Modeling, Evaluation, and Implementation of Ring-Based Interconnects for Network-on-Chip

Stephan Bourduas

McGill University
Faculty of Engineering
Department of Electrical and Computer Engineering

May 14, 2008
Montreal, Canada

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Zeljko Zilic, for his guidance over the course of my studies at McGill University. Through his wisdom, he has guided me towards fruitful avenues of research. He has also provided me with insights and feedback that have proven to be invaluable; I am greatly appreciative of his contributions.

My time at McGill University as a member of the MACS lab (now IML) has been stimulating and enjoyable due in large part to my fellow lab members, with whom I have had many productive and interesting conversations. I would like to thank my colleague Jean-Samuel Chenard, who has been a sounding board for ideas and who has collaborated with me on several occasions. I would also like to acknowledge my office-mate Henry Chan for his collaboration and insights over the course of our many discussions. Likewise, I thank my fellow lab members Jean-François Boland, Atanu Chatopadhyay, Marc Boulé, and Benjamin Kuo for their contributions.

I can never repay the debt owed to my parents and extended family for their support over the course of my studies. I thank you all from the bottom of my heart. Finally, I give thanks to Rosemary Seliskar, who has supported and encouraged me to keep striving forward. I appreciate her patience, understanding, and sacrifice, and I hope to someday repay her for her many kindnesses.

Abstract

This thesis investigates the properties of a hierarchical ring architecture, which is composed of several unidirectional rings arranged to form a hierarchy. The proposed hierarchical ring topology exhibits several characteristics that make it well suited for on-chip use as a system-on-chip (SoC) interconnect. First, unidirectional rings reduce routing complexity thereby lowering buffer, area and energy requirements. Second, the simplicity of the routing logic results in low latencies and high clock rates. Finally, the hierarchical ring structure can be partitioned into multiple clock domains enabling the clock rates of individual rings to be tuned to save power while meeting design constraints. The hierarchical ring architecture has been evaluated using high-level behavioral models as well as a low-level register transfer-level (RTL) implementation. Furthermore, the hierarchical rings are combined with the popular two-dimensional mesh architecture to form several composite architectures in order to improve network performance. The mesh architecture exhibits increased latencies, hop-counts, and congestion with increasing network size. To combat these scalability issues, the hierarchical rings are used in the composite architectures to relieve congestion in the center of the mesh and to reduce hop-counts and latencies for long-distance communication, thereby achieving an overall improvement in performance. Simulation results show that the composite architectures decrease the latencies and hop-counts incurred by global traffic, thereby validating the claim that the use of hierarchical rings for global routing can in fact increase the scalability of the normal mesh network used for network-on-chip (NoC) implementations. Finally, wormhole routed mesh networks can suffer from blocking due to *contention* when multiple packets are routed along the same path. A novel task-assignment strategy that accounts for blocking is presented. The strategy assigns tasks to nodes in a way that tries to minimize contention, reduce latencies, and more evenly distribute traffic. Results show that the methodology is effective at reducing blocking costs and latencies when compared to minimizing communication distances only.

Abrégé

Cette thèse étudie les propriétés d'une interconnexion hiérarchique composée d'anneaux unidirectionnels. La topologie d'anneaux hiérarchique possède plusieurs caractéristiques souhaitables pour être utilisée comme interconnexion pour réseau-sur-puce (NoC). En premier lieu, la structure unidirectionnelle des anneaux sert à réduire la complexité de routage, ce qui implique une diminution de l'importance des mémoires tampon requises et économise l'énergie consommée par l'interconnexion. En second lieu, les faibles temps de latences et d'horloge système élevé résultent de la simplicité logique de chaque routeur. Finalement, la structure de l'interconnexion facilite une partition où chaque anneau appartient à son propre domaine contrôlé par une horloge individuelle, ce qui rend possible l'application de stratégies dynamiques permettant l'économie d'énergie. L'architecture proposée a été évaluée grâce à des simulations de modèles de hauts niveaux et par une implémentation logique résistance-transistor (RTL). De plus, les anneaux hiérarchiques sont combinés avec l'architecture de maille (« mesh ») bidimensionnelle pour former plusieurs architectures hybrides afin d'améliorer la performance du réseau. La topologie de maille démontre l'augmentation de latences, du nombre de sauts, et de la congestion avec l'agrandissement du réseau. Cependant, les architectures hybrides utilisent les anneaux hiérarchiques pour réduire la congestion au centre du réseau et diminuer le nombre de sauts et les temps de latences associés avec les communications à longue distance. Il en résulte donc une amélioration globale de la performance du système. Les résultats des simulations démontrent que les architectures hybrides servent à diminuer les temps de latences et le nombre de sauts encourus par les paquets qui traversent de longues distances. Ceci démontre que l'addition des anneaux hiérarchiques au réseau de maille améliore son extensibilité. Finalement, les réseaux maillés à commutation de paquets peuvent souffrir d'une baisse de performance causée par la contention lorsque plusieurs paquets doivent passer par le même port d'accès d'un routeur. Une nouvelle stratégie d'allocation de tâches aide à minimiser la contention afin de réduire les latences et de mieux repartir le trafic sur le réseau. Les résultats démontrent que la méthode parvient à réduire les latences causées par la contention lorsque celle-ci est comparée à une stratégie d'allocation qui minimise uniquement les distances de communication.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

In 1965, a trend was perceived in semiconductor technology whereby the number of transistors that could fit in an integrated circuit (IC) was increasing exponentially — doubling every two years [1]. This observation, dubbed *Moore's Law*, would herald the technological revolution that started decades ago and has continued through to the present day. The increasing chip densities, made possible by improving fabrication technologies, have exposed the limited scalability of traditional design methodologies such that they will become unsuitable in the near future. It is therefore necessary to transition to more *scalable* design methodologies that will enable the implementation of large systems using current and future fabrication technologies.

An *embedded system* is a specialized computer system that is designed to perform a limited number of dedicated functions. The hardware components of an embedded system are often optimized for cost, performance, power consumption and reliability. These specialized systems have become widespread and can be found in consumer electronics such as mobile phones and televisions, as well as in military, medical, and aerospace applications. Early embedded systems were comprised of several individual components integrated onto a circuit board. As fabrication technologies progressed according to Moore's law, increasing chip densities enabled more components to be *crammed* [1] onto a single chip. Instead of requiring several chips to implement a given function, the same functional-

ity could now be achieved by combining several components onto a single chip, hence the era of the *system-on-chip* (SoC) has emerged.

The SoC design methodology offers the potential to integrate processing, storage, and other functionality into a single package in order to achieve improvements in performance, reductions in energy consumption, and savings in overall system cost. Unlike conventional *application specific integrated circuit* (ASIC) design, there is a greater reliance on the reuse of predefined *intellectual property* (IP) cores, which reduces time-to-market and time-to-profit [2, 3]. Early SoC designs were comprised of a handful of components, thus interconnecting them was a relatively simple task. Since designers were already familiar with board level bus-style interconnects, the same concepts were adapted for on-chip use. The continuation of Moore's law eventually enabled chip designers to move from relatively small systems consisting of a few specialized IP cores, to integrating multiple processors to form single-chip parallel computing systems, or *chip multi-processor* (CMP). The advantages of SoC multiprocessors include the provision of throughput that is equivalent to a single large processor, but with reduced energy consumption from using lower voltages and clock frequencies for the individual SoC processors [2]. Another advantage of integrating multiple processors is improvement in manufacturing yield because a defect on one processor need not render an entire chip useless. Commercially available multi-core SoCs include the Sony Emotion Engine [4], the IBM Cell processor [5, 6], the AMD Turion X2 and the Intel Core2 Duo — all examples of early SoC systems consisting of a relatively small number of cores.

When only a few cores are integrated on a chip, the system performance is limited by the *processing elements* (PEs). Therefore, designers concentrate their efforts on optimizing the PEs, trying to squeeze every drop of performance out of them. As more components are integrated, the interconnect begins to affect system performance, until the bottleneck becomes the interconnect and not the processing elements. With increasing clock speeds and chip densities, the on-chip interconnect will become a limiting factor for performance and energy consumption [7, 8, 9]. In fact, it can account for a significant fraction of the total system energy; some sources [9] report figures of over 50%. SoC designs with multiple processors have often relied on simpler bus-based architectures or other ad-hoc

interconnection schemes that do not scale well [10]. If the interconnect cannot meet the communication requirements of the system, the PEs will be underutilized as they sit idle waiting for data. Therefore, the focus must eventually shift to a scalable *communication centric* design approach.

The limits of bus-based architectures are already starting to be realized, and current trends in technology scaling are expected to continue according to Moore's law for the foreseeable future [2]. It is therefore crucial to explore alternatives to bus-based approaches for SoC interconnects because they will be required in the near future. The natural evolution of the on-chip interconnect is to move from a shared-medium approach to a network based architecture — or *network-on-chip* (NoC) [10], which borrows concepts from the domain of computer networking.

1.1 Problem Description

The complexity of current and future SoC designs can be compared to that of an aircraft carrier [11]. In the past, the system complexity was dominated by individual elements where a single component such as a processor represented a significant design effort. For large systems consisting of *tens* or *hundreds* of components, the complexity lies in providing an *efficient communication mechanism*. The components that were previously thought of as complex have now become sub-components of much larger systems, and are simply viewed as blocks to be instantiated as needed. The following analogy between the complexity of a building and a city can be made: where the blueprints of a building are complex, the building itself is still just a small part of the larger city to which it belongs.

A *shared medium* is one of the simplest interconnection structures, where the communication medium, called a *bus*, is shared by all communication devices. Only one component, called a *master*, can be granted access to the network at a time. Once granted access, the master can transfer data to one or more *slaves*. Access to the bus is usually centrally controlled by an *arbiter*. As the number of components connected to the bus grows, so do contention and arbitration times. Therefore, the bandwidth available to each component is inversely proportional to the number of connected components. In addition to the problem of contention,

the capacitance of the bus is the sum of the input capacitances of each connected component, so the current required to drive the bus increases with the number of components. It is therefore unavoidable that the power dissipation and scalability restrictions of the shared medium approach will require a shift to alternative methods for on-chip communication.

An alternative to using a shared medium is to construct an on-chip communication network — a network-on-chip (NoC). In a NoC based approach, a network is constructed by connecting routers together to form a topology using *point-to-point* links, which require less energy to drive, and less time for signals to propagate from sender to receiver. In such a networked architecture, communication between components can be initiated in parallel, thus avoiding the bottleneck of the shared medium approach. A NoC architecture is also more scalable because adding more nodes *increases* the total available bandwidth without adding complexity or increasing contention. A significant advantage over the shared medium approach is that no centralized control and complex arbitration schemes are necessary.

The NoC design methodology was proposed relatively recently [10, 12, 13], and is still a developing research area with many open problems[1]. At first, NoC research focused on trying to port ideas from the fields of networking and parallel computing to the realm of VLSI. It became apparent that this approach was infeasible because NoCs and traditional networks exist in two different environments with conflicting requirements. SoCs differ from traditional networks because of local proximity and because they exhibit much less non-determinism [11]. For example, the *transmission control protocol* (TCP) suite was developed to provide reliable communication over the internet, a world-wide distributed network, over which the sender has no control. The goal of TCP is to provide reliable communication over an unreliable network; as such, the protocol itself is quite complex. On the other hand, an on-chip network exists in a much more stable environment, and the major requirement is for fast, low latency communication[2].

Early papers [10, 13] about NoCs proposed implementing protocol stacks sim-

[1] In fact, the *first* International Symposium on Networks-on-Chip (NOCS) was recently held in Princeton, New Jersey, in May 2007.

[2] Reliable communication is also a requirement, but less of an issue as compared to the internet.

ilar to the 7-layer OSI reference model [14]. These approaches would yield network implementations that would require excessive resources (e.g., area, energy) and be too slow for on-chip communication. To provide fast communication, the approach must be to simplify the network implementation as much as possible [11, 15] — the simpler the network, the fewer resources it requires. Minimizing the resource requirements of the SoC interconnect network is of paramount importance because the network is a means to an end; the real work is being done by the processing elements. As such, a simpler network implementation allows for more resources to be allocated to the processing elements.

The NoC paradigm requires the collaboration of researchers from a range of specialties because there are issues at several levels of abstraction [11]:

- *Software Layer*: Future SoCs are expected to be highly programmable due to the inclusion of general purpose processors. Software written for these systems will have to be highly parallelized and optimized to take advantage of specialized hardware. The full potential of the on-chip network will only be realized if software is properly written to take advantage of the underlying hardware.

- *Network and Transport Layers*: Below the software layer, the network and transport layer are responsible for routing data from source to destination, as well as decomposing and reassembling large messages into packets. The implementation choices made here greatly affect the performance and resource requirements of the network.

- *Physical Layer*: As feature sizes continue to decrease, the wires connecting components are starting to become unreliable. It is therefore necessary to consider communication links as *lossy*. Former design styles utilized strict design rules in order to ensure deterministic circuit behavior. In the future, circuits will be allowed to produce errors, which will be contained, detected, and possibly corrected (e.g. using *error correcting codes* (ECCs)). This will be made possible in NoCs because of the layered design approach that will consider the underlying communication medium as *unreliable*.

In addition to the aforementioned issues, current *electronic design automation* (EDA)

software enables engineers to automate certain aspects of the design flow, but implementing the system interconnect is still a task left largely to the designer. To realize the potential benefits offered by the NoC paradigm, it is important to transition from ad-hoc design methodologies towards automated design flows that will facilitate the instantiation of IP cores and the automatic generation of the system interconnect. A further discussion of open research areas for NoC will be presented in Section 2.1.

1.2 Topology Selection, Modeling, and Implementation

The topology plays a major role in the performance and resource requirements of the network. Currently, only a limited number of topologies have been studied. The development of a library of topologies is needed so that the most suitable candidate can be selected to meet a given set of constraints. The first topology proposed for NoC was a two-dimensional mesh, which consists of tiles placed in a grid. The mesh topology is often used for on-chip implementation because of its planar structure and scalability; the network can be easily enlarged by adding rows and columns. However, the problem with the mesh topology is that it is not suitable for all situations, and the vast NoC design space requires that other topologies be investigated. The topic of this thesis is primarily concerned with investigating the suitability of adapting a hierarchical ring interconnect for on-chip implementation. In addition, several ring-enhanced topologies that attempt to deal with the limitations of the mesh architecture are proposed.

The development of SoC multiprocessors critically relies on system modeling that can quickly and inexpensively verify the behavior of the hardware and software. The software model should facilitate the architectural exploration in a timely fashion, allowing the designers to fine-tune the design at early stages of the project and avoid expensive modifications at later design stages. As SoC complexity increases, it will become more difficult to capture their functionality with deterministic models. Standardized test benches for NoC simulations are critically lacking at this time; the only alternative is to use synthetic (and

stochastic) models to generate traffic on the network. The network topologies explored in this thesis have been studied by using original simulation models. Traffic generation was achieved through the use of stochastic techniques and a task-assignment application written especially for the purpose of generating task graphs for the network simulators.

Many researchers work with abstract high-level models because of the large effort required to construct low-level implementations. To build a full SoC, in addition to implementing the network routers, PEs must be connected to the network, and the actual application code that will run on the platform must be written. Furthermore, writing a scalable parallel software for NoC requires a substantial time investment, while the development of the enabling technologies (e.g., compilers, APIs) is still an open research problem. Also, the resource requirements of these large systems are such that prototyping them on even the largest FPGAs available today is not possible[3]. Due to the aforementioned difficulties, only a limited number of NoCs have been fully realized. As a part of the evaluation of the hierarchical ring interconnect, a RTL level model has been implemented and compared to other implementations in the literature.

1.3 Statement of Original Contribution

The work presented in this thesis deals primarily with investigating issues relating to network topologies for NoCs. The contributions of this thesis are threefold:

1. To show through simulation and RTL implementation that the hierarchical ring topology is suitable for on-chip implementation due to its energy efficiency, planar layout, simplicity of construction, and speed of communication. Furthermore, the structure of the hierarchical rings lends itself to the application of dynamic energy optimization that can significantly reduce the energy required by the interconnect.

2. Several hybrid network topologies for large scale NoCs that combine hierarchical rings with the mesh network are presented. The new topolo-

[3]Limited prototypes are possible, but large systems consisting of many processing elements will not fit on current FPGAs.

gies leverage the advantages of each case against their limitations, yielding an alternative to the popular mesh topology. The effectiveness of the hybrid topologies at reducing average hop counts and latencies is confirmed through the use of behavioral models.

3. The mesh network suffers from congestion at the center, which gets worse as the network size increases. The reason for the congestion is that the most common *task-assignment* strategy is to minimize communication distances between nodes. A new approach that takes contention into account when performing task assignment is presented. This *blocking aware* task assignment algorithm shows that latencies can be reduced while more evenly distributing traffic on the network.

1.4 Self-Citations

The following publications have contributed to the material presented in this thesis:

- *Modeling and evaluation of an energy-efficient hierarchical ring interconnect* [16]: An energy model was used to explore the effects of tuning design parameters on energy consumption and application performance. Furthermore, dynamic clock throttling was efficiently used to reduce the energy consumption of the interconnect without adversely affecting performance.

- *A RTL analysis of a hierarchical ring interconnect for network-on-chip multiprocessors* [17]: The hierarchical ring interconnect that was previously modelled [16] in SystemC was implemented in VHDL. Synthesis results of the implementation were shown to be competitive or superior (area and speed wise) to other NoC implementations reported in the literature.

- *Hybrid ring/mesh architecture* [18]: The delays incurred by long-distance communications grows linearly with network size for the mesh topology. A hybrid network whereby a large mesh is split into smaller sub-meshes that

are connected by a hierarchical ring can help reduce the time needed to traverse long distance in the network. The concept can be compared to adding a highway for long-distance traffic, so that the delays associated with global traffic can be reduced.

- *Augmented ring/mesh architecture* [19]: The hybrid [18] topology can suffer from congestion for certain traffic patterns. A mesh network that is left intact is augmented by a hierarchical ring interconnect used for global routing [19]. In the augmented architecture, multiple paths exist between any two nodes, which enable the introduction of adaptive routing strategies to improve performance.

- *Blocking-aware task assignment* [20]: The problem of task-assignment in a NoC environment is non-trivial due to the many variables that affect performance. Mesh networks often suffer from congestion in the center similar to the way large cities experience traffic in central downtown areas. This problem can be exacerbated by task assignments that try to minimize the distance between communicating nodes without taking into consideration the network architecture. In an effort to spread traffic more evenly over a wormhole-routed mesh network, a new task assignment methodology that aims to reduce congestion was developed [20].

