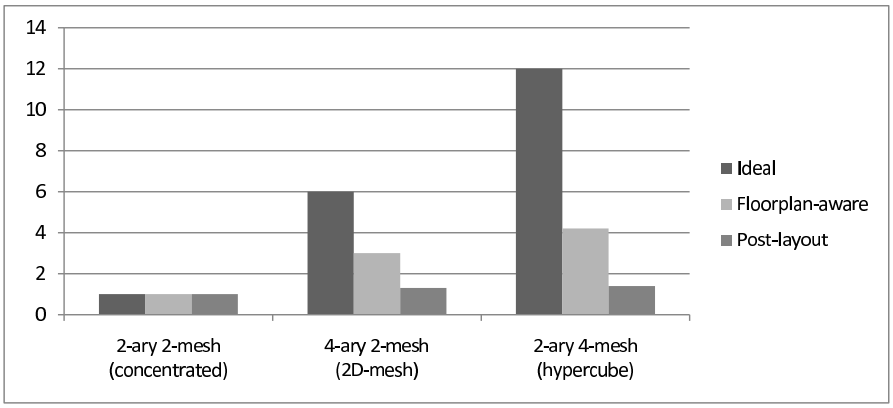


Figure 3.23: Total wire length.

the longest links in each topology and coming up with scalable floorplans. For the 2-ary 2-mesh, the computation tiles have been placed around the switch they are attached to, as it is the placement less sensitive to link latency, as demonstrated in Section 3.3.4. 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 links.

As a result of topology synthesis and place&route, Figure 3.23 shows the total wire length for the three topologies (*Post-layout* bar), normalized to the least wire-hungry topology. Bar *Ideal* computes wire length based on the theorethical formula given in Section 3.2.2. Bar *Floorplan-aware* updates the previous formula with the knowledge of the asymmetric core size and of switch placement. The ideal analysis largely overestimates the amount of wiring needed for the 2-ary 4-mesh. 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 synthesis results, where the post-layout report of total wire length gives only a 10% overhead of the 2-ary 4-mesh wiring with respect to 2D-mesh and a 43% with respect to the concentrated solution. 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 concentrated topology (2-ary 2-mesh) to 26% for the 2-ary 4-mesh. This explains the relatively small total wire length difference between the

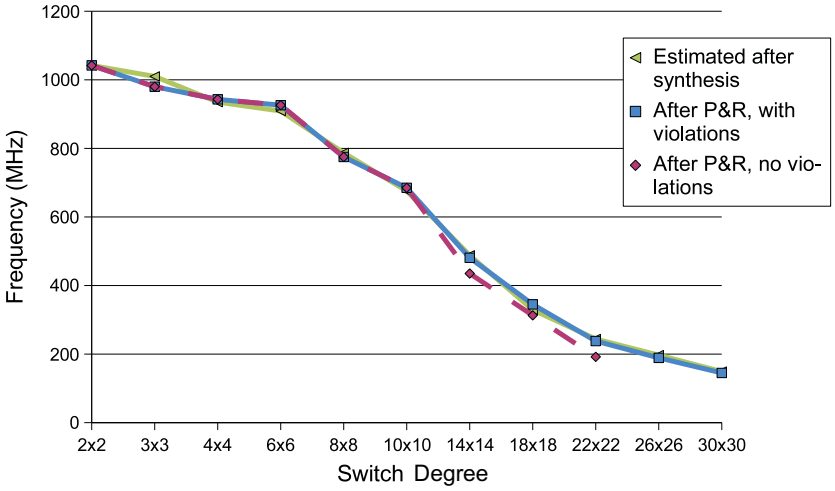


Figure 3.24: Switch frequency scaling (courtesy of [123]).

topologies. This scenario plays in favor of engineering performance-optimized inter-switch links with a possibly minor impact on topology cost metrics.

3.5.2 Switch degree

As mentioned, there is a direct correlation between the switch degree and the final operating frequency of the switch itself. Essentially, an increment of the switch degree has a direct consequence on the delay of the arbitration logic and of the crossbar selection logic, which adds up to the critical path. The effects of switch degree over the reference architecture have been previously analyzed in [123]. What follows is a brief summary of [123], highlighting the most relevant conclusions for this dissertation.

Figure 3.24 shows the evolution of the maximum achievable operating frequency of a switch as its degree is increased. As can be seen, the above mentioned trend is unmistakable. Regarding area and power figures, they increase as the switch degree increases, as expected.

The most interesting result concerns feasibility of switches with a large degree with a standard cell design flow. In fact, logic synthesis tools are typically aware of placement but not yet of routing. As a consequence, for a 14x14 switch, the wire density in the switch crossbar becomes unmanageable while

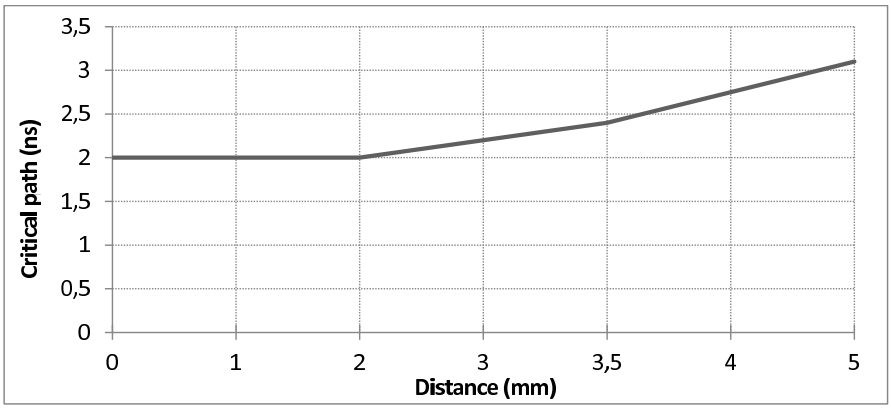


Figure 3.25: Critical path trend when increasing link length.

keeping a high operating frequency. Two possible solutions exist: increasing the switch area or decreasing the switch frequency. While the latter solution obviously penalizes overall system performance, the former option proves only partially effective, as the area efficiency of the switch quickly diminishes. In fact, as the switch degree keeps growing, the area overhead of such solution quickly becomes unaffordable.

Even in cases where those issues are fixed by some means, achieved results suggest that avoiding too large switches may be the best option. This is also due to system-level effects that would result from using large centralized blocks (i.e., many cores attached to the same switch), which are not immediately apparent from the results reported here. For example, the many cores connected to such a switch would ideally need to be physically placed just around it, causing obvious congestion in the floorplan. Alternatively, they could be spread around, but then several long links would be needed to connect remote cores to the switch. These links would require pipelining, bringing further latency, area and power costs.

3.5.3 Link length

An interesting physical design issue is the location of the critical path. Traditionally, it is assumed to be inside the more complex network blocks (i.e.,switches or network interfaces). However, the reverse scaling of the delay of on-chip interconnects questions this assumptions. Let us now consider a simple system

where, initially, two switch blocks are placed at the distance of 1mm. After placement and routing at such distance, the critical path of the whole architecture is still inside the switch block. The reason is that the delay of the inter-switch channel is not dominating yet the overall timing of the system. By increasing the distance between both switches results differ, as depicted in Figure 3.25. Switch block were synthesized with a target frequency of 500MHz (critical path of 2ns), and even with such a low operating frequency the critical path shifts to the link already at the short distance of 2mm. In fact, from this point on, the critical path is determined by the link delay, which is a typical phenomenon in interconnect dominated designs. That is, the link delay dominates the whole design, shifting the critical path from the switch to the communication channel going from the output of the first switch to the input of the second one.

In general, this result points out the critical role played by the interconnects in the on-chip domain. In fact, it defines the performance of the network topology. If the main goal is performance, a topology with long links can not be a good candidate as it will not be able to operate at high frequencies due to large link delays. Even worse, as technology keeps scaling to the nanoscale regime, the critical path tends to move to the switch–to–switch links at progressively shorter link lengths than illustrated in Figure 3.25, although the break-even point depends on the actual constraints guiding switch synthesis. For example, a switch synthesis targeting at low power designs, it is likely the break-even point will be shifted to a large distance. In this context, link performance boosting techniques are beneficial for the performance of the entire network, and will be investigated in next section.

3.5.4 Link performance boosting

In this section, three fundamental link inference techniques are evaluated when laying out a set of representative topologies. The objective is to speed up topology implementation by boosting its links. Therefore, the primary design objective is high-performance.

Performance boosting technique

The topologies under test are again those of Figure 3.22. The first round of topology implementations implements un-repeated links. In essence, the backend tools have been prevented from instantiating buffers or inverters along the link.

The second round employs a performance boosting strategy consisting on the insertion of repeaters along the link. The objective of those repeaters is to strength the signal at some points along the link, allowing the signals to travel a longer distance in the same time. An increase of instantiated buffers can be expected, as well as a significant cut down on link delay.

Finally, link pipelining techniques were applied to break long timing paths across links. Those techniques split problematic links into segments with the aim to remove the critical path from them, but at the cost of introducing multi-cycle links. That its, they offer a trade-off between link operating frequency and the cost in cycles of traversing the link. Synthesis tools do not provide a native support for this, thus requiring additional design effort. The ideal approach is not to manually place pipeline stages in the floorplan, but to let the tool handle this based on design constraints and optimization directives, as the tool has better visibility of floorplan constraints and design rules. When implementing link pipelining, flow control issues need to be re-considered. One solution is to implement pipeline stages not as simple retiming stages, but as retiming and flow control stages, as is the case of the reference architecture (see Section 2.1.9), where pipeline stages consist of 2 flit buffers plus a control logic and handle the STALL/GO flow control protocol.

Timing closure

Since the final goal is high-performance, the objective is to materialize the maximum speed achievable by the slowest NoC module (synthesized in isolation with post-place&route timing analysis). In the reference architecture the critical module turns out to be always the switch with the highest degree in the topology. The resulting delay defines the system speed upper bound, and ignores the effect of inter-switch links (Table 3.7, first and second row). For the slowest switch of each topology in isolation, the gap between post-synthesis

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 3.7: Timing results.

and post-place&route speed ranges between 12% to 21%. The 2-ary 2-mesh exhibits the highest post-synthesis delay due to its high switch degree.

When considering timing closure for the whole topology, the degradation associated with inter-switch link routing becomes apparent. In the case of repeater-less links first (Table 3.7, third row). While the 2D-mesh and the 2-ary 4-mesh present similar switch delays (since they have switches with the same maximum degree), the complex routing of the 2-ary 4-mesh translates into a 39% degradation of the critical path, while routing of the 2D-mesh is less critical. Regarding the 2-ary 2-mesh, inter-switch routing has an even more relevant impact, has it presents some links longer than 4mm. This effect masks the issues of the higher switch degree, which in this case is not the responsible for the slower operating frequency. For all topologies, the critical path goes through the network links.

It is possible to optimize problematic links to overcome these large delays. Activating repeater-insertion during topology synthesis enables the speedups illustrated by the fourth row of Table 3.7, which denotes 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 in a delay of 1.19ns. This value is quite close to the performance upper-bound (1.12ns), thus indicating that the degradation of topology performance induced by network links can be made irrelevant in this case.

On the contrary, the 2-ary 4-mesh does not improve it critical path when

inserting repeaters in the problematic links. The reason for this lies in the fact that horizontal routing channels (see Figure 3.22(b)) were sized conservatively small: approximately 2.5 times the switch side. For this reason, switch–to–switch links sometimes end up finding a switch on their way. This is a placement constraint for the repeaters that prevents their insertion with ideal spacing. The result is that link performance is not improved in this case, demonstrating 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 to 4 times 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, as its connectivity pattern is trivial. Thus, it improves the critical delay by 10% over the repeater-less case.

Further critical path improvements were expected from link pipelining. When applying this technique, the objective was to materialize the same critical delay of the 2D-mesh (with repeaters) even for the 2-ary 4-mesh and the 2-ary 2-mesh. Interestingly, link pipelining turns out to be effective even for the 2-ary 4-mesh solution, 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-degree switches (1.4ns), far worse than the 2D-mesh link delay of 1.19ns.

Area overhead for link performance boosting

In order to assess the implementation cost for link performance boosting techniques, area reports are illustrated in Figure 3.26. Results are grouped by topology and normalized to the baseline implementation (repeater-less) of each topology.

Repeater insertion is quite cheap for the 2D-mesh and the 2-ary 4-mesh, while it 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

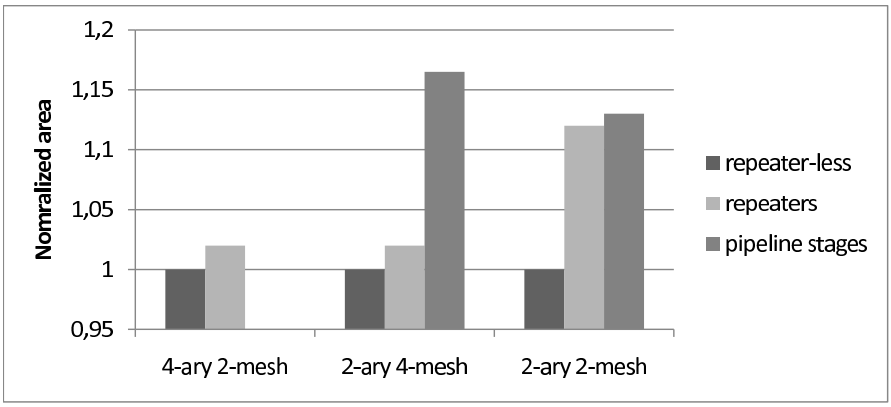


Figure 3.26: Normalized area.

of long links by dramatically increasing the number of buffering gates. As 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 is due to the fact that by inserting pipeline stages the tool was able to remove an equal amount of buffering area introduced by the use of repeaters, so that the two contributions offset each other.

This does not hold for the 2-ary 4-mesh, which raises its area by 14% over the 2-ary 2-mesh solution when pipelining is used. Not only the area overhead of the pipeline stages is incurred, but many repeaters are kept in the link segments between pipeline stages, since the frequency boost is significant when moving from repeaters to pipelining. The trend of leakage power fully reflects that of area overhead.

Energy efficiency

Although there is a price to pay to boost link performance in terms of area and power, the resulting speedup can be exploited to cut down the total energy required to finish a given task by reducing its execution time. However, this energy saving materializes only if the gain in execution time outweighs the power overhead.

The power efficiency of the evaluated systems was analyzed with a set of experiments with a traffic configuration similar to the one presented in Section 3.3.4. In this case, the workload distribution scheme was configured with

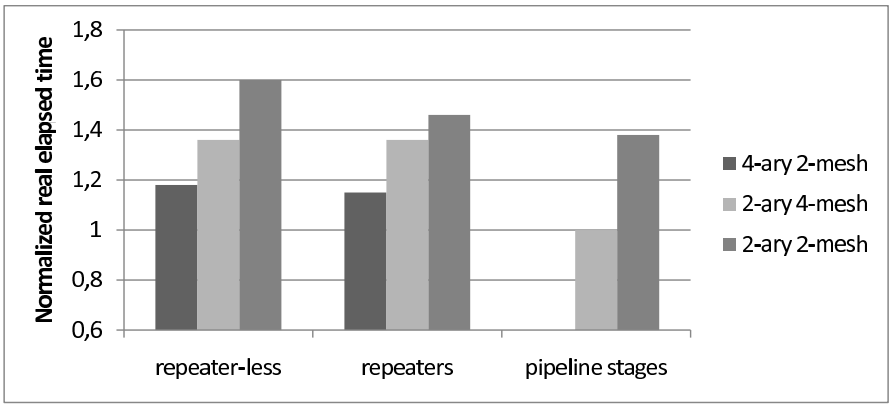


Figure 3.27: Real elapsed time of topologies under test.

1 producer task, 14 worker tasks and 1 consumer task, but the communication protocol was simplified. In particular, data is sent between the different tasks without any previous check for destination availability to accept the message, simplifying the power efficiency analysis by removing the additional communications steps required by the original communication protocol. Also, traffic generation rates were set to have a balanced system operation (i.e. to avoid system related bottlenecks, like the I/O) thus avoiding to stress other design issues different than network bandwidth and link performance. Physical synthesis results above exposed were back-annotated into the proposed simulation framework in order to obtain realistic performance estimations of the topologies under test.

Real elapsed time results for the different implementations of each topology are reported in Figure 3.27. In those experiments, each system works at the maximum achievable post-place&route speed (see Table 3.7). Since the reference network interface implements a frequency-ratioed clock domains crossing mechanism, a clock divider of 2 is applied, forcing the tiles to run at half the speed of the network. Regarding 2-ary 4-mesh, its execution time is always higher in the repeater-less and the repeated implementations, due to its high critical delay. The situation is even worse for the 2-ary 2-mesh, which is the worst topology in those two implementations.

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

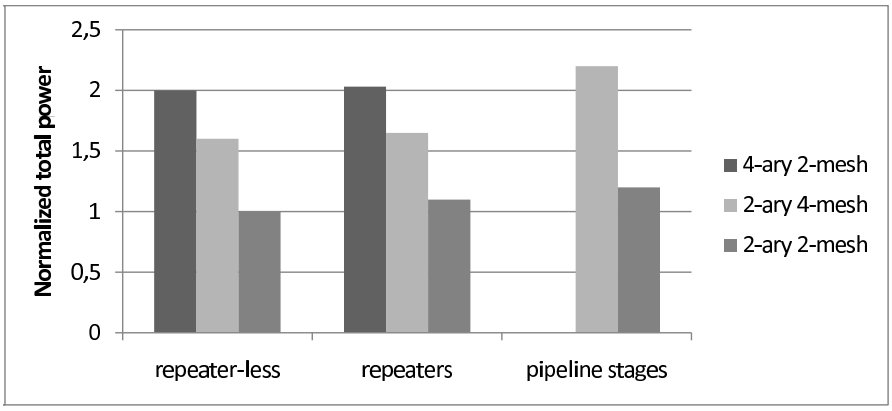


Figure 3.28: Total power.

solution. The interesting point in this conclusion is that the increment of the link latency in terms of cycle introduced by link pipelining techniques does not have a negative impact over total performance. The reason behind this behaviour is that in the reference architecture, pipeline stages provide 2 additional buffer slots, thus link pipelining enables a significant topology speedup, while providing additional buffering to the topology.

Figure 3.28 shows the power cost incurred by link boosting techniques. Power is affected by the operating frequency of each implementation. The first conclusion that can be drawn is that the use of repeaters does not have a relevant cost in terms of power for the 2D-mesh and for the 2-ary 4-mesh, due to the low number of inter-switch links when compared to the overall wire length. On the contrary, this is not true for the 2-ary 2-mesh, due to the large power cost of driving such long links effectively.

Regarding link pipelining, it use abruptly increases power costs, especially for the 2-ary 4-mesh. 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 2-ary 4-mesh with respect to the 2D-mesh mostly derives from its lower speed, although this is not the only reason. This is confirmed also by the marginal power overhead of the 2-ary 4-mesh when it can run at the same speed of the 2D-mesh (see 2D-mesh with repeaters vs pipelined 2-ary 4-mesh in Figure 3.28). This is counterintuitive, since the 2-ary 4-mesh has many more buffering resources than the 2D-mesh. The answer

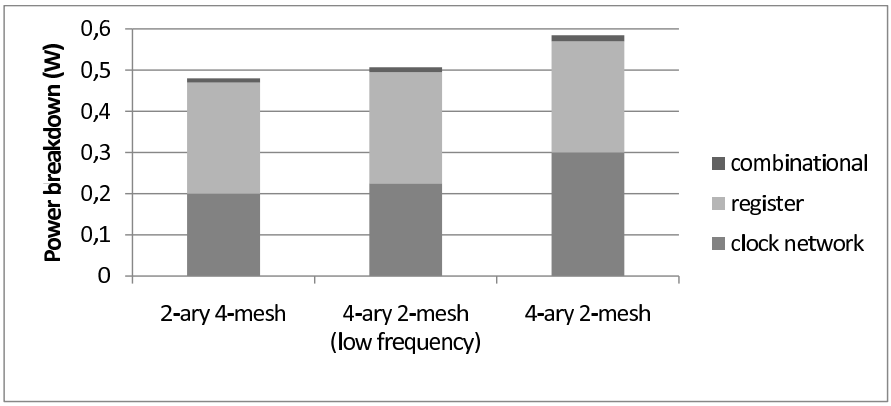


Figure 3.29: Clock tree power impact.

has been sought in the power breakdown. This is reported in Figure 3.29 for the repeater-less variants of the two topologies. Additionally, the 2D-mesh was re-synthesized and analyzed both at the operating frequency of the 2-ary 4-mesh for a fair comparison. In all cases, it can be observed that the clock tree has a relatively lower impact than register power (which depends mainly on the buffer resources) in the 2-ary 4-mesh, while in the 2D-mesh it weighs more. As a consequence, when the same speed is enforced, the two topologies present the same power. While register power obviously increases for the 2-ary 4-mesh (due to the abundance of buffers), the clock tree is cheaper, and this explains those results. The reason lies in the fact that while the 2D-mesh has heterogeneous switch degrees, the 2-ary 4-mesh has all switches with exactly the same degree. Hence, the clock tree is inherently more balanced and thus easier to synthesize while meeting skew constraints.

By combining the performance results of Figure 3.27 with the power reports of Figure 3.28, the energy results of Figure 3.30 are achieved. Overall, the addition of repeaters to the links of a 2D-mesh is always an energy-efficient strategy. On the contrary, the 2-ary 4-mesh and the 2-ary 2-mesh show minor energy variations when moving from one technique to the other. This indicates that performance improvements have been achieved at the cost of a proportional power overhead. That is, despite a reduced execution time, the energy expended is the same.

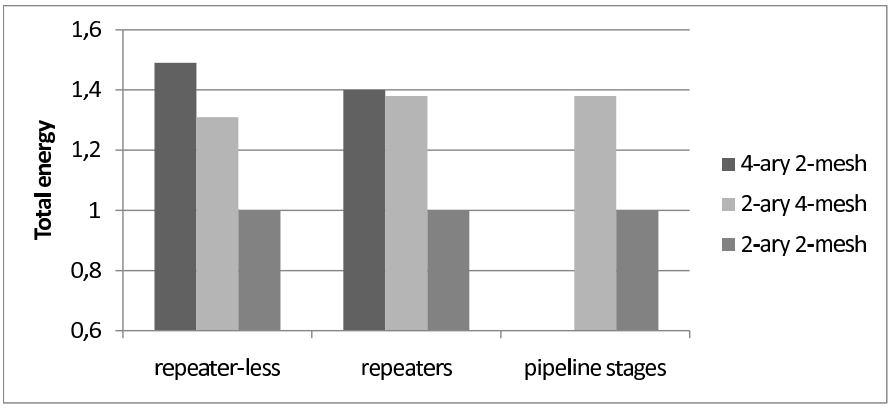


Figure 3.30: Normalized time, power and energy with link synthesis techniques.

3.6 Case study: 64-tiles topologies

The goal of this case study is to expand the physical design concepts presented in the previous one to a wider set of topologies. Unless otherwise noted, assumptions made in the previous case study still holds for this section. Specifically, this analysis concerns some of the large topologies for 64 cores presented in Section 3.2.3. This study aims at assessing the feasibility of the connectivity patterns of those topologies in the context of nanoscale silicon technologies. Like for 16-tiles topologies, the set of considered topologies for 64-tiles networks has been restricted for the need to keep the study focused. The criterion to select topologies for physical implementation was based on their suitability for a switch architecture without virtual channel. Introducing virtual channels is yet another degree of freedom which we prefer not to consider in this analysis for the sake of clarification, but it will be tackled in Chapter 4. So, the exclusion of a topology from the following analysis does not mean that the topology is not competitive. The k-ary n-cube (Section 3.2.2) and the k-ary n-rec (Section 3.2.2) family of topologies have issues associated with the implementation of routing algorithms, as both feature abundant cycles that must be broken in order to provide deadlock free communication. In the case of torus topologies with wormhole switching, the most efficient ways of providing deadlock free routing require the use of several virtual channels [52],

while the WK-Recursive topology requires the use of virtual channels plus some additional hardware in order to be deadlock free [126]. Anyway, there are still several generic routing strategies that allow to provide deadlock-free deterministic routing in both cases without the need to modify the NoC architecture [52], but they are quite inefficient. In the case of the torus topology, the turn model has been proposed to break cycles by prohibiting some turns in the network. Unfortunately, this design choice transforms the torus into an equivalent mesh, by breaking the cycles introduced by wrap-around links. Also, the Up/Down model, which transforms the topology into a spanning tree, has been proposed for its use in both topologies, but it incurs congestion issues in the root switches, greatly reducing the performance of the topology. For these reasons, k-ary n-cube and k-ary n-rec topologies have not been considered for the physical implementation analysis that follows. Also multi-stage interconnection networks have not been considered in this case study since an assessment of their implementation quality will be reporter in the next one.

3.6.1 Characterization methodology

The goal of this case study is to expand the physical design concepts presented so far to a wider set of topologies. Specifically, this analysis includes large topologies with 64 tiles, presented in Section 3.2.3. The main objective is to point out and evaluate the feasibility of such large scale networks for the on-chip domain. The reference switch architecture was used as the basic building block to construct the 64-tiles topologies under test. However, exploring the design space of topologies with such a large number of tiles by means of their actual physical synthesis proved impractical, due to synthesis time and memory capacity requirements. Therefore, a way to cut down on the number of physical synthesis tests while still characterizing quality metrics of the full topology with high accuracy was devised.

As reported in the previous case study, the performance bottleneck of a topology lies in its longest switch-to-switch link. Aware of this, for each topology under investigation, a sub-system composed of two communicating switches at the maximum possible distance in the topology was built, thus capturing the largest link delay in that topology floorplan. Logic synthesis for maximum performance and place&route were then performed to capture the

real switch-to-switch delay. Such delay (which is the critical path delay of the network too) is then used as the *target delay* to re-synthesize, and place&route all the possible switch-to-switch link distances in the topology under test. The reason for this is to accurately capture the switch cell area at a certain target speed. It is well known from logic synthesis theory that as the target speed is decreased, large area savings can be achieved. 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).

With this methodology, only few selected synthesis runs for each topology need to be performed in order to obtain its delay and area as a whole. It is assumed that enough routing channel space for regular routing of NoC links is available. Although this methodology allows to efficiently asses topology physical characteristics, the provided results are not 100% accurate, as the impact of wire delays for links that undergo bending in the actual layout is neglected. Anyway, the evaluated topologies present quite regular wiring patters that are somehow based on grid structures, thus diminishing the effects of this source of inaccuracy. Moreover, with this method it is also possible to capture the link buffering cost when link performance is boosted via repeater insertion. By leveraging the report of the utilized physical synthesis tool, the inferred repeaters along switch-to-switch links can be traced.

In order to provide area reports as accurate as possible, switch ports connecting to the network interfaces of attached tiles were left unconnected. This way, the tool is prevented from using large driving strengths for the gates on those ports. Finally, for those cases where switches feature ports connected to links of different lengths, the area of each switch output port is corrected by a coefficient that reflects the driving strength needed to drive the corresponding link. Such coefficients are experimentally derived beforehand by means of a dedicated set of tests.

Also, with this methodology it is possible to estimate the number of pipeline stages for each link required to speed up a topology. For this purpose, such pipeline stages are instantiated along the communication links thus breaking the switch-to-switch critical path. By incrementing the number of pipeline stages, the critical path can be brought back to the switch. How-

Topology	Degree	Post-synthesis critical delay	Post-p&r critical delay	Longest link
8-ary 2-mesh	6	0.93 ns	1.12 ns	1.5 mm
8-cmesh	6	0.93 ns	4 ns	6.75 mm
4-ary 3-mesh	8	1.05 ns	4.54 ns	6.9 mm
2-ary 6-mesh	8	1.05 ns	4.54 ns	6.9 mm
2-ary 5-mesh	9	1.23 ns	4.34 ns	6.96 mm
4-ary 2-mesh	12	1.38 ns	1.88 ns	3.0 mm
2-ary 4-mesh	12	1.38 ns	3.84 ns	6.4 mm
4-cmesh	12	1.38 ns	3.84 ns	6.4 mm
4-ary 3-flat	14	1.63 ns	8.33 ns	13.29 mm

Table 3.8: Topologies under test.

ever, in order to limit the area overhead, in the experiment that follows it was decided to bring back the critical path to the links of the first or second dimension of each topology, thus having a moderate impact on link area.

Next subsection starts by commenting physical synthesis results achieved for 64-tiles topologies without link pipelining. Later, the analysis is shifted to pipelined networks. Finally, physical synthesis results will be back-annotated into the simulation framework presented in Section 3.3, thus gaining layout awareness during the topology selection process.