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 presents an overview of the network-on-chip paradigm, followed by a brief introduction to networking basics and topologies. Chapters 3 and 4 present a description of the different architectures, and Chapters 5 and 6 deal with their simulation using high-level behavioral models. Following the high-level simulation results, Chapter 7 presents an investigation into the energy consumption of the hierarchical ring interconnect as well as the effect of the application of dynamic frequency scaling. Chapter 8 presents simulation and synthesis results of the RTL implementation of

the hierarchical rings. Finally, Chapter 9 summarizes the experimental findings and outlines possible future research directions.

The organization can be loosely partitioned into two parts. First, chapters 3–6 deal with high-level modeling of the presented architectures using a custom simulation platform. Second, 7 and 8 investigate the physical properties of the interconnect using energy models and RTL synthesis.

In addition, some preliminary work on future research directions is presented in two appendices. First, a comparison of simulation and synthesis results of an improved hierarchical ring architecture, called a hyper ring, is presented in Appendix A. Second, wormhole routed versions of both the hierarchical and hyper ring architectures that support virtual channels are presented in Appendix B.

Chapter 2

Background and Related Work

This chapter will first present an overview of the emerging paradigm for on-chip communication, *network-on-chip* (NoC) [10, 12, 13, 21], including the reasons for which they are projected to become the preferred method for intra-chip communication. Second, since the topic of this thesis relates to on-chip interconnection networks for system-on-chip (SoC) applications, an introduction to basic networking concepts and terminology will be presented. Lastly, related works will be discussed.

2.1 Network-on-Chip

The complexity of designing efficient and scalable on-chip communication interconnects will continue to grow as increasing numbers of cores are integrated onto a single chip. A major challenge in chip design will be to provide a scalable, functionally correct, and a reliable communication mechanism that will ensure correct system behavior [10]. Trends in technology scaling [2] will render synchronizing components using a global clock infeasible [22, 23, 24]. Thus, a new synchronization method that does not depend on a single timing reference will need to be employed for future designs, namely the *globally asynchronous, locally synchronous* (GALS) [25] design methodology. In a system employing the GALS design methodology, it is impossible to exercise control of global traffic be-

cause of the lack of global synchronization [10, 13, 24]. Therefore, a distributed communication mechanism that enables components to initiate data transfers autonomously without the need for global synchronization is needed — enter the *network-on-chip* (NoC) [10, 12, 13] paradigm for on-chip interconnects.

2.1.1 Paradigm Shift

The rapid advances in microelectronics fabrication technology are enabling increasingly complex components to be integrated onto a single chip. Originally, *small scale integration* (SSI) and *large scale integration* (LSI) technologies enabled chips with gate counts numbering in the tens of thousands [26]. As feature sizes continued to get smaller, *very large scale integration* (VLSI)[1] produced chips with gate counts numbering in the hundreds of thousands [26]. While VLSI technology enabled the design and manufacture of increasingly complex chips, the number of cores that would fit on a single chip was still relatively small, such that complex systems were constructed by connecting several VLSI chips together on a circuit board. Approximately 5 to 7 years ago, what would have previously required multiple VLSI chips could be now be implemented on a *single* chip, hence the term *system-on-a-chip* (SoC) [27]. Early SoC architectures consisted of multiple cores interconnected in an ad-hoc *point-to-point* manner, which eventually gave way to bus-based architectures [28, 29, 30, 31].

The integration of an increasing number of components has made the limitations of bus-based architectures apparent [9, 32, 33, 34]. Increasing the number of cores connected to a bus results in an increase in parasitic capacitance, propagations delays, power consumption and arbitration times. Furthermore, as all the cores must share the bandwidth of the bus, the fraction of bandwidth available to a single core decreases as the total number of connected cores increases. This is due to the fact that only one device can send data over the bus at any one time. Any other device that wishes to send at the same time must wait. For systems consisting of more than 20 cores [35], a bus interconnect quickly becomes the system bottleneck, to the point where performance degrades such that it is

[1] The term *ultra large scale integration* (ULSI) is sometimes used to describe VLSI circuits with more than 1 million gates [26].

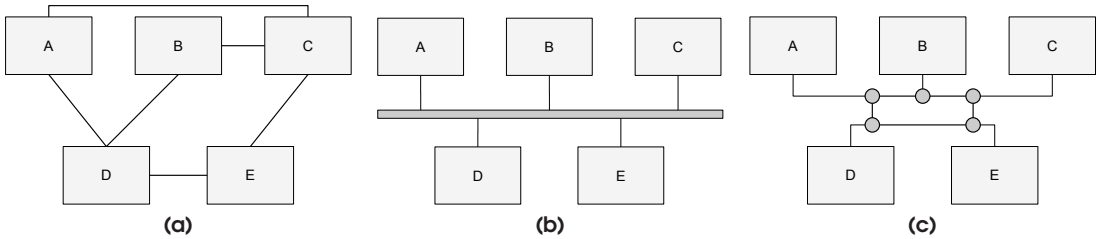


Figure 2.1: Three interconnect styles. (a) Point-to-point connectivity. (b) Bus-based interconnect. (c) A simple network interconnection.

no longer a feasible solution to the communication problem. In addition to considering issues governing performance, energy consumption is also an important factor that limits the scalability of buses. Since the capacitance increases with bus size, so does the energy required to drive the bus. Furthermore, many embedded systems are powered by batteries, and excessive energy drainage can reduce the effective operating time. Even without such limitations, heat generation can prevent proper operation of a device. Hence, the reduction of power consumption is of critical importance in SoC implementations when tens or hundreds of cores are integrated onto a single chip. It is therefore necessary to consider alternatives to bus-based approaches for future systems consisting of many interconnected cores.

The key problem with point-to-point and bus-based approaches are their limited *scalability* and *flexibility*. When it became apparent that new methods of on-chip communication would be needed for the future, the research community started borrowing ideas from the world of networking and parallel processing, which have been dealing with interconnecting a large number of components for decades. The realization that networking concepts could be adapted for on-chip use lead to the introduction of the *network-on-chip* (NoC) [10, 12, 13] for large scale SoC implementations. Examples of 5 cores connected in a point-to-point manner, by a bus, and using an on-chip network are shown in Figure 2.1.

2.1.2 Impact of Technology Scaling

In the early days of electronics, *computation* was expensive and *communication* cheap [35]. That is to say that the processing elements on a chip had the largest

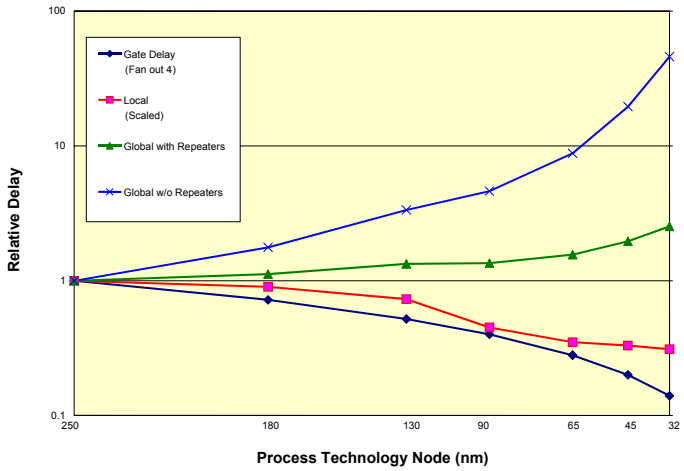


Figure 2.2: Delay for Metal 1 and Global Wiring versus Feature Size (2).

impact on performance, while the method of communication was treated as a secondary concern. Shrinking feature sizes have reduced the wire lengths at the *local* level, but the length of *global* wires remains constant [22, 35]. Furthermore, as clock speeds continue to increase, the time spent on global communication relative to the time spent on local processing has increased [35]. Figure 2.2 shows the projected delay for global wires, local wires and logic gates for future technology nodes, where it can be seen that delays due to wiring dominate gate delays. The implications of Figure 2.2 to chip designers are profound; the performance of the interconnect will greatly impact system performances [7, 8, 9, 23, 36]. The method of communication cannot be treated as an afterthought, it must be considered early in the design process and treated as a primary design criteria.

As discussed in Section 2.1.1, energy consumption in SoCs is a major concern. Whereas technology scaling has benefited computational cores and memories, the energy consumption of global communication has not [22, 23]. In fact, the energy consumption of the interconnect is projected to take up a large proportion of the total energy consumption of the system [9, 34], and the energy required by global wires will actually *increase* [22, 23]. Hence, it is necessary to design the interconnect to be energy efficient, so that as much of the total energy budget as possible can be allocated to computational resources. As the number of cores that must be interconnected on a chip increases to dozens, and even hundreds, designing efficient interconnects will become of paramount importance.

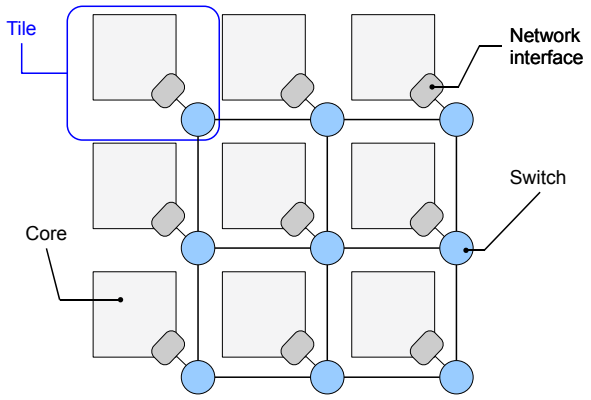


Figure 2.3: Example of a 3×3 NoC where the cores have been arranged in a two-dimensional grid.

2.1.3 Basic Network-on-Chip

The most popular NoC architecture is the two-dimensional mesh network. A mesh consisting of 9 *tiles* arranged in a 3×3 grid is shown in Figure 2.3. Each tile consists of an *intellectual property* (IP) core, a *network interface* (NI)[2], and a switch. The switches are interconnected by physical links. The network interface component decouples the core from the network. The switch implements the routing strategy needed to transmit data from one core to another. A common simplification is to combine all three components and to refer to them simply as a *node* (or tile) on the NoC.

Generally, NoC implementations are either *homogeneous* or *heterogeneous*. A homogeneous NoC is when each node consists of the same kind of *processing element* (PE), as in Figure 2.3. If each PE is a processor capable of performing general purpose computations, the system is commonly referred to as a *chip multi-processor* (CMP) [37, 38, 39]. Conversely, a heterogeneous NoC is usually application–specific, consisting of disparate PEs that can provide a wide range of functionality. It is important to realize that the traffic generated by both types of systems have different characteristics, which need to be properly accounted for when selecting a NoC design. For example, homogeneous CMP systems usually exhibit regular traffic patterns [35]. Conversely, heterogeneous systems exhibit diverse traffic patterns because some nodes on the network are highly solicited,

[2]Sometimes referred to as a *network adapter* (NA).

making them system bottlenecks. Examples of these so-called *hot-modules* [40] are memory controllers, floating point units, on-chip caches, or special purpose processors.

2.1.4 Layered Approach

In the past, the bus was the interconnect most often used for systems consisting of a small number of cores. Even though the bus has all the limitations discussed previously in Section 2.1.1, it does have the advantage of being a well known architecture that has a proven track record. The AMBA [41] bus standard was developed in order to provide a higher level of abstraction so that system designers need not worry about low-level implementation details of the bus. Advantages to using a unified bus standard such as AMBA include [41]:

- *Design portability and reusability*: Components that conform to the standard can be easily reused in new designs.
- *Design productivity*: Reusing a proven communication medium decreases time-to-market.
- *Reliability*: Implementation and testing of a custom communication medium is costly and difficult.
- *Software portability*: Communication primitives remain constant across all AMBA based systems, so embedded software can easily be ported to new platforms.

Similarly, the NoC research community has adopted a *layered* approach akin to the 7–layer OSI reference model [14, 42], which is used in macronetworks. A layered approach provides different levels of abstraction, from the low level physical implementation of wires, to higher level communication primitives such as streams and packets [43]. In general, NoC researchers view the system as having four layers of abstraction [35]:

1. *System*: The system level corresponds to the highest levels of the OSI model, and encompasses applications and the network architecture. At this level,

network implementation details are hidden and high-level communication primitives are used.

2. *Network Interface*: The NI decouples the core from the network and handles end-to-end flow control. The NI is responsible for transparently handling *messages* (or *transactions*) originating from the cores. The NI breaks these messages into *packets* that will be handled by the network.

3. *Network*: The network consists of switches interconnected by links, that together form the topology. At this level, node-to-node flow control is implemented.

4. *Link*: The link level deals with the physical connections between switches. At this level, packets are broken into *flits* (flow control units). A flit can be one or several words long, and can take several clock cycles to send over the link. A flit can be further decomposed into *phits* (physical units). A phit is smallest unit of data that can be transmitted in one clock cycle over a link. Typically, a phit and flit are equivalent unless serialized links are used.

For NoC application, the layers described above are more closely related than in macro-networks since the entire network will be integrated onto a single chip. Unlike macro-networks, the network architecture is fixed and known, meaning that the design of the protocol stack need not be as general as the OSI layers.

2.1.5 Open Problems

NoC is currently considered to be an emerging technology, and as such, there are still several problems that need to be addressed before it is ready for mainstream use. The biggest challenge associated with the NoC paradigm is bridging the gap between numerous domains such as networking, graph theory, optimization algorithms, real-time systems, synthesis, design automation, and others, such that a unified design methodology can be developed and successfully deployed. As discussed in Section 2.1.4, NoC design involves working at varying levels of abstraction, from the system level, down to the physical level. The design of a

NoC involves, but is not limited to, selecting a topology, routing strategy, and application mapping [44].

Early NoC designs focused on using the mesh topology because of its simplicity and planarity. The problem is that the application should not be made to conform to the interconnect. Rather, the interconnect should be tailored to match the needs of application. Instead of selecting a fixed topology beforehand, custom topologies can be generated to match the bandwidth and latency demands of the application [45, 46, 47, 48, 49]. Topology selection is a difficult problem to solve because the communication requirements of different cores can vary widely. For example, designing the network to meet the requirements of highly communicating cores can result in underutilization of portions of the network. Conversely, designing to meet the average bandwidth requirements can result in severe bottlenecks that can negatively impact performance [44]. Although synthesizing custom topologies can improve performance, there can be significant implementation issues due to the irregularity of the custom topology [44].

Proper topology selection requires a method by which to compare different candidate solutions. Classic benchmarks for multiprocessor systems are application–oriented and cannot be used directly for communication centric architectures [50]. Furthermore, in contrast to multiprocessor systems that are homogeneous, most NoC systems are projected to be heterogeneous, which means that a suite of benchmarks designed strictly for NoCs is needed [50, 51]. Ideally, real application traffic would be used to evaluate the performance of an interconnect. The problem is that running a real application on a NoC requires that all of the components, hardware and software, be fully implemented and integrated. This is often not feasible for individual researchers concentrating on some specific aspect of NoC design. In addition, real application traces are often considered trade secrets due to the high cost associated with producing them, consequently, companies are unwilling to donate their traces to the public domain [50]. The NoC research community has thus been mostly limited to using pseudo-random graphs [52], or stochastic traffic generation [50, 51], making it difficult for other researchers to reproduce the results.

The routing algorithm also plays an important role in the performance and cost of the interconnect. More complicated routing algorithms can improve per-

formance, but at the cost of area and power. Hence, designers need to also evaluate the trade-offs associated with algorithm selection [50]. Closely related to routing is the switching technique used by the network. Higher performance can be achieved by using virtual channels and complicated arbitration schemes, but again, there is a trade-off between complexity and performance versus cost. The switching technique can also be constrained by the application requirements. For example, an application under *quality of service* (QoS) constraints [53] may require circuit switching be used even though wormhole routing has better average latency characteristics.

Thus far, relatively low-level implementation details have been discussed. Once the hardware has been implemented, the application mapping and core placement still needs to be addressed. Much work regarding task scheduling has been done in the field of *real-time systems* (RTS), however, much of the work assumes deterministic communication patterns. Recall from previous discussion that shared-medium (bus) interconnects have been commonly used in embedded systems. A bus has a more predictable behavior than a network. Where the communication latencies of a bus can be statically calculated, the communication delays over a network are more dynamic and depend on numerous variables. Therefore, the scheduling problem remains an important problem for NoC design [44].

Related to application mapping, IP core placement is also an issue that requires addressing. To optimally use the available network resources, the IP placement should be performed in conjunction with the application mapping. This is a complex problem because each can affect the other, and hence some form of feedback should be used such that the combined process can be performed iteratively [44].

2.1.6 NoC Modeling

The widespread adoption of *high-level description languages* (HDLs) such as VHDL and Verilog have enabled a large increase in productivity. The HDLs allowed designers to specify complex functionality at the *behavioral* and *register transfer level* (RTL) instead of manually having to perform schematic and hand-layout as was previously the norm [54]. The emergence of the SoC and NoC paradigms have

exposed the limitation of HDLs for modeling large and complex systems. The problem with HDLs is that designers are forced to deal with low-level details, and sophisticated data abstraction capabilities such as those provided by high-level programming languages are not available [54]. Additionally, creating test-bench code for anything more than simple module level testing is time-consuming and inefficient. Since system level test-bench code is likely to be more complex, and must detect and report errors, the expressiveness and flexibility provided by high-level languages such as C++ is better suited to the task. The *SystemC* [55] modeling language is a C++ library and simulation kernel that enables hardware constructs such that simulation models can be written in C++. SystemC supports modeling at the system, behavioral, and RTL levels, which can be used simultaneously within the same design. It provides the ability to use any C++ library to provide added functionality not available in HDLs for the purposes of rapid prototyping, for example:

- Random number libraries [56] can be used to generate random traffic.
- Logging frameworks [57] can provide sophisticated debugging and trace capabilities.
- Command line parser libraries enable input parameters to be easily added.
- Container classes such as the STL `Vector` can provide added flexibility and ease of use, since managing dynamically created objects is easier than having to deal with raw pointers.

The simulation times are just as important as the flexibility provided by the modeling language. When HDLs were first introduced, hardware modules were designed and implemented independently from the software. Currently, and in the future, large scale SoCs will undoubtedly include processor cores and embedded software, necessitating time consuming co-simulation of hardware and software. A faster, more efficient method for simulating hardware and software components in the same environment is needed [54]. For this reason, SystemC has become popular in NoC research for modeling and simulating large and complex systems. Figure 2.4a shows how the SystemC libraries and C++ compiler

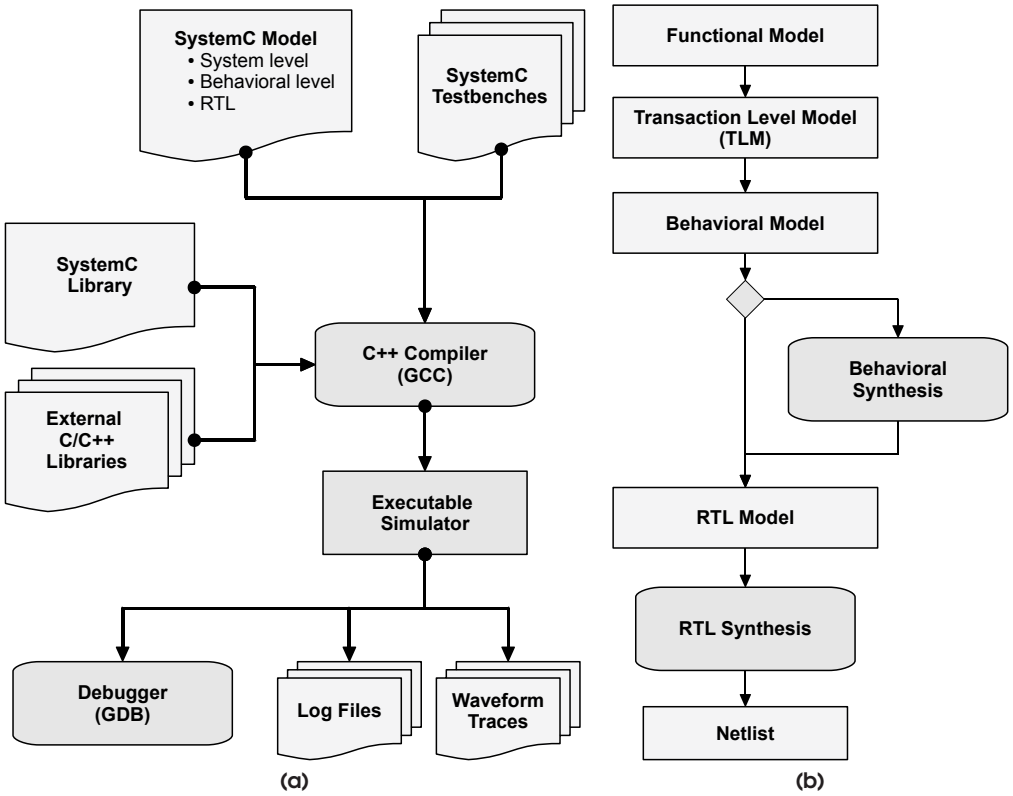


Figure 2.4: (a) SystemC simulation is performed by compiling the models and testbenches into executable binaries. (b) SystemC can be used to describe models at the system, behavioral and RTL levels of abstraction.

are used to create an executable specification that can be debugged using a standard debugger. A major advantage of SystemC that has spurred its widespread adoption by industry and university researchers is the fact that the libraries are *open source*, and no third party EDA software is needed for simulation — the user can run the binary executable from the command line. Figure 2.4b shows how SystemC can be used to model a system at different abstraction levels. When SystemC was first introduced, it was mainly used to perform high-level modeling, and synthesis was not possible. Recently, EDA tools that can synthesis a *subset* of SystemC have emerged, but they are currently not as advanced as VHDL/Verilog synthesis tools.

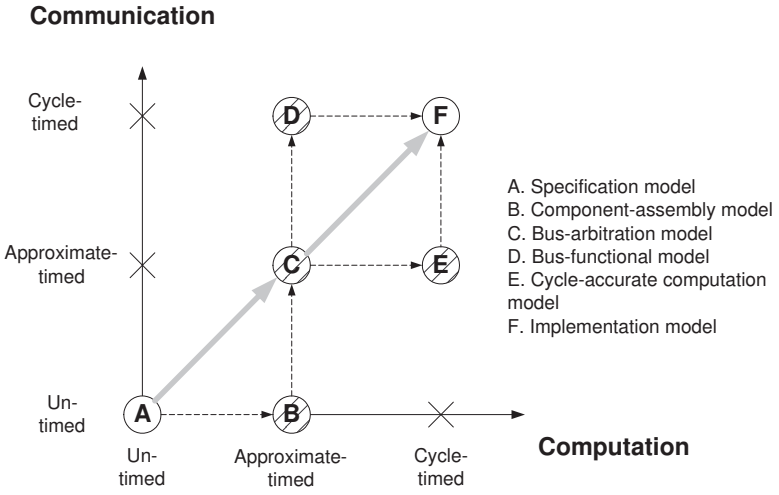


Figure 2.5: System modeling graph [58].

2.1.7 Transaction-level Modeling

The *transaction-level model* (TLM) shown in Figure 2.4b has garnered much interest because low level implementation details are omitted, and communication between components is achieved through atomic *transactions*. Communicating components are connected by *channels*, over which transactions can be initiated by calling interface functions of the channel interface. This approach permits the designer to experiment with different architectures by replacing the communication model with another. Note that this *pluggable* characteristic of TLM models is only possible if each model respects a common interface.

The TLM level of abstraction shown in Figure 2.4b is depicted as a distinct level, different from the functional and behavioral levels. In fact, the demarcation between levels is blurred and TLM level models can themselves vary in abstraction level. In [58], the authors present a system modeling graph as shown in Figure 2.5, which describes the different abstraction levels available for transaction level models. The x-axis represents computation and the y-axis represents communication. Each axis has three degrees of time accuracy [58]:

1. *Untimed* models represent only the functionality of the system without any timing or architectural information. This is the highest level of abstraction that can be used to quickly prototype systems.

2. *Approximate-timed* models include some basic architectural and timing information. Simulations are performed without the aid of RTL or instruction set simulators (ISS).

3. *Cycle-timed* models contain implementation details that enable cycle-accurate simulation using RTL/ISS simulators. These type of models can require a large amount of time to simulate.

A completely untimed model corresponds to node A in Figure 2.5, termed a *specification model* [58]. Such models often pass data between processes using shared variables, and so the architecture of the communication medium is completely absent. This lack of communication specification limits the usefulness of specification models for NoC modeling, which is primarily concerned with interconnect details. Node F corresponds to a cycle-accurate model, called the *implementation model* [58]. As the name suggests, the cycle-timed model requires low-level implementation details to be implemented so that RTL and/or ISS simulations can be performed. Since RTL level implementation takes the longest to simulate, it is not appropriate for design space exploration of NoC interconnects either. Furthermore, because a RTL model is a low-level implementation, many design decisions must be made before simulation is even possible. For modeling NoC interconnects, the accuracy of the computational cores is less important than that of the communication medium. Therefore, nodes C and D are the most useful for evaluating design trade-offs relating to the interconnect. The *bus-arbitration model* [58] models the architecture of the interconnect, but data is passed using message passing, where approximate timing information may be included. TLM models are often implemented at the bus-arbitration level because the low-level protocols used by the interconnect are not needed, and the system prototypes can be quickly implemented and simulated. The *bus-functional model* contains cycle-accurate communication, and represents a detailed implementation of the communication mechanisms such as flow-control, buffer sizes, routing, etc. Note that the models presented in [58] use buses as the communication medium between processing elements, but the concepts are equally applicable to modeling on-chip networks.

2.2 Networking Basics

The basic principles of interconnection networks are relatively simple [59], the difficulty lies in tuning the many design parameters such that the final implementation meets the requirements of the target application. To meet performance requirements, network designers need to select the proper *topology*, *routing* strategy, and *flow control*:

- The *topology* relates to the physical structure of the network.
- The *routing* strategy is the method by which traffic is directed through the network.
- *Flow control* provides a mechanism whereby access to shared resources is controlled.

Each of the aforementioned design criteria have a large impact on the performance and cost of the final implementation of the system, and so careful consideration of each is warranted. Also, the criteria are not strictly independent of each other. For example, the choice of topology can restrict the choice of available routing strategies.

The remainder of this chapter will present the basic networking concepts needed to understand the work presented in later chapters.

2.2.1 Network Terminology

Much of the terminology used to describe networks borrows from graph theory. This section will introduce the basic notation and terminology used to describe networks as presented in [59].

The topology of a network is described by a set of vertices, or *nodes* $\mathbb{V}$ connected by a set of edges, or *channels*. The set of edges $\mathbb{E}$, where an edge

$$e_{x,y} = (x, y) \in \mathbb{E} \mid x, y \in \mathbb{V} \ , \tag{2.1}$$

connects a source node x to a destination node y. A network topology can

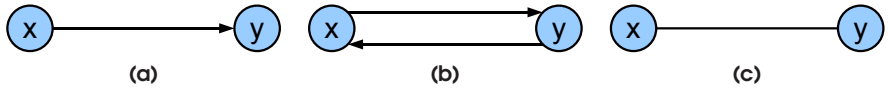


Figure 2.6: Unidirectional and bidirectional channels. (a) A unidirectional channel. (b) Two unidirectional channels. (c) A bidirectional channel is equivalent to two unidirectional channels.

therefore be represented by the graph

$$G = (\mathbb{V}, \mathbb{E}) \ . \tag{2.2}$$

A communication channel $e_{x,y}$, as shown in Figure 2.6a, is characterized by:

- The width w_e, is the number of bits or parallel signals that constitute the channel.
- The frequency f_e, is the rate at which bits are transported over the channel.
- The latency t_e, is the time required for a bit to travel from x to y.

The two unidirectional channels between nodes x and y, shown in Figure 2.6b, are equivalent to the single bidirectional channel shown in Figure 2.6c. Unless specified otherwise, an undirected link between two nodes denotes a bidirectional channel.

The latency is a function of the channel length l_e, and can be described by

$$t_e = v \cdot l_e \ , \tag{2.3}$$

where v is the propagation velocity of the electrical signal over the wire. Lastly, the bandwidth of a channel is described by

$$b_e = w_e \cdot f_e \ . \tag{2.4}$$

Representing network topologies as graphs does not take into account physical implementation issues[3], but it does allow interesting properties to be studied such as [60]:

[3]The complexity of a graph can give insight into the area and wiring requirement of a topology.

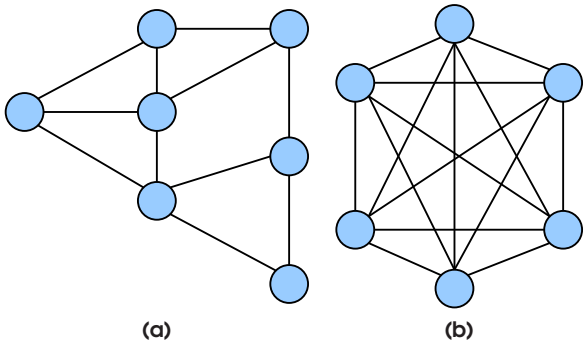


Figure 2.7: Network topologies can be represented graphically using *graphs*. (a) An example of an irregular topology. (b) A regular network that is also fully connected.

- *Node degree*: The number of channels that connect a node to its neighbors.
- *Diameter*: The maximum distance between two nodes in the network.
- *Regularity*: A network is *regular* when all the nodes have the same node degree.
- *Symmetry*: A network is *symmetric* when it looks alike from every node.

Figure 2.7a shows an example of an irregular network topology, while Figure 2.7b shows an example of a regular as well as symmetric topology.

To evaluate the degree of a node, one has to count the number of incoming and outgoing edges. The edge set $\mathbb{E}_x$ for a node x is given by

$$\mathbb{E}_x = \mathbb{E}_{x,in} \cup \mathbb{E}_{x,out} \ , \tag{2.5}$$

where $\mathbb{E}_{\text{x,in}}$ is the input channel set and $\mathbb{E}_{\text{x,out}}$ is the output channel set. The degree of the node x can then be expressed as the sum of the number of input and output channels

$$d_x = |\mathbb{E}_x| = |\mathbb{E}_{\text{x,in}}| + |\mathbb{E}_{\text{x,out}}| \ . \tag{2.6}$$

The node degree is an important metric for on-chip implementation as it affects the planarity of the topology as well as the complexity of the switch implementation.

Another important property of a network is its *bisection bandwidth*, which can be described as the bandwidth of the links that must be *cut* in order to partition the network into two equal parts having roughly the same number of nodes. It is a metric that is often used to compare two network topologies such that a topology with a greater bisection bandwidth will most likely perform better under *uniform* traffic conditions. More formally, a *cutset* is the minimal set of edges whose removal from a graph results in two disconnected graphs. If the vertices belonging to a graph G are partitioned into two sets $\mathbb{X}$ and $\mathbb{Y}$, then the cutset

$$\mathbb{C} = \forall e_{x,y} \in \mathbb{E} \mid x \in \mathbb{X}, y \in \mathbb{Y} \ . \tag{2.7}$$

The bisection bandwidth B can then be expressed as

$$B = \sum_{c \in C} b_c \ . \tag{2.8}$$

Note that a uniform traffic distribution may represent a sub-optimal application mapping. An intelligent mapping that takes advantage of the network architecture can result in significant performance gains. While useful as a metric to quickly compare architectures, the bisection bandwidth does not tell the whole story as there are many other factors that can affect performance.

In multi-node networks such as those shown in Figure 2.7, communication between any two nodes will involve traversing one or more edges. The ordered set of edges

$$\mathbb{P}_{s,d} = \{e_1, e_2, e_3, \ldots, e_n\} \ , \tag{2.9}$$

denotes the path taken through the network for some communication between a source node v_s and a destination node v_d. The number of edges in $\mathbb{P}_{s,d}$ is called the *hop count*, and is used to evaluate different routing strategies. Intuitively, the lower the hop count, the lower the latency will be. If more than one path exists between two nodes v_s and v_d, the minimal path is the one with the lowest hop count. Lastly, the *diameter* of a network is the largest minimal path for all pairs of nodes in the network. The diameter can also be used to compare two network topologies as it gives an idea of how the hop counts and latencies will compare.

2.2.2 Network Topologies

The network architecture, or *topology*, describes the physical organization of the interconnections network. A network topology can be classified as being either *direct* or *indirect*. A node in a network can be a *terminal* node, which acts as a source and sink for data, a switch that routes data, or both. In a direct network, every node acts as a terminal node. In an indirect network, a node is either a terminal or a switch node. The networks shown in Figure 2.7 are direct networks. A direct network can be redrawn as indirect by redrawing each node as two nodes and showing the switch and terminal nodes separately.

Designers of large-scale SoCs must be aware of the advantages and disadvantages of each architecture in order to select an appropriate candidate for their implementations. The metrics that are of interest can be broadly categorized as [61]:

- performance (latency, throughput, cross-section bandwidth),
- energy consumption,
- reliability (error detection and/or correction),
- scalability,
- implementation cost (area).

As discussed in Section 2.1.2, technology scaling is causing the energy consumption of the on-chip network to become an increasingly important design criteria. The goal of macronetworks is to maximize performance without regard for energy consumption, especially for large scale parallel computers where throughput and latency are of primary importance. It therefore stands to reason that a straightforward adaptation of macronetwork implementations for network-on-chip is not appropriate. The problem faced by chip designers is that the design criteria run contrary to one another:

- Minimizing the energy consumption and maximizing performance are usually conflicting goals.

- Increased reliability usually means higher complexity, which results in larger area, degraded performance, and higher energy consumption.

Therefore, designing a NoC interconnect requires searching through a *vast* multidimensional design space. There are many design parameters that can affect system performance and cost, but the design decision that has the largest impact is the choice of topology. The remainder of this section will briefly discuss the basic network topologies that other topologies are derived from.

2.2.2.1 Mesh and Torus Topologies

A *torus* is described as a *k-ary n-cube*, where n is the number of dimensions of the torus, k is the number of nodes in each dimension, and the total number of nodes is $N = k^n$. The simplest torus topology is a single ring, or *k-ary 1-cube*, as shown in Figure 2.8a. The most well known example of a ring-based network topology is the *token ring* [62], which was developed in the late 1970s. Ring networks possess several characteristics that make them well suited for on-chip implementation [59]:

- They have regular physical arrangements that make them well suited for on-chip layout.
- At low dimensions, the physical wires between neighbouring switches is short, allowing high speed operation and low energy usage.
- For local communication patterns, they exhibit low latency and high throughput.
- Depending on the architecture, tori have high path diversity[4].

A direct physical mapping of the torus shown in Figure 2.8a would result in asymmetric channel lengths. As can be seen from the figure, a wraparound link is needed to connect the first and last nodes. The wraparound link must be long enough to span the length of all k nodes, such that

$$l_{k-1,0} = k \cdot l_{i,i+1} \mid i \in \{0, 1, \cdots, k-2\} \ , \tag{2.10}$$

[4]High dimension tori have higher path diversity.

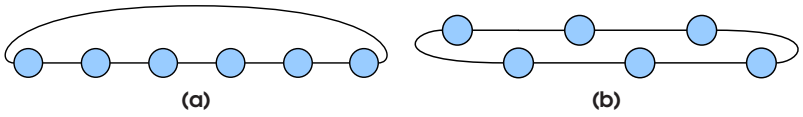


Figure 2.8: Two configurations of a single ring (torus). (a) A straightforward arrangement will require more area and longer wires. (b) A folded torus is more tightly packed and will yield a better physical implementation.

where $l_{k-1,0}$ is the length of the wraparound link, and $l_{i,i+1}$ is the length a normal link between adjacent nodes. The long wraparound channel can negatively impact latencies because of the increased propagation delay over the longer channel. Also, depending on the implementation, the operating frequency of all links may be set to satisfy the physical requirements of the longest link to ensure that the hop delays are constant throughout the network[5], thereby requiring that the network operate at sub-maximal speeds. This problem can be avoided by *folding* the torus as shown in Figure 2.8b. Folding eliminates the long wraparound channel, allowing all channels to be of equal length.

A mesh network is constructed the same way as a torus, except that the wraparound links $l_{k-1,0}$ are omitted. Similar to the torus, a mesh can be described as a *k-ary n-mesh*. Figure 2.9a shows a *3-ary 2-mesh*, which is 3^2 nodes arranged in a two-dimensional grid, and Figure 2.9b shows the equivalent torus network. Note that in the case of the torus, the wraparound links are present for both dimensions. For higher dimensions, the wiring complexity of the torus is higher than that of the mesh, which is why the mesh network is generally preferred over the torus for larger size interconnects. As with the single ring shown in Figure 2.8, the two-dimensional torus can also be folded.

Arbitrary dimensions of the torus and mesh can be constructed by iteratively adding dimensions. The two-dimensional mesh shown in Figure 2.9a can be constructed by connecting three *3-ary 1-meshes*. Going one step further, a three-dimensional mesh network, or *cube*, can be constructed by connecting three *3-ary 2-meshes*, as shown in Figure 2.10. While it is tempting to use the three dimensional cube (and perhaps higher dimensioned tori/meshes) because of the high bisection bandwidth and increased path-diversity, one needs to consider some of

[5]Similar to the maximum clock rate of a digital design being restricted to the critical path of a circuit.