3.6.2 Physical implementation

Table 3.8 shows the 64-tiles topologies considered for physical implementation. The maximum switch radix for the topologies under test ranges from a reasonable value of 6 to a large value of 14, which complicates the place&route of switches (see Section 3.5.2). Please notice that the tile architecture considered in this case study uses two port of the attached switch (as exposed in Section 3.3.4). Post-synthesis frequency results refer to the switches in isolation, reflecting the critical path delay of the switch with the highest degree in the topology.

After place&route, the effect of the link delay comes into play. Most of the topologies suffer from long switch-to-switch links. For the sake of the analysis, only the longest link length per topology is reported in the 5th column of

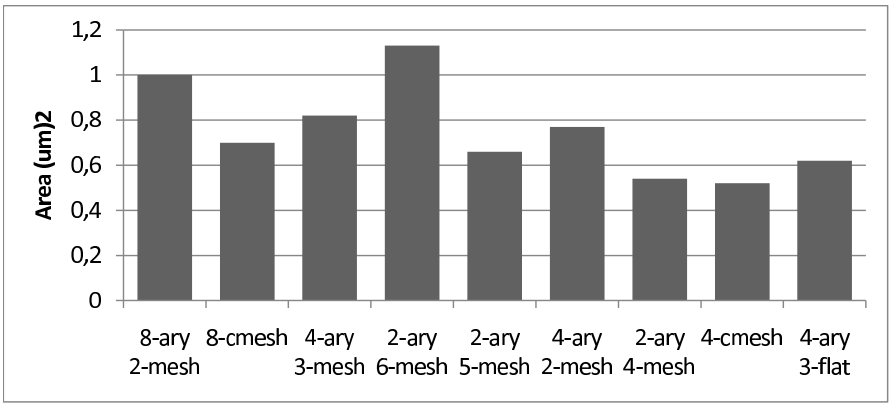


Figure 3.31: Normalized area for 64-tiles topologies.

Table 3.8. The comparison of such column with the 4th one, shows a clear correlation between the increasing link length and the decreasing operating speed of the topology under test. Only topologies with short links (e.g., 8-ary 2-mesh and 4-ary 2-mesh) can work at a reasonable frequency.

Figure 3.31 reports the area cost of the evaluated topologies. Those results are influenced by the combination of many parameters such as: number of switches in the topology, their degree and their operating frequency, and the number of link repeaters implemented. As an example, let us consider a very slow topology like the 2-ary 6-mesh. This topology features a larger area footprint than 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 of the 8-ary 2-mesh is 10% lower than the one in the 2-ary 6-mesh.

Another interesting result concerns the 4-ary 2-mesh. This topology has a relatively short worst case link (3mm), thus it does not suffer from a high speed degradation after place&route. As reported in Table 3.8, this topology is the only one (along with 8-ary 2-mesh) that has a final working speed above 500MHz. Interestingly, its area is 20% lower than in the 8-ary 2-mesh. This is due to its low switch count (16), and to the fact that its high switch degree (up to 12 ports) is compensated by its lower working speed than in the 8-ary -2-mesh. 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

Topology	Degree	Post-synthesis critical path	Post-place&route critical path
8-ary 2-mesh	6	0.93 ns	1.12 ns
8-cmesh	6	0.93 ns	1.12 ns
4-ary 3-mesh	8	1.05 ns	1.17 ns
2-ary 6-mesh	8	1.05 ns	1.17 ns
2-ary 5-mesh	9	1.23 ns	1.78 ns
4-ary 2-mesh	12	1.39 ns	1.88 ns
2-ary 4-mesh	12	1.39 ns	1.88 ns
4-cmesh	12	1.39 ns	1.88 ns

Topology	# of pipe-stages per dimension	# links	total pipe-stage area (um^2)	total switch area (um^2)
8-ary 2-mesh	0	112	0	2327712.8
8-cmesh	express link⇒4	128	193425.9	2752108.8
4-ary 3-mesh	dim.3⇒4	144	660216.3	3182953.2
2-ary 6-mesh	dim.3,4⇒1 dim.5,6⇒5	192	1087918.1	4362092.8
2-ary 5-mesh	dim.3⇒1 dim.4,5⇒3	80	293081.6	2758480.4
4-ary 2-mesh	0	24	0	1860718.3
2-ary 4-mesh	dim.3,4⇒3	32	125574.7	2328426.4
4-cmesh	express link⇒3	32	62787.4	2328426.4

Table 3.9: Post-place&route results of 64-tiles topologies with pipeline stage insertion.

final operating frequency. A natural way to tackle this problem is to implement link pipelining on such long links but the insertion policy has to be carefully considerd. In fact, the studied 64-tiles topologies feature a high number of long links that could quickly raise the area overhead to an unaffordable budget.

3.6.3 Pipeline stage insertion for 64-tiles systems

In order to cope with the high speed degradation of most topologies analyzed in this case study, pipeline stages need to be inserted in very long links. By adding pipeline stages it is possible to partially (if not completely) recover the initial operating frequency of the basic switch block. Of course, this comes

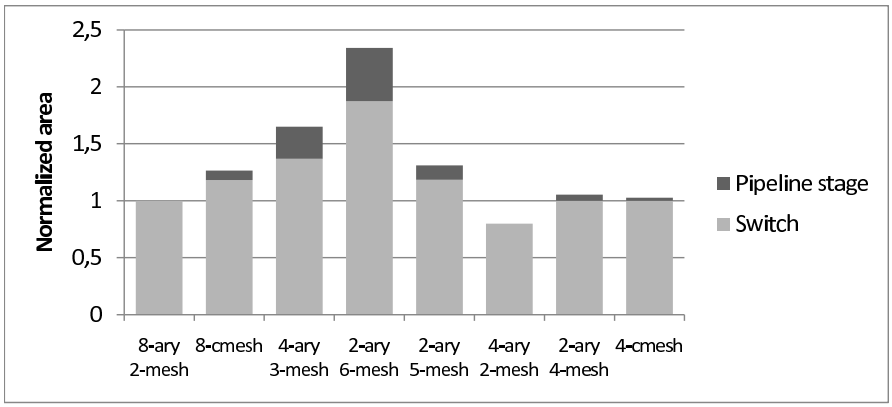


Figure 3.32: Normalized area for 64-cores topologies with pipeline stages.

with a high cost in terms of area and power. Furthermore, for some topologies with a high number of such long links, this technique may turn out to be unaffordable due to the extremely high amount of additional area required to implement such a strategy. 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 the 8-ary 2-mesh and the 4-ary 2-mesh have not been modified. Table 3.9 collects the results of this experiment. Please note that the 4-ary 3-flat topology (with 4 tiles per switch) has not been considered for the following analysis because its switch radix (14 ports) would be a major limiting factor for the speed of the whole topology and for its layout feasibility. A straightforward way to overcome such limitation would be to consider a pipelined switch architecture or/and even a full custom design, that fall outside the scope of this dissertation.

As clearly reported in the 3rd and 4th column (up), the insertion of pipeline stages is a very effective way to reduce post-place&route frequency degradation. Column 2 (down) reports the number of pipeline stages inferred in each link based on its dimension, whereas the 3rd column (down) points out the number of links of each topology. The area weight of link pipelining comes from the combination of these two factors and it is reported in the 4th column (down). Total cell area of the topologies along with the contribution of pipeline stages is reported in Figure 3.32. Please note that the number of pipeline stages per link depends on the maximum achievable frequency (dic-

tated by the maximum switch radix) along with the link length which is an intrinsic characteristic of each topology. As reported in Figure 3.32, the 2-ary 6-mesh is the most area greedy topology because it has the highest number of switches (64) which are placed&routed at the high frequency of 855MHz. Moreover, this topology features 192 links with up to 5 pipeline stages on the longest one. The main conclusion that can be extracted from those experiments is that each topology has to pay a different price to restore the working frequency allowed by the switching fabric.

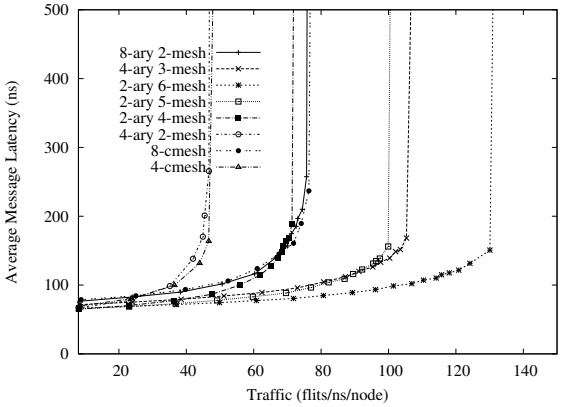
3.6.4 The performance prediction gap

This section re-evaluates the topology abstract quality metrics for 64-tiles topologies presented in Section 3.2.3 in light of the efficiency of their physical implementation, thus pointing out how misleading conclusions can be when the physical synthesis effects are ignored or underestimated.

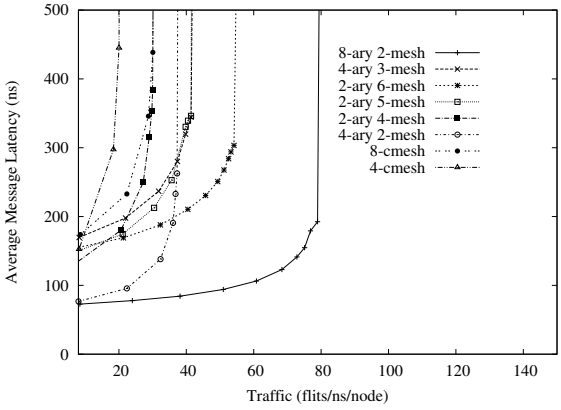
For these experiments, dimension-order routing (DOR) was considered. In order to remove the cycles introduced by the express links of k-ary n-cmesh topologies, express links will be used only to advance messages that travel along the perimeter of the topology a distance equal or higher than the switches that this express link overcomes, like in [23].

Results exposed in this section were obtained by feeding the proposed simulation framework with the high-level analysis assumptions and with the results of the physical characterization of the evaluated topologies. In these experiments, OCP generators were configured so destinations are generated based on several synthetic traffic patterns. The size of the generated OCP transactions is randomly selected with a minimum value of 4 burst beats, and a maximum of 16 burst beats.

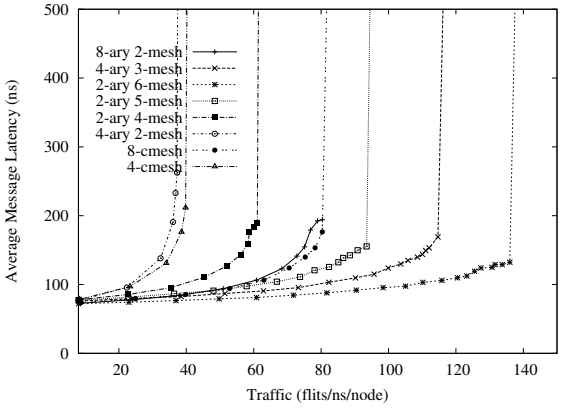
Figure 3.33(a) depicts accepted traffic vs. average message latency for a uniform distribution of message destinations for different topologies when considering high-level estimations. Obtained results reflect the conclusions drawn in Section 3.2.3. Figure 3.33(b) shows the same analysis where each topology works at the operating frequency reported in the previous section when pipeline stage insertion was not considered (see Table 3.8). By comparing Figure 3.33(a) against 3.33(b), there is a misleading gap between the performance predictions of the high-level analysis and the layout-aware one. In fact,



(a) Theoretical



(b) Layout-aware, no-pipelining



(c) Layout-aware, with pipelining

Figure 3.33: Performance of 64-tiles systems with uniform traffic.

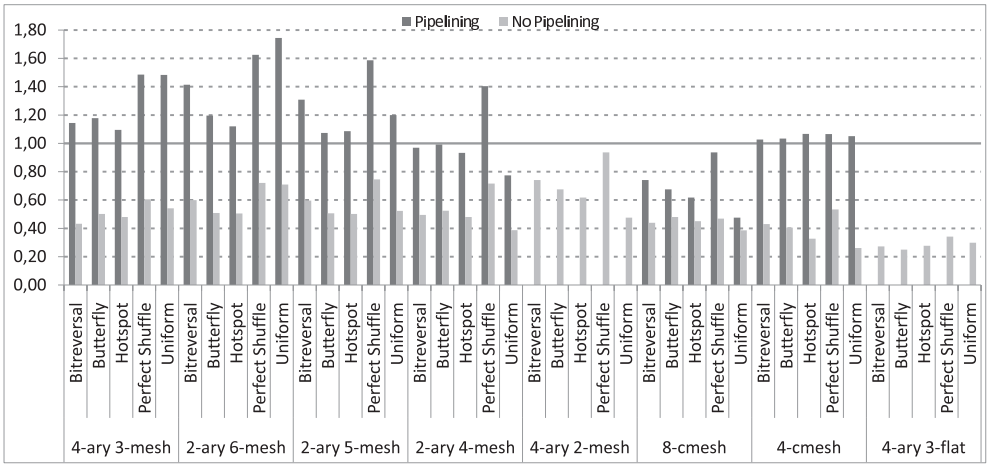


Figure 3.34: Normalized performance of 64-tiles systems.

while the theoretical results reported in 3.33(a) 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 3.33(b). It is obvious that there is a direct correlation between the operating frequency and the achieved 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, layout degradation 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 3.33(c) reports the same analysis results when each topology works at the operating frequency (see Table 3.9) enabled by the usage of link pipelining, and also accounting for the higher latency of pipelined links. 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. Similar curves have been drawn for several traffic patterns for each topology.

While the measured average latency did not vary too much among topologies, throughput results were more interesting. Those results are summarized in Figure 3.34. This figure shows the normalized maximum throughput of each topology with respect to the 8-ary 2-mesh solution. In this plot, a bar

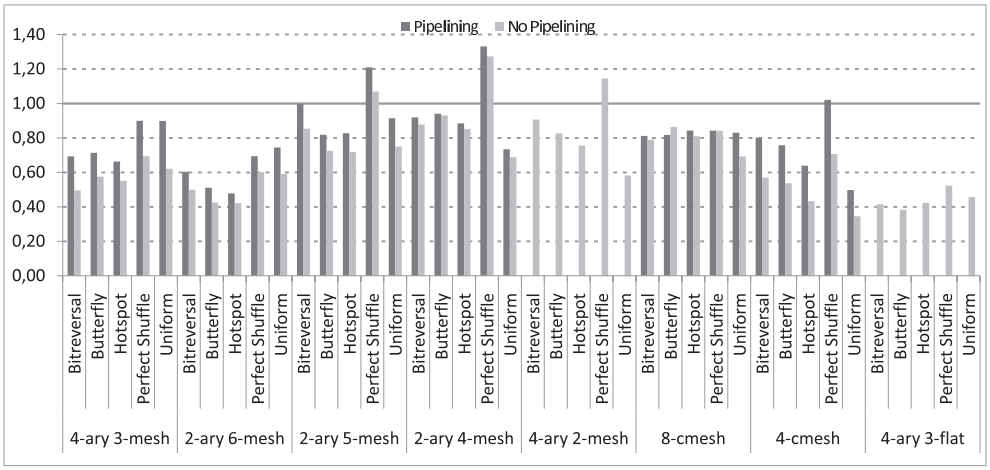


Figure 3.35: Normalized area efficiency of 64-tiles systems.

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 than 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 3.35, 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 idea is that the performance improvements achieved by complex topologies with pipelined links are not cost-effective. The only exception is when the traffic pattern favors topologies with a low hop count, as in the

case of the perfect shuffle traffic. This characteristic, along with the fact that some topologies feature a low area cost, leads to a higher area efficiency with respect to the 8-ary 2-mesh.

3.7 Case study: assessing Multi-stage Interconnection Networks

This case study analyses the layout feasibility and the effects of physical synthesis over performance figures of Multi-stage Interconnection Networks (MINs) in NoCs. MINs present a highly irregular wiring pattern, relatively large switch degree and significant area and power overheads when compared against 2D-mesh topologies, the *de facto* solution for NoC designs. This raises some skepticism about their practical feasibility. In spite of these concerns, those topologies still seems to be competitive mainly due to their promising performance scalability, which is a well known issue of 2D-mesh topologies.

Two families of MIN are analysed as an alternative to 2D-meshes. The first family are fat-trees (see Section 3.2.2), which promise high performance scalability at the cost of a high implementation cost. The second one is a unidirectional MIN, in particular a new topology family that aims at solving 2D-mesh scalability issues with an affordable implementation cost: Reduced Unidirectional Fat-Tree (RUFT). At first, the physical implementation of the candidate topologies is evaluated for designs of 16 and 64 cores. Then, the proposed simulation framework is used to estimate performance figures of the considered topologies by back-annotating implementation parameters (i.e., link latency and operating frequency).

3.7.1 Topologies under test

Table 3.10 shows the high-level properties for the studied topologies. For each topology, half of the cores are processor cores and half are memory cores. In this analysis, the k-ary n-mesh topology family (see Section 3.2.2) is represented by two 2D-mesh configurations which are considered as the baseline solutions: 4-ary 2-mesh for 16 cores configurations and 8-ary 2-mesh in the 64 cores category. In those topologies, the implemented routing algorithm is

Network size	**16 Cores**			
Topology	4-ary 2-mesh	2-ary 4-tree	2-ary 4-ruft	4-ary 2-ruft
Switches	16	32	32	8
Cores/Switch	1	0 or 2	0 or 2	0 or 4
Max. Degree	5	4	2	4
Unidir. links	48	96	48	16
Bis. Bandwidth	8	16	8	8
Hop count	7	7	4	2
Network size	**64 Cores**			
Topology	8-ary 2-mesh	2-ary 6-tree	2-ary 6-ruft	4-ary 3-ruft
Switches	64	192	192	48
Cores/Switch	1	0 or 2	0 or 2	0 or 4
Max. Degree	5	4	2	4
Unidir. links	224	640	320	128
Bis. Bandwidth	16	64	32	32
Hop count	15	11	6	3

Table 3.10: Network topologies under test.

Dimension Order Routing (DOR) (see Section 3.3.4).

Regarding fat-trees, a 2-ary 4-tree is analyzed for the 16 cores case, while the 64 cores one contemplates a 2-ary 6-tree. Notice that in both cases the degree of the switches is 4 ports, which allows to keep a competitive maximum operating frequency in the switches. Although solutions with less switches are possible (4-ary 2-tree and 4-ary 3-tree for 16 and 64 cores, respectively), in this case the degree of the switches would have been 8 ports. As demonstrated in the previous case studies, the operating frequency slowdown introduced by the use of this high degree switches would not be acceptable, thus overcoming the theoretical advantages of the k-ary n-tree topology in terms of performance over the 2D-mesh while conserving its increased implementation cost.

Routing in Fat-Trees is performed in two phases: ascending and descending. In the ascending phase, packets are forwarded upwards in the tree until

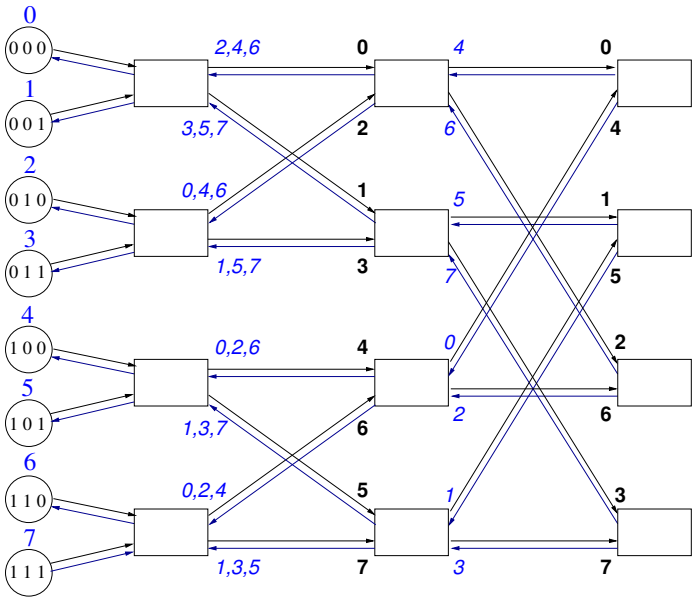


Figure 3.36: A 2-ary 3-tree topology.

one of the *nearest common ancestors*[1] between source core and destination core is reached. At this point, the descending phase is started. While the ascending phase present several minimal paths between any pair of cores, the descending phase provides a single path to destination, which depends of the common ancestor reached during the ascending phase. To provide a deterministic routing algorithm, as required by the reference architecture, a unique path must be selected for each origin-destination pair among all the possible ones in the ascending phase. To address this issue, the deterministic routing algorithm presented in [66] is used. In this algorithm, during the ascending phase, consecutive destinations are shuffled among the different ascending links of the switches, as in Figure 3.36: each ascending port is labeled in italics with the destination cores that are reachable through it. Also, Figure 3.36 shows in bold how destinations are distributed in the descending phase. As can be seen, each descending link is only used to reach a single destination, alleviating in this way the effect of the Head-of-Line (HoL) blocking effect explained in

[1] A nearest common ancestor between two cores is the switch of the lowest possible stage that allows the packet to reach the destination core by using only descending links.

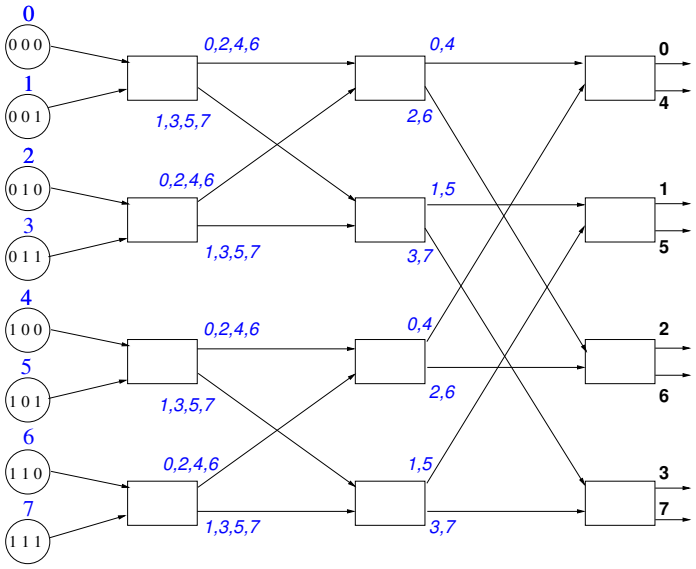
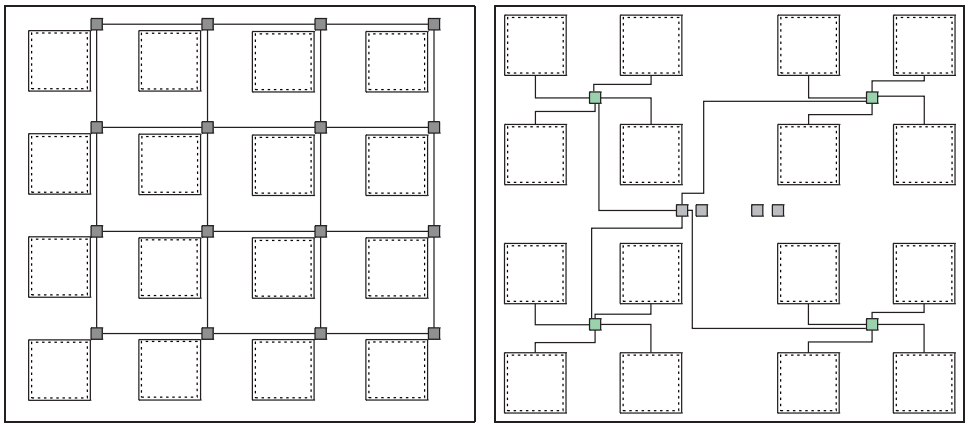


Figure 3.37: RUFT derived from a 2-ary 3-tree.

Chapter 2.

Finally, a new family of topologies is analyzed in this case study. *Reduced Unidirectional Fat-Tree (RUFT)* was first introduced in [127] as a cost performance trade-off. Like the butterfly topology introduced in Section 3.2.2, RUFT are part of the uMIN topology family, and present similar high-level properties. RUFTs are the result of the simplification of a k-ary n-tree when using the deterministic routing algorithm for fat-trees above exposed. When using this deterministic routing algorithm, if all packets are forced to reach the last stage of the network, 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. This is possible thanks to the routing algorithm that makes that each switch of the last stage only receives packets destined to as many destinations as output descending ports are in the switches; that is k destinations. For example, Figure 3.37 shows the RUFT topology derived from a 2-ary 3-tree. In the figure, each switch shows in italic the destinations reachable through its up ports. As can be seen, switches of the last stage only receive packets destined to two destinations. Although the use of long links may compromise the feasibility of this topology, all the



(a) 4-ary 2-mesh (b) 4-ary 2-ruft

Figure 3.38: Floorplans of topologies of 4-ary 2-mesh and 4-ary 2-ruft. Only the main wiring patterns are reported.

hardware resources related to the descending phase are reduced to these long links, simplifying the switch architecture. The resulting topology resembles a unidirectional butterfly, with a permutation of the reachable destinations from the last stage, which allows to keep the impact over HoL blocking of the original routing algorithm.

When evaluating RUFT, we consider two different topologies for each network size. The first ones are the unidirectional networks resulting from the simplification of the considered fat-tree configurations: a 2-ary 4-ruft in the 16 cores category, and a 2-ary 6-ruft 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 3.10, the switch degree is reduced to one half. This opens the way to consider an alternative RUFT implementation, trading switch degree (making it equal to that of the equivalent fat-tree) for the switch count. Thus, a 4-ary 2-ruft and a 4-ary 3-ruft were considered for 16 and 64 cores, respectively. In this way, the total number of switches is lower than that in the original fat-tree, as reported in Table 3.10.

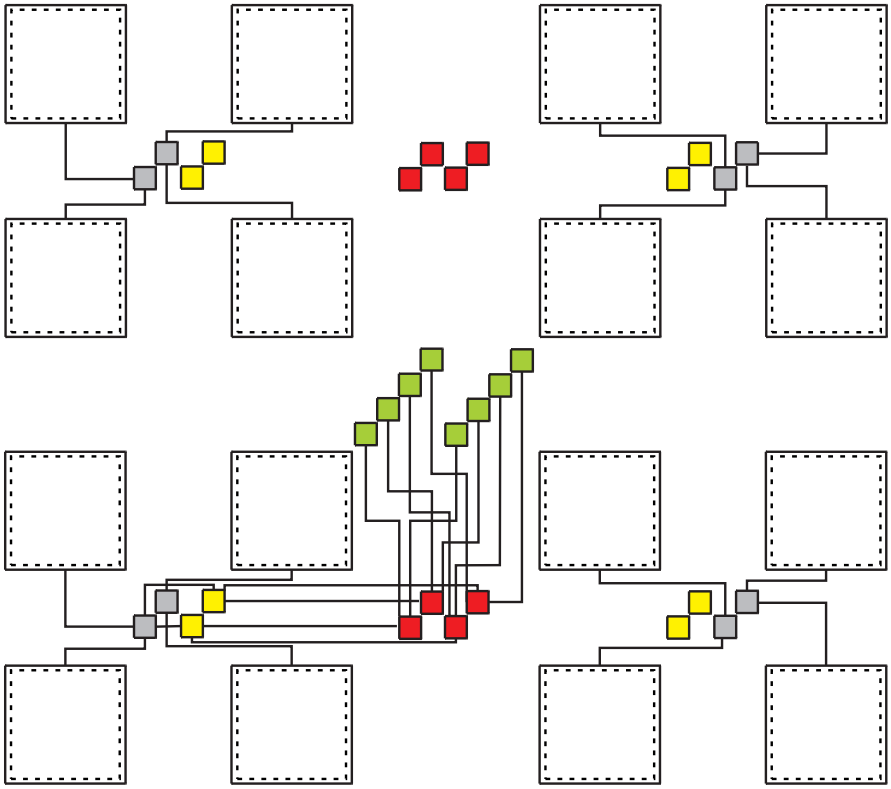


Figure 3.39: Floorplan of a 2-*ary* 4-*tree*. Only the main wiring patterns are reported.

3.7.2 Floorplanning

This section presents the floorplan directives assumed for each topology. Processor and memory cores size is assumed to be $1mm$ x $1mm$. To account for the worst case, over-the-cell routing was prevented for processing cores.

In the case of both 2D-mesh configurations (4-ary 2-mesh and 8-ary 2-mesh) the floorplan is straightforward due to its regular grid structure matching the 2D silicon surface. As an example, Figure 3.38(a) shows the mapping strategy of the 4-ary 2-mesh.

The floorplanning of the fat-tree solutions (2-ary 4-tree and 2-ary 6-tree) is more complex. Figure 3.39 shows the mapping strategy for the 2-ary 4-tree design. This topology consists of 4 switch stages with 8 switches each one. The floorplanning objective 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 stage are placed in the middle of each cluster (highlighted as grey switches in the figure). Each switch of the second stage is placed in the middle of the same cluster where the connected switches of the first stage are placed (yellow switches). The third switch stage is split into 2 subgroups and placed between the upper and lower clusters (red switches). Each subgroup serves its relative counterpart from the first and second stage. The last switch stage is located in the center of the chip (green switches). The presented layout exhibits equalized wirelengths between the second, the third and the last stage of switches. This floorplan can be easily scaled to the 64 cores configuration (2-ary 6-tree). In fact, the 64 cores fat-tree can be viewed as built up by four clusters of the 16 cores one, with two additional switch stages connecting them to each other. This new clusters can be placed as the cluster of nodes in the 16 cores case. The additional switch stages can be positioned as follows: the fifth stage is split and put in the upper and lower part in the middle of the chip, in a similar way of the third stage in the 16 cores case, while the last stage is in the chip center.

The 2-ary 4-ruft (Figure 3.40) is a novel unidirectional fat-tree which has never been laid out before, but it presents the same generic wiring characteristics as other uMIN topologies. In particular, switches belonging to the last stage are attached to the network interface of a core. This link is the intuitive weakpoint of the layout of uMINs. 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 (highlighted as green switches in the figure). 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 (grey switches). 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 stage are placed at the left and at the right of the chip center (red switches) . This also achieves an easy connection with the second stage, which is positioned in the center of the chip (yellow switches) . An interesting property of the presented floorplan is that the link length is kept almost constant

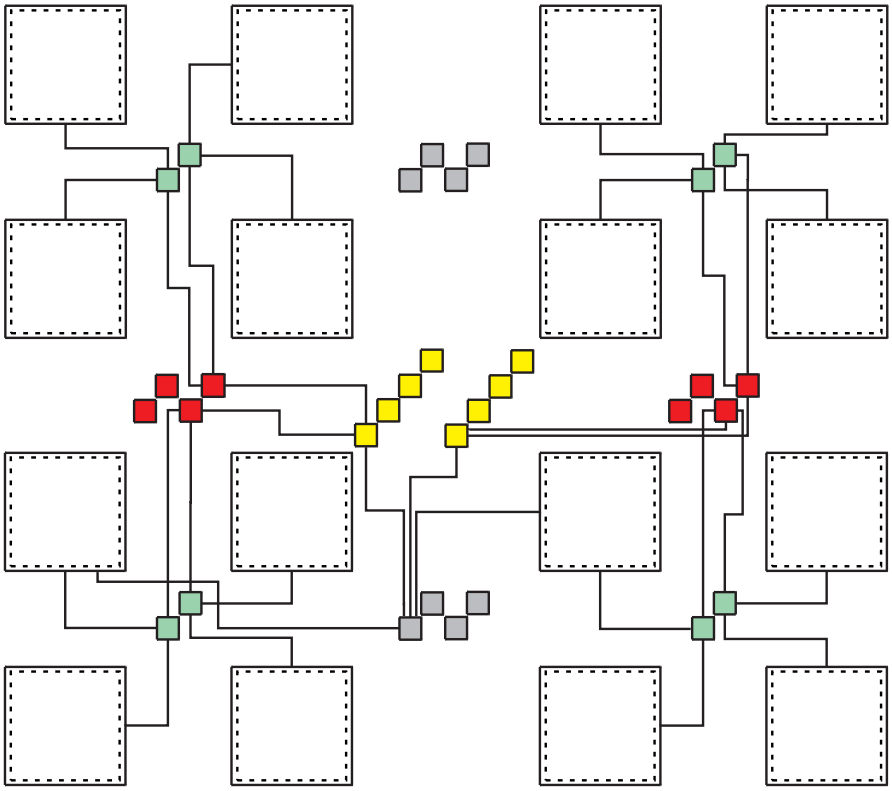


Figure 3.40: Floorplan of a 2-ary 4-ruft. Only the main wiring patterns are reported.

on a stage-by-stage basis. For the 2-ary 6-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-cores 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.

Finally, the floorplan for a 4-ary 2-ruft is illustrated in Figure 3.38(b). 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

Topology	Max. Degree	Post-p&r critical delay
4-ary 2-mesh	5	1.19 ns
2-ary 4-ruft	2	1.15 ns
2-ary 4-tree	4	1.29 ns
4-ary 2-ruft	4	2.1 ns

Table 3.11: Critical delay for 16-cores systems.

same floorplanning strategy of the 2-ary 4-ruft, a switch from the last stage has to be placed in the middle of a 4-core cluster (highlighted as green switches in the figure). Unfortunately, in this case the switch has to be interconnected also to all the switches of the first stage (grey switches), 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. Finally, the 4-ary 3-ruft floorplanning turns out to be as inefficient as that of the 2-ary 6-ruft topology, and results in very long links between the last stage of switches and the network interface of connected cores.

3.7.3 Physical evaluation

The physical evaluation framework presented in Section 3.6 was used to characterize the physical features of the topologies under test.

Timing

In all topologies, the network building blocks were synthesized for maximum performance. Link pipelining techniques were not employed for the 16-cores systems, as the area overhead is too high for such a small system scale. On the contrary, when scaling the system to 64 cores, the high wiring complexity of some topologies made necessary the use of link pipelining in order to mitigate the impact of link delay over system performance.

Table 3.11 shows the post-layout speeds achievable for 16-cores topologies. For all topologies (and even for the 2D-mesh) the critical path goes through the switch–to–switch links. The effect of the complexity of the connectivity pattern of each topology is obvious, as critical delays of the topologies are

differentiated by the longest link in that topology. The main advantage of the 2-ary 4-ruft is the high operating frequency of its switches, derived for their low degree. As can be seen, this advantage is overcame by its intricate wiring pattern, achieving a speed similar to that of the 2D-mesh. The 2-ary 4-tree also incurs some speed degradation, as its post-synthesis performance was lower than the 2D mesh even when the switch degree is one port lower. Finally, in the case of the 4-ary 2-ruft those effects are aggravated due to the longer wires that are needed, suffering a huge speed degradation after place&route, and becoming the slowest solution among all the configurations evaluated.

Regarding the 64 cores systems, the use of link pipelining is required in all MIN solutions. For this reason, those solutions were re-synthesized with a target frequency of 750 MHz (a delay of 1.33 ns) for the network, inserting pipeline stages in the links that were too long to achieve the target speed. This target delay is close to the maximum speed of the 2D-mesh. At this target speed, a high number of stages (11) were needed in the links connecting the last stage to the network interfaces of destination cores in the 2-ary 6-ruft. For the 2-ary 6-tree the latency of the links grows in the later stages. However, due to the better scalability of the fat-tree floorplan, the worst-case latency of the fat-tree is lower than in the 2-ary 6-ruft. The use of a lower target delay would require a higher amount of pipeline stages, that will likely results in a massive area overhead cost.

Overall, MINs typically suffer from a more significant performance degradation after place&route due to their more complex wiring pattern.

Area

Figure 3.41, reports the total floorplan cell area of the analyzed topologies, normalized to the 2D-mesh. In the 16 cores configurations, the 2D-mesh and the 2-ary 4-ruft exhibit almost the same area. This is due to the fact that both topologies have the same number of switch ports, which are the ones that mainly determine the switch area. The 2-ary 4-tree has a significant 60% area overhead with respect to the 4-ary 2-mesh, due to the increase in the number of switch ports. As expected, the 2-ary 4-ruft achieves a significant area saving with respect to the 2-ary 4-tree. Finally, the 4-ary 2-ruft presents the lowest area footprint with just 8 4×4 switches.

Figure 3.41: Normalized area of 16-cores systems.

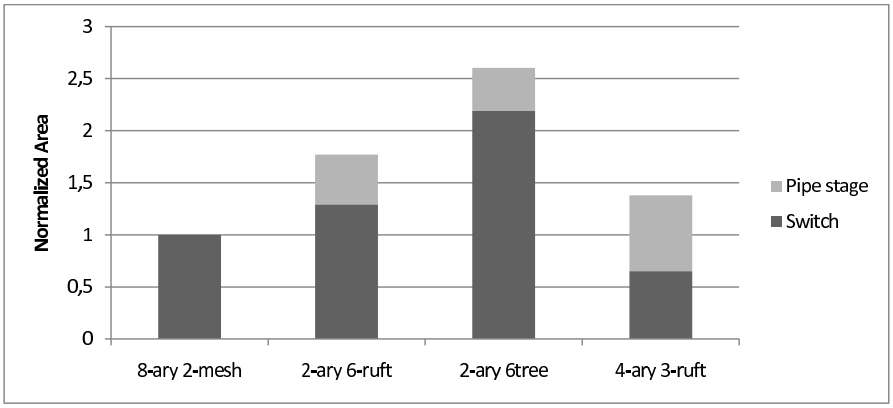
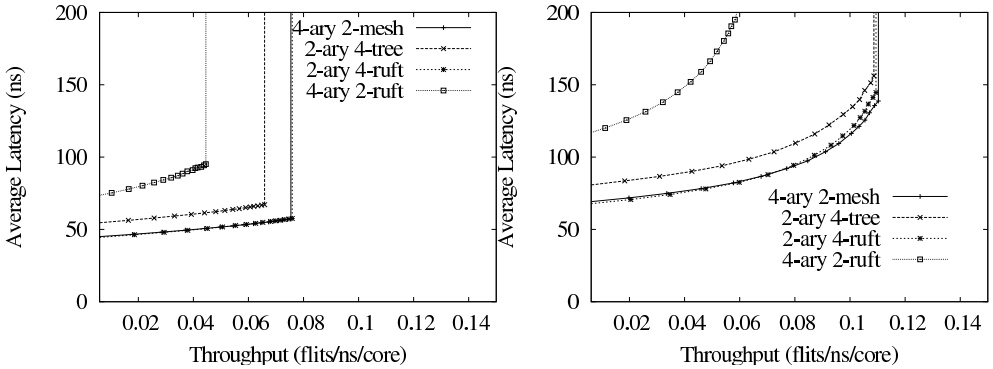


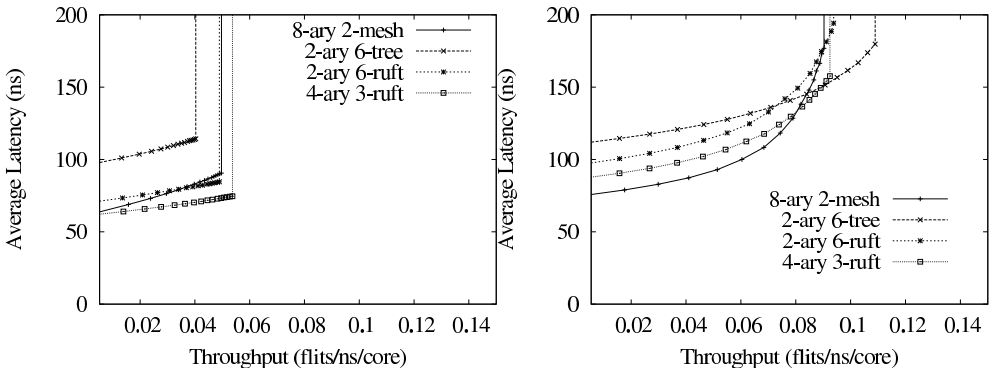
Figure 3.42: Normalized area of 64-cores systems.

The area projections for 64-cores systems are reported in Figure 3.42. As can be appreciated, the area of the MINs solutions has increased much more than that of the 2D-mesh, due to the higher number of switches required by MINs when interconnecting a large number of cores, as well as to the high number of pipeline stages required to provide competitive operating frequencies. This makes area footprint of the 2-ary 6-tree hardly affordable and that of 2-ary 6-ruft larger than the 2D-mesh. The 4-ary 3-tree is the more penalized topology by the insertion of repeater stages: more than 50% of its area. Please consider that this is a worst-case scenario pointing out scalability issues, which can be mitigated by custom floorplans for a given system size.



(a) 16 cores. OCP read transactions

(b) 16 cores. OCP write transactions



(c) 64 cores. OCP read transactions

(d) 64 cores. OCP write transactions

Figure 3.43: Performance of evaluated topologies.

3.7.4 Performance evaluation

The physical parameters reported above were annotated in the TL simulation framework. Performance figures were obtained with a synthetic uniform traffic pattern. In particular, OCP transaction destinations are randomly chosen among all the available memories in the system. Transaction sizes are randomly chosen with a minimum size of 4 data burst beats and a maximum size of 16 burst beats. Different OCP transactions allows to test different network characteristics. 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 reference architecture network interface,

while write transactions are not (i.e., they are posted).

Regarding 16 cores systems, Figure 3.43(a) and Figure 3.43(b) shows the average latency versus accepted throughput estimated for OCP read and write transactions, respectively. As can be observed, in both cases the 4-ary 2-ruft solution is unusable due to very low operating frequency. Despite the additional resources required by the 2-ary 4-tree, it shows a higher latency and an equal or lower throughput than the 4-ary 2-mesh solution. This latter presents a similar performance than the 2-ary 4-ruft, while requiring a similar area cost (see Figure 3.41) and displaying a lower wiring complexity. Thus, for a 16 cores system, the 2D-mesh (4-ary 2-mesh) might be the best option.

In the case of the 64 cores solutions, Figure 3.43(c) and Figure 3.43(d) show the performance projections for OCP read and write transactions, respectively. Since link pipelining is now used, the same frequency of 750 MHz is enforced and the proper latency is assumed for the links of the MINs. As can be seen, this case presents a neater performance differentiation between topologies. When considering read transactions, each transaction has to cross the network twice: one for the request to reach the target core and another for the response to reach the requesting core. In fact, due to the blocking behaviour of the network interfaces on read transactions (until the response packet arrives), the maximum number of outstanding transactions is one for each processor core. It follows that performance is driven by the diameter of each topology. Under these conditions, the 4-ary 3-ruft becomes the best topology, with the lowest average latency and the highest maximum throughput. On the contrary, the high diameter of the 2-ary 6-tree derived from its physical implementation makes this topology the worst choice in this case. But this conclusions may change when considering only write transactions. While the 2D-mesh solution is still the best overall solution (due to its low latency), for systems with a high throughput requirement, the 2-ary 6-tree becomes the best choice by a tight margin, although at an unaffordable area cost overhead: more than twice the area of the 2D-mesh.

3.8 Conclusions

This chapter introduces the layout-aware transaction-level simulation framework for NoCs developed in this thesis. Experiments on selected test-cases proved an excellent accuracy with respect to RTL simulation, while providing significant simulation time speed-ups.

This chapter also summarizes the first steps taken in the direction of developing DSE techniques. In particular, it aims at evaluating network topologies proposed in the open literature for NoCs. At first, a short summary of these topologies was presented, together with a comparative analysis of their high-level properties.

Later on, the developed simulation framework is used in order to demonstrate the risks of evaluating NoCs solutions based on high-level methodologies oblivious to effects of the physical layer over overall system performance. Then, two cases studies are presented with the aim of showing the potential of the developed simulation framework. In the first case study, the proposed simulation framework is used to perform a parametrical exploration of several mesh configurations. Following, several case studies were presented in which the proposed simulation framework is used to perform layout-aware topology explorations by back-annotating physical parameters. In the case in which the layout-awareness comes from real backend synthesis tools, although highly accurate, the cost in terms of synthesis was too high to support the quick topology explorations required for a DSE tool. For this reason, a slightly less accurate methodology that requires less synthesis iterations was devised.

Those case studies demonstrate that the conclusions drawn by a pure high-level analysis of topology performance based on high-level metrics may be highly misleading if not enlightened by the information provided by the physical synthesis. For example, a basic implementation of investigated topologies in which simple non-pipelining links are considered (similar to the ones commonly used in high-level analysis) leads to very poor performance due to long links forcing unacceptable critical path delays. A fact that is usually omitted in those high-level analysis. A layout-aware analysis shows that it is possible to improve the performance of those topologies, by alleviating the impact of long wires over the critical path with pipeline stage insertion. In spite of

a very competitive performance for such augmented topologies, the incurred area overhead (area of pipeline stages plus that of the switches re-synthesized to support the increased operating speed) is still too severe to make these solutions fully cost-effective with respect to a 2-D mesh. Therefore, they are a trade-off solution between communication performance and implementation cost.

A good example of a family of topologies with high theoretical performance scalability with implementation issues in the NoC environment are the Fat-trees. When their performance figures are enlightened with the physical characteristics of their implementation in NoCs, fat-trees can hardly materialize such scalability, requiring big amounts of additional resources in order to compensate the physical degradation of their complex wiring patterns. On the other hand, other topologies are able to materialize their theoretical advantages. That is the case of unidirectional MINs, like RUFT, the novel topology evaluated in this chapter. RUFT becomes attractive for its reduced power and area overhead. Unfortunately, their advantages cannot be easily materialized due to a more intricate layout design, an issue that is usually underestimated in high-level analysis.

Chapter 4

Virtual Channels for Networks On-Chip

> *"Don't get set into one form, adapt it and build your own, and let it grow, be like water. Empty your mind, be formless, shapeless like water. Now you put water in a cup, it becomes the cup; You put water into a bottle it becomes the bottle; You put it in a teapot it becomes the teapot. Now water can flow or it can crash. Be water, my friend."*
>
> Bruce Lee, A Warrior's Journey

This chapter discusses the adaptation of design techniques developed for interconnection networks to the on-chip environment. As it will be shown the same design must not be assumed for granted, instead they must be adapted. This chapter presents an example how the virtual channels flow control mechanism originally designed for interconnection networks can be adapted to the NoC environment in order to provide an efficient implementation. Section 4.1 introduces the key topics of this chapter. Then, the classic virtual channel switch fabric architecture as well as its adaptation to the reference architecture is explained in Section 4.2. After, two proposals of optimized implementations of virtual channels for NoCs are presented in Section 4.3, which are compared against the classical virtual channel architecture in Section 4.4. Finally, some conclusions are drawn in Section 4.5.

4.1 Introduction

This chapter delves in the dangers of adapting to NoCs design techniques developed from a different domain. While, Chapter 3 illustrates those risks in the case of network topologies, this chapter will further develop this topic by demonstrating the risks of blindly adopting design techniques with deeper implications in the system characteristics, like the network architecture. Due to the maturity of the research in the interconnection networks field and to the similitudes between this field and NoCs, designers may resort to this field in order to learn many handful design techniques. Although the advantages of this practice should not be neglected, this practice can be dangerous, due the differences between both environments. The physical implementation of on-chip designs with design techniques inherited from the off-chip domain may result in highly inefficient systems at best, and in unusable design at worst.

A good example of this practice in the open literature can be found in the virtual channel flow control. First proposed by [47] as an effective workaround for Head-of-Line (HoL) blocking, virtual channels are an appealing flow control technique for NoCs, in that they can potentially avoid deadlock and improve *link utilization* and *network throughput*. The original virtual channel switch architecture from the interconnection networks domain has been adopted for NoCs with minimal changes, being widely accepted in the open literature as the *de facto* virtual channel switch architecture.

However, their use in the resource-constrained multi-processor system-on-chip (MPSoC) domain is still controversial, due to their significant overhead in terms of area, power and cycle time degradation. Even considering that a virtual channel switch can implement the same amount of buffering resources of its VC-less counterpart by simply splitting a single buffer into multiple smaller virtual channels instead of a single queue [128], the incremental complexity when augmenting a baseline switching fabric with virtual channels is still severe [25, 41, 98]. Virtual channel overhead is associated with the additional levels of arbitration, the circuits keeping virtual channel state and the management of flow control on a virtual channel basis (instead of at a port based basis). Moreover, in order to avoid starvation between virtual channels, it is desirable to perform arbitration at flit-level instead of packet-level granu-

larity, at the cost of an increased switching activity. Finally, the delay of the additional control logic typically adds up to the critical path, thus resulting in a cycle time degradation that might offset the throughput improvements achieved by HoL blocking alleviation. Those issues make virtual channels the perfect candidate to demonstrate the advantages of adapting design techniques, instead of just adopting them.

Most research efforts documented in the open literature aiming at mitigating such an overhead by providing optimized virtual channel switch architectures fall in the domain of chip multi-processors (CMPs). Those researches fundamentally aim at delay-optimized implementations [84, 106, 118]. This is justified by the requirements of that domain, implying pipelined switches operating in the multi-GHz speed range, while shared memory traffic exerts stringent latency-throughput demands. In those cases, full-custom design techniques are typically adopted, thus making even single cycle implementations of the classical virtual channel switch architecture feasible while meeting the mentioned speed requirements.

In contrast, Multi-Processor Systems-on-Chip (MPSoCs) represent a much more resource-constrained domain. Area and power efficiency are main objectives to fulfill the requirements of embedded system platforms (e.g., multimedia, broadband or networking applications). Here, operating frequencies are typically lower, switches are not pipelined and the design flow is mainly synthesis-based [121]. In this domain, resource oversizing is clearly not affordable, therefore virtual channels are an attractive solution to *maximize link utilization*. Unfortunately, MPSoCs are most sensitive to area and power overheads, and this explains why their use is still controversial in this domain and an intensive effort is underway to minimize their implementation cost [97,160].

This chapter proposes a simple yet efficient approach to virtual channel implementation, which results in more area- and power-saving solutions than the classic virtual channel architecture. While the latter replicates only buffering resources for each physical link, the key idea behind this chapter is to replicate not just the buffering resources for each physical link but rather the entire VC-less switch as many times as the desired number of virtual channels. The resulting design is counterintuitively more area/power efficient while potentially operating at higher speeds. Those results are based on a well known

principle of logic synthesis, namely the area-performance trade-off for inferring the gate-level netlist of combinational logic. In practice, a design will be synthesized as a low-area netlist when delay constraints are loose, while high-performance designs can be materialized at the cost of area. In this way, we will prove that when a designer is aware of the distinctive features of NoC physical synthesis, highly optimized architectures can be conceived.

This proposal can be pushed to the limit by replicating not just the switch but even the entire network multiple times. Although this solution was first proposed by [23] and commercially exploited by [153], we feel that replicated networks are a fair alternative to our multi-switch approach only while keeping the aggregate switch buffering and flit width the same. The original proposal in [23] envisions a replication of sub-networks that results in an increase of link bandwidth: this solution addresses a different problem than the one investigated in this chapter, which instead aims at switch architectures that improve link bandwidth utilization with minimum area/power overhead.

4.2 Classical virtual channel design

This section will focus on the classical virtual channel switch architecture. At first, the conventional virtual channel switch architecture is introduced. Then, the reference switch architecture is augmented with virtual channels by following conventional design techniques.

4.2.1 Classical virtual channel switch architecture

The original virtual channel architecture was first introduced in [47], and an in depth explanation can be found in [45,118]. The following is a summary of this architecture pointing out its most relevant characteristics for this dissertation.

Figure 4.1 illustrates the wormhole virtual channel switch architecture usually employed in the open literature for NoC designs. As shown, the parameters affecting the delay of the various modules are: the number of I/O ports (P), and the number of virtual channels (V). Consider a packet traversing the switch of Figure 4.1 in isolation (i.e., no other packets are traversing the same switch). Initially, when the header flit is received at an input port its virtual

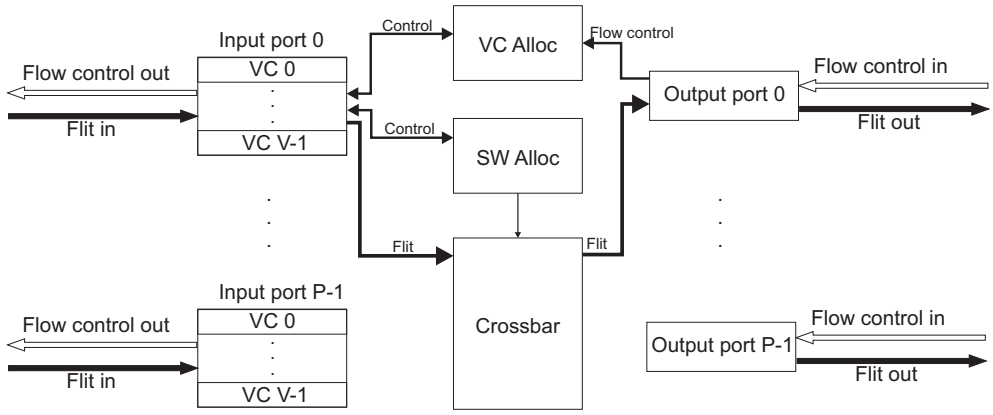


Figure 4.1: Schema of a conventional wormhole virtual channel switch architecture.

channel id field is decoded and the entire flit is buffered at the appropriate virtual channel.

In the next cycle, the header flit is decoded in order to obtain the list of output virtual channels usable by this packet. Then, a request for the desired output virtual channels is sent to the *virtual channel allocator*, which collects all the requests from each virtual channel of the input ports and returns available output virtual channels to successful requesters. Once the packet wins the arbitration of an output virtual channel, this virtual channel is assigned to it until the tail flit traverses the switch, thus virtual channel allocation is done in a packet–by–packet basis.

Next, a request for the required output port is sent to the *switch allocator*, that controls the access to the crossbar. In this case, arbitration is carried out in a flit–by–flit basis, in order to avoid starvation issues due to unfair distribution of the crossbar. Once the switch arbitration is won, a single flit is sent through the crossbar from the input virtual channel to the output port. If output ports are unbuffered, the flit will be directly sent through the output link. In the case when output ports are buffered, output buffers must also be structured as virtual channels, in order to accommodate flits in their corresponding virtual channel. Finally, when the tail flit of the packet crosses the crossbar, the switch allocator must notify the virtual channel allocator in order to free the resources reserved to this packet. Notice that both allocation

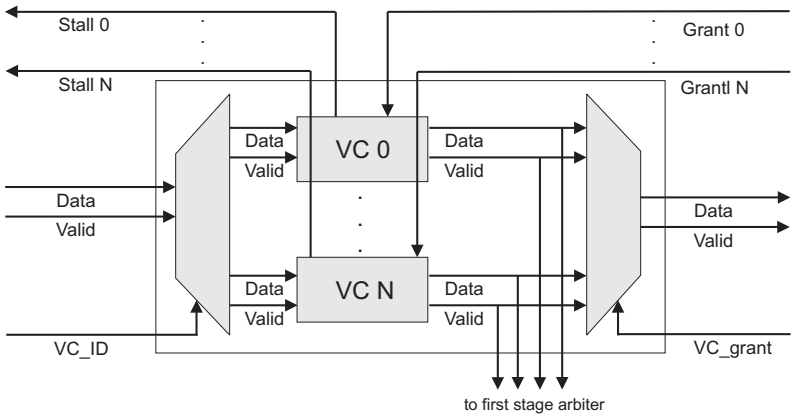


Figure 4.2: Input port schematic of the multi-stage virtual channel switch.

steps are carried out in a single cycle.

4.2.2 Conventional multi-stage virtual channel switch

In this section the reference architecture (denoted as VC-less from now on) is upgraded with virtual channel flow control by means of conventional design techniques. This new architecture will act as the reference to which this chapter proposals will be compared.

The reference architecture already presents some characteristics that help to reduce switch delay. First, the use of source based routing reduces the complexity of the switch logic, as routing logic is reduced to retrieve the desired output port of each packet from the header flit. Second, the use of deterministic routing reduces the complexity of the allocators, as requests will be constrained to a single request per packet. Finally, it is assumed that packets are delivered in the same order in which they are injected for a given origin and destination pair. The OCP models require in-order delivery of packets to avoid read–after–write errors. When in-order delivery is not enforced, it is possible that when processor tries to read a memory address that it has written in a previous transaction will instead get the unwritten memory address, as the read transaction may be delivered faster than the write one. This implies the use of statically allocated virtual channels, thus avoiding racing conditions between virtual channels.