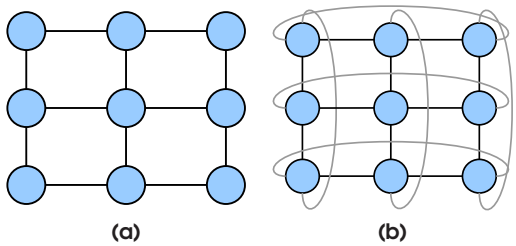


Figure 2.9: Example of two-dimensional mesh and torus networks. (a) A *3-ary 2-mesh*. (b) A *3-ary 2-cube*.

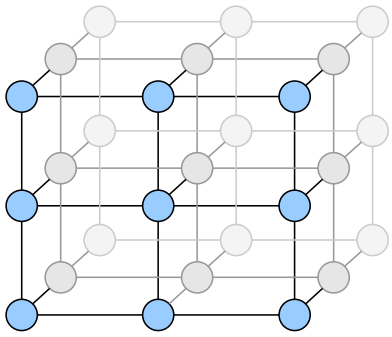


Figure 2.10: A *3-ary 3-mesh*, or *cube* network, consisting of a total of 3^3 nodes.

the costs associated with such a topology. The higher degree switches will be larger due to an increased number of buffers and complexity, thus requiring more area and power. Furthermore, three-dimensional topologies must be mapped onto a two-dimensional surface for chip fabrication, and so the wiring complexity may result in longer wires and larger area requirements. For these reasons, lower dimensioned topologies that can be easily mapped to a two-dimensional space are largely preferred by the NoC research community.

2.2.2.2 Tree Networks

A *tree* network is a hierarchical arrangement of nodes that resembles an inverted tree when drawn as a graph. One of the simplest tree networks is the *binary tree*, as shown in Figure 2.11a, which has the property that each node in the tree has 2 children. A tree consists of a single *root* node at the top-most level of the hierarchy that is connected to the nodes that belong to the level immediately beneath that of the root. A tree can be recursively constructed by connecting

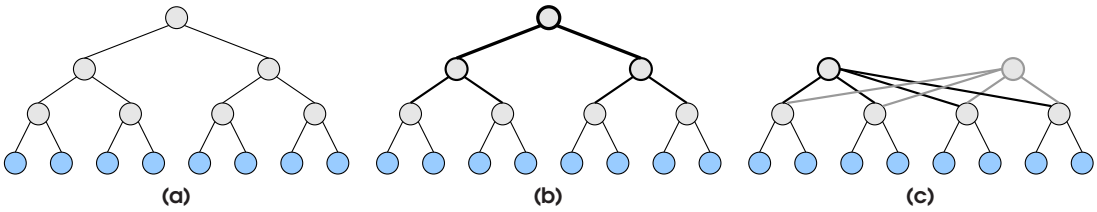


Figure 2.11: Examples of tree networks. (a) A binary tree. (b) A binary fat-tree. (c) A fat-tree with 2 roots.

each node to several children nodes, until the desired depth is reached. The nodes at the bottom of the hierarchy are called *leaf* nodes because they have no descendants; it is these nodes that are the endpoints in the network. Each node in a tree has exactly one parent node, and thus a tree network is *acyclic*.

The main drawback of tree networks is that the nodes at the highest[6] levels can become system bottlenecks. This problem can be alleviated by allocating more bandwidth to the links closest to the root node, thus making some links *fatter* than others; this kind of tree is called a *fat-tree* [63, 64], as shown in Figure 2.11b.

A tree network need not consist of a single root node. A generalized method for recursively constructing trees with arbitrary numbers of roots is given in [65, 66]. Figure 2.11c shows a fat-tree with 2 root nodes — note that multi-root trees are also referred to as fat-trees.

2.2.2.3 Other Topologies

The bisection bandwidth and path diversity of higher dimensioned topologies, such as the *3-ary 3-mesh* shown in Figure 2.10, makes them ideal for interconnecting a large number of cores. However, the high node degree and wiring complexity makes them expensive in terms of area and power for on-chip implementation. The problem has to do with mapping a three (and higher) dimensional structure to a two-dimensional surface.

The *cube-connected cycles* (CCC) [67] topology is a substitute for the n–mesh network that uses interconnected rings to reproduce the structure of a higher dimensioned mesh while using switches of fixed node degree. Figure 2.12 shows an example of the CCC topology where multiple rings are connected such that

[6]The root node and the nodes closest to it.

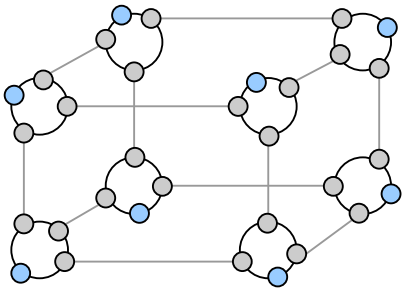


Figure 2.12: Cube-connected cycles.

a *k-ary 3-mesh* topology is reproduced. The CCC topology has the following properties [67]:

- The node degree of all switches is 3.
- Processing time is not significantly increased with respect to that achievable by the *k-ary 3-mesh* (3-dimensional hypercube).
- The structure is more easily mapped to a two-dimensional surface for on-chip implementation.

A *directed cube-connected cycles* DCCC topology [68] that uses unidirectional links has been shown to be even more layout efficient than the normal CCC topology.

An *express cube* is a *k-ary n-cube* network augmented by *express channels* that reduce the path lengths for non-local messages [69]. The *express channels* can be inserted into an existing network without changing the implementation of the switches. Figure 2.13 shows a network that has been augmented by two levels of express channels, where the highest level bypasses the largest number of nodes. A hierarchical express cube has the locality of a torus and a diameter approaching that of a fully connected network [59]. The drawback of express channels is that the required number of channels increases with the dimension, leading to a larger area overhead associated with the extra routers and links.

A *chordal ring* is a ring network in which some/all nodes have an additional link, called a *chord*, to some other node across the network [70]. The chords provide shortcuts that skip over parts of the ring, thus reducing hop counts for long distance traffic. A chordal ring is characterized by the number of nodes

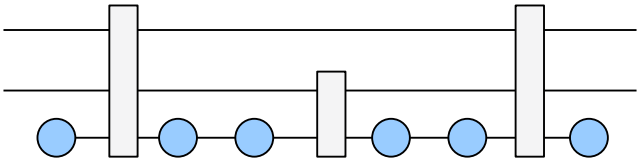


Figure 2.13: Two-level hierarchical express channels.

n, and the *chord length* w. The chord length w is the number of hops that are bypassed by a chord. Figure 2.14a shows an example of a chordal ring with 8 nodes and a chord length of 3. In [70], the chordal ring is described as being symmetric, where each node has a chordal link and has a degree of 3. It is not necessary to construct a symmetric chordal ring; mixing nodes of degree 2 and 3 is permitted [71]. Figure 2.14b shows a multi-ring network, which is constructed by appending rings in an edgewise manner to form a single level topology [71, 72]. Traffic is routed between rings via interface components that connect two rings together. For large multi-rings, long distance traffic can be routed through several rings, which can result in congestion in heavily traversed rings. The disadvantage is that traffic local to a ring can suffer large latency penalties caused by global traffic that is only passing through. An alternative is to use a hierarchical ring configuration, as shown in Figure 2.14c, which is constructed by appending at most one subsidiary ring to each node of a ring, and recursively to each node of each subsidiary ring [71]. A hierarchical ring forms a "tree of rings", where terminal nodes are placed on the rings at the lowest level of the hierarchy. Similar to tree networks, global traffic is routed upwards through the hierarchy.

A *butterfly* network is described as a *k-ary n-fly*, where the network is constructed by connecting n stages of degree k switches. The network consists of k^n source terminals and k^n destination terminals. Figure 2.15a shows an example of a *2-ary 3-fly* network, where the source terminals are located on the left side of the network, and the destination nodes are on the right. The channels in the butterfly network are unidirectional, and flow from left to right.

A variation of a tree network, called the *butterfly fat-tree* [73, 74] is shown in Figure 2.15b. Each switch in the network has four children and two parents. The number of levels required depends on the total number of processing elements and increases logarithmically. The number of switches needed for a network size

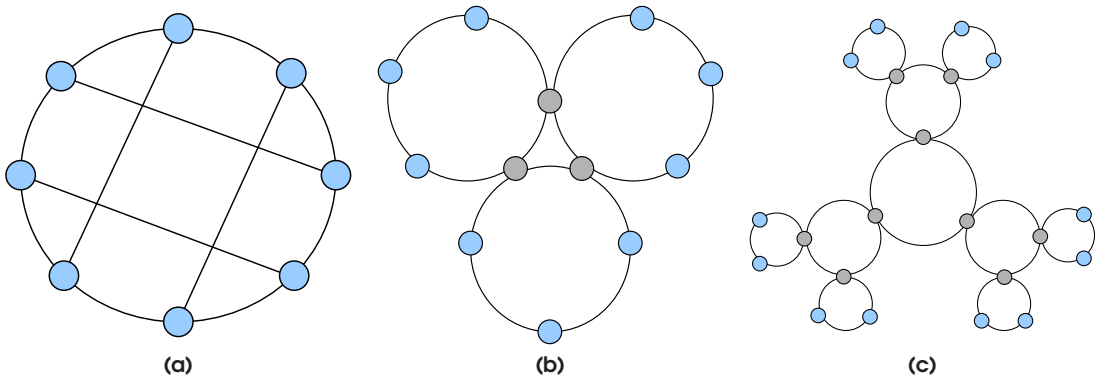


Figure 2.14: (a) Chordal ring with $n = 8$ and $w = 3$. (b) A multi-ring network consisting of 3 interconnected rings. (c) A hierarchical ring of depth 3.

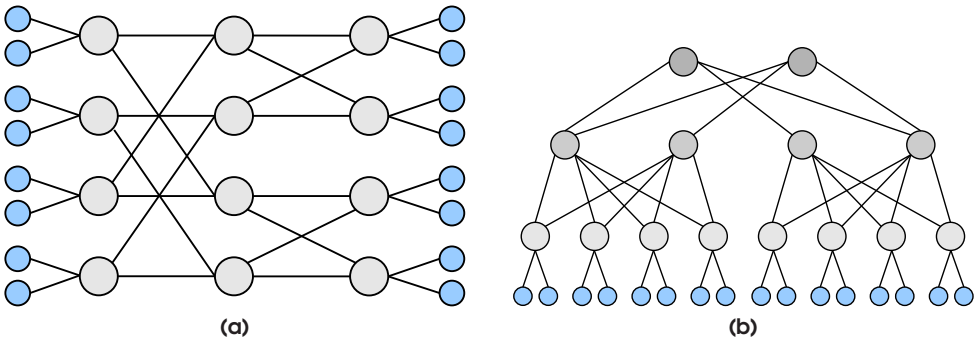


Figure 2.15: (a) A 2–ary 3–fly butterfly network. (b) A butterfly fat-tree (BFT) network.

N converges to $\frac{N}{2}$ as N grows arbitrarily large [74].

2.2.3 Switching, Routing, and Flow-Control

Switching, routing, and flow-control are all network characteristics that affect performance and resource requirements. The following subsections will briefly describe each.

2.2.3.1 Switching

The switching technique determines how data flows through a router, from its input port to its output ports. There are three switching techniques, circuit, packet, and wormhole switching.

In *circuit switching*, a physical path consisting of a series of links and routers is reserved from the sending node to the destination node. The *setup* time refers to the time required to reserve the resources, and the *tear-down* time refers to the time required to release them. Circuit switching has a high initial latency due to the setup time, but it exhibits high throughput because the bandwidth is guaranteed due to the reserved resources. The disadvantage is that during the setup and tear-down times, when data is not being transmitted, the network resources are underutilized.

In *packet switching*, large messages are broken up into smaller pieces called *packets*. Each packets flows through the network independently, possibly along different routes, from sender to receiver. Each packet must be stored in its entirety before being forwarded to the next node on the network, called *store-and-forward*, which can result in large buffer requirements. Since no resources are explicitly reserved, there is the possibility that two or more packets may wish to use the same resources at the same time, called *contention*. When contention occurs, one packet is granted the resource, and all others must wait. The delays caused by contention are variable, and depend largely on the amount of traffic on the network.

Wormhole switching attempts to combine the advantages of circuit and packet switching. In wormhole switching, a packet, also called a *worm* is composed of a series of flits. An example of a worm is shown in Figure 2.16, where

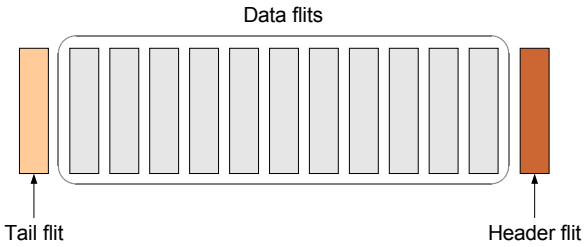


Figure 2.16: Packet, or worm, format for a wormhole routed network.

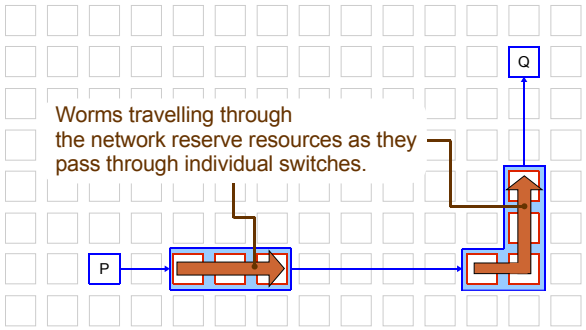


Figure 2.17: Routing two packets from $P \rightarrow Q$ over a wormhole routed mesh. A worm can span several switches.

- the *header* flit contains routing information and reserves routing channels of each switch,
- the *body* flits contain data and follow the reserved channel,
- the *tail* flit will release the channel reservation as it passes through a switch.

The major advantage of wormhole switching is that it does not require the complete packet to be stored in the switch before being forwarded to the next node [59]. As soon as the header flit reserves an outgoing channel it can be forwarded, and the rest of the flits will eventually catch up. Therefore, wormhole switching reduces the delays and buffer requirements associated with packet switching. Figure 2.17 shows an example of several worms traveling through a mesh network. The resources reserved by a worm can span several nodes in the network, which can result in problems such as *deadlock* and *livelock* in certain situations.

2.2.3.2 Routing

Routing involves the selection of a path from a source node to a destination node. Routing is dependent on the topology, where a routing algorithm must choose between alternate paths in the network. The topology dictates the *ideal* performance of the network, and the routing algorithm determines how close to ideal the performance will be [59]. An effective routing algorithm will balance the load across the network channels, and the more balanced the distribution, the closer to ideal the performance of the network will be [59]. Routing algorithms can be classified as:

- *Deterministic* routing always selects the same path between two nodes, even if there are multiple paths.
- *Oblivious* routing does not consider the state of the network when making decisions[7].
- *Adaptive* routing uses information about the state of the network to make routing decisions. These algorithms attempt to circumvent congestion points in the network in an effort to more evenly distribute traffic.

The most commonly used routing algorithms in NoCs are deterministic because they are easy to implement, thus requiring less resources than complex algorithms. Furthermore, deterministic algorithms are easy to make deadlock-free [59]; a necessary and sufficient condition for deadlock-free routing is the absence of cycles in a channel dependency graph [75]. Figure 2.18 shows graphically an example of deadlock caused by a cycle; a properly designed routing algorithm will avoid such a situation. In fact, the *turn model* [76] describes how to avoid cycles by restricting the *turns* a routing algorithm can take, resulting in a provably deadlock free routing algorithm.

A popular deterministic routing algorithm for NoC is *dimension-order routing*, where data is routed in each successive dimension until the destination is reached. For example, for the mesh topology, dimension-ordered routing is called *xy-routing*, where packets are routed in the x dimension first, and then in the y

[7] Deterministic routing is a subset of oblivious routing [59].

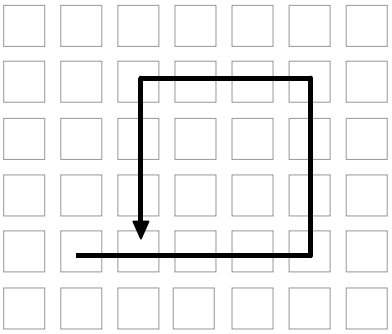


Figure 2.18: Example of cycle causing deadlock in a wormhole-routed mesh network.

dimension[8]. Dimension routing is deadlock free because cycles as shown in Figure 2.18 cannot occur.

2.2.3.3 Flow Control

Flow control determines how the resources in the network are allocated to packets travelling through the network [59]. The goal of flow control is to maximize performance, which is not a simple task. For example if the flow control algorithm is too aggressive, resources may be left unnecessarily idle, which results in wasted bandwidth. Flow control algorithms can be classified as *lossy*, where packets can be dropped when congestion occurs, and *lossless*, where packets are never dropped.

The main task of the flow control algorithm is to manage buffer allocation at a receiving node. The receiving signals the sending nodes via a *backpressure* signal when its buffers reach capacity. Upon reception of the backpressure signal, the sending nodes will stop transmitting data until the signal is de-asserted. There are three types of low-level flow control [59]:

1. *Credit-based*: The sending node maintains a count of available downstream buffers. Each time a node sends a packet, the count is decremented until it reaches zero, at which time the sender cannot send anymore. When the receiver receives a packet, it will send a *credit* back to the sender, incrementing

[8]The *xy*-routing algorithm is also *minimal*, because the deterministic path taken to reach a destination is also the shortest.

the count and allowing it to resume sending packets.

2. *On/Off*: The receiver signals the sending node using a single bit that controls whether it is allowed to send (on) or not (off). The signal changes state depending on the buffer occupancies of the receiver.

3. *Ack/Nack*: The sender node optimistically sends flits over the network. When the receiver accepts a flit, it sends an acknowledgement (ack) back to the sender. If there are no buffers available, the receiver will drop the flit and send back a *negative acknowledgement* (nack). The sender will not discard sent flits until it receives an ack, and if it receives a nack, it retransmits the flit in question. The *sliding window* and *go-back-n* protocols are examples of ack/nack flow control.

Since *on/off* flow control is the simplest of the three techniques, it requires less resources, and is thus often used for NoC implementations.

2.3 Related Research Developments

The topologies proposed for use in NoCs can be classified as either flat or hierarchical [77, 78]. The two-dimensional mesh is the most popular, with *torus* and *folded torus* being derived by connecting the edge elements to the opposite side to form rings. The torus interconnect has a higher bisection bandwidth than the mesh, but also exhibits higher energy consumption [13]. The Intel *Teraflops* [79] research chip consists of 80 cores connected using the two-dimensional mesh topology.

An example of a hierarchical topology is the *fat tree* presented in [12], which can provide low latencies and high bandwidths, but has higher wiring complexities and larger switches (depending on the fan-in/out) as compared to other topologies. Another example of a hierarchical topology is the *butterfly fat-tree* (BFT) [80], where the number of switches converges to a constant depending on the number of levels. Unlike the mesh architecture proposed in [13, 81], where each cell is composed of a PE and a switch, the fat-tree and butterfly fat-tree

place their processing elements at the leaves and the switches at the vertices of the tree [12, 80].

The *Proteo* [82] NoC is a hierarchical network topology that uses a global bidirectional ring to connect several subnets together. The topology of each subnet is chosen to suit local traffic requirements. The architecture presented in [82] has been built using the *Scalable Coherent Interface* (SCI) standardized by IEEE [83]. This standard has garnered some industrial acceptance as a ring-based network topology with its own distributed directory cache coherency protocol, which being cache-based, requires a linked list of cache locations to be maintained to keep the shared memory data coherent. Practical problems with SCI in the past arose exactly because of the need to traverse the network following the linked list for each coherence operation.

Similarly, the *Ring Road* [84] topology was proposed with the idea of using ring switching elements in a manner that provides more bisection bandwidth and eliminates hotspots in the center. There are two types of ring interfaces: those that are on local rings, and those that are on the intersection of a pair of rings. The motivation for this kind of interconnect is avoiding congestion in the center of the area, similar to the rings of roads outside large cities. The Ring Road topology is not hierarchical, and it is not strictly a ring-based architecture; it uses a polar-like mapping of the *xy* coordinate space. Routing is achieved by evaluating the radial distance to the destination node.

In [85], a reconfigurable system was proposed based on a hierarchical mesh interconnection network consisting of *nearest neighbor* connectivity at the lowest hierarchy level, together with horizontal and vertical buses for global connectivity. The architecture presented in [86] takes the opposite approach and uses a hierarchical interconnect to link together multiple bus-based SoCs together.

A parameterizable library of components called *xPipes* which can be used to generate domain-specific heterogeneous architectures is described in [47]. The architectures discussed in [47] are not hierarchical in nature and the problems of growing hop-counts and latencies associated with increasing network size are not addressed, however the authors state that arbitrary topologies can be achieved by their tool. Similarly, ParIS [87] is a parameterizable soft-core router that support the automatic synthesis of NoCs with different sizes and cost/performance cri-

teria. Ongoing work includes modeling the ParIS library at the transaction level to enable rapid performance evaluation [87].

In [69], the problem of large hop counts associated with routing packets over long distances in 2D-meshes is addressed through the use of *express channels* that span multiple hops. The drawback of using express channels is that they require long wires in each dimension and they also increase the router complexity. The express channel approach was originally proposed for interconnection networks with the aim of reducing hop-counts, as such, the area and complexity penalties were of secondary concern. Similarly, *express virtual channels* [88] were proposed as an attempt to adapt express channel for on-chip networks, where packets can bypass nodes virtually by using specially designed virtual channels. The advantage of virtual channels is that the approach does not require the addition of extra dedicated wires. In a similar attempt to deal with the scalability issues of mesh networks, a *hierarchical graph* network structure that consists of several planar graphs interconnected to form a hierarchy is proposed [89]. The hierarchical graph network was shown to outperform the mesh network for several benchmarks despite having a lower bisection bandwidth [89].

The *NUMAchine* [90] multiprocessor, which uses a hierarchical ring interconnect, was designed at the University of Toronto. Good speedups were observed for virtually all multiprocessor benchmarks, in spite of the apparent bisection bandwidth limitation of the rings [90]. The architecture was shown amenable to efficient implementations, and the cache coherence protocol incorporated in NUMAchine exploited well the given topology resulting in a feasible and correct implementation. In addition, the hierarchical rings architecture was shown to compare favorably to the mesh network for certain benchmarks [91]. In [92], the authors explored the performance of several variations of the hierarchical rings, and it was shown that as more hierarchies are added to the system, the constant bisection bandwidth of network begins to affect performance, as is common to all tree-like networks. The suggested solution is to increase the bandwidth of the global ring, or to use a wide and shallow configuration instead of a narrow and deep one.

In [93], the authors present a VHDL implementation of a hierarchical ring network targeted towards a shared-memory architecture whereas in the imple-

mentation presented in [17], components communicate via explicit messaging. Furthermore, in [93], the interface between processing elements (PE) and the interconnect requires that a special instruction be added to the processor in order to allow communication at the software level. The implementation presented in [17] uses memory mapped IO to communicate with the interconnect thus making the implementation more general.

In [94], the impact of process technologies on network energy consumption was evaluated for a range of topologies. The authors characterized the energy consumption of each network, which enabled predictions to be made based on the results obtained by using application traces generated by a actual case study [94].

Polaris [95] is a system-level roadmap for on-chip interconnection networks that aims to guide designers towards a suitable network topology that meets performance and resource requirements; the roadmap is reported to incorporate 7872 NoC design points. An automated toolchain iterates over the available architectures and selects suitable candidates in an automated fashion that can take over 100 hours to run on a typical desktop machine [95, 96]. A more formal approach to finding an optimal solution is to use a linear programming (LP) based technique for synthesis of custom NoC architectures [97]. The technique consists of two stages. First, a floorplan is developed that determines the locations of the routers and cores based on design constraints that can be specified on a per-problem basis [97]. Second, the floorplan from the first stage is used to generate the NoC topology such that the bandwidth requirements of the applications are met [97].

The decision to study hierarchical rings has been further validated by recent commercial products; the IBM *Cell* processor makes use of a ring-based interconnect and achieves high throughput while still maintaining relatively low energy requirements [5, 6]. Furthermore, the *ATI Radeon X1800* series of graphics cards use a ring-based network to achieve high bandwidth memory access. Similarly, the latest sound processor engine from Creative Labs, the *X-Fi*, uses a ring interconnect for the processing units to exchange data.

Chapter 3

Hierarchical Rings Architecture

In the early days of network topology research, ring-based topologies attracted a lot of interest [98] because of their structural simplicity and low resource requirements. In the early nineteen-eighties, the *token ring* [62] and *Ethernet* [99] networks were competing for dominance in the *local area network* (LAN) space. The simplicity of the ethernet protocol eventually won out, but interest in the ring network persisted, especially for shared memory multiprocessors.

The topology of unidirectional rings connected in a hierarchical manner exhibits characteristics that are of importance to NoC implementations. The simplicity of the rings reduces the complexity at each node, which in general results in reduced buffer, area and energy requirements. Furthermore, the topology discussed in this thesis has no global routing so place-and-route will also be more efficient than using global channels as was shown in [17]. The unidirectional nature of the rings reduces the overhead associated with routing and thus results in low latencies and high throughput.

Also of interest is the fact that the hierarchical ring interconnect can be easily partitioned into multiple clock domains, giving designers increased flexibility when tuning design parameters for individual applications. As discussed in [16], distinct clock domains enable the application of dynamic frequency and/or voltage scaling (DVS) techniques for energy optimization. Taking the ideas from [16] one step further, clock throttling can also be applied to an entire sub-mesh, which

can be easily achieved since each mesh can exist in a separate clock domain. In the case of a heterogeneous architecture where certain types of computational units are assigned to a specific mesh, the energy savings can be significant if entire sub-meshes could be powered down during idle periods.

3.1 Hierarchical Ring Topology

The unidirectional ring is the simplest form of *point-to-point* interconnection, which results in a minimum number of links per node and simpler interface hardware [100]. The simplicity of the rings requires a straightforward routing mechanism such that the only decision needed to be made is whether to remove a flit from the ring or to forward it to the next node. Hence, the buffer requirements of each switch are reduced, the maximum speed at which each switch can operate is increased, latencies are reduced, and the point-to-point links between nodes are better utilized. Furthermore, point-to-point connections are so fast that the transmission delays are dominated by the latency of the switches, therefore making fast and efficient routing critical [100, 101].

The hierarchical ring architecture under consideration is an adaptation of the NUMAchine [90] shared memory multiprocessor. The generalized architecture of the hierarchical rings can be described as having k number of *local rings* connected by a central, or *global* ring. Each local ring can consist of l stations (or nodes). While there are many possible configurations, the one shown in Figure 3.1, which consists of four local rings connected to a global ring, each local ring having four stations, was chosen for study[1]. The reason being its symmetry, and the ease with which it can be combined with the traditional mesh architecture to form hybrid architectures, which will be discussed in Chapter 4.

The hierarchical rings architecture is a combination of a *k-ary n-tree* and *k-ary n-cube* network topologies. The local rings can be treated as the leaf nodes in a *4-ary 1-tree*, while the global ring is a single switch, which is an interpretation that bears significance in the development of efficient cache coherence schemes implemented in the NUMAchine multiprocessor [90]. The local rings are themselves *4-ary 1-*

[1]This configuration was also used by NUMAchine [90, 102].

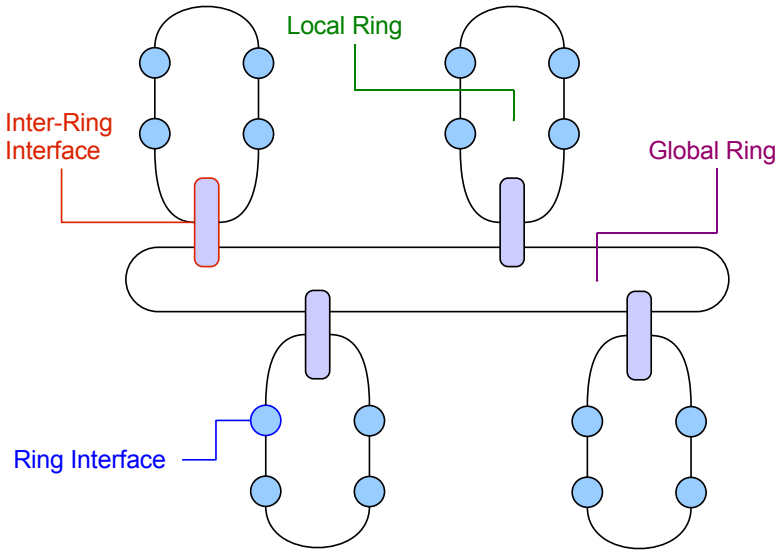


Figure 3.1: An example of a hierarchical ring interconnect with 4 local rings, 1 global ring. Each local ring has 4 ring interfaces.

cube structures. Similar to the other hierarchical structures such as trees, the depth of the network grows logarithmically as a function of the cardinality of the local rings and the total number of processing nodes.

To keep buffering and latencies to a minimum, the rings are *slotted* [100, 101]; fixed size frames, or *slots*, are circulated around the ring. Each slot has a *full* or *empty* bit that denotes whether or not it contains data. When a node wishes to transmit data, it waits until an empty slot arrives, at which point it writes its data and marks the slot as full. Many slotted ring implementations [90, 91, 100, 101] have multi-word slots that can require multiple cycles to transmit between nodes. In the presented architecture, each *flit*[2] is equal to a *phit*[3], which can be forwarded in a single clock cycle. Most NoC implementations follow the convention of having a flit equal to a phit in order to simplify the implementation of the routers and link interface hardware. For example, if a flit were to consist of several phits, then serialization and de-serialization hardware would be required to send a single flit between nodes on the network.

[2]A *flit* is the unit of information at the link layer, one or more words long. A flit can take multiple cycles to transmit between nodes.

[3]A *phit* is the smallest physical unit of information at the physical layer that can be transferred in one clock cycle.

In the architecture shown in Figure 3.1, flits are routed onto a local ring via a *ring interface* (RI). Once on the local ring, a flit can be forwarded to another RI on the same local ring or it can be routed upwards to the global ring via the *inter-ring interface* (IRI). It is important to note that the architecture presented here is *not* a token ring [62] network. There is no special *token* frame that circulates on the ring that regulates access to the network. Furthermore, only one node can transmit data at a time on a token ring network; the hierarchical ring architecture presented here allows multiple nodes on each ring to inject flits simultaneously due to the slotted implementation of the rings. Therefore, the architecture presented here can support many simultaneous parallel communications.

Referring to Figure 3.1, a labeling can be developed such that each ring interface component can be represented by a tuple (s, t),

$$\begin{aligned} \mathbb{ST} &= \mathbb{S} \times \mathbb{T} \\ \mathbb{S} \times \mathbb{T} &= \{(s,t) \mid s \in \mathbb{S}, t \in \mathbb{T}\} \\ \mathbb{S} &= \{0, 1, \ldots, n_{lr} - 1\} \\ \mathbb{T} &= \{0, 1, \ldots, n_{ri} - 1\} \end{aligned} \tag{3.1}$$

where n_{lr} is the number of local rings, and n_{ri} is the number of RI components per local ring. The total number of RI components is $n_{ri} \times n_{lr}$. The number of IRI components that constitute each global ring is equal to the number of local rings n_{lr}. In Figure 3.1, $n_{ri} = 4$ and $n_{lr} = 4$, resulting in a total of 16 RI components. Similarly, each inter-ring interface component can be labeled as a tuple (u, t),

$$\begin{aligned} \mathbb{UT} &= \mathbb{U} \times \mathbb{T} \\ \mathbb{U} \times \mathbb{T} &= \{(u,t) \mid u \in \mathbb{U}, t \in \mathbb{T}\} \\ \mathbb{U} &= \{0, 1, \ldots, n_{gr} - 1\} \end{aligned} \tag{3.2}$$

where n_{gr} is the number of global rings. In the case of Figure 3.1, the number of global rings is 1, so the labelling of each inter-ring interface would be $(0, t_i)$.

The hierarchical ring interconnect can be partitioned into separate clock domains [16, 17]. The fact that the clock rate of the different rings can be independent allows for increased flexibility when tuning the interconnect for specific

applications. For example, the clock rate of the global ring can be greater than that of the local rings so as to increase the bandwidth of the global ring [91]. The use of separate clock domains means that the design can be termed *globally asynchronous, locally synchronous* (GALS) [25]. A GALS system connects synchronous cores that operate at their own speeds and are connected together, which allows proven synchronous design methodologies to be used. The interface between each clock domain must be made clock-independent, either through the use of asynchronous logic or special synchronizer circuitry. The greatest advantages to using GALS systems are [103]:

- The possibility to reuse existing synchronous IP cores.
- Standard synchronous electronic design automation (EDA) tools to design and verify new IP cores.
- The ability to run SoC components at different frequencies, which contributes to power savings.

Basically, the GALS approach has been shown to increase flexibility with regards to clock distribution, yield better performance, and reduced power consumption [104, 105, 106]. The partitioning of each ring into separate clock domains provides the facility for the eventual introduction of dynamic clock throttling [16], which can allow rings to be slowed down or sped up to accommodate changing bandwidth requirements while reducing energy consumption. Other NoC topologies such as the mesh cannot be so easily partitioned because many links would need to cross the clock boundaries. In contrast, the structure of the hierarchical rings lends itself well to partitioning because of the relatively few number of components that cross clock boundaries. In fact, two asynchronous on-chip network architectures have been presented in [107]: a GALS bus, and a ring structure. In [107], the authors state that the ring structure will lead to lower interconnect lengths between modules as well as lower power consumption than that of a bus.

3.2 Switch Implementations

Data is routed through the interconnect via the inter-ring and ring interface components (shown in Figure 3.2), which can be connected to form a 2-D hierarchical ring structure. The GALS implementation of the architecture requires that the problem of *synchronization* and *metastability*[4] be addressed, otherwise the physical implementation may not work properly, or even fail outright. Thus, transferring data between mutually asynchronous clock domains requires safe synchronization [109, 110]. Normally, synchronization can be achieved by using a simple arrangement of several registers connected in series [103, 109], but for NoC implementations, *bi-synchronous* FIFOs, which are driven by both clock signals are a better solution because they provide higher throughput. A bi-synchronous FIFO can be written to from one clock domain, and read from the other. The more common implementation of a bi-synchronous FIFO is the *Gray FIFO*, which uses gray codes [111] to encode/transfer/decode read and write pointers between clock boundaries[5]. An alternative to using bi-synchronous FIFOs is to use asynchronous channels to cross clock boundaries [103, 110], but these approaches are not readily integrated with current EDA design flows.

Current NoC research anticipates a large number of cores being integrated onto a single chip, in addition, deep sub-micron effects will only exacerbate the problems associated with clock distribution [2], resulting in NoC architectures moving towards GALS architectures [113]. The importance of efficient implementations of bi-synchronous FIFOs for GALS NoCs has motivated the search for alternatives to the Gray FIFO [114, 115].

As previously discussed, each ring in the architecture can belong to a different clock domain. Figure 3.2 shows the implementation of the ring and inter-ring interfaces, where the clock boundary is denoted by dotted lines. The ring interface is shown in Figure 3.2a, where the *input* and *output* FIFOs cross the clock boundary, thereby necessitating the use of two bi-synchronous buffers. Under strict

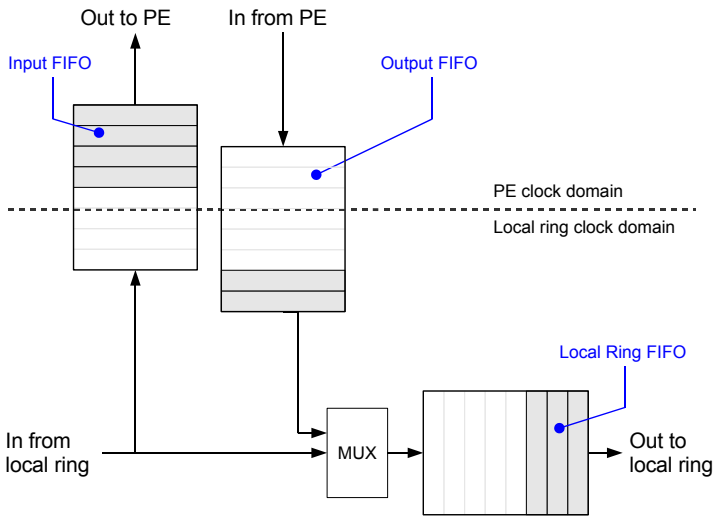
[4]When the timing requirements of a register (e.g. setup and hold times) are violated, the output can hover in an undefined, or metastable, state (i.e. between the values of 0 and 1) [108].