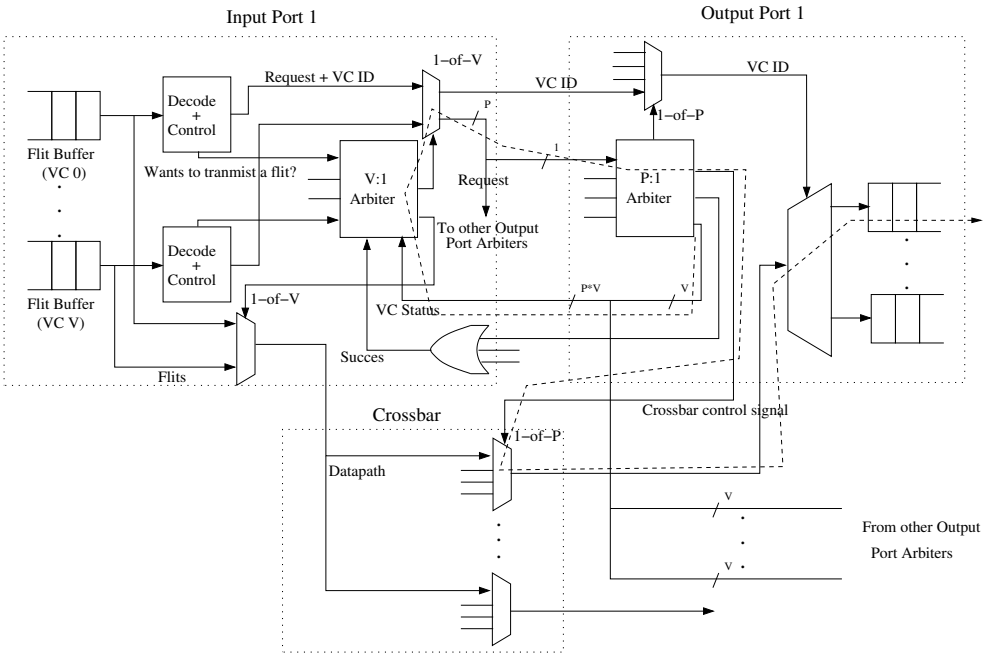


Figure 4.3: Detailed control path of the multi-stage virtual channel switch.

In *static allocation* of virtual channels, the virtual channel of a given packet is determined and fixed at injection time, and kept along the path of the packet through the network. Static virtual channel allocation is nonetheless of paramount importance whenever message-dependent deadlock has to be removed (different kinds of messages can be routed through distinct virtual channels) or in-order delivery of packets needs to be enforced. This latter is a critical abstraction for many higher-level protocols running on top of on-chip interconnection networks (e.g., file transfers, optimized cache coherence protocols) [53, 73]. The reference virtual channel switch architecture is significantly optimized for this scenario, and large effort was devoted to this optimization step. We wanted our new approach to virtual channels implementation to prove more area and power saving than the conventional one because of the effectiveness of the underlying design principle, and not of the inefficiency of the compared architecture.

A switch input port receives the virtual channel ID together with the flit from the upstream switch (Figure 4.2). This ID is used to select the virtual

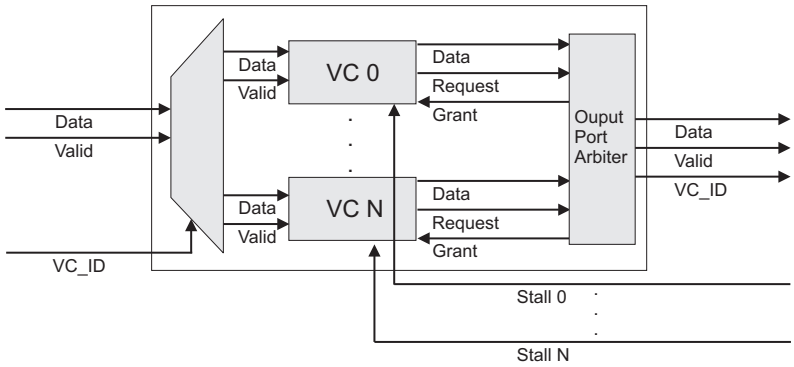


Figure 4.4: Output port schematic of the multi-stage virtual channel switch.

channel where arriving flits must be stored. Also, a stall signal is generated by each virtual channel and propagated upstream to the attached output port to notify availability of buffer space on a per-VC basis. Each virtual channel implements its own buffering space and a very simple decoding logic that reads the target output port for each packet from the routing field in the header.

In turn, each virtual channel sends head flits, that must be arbitrated, to the internal arbitration logic and all flits to the datapath crossbar going through a multiplexer. Flit space in the VC buffer remains reserved until a grant signal from the arbitration logic is received. When this occurs, the arbiter also drives the multiplexer control signals (see Figure 4.3-left).

Similarly to [84], switch allocation is performed immediately after the flit arrives, and the routing information is used to identify the intended switch output port. Virtual channels are assigned non-speculatively after switch allocation: the winning virtual channel that is granted access to a given output port automatically reserves the virtual channel with the same ID at that output port. This is because virtual channels are statically allocated. As we will clarify shortly hereafter, it can never occur that a virtual channel is granted access to an output port and the intended VC at that port is occupied.

Switch allocation implies a second stage of arbitration, like in [106, 118]. The detailed architecture of a switch with P I/O ports and V virtual channels per port is illustrated in Figure 4.3. We denote the resulting virtual channel switch as the *multi-stage architecture*, due to the two stages of the arbitration.

First, each input port arbitrates among its virtual channels through a $V : 1$

VC arbiter, selecting a single virtual channel out of V that is considered for arbitration in the next phase. Second, each of the P output ports makes use of a $P : 1$ *port arbiter* to discriminate among all the input virtual channels winning the first-stage arbitration and requiring that specific output port. All arbiters implement a round-robin policy and, differently than the VC-less switch, *arbitration is now performed at flit-level instead of packet-level* in order to avoid starvation.

One rule that is enforced during switch allocation is that a flit can only win the arbitration in the VC (first stage) arbiter if it is a payload flit or it is a header that requires an output virtual channel that has free buffer space to allocate the header and it is not in use by another input virtual channel. In practice, the first stage arbiter filters the requests for non-free output virtual channels. The port arbiters update at each cycle the status of each output virtual channel (see signal "VC Status" in Figure 4.3), and the first stage arbiters use that information to validate incoming requests from the input virtual channels. This way, it is not possible to waste a cycle by selecting a winner in switch allocation that will find its target virtual channel reserved or with no space. To provide fairness among all the input virtual channels, if the winner of the VC arbiter does not win the port (second stage) arbitration, it receives the highest priority in the virtual channel arbiter. We therefore make sure that the last winner will be proposed again as soon as possible.

This architecture limits the amount of virtual channels of each output port that can be allocated in a single cycle to 1. But, as the crossbar is only PxP, even if several output virtual channels of the same output port were assigned in the same cycle, only a single flit would be able to reach that output port at each cycle. So there is no performance penalty.

Figure 4.4 shows the schematic of the switch output port. Toward the switch side, together with the new flit to be stored, the port arbiter indicates the virtual channel that must store it. The output port implements a buffer for each virtual channel. At the interface to the downstream switch, there is a further arbiter that decides which virtual channel will send a flit based on a round robin policy. A virtual channel only activates its request signal if the corresponding stall signal of the downstream device is not asserted. When an output port sends a flit out, it has to send also the ID of the virtual channel

Architecture	Critical Path (ns)	Area (normalized)
VC-less Switch (MAX Perf.)	0.98	0.52
M.Stage VC Switch (MAX Perf.)	1.2	1
VC-less Switch (Relaxed)	1.2	0.43

Table 4.1: Overhead for VC support.

that is actually sending the flit.

The dashed line in Figure 4.3 illustrates the critical path of the multi-stage VC switch. Like the VC-less switch, it includes part of the control and of the datapath (essentially, arbitration and crossbar selection). However, *it is actually longer due to the more complex 2-stage arbitration* that was not there in the VC-less switch.

4.3 Bringing virtual channels to Networks On-Chip

This section presents the proposals for optimized virtual channel switch architectures for NoCs, aiming at networks with static allocation of virtual channels that implement deterministic source based routing algorithms.

4.3.1 Motivation

By inferring a 5x5 VC-less switch and a corresponding multi-stage switch architectures in the same 65nm technology library with Synopsys Physical Compiler, the critical path delay and area results shown in Table 4.1 were obtained. Two virtual channels were considered for the multi-stage architecture.

The conventional multi-stage realization of a virtual channel switch incurs a delay overhead of 20%, associated with the more complex arbitration. For the sake of fair area comparison, the VC-less switch was re-synthesized to match the same delay of the multi-stage one. This relaxation of delay constraints enabled the logic synthesis tool to infer the same logic functions with a more compact gate-level netlist, in practice moving the design point along the performance-area optimization curve. This is a well known principle of logic synthesis [100] and therefore holds in general, while the amount of achieved

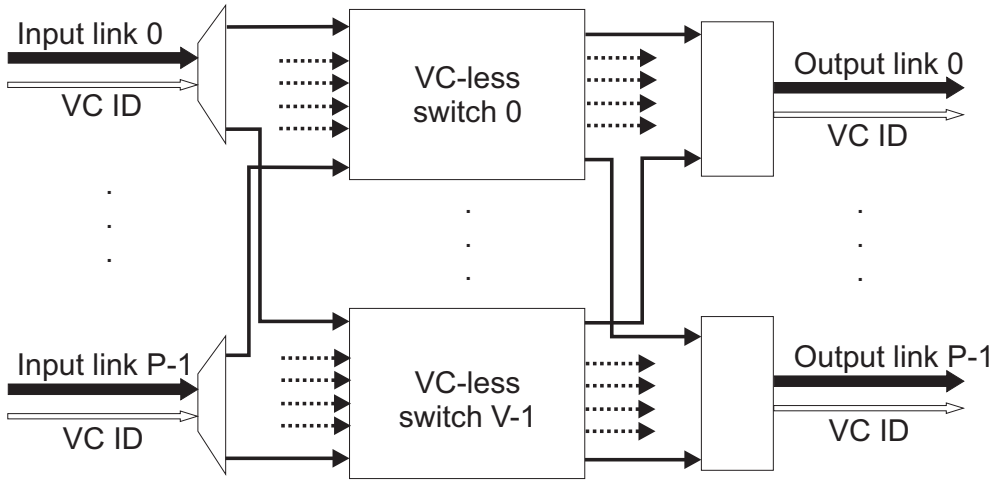


Figure 4.5: Multi-switch implementation of a virtual channel switch.

area savings depends on the specific design, on the set of cells available in the technology library and on the optimization techniques implemented by the synthesis tool at hand, resulting in different combinational logic implementations.

When focusing on the crossbar, Synopsys Physical Compiler tends to implement it under loose delay constraints by means of a tree of smaller multiplexers that combine more than two inputs at each level of the tree. As timing constraints are made tighter, some optimizations are used automatically by the tool that results in an increment of the area and power.

By looking at area numbers of the relaxed netlist, it can be observed that twice the area of a VC-less switch accounts for only 86% of the area of the multi-stage switch and that the relaxation resulted in almost 10% area savings with respect to the netlist synthesized for maximum performance. This result suggested the novel VC switch implementation which will be illustrated hereafter.

4.3.2 Proposed multi-switch implementation

Typically, a single buffer is associated with each physical channel. Virtual channels provide multiple buffers for each channel, so that when a certain virtual channel is congested, the packets in the other virtual channels can

still progress through the same physical channel and the network throughput can be significantly improved. However, this technique makes switch control logic more complex and the critical path delay is increased, as proved by the multi-stage architecture.

As an alternative, the virtual channel switch architecture that we propose consists of replicating not just buffers per channel, but rather the entire baseline *VC-less switch* as many times as the intended number of virtual channels. Replicated switches then share the same physical input and output links, similar to what conventional virtual channels do, but with the main difference that in the new implementation virtual channels have their own access to a replicated crossbar and the first stage of arbitration can be finally removed, as illustrated in Figure 4.5. We call this the *multi-switch virtual channel implementation*. The underlying principle is simple: instead of replicating buffering resources inside a switch, we rather propose to replicate the baseline VC-less switch without impacting its internal critical path.

Similar to the multi-stage architecture, also this solution requires an additional stage of *link arbitration* in order to multiplex the outputs of the baseline VC-less switches into the same physical output links connecting to downstream switches. As Figure 4.5 indicates, this stage is cascaded to the replicated VC-less switches and it is exactly the same which is used at the output of the multi-stage implementation (see *output port arbiter* in Figure 4.4). Like this latter, it arbitrates on a flit–by–flit basis while the arbiters of the replicated switches keep arbitrating at the packet level.

Interestingly, delay of this arbitration stage does not add up to that of the VC-less switches to determine the critical path, since they are separated by a retiming stage (the switch output buffers). In practice, *the critical path of the multi-switch architecture is the same of a VC-less switch*, since it does not make use of a multi-stage arbiter. However, one might argue that this comes at the cost of replicating more physical resources (e.g., the crossbars).

At this point, a basic principle of logic synthesis comes into play and leads to opposite conclusions. When comparing the multi-stage with the multi-switch virtual channel implementations, this latter has less functions on the critical path hence potentially resulting in a more area/power-efficient gate-level netlist after logic synthesis. In fact, the multi-switch architecture cer-

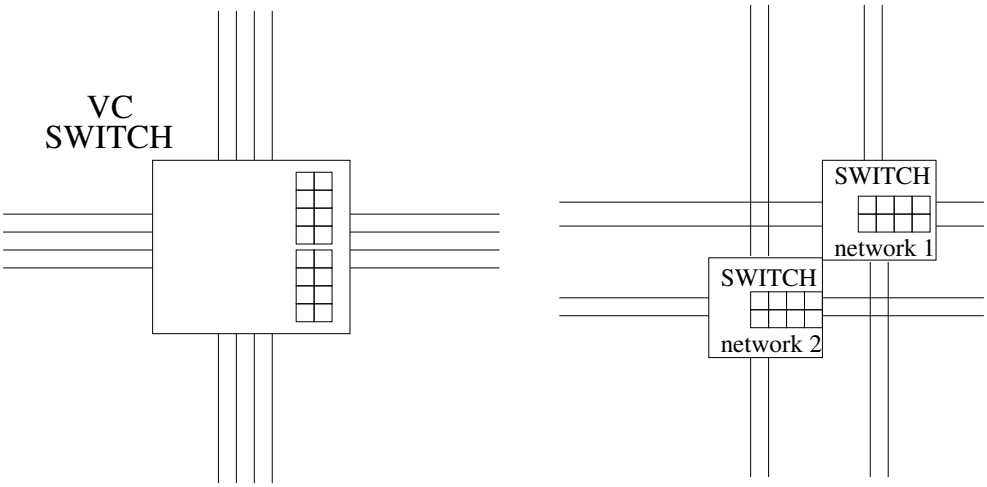


Figure 4.6: Multi-stage virtual channel switch vs multi-network compound switch: aggregate link width and switch buffering are kept constant.

tainly provides a higher maximum speed than the multi-stage one. However, if we require the two architectures to be aligned to the speed of the slowest one (the multi-stage), then combinational logic of the multi-switch design can be inferred with relaxed delay constraints and therefore optimized for area and power. In practice, a different design point along the area-performance optimization curve is inferred.

Now, the issue is to determine whenever the area savings achieved by logic synthesis are enough to compensate for the larger amount of hardware resources that are instantiated in the multi-switch architecture, especially the replicated crossbars. Please observe that the multi-stage and the multi-switch architectures can be designed to instantiate the same overall amount of buffering resources: V virtual queues in the multi-stage switch are equivalent to a single queue in V replicated switches.

4.3.3 Full network replication

By pushing the above mentioned strategy, the entire physical network should be replicated multiple times instead of just the switches. This solution has already been adopted, e.g., in [143, 153]. In [143] four separate and independent NoCs are used, while [153] uses 5 mesh networks and leverages the on-chip

wiring resources to provide massive on-chip communication bandwidth. In this dissertation, a different perspective is considered: the available link bandwidth is satisfactory and we search for cost-effective design techniques for its optimal exploitation. As a consequence, for the sake of comparison we consider replicating multiple physical networks *while keeping the aggregate link width and aggregate switch I/O port buffering* the same. In practice, in order to construct the *multi-network* architecture, we equally partition the same link bandwidth of the multi-stage/switch NoC across the multiple physical networks (see Figure 4.6).

Therefore, aggregate switch buffering in all architectures is the same. In the multi-stage solution, V virtual queues are instantiated in the same switch for each input and output port. In the multi-switch solution, V replicated switches have a single queue per port. In the multi-network architecture, switches in each network have a flit width V times lower and single queues with a size (in slots) V times larger. This way, the overall amount of flip flops for buffering purposes stays the same throughout all the architectures under test.

One clear disadvantage of the multi-network architecture lies in the fact that due to the smaller flit width of its switching sub-modules, packet latency increases. In particular, with V virtual channels in the multi-stage architecture, in each of the V replicated networks packets suffer from a latency increase which is more than V times. This is due to the *control bits* in the network protocol. In particular, in the reference architecture 3 bits in each flit are devoted to the *flit type*, which in the simplest case can be either head, payload or tail. Such constant size flit type consumes a percentage of the flit width which grows as the flit width becomes smaller. In practice, as the number of replicated networks increases a lot to mimic a large number of virtual channels, the packet latency grows unacceptably.

4.4 Architecture comparison

This section will provide an in depth comparison of the reference architecture upgraded with virtual channel by the classical approach (multi-stage implementation) and both proposals (multi-switch and multi-network). At first, the

	Multi Stage	Multi Switch	Multi Network
2 virtual channels			
Number of sub-modules	–	2 switches	2 networks
Flit width per sub-module (bits)	32	32	16
Aggregated link width (bits)	32	32	32
Input queues per Sub-module	2	1	1
Output queues per Sub-module	2	1	1
Input queue slots (flits)	2	2	4
Output queue slots (flits)	6	6	12
Aggregated switch buffering (bits)	1280	1280	1280
4 virtual channels			
Number of sub-modules	–	4 switches	4 networks
Flit width per sub-module (bits)	32	32	8
Aggregated link width (bits)	32	32	32
Input queues per Sub-module	4	1	1
Output queues per Sub-module	4	1	1
Input queue slots (flits)	2	2	8
Output queue slots (flits)	6	6	24
Aggregated switch buffering (bits)	2560	2560	2560

Table 4.2: Configurations for the compound switch architectures for 32 bits.

physical synthesis of all three implementations will be evaluated. Then, an accurate performance comparison among all three architectures is presented.

4.4.1 Physical synthesis

All three architectures were synthesized by means of the backend synthesis flow employed in Chapter 3, for a 65nm STMicroelectronics technology. In all experiments a switch degree of 5 was considered. Each experiment is defined by two factors: the number of virtual channels and the flit/link width. In particular, experiments were performed for configurations with 2 and 4 virtual channels, and with 32 and 64 bits of aggregated link width. Table 4.2 and Table 4.3 show the configurations for the 32 and 64 bits experiments, respectively.

	Multi Stage	Multi Switch	Multi Network
2 virtual channels			
Number of sub-modules	–	2 switches	2 networks
Flit width per sub-module (bits)	64	64	32
Aggregated link width (bits)	64	64	64
Input queues per Sub-module	2	1	1
Output queues per Sub-module	2	1	1
Input queue slots (flits)	2	2	4
Output queue slots (flits)	6	6	12
Aggregated switch buffering (bits)	2560	2560	2560
4 virtual channels			
Number of sub-modules	–	4 switches	4 networks
Flit width per sub-module (bits)	64	64	16
Aggregated link width (bits)	64	64	64
Input queues per Sub-module	4	1	1
Output queues per Sub-module	4	1	1
Input queue slots (flits)	2	2	8
Output queue slots (flits)	6	6	24
Aggregated switch buffering (bits)	5120	5120	5120

Table 4.3: Configurations for the compound switch architectures under test for 64 bits.

As can be observed, at any given experiment aggregated input/output link width and aggregated switch I/O port buffering is the same for all compound switches.

Initially, each configuration was synthesized at its maximum performance. Next, the delay constraint was gradually relaxed in order to get area/critical path curves. Figure 4.7 and Figure 4.8 illustrate such curves for the 2 and 4 virtual channels experiments, respectively. By looking at the figures, several observations can be made.

First, the multi-switch architecture can achieve a higher speed than the multi-stage one, since it implements less control functions on the critical path.

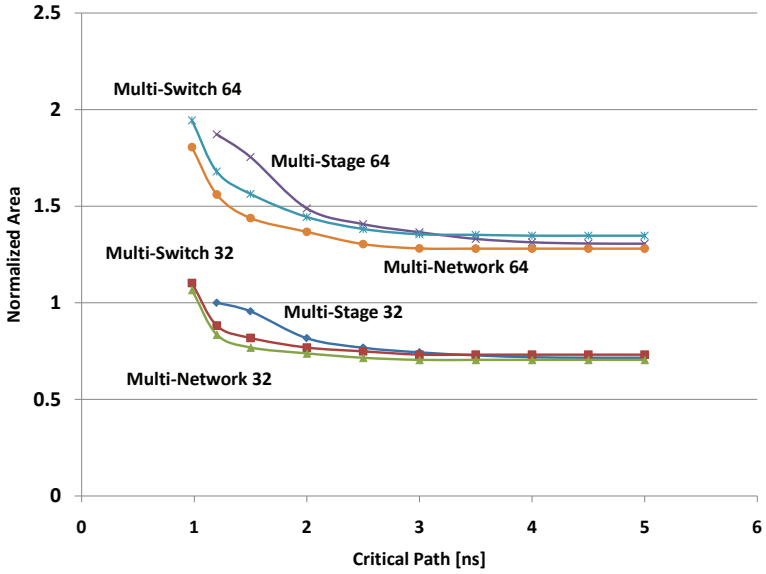


Figure 4.7: Area scalability as a function of the target delay constraint. 2 virtual channels.

Therefore, the above mentioned principle of physical synthesis can be exploited in order to reduce area of this design while relaxing its performance constraint. Therefore, it is possible to match the same maximum speed of the multi-stage architecture while incurring a lower area, since the area scalability process for the internal combinational logic (e.g., the crossbar) is very effective. Area savings in almost all cases amount to 10%.

Second, when operating at a lower speed, both gate-level netlists can be optimized for the relaxed timing constraint and therefore save area. This optimization process saturates around a cycle time of 3ns for the multi-switch solution, while the multi-stage can still be optimized until 4.5ns. As a consequence, there is a target cycle time (3ns) beyond which the multi-stage architecture actually saves area. Apart from the unrealistically low operating speed at the break-even point, the area savings from there on are marginal (maximum of 5% at 200 MHz for 4 virtual channels).

Third, while it is true that replicating a 64-bit mux-based crossbar is in principle more expensive, it has to be considered that the multi-stage architecture employs additional mux-based logic: one to select one virtual channel per

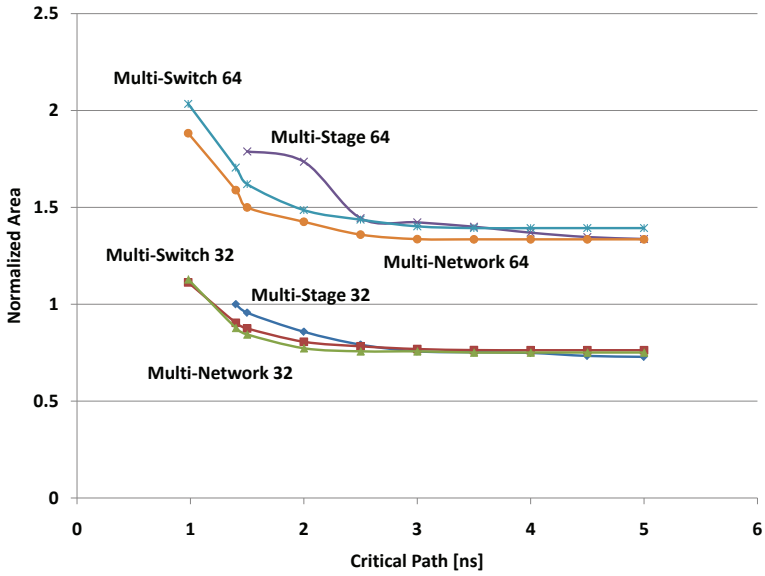


Figure 4.8: Area scalability as a function of the target delay constraint. 4 virtual channels.

input port and one demux to send the crossbar output to one output virtual channel. In practice, the multi-stage solution has a larger overall mux-based logic. Therefore, the results are again in favour of the multi-switch solution (area-wise) even at 64 bits.

Finally, regarding multi-network, this architecture scales in the same way as the multi-switch one. However, area is always consistently better. This is due to the absence of de-mux logic at switch inputs, of the mux logic at switch output and of the link arbiter controlling this latter. Moreover, this architecture makes use of multiple smaller crossbars with respect to the multi-switch solution. On average, area savings amount to around 5%. Even at very low speeds, this solution is still very competitive area-wise with the multi-stage architecture.

Let us now better detail how the area optimization process operates in the multi-switch architecture (2nd and 3rd bars in Figure 4.9). When timing constraints are relaxed, area of the non-combinational circuits remains almost unchanged. In contrast, gate-level netlist transformations during logic synthesis enable a significant reduction of combinational logic area (crossbar,

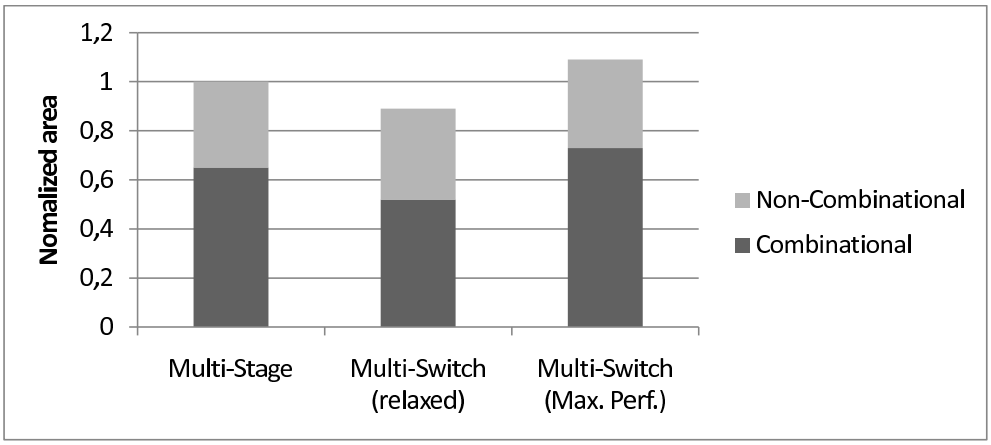


Figure 4.9: Area breakdown of 32-bit multi-switch (maximum and relaxed performance) and multi-stage implementations.

arbiters, multiplexers, buffer control logic). For the crossbar, what actually happens when a lower delay is required is that driving strength of gates is largely increased, and complex logic cells (like 4-input multiplexers) are decomposed into simpler and individually tunable logic gates. When comparing 1st and 2nd bar in Figure 4.9, it is clear that the capability to relax performance and optimize area of the multi-switch gate-level netlist enables a higher area efficiency.

By comparing Figure 4.7 and Figure 4.8, it can be observed that for multi-switch implementations, the maximum performance is always the same regardless the number of virtual channels, while for the multi-stage case the addition of one virtual channel incurs an 8% degradation of the maximum speed (around 16% for 2 virtual channels and so on). The multi-switch architecture thus avoids the well-known degradation of maximum achievable speed with number of virtual channels, since it keeps adding resources in parallel without impacting the critical path. This holds until the critical path moves from the switch internal logic to the switch-to-switch link (as explained in Chapter 3). At that point, multi-stage and multi-switch architectures would feature the same degradation since they have the same link architecture (including the same link arbitration logic).

Regarding power-analysis, Figure 4.10 shows the resutls of all switch im-

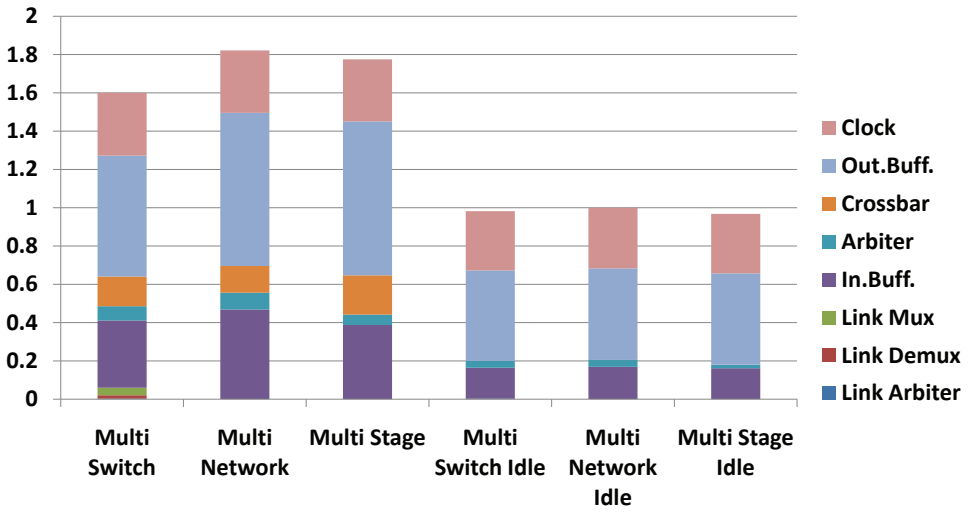


Figure 4.10: Power analysis with 50% switching activity and with idleness.

plementations after place&route, for the 2 virtual channels with 32 bits of flit/link width experiment. All architectures are operated at the speed of the multi-stage one (700 MHz).

Idle power is typically determined by registers and by the clock tree. Since all architectures have been designed with the same amount of aggregate buffering, their idle power is almost the same. Considering that a low-power, especially low-leakage industrial technology library at 25 degrees was used, leakage contribution was not significant in this results.

By experimenting with an average switching activity at switch inputs of 50% (target output ports and virtual channel IDs randomly chosen), it is possible to notice a large power consumption increment in the combinational logic blocks (crossbar, arbiter). One thing to notice is that the multi-switch total crossbar power is lower than the multi-stage one. This is due to two reasons. First, the multi-stage switch was synthesized at maximum performance while

	Multi-Switch/ Multi-Stage	Multi-Network 2 VCs	Multi-Network 4 VCs
32 bits	23 flits	36 flits	95 flits
64 bits	23 flits	34 flits	69 flits

Table 4.4: Packet length for a burst of 10 32-bit words. Aggregate link width in the NoC: 32 bits.

the multi-switch was synthesized with relaxed delay constraints. Second, the link injection rate typically constrains the crossbar transmission rate. Despite having replicated crossbars, multi-switch does not have replicated links. Usually there is only one flit that can cross the crossbars per input port at a given cycle, due to the link injection rate of one flit per cycle. Even though, there are some cases in which it is possible to have several flits that can cross both crossbars at the same cycle. For example, an input port with 2 virtual channels may have one virtual channel recovering from a stall condition, while the other will be transmitting a packet. In this case, both crossbars will be in use until the buffers are depleted, after that the link injection rate will be constraining the crossbar transmission rate to 1 flit per cycle. If we also consider that input and output buffers in multi-stage are more complex than the sum of the input and output buffers in the two VC-less switches, and that some arbiters in the multi-switch architecture arbitrate on a packet- rather than flit-basis, then it is possible to understand why multi-switch saves about 18% power with respect to multi-stage.

Regarding multi-network, its power consumption is 21% higher than the multi-switch one. Packets in multi-network are in fact longer than in multi-switch (higher number of flits for the same amount of real data). So, for a given set of transactions, multi-network will always send a higher number of flits, which translates into a higher switching activity. Thus, it features a higher power consumption.

4.4.2 Performance comparison

Simulation framework setup

The simulation framework presented in Chapter 3 was upgraded with accurate models of the evaluated virtual channel implementations. Then, physical synthesis results were backannotated into the simulation framework, in order to provide accurate performance estimations. In those experiments, it is assumed that all the networks operate at the speed of the slowest one (multi-stage).

As mentioned, to provide a fair comparison between the considered architectures, the aggregated switch buffering and link bandwidth of all three solutions remain constant. Regarding the simulator framework, this decision affects the configuration of the multi-network solution. The buffering flit slots of every switch and network interface are increased over the other solutions as many times as the number of virtual channels to account for the lower flit width. Therefore, in multi-network, packet length for a single transaction increases with the number of virtual channels. For this reason, all performed simulations do not keep constant the packet length but rather the length of the processor transactions, which range from 4 to 16-beat bursts.

Each data word of the burst consists of the actual 32 bits of data plus 4 byte enable control bits (64 bits of data plus 8 byte enable control bits for 64 bits architectures). Table 4.4 shows, for each switch architecture, the packet lengths corresponding to a processor burst transaction of 10 words, for a NoC aggregate link width of 32 and 64 bits. As can be observed, the packet length in the case of multi-switch (or multi-stage) with 32 bits is 2 flits (3 flits for the header and 20 for the payload). Notice that each processor data word is translated into two flits. In the reference architecture, the flit generation of consecutive words of data is decoupled from each other by guaranteeing that a flit contains only bits of a single data word. In this way, the complexity (and therefore the delay) of the translation processes is minimized. However, this implies the use of padding when necessary, since the second flit decoding the data word might be partially empty. In fact, as the link bandwidth grows up, a higher amount of padding is required.

In contrast, by reducing the flit width, the removal of padding bits allows the packet length to increase less than linearly. For example, in Table 4.4 the

multi-network solution with 4VCs for 32 bits requires 95 flits. This is less than 4 times more than the equivalent multi-switch solution, despite having a flit width four times smaller (the flit width is 8 bits).

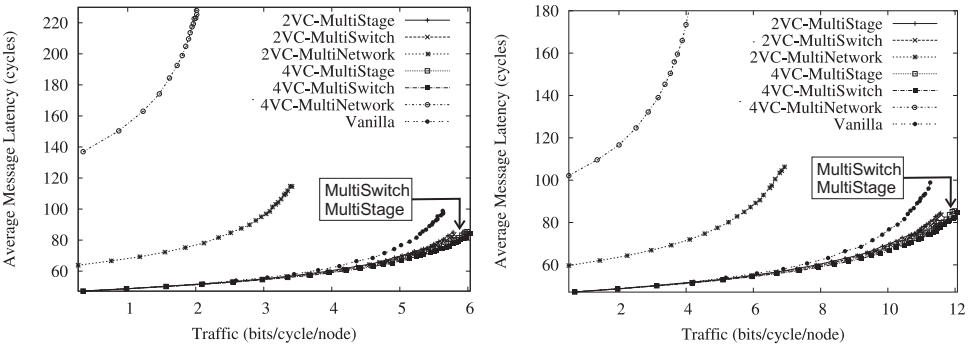
For multi-stage and multi-switch, the ideal link width to accommodate 32 bit processor data words into the lower number of flits and limit padding would be 39 bits: 36 bits for data plus byte enable and 3 bits for flit type encoding. Anyway, *the standard 32 bit NoC link configuration is kept, which enables to somehow containing packet length in the multi-network architecture and therefore represents a best case for it.* Similar considerations hold for 64 bit architectures.

Finally, for each test, we have considered two network sizes: a 4x4 mesh with 16 cores and a 16x16 mesh with 256 cores. Both topologies were considered with 2 and 4 virtual channels, and for 32 bit and 64 bit NoC architectures.

Performance analysis

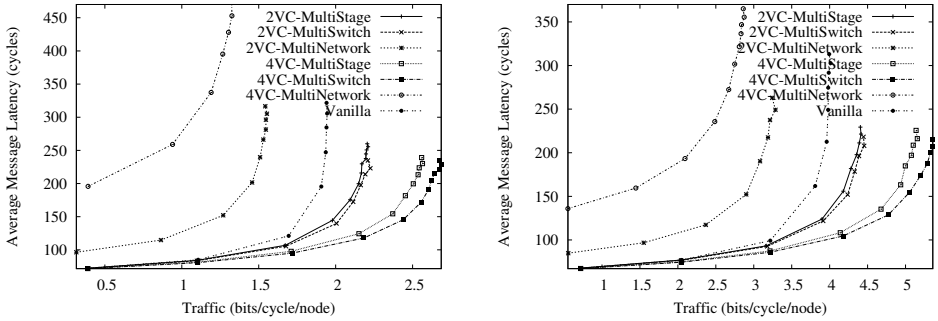
Figure 4.11(a) shows the throughput versus average latency of a 4x4 mesh when considering an uniform traffic pattern with a link width of 32 bits. It shows the results for all three switch architectures with 2 and 4 virtual channels. Regarding the comparison between multi-stage and multi-switch, the use of two virtual channels increases the maximum throughput and decreases the average latency over the vanilla VC-less architecture, but as we increase the number of virtual channels, the improvement in performance becomes smaller. The reason for this lies in the small size of the topology: the HoL blocking effect is almost removed when two virtual channels are used. Notice that the difference in performance between multi-stage and multi-switch is really small. This is due to the fact that 16 cores can not inject traffic enough to saturate the network before saturating the destinations, in other words, the ejection links of each switch become congested sooner than the switch-to-switch links, thus making the impact of the switch architecture over performance marginal.

On the other hand, multi-network achieves a lower maximum throughput than the vanilla architecture, while obtaining a higher latency. This behavior is caused by the increased packet length. Even in the absence of network congestion, as the flit width is decreased, each packet requires a higher amount of cycles to reach the destination, so the base latency increases with the number



(a) 4x4 mesh with 32 bits

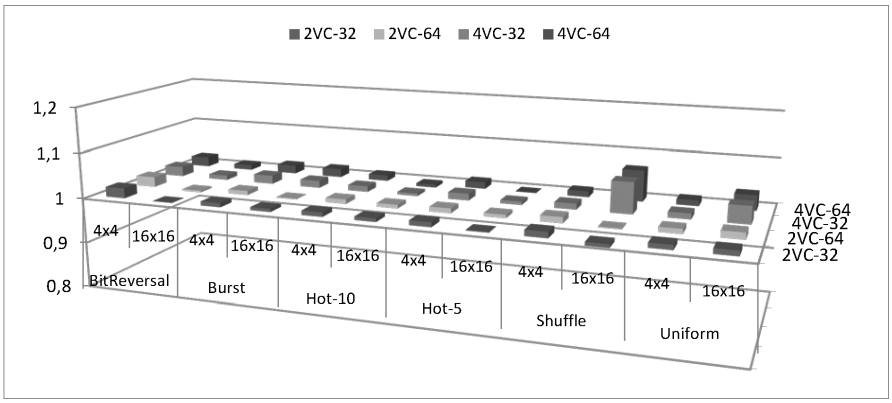
(b) 4x4 mesh with 64 bits



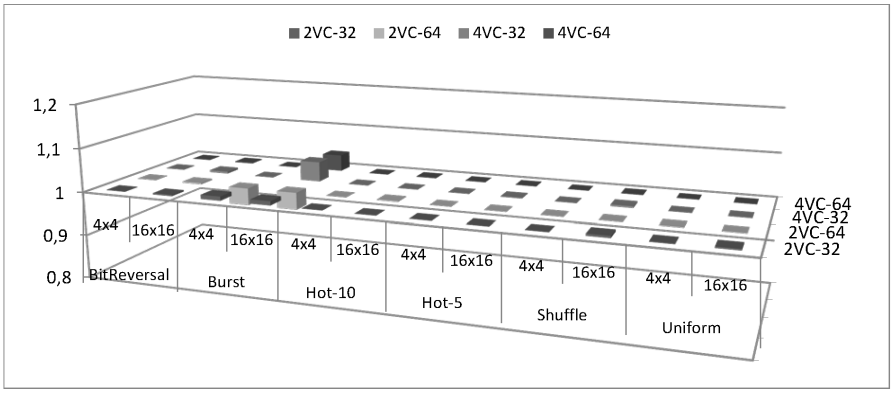
(c) 16x16 mesh with 32 bits

(d) 16x16 mesh with 64 bits

Figure 4.11: Average latency vs Throughput for uniform traffic.



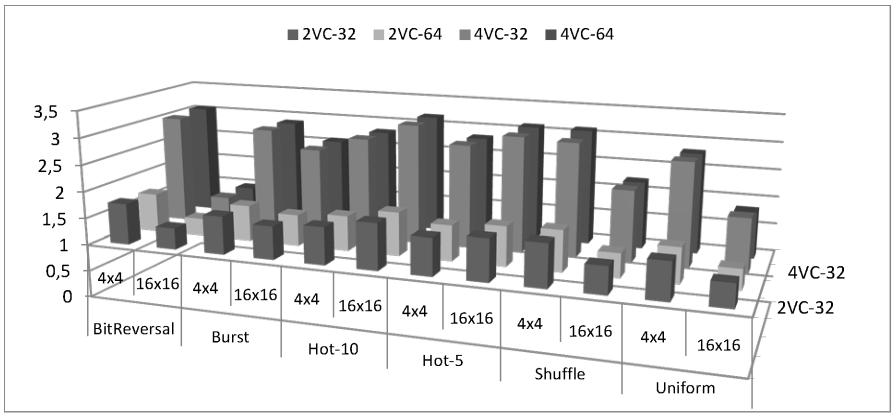
(a) Throughput



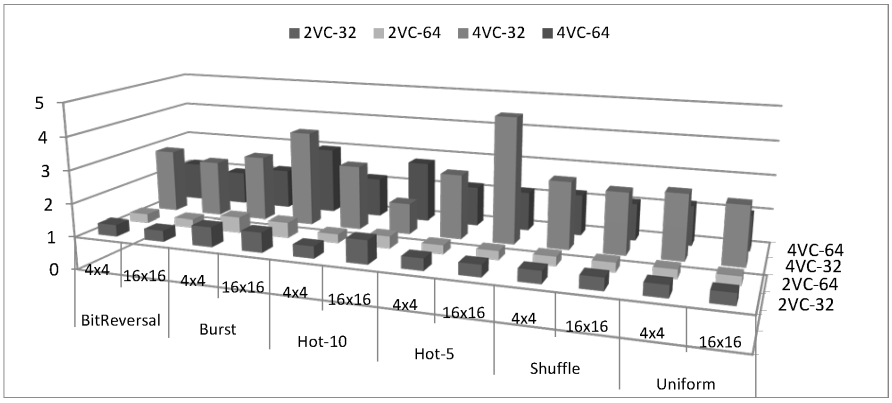
(b) Average latency

Figure 4.12: Multi-switch performance normalized to multi-stage.

of virtual channels. Also, as the flit width is reduced, the number of bits that can be injected into the network in a single cycle decreases, so the throughput is also reduced as the number of virtual channels is increased. This produces an undesired effect: as we increase the number of sub-networks in multi-network, we are reducing the performance of the system. This effect is so critical that even the vanilla solution outperforms the multi-network architecture with two virtual channels. Figure 4.11(b) shows the same results for the 64 bits systems. Asimilar trend can be observed in those experiments, with the difference that the use of 64 bits architectures produce an overall increment in throughput for all solutions, as expected.



(a) Throughput



(b) Average latency

Figure 4.13: Multi-switch performance normalized to multi-network.

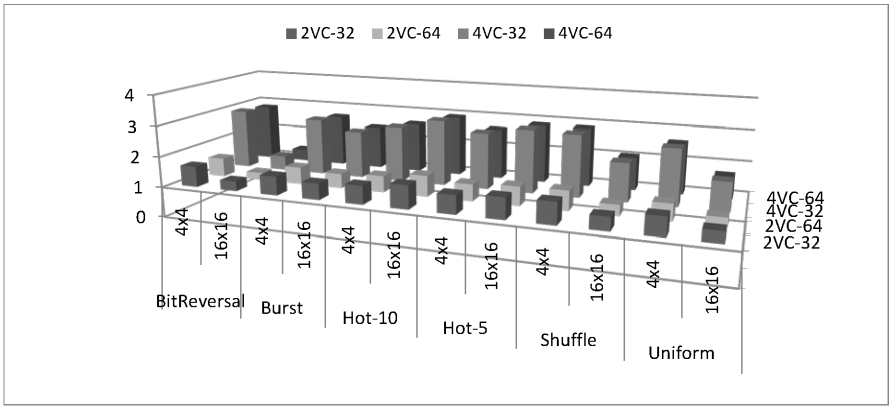
Figure 4.11(c) and Figure 4.11(d) show the same results but for a 16x16 mesh with a link width of 32 and 64 bits, respectively. In those experiments, multi-network is again the worst solution, even worse than the vanilla solution. If we focus on comparing the multi-stage and the multi-switch solutions, the use of two virtual channels is not enough to completely alleviate the effect of HOL over performance. So, when four VCs are considered, the performance is still improved, noticeably decreasing latency and increasing throughput over the two virtual channels case. Regarding the comparison between multi-stage and multi-switch, we can see that, as we increase the number of virtual chan-

nels, the multi-switch solution clearly outperforms the multi-stage one. As the network becomes bigger, the bottleneck moves from the ejection links to the switch-to-switch links, and in this case the replicated crossbars of multi-switch architecture becomes an advantage. Multi-switch is able to move several flits at the same cycle from different input ports to the same output port, so in the case that a virtual channel becomes congested, it has a higher probability to have another flit available to be transmitted through the output link.

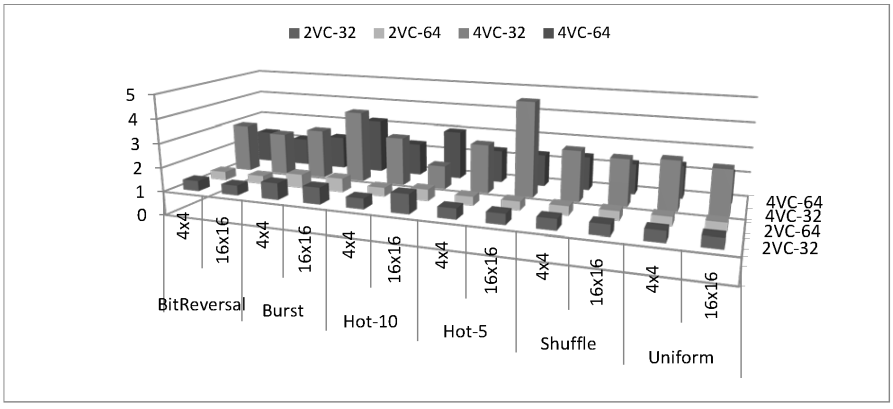
Similar curves were derived for several synthetic traffic patterns for each switch architecture. Those results are summarized in Figure 4.12 and Figure 4.13. Figure 4.12(a) and Figure 4.13(a) show the normalized throughput of multi-switch with respect to multi-stage and multi-network, respectively. Figure 4.12(b) and Figure 4.13(b) show the normalized minimum average latency of multi-switch with respect to multi-stage and multi-network, respectively. In these plots, a bar higher than 1 means an improvement on performance of multi-switch over the other solutions, while a bar lower than 1 means than multi-switch achieves a lower performance.

In Figure 4.12(a) we can observe that the differences between multi-switch and multi-stage throughputs are minimal (the maximum difference observed in the tests was 2%). A similar trend regarding the latency metric can be observed in Figure 4.12(b). Those figures show an important point: multi-switch always achieves a performance similar to (or slightly better than) multi-stage, but never worse.

On the other hand, as can be seen in Figure 4.13(a), multi-switch always achieves a significant higher throughput than multi-network, and this improvement grows up with the number of virtual channels. This is a constant for each test that we have performed, achieving minimum improvement of 34% and a maximum improvement of 200%. Notice that in the figure the improvement of multi-switch over multi-network is lower for 2 VC configurations than for 4 VC ones. The reason behind this behavior is that while in multi-switch the throughput increases with the number of VCs, in multi-network it is the opposite. Also, the improvement of multi-switch in 16x16 meshes for 4 VC configurations is lower than the one obtained for 4x4 meshes. This is because the diameter of a 16x16 mesh is large enough to avoid the generation of a congestion path going from the source to the destination cores in multi-



(a) Throughput/Area efficiency



(b) Average latency/Area efficiency

Figure 4.14: Multi-switch area efficiency normalized to multi-network.

network, thus alleviating the impact of the longest packets over performance. Once again, the average latency metric follows a similar trend, as illustrated in Figure 4.13(b).

Area efficiency

In the case of the comparison between multi-switch and multi-stage it is clear that multi-switch is the best: it achieves higher or equal performance while having a lower area requirement. On the other hand, although multi-network is more area-saving than multi-switch, this latter clearly outperforms multi-

network, so we need a comprehensive metric to assess the two architectures.

We have considered the *area efficiency metric*, defined as $performance/area$, which represents the area cost for achieving a particular performance. Those results are shown in Figure 4.14, that plots the normalized area efficiency of multi-switch with respect to multi-network. In particular, Figure 4.14(a) illustrates throughput area efficiency, while Figure 4.14(b) shows the average latency area efficiency. In the plots, a bar higher than 1 means a higher area efficiency of multi-switch over multi-network, and viceversa. As can be observed, multi-switch is consistently more area efficient than multi-network: although multi-network has a lower area footprint, this is not enough to overcome the large performance gap between both architectures.

4.5 Conclusions

This chapter presents an example of the advantages of adapting off-chip design techniques to the NoCs environment, instead of just adopting them. In particular, we have described a virtual channel switch architecture optimized for a particular NoC scenario: a NoC architecture based on source routing that implements deterministic routing algorithms where in-order delivery is enforced.

With respect to a conventional virtual channel architecture, by simply replicating switches as many times as the intended number of VCs, a simple yet efficient implementation can be achieved. It can provide the same or better cycle time at lower area and power due to the netlist transformations during logic synthesis that can relax delay while saving area/power. Anyway, although superior area properties were demonstrated for a flit width of 32 and 64 bits, it is possible that if flit width keeps increasing, at some point the classical approach will be more area/power efficient, due to the lack of replicated crossbars.

Regarding multi-network solutions, replicating the entire network while keeping the aggregate link width and compound switch buffering constant is slightly even more area efficient but results in a severe degradation in performance. This is mainly due to the increment in packet length and to the reduction in the bandwidth of the injection/ejection links. Although this con-

clusions are valid for 32 and 64 bits architectures, it is likely that if flit width is several times higher than processor word bitwidth, the conclusions might change, being multi-network the best architecture under these new conditions. Anyway, the proposed multi-switch VC architecture is likely to be the best choice for the embedded computing domain due to its superior power-efficiency. Regarding, current CMPs, this conclusion still holds, however this conclusion might change if the flit width increases dramatically (beyond 128 bits), as in this case the cost of replicating the crossbar may be prohibitive.

Summarizing, the proposed multi-switch implementation has demonstrated its superiority for the analyzed case. This proves the benefits of *adapting* interconnection networks solutions to the on-chip environment, opposed to the most common approach of just *adopting* them. In particular, the proposed multi-switch architecture may be synthesized to provide area/power savings over the classical approach, or just a higher operating frequency. Additionally, this proposal is optimal for those cases in which a VC-less switch architecture is already available, reducing the design, validation and verification costs.

Chapter 5

Design Space Exploration for Networks On-Chip

"From then on, when anything went wrong with a computer, we said it had bugs in it."

Grace Hopper, On the removal of a 2-inch-long moth from the *Harvard Mark I* experimental computer at Harvard (1945)

This chapter presents the tool to perform automated Design Space Explorations (DSE) during a stay of the author of this dissertation in the company Lantiq Deutschland GmbH [11], in the context of the collaborative European research project NaNoC [12]. Although most of the work performed during the stay is included in this chapter, there are some details that are not published due to a confidentiality agreement between Lantiq Deutschland GmbH and the author of this dissertation. This chapter starts with a short introduction of the DSE topic presented in Section 5.1. Next, Section 5.2 summarizes the inconsistencies between the work presented in the previous chapters of this dissertation and the needs of the industry in the real world that guided the development of the work presented in this chapter. After, the first version of the tool is detailed in Section 5.3. Following, the second step of the tool development is introduced in Section 5.4: the high level place&route model. Finally, some conclusions are drawn in Section 5.5.

5.1 Introduction

Despite its many advantages, the use of the System-on-Chip (SoC) design paradigm greatly increases the complexity of the design process, due to the variety and number of components that are integrated into a single chip. When NoCs are used to provide the interconnection infrastructure required by the design, this complexity is even further increased, due to the high heterogeneity of the solutions provided by the NoC design paradigm. In overall, the number of possible solutions might be overwhelming, making it hard to select the optimal configuration for a given design. Additionally, because of the complexity of some of these solutions, evaluating their performance in the early stages of the design process becomes a very challenging task. This is a potential design risk that raises the need for new tools to guide the designer to the best candidates inside the design space.

In order to address this challenge, early Design Space Exploration (DSE) is required to find appropriate system architectures out of many candidate architectures. However, two fundamental issues exist. On one hand, as the number of components of a SoC increases, the size of the design space and the complexity of the candidates that compose it become too high to be handled manually by the designer. This demands the use of automated tools, requiring minimal interaction with the designer while exploring the design space. On the other hand, early explorations are usually carried out by means of high level tools that provide performance estimations that the designer will use in order to select the most promising candidates at the earlier stages of the design flow. But, as demonstrated in Chapter 3, there is a gap between high level performance predictions and the real system performance, and this gap grows as the scale of new technologies is reduced. This is a design issue, as it increases the number of design re-spins caused by inaccurate high level predictions, resulting in an increment of the design costs. Chapter 3 also demonstrates that in order to minimize the negative impact of this gap, layout-aware tools must be used in each step of the design process, thus considering the impact over performance of the physical design effects.

In this context, a DSE tool must be capable of building and explore the design space defined by the designer. This process should be as autonomous

as possible, therefore alleviating the exploration cost of large design spaces by minimizing the interaction with the designer. The first step would be the definition of the design space by the designer, and should be the only one requiring interaction with the user. The design space should be defined as accurately as possible, as the smaller the design space, the lower the DSE complexity.

The second step is the population of the design space, that is, the generation of all the candidates to evaluate. In this step, smart population techniques will provide impressive performance improvements, as the lower the number of candidates, the lower the DSE cost.

In the third step, the candidates must be evaluated in order to select the best ones for the design at hand. This step presents two opportunities to improve the tool performance and accuracy. On one hand, advanced exploration algorithms will minimize the number of solutions considered in the DSE process. On the other hand, an exploration methodology similar to the one presented in Chapter 3 will reduce the number and impact of design re-spins due to inaccurate predictions misleading designers towards non-optimal, or even non-valid, solutions.

Finally, in the fourth step, the outcome of the exploration will be provided to the designer. Typically, it is hard to achieve the best solution for a given SoC, as all the solutions cannot satisfy all the requirements to the same extent. However, there could be a best solution for any given scenario (e.g., overall highest throughput or overall lowest latency).

In this chapter we present Lantiq's Design Space Exploration Tool (LDSET), a tool chain for performing early DSE in which performance evaluation is carried out using the Electronic System Level (ESL) performance evaluation framework SystemQ [137]. Many parameters of the NoC, such as clock frequency, buffer sizes, and topology are considered by the tool. It emphasizes the importance of realistic traffic flows and layout-awareness for the performance evaluations, while still retaining considerably high abstraction levels for the simulations. It also implements a high-level *place&route* algorithm capable of predicting the physical characteristics of the NoC topology in the earlier stages of the design process, thus reducing the number of design re-spins due to inaccurate performance predictions.

5.2 The industry point of view

While the work presented in the previous chapters was developed from an academic point of view, the work summarized in this chapter was developed from an industrial one. This shift of the view point was the perfect opportunity to bridge the gap between academy and industry. Following, the main differences between the work presented in the previous chapter and in this chapter is presented, highlighting the ways in which that academy works may influence the development of industrial designs.

5.2.1 Network architecture

LDSET is not constrained to work with a single network architecture. Instead, the network architecture becomes another design parameter that the designer can modify in order to define which networks architectures are considered for each design. As a development requirement of the tool was to facilitate the inclusion of new architectures in the tool, minimal effort would be required in order to introduce models for other architectures. This opens the way to introducing innovative architectures developed in the academy into industry designs flows. For example, the multi-switch virtual channel architecture presented in Chapter 4 could be easily added to LDSET in order to verify its advantages under industry standards.

5.2.2 Target domain

Previous chapters were aimed to a homogeneous domain: the experiments presented were performed in regular tile-based systems, in which all cores were identical to each other. On the contrary, the development process of LDSET was focused to a heterogeneous domain, in which each core may have completely different characteristics and functions (i.e., processors, hardware accelerators, peripherals, and memories). This opens the way to generalize the conclusions extracted from those chapters in a wider variety of designs, as homogeneous designs are a subset of the heterogeneous domain. Regarding the conclusions exposed in Chapter 3, the physical characteristics of a topology will still impact its performance regardless of the target domain, that is, long links will still require pipelining techniques to remain competitive and high degree

switches will still present operating frequency issues, although the exact value in which the issues will arise might change with the switch architecture and the technology library. Finally, Chapter 3 and Chapter 4 both demonstrate that techniques developer from off-chip networks should not be blindly adopted into NoCs, and this conclusion still holds regardless of the design domain.

5.2.3 Traffic characterization

In the previous chapters, the considered tiled-based designs affected the traffic characterization. Those designs are usually employed in general purpose systems, in which it is difficult to predict the exact traffic patterns of the applications that will be executed in the system. For this reason, previous abstract level evaluations were carried out by using synthetic traffic patterns or generic parallel benchmarks.

On the contrary, LDSET aims at providing heterogeneous MPSoC designs, in which the set of functions that each core must perform is usually known at design time. That is, the designer knowns the traffic needs of each core. In particular, LDSET traffic characterization is based on individual master-slave flows for each pair of cores. Two main properties are especially important for each flow: throughput and latency. Traffic flows with high throughput requirements (i.e., bandwidth) need to be prioritized while not starving low-throughput flows. Low-latency flows must be scheduled quickly by the interconnect independently of their throughput requirement.