[5]Grey code is a method of encoding binary numbers such that successive values differ only in 1 digit; a property enables the safe transfer of multi-bit values between clock domains [112].

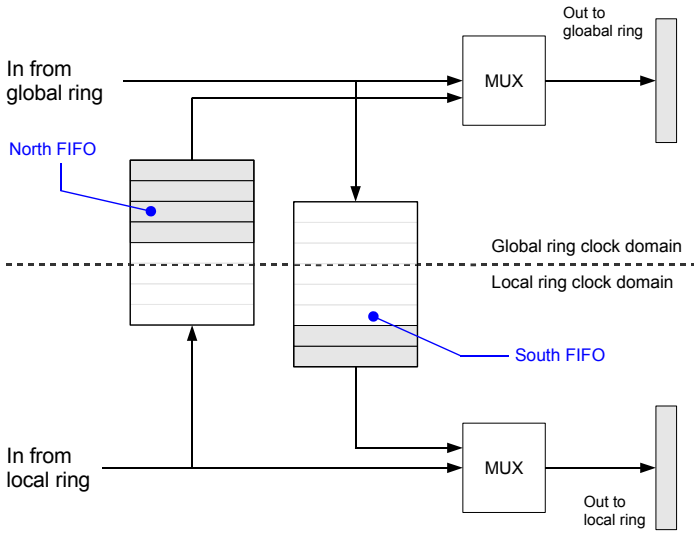
area constraints, the local ring FIFO shown in Figure 3.2b can be replaced by a single register. However, the performance of the network is better when FIFOs are used due to the way in which the flow-control is implemented, which will be discussed in Section 3.3. Figure 3.2b, requires that the *north* and *south* FIFOs be bi-synchronous since the global and local rings belong to different clock domains. Since bi-synchronous FIFOs are more complex than normal synchronous FIFOs, they have a larger area requirement. However, the overhead is acceptable given the flexibility that they afford in terms of enabling the use of a GALS style architecture, running the rings at different speeds, and clock throttling to save energy.

3.3 Routing and Flow Control

The hierarchical rings provide lossless communication through the use of a backpressure mechanism for handling network congestion, which prevents flits from being dropped. At each hierarchy level, a backpressure signal is propagated to prevent injection of new flits until the backpressure signal is de-asserted. While no new flits can be injected when backpressure has been asserted, flits that are already circulating on the rings are allowed to continue propagating. The larger the local ring FIFO (shown in Figure 3.2a), the greater the number of possibly outstanding flits. The result is that while no new flits can be injected into the network while backpressure is asserted, there can be many flits that can continue being delivered. If the FIFOs are large enough, it is entirely possible that the backpressure signal can be de-asserted before all outstanding flits are delivered. This situation would result in a relatively even network utilization because the interconnect would not likely become completely idle when backpressure is in effect. Conversely, if a single register were used instead of a local ring FIFO, the number of outstanding flits would be small. Therefore, all outstanding flits could potentially be delivered while the backpressure signal is still asserted, meaning that the interconnect would be idle until the backpressure signal is de-asserted. The net result is that if the local ring FIFOs are too small, the network may be underutilized if backpressure is asserted often enough.



(a) Ring Interface (RI)



(b) Inter-Ring Interface (IRI)

Figure 3.2: Architecture of the ring interfaces showing FIFOs and clock domains. FIFOs that are intersected by the dotted line, that indicates the clock boundary, are required to be bi-synchronous.

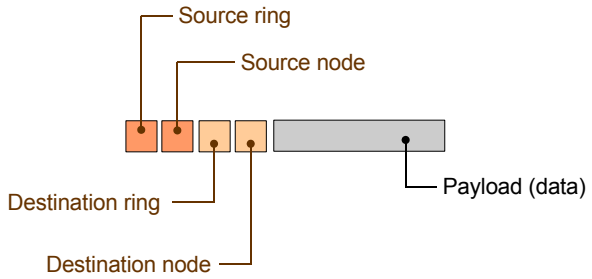


Figure 3.3: The format of a flit in the hierarchical rings. The header is made up of four fields, two of which encode the address of the sender and the remaining two encode the address of the receiver.

The hierarchical ring interconnect was originally designed to be realistically used on an FPGA [17]. The interconnect had to be area-efficient, yet needed to support point-to-point addressing, multi-casting and broadcasting. This was solved by virtue of the hierarchical configuration of the rings coupled with a one-hot encoding of the addresses [17, 90]. The aforementioned one-hot encoding of the addresses results in simple switching logic as routing decisions can be made by simply applying bit-masks to the headers of each flit. A flit destined to a single receiver would have a destination field with only a single bit set, whereas a multi-cast/broadcast flit would have multiple bits set. Routing of multicast/broadcast flits does not require any additional hardware as the bit-masks are applied at each switching node regardless of whether or not a flit is a multicast one.

The use of one-hot encoding simplifies the steps needed to route flits throughout the interconnect. One can infer that the interconnect can process flits quickly and that the switching logic will be small; this is supported by synthesis results shown in Section 8.3.

3.3.1 Flit Format

Each flit consists of a header which is used by the switches for routing, and the payload, which contains the actual data being sent. The actual header consists of four separate fields as shown in Figure 3.3. Using (3.1), each node on the network can be addressed by the local ring to which it belongs coupled with its location on that ring. Since addresses are one-hot encoded, one bit per ring is needed for the ring source/destination field and one bit per source/destination node. The

Table 3.1: The number of bits needed to construct a flit for a hierarchical rings configuration which has n_{lr} local rings and n_{ri} ring interfaces per local ring.

Field	Sub-field	bits
source	ring	n_{lr}
	node	n_{ri}
destination	ring	n_{lr}
	node	n_{ri}
payload	n.a.	n_{payload}

number of bits for each field required for the configuration shown in Figure 3.1 is shown in Table 3.1, where $n_{payload}$ is the number of bits required by the payload data.

A disadvantage of one-hot encoding is that it has limited scalability since it requires 1 bit for every destination node/ring. For example, to address the 16 nodes in the hierarchical rings using the described one-hot encoding, 8 bits are required. On the other hand, encoding the addresses using integers instead requires 4 bits. While integer encoding is more scalable, it does not support multicast/broadcast as well as the one-hot encoding does. In the case were multicast capabilities are not needed, it may be preferable to use integer encoding to save area.

3.3.2 Routing Flits

The logic needed to route data through the hierarchical ring interconnect is simple and can be performed quickly, thereby resulting in high achievable clock rates and low latencies. The ring-interface component shown in Figure 3.2a makes routing decisions by performing the following steps:

1. The first step is to check whether the incoming flit contains data[6].

2. If the incoming flit does *not* contain data, then an outgoing flit is read from the output FIFO and placed on the ring via the ring FIFO.

[6]Recall from Section 3.1 that a slot or frame is equal to a flit in the architecture studied here.

3. If the incoming flit contains data, then it is processed by the ring interface, which will either remove the flit from the ring, or forward it. If the flit is to be removed from the ring, the ring-interface needs to determine whether or not it is a multicast flit:

    - If multicast: forward flit to next node on the ring.
    - If not multicast: read a flit from the output FIFO and place it on the ring.

The inter-ring interface (Figure 3.2b) has two input ports. Flits that enter from the local ring are routed in the following manner:

1. A determination of whether or not the incoming flit contains data is made.
2. If the incoming flit contains data, then the flit is handled differently depending on whether or not it is a multicast:

    - multicast: it is forwarded to the local ring output port, as well as written to the north FIFO.
    - unicast: it is forwarded to the local ring output port.

3. If the incoming flit does not contain data, then a flit is read from the south FIFO and forwarded to the local ring output port.

The routing of flits from the global ring is performed in the same manner, except that flits are written to the south and read from the north FIFOs.

There is a special case of multicast flit which was omitted for simplicity in the description of the steps taken by the inter-ring interface when routing a flit. Consider a multicast flit that enters from the local ring input port, and needs to be routed upwards to the global ring *without* being forwarded to the local ring output. In this situation, the local ring output port would not be written to, resulting in the output port being unused for that clock-cycle. The RTL implementation described in Chapter 8 has been improved so that the output ports of the switches are not idle if there are flits waiting to be routed in the buffers.

The use of one-hot encoding for the addresses means that routing decisions can be made simply by applying routing masks to the header fields. Table 3.2

Table 3.2: Example of unicast, multicast and broadcast flits for the hierarchical rings shown in Figure 3.1 (payload not shown).

type	Source		Destination	
	ring	node	ring	node
unicast	0001	0001	1000	0100
multicast	0001	0001	1000	1111
broadcast	0001	0001	1111	1111

shows examples of flits for the hierarchical rings configuration shown in Figure 3.1, which has 4 local rings and 4 nodes per local ring. The routing decision at each node is made by applying a mask which corresponds to the one-hot encoded value of the node address. The routing mask for each ring interface component is described by

$$m_{s,t} = \{H(s), H(t)\} \tag{3.3}$$

where $H(x)$ is a function which returns the one-hot encoded value of x. The routing decision at each node is made by first evaluating whether or not the flit is addressed to the current node, then determining whether or not the flit is a multicast. The steps required to process a flit are described in Listing 3.1.

To perform routing decisions, a *local* routing function is first defined,

$$R_L(m_{s,t}, f_i) = m_{s,t} \wedge f_{i,dst} \begin{cases} \neq 0 : match \\ = 0 : forward \end{cases}, \tag{3.4}$$

where f_i is the flit being routed. If the result of the routing function is non-zero, the flit has arrived at its destination, whereas a zero result means that the flit needs to be forwarded to the next node. The routing function defined in (3.4) is simply a boolean *and* operation, where the bit corresponding to the current node is tested to see if it is set or not.

In (3.4), the bits corresponding to other nodes are ignored. For the case of *multicast* flits, the bits that were ignored by (3.4) due to the routing mask need

to be considered. The routing function R_M for multicast flits is given by

$$R_M(\neg m_{s,t}, f_i) = \neg m_{s,t} \wedge f_{i,dst} \begin{cases} \neq 0 : forward \\ = 0 : do\ \ nothing \end{cases} \tag{3.5}$$

where $\neg m_{s,t}$ is the complement of the routing mask. For the case of multicast and broadcast flits (as shown in Table 3.2), R_M tests the bits corresponding to addresses other than the current node, hence the use of the complement. If R_M yields a non-zero result, the flit is a multicast flit and must be forwarded to the next node. When forwarding a multicast flit, the bit corresponding to the current node needs to be zeroed as shown on Line 12 of Listing 3.1 so that the flit does not circle around the ring and come back.

The inter-ring interface uses similar routing functions as (3.4) and (3.5), except that it only considers the destination ring field (i.e. the destination node is not needed). When an inter-ring interface injects a flit into either the local or global ring, it performs a zeroing the bit corresponding to its address in the destination ring address field shown in Figure 3.3. The zeroing of the address bit prevents flits from circulating around the local and global ring infinitely.

It should be noted that the multicast capability of the architecture is limited such that sending to differently numbered nodes on different rings is not possible. For example, sending to nodes 3 and 4 on rings 0 and 1 (i.e. sending to (0,3) and (1, 4)), will not work with the one-hot encoding scheme. The solution for this example is to send the same message twice, once for each destination.

3.3.3 Flow Control

As previously mentioned, the hierarchical rings provide *lossless* transmission of data. Flow control is achieved by means of *backpressure* signals which prevent injection of new flits until a sufficient number of outstanding flits have been drained from the interconnect so as to relieve congestion. The lossless nature of the interconnect obviates the need to the network layer to implement a higher level flow control mechanisms such as a *sliding window protocol* in order to handle lost flits. The network architecture of the hierarchical rings guarantees *in-order*

Listing 3.1: Pseudocode showing how a flit is processed at each ring interface (RI) component.

```
1   // check to see if flit has
2   // reached its destination
3   if(R_L(m_{s,t}, f_i) ≠ 0 )
4   {
5     // deliver flit to processing element.
6     fifoIn.write(f_i);
7
8     // check for multicast flit.
9     if(R_M(¬m_{s,t}, f_i))
10    {
11      // unset bit corresponding to current node.
12      f_i = f_i ∧ H(t_i)
13      // forward flit to next node on the ring.
14      ringFifo.write(f_i);
15    }
16  }
17  else
18  {
19    // forward flit to next node on the ring.
20    ringFifo.write(f_i);
21  }
```

delivery of flits, again removing the need for complicated protocols at the network layer. The net result is that sending and receiving of data over the rings requires a minimal amount of processing and buffer space as packet reassembly is simple compared to interconnects which allow packets to be dropped when congestion occurs. The lossless nature means that the interconnect is well suited for shared-memory multiprocessor systems which need to efficiently implement cache coherency primitives [90, 102]. Furthermore, since no control packets are needed to be transmitted back to the sender by the receiver, as is common for more complex protocols[7], the bandwidth that would have been used is available for normal data packets instead.

The hierarchy of rings shown in Figure 3.1 necessitates backpressure signals for each level. The case of congestion occurring at the local ring level is first considered, followed by a description of how backpressure works at the global ring level.

[7]Credit based flow control or *stop-and-wait* are examples of more complex flow control.

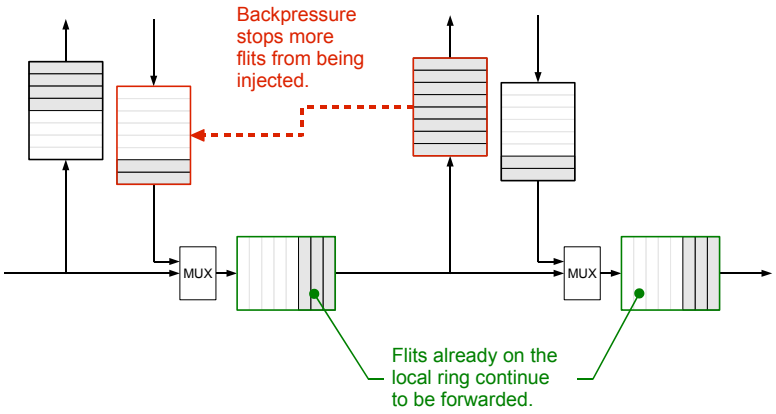


Figure 3.4: Local-ring backpressure mechanism.

3.3.3.1 Local Ring Flow Control

When the input FIFO of the ring interface (shown in Figure 3.2a) is almost full, new flits must be prevented from being injected onto the ring because of the possibility that they cause the FIFO to overflow. To avoid this situation, a backpressure signal is asserted by the ring interface component which is in danger of overflowing its input FIFO as shown in Figure 3.4. The signal is propagated backwards through the local-ring, preventing upstream nodes from injecting new flits onto the interconnect. It is important to note that flits that are already present on the ring[8] are allowed to continue circulating as shown in Figure 3.4, otherwise *deadlock*[9] could arise if the ring were allowed to stall. It should be noted that the backpressure signal is asserted after some threshold has been exceeded; this will be more fully discussed in Section 3.4.

3.3.3.2 Global Ring Flow Control

It can be observed from Figure 3.1 that the system bottleneck is the global ring, thus it is susceptible to congestion when too many nodes send global traffic simultaneously. When the *north* FIFO (shown in Figure 3.2b) is almost full, a backpressure signal is propagated downward to the local ring level to stop injec-

[8]These are flits which are already in the *local ring* FIFOs of each ring interface component shown in Figure 3.2a.

[9]The hierarchical rings are claimed to be deadlock-free in a specific multiprocessor use [90, 102].

tion of new flits. There are three situations that can cause the north FIFO of a inter-ring interface component to fill up:

1. Multiple nodes on the local ring send global traffic simultaneously at a faster rate than the global ring can process them.

2. A backpressure signal that was first asserted on another local ring is eventually propagated upward through the global ring and then back down to the other local rings, as shown in Figure 3.5a.

3. A backpressure signal is asserted by a downstream inter-ring interface due to the south FIFO being almost full, as shown in Figure 3.5b. While the signal is asserted, flits are prevented from being injected into the global ring, causing the north FIFOs to eventually fill up.

As with the local ring flow control, flits are allowed to continue propagating through the interconnect so as to avoid deadlock and allow the FIFOs which caused the backpressure signal to be drained. In the case of a global backpressure signal, the south FIFO is allowed to continue injecting flits into the local ring, thereby relieving congestion on the local ring.

To summarize, packet loss is avoided through the use of backpressure signals which are asserted when the capacity of certain FIFOs in the ring interface (Figure 3.2a) and inter-ring interface (Figure 3.2b) components reach a pre-determined level. Deadlock is avoided by allowing flits that have already been injected to drain. This is an important point because traffic is *never* halted due to blocking as can happen in other network topologies such as the mesh.

3.3.3.3 Backpressure Implementation Choices

The backpressure signals used for flow control in the hierarchical rings can be implemented in two ways. The first way is to use a single shared signal for each ring, such that when one node triggers backpressure, all stations must stop sending. While simple, this approach may result in sub-optimal performance because the other nodes on the ring may not be congested. A second approach is to have 1 backpressure signal for each destination on the ring. While slightly

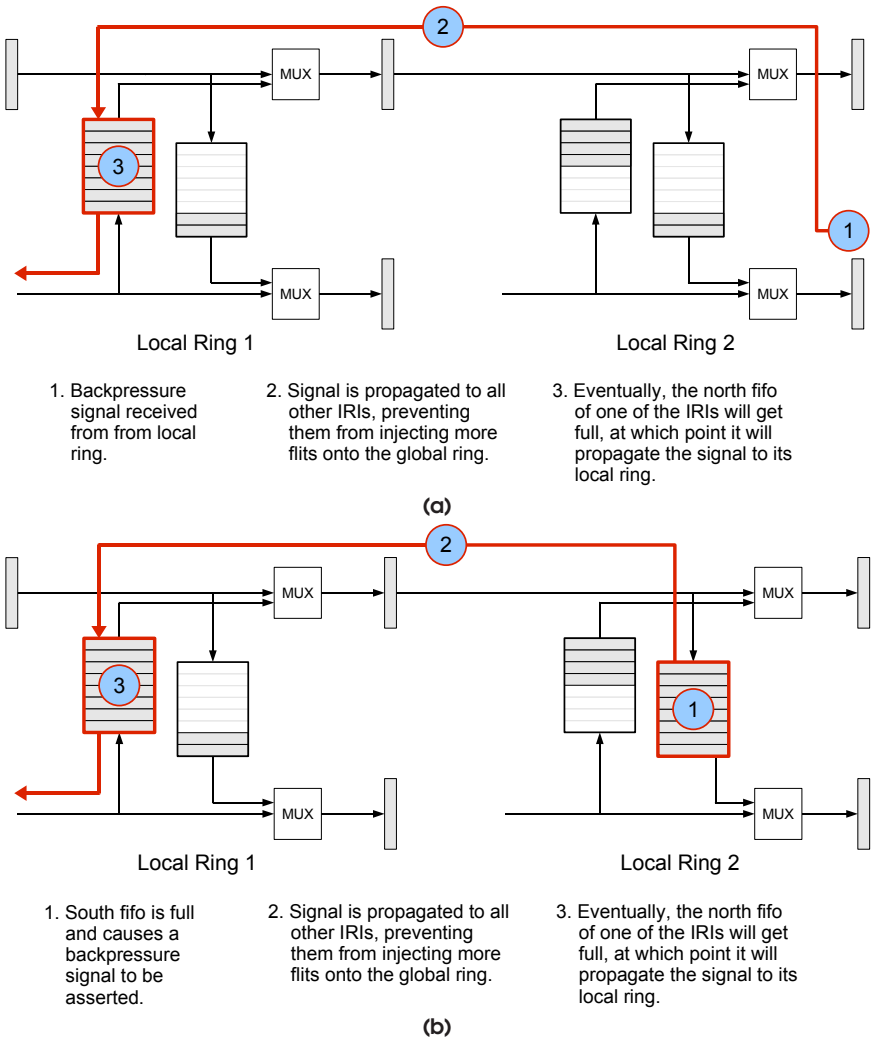


Figure 3.5: Propagation of backpressure signal through the global ring. When asserted, the backpressure signal causes the inter-ring interface components to stop injecting flits onto the global ring. Flits that are already present in the south FIFO are allowed to drain onto the local rings. (a) A backpressure signal is first asserted on a local ring. The signal is then propagated backwards to the global ring. (b) A backpressure signal is caused by the south FIFO in the inter-ring interface reaching capacity. The signal is propagated backwards to all inter-ring interfaces connected to the global ring.

more costly in terms of area (i.e. increased number of control signals between nodes), the increased flexibility makes the tradeoff interesting. For example, when a node asserts backpressure, all other nodes on the ring can continue injecting new flits as long as they are not destined to the congested node, thus achieving better network utilization.