5.2.4 Core placement strategies

Due to the use of tile-based designs, previous experiments presented in this dissertation considered the core placement strategy irrelevant. In those experiments, while the areas assigned as slots where to place a core were defined by the topology, most (of all) cores were interchangeable: the final slot in which a particular core was physically placed was irrelevant. In this context, task mapping and core placement are decoupled, as any task can be assigned to all, or to a wide range, of cores.

On the contrary, in the heterogeneous designs targeted by LDSET, task and core are tightly related. In this context, the position of a given core inside

the chip area is affected by its communication requirements: cores related by the most demanding communication flows must be physically placed as close to each other as possible while caring about network resource congestion (e.g., trying to avoid hot-spots in the network). Also, the core placement is further complicated by the heterogeneous nature of the cores. Chapter 3 presented a case study in which rectangular tiles affected the wiring pattern of several topologies, and in this example all the tiles presented the same geometry (i.e., shape and size). This scenario is pushed to the limit in heterogeneous designs, as each core may present a completely unique geometry. Therefore, the physical placement of any given core is a trade-off between its communication requirements and the effects of its geometry over the topology layout.

5.2.5 Full custom design

The methodology used in Chapter 3 and in Chapter 4 was driven by synthesis results. That is, in a first step, the synthesis of the systems to evaluate was performed. At a second step, those synthesis results were backannotated into the simulation framework introduced in Chapter 3 in order to obtain accurate performance estimations. Although extremely accurate, this methodology exhibits serious scalability issues. In the case study for 64-tiles systems presented in Chapter 3, those issues were serious enough to force the substitution of real physical synthesis results by prediction of a more abstract model that partially addressed those scalability issues. For this reason, this methodology is ill-suited for wide range DSE explorations, in which dozens or even hundreds of candidates must be evaluated.

In LDSET, the simulation framework does not backannotate synthesis results, but provides the designer with the optimal physical characteristics for any given design. Later in the design process, the network modules must be adapted to fulfill these characteristics. Anyway, this process can produce unrealistic results. For this reason, the designer is able to define feasible values of some physical characteristics, for example, providing a realistic range of operating frequencies in which the NoC may operate.

Moreover, designs are affected by topology layout constrains. Long links still pose limits to the maximum achievable frequency of the design (as demonstrated in Chapter 3). For this reason, LDSET still backannotates information

related to the link synthesis, for example, the maximum achievable distance of a wire for a given operating frequency. In this way, the tool may predict when and where pipelining stages are required, therefore providing accurate estimations regarding link latency.

5.2.6 Simulation framework

In order to maximize LDSET compatibility with the other NaNoC partners, a new simulation framework was implemented utilizing the development environment accorded by the NaNoC partners.

In particular, the new simulation framework was developed in SystemQ [137]. SystemQ is an Electronic System Level (ESL) modelling and simulation framework that can be used to implement abstracts models of a wide range of SoC components (i.e., memories, processors, switches, links, etc.). SystemQ is implemented in SystemC [8], a language for system-level modeling, design and verification. A more detailed explanation of the new simulation framework will be provided in Section 5.3.1.

5.2.7 NaNoC compatibility

LDSET was not conceived as a stand-alone tool. In fact, in the future, LDSET will be part of a toolchain composed by LDSET and other tools developed in the context of the NaNoC project. Therefore, based on the eXtensible Markup Language (XML) standard [13], the NoC description language CEF (Communication Exchange Format) was defined by the NaNoC partners with the aim of improving productivity by removing the translation costs related to the use of different files formats between partners and tools. For this reason, it was a main design constraint of LDSET to use CEF whenever possible, even extending it when needed.

From now on, when referring to the CEF format, a tag will be an identifier placed between the symbols <>. A tag defines the beginning of a new block, while the end of the block is represented by the same tag preceded by a / symbol. For example, the *example* block refers to everything contained between <example> and </example>.

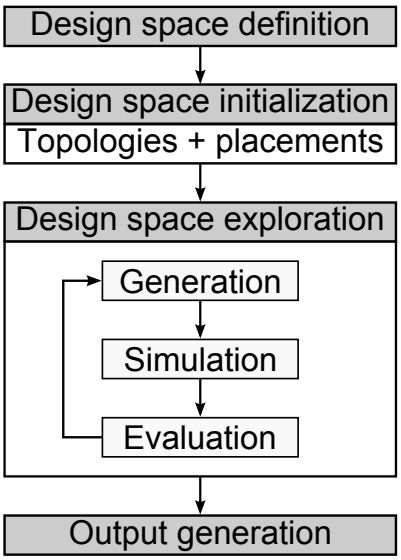


Figure 5.1: LDSET execution flow

5.3 LDSET

Lantiq Design Space Exploration Tool (LDSET) is a tool built to perform the exploration of user defined design spaces. LDSET was designed as two independent, but tightly coupled, parts. First, it implements a versatile and powerful simulation framework. Able to model any topology, it is used to simulate the candidates from the design space. Around this backbone, Perl [14] was used to build the required algorithms to perform the DSE, providing to the designer a set of the best candidates of the given design space. It is important to mention that due to the vast size that a design space may grow into, the tool trades-off precision for performance in many algorithms, providing good candidates instead of the best possible ones. For example, the first version of the core placement algorithm was able to provide optimal candidates, but at the cost of iterating several hours for small spaces (up to 10 cores), and up to several days for large spaces (close to 100 cores), while the current version trades off accuracy for speed, being able to place one hundred core designs in less than 15 minutes.

Any DSE process is composed of 4 main stages: *design space definition*, *initialization*, *exploration* and *output generation*. Figure 5.1 shows the main

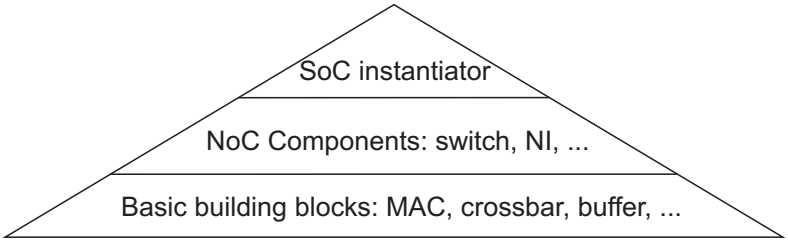


Figure 5.2: NoC simulator hierarchy

parts of LDSET and the data flow that links them. As depicted, the above mentioned stages are present, although the frontiers between each part are blurred, as candidate generation (initialization) and evaluation (exploration) are partially mixed, in order to provide support for more advanced space exploration heuristics (as will be explained in Section 5.3.4).

What follows is a brief summary of each stage. A more detailed explanation of each stage will be provided in the next sections of this chapter. The first stage consists of the definition of the design space exploration (its boundaries, targets, and characteristics) as well as the user commands to the tool. The outcome of this stage is the design space. In the initialization stage, the design space is prepared for the exploration process. This stage carries out part of the candidate generation. In particular, it generates the base candidates, that are like templates that define the immutable characteristics of each family of candidates. An example of a template could be an instance of a topology with a particular core placement. The exploration stage contains the interaction between the simulator framework and LDSET. It carries out simulations and analyzes results. Also, this stage instantiates detailed candidates starting from the templates defined in the previous stage. Finally, in the output generation stage, the user is provided with performance summaries, candidate lists and detailed descriptions for each candidate.

5.3.1 LDSET NoC simulator framework

As mentioned, a new simulation framework for LDSET was developed. This section will provide some generic information about it. It consists of a set of modules developed in SystemQ [137]. Aimed at supporting a wide range of SoC

wit NoC-based interconnects, the design and development of this simulation framework was guided by four objectives:

- **Accuracy**: required in order to safely explore the design space, thus minimizing the risk of discarding valid candidates due to inaccurate performance estimations.
- **Speed**: as the number of candidates of the design space increases, the speed of the simulations becomes critical in order to explore the design space in an affordable time.
- **Versatility**: indispensable due to the wide range of possible candidates that may be contained in a single design space.
- **Reusability**: required in order to provide a tool that can be easily adapted to satisfy diverse needs and/or environments (e.g., other NaNoC partners' needs)

The developed modules were organized in a hierarchy, as depicted in Figure 5.2. The lowest level represents basic building blocks (e.g., crossbar, routing modules). The middle level contains whole network components (e.g, switches, network interfaces) built with the lowest level blocks. The top most level is the most complex. It contains the models of the traffic generators and the top level module, which is able to instantiate a wide range of NoC configurations by using the lower level modules, thus, providing the degree of *versatility* required to represent the wide variety of candidates that may be contained inside the design space.

In this context, the *speed* objective was achieved in two ways. First, each module was heavily optimized to reduce memory cost and processor usage. Second, SystemQ is capable of modifying the behavior of any module, up to some degree, without forcing to recompile them, by means of the parameter file paradigm defined in [137]. In this way, the NoC simulator can be theoretically configured to implement any possible NoC topology composed by any number of cores and switches after a single compilation process. Therefore, simulation time remains almost unaffected by the design space variety, even when working with a multitude of network architectures and/or topologies. In

fact, the builder algorithm does not impose any limitation to the system characteristics other than the limitations imposed by the network architecture and the computational resources available at the execution platform (i.e., number of processors and amount of memory and speed at each one).

Finally, the *accuracy* objective is achieved by modeling any component relevant to the system performance, in such a way that the simulation behavior is as close to the real system as possible. Moreover, the high degree of modularity of this approach favors the *reusability* of the tool, for example, by allowing designers to change some basic blocks in order to modify the network architecture without affecting the more complex modules. Although the current version of the NoC simulator implements as single network architecture, the top level module is prepared to support any number of architectures. Therefore, the effort of adding a new architecture are related to the actual development, verification and validation of the new modules, while minimal changes are required in the top level module.

Finally, the NoC simulator uses advanced traffic generators that can be programmed to inject a wide arrange of synthetic traffic patterns as well as use traffic traces, thus alleviating the impact of design re-spins produced by inaccurate traffic estimations.

5.3.2 Design space definition

The first step is the definition of the design space constraints and parameters by the designer. It comprises three main classes: *design definition*, *design constraints* and *performance objectives*. In this context, LDSET builds the design space by expanding the *design definition* with the *design constraints*, and provides the best candidates that meet the *performance objectives*.

The *design definition* class includes all parameters that allow the designer to define the design space, like the number of cores, the type of each core, the different topology types that should be considered (i.,e. 2D mesh, rings, trees, ...) or the traffic flows that define the design communication pattern. The *design constraints* class is composed by the range of parameters that define the boundaries of the design space (e. g. frequency ranges, buffer sizes or ranges of virtual channels). Finally, the *performance objectives* class defines the evaluation rules that will distinguish between optimal and subop-

```
<design_space_exploration>
 <configuration>
  <design_name>Example</design_name>
  <bounds>
   <bound>
    <name>max_terms_x_switch</name>
    <value>8</value>
   </bound>
   <bound>
    <name>topology</name>
    <value>All</value>
   </bound>
  </bounds>
 </configuration>
 <groups>
  <group>
   <constraints>
    <constraint>
     <name>num_vcs</name>
     <value>4</value>
    </constraint>
   </constraints>
   <evaluation_rules>
    <evaluation_rule>
     min(avg_latency) OR max(throughput)
    </evaluation_rule>
   </evaluation_rules>
  </group>
 </groups>
</design_space_exploration>
```

Figure 5.3: Example of CEF input file DSE definition

timal candidates, like throughput or latency requirements. Figure 5.3 shows an example of design space definition in the CEF format. The design space

```
<value>[0,4]:1  </value>   =>   "0,1,2,3,4"
<value>[0,4]    </value>   =>   "0,1,2,3,4"
<value>[8,0]:-1 </value>   =>   "8,7,6,5,4,3,2,1,0"
<value>[8,0]    </value>   =>   "8,7,6,5,4,3,2,1,0"
```

Figure 5.4: Example range of values

```
<evaluation_rule>latency[0,4]</evaluation_rule> => 0<=latency<=4
<evaluation_rule>latency]0,4[</evaluation_rule> => 0<latency<4
<evaluation_rule>latency[0,]</evaluation_rule>  => 0<=latency
<evaluation_rule>latency[,4]</evaluation_rule>  => latency<=4
```

Figure 5.5: Example numerical comparison in evaluation rules

is defined inside the *design_space_exploration* block. In this block, *design definition*, *design constraints* and *performance objectives* are specified under the tags *configuration*, *constraints* and *evaluation_rules*, respectively. The *configuration* block defines the design space identification (*design_name*) and the design space fixed boundaries, that is, values that the tool cannot modify. Examples of those boundaries are the list of architectures and topologies allowed in the design space, and the maximum number of cores that can be attached to a single switch for any candidate.

Design constraints are identified by an identifier (*name* block) and a value (*values* block), as depicted in Figure 5.3. Examples of design constraints are the range of the number of virtual channels, and the range of sizes of each buffer of the system (e.g., switch input buffer, switch output buffer or NIs buffering) allowed in the candidates contained in the design space. The *value* blocks can be of three types: simple value, list or range. A simple value is a single number, either integer or real. A list of values is composed of a variable quantity of numeric of values, separated by commas. Finally, a range of values is defined in the following format: [min,max]:inc. This will define list of values starting from *min*, finishing at *max* by steps of *inc*, if *inc* is not specified, it is assumed to be 1 or -1, depending on the relative values of *min* and *max*. Several examples of ranges can be found in Figure 5.4.

Regarding the *performance objectives*, their definition is critical to the outcome of the DSE process. For this reason, LDSET implements a reduced but

powerful language to define performance objectives. The evaluation language supports the following operators: *max*, *min*, numerical comparisons and the logic operator AND, OR and NOT. This operators can be applied to any evaluable metric, like throughput or latency.

- **max**: This operator means that only the candidates with the maximum value in the specified metric will be accepted. Example: <evaluation_rule> max(throughput) </evaluation_rule>
- **min**: This operator means that only the candidates with the minimum value in the specified metric will be accepted. Example: <evaluation_rule> min(latency) </evaluation_rule>
- **Numerical comparison**: The XML standard does not allow the direct usage of the > and the < characters, but defines the entities '>' and '<' respectively. To provide a more manageable format, numerical comparisons are specified as intervals: valid_metric[low,high]. If the first character is [then $valid\_metric >= low$, if it is a] then $valid\_metric > low$. On the other hand, if the last character is a] then $valid\_metric <= high$, while if it is a [then $valid\_metric < high$. Also, in case one of the boundaries is non-relevant, no value in the boundaries should be used. Several examples of numerical comparisons can be found in Figure 5.5.
- **Logical operators**: complex evaluation rules can be defined with the use of the logical operators AND, OR and NOT. Also, the use of brackets it is enforced to define the order operators in complex rules. Example: <evaluation_rule> (min(latency) AND max(throughput)) OR latency],100] </evaluation_rule>

Notice that all the performance objectives are defined as a rule applied to one or more metrics. The current implementation of the tools supports the following metrics: throughput, average latency from generation, and maximum latency. Anyway, the evaluation engine is designed to simplify the addition of new metrics as much as possible. As a result, most of the effort that must be dedicated to the addition of new metrics will be dedicated to the calculation of the metric in the simulation framework, and not to the actual addition of the metric to the LDSET engine.

```
<comm_flows>
 <usecase>
  <name>Default Mode</name>
  <id>0</id>
  <initiators>
   <initiator>
    <name>INITIATOR</name>
     <id>0</id>
     <targets>
      <target>
       <name>TARGET</name>
       <id>30</id>
       <flows>
        <flow>
         <type>WR</type>
         <load>0.40</load>
         <data_length>4</data_length>
        </flow>
        <flow>
         <type>RD</type>
         <load>0.20</load>
         <data_length>8</data_length>
        </flow>
      </flows>
     </target>
    </targets>
   </initiator>
  </initiators>
 </usecase>
</comm_flows>
```

Figure 5.6: Example of traffic flow description in CEF.

Although the DSE process should run autonomously, we believe that the designer must have a certain degree of control over the DSE decisions. In

our approach *design constraints* and *performance objectives* are clustered in groups that will be processed individually. In this way, the designer can sort the groups in order to determine which constraints are evaluated first and thereby defining a path through the design space. This is the role of the *groups* tag in Figure 5.3.

Inside the *design space definition* class, the definition of the traffic flows is the most complex and most important task as it defines the rules to guide the core mapping for different topologies in the design space. The traffic flows also impact the results of the DSE tool since the accuracy of the simulations is heavily dependent on the way traffic generators work.

In the literature three major traffic generation approaches exist. First, synthetic traffic patterns are used to send generic messages to predefined or random destinations at a given rate [125] but this is too generic for the heterogeneous SoC designs at which LDSET is aimed to. Second, benchmarks and traces can be used but hardware architectures and application constraints must match the target design. Unfortunately, this information may not be always available in the earlier design stages. Third, mathematical models could be used, however it is difficult to ensure that they are representative for any given application constraints. Our approach to address this issue is to use fully programmable traffic generators, that allow the designer to specify the traffic requirements of each core with as much accuracy as possible, for example, providing support to several traffic flows per core (each one with different parameters, like communication frequency or size), or by distinguishing between transaction types. The CEF format does not include the definition of the traffic flows in the same tag as the rest of the *design space definition*, defining instead a special tag to specify traffic flows: the *communication_flows* tag. Figure 5.6 shows how to configure one of LDSET traffic generators in CEF format. In this example, a single traffic generator is configured to manage a communication flow to a single target. The flow is composed of two sub-flows: generating write transactions for 40% of the simulation time, and read transactions for only 20% of the simulation time. For the rest of the time (40%) the communication flow is idle.

Finally, LDSET accepts component libraries as a way to control the composition of the design space, for example by limiting the possible switch degrees

```
<blocks>
 <block>
  <name>2x2</name>
  <id>0</id>
  <type>300</type>
  <description>2x2 switch</description>
  <size>
   <x>158.114</x>
   <y>158.114</y>
  </size>
  <orientation>
   <initialrotation>0</initialrotation>
  </orientation>
  <inports>2</inports>
  <outports>2</outports>
 </block>
 <block>
  <name>Generic NI</name>
  <id>1</id>
  <type>200</type>
  <description>Just and example of NI</description>
  <size>
   <x>158</x>
   <y>158</y>
  </size>
  <orientation>
   <initialrotation>90</initialrotation>
  </orientation>
 </block>
</blocks>
```

Figure 5.7: Example of library with a switch and a NI

to use by the tool. Those libraries define the available switches, repeaters and NIs that the tool can use to build any NoC. The libraries are defined in

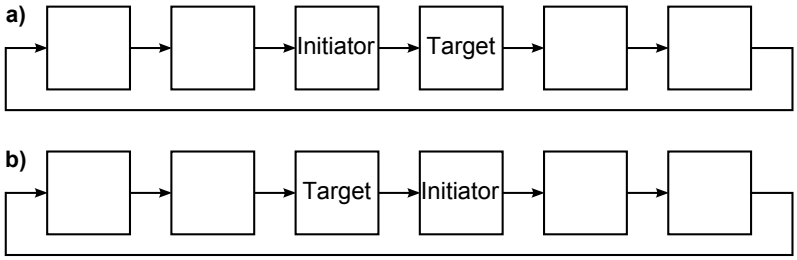


Figure 5.8: Optimal placement of target and initiator a) for a write transaction and b) for a read transaction.

CEF format, as shown in Figure 5.7, which shows an example of a library with one switch and one NI. Repeater stages are defined in a similar way as NIs, being the only difference the *type* tag. As can be observed, libraries also include information about the geometrical properties of the components, this information will be used in the place&route algorithm presented in Section 5.4.

5.3.3 Design space initialization

In the first step of the initialization, each traffic flow is assigned a weight based on the traffic flow communication frequency and the amount of data of each transaction. In this step, the differentiation between read and write transactions is very important. As an example, Figure 5.8 shows the optimal placement of a flow in a unidirectional ring based on the transaction type. In case of a write transaction, the distance from the initiator to the target is critical, while the distance from the target to the initiator is irrelevant. On the other hand, as read transactions are composed of a short request message from the initiator to the target and a longer response message from target to initiator, the distance from the target to the initiator is the one that becomes critical.

In the second step the possible topologies that may accommodate the cores of the design are calculated based on the design space definition. The tool only implements a minimal set of topologies: rings (both unidirectional and bidirectional), 2D-meshes and 3D-meshes. This small set of topologies was selected with the aim of defining a proof of concept environment. It contains simple

```
FOR (current_flow=0 to total_flows-1)
 IF (origin_placed) THEN
  topology1=get_origin_topology()
 ELSE
  topology1=new_topology(origin)
 ENDIF
 IF (destination_placed) THEN
  topology2=get_destination_topology()
 ELSE
  topology2=new_topology(origin)
 ENDIF
 new_topologies=merge(topology1,topology2)
 get_best_topology(new_topologies)
ENDFOR
```

Figure 5.9: Core placement algorithm

topologies (rings), the baseline topology (2D-mesh) and a wiring intensive one (3D-mesh). As LDSET was designed to be easily upgradeable, it is possible to easily add new topologies to the tool with a minimal impact on the rest of its features. Once the topologies are calculated, for each one, cores are placed based on the weight of the flows, in which they are involved, with the aim of minimize the distance (in hops) of the flows with higher weights at the cost of flows with lower weights. The outcome of this process is the set of *template candidates*. A template candidate is defined by the topology instance (e.g, 4x4 mesh with 1 core per switch or bidirectional ring with 4 switches and 4 cores per switch) and by a core placement for its topology instance. Those characteristics are immutable, that is, LDSET never modifies them. Template candidates are not valid candidates but the backbone from which new candidates will be generated.

It is possible to employ methods that will ensure that the best core placement for any given topology inside the design space is chosen. However, these methods are too complex, requiring high amounts of memory and time to process medium sized spaces (20 to 30 cores). For this reason, the current implementation employs an advanced heuristic in order to provide good placements

at the same time that it is able to cope with huge design spaces, being able to calculate several topologies for a 100 cores design in a matter of minutes. 0Following, an explanation of the main strategy behind the placement algorithms will be presented.

Figure 5.9 shows the pseudo-code of the generic placement algorithm. The algorithm processes the communications flows one at a time, based on their weight. For each core involved in flow (origin/destination), it checks if the core was placed in a previous flow. If it is already placed, it retrieves its current topology from the list of topologies, if not, it places the cores in an empty topology composed by a single switch. Then, it merges the topologies that contain the origin and the destination cores of the transaction in a single topology. The merge process calculates all the possible combinations of this two topologies, by means of basic operations (e.g., rotation, append, ...). Finally, among all the options provided by the merge operation, the one with the lower overall weight is provided as the best candidate. The overall weight of a topology is calculated based on the distance between origin and destination of each flow placed in it. This step removes the two topologies that contained the cores involved in the transaction from the list of current topologies and adds the new topology.

It is critical to generate only optimal template candidates, since if irrelevant candidates are allowed in the set of templates, the later stages of the DSE may waste a considerable amount of time in candidates that will be discarded. Figure 5.10 shows the time spent calculating valid candidates for an optimal set of template candidates and for a suboptimal one, for SoC composed by different number of cores. The optimal set of topologies is composed of those templates that have the best overall weight, while the suboptimal set was calculated by a brute force approach, and includes all possible templates defined in the design space. As shown in the figure, the calculation time is greatly reduced when irrelevant candidates are removed as soon as possible.

5.3.4 Design space exploration

The evaluations of the candidates is carried out by the simulation engine. It is the core of the DSE, playing a vital role in the DSE tool accuracy and speed. Although the tool currently implements two heuristics, a brute force approach

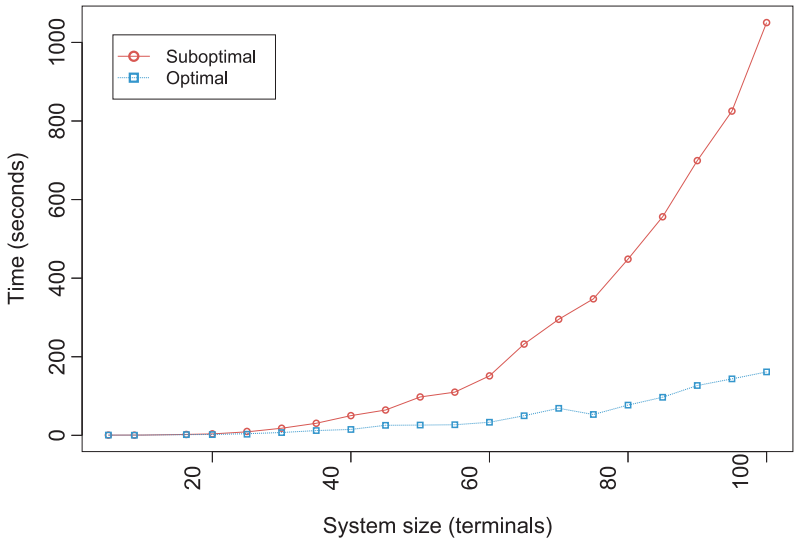


Figure 5.10: Candidate calculation for optimal and suboptimal sets of candidate templates for different system sizes.

and a genetic one, it is designed as a framework for simulation management where different heuristics may be added. In any heuristic, the candidates are generated by applying different values of the design constraints to the template candidates or to other candidates.

The brute force approach is suitable for small designs, and it explores all candidates inside the design space, guaranteeing that the best candidate inside the design space is found.

Due to the high number of possible candidates that may form a design space, advanced techniques to reduce the number of candidate evaluations are required in order to keep the performance of the DSE tool within acceptable levels. A common approach in this case is the use of genetic algorithms [124]. A good example of genetic algorithm is the NSGA-II algorithm [50] that enables the exploration of huge sets of multi-parametric spaces at a relatively low cost while keeping a good accuracy. Genetic algorithms are carried out in several iterations, in which the best candidates of the previous iterations evolve into a new generation of candidates during a pseudo-random mutation process. Although it is not guaranteed that the best candidate will be found, this method will provide good-enough results while significantly reducing the

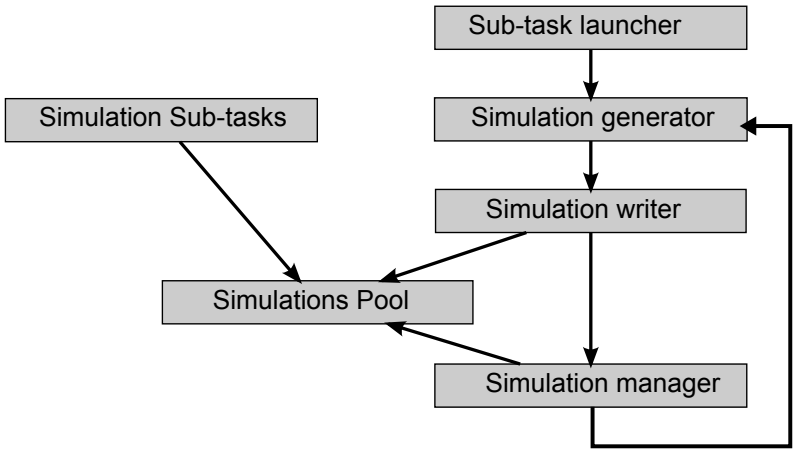


Figure 5.11: Flow diagram of the simulation engine.

number of candidates evaluated, thus becoming a good trade-off solution between accuracy and speed.

The implemented genetic heuristic starts by choosing a pseudorandom set of candidates from the design space, which are simulated and evaluated. The best candidates are chosen as survivors, and form the first generation of candidates. At this point, the genetic algorithm will mutate the first generation into a new set of candidates that will be simulated and evaluated, forming the second generation with the best candidates among the first generation and the new candidates. The mutation process randomly modifies one constraint of the design to a different value (e.g., modifying the number of virtual channels). This process is repeated until the whole design space defined is explored or until the maximum number of generations defined by the user is reached. It must be noted that this heuristic is a performance-accuracy trade-off, as it will not obtain the best candidate of the design space, but the best candidate among all the evaluated ones.

Additionally, the simulation engine is highly parallelized, being capable of calculating new simulation scenarios, while at the same time performing and evaluating simulations in different processes. Moreover, simulations are performed in parallel, exploiting the capabilities provided by modern clusters of computers.

The simulation engine flow is depicted in Figure 5.11. It starts by ex-

ecuting a variable number of subtasks that are responsible for running the simulations. These tasks are the key enabler for parallel execution of LDSET on different machines. They keep monitoring the pool looking for new simulations to execute. When a simulation is found, the task executes the simulation and notifies LDSET about the finalization of the simulation. After launching the subtasks, the simulation engine initializes the subspaces and generates candidates based on the selected heuristic. Then, each candidate is translated into a simulation scenario that is added to the pool. The engine waits until all the simulations required by the heuristic finalize. At this point, the simulation manager is called to evaluate the candidates based on the defined evaluation rules. Finally, the heuristic is called again to decide if the DSE has finished or new simulations are required. The engine facilitates the introduction of new heuristics by modifying only the *simulation generation*, and the *simulation manager* steps. In this way, new heuristics will decide when and how new candidates will be generated and evaluated.