3.4 Buffer Sizes

The flow control mechanism described in Section 3.3.3 causes backpressure signals to be asserted when the occupancy of certain FIFOs reaches a pre-determined threshold. When the backpressure mechanism has been triggered, the flits that are already inside the network are allowed to propagate. This has implications on the FIFO sizes and thresholds such that they must satisfy certain conditions in order to avoid packet loss. The case of a single local ring will be discussed, followed by a discussion of how the global ring affects the FIFO requirements of the local rings.

3.4.1 Local Ring Buffer Requirements

At the local ring level, the input FIFO of each ring interface is required to be large enough to accept outstanding flits on the ring after a backpressure signal has been asserted. In the worst-case, all of the outstanding flits would be destined to the station which cause the backpressure signal. The following describes the worst-case scenario on the local ring:

- All the stations on the ring send data to a single receiver station.
- The input FIFO of the receiver station reaches its *full threshold*, triggering the backpressure signal which prevents the other nodes from injecting more flits.
- The backpressure signal takes some time to propagate to all the nodes, during which time each node can inject one flit for every clock cycle of delay.

- The flits which were already on the ring, and were injected due to the propagation delay of the backpressure signal, will continue to circulate towards the receiver node.

From the aforementioned scenario, it is clear that once the backpressure signal is asserted, the input FIFO of the ring interface must have enough space left to store *all* outstanding flits which may arrive while the backpressure signal is high. The size of the input FIFO $|F_{in}|$ of the ring interface can therefore be given as

$$|F_{in}| \geq ((n_{ri} - 1) \cdot |F_{ring}|) + F_{in,full} + \sigma_{local} \ , \tag{3.6}$$

where $|F_{ring}|$ and $|F_{in,full}|$ are the sizes of the ring and input FIFOs; $F_{in,full}$ is the full threshold of F_{in}, and n_{ri} is the number of ring interfaces per local ring from (3.1). In (3.6), the last term σ_{local} represents the number of extra flits injected due to the propagation delay of the local backpressure signal, which will be discussed further in Section 3.4.3. Note that in (3.6), 1 is subtracted from n_{ri} because the ring interface that generated the backpressure signal is not counted.

3.4.2 Global Ring Buffer Requirements

In Section 3.4.1, the presence of a global ring, and hence a inter-ring interface on the local ring was neglected. In the case of global backpressure, the sizes of the *south* and *north* buffers of the inter-ring interface must be carefully adjusted so as to avoid flits from being dropped.

When a backpressure signal is propagated downwards onto a local ring by a inter-ring interface, the ring interfaces are prevented from injecting new flits into the ring. In this situation, the worst case would be when all outstanding flits are global flits and must travel through the inter-ring interface. Therefore, its north FIFO must have sufficient room left to accommodate receiving all of the outstanding flits that are stored in the ring FIFOs of *each* ring interface on the local ring, as well as the flits that are already in the FIFO. The size of the north buffer is given by

$$|F_{north}| \geq (n_{ri} \cdot |F_{ring}|) + F_{north,full} + \sigma_{local} \ , \tag{3.7}$$

which is similar to (3.6), except that *all* of the ring interface components have been accounted for.

The size of the south FIFO of the inter-ring interface is minimally constrained to be large enough to hold all flits that are circulating on the global ring. From the architecture of the ring interface shown in Figure 3.2b, the only buffering that occurs on the global ring occurs at the output register of each switch, so there can therefore be a maximum of n_{iri} flits on the global ring. Simply stated, the size of the south FIFO is given by

$$|F_{south}| \geq n_{iri} + \sigma_{global} + \delta \ , \tag{3.8}$$

where δ represents the number of flits propagated onto the global ring before the backpressure signal arriving from a local ring crosses the clock boundary between the local and global rings. The value of δ is implementation dependent.

3.4.3 Pipelined Flow Control

In Section 3.3.3, the backpressure signal is a globally shared signal which can be implemented in combinational logic[10]. Note that each ring has it's own global backpressure signal. As discussed earlier, propagation of the backpressure signal requires synchronization circuitry when crossing clock boundaries (i.e. between rings). Since this signal is globally shared by all nodes on a ring, the wire length will be longer than the point-to-point connections between each node. Basically, each backpressure signal must span the breadth of a ring. A drawback to this approach is that long wires require more energy to drive and can also introduce a critical path into the circuit, which can negatively impact its maximum operating frequency. Fortunately, the architecture of the hierarchical rings mitigates the length of the backpressure signals because each ring has its own signal which is *locally* global. Since the signals are registered when crossing clock boundaries, the wire length of any backpressure signal will not exceed the length required to span a *single* ring.

For the implementation of the global backpressure signal, the number of flits

[10]The signal can be implemented using an *or* gate.

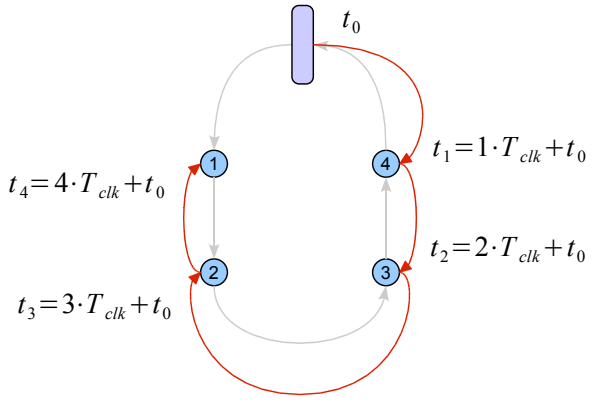


Figure 3.6: Delay of the pipelined backpressure signal when propagated synchronously through a local ring, where t_0 is the time the backpressure signal is asserted. The signal takes $\Delta t_{4,0} = (t_4 - t_0)$ time to propagate from the inter-ring interface to the first ring interface.

injected due to the propagation delay through the rings is given by

$$\sigma_{local} = n_{ri} \cdot \frac{\Delta t_{bp,l}}{T_l} \ , \tag{3.9}$$

$$\sigma_{global} = n_{iri} \cdot \frac{\Delta t_{bp,g}}{T_g} \ , \tag{3.10}$$

where Δt_{bp} is the time is takes for the backpressure signal to be read by the ring and ring-interface components.

An alternative to using global signals in each ring is to *pipeline* the signals by routing them through point-to-point connections using synchronous logic. Thus, the maximum length of the wires needed to physically route the signals becomes bounded by the maximum distance between consecutive ring interface components on a ring. A pipelined implementation results in a delay of several clock cycles for the backpressure signal to travel all the way around the ring as shown in Figure 3.6. Since each switch on a ring will receive the signal at different times, the number of flits that an individual switch will inject is a function of its distance to the node from which the backpressure signal originated. The total number of flits injected due to the propagation delay is thus the sum of the packets injected by each switch, and can be expressed as a more general form

Table 3.3: The maximum number of flits injected by each node n_i on the ring due to the propagation delay of the pipelined backpressure signal.

i	σ_i
1	4
2	3
3	2
4	1

of (3.10) as

$$\sigma = \sum_{i=0}^{n-1} \left(\frac{t_{i+1} - t_0}{T} \right) , \tag{3.11}$$

where n is the total number of ring-interface components or inter-ring interface components on the ring, depending on whether or not the ring is local or global. For example, the number of flits injected due to the propagation delay of the pipelined backpressure signal shown in Figure 3.6 can be calculated using (3.11). Table 3.3 shows the maximum number of flits injected by each node on the ring, giving a total of $\sigma = 10$. Therefore, if a pipelined implementation of the backpressure signal is desired, the input buffers of the ring-interfaces or the north FIFO of the inter-ring interface must be able to account for the 10 extra flits which may be injected *after* the backpressure signal is asserted.

3.5 Planarity of Hierarchical Rings

A compelling reason for using hierarchical rings for on-chip interconnections is their *planarity*. Simply put, a graph is said to be planar if it can be drawn on a plane with no intersecting edges. The hierarchical ring configuration shown in Figure 3.1 has no intersecting edges, and is thus planar. The implication is that planar structures can be tightly packed onto two-dimensional surfaces [71]. The planarity of the hierarchical rings will therefore result in area efficient layout when mapping to hardware.

In general, higher node degrees will result in non-planar topologies for large

enough networks[11]. Node degrees greater than 3 will result in higher dimensioned topologies, such as the 3D-mesh discussed in Section 2.2.2.3, which will not be planar.

The *PIGALE* [116] graph editor is a freely available program that is primarily concerned with planar graphs. The planarity of the hierarchical rings was confirmed using several algorithms [117, 118, 119] that are implemented by PIGALE. In addition, the *3-ary 3-mesh* shown in Figure 2.10, was shown to be non-planar by PIGALE.

3.6 Improving the Hierarchical Rings

The two-level hierarchical rings architecture shown in Figure 3.1 has the characteristic that the global ring is the system bottleneck. It has been shown quantitatively in [120] that the global ring saturates before the local rings when traffic patterns exhibit low locality. It was further shown in [120] that the configuration of the hierarchical rings can greatly affect system performance, and speeding up the global ring by a factor of two [91] can yield a significant reduction in latencies for global traffic[12]. While increasing the bandwidth of the global ring does improve system performance, the architecture of the network remains the same.

The *torus ring* [121] architecture is presented as an alternative configuration of the hierarchical rings such that each local ring is connected to two neighboring rings via inter-ring interface components. Global communication is achieved by routing traffic through consecutively connected local rings until the destination is reached. In [121], the authors state that a 5% reduction in latency was achieved for communication patterns exhibiting high locality. The disadvantage of the torus ring configuration is that the hop count for global traffic is increased, and more significantly, global traffic must travel through local rings. In the hierarchical ring architecture, global traffic is routed upwards through the global ring, thereby freeing up bandwidth in the local ring which can be used for local communication. In the torus rings architecture, since global traffic travels through

[11] A network consisting of very *few* high-degree nodes may be planar.

[12] The bandwidth of the global ring can be increased two ways: either speed up, or increase the width of the ring.

local rings, less bandwidth is available for local traffic. Furthermore, global traffic that travels through several local rings will impact the available bandwidth of each local ring it passes through. In the hierarchical rings architecture, only the sending ring and receiving ring see global traffic. In contrast, each local ring that is along the path of a global communication will be impacted in the torus ring. A possible improvement to both architectures would be to combine the hierarchical rings configuration with the torus rings architecture, thereby harnessing the desirable properties of each.

With a view to improving performance, two alternative configurations of the hierarchical rings are presented in the following subsections.

3.6.1 Enhanced Hierarchical Rings

While trying to optimize the performance of the RTL[13] implementation of the hierarchical rings, it was realized that the backpressure mechanism can result in unnecessary delays for local and global traffic. Recall from the discussion of the backpressure mechanism in Section 3.3.3, when the occupancy of certain FIFOs in the ring and inter-ring interface components reach a certain threshold, new flits are prevented from being injected until enough flits have been drained from the network. Furthermore, recall from Section 3.4 that the worst case buffer requirement is calculated based on the assumption that when a backpressure signal occurs, *all* outstanding flits that are circulating on a local ring can be destined for a *single* destination node. Hence, the worst case FIFO sizes are as defined in (3.6), (3.8), and (3.7).

It was observed during RTL simulations that ring utilization could be increased by taking advantage of the properties of local and global traffic, namely:

- Local traffic travels around a ring until it reaches its destination node, which is *on the same ring*.
- Global traffic gets routed onto the global ring by the inter-ring interface. Once on the global ring, the data is routed downwards onto a *different* local ring.

[13]The hierarchical rings were implemented in VHDL.

The important distinction between local and global traffic is that for local traffic, only a single local ring in involved, whereas for global traffic, the global ring and two local rings are involved. This distinction is key to improving the performance of the interconnect. Careful inspection of the backpressure mechanism reveals that global and local traffics can be treated differently depending on the source of a backpressure signal. The following definitions will be used to differentiate between sources of backpressure:

- *Local backpressure* occurs when a backpressure signal is asserted by a ring-interface component as described in Section 3.3.3.1.
- *Global backpressure* occurs then a backpressure signal is asserted by a inter-ring interface due to congestion on the global ring, as discussed in Section 3.3.3.2.

Consider the following two examples which illustrate the two types of backpressure mechanisms:

1. *Local*: A ring interface component asserts the backpressure signal because its input FIFO is full, thereby preventing all other nodes on the ring from injecting new traffic.
2. *Global*: An inter-ring interface component asserts a backpressure signal because its north-fifo is nearing capacity (see Figure 3.2b and Figure 3.5), and thus prevents all nodes on its local ring from injecting traffic on the local ring.

In the case of local backpressure, nodes on the same local ring are prevented from injecting new traffic; if they are sending to the congested node, there is a possibility of buffer overflow (dropped packets). Since it is known that the occupancy of the north FIFO belonging to the inter-ring interface is unaffected by local traffic, there is no danger of buffer overflow if the nodes on the ring were allowed to send global traffic while the local backpressure signal is asserted. The key point here is that sending global traffic to another ring can *never* cause the input FIFO of a ring interface on the same ring to overflow. Alternately, when global backpressure occurs, nodes on the local ring are prevented from injecting

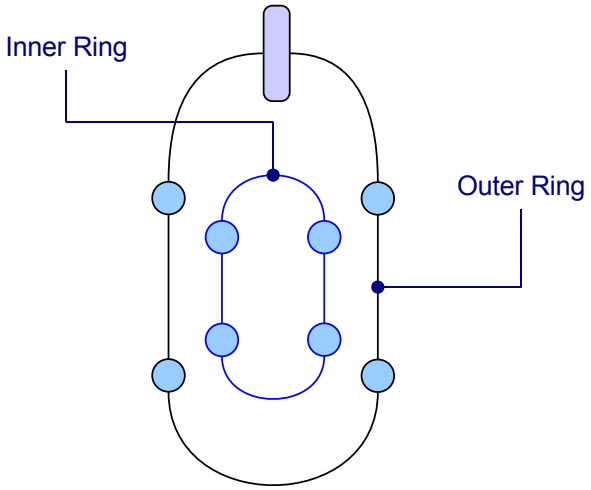


Figure 3.7: The figure shows 1 local ring which has been enhanced by the addition of a second *inner* ring for local traffic.

new traffic because of the possibility of causing the north FIFO of the inter-ring interface to overflow. Since it is known that local traffic does not impact the occupancy of the north FIFO, local traffic can be allowed to be injected without fear of a buffer overflow occurring at any of the ring interfaces on the same ring. In short, when global backpressure has been asserted, nodes on each ring can be permitted to send locally without risking packet loss due to buffer overflow.

The *enhanced* backpressure mechanism can be implemented thusly:

- When *local* backpressure occurs, global traffic can be permitted to be injected.
- When *global* backpressure occurs, local traffic can be permitted to be injected.

The two cases of backpressure can be thought of as providing two channels of communication, one for local, and one for global traffic. Figure 3.7 shows how a local ring can be enhanced by adding an *inner* ring for routing global traffic. A straightforward implementation would be to simply instantiate four extra ring interfaces and connect them to form a ring which bypasses the inter-ring interface. Global traffic can then be routed using the *outer* ring.

The straightforward implementation of the enhanced architecture by adding an extra ring, has the advantage of doubling the local bandwidth. However, the

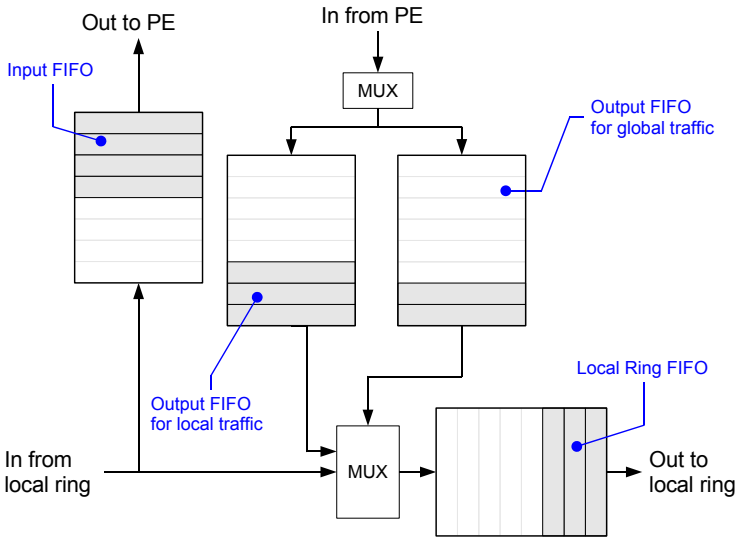


Figure 3.8: The ring interface shown in Figure 3.2a has been enhanced with an extra output FIFO so that global and local traffic can be sent independently of each other.

resource requirements are also doubled. Furthermore, the actual utilization of each ring will depend on the ratio of local to global traffic. For instance, it is possible that the outer ring used for global traffic can be underutilized if there is not much global traffic. As an alternative to instantiating extra components to form the inner ring, a *virtual* inner ring can be formed by adding a single extra FIFO to the ring interface component. Figure 3.8 shows the ring interface component from Figure 3.2a that has had an extra output buffer added to it. When flits arrive from the processing element, they are categorized as global or local, and stored in the appropriate buffer (inner for local, outer for global). Under normal operation, a round-robin scheduling can be used to share the available network bandwidth evenly between global and local traffic. Alternatively, priority can be given to one type of traffic by using weighted round-robin scheduling. When a global backpressure signal is asserted, flits can continue to be read from the global output FIFO and place on the ring via the ring FIFO. Similarly, when a local backpressure signal is asserted, flits continue to be read from the outer ring FIFO and written to the ring FIFO. Thus, the decoupling of the local and global backpressure signals can be achieved at the cost of one extra FIFO per ring interface (Figure 3.8).