5.3.5 Output generation

The last step is the generation of the DSE output. Although less critical to the tool performance, this decision is important to the global design process of any SoC. Re-spins may happen during a SoC design process. Although DSE tools may help to reduce the number of re-spins by guiding the designer to the best candidate, in case the candidate becomes unsuitable for the SoC requirements in the later stages of the system design process, the designer must restart the whole DSE process with a different design space definition. The negative impact of the restart can be alleviated by providing a set of best candidates, instead of a single one, when possible, thus giving the designer the possibility to quickly change the candidate for the rest of the SoC design process.

In LDSET the designer may choose to have a single or a set of best candidates by setting the performance objectives of the DSE. If those objectives are very tight, LDSET output will consist of a reduced set of candidates, even a single candidate in some cases. If those objectives are relaxed, it will output a larger set of candidates, organized by network architectures and/or topologies, and classified based on their suitability for latency or throughput

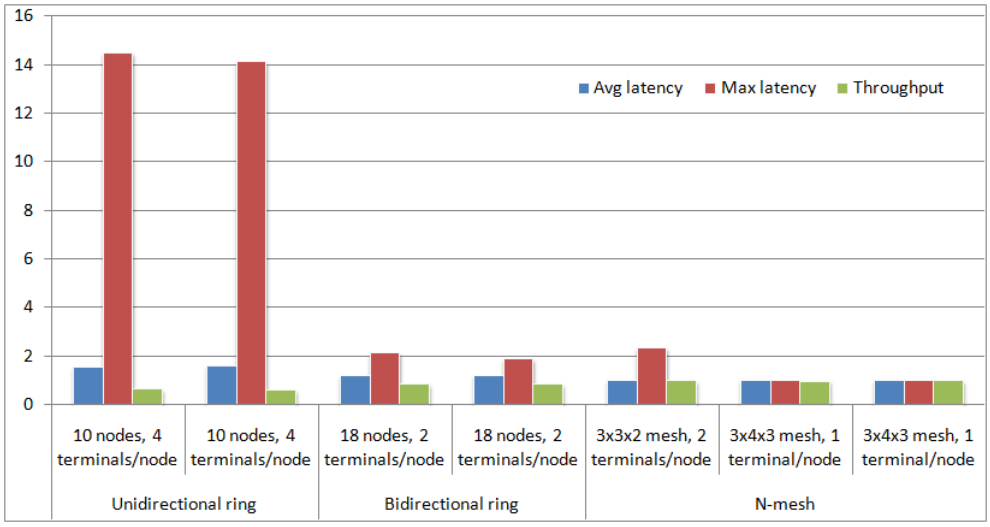


Figure 5.12: Performance results of a DSE for an SoC with 36 cores.

sensitive scenarios.

The resulting set of candidates are also written out in the CEF format, so that a whole description of each candidate is available for further processing by other tools. This description consists of a detailed characterization of every network component (i.e., switches, NIs, links, cores) including placement information and network parameterization (number of buffers slots per buffer, buffer distribution, ...).

5.3.6 Example

As an example, this section shows the performance results obtained by LDSET for an SoC composed by 16 processors that communicate with their local memories using read and write transactions. Additionally, four shared memories exist that are occasionally accessed by all processors. The design space is composed of several NoC topologies that are capable to interconnect 36 cores. The design allows several buffer sizes in the ports of both network switches and Network Interfaces (NIs), together with several possible numbers of virtual channels per port (both in the switches and in the NIs). We also allow the tool to attach up to four cores per switch.

Figure 5.12 depicts the DSE results grouped by topology. The figure shows

the suitability for latency or throughput sensitive scenarios normalized to the best candidate for each metric. Notice that in this step of the DSE, a given topology can be part of two different candidates with different characteristics (i.e., buffer allocation or number of virtual channels). For the sake of simplicity, we show here only a few candidates for each topology. For latency sensitive scenarios, we can classify the candidates in two groups, the ones that provide the best average latency and the ones that provide the lowest maximum latency. While the former candidates are best suited for overall latency, the latter are better for systems with a tight upper bound on communication latency. As illustrated in the figure, mesh configurations show a very similar average latency, while the 3×4×3 mesh presents the lowest maximum latency, making this topology the best choice for latency sensitive scenarios. This topology also shows the best throughput of the whole design space. This topology requires 36 switches, while the other options, like the 3×3×2 mesh and the bidirectional ring with 18 switches require half this number of switches.

5.4 Place & Route

As demonstrated in Chapter 3, in order to provide accurate performance estimations, the simulator framework should consider the physical characteristics of each topology. Our approach to address this topic was a second development stage in which high-level *place & route* capabilities were added to LDSET.

One of the most relevant physical properties of a NoC from a performance point of view is the operating clock frequency, as a too low operating frequency will probably lead to a non-valid candidate, regardless of any other advantages it may present. In a NoC there are two main causes of NoC low frequency: switches and links. Usually, when the critical path of the NoC is in the switches, the straightforward solution is either to use an alternative topology with smaller (and thus faster) switches, or to use another switch architecture or technology library able to improve it. In the former case LDSET has a little room for improvement: the tool already implements support for alternative topologies/architectures. Regarding technology libraries, LDSET and the simulation framework were both modified in order to accept some parameter of the technology library as an input parameter. In this way, the

```
<block>
 <name>TERMINAL</name>  <!-- Name of the block  -->
  <id>0</id>      <!-- Unique numerical ID -->
  <type>101</type>
  <description>Terminals example</description>
  <size>
   <x>1200</x>  <!-- in micro-meters -->
   <y>500</y>  <!-- in micro-meters -->
  </size>
 <bottomleftcorner_position>
  <x>0</x>  <!-- in micro-meters -->
  <y>0</y>  <!-- in micro-meters -->
  <initialx>0</initialx>  <!-- in micro-meters -->
  <initialy>0</initialy>  <!-- in micro-meters -->
  <fixed>false</fixed>  <!-- in micro-meters -->
 </bottomleftcorner_position>
 <orientation>
  <rotation>0</rotation>
  <initialrotation>0</initialrotation>
 </orientation>
 <bitwidth>32</bitwidth>
</block>
```

Figure 5.13: Example of core definition in CEF format.

range of operating frequencies of the design can be provided to the tool.

Regarding link length, a network link may become the critical path if it is too long. In order to correctly evaluate the length of a link it is necessary to synthesize the system and perform its *place&route* (as shown in Chapter 3). However, this task is quite time consuming, thus being ill-suited for a DSE tool designed to work with design spaces with dozens or hundreds of candidates. The solution we engineered consists of a high-level *place&route* algorithm, able to estimate the length of a link based on several parameters of the design space and of each candidate. In a heterogeneous system, each switch and core may have a different area, shape or size. It is important to consider

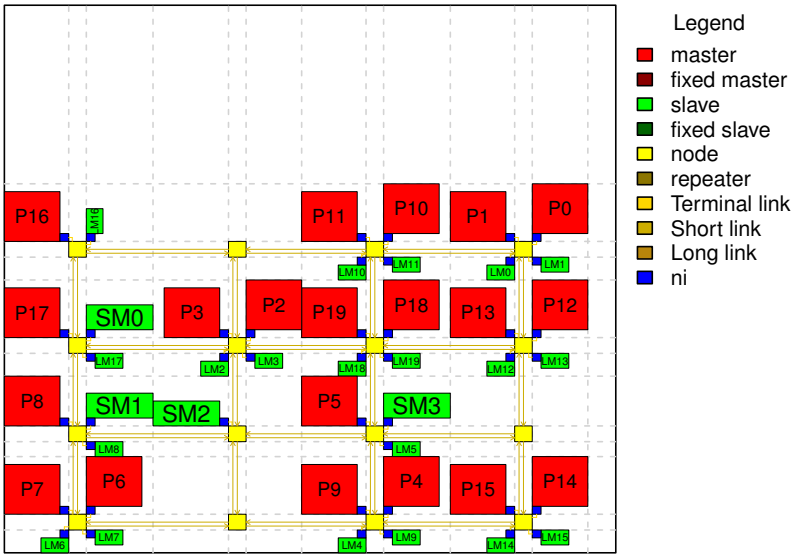


Figure 5.14: Place and route at 1 mm inter-switch spacing at a target speed of 1 GHz (IP Core size not drawn to scale).

the impact of the geometry of each component once placed into the NoC, as links will be wired around them. Also, the buffering strategy employed has a direct impact over link length, as link length is less constrained when switches may implement buffers between themselves and the links. The topology of each candidate is required in order to perform the layout of the candidate's network. Additionally, the algorithm requires to know the maximum length that a link operating at 1 GHz can reach, with and without buffers between links and switches. Then the target frequency of each candidate is required. With this data, using a methodology similar to the one presented in Chapter 3, the tool can extrapolate the operating frequency of a link based on its length. As an additional parameter, we added the possibility of defining fixed cores, whose positions in the layout are fixed by the designer and cannot be modified by the tool. In this case, the algorithm will take care of them, thus adapting the rest of the layout to those placement constraints. Fixed cores are useful when some special layout constraint is present. For example, a core requiring pin connections towards off-chip devices that is placed in the middle of the chip will require wires between itself and the pins of the chip, which

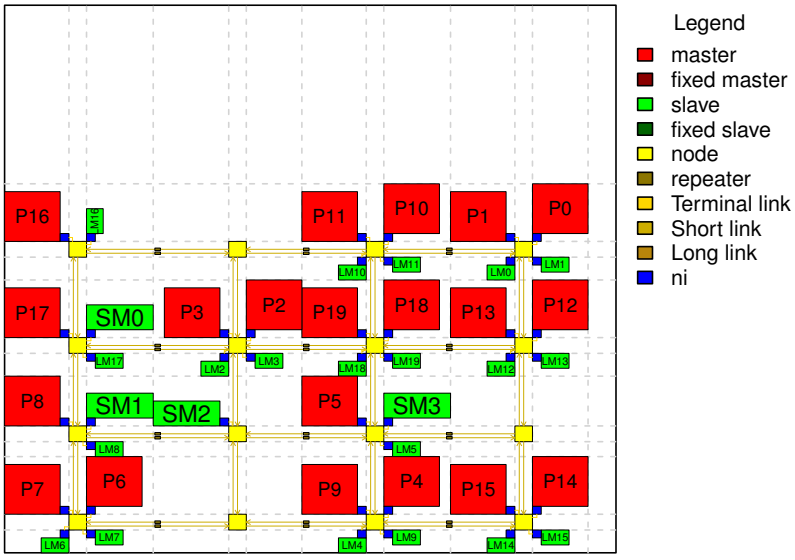


Figure 5.15: Place and route at 1.5 mm inter-switch spacing at a target speed of 1 GHz (IP Core size not drawn to scale).

are commonly placed in the sides of the chip. These wires are likely going to interfere with the links of the network, increasing the complexity of the design and reducing its maximum performance. On the contrary, if this core is place in the edges of the chip, those issues are greatly alleviated. Figure 5.13 shows the CEF code required to configure the placement of a single core.

Figure 5.14, Figure 5.15 and Figure 5.16 were generated by LDSET when working with a 4x4 mesh and the cores are separated 1 mm, 1.5 mm and 3 mm far apart, respectively. In those experiments, LDSET was fed with a link model derived from a 65nm technology. The target frequency is 1 GHz in all cases. As can be observed, in order to achieve the target speed, the tool inserts an increasing number of repeater stages as the interswitch spacing increases. With this data, the algorithm generates a projection of the topology in a 2D-plane, calculating an approximate value for the length of each link. Then, it calculates the number of repeater stages required at each link in order to achieve the target frequency.

In order to introduce this feature in LDSET, both parts of the tool were modified. First, the network architecture was updated in order to implement a

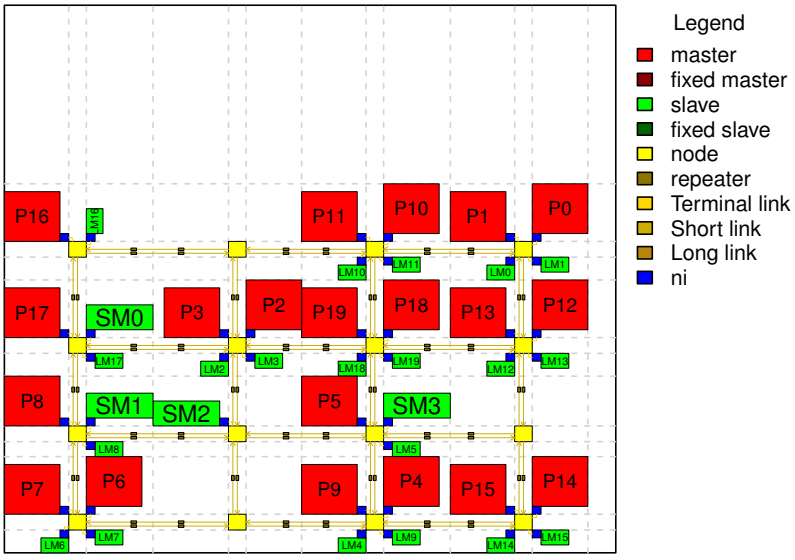


Figure 5.16: Place and route at 3 mm inter-switch spacing at a target speed of 1 GHz (IP Core size not drawn to scale).

flow control mechanism similar to the Stall&Go mechanism exposed in Chapter 3. Second, the NoC simulator was upgraded, adding models of multi-cycle links, in order to represent the additional latency introduced in links when repeater stages are needed for breaking long timing paths. Finally, the topology generation algorithm was updated with an additional step: *place & route*.

The *place&route* process consists of two parts. First, the communication requirements of the design are considered in order to build a core placement, similar to the one presented in Section 5.3.3. In this step, a memory structure is generated that represents the switches of the NoC, as well as the wiring pattern of the topology and the specific cores attached to each switch. Then in the second part, the geometry of the topology and all the cores is considered, calculating the layout of the topology. The placement heuristic of each topology is different, but the underlying method is the same. It is modeled as 2D matrix fill algorithm. The model requires the number of nodes that would be allocated in the X and Y dimensions. Next, it forms a matrix of nodes that represents the layout. In this matrix a node is internally represented as depicted in Figure 5.17. As can be seen, each node is composed of nine cells.

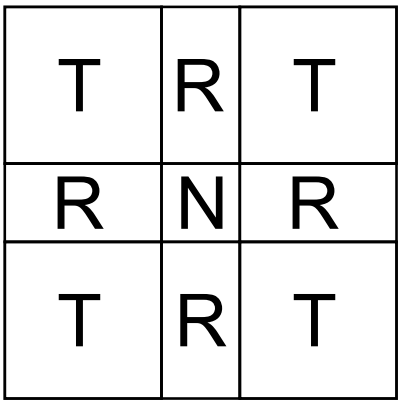


Figure 5.17: Cell distribution of a core.

The center cell is reserved for the switch (label as N), while the cells that are side by side to the node are routing channels (label as R). Finally, the most external cells are reserved for cores (label as T). A single switch can be placed in the center cell, while each core cell can allocate up to two cores. This limitation in the number of cores by cell was imposed in order to reduce the complexity of the automatic place&route algorithm. Notice that each node size is determined based on the cores defined inside it, so the columns and rows of the matrix are as tall/wide as the tallest/widest node they contain.

Once the matrix of nodes is built, connections between the nodes must be defined, thus building the network topology. The connections maybe short or long. Short connections are defined between adjacent nodes, while long connections are defined between non-adjacent nodes. In case that the nodes do not share the same row or column, a link with a turn is automatically added. Finally, NIs and channels between NIs and nodes are automatically added. Some special limitations follow for the case where fixed cores are present, with the aim of lowering the computational cost of the place&route algorithm:

a) Fixed cores must be placed with one side on the border of the floorplan

b) Fixed cores must be inside the floorplan

c) It is not possible to attach more than 4 fixed cores to the same node

d) It is not possible to attach more than two fixed cores of the same side of the floorplan to the same node

Finally, the place&route algorithm is built as an additional layer providing the basic functions required to implement automatic place&route algorithms for any topology. Currently, only the topologies already supported by LDSET were upgraded with place&route capabilities.

5.5 Conlucions

In modern SoC designs, it is required to have a DSE tool in order to reduce the design costs down to acceptable levels. In this chapter, we have shown how early design space exploration and performance evaluation can be used to find an optimal system architecture for a NoC-based SoC with a reasonable design effort. We introduced LDSET, a DSE tool designed to work with large design spaces, and to suit the needs of competitive industrials environments. For this reason, LDSET is heavily optimized, implementing advanced heuristics in order to reduce the exploration cost of any given design space, at the cost of the results accuracy. In particular, two parts of the tool implement such heuristics. First, the topology builder is able to process design spaces with up to 100 cores and complex traffic patterns in a matter of minutes. Second, the simulation engine implements a genetic algorithm in order to reduce the number of simulations required to explore any given design space.

Additional effort was dedicated to help reducing the number and cost of design re-spins, an issue that is ever present in current industry design flows. On one hand, the CEF format was used for any interaction between the user and LDSET, thus helping with the integration of LDSET with other tools. On the other hand, the designer is provided with a list of candidates classified based on their suitability for different scenarios, as well as a detailed description (in CEF format) of all of those candidates. Finally, the approach to address the gap between high-level performance estimations and real system performance was presented, consisting of a high-level *place&route* algorithm, able to predict the impact of the system layout and frequency on system performance.

Chapter 6

Conclusions

"Simple things should be simple, complex things should be possible."

Alan Curtis Kay

In this final chapter, we present the conclusions of this dissertation and a brief list of points that will be addressed in the future. Finally, we also expose the results in terms of scientific publications and other contributions derived from the work presented in this dissertation.

6.1 Conclusions

In this dissertation, we have focused on proposing a complete methodology to perform early design space explorations of SoCs that rely on the NoC paradigm for their interconnection infrastructure.

First, we developed a layout-aware simulation framework for NoCs, with the aim to enable fast and accurate network topology explorations. This latter is not only able to backannotate synthesis results, but it is also capable of performing parametrical explorations of the design space in order to guide the synthesis process. Also, it provides excellent accuracy with respect to RTL simulation, as well as significant simulation time speed-ups when compared to signal-accurate synthesizable models.

Then, we demonstrated the risks of blindly adapting off-chip design techniques to NoCs. In particular, we focused on the topology exploration topic.

By using the developed framework simulation, we show how performance estimations obtained by means of layout-oblivious methodologies are highly inaccurate when compared to layout-aware ones. Additionally, an in-depth analysis of NoC physical design was provided, analyzing the most common design pitfalls and the design techniques that can be used to solve them. The results show that the more complex the wiring pattern of a topology, the higher the performance degradation of its implementations, thus raising the need for expensive design techniques to compensate. Although the methodology allows to characterize the physical implications of a given topology, the cost in terms of synthesis time made our synthesis based methodology oddly suited for quick topology explorations and for larger scale networks, which are the target of this dissertation. For this reason, an approximate methodology that requires less synthesis iterations was devised, capable of preserving visibility of layout effects..

After, we delved into the adaptation of design techniques developed for off-chip networks to the on-chip environment by focusing into network architectures designed for off-chip networks. In particular, we centered our efforts into the virtual channels flow control mechanism. We selected this design technique for two reasons. First, it is one of the most popular network performance boosting techniques in both domains, NoCs and off-chip interconnection networks. Second, the most popular virtual channels switch architecture for NoCs is mostly adopted from the off-chip interconnection networks domains, with minimal changes. The presented experiments showed that for a very common system environment in NoCs (source routing schema working with deterministic routing algorithms and enforcing in-order delivery of packets), by simply replicating switches as many times as the intended number of VCs, a simple yet efficient implementation can be achieved, with respect to a conventional virtual channel architecture. The resulting multi-switch architecture may be synthesized to provide area/power savings over the classical approach, or just a higher operating frequency, while providing excellent Intellectual Property (IP) reuse capabilities. In this way, we showed with a clear example how to optimize for NoCs an off-chip design technique. Even more, we did that by replicating even the crossbar, something that is unthinkable in the off-chip domain.

Summarizing, up to this point, the research efforts of this dissertation were aimed at pointing out the key layout effects that should not be ignored even by abstract design space exploration and synthesis flows. In other words, it helped find the proper mix of layout annotations and low-level detail abstractions that enables a DSE tool to run in reasonable time while generating predictable results.

Finally, based on our previous research, we developed an industrial tool to perform early Design Space Exploration of SoC: LDSET. LDSET was designed to work with large design spaces, and to suit the needs for competitive industrials environments. Starting from the definition of the design parameters, LDSET is able to automatically build and explore the design space, guiding the designer towards the most promising solutions. It is heavily optimized to work with large design spaces, implementing advanced heuristics in order to reduce the exploration time. Additionally, the tool is heavily parallelized, exploiting the potential of modern cluster of computers to further reduce the exploration cost. Special effort was dedicated to reduce the number and cost of design re-spins, an issue that is ever present in current industry design flows.

Summarizing, this dissertation develops a highly accurate DSE methodology. Starting by showing the gap between high-level and synthesis based performance estimations of NoC systems, a high-level layout aware simulation framework was developed. After, this simulation framework is used to develop layout-aware methodologies to evaluate the effectiveness of several key factor in the design of NoCs. Finally, those methodologies were used as the foundation to develop an industrial tool designed to carry out earlier DSE optimized for large design spaces.

6.2 Future Work

Our work present several research points that can be further expanded in the future.

Regarding the topology exploration topic, our analysis was affected by the considered reference architecture. In particular, the cross-domain clocking mechanism implemented tightly relates network and core operating frequency. This mechanism translates into an extremely simple, fast and cheap implemen-

tation, but discourages the use of high-degree switches and/or long links, as slower topologies will aggravate their performance issues by further decreasing the operating frequency of the cores. Therefore, slow topologies require the use of expensive performance boosting techniques in order to achieve competitive performance. This effect can be partially alleviated by the use of Global Asynchronous Local Synchronous (GALS) design techniques, that enables different sections of the chip to run at different clock speeds. That is, while portions of the system may run synchronously, the system as a whole is asynchronous. By designing NIs using a GALS approach, the operating frequency of the network and the cores can be completely decoupled, favoring the use of slow topologies.

Regarding the proposed multi-switch implementation of virtual channels, the proposal can be expanded in two different ways. First, it is possible to expand the functionality of the output port arbiters in order to allow packets to shift virtual channels, improving the performance of the architecture when in-order delivery of packets is not required. Second, the replication of resources presents an optimal scenario to improve the reliability of the multi-switch architecture. For example, it is possible to disable a virtual channel (i.e., a VC-less switch) that presents any kind of failure in order to keep the compound switch operative, although with diminished capabilities.

Finally, several directions exists in which LDSET can be extended. Apart of the use of the above mentioned GALS approach, the inclusion of different use-cases can also prove to be useful. A system may be used with different operating conditions, some of which may be orthogonal to each other. This means that they may not occur at the same time. Specially with multi-function mobile devices, a lot of different use-cases may exist with significant communication requirements, but never at the same time (e.g. video capture and video playback on a smartphone). Therefore, these use-cases exist separately by themselves and their flows may not affect each other. On such occasions, the tool will have to optimize the architecture in order to make it compliant with all the different requirements. At the end, the architecture has to be able to sustain all the traffic as generated by the separate use cases. Finally, the inclusion of more different topologies besides the currently supported ones will render the tool even more flexible and applicable to real-life industry problems. Ideally, LDSET would be able to support irregular structures, where the

topology is built to be a custom fit for the system at hand, but this implies an even larger design space to explore.

6.3 Contributions

While conducting the research presented in this dissertation we had the opportunity to publish our work in scientific conferences, journals, and books related to the topics addressed in this dissertation. Following, we present the list of published works for each of the proposals of this dissertation.

The publications in conferences corresponding to the topology exploration topic are [44, 55, 57–59, 64, 65, 91, 92, 134]:

- F. Gilabert, M.E. Gómez, P. López y J. Duato. *"Analysis of Topologies in the Performance of On-Chip Networks"*. Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip as part of 2007 International Conference on High Performance Embedded Architectures & Compilers. January 2007. ISBN 978-90-382-1127-5

- F. Gilabert, M.E. Gómez, P. López y J. Duato. *"Performance Analysis of Multidimensional Topologies for NoC"*. ACACES 2007 Third International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems. July 2007. ISBN 978-90-382-1127-5.

- S. Medardoni, F. Gilabert, D. Bertozzi, M.E. Gómez, and P. López. *"Towards an Implementation-Aware Transaction-Level Modeling of On-Chip Networks for Fast and Accurate Topology Exploration"*. Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip as part of 2008 International Conference on High Performance Embedded Architectures & Compilers. January 2008.

- F. Gilabert, S. Medardoni, D. Bertozzi, L. Benini, M.E. Gómez, P. López and J. Duato. *"Exploring High-Dimensional Topologies for NoC Design Through an Integrated Analysis and Synthesis Framework"*. The 2nd IEEE International Symposium on Networks-on-Chip (NoCS 2008). April 2008. ISBN 978-0-7695-3098-7

- F. Gilabert, S. Medardoni, D. Bertozzi, L. Benini, M.E. Gómez, P. López and J. Duato. *"High-Dimensional Topologies for NoCs"*. XIX Jornadas de paralelismo. September 2008. ISBN 978-84-8021-676-0

- D. Ludovici, F. Gilabert, C. Gómez, M. E. Gómez, P. López, G. Gaydadjiev. *"Buttery vs. Unidirectional Fat-Trees for Networks-on-Chip: not a Mere Permutation of Outputs"*. 3rd Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip, The 4th International Conference on High Performance and Embedded Architectures and Compilers. January 2009.

- D. Ludovici, F. Gilabert, S. Medardoni, C. Gómez, M.E. Gómez, P. López, G.N. Gaydadjiev, D. Bertozzi *"Assessing fat-tree topologies for regular network-on-chip design under nanoscale technology constraints"*. DATE Design, Automation & Test in Europe. April 2009. IEEE Computer Society Press. ISBN 978-1-4244-3781-8.

- 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"*. 2009 International Workshop on Multi-Core Computing Systems (MuCoCoS'09), as a part of International Conference on Complex, intelligent, and Software Intensive Systems. March 2009. ISBN 978-0-7695-3575-3

- F. Gilabert, D. Ludovici, S. Medardoni, C. Gómez, M.E. Gómez, P. López, G.N. Gaydadjiev, D. Bertozzi. *"On the feasibility of fat-tree topologies for Networks-on-chip"*. XX Jornadas de paralelismo. September 2009. ISBN 84-9749-346-8

- Daniele Ludovici, Georgi N. Gaydadjiev, Francisco Gilabert, Maria E. Gómez, Davide Bertozzi. *"Contrasting Topologies for Regular Interconnection Networks under the Constraints of Nanoscale Silicon Technology"*. Third International Workshop on Networks On Chip Architectures (NoCARC). December 2010. ISBN 978-1-4503-0397-2

In addition, there are several works published as book chapters corresponding to the topology exploration topic:

- F. Gilabert, D. Bertozzi, L. Benini, and G. De Micheli. *Networks-On-Chip: An Interconnect Fabric For Multi-Processor Systems-On-Chip.* Chapter in *"Embedded Systems Handbook, Second Edition: Embedded Systems Design and Verification"*. CRC Press/Taylor & Francis. 2009. ISBN 978-14-398-0755-2

- F. Gilabert, Daniele Ludovici, M.E. Gómez and D. Bertozzi. *Topology Exploration.* Chapter in *"Designing Network-on-Chip Architectures in the Nanoscale Era"*, Chapman & Hall/CRC Computational Science. 2010. ISBN 978-1439837108

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

The publications corresponding to the virtual channels for NoCs topic are [56, 63]:

- F. Gilabert, S. Medardoni, M.E. Gómez, D. Bertozzi. *"Simple and Efficient Implementation of Virtual Channels for NoCs"*. The 7th International System-on-Chip (SoC) Conference, Exhibit & Workshops. November 2009.

- F. Gilabert, S. Medardoni, M.E. Gómez, D. Bertozzi. *"Improved Utilization of NoC Channel Bandwidth by Switch Replication for Cost-Effective Multi-Processor Systems-on-Chip"*. The 4th ACM/IEEE International Symposium on Networks-on-Chip. May 2010. ISBN 978-0-7695-4053-5

In addition, there is a work published as a book chapter corresponding to the virtual channels for NoCs topic:

- Davide Bertozzi, Simone Medardoni, Antoni Roca, José Flich, Federico Silla, and Francisco Gilabert. *Appendix: Switch Models.* Chapter in *"Designing Network-on-Chip Architectures in the Nanoscale Era"*, Chapman & Hall/CRC Computational Science. 2010. ISBN 978-1439837108

Finally, the following publication was made regarding the design space exploration topic for NoCs [136]:

- Sören Sonntag and Francisco Gilabert. *"Design Space Exploration and Performance Evaluation at Electronic System Level for NoC-based MP-SoC"*. Tutorial session and article at International Conference in Computer Aided Design (ICCAD). November 2010. ISBN 978-1-4244-8193-4

Bibliography

[1] http://www.chipwrights.com/.