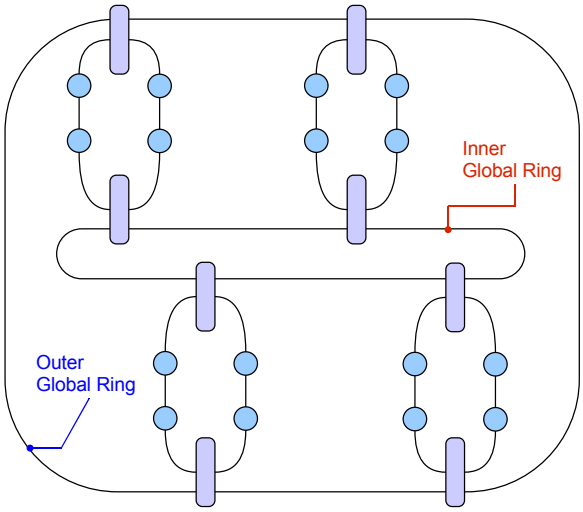


Figure 3.9: Hyper-ring configuration of the hierarchical rings. Since there are two global rings, half the RIs send over the inner global ring, and the other half send over the outer global ring.

3.6.2 Hyper-Ring Configuration

Similar to the hierarchical rings topology, the *hyper-ring* [122, 123, 124, 125] topology consists of multiple rings connected together to form a hierarchy. However, unlike the hierarchical rings, the hyper-ring is not constructed in a tree-like manner where the root of the tree is a global ring. Instead, the hyper-ring can be constructed recursively by adding rings in each dimension. Higher dimensioned hyper-rings will not be planar, and thus will be difficult to map to hardware efficiently. As such, it is best to restrict the hyper-rings to a few dimensions. Interestingly, the two-dimensional hyper ring shown in Figure 3.9 is similar to the hierarchical rings, except that is has one added global ring. The second global ring serves to increase the bisection bandwidth of the hierarchical rings, as well as adding path diversity. Therefore, the hyper-ring architecture is an attractive alternative to the hierarchical rings for applications that require more bandwidth.

The hyper ring architecture is discussed further in the appendices at the end of this thesis. First, a comparison of SystemC simulation and ASIC synthesis results of the hierarchical and hyper ring architectures is presented in Appendix A. Second, wormhole routed versions of both the hierarchical and hyper ring architectures that support virtual channels are presented in and Appendix B. Results

show that the hyper rings outperform the hierarchical rings while incurring a reasonable area penalty of approximately 17%.

Chapter 4

Composite Ring/Mesh Architectures

In a mesh network, express channels can help reduce the latencies associated with routing global packets [69]. The major disadvantage of this approach is that express channels are needed in both the x and y directions. Therefore, the routing complexity of this topology increases with the number of express channels; a problem which is further exacerbated as the mesh size gets larger, and longer express channels are needed. In order to simplify global routing, the use of rings for routing global traffic is proposed. Instead of modifying the existing mesh interconnect to support the rings, bridge (or gateway) components are used to connect the different architectures together, thus obviating the need to increase the complexity of the mesh interconnect. The circular structure of the rings implies that traffic can traverse the x and y directions simultaneously, hence global traffic need not come back down to the lowest level in the hierarchy in order to be routed in a different dimension. Also, the mesh and hierarchical ring architectures lend themselves well to planar layout, and the combination of the two does not require the third dimension routing resources as required by a 3D mesh (cube).

In Section 2.3, several tree-like hierarchies were briefly discussed. When comparing hierarchical architectures to a 2D-mesh architecture, it can be seen that the hop count for global packets is smaller on average, but that the aggregate bandwidth also becomes reduced because of the bottlenecks associated with routing

global packets through a small number of switches.

In [18], a hybrid ring/mesh architecture was proposed, where a large mesh was divided into several smaller meshes that were connected using a hierarchical ring interconnect. As the hybrid architecture has a smaller bisection bandwidth than a normal mesh, it can still potentially suffer from congestion under heavy load, especially if the application mapping does not exploit locality. The augmented architecture [19] addresses this issue by keeping the mesh intact while *augmenting* it with the hierarchical ring interconnect to reduce the latency incurred by global traffic, while still maintaining the high throughput that meshes exhibit for local traffic. Also, instead of using a hierarchical ring interconnect, the effect of adding individual rings to a mesh that is kept intact is explored.

The various proposed architectures that combine the rings with the mesh network are globally referred to here as *composite* architectures. The composite architectures consist of two and three-level hierarchies, where the lowest level uses wormhole-routed mesh(es) for local routing and either a two-level hierarchical ring interconnect or unidirectional ring(s) for global routing. In contrast to other approaches that introduce global wiring [12, 69], an approach that adds its own complexities not unlike the FPGA routing, the mesh architecture is left unmodified. Processing elements in certain cells are replaced by bridge components that link the two interconnects together. One limitation of the NUMAchine architecture is that the total number of nodes will be kept modest for a fixed number of hierarchy levels, since buses are used at the lowest level of the hierarchy in [90]. By replacing local buses with meshes, more processors can be accommodated for the same hierarchical ring.

The following subsections describe in more detail the individual components which make up the composite architectures. It should be noted that the configuration being used here is one of several possible architectural variations. For example, for smaller mesh sizes, a simple unidirectional ring can be used for the global interconnect, or even several unidirectional rings can be used, similar to what was done in the IBM Cell processor.

4.1 Wormhole Routed Mesh

A mesh network consists of a number of tiles arranged in a two-dimensional grid, where each tile can be labeled as a tuple (x, y) such that

$$\begin{aligned} \mathbb{X}\mathbb{Y} &= \mathbb{X} \times \mathbb{Y} \\ \mathbb{X} \times \mathbb{Y} &= \{(x, y) \mid x \in \mathbb{X}, y \in \mathbb{Y}\} \\ \mathbb{X} &= \{0, 1, \ldots, M-1\} \\ \mathbb{Y} &= \{0, 1, \ldots, N-1\} \end{aligned} , \tag{4.1}$$

where M and N represent the width and height of the mesh, respectively. The length of the path p_i between two nodes $P : (x_p, y_p)$ and $Q : (x_q, y_q)$ can be described by

$$f_{hops}(p_i) = |x_p - x_q| + |y_p - y_q| \ . \tag{4.2}$$

The maximum number of hops that a packet will travel is when a packet is sent from a corner node to its diagonal opposite. For example, if node $P : (0, 0)$ sends a packet to node $Q : (M-1, N-1)$, then the worst-case hop count can be expressed using (4.2) as

$$H_{worst} = f_{hops}(P \rightarrow Q) = [(M-1) - 0] + [(N-1) - 0] \ , \tag{4.3}$$

or $2(N-1)$ for when $M = N$. As the size of the mesh increases, H_{worst} may result in latencies that are unacceptably large with regards to application requirements.

The switch used in the mesh portion of the NoC simulation model has five input ports, all of which have input FIFOs to store incoming packets. The output ports are unbuffered and are connected directly to their neighbors. Flow control between nodes on the mesh is achieved through the use of "on-off" flow control [113][1]. The hybrid architecture in [18] was designed to route flits independently in order to interface easily with the hierarchical ring interconnect. While simple, the hybrid architecture is not easily scalable as the overhead required for addressing information becomes large with increasing mesh sizes. The aug-

[1]When an incoming buffer is full, the flow control signal is asserted (turned on), and the downstream neighbour stops sending until the signal is de-asserted (turned off).

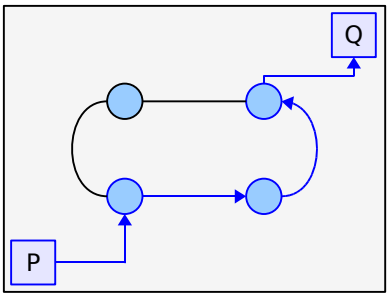


Figure 4.1: A single express ring has been added to a normal mesh. The number of hops for the communication $P \rightarrow Q$ can be reduced by travelling through the ring instead of only through the mesh.

mented architecture presented here makes use of a *wormhole* routed mesh, where a package of data travelling through the mesh is called a *worm*.

A worm is a collection of flits whereby the first flit, called the *header flit*, contains the addressing information of the packet. As the header flit passes through a switch, it obtains a lock on the output port through which it is being routed so as to prevent interleaving with other worms. The next flit(s) after the header is termed *body* flit(s), which contain the actual data being transmitted. The last flit in a worm is called a *tail* flit, which releases the port that was locked by the header flit after it gets routed through it.

4.2 Express Rings

In [69], *express channels* in each dimension were used to bypass sections of the mesh and reduce the hop counts for long distance traffic. The original idea has been adapted here such that unidirectional rings, or *express rings*, are used to accomplish the same goal. In contrast to express channels [69], the circular nature of rings does not require special routing in each dimension.

The most straightforward implementation of an express ring is shown in Figure 4.1, where a single ring has been added to a normal mesh. As can be seen from the figure, the number of hops required for global traffic can be reduced by routing through the global ring. For larger mesh sizes, the architecture shown in Figure 4.1 can be scaled in three ways using single one-dimensional rings. The first, and most straight-forward way to scale, would be to simply enlarge the sin-

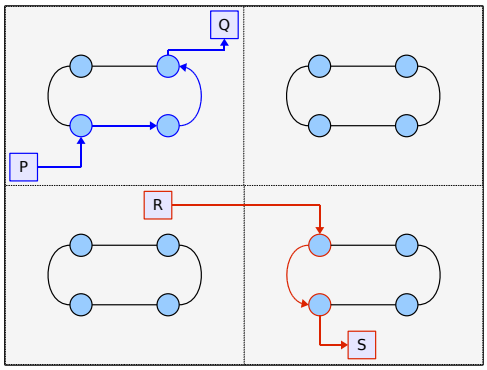


Figure 4.2: The *tiled* architecture where the mesh from Figure 4.1 has been copied four times in order to produce a large mesh with four express rings.

gle ring as mesh size grows. While simple, this approach may lead to a ring with large hop counts. Also, as nodes are added to the ring, the bisection bandwidth of the single ring may limit performance under heavy load. Alternatively, second method is to tile the mesh shown in Figure 4.1 to produce the larger mesh shown in Figure 4.2. The advantage of the tiling strategy is that it mirrors the natural way in which the mesh architecture scales, and does not require any additional modifications to the architecture. The disadvantage of this approach is that there may still be a large amount of congestion in the center of the mesh, especially since there may be a lot traffic between the rings. To address this issue, a third option shown in Figure 4.3, is to add concentric rings to the architecture as the size of the mesh increases. The concentric approach will more evenly spread traffic throughout the mesh network and will not be prone to a hot-spot in the center of the mesh like the tiled approach. Furthermore, the addition of more rings will spread the bandwidth demands over several rings, resulting in a more scalable alternative to using a single ring.

4.3 Hierarchical Ring Augmented Mesh

Figure 4.4 shows the *augmented* wormhole mesh where a hierarchical ring is used to route global traffic. The hierarchical ring depicted in Figure 4.4 consists of four local rings, each having four bridge components. In this case, the number

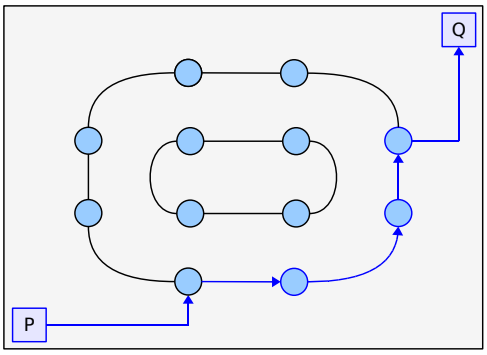


Figure 4.3: The *concentric* architecture is an alternate method of scaling up the single express ring from Figure 4.1 by adding a larger concentric ring.

of nodes n_{pe} available for processing elements on the mesh is described by

$$n_{pe} = (M \times N) - k_{hr} \ , \tag{4.4}$$

where k_{hr} is the number of nodes required by the hierarchical rings to connect to the mesh ($k_{hr} = 16$ in Figure 4.4). Note that the value of k_{hr} is dependent on the configuration of the hierarchical rings, and not on the size of the mesh. Sacrificing PEs for bridge components is an acceptable trade-off in light of the latency reductions that will be achieved. Furthermore, for large enough values of N, the removal of at most k_{hr} PEs will result in a small reduction of the total available computing nodes.

The augmented architecture routes global traffic through the hierarchical ring. In a mesh, the hop count for the communication shown between nodes P and Q in Figure 4.4 would be $2(N-1)$. In the augmented architecture, the hop count for the path $P \rightarrow Q$ is described by:

$$\begin{aligned} g_{hops}(P \rightarrow Q) = f_{hops}(P \rightarrow B_p) \\ + f_{hops}(B_q \rightarrow Q) + h_{hops}(B_p \rightarrow B_q) \end{aligned} \tag{4.5}$$

where B_p is the bridge component nearest P, B_q is the bridge component Q and h_{hops} is the number of hops through the hierarchical rings. For the communication $P \rightarrow Q$, the number of hops through the ring hierarchy is 7, while the worst case hop count for the hierarchical rings is 11. The worst case hop count for the

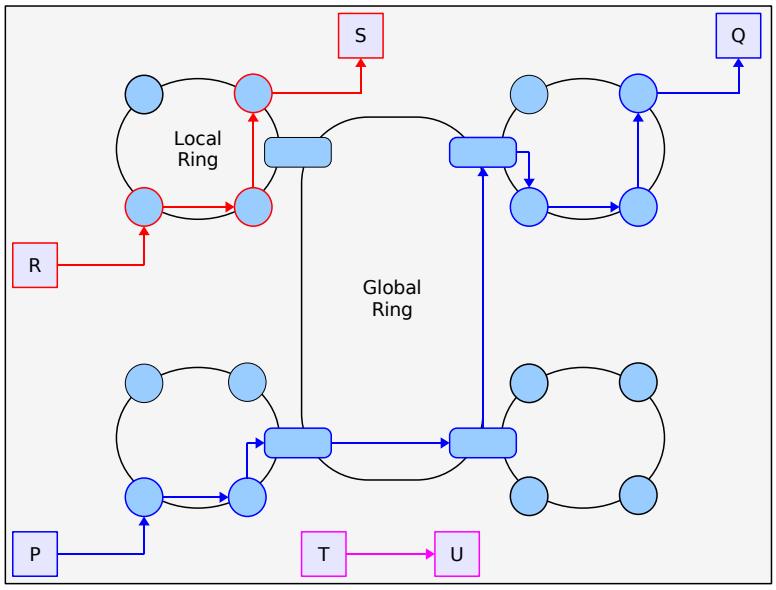


Figure 4.4: Augmented mesh architecture where a hierarchical ring interconnect is used to route global traffic. Long distance traffic travels through the rings as can be seen by the communication between $P \rightarrow Q$ and $R \rightarrow S$. Short distance traffic travels at the lowest level of the hierarchy through the mesh as can be seen by the communication between $T \rightarrow U$.

augmented architecture grows as a logarithmic function, rather than a square root function of the size [18]. Note that (4.5) can be applied to the express ring architectures as well.

4.4 Hybrid Ring/Mesh Interconnect

Figure 4.5 shows the hybrid-mesh architecture where a large mesh has been split into several smaller meshes that are globally connected using a two-level hierarchical ring interconnect. A *sub-mesh* is defined as the smallest mesh in the system, and a *local mesh* is the mesh obtained by combining the sub-meshes of a local ring together. From Figure 4.5, if the mesh being partitioned is of width N, then the width of a sub-mesh (W_{sub}) and the width of a local mesh (W_{local}) are

$$W_{sub} = \frac{M}{4}, W_{local} = \frac{M}{2} \ . \tag{4.6}$$

Routing traffic through the hybrid interconnect requires a labeling that relates

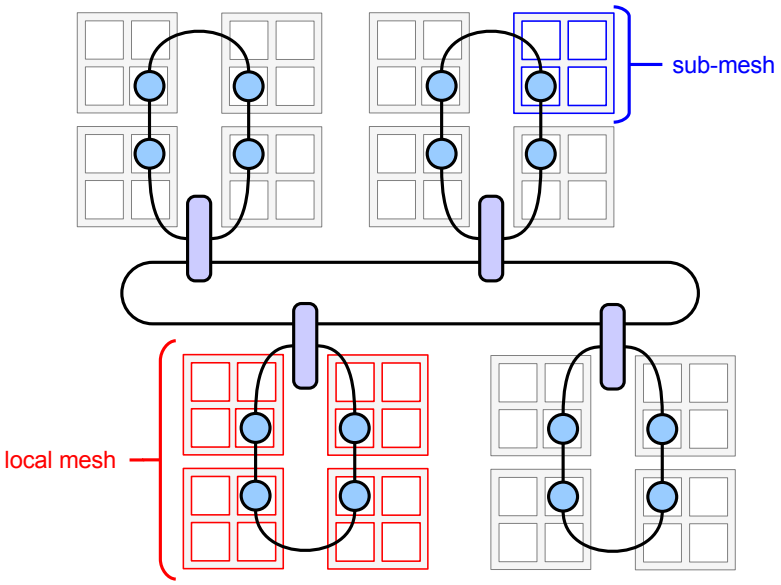


Figure 4.5: Hybrid mesh architecture using hierarchical rings for global interconnect. A *sub-mesh* is the smallest mesh in the system. A *local mesh* is the theoretical mesh obtained when combining all the sub-meshes belonging to the same local ring.

the xy-cordinates of a tile in the mesh to the appropriate ring addresses in the hierarchical rings. The following mapping function can be used:

$$f : \