[2] http://www.intel.com.

[3] http://www.amd.com.

[4] http://focus.ti.com.

[5] http://www.arm.com.

[6] http://www.tilera.com.

[7] http://www.ocpip.org.

[8] http://www.systemc.org/.

[9] http://www.arm.com/products/CPUs/ARM926EJ-S.html.

[10] http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html.

[11] http://www.lantiq.com/.

[12] http://www.nanoc-project.eu/.

[13] http://www.w3.org/XML.

[14] http://www.perl.org/.

[15] International Technology Roadmap for Semiconductors. http://www.itrs.net.

[16] Networked Embedded Systems, vol. 2, Embedded Systems Handbook 2nd edition. CRC Press/Taylor & Francis, 2009, Edited by Richard Zurawski.

[17] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C.A. Zeferino. Spin: a scalable, packet switched, on-chip micro-network. In *Design, Automation and Test in Europe Conference and Exhibition, 2003.*

[18] K. Agarwal, D. Sylvester, D. Blaauw, F. Liu, S. Nassif, and S. Vrudhula. Variational delay metrics for interconnect timing analysis. In *Design Automation Conference, 2004. Proceedings. 41st.*

[19] V. Agarwal, M.S. Hrishikesh, S.W. Keckler, and D. Burger. Clock rate versus ipc: the end of the road for conventional microarchitectures. In *Computer Architecture, 2000. Proceedings of the 27th International Symposium on.*

[20] Tapani Ahonen, David A. Sigüenza-Tortosa, Hong Bin, and Jari Nurmi. Topology optimization for application-specific networks-on-chip. In *Proceedings of the 2004 international workshop on System level interconnect prediction*, SLIP '04, pages 53–60, New York, NY, USA, 2004. ACM.

[21] P.J. Aldworth. System-on-a-chip bus architecture for embedded applications. In *Computer Design, 1999. (ICCD '99) International Conference on.*

[22] M. Arjomand and H. Sarbazi-Azad. Performance evaluation of butterfly on-chip network for MPSoCs. In *International SoC Design Conference 2008*, pages 296–299, Washington, DC, USA, 2008. IEEE Computer Society.

[23] James Balfour and William J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 187–198, New York, NY, USA, 2006. ACM.

[24] Arnab Banerjee, Robert Mullins, and Simon Moore. A Power and Energy Exploration of Network-on-Chip Architectures. In *Proceedings of the*

First International Symposium on Networks-on-Chip, NOCS '07, pages 163–172, Washington, DC, USA, 2007. IEEE Computer Society.

[25] N. Banerjee, P. Vellanki, and K.S. Chatha. A power and performance model for network-on-chip architectures. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 2, pages 1250 – 1255 Vol.2, 2004.

[26] Daniel U. Becker and William J. Dally. Allocator implementations for network-on-chip routers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 52:1–52:12, New York, NY, USA, 2009. ACM.

[27] L. Benini. Application specific noc design. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings.*

[28] L. Benini and D. Bertozzi. Network-on-chip architectures and design methods. volume 152, pages 261 – 272, 2005.

[29] L. Benini and G. De Micheli. Networks on chips: a new SoC paradigm. *Computer*, 35(1):70 –78, 2002.

[30] Luca Benini. Application specific NoC design. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 491–495, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

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

[32] D. Bertozzi, A. Jalabert, Srinivasan Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *Parallel and Distributed Systems, IEEE Transactions on*, 16(2):113 – 129, 2005.

[33] Davide Bertozzi, Luca Benini, and Giovanni De Micheli. *Energy-reliability trade-off for NoCs*, pages 107–129. Kluwer Academic Publishers, Hingham, MA, USA, 2003.

[34] L. N. Bhuyan and D. P. Agrawal. Generalized Hypercube and Hyperbus Structures for a Computer Network. *IEEE Trans. Comput.*, 33(4):323–333, 1984.

[35] F. Boekhorst. Ambient intelligence: the next paradigm for consumer electronics: how will it affect silicon? In *Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International*.

[36] L. Bononi and N. Concer. Simulation and analysis of network on chip architectures: ring, spidergon and 2d mesh. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*.

[37] L. Bononi, N. Concer, M. Grammatikakis, M. Coppola, and R. Locatelli. Noc topologies exploration based on mapping and simulation models. In *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*.

[38] R.V. Boppana and S. Chalasani. Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks. In *IEEE Trans. Computers*, volume 44, 1995.

[39] L.P. Carloni, K.L. McMillan, and A.L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(9):1059 –1076, 2001.

[40] J. Chan and S. Parameswaran. NoCEE: energy macro-model extraction methodology for network on chip routers. In *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, pages 254 – 259, 2005.

[41] Xuning Chen and Li-Shiuan Peh. Leakage power modeling and optimization in interconnection networks. In *Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on*.

[42] Chen-Ling Chou and Radu Marculescu. User-centric design space exploration for heterogeneous network-on-chip platforms. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09,

pages 15–20, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[43] B. Cordan. An efficient bus architecture for system-on-chip design. In *Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999.*

[44] C. Gómez M. E. Gómez P. López G. Gaydadjiev D. Ludovici, F. Gilabert. Buttery vs. Unidirectional Fat-Trees for Networks-on-Chip: not a Mere Permutation of Outputs. In *3rd Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip, The 4th International Conference on High Performance and Embedded Architectures and Compilers*, January 2009.

[45] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[46] W.J. Dally. Express cubes: improving the performance of k-ary n -cube interconnection networks. *Computers, IEEE Transactions on*, 40(9):1016 –1023, 1991.

[47] W.J. Dally. Virtual-channel flow control. *Parallel and Distributed Systems, IEEE Transactions on*, 3(2):194 –205, 1992.

[48] W.J. Dally and C.L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *Computers, IEEE Transactions on*, C-36(5):547 –553, 1987.

[49] W.J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation Conference, 2001. Proceedings.*

[50] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182 –197, April 2002.

[51] G. Della Veccia and C. Sanges. Recursively Scalable Networks for Message Passing Architectures. In E. Chiricozzi and A. D'Amico, editors, *Parallel Processing and Applications*, pages 33–40. Elsevier Science Publishers, North-Holland, 1987.

[52] Jose Duato, Sudhakar Yalamanchili, and Ni Lionel. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[53] Natalie D. Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 35–46, Washington, DC, USA, 2008. IEEE Computer Society.

[54] I.Hatirnaz et al. Early Wire Characterization for Predictable Network-on-Chip Global Interconnects . *SLIP'07*, 2007.

[55] D. Bertozzi L. Benini M.E. Gómez P. López F. Gilabert, S. Medardoni and J. Duato. High-Dimensional Topologies for NoCs. In *XIX Jornadas de paralelismo*, September 2008.

[56] M.E. Gómez D. Bertozzi F. Gilabert, S. Medardoni. Simple and Efficient Implementation of Virtual Channels for NoCs. In *The 7th International System-on-Chip (SoC) Conference, Exhibit & Workshops*, November 2009.

[57] P. López F. Gilabert, M.E. Gómez and J. Duato. Analysis of Topologies in the Performance of On-Chip Networks. In *Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip as a part of 2007 International Conference on High Performance Embedded Architectures & Compilers*, pages 49 – 52, January 2007.

[58] P. López F. Gilabert, M.E. Gómez and J. Duato. Performance Analysis of Multidimensional Topologies for NoC. In *ACACES 2007 Third International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems*, July 2007.

[59] S. Medardoni C. Gómez M.E. Gómez P. López G.N. Gaydadjiev D. Bertozzi F. Gilabert, D. Ludovici. On the feasibility of fat-tree topologies for Networks-on-chip. In *XX Jornadas de paralelismo*, September 2009.

[60] J. Flich and D. Bertozzi. *Designing Network On-Chip Architectures in the Nanoscale Era*. CRC Press, December 2010.

[61] J. Flich, S. Rodrigo, and J. Duato. An efficient implementation of distributed routing algorithms for nocs. In *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*.

[62] M. Galles. Spider: a high-speed network interconnect. *Micro, IEEE*, 17(1):34 –39, 1997.

[63] F. Gilabert, M.E. Go andmez, S. Medardoni, and D. Bertozzi. Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip. In *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, pages 165 –172, May 2010.

[64] F. Gilabert, D. Ludovici, S. Medardoni, D. Bertozzi, L. Benini, and G.N. Gaydadjiev. Designing regular network-on-chip topologies under technology, architecture and software constraints. In *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on*, pages 681 –687, 2009.

[65] Francisco Gilabert, Simone Medardoni, Davide Bertozzi, Luca Benini, María Engracia Gomez, Pedro Lopez, and José Duato. Exploring High-Dimensional Topologies for NoC Design Through an Integrated Analysis and Synthesis Framework. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '08, pages 107–116, Washington, DC, USA, 2008. IEEE Computer Society.

[66] C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez, and J. Duato. Deterministic versus adaptive routing in fat-trees. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*.

[67] C. Gomez, M.E. Gomez, P. Lopez, and J. Duato. An efficient switching technique for nocs with reduced buffer requirements. In *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*.

[68] P. Gratz, Changkyu Kim, R. McDonald, S.W. Keckler, and D. Burger. Implementation and evaluation of on-chip network architectures. In *Computer Design, 2006. ICCD 2006. International Conference on*.

[69] Cristian Grecu, Andre Ivanov, Res Saleh, and Partha Pratim Pande. NoC Interconnect Yield Improvement Using Crosspoint Redundancy. *Defect and Fault-Tolerance in VLSI Systems, IEEE International Symposium on*, 0:457–465, 2006.

[70] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*.

[71] F. K. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, and W. Fichtner. GALS at ETH Zurich: Success or Failure? *Proceedings of the Twelfth IEEE International Symposium on Asynchronous Circuits and Systems*, 2006.

[72] S. Hauck, G. Borriello, and C. Ebeling. Mesh routing topologies for multi-FPGA systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 6(3):400 –408, 1998.

[73] John L. Hennessy and David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[74] Jingcao Hu, Yangdong Deng, and Radu Marculescu. System-Level Point-to-Point Communication Synthesis Using Floorplanning Information. In *Proceedings of the 2002 Asia and South Pacific Design Automation Conference*, ASP-DAC '02, pages 573–, Washington, DC, USA, 2002. IEEE Computer Society.

[75] Ting-Chun Huang, Umit Y. Ogras, and Radu Marculescu. Virtual Channels Planning for Networks-on-Chip. In *Proceedings of the 8th International Symposium on Quality Electronic Design*, ISQED '07, pages 879–884, Washington, DC, USA, 2007. IEEE Computer Society.

[76] Thomas Hubbard, Raimondas Lencevicius, Edu Metz, and Gopal Raghavan. Performance Validation on Multicore Mobile Devices. In Bertrand Meyer and Jim Woodcock, editors, *Verified Software: Theories, Tools, Experiments*, pages 413–421. Springer-Verlag, Berlin, Heidelberg, 2008.

[77] A. Jalabert, S. Murali, L. Benini, and G. De Micheli. Xpipescompiler: a tool for instantiating application specific networks on chip. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings.*

[78] A.B. Kahng, Bin Li, Li-Shiuan Peh, and K. Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*

[79] A.B. Kahng, Bao Liu, and I.I. Mandoiu. Non-tree routing for reliability and yield improvement. In *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on.*

[80] N. Kavaldjiev, G.J.M. Smit, and P.G. Jansen. A virtual channel router for on-chip networks. In *SOC Conference, 2004. Proceedings. IEEE International.*

[81] John Kim, James Balfour, and William Dally. Flattened Butterfly Topology for On-Chip Networks. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 172–182, Washington, DC, USA, 2007. IEEE Computer Society.

[82] M. Kreutz, C. Marcon, L. Carro, N. Calazans, and A.A. Susin. Energy and latency evaluation of noc topologies. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on.*

[83] M. Krstic, E. Grass, C. Stahl, and M. Piz. System integration by request-driven GALS design. *Computers and Digital Techniques, IEE Proceedings* -, 153(5):362 – 372, 2006.

[84] A. Kumar, P. Kundu, A.P. Singh, L.-S. Peh, and N.K. Jha. A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator in 65nm cmos. In *Computer Design, 2007. ICCD 2007. 25th International Conference on*.

[85] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 150–161, New York, NY, USA, 2007. ACM.

[86] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. In *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*.

[87] D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, Y. Durand, J. Durupt, D. Varreau, P. Vivet, P. Penard, A. Bouttier, and F. Berens. A telecom baseband circuit based on an asynchronous network-on-chip. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*.

[88] Charles E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34:892–901, 1985.

[89] P. Lopez, J.M. Martinez, and J. Duato. A very efficient distributed deadlock detection mechanism for wormhole networks. In *High-Performance Computer Architecture, 1998. Proceedings., 1998 Fourth International Symposium on*.

[90] Zhonghai Lu and A. Jantsch. Flit admission in on-chip wormhole-switched networks with virtual channels. In *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*.

[91] D. Ludovici, F. Gilabert, S. Medardoni, C. Gomez, M.E. Gomez, P. Lopez, G.N. Gaydadjiev, and D. Bertozzi. Assessing fat-tree topologies for regular network-on-chip design under nanoscale technology constraints. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 562 –565, 2009.

[92] Daniele Ludovici, Georgi N. Gaydadjiev, Francisco Gilabert, Maria E. Gomez, and Davide Bertozzi. Contrasting topologies for regular interconnection networks under the constraints of nanoscale silicon technology. In *Proceedings of the Third International Workshop on Network on Chip Architectures*, NoCArc '10, pages 37–42, New York, NY, USA, 2010. ACM.

[93] Daniele Ludovici, Georgi Nedeltchev Gaydadjiev, Davide Bertozzi, and Luca Benini. Capturing topology-level implications of link synthesis techniques for nanoscale networks-on-chip. In *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, GLSVLSI '09, pages 125–128, New York, NY, USA, 2009. ACM.

[94] Giovanni Mariani, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. An Efficient Design Space Exploration Methodology for Multi-Cluster VLIW Architectures based on Artificial Neural Networks. In *IFIP VLSI-SoC 2008, International Conference on Very Large Scale Integration of System-on-Chip. Proceedings*, pages 213 –218, 2008.

[95] H. Matsutani, M. Koibuchi, and H. Amano. Performance, cost, and energy evaluation of fat h-tree: A cost-efficient tree-based on-chip network. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*.

[96] H. Matsutani, M. Koibuchi, D.F. Hsu, and H. Amano. Three-dimensional layout of on-chip tree-based networks. In *Parallel Architectures, Algorithms, and Networks, 2008. I-SPAN 2008. International Symposium on*.

[97] Hiroki Matsutani, Michihiro Koibuchi, Daihan Wang, and Hideharu Amano. Adding Slow-Silent Virtual Channels for Low-Power On-Chip Networks. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '08, pages 23–32, Washington, DC, USA, 2008. IEEE Computer Society.

[98] Aline Mello, Leonel Tedesco, Ney Calazans, and Fernando Moraes. Virtual channels in networks on chip: implementation and evaluation on

hermes NoC. In *Proceedings of the 18th annual symposium on Integrated circuits and system design*, SBCCI '05, pages 178–183, New York, NY, USA, 2005. ACM.

[99] Paolo Meloni, Igor Loi, Federico Angiolini, Salvatore Carta, Massimo Barbaro, Luigi Raffo, and Luca Benini. Area and Power Modeling for Networks-on-Chip with Layout Awareness, 2007.

[100] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994.

[101] M. Mirza-Aghatabar, S. Koohi, S. Hessabi, and M. Pedram. An empirical investigation of mesh and torus noc topologies under different routing algorithms and traffic models. In *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*.

[102] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *Proceedings of the joint conference on Languages, compilers and tools for embedded systems: software and compilers for embedded systems*, LCTES/SCOPES '02, pages 18–27, New York, NY, USA, 2002. ACM.

[103] M. Mondal, T. Ragheb, Xiang Wu, A. Aziz, and Y. Massoud. Provisioning On-Chip Networks under Buffered RC Interconnect Delay Variations. In *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, pages 873 –878, 2007.

[104] G.E. Moore. Cramming More Components Onto Integrated Circuits. *Proceedings of the IEEE*, 86(1):82 –85, 1998.

[105] S.S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The alpha 21364 network architecture. In *Hot Interconnects 9, 2001.*

[106] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*.

[107] Srinivasan Murali and Giovanni De Micheli. Bandwidth-Constrained Mapping of Cores onto NoC Architectures. In *Proceedings of the conference on Design, automation and test in Europe - Volume 2*, DATE '04, pages 20896–, Washington, DC, USA, 2004. IEEE Computer Society.

[108] Srinivasan Murali, Paolo Meloni, Federico Angiolini, David Atienza, Salvatore Carta, Luca Benini, Giovanni De Micheli, and Luigi Raffo. Designing application-specific networks on chips with floorplan information. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ICCAD '06, pages 355–362, New York, NY, USA, 2006. ACM.

[109] Umit Y. Ogras. Application-specific network-on-chip architecture customization via long-range link insertion. In *Proc. ICCAD*, pages 246–253. IEEE, 2005.

[110] U.Y. Ogras and R. Marculescu. Application-specific network-on-chip architecture customization via long-range link insertion. In *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*.

[111] Randy B. Osborne. Speculative computation in multilisp. In *Proceedings of the 1990 ACM conference on LISP and functional programming*, LFP '90, pages 198–208, New York, NY, USA, 1990. ACM.

[112] Gianluca Palermo and Cristina Silvano. PIRATE: A Framework for Power/Performance Exploration of Network-on-Chip Architectures. In Enrico Macii, Vassilis Paliouras, and Odysseas Koufopavlou, editors, *Integrated Circuit and System Design*, volume 3254 of *Lecture Notes in Computer Science*, pages 521–531. Springer Berlin / Heidelberg, 2004.

[113] Partha Pratim Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *Computers, IEEE Transactions on*, 54(8):1025 – 1040, 2005.

[114] P.P. Pande, C. Grecu, A. Ivanov, and R. Saleh. Design of a switch for network on chip applications. In *Circuits and Systems, 2003. ISCAS '03.*

Proceedings of the 2003 International Symposium on, volume 5, pages V–217 – V–220 vol.5, 2003.

[115] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C.R. Das. Exploring fault-tolerant network-on-chip architectures. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*.

[116] Dongkook Park, C. Nicopoulos, Jongman Kim, N. Vijaykrishnan, and C.R. Das. A distributed multi-point network interface for low-latency, deadlock-free on-chip interconnects. In *Nano-Networks and Workshops, 2006. NanoNet '06. 1st International Conference on*.

[117] Sudeep Pasricha, Nikil Dutt, Elaheh Bozorgzadeh, and Mohamed Ben-Romdhane. Floorplan-aware automated synthesis of bus-based communication architectures. In *Proceedings of the 42nd annual Design Automation Conference*, DAC '05, pages 565–570, New York, NY, USA, 2005. ACM.

[118] L.-S. Peh and W.J. Dally. A delay model and speculative architecture for pipelined routers. In *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pages 255 –266, 2001.

[119] Fabrizio Petrini and Marco Vanneschi. K-ary N-trees: High Performance Networks for Massively Parallel Architectures. Technical report, 1995.

[120] Francesco Poletti, Antonio Poggiali, Davide Bertozzi, Luca Benini, Pol Marchal, Mirko Loghi, and Massimo Poncino. Energy-Efficient Multiprocessor Systems-on-Chip for Embedded Computing: Exploring Programming Models and Their Architectural Support. *IEEE Trans. Comput.*, 56:606–621, May 2007.

[121] A. Pullini, F. Angiolini, P. Meloni, D. Atienza, Srinivasan Murali, L. Raffo, G. De Micheli, and L. Benini. NoC Design and Implementation in 65nm Technology. In *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, pages 273 –282, 2007.

[122] Antonio Pullini, Federico Angiolini, Davide Bertozzi, and Luca Benini. Fault tolerance overhead in network-on-chip flow control schemes. In

Proceedings of the 18th annual symposium on Integrated circuits and system design, SBCCI '05, pages 224–229, New York, NY, USA, 2005. ACM.

[123] Antonio Pullini, Federico Angiolini, Srinivasan Murali, David Atienza, Giovanni De Micheli, and Luca Benini. Bringing NoCs to 65nm. *IEEE Micro Magazine*, 12(5, September/October):75–85, 2007.

[124] J. Rabindra and M. Prabhat. Design Space Exploration of NoC: A System Level Approach. In *International Journal of Computing and ICT Researc*, 2008.

[125] A.-M. Rahmani, I. Kamali, P. Lotfi-Kamran, A. Afzali-Kusha, and S. Safari. Negative Exponential Distribution Traffic Pattern for Power/Performance Analysis of Network on Chips. In *VLSI Design, 2009 22nd International Conference on*, pages 157 –162, 2009.

[126] D. Rahmati, A.E. Kiasari, S. Hessabi, and H. Sarbazi-Azad. A Performance and Power Analysis of WK-Recursive and Mesh Networks for Network-on-Chips. In *Computer Design, 2006. ICCD 2006. International Conference*, Washington, DC, USA, 2006. IEEE Computer Society.

[127] Crispin Gomez Requena, Francisco Gilabert Villamon, Maria Gomez, Pedro Lopez, and Jose Duato. Beyond Fat–tree: Unidirectional Load–Balanced Multistage Interconnection Network. *IEEE Computer Architecture Letters*, 7:49–52, 2008.

[128] Mostafa Rezazad and Hamid Sarbazi-azad. The Effect of Virtual Channel Organization on the Performance of Interconnection Networks. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 14 - Volume 15*, IPDPS '05, pages 264.1–, Washington, DC, USA, 2005. IEEE Computer Society.

[129] Alienor Richard, Dragomir Milojevic, Frederic Robert, Antonis Bartzas, Alexandros abd Papanikolaou, Kostas Siozios, and Dimitrios Soudris.

Fast Design Space Exploration Environment Applied on NoC's for 3D-Stacked MPSoC's. In *ARCS '10 - 23th International Conference on Architecture of Computing Systens 2010*, 2010.

[130] E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 10350, Washington, DC, USA, 2003. IEEE Computer Society.

[131] L. Scheffer. Methodologies and tools for pipelined on-chip interconnect. In *Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002 IEEE International Conference on.*

[132] D. Siguenza-Tortosa and J. Nurmi. VHDL-based simulation environment for Proteo NoC. In *Proceedings of the Seventh IEEE International High-Level Design Validation and Test Workshop*, pages 1–, Washington, DC, USA, 2002. IEEE Computer Society.

[133] M. Simone, M. Lajolo, and D. Bertozzi. Variation tolerant noc design by means of self-calibrating links. In *Design, Automation and Test in Europe, 2008. DATE '08.*

[134] D.Bertozzi M.E. Gomez S.Medardoni, F.Gilabert and P.Lopez. Towards an Implementation-Aware Transaction-Level Modeling of On-Chip Networks for Fast and Accurate Topology Exploration. In *Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip as part of 2008 International Conference on High Performance Embedded Architectures & Compilers*, pages 97–108, January 2008.

[135] James E. Smith and Andrew R. Pleszkun. Implementation of precise interrupts in pipelined processors. In *Proceedings of the 12th annual international symposium on Computer architecture*, ISCA '85, pages 36–44, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.

[136] S. Sonntag and F. Gilabert. Design space exploration and performance evaluation at electronic system level for noc-based mpsoc. In *Computer-*

Aided Design (ICCAD), 2010 IEEE/ACM International Conference on, pages 336 –339, 2010.

[137] Sören Sonntag and Wenjian Wang. Area and power consumption estimations at system level with systemq 2.0. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 25:1–25:8, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[138] V.. Soteriou, N.. Eisley, Hangsheng Wang, Bin Li, and Li-Shiuan Peh. Polaris: A System-Level Roadmapping Toolchain for On-Chip Interconnection Networks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 15(8):855 –868, 2007.

[139] K. Srinivasan, K. S. Chatha, and G. Konjevod. An automated technique for topology and route generation of application specific on-chip interconnection networks. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, ICCAD '05, pages 231–237, Washington, DC, USA, 2005. IEEE Computer Society.

[140] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, and G. De Micheli. xpipes lite: a synthesis oriented design library for networks on chips. In *Design, Automation and Test in Europe, 2005. Proceedings.*

[141] Alessandro Strano, Carles Hernandez, Federico Silla, and Davide Bertozzi. Process Variation and Layout Mismatch Tolerant Design of Source Synchronous Links for GALS Networks-on-Chip. In *12th IEEE International Symposium on System-on-Chip (SoC 2010)*, 2010.

[142] S. Suboh, M. Bakhouya, and T. El-Ghazawi. Simulation and Evaluation of On-Chip Interconnect Architectures: 2D Mesh, Spidergon, and WK-Recursive Network. In *NOCS '08: Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, pages 205–206, Washington, DC, USA, 2008. IEEE Computer Society.

[143] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, Jae-Wook Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. *Micro, IEEE.*

[144] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar Operand Networks: Design, Implementation, and Analysis.

[145] N. Terrassan, D. Bertozzi, and A. Bogliolo. Spice-accurate systemc macromodels of noisy on-chip communication channels. In *Signal Propagation on Interconnects, 2007. SPI 2007. IEEE Workshop on.*

[146] M. Vachharajani, N. Vachharajani, D.A. Penry, J.A. Blome, and D.I. August. Microarchitectural exploration with Liberty. In *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, pages 271 – 282, 2002.

[147] Ruud van der Pas. Memory Hierarchy in Cache-Based Systems. *Sun Microsystems Blueprints, 2002.*

[148] S. Vangal, N. Borkar, and A. Alvandpour. A six-port 57gb/s double-pumped nonblocking router core. In *VLSI Circuits, 2005. Digest of Technical Papers. 2005 Symposium on.*

[149] T. Villiger, H. Kaslin, F.K. Gurkaynak, S. Oetiker, and W. Fichtner. Self-timed ring for globally-asynchronous locally-synchronous systems. In *Asynchronous Circuits and Systems, 2003. Proceedings. Ninth International Symposium on.*

[150] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on.*

[151] Hangsheng Wang, Li-Shiuan Peh, and S. Malik. A technology-aware and energy-oriented topology exploration for on-chip networks. In *Design,*

Automation and Test in Europe, 2005. Proceedings, pages 1238 – 1243 Vol. 2, 2005.

[152] Roshan Weerasekera, Li-Rong Zheng, Dinesh Pamunuwa, and Hannu Tenhunen. Extending systems-on-chip to the third dimension: performance, cost and technological tradeoffs. In *ICCAD '07: Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, pages 212–219, Piscataway, NJ, USA, 2007. IEEE Press.

[153] D. Wentzlaff, P. Griffin, H. Hoffmann, Liewei Bao, B. Edwards, C. Ramey, M. Mattina, Chyi-Chang Miao, J.F. Brown, and A. Agarwal. On-Chip Interconnection Architecture of the Tile Processor. *Micro, IEEE*, 27(5):15 –31, 2007.

[154] Paul Wielage and Kees Goossens. Networks on Silicon: Blessing or Nightmare? In *EUROMICRO SYMPOSIUM ON DIGITAL SYSTEM DESIGN (DSD 2002)*, pages 196–200, 2002.

[155] P.T. Wolkotte, G.J.M. Smit, G.K. Rauwerda, and L.T. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*.

[156] F. Worm, P. Ienne, P. Thiran, and G. De Micheli. A robust self-calibrating transmission scheme for on-chip networks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(1):126 – 139, 2005.

[157] Sung-Tze Wu, Chih-Hao Chao, I-Chyn Wey, and An-Yeu Wu. Dynamic Channel Flow Control of Networks-on-Chip Systems for High Buffer Efficiency. In *Signal Processing Systems, 2007 IEEE Workshop on*, pages 493 –498, 2007.

[158] Jinwen Xi and Peixin Zhong. A Transaction-Level NoC Simulation Platform with Architecture-Level Dynamic and Leakage Energy Models. In *Proceedings of the 16th ACM Great Lakes symposium on VLSI*, GLSVLSI '06, pages 341–344, New York, NY, USA, 2006. ACM.

[159] T.T. Ye, L. Benini, and G. De Micheli. Analysis of power consumption on switch fabrics in network routers. In *Design Automation Conference, 2002. Proceedings. 39th*, pages 524 – 529, 2002.

[160] Min Zhang and Chiu-Sing Choy. Low-Cost VC Allocator Design for Virtual Channel Wormhole Routers in Networks-on-Chip. In *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, pages 207 –208, 2008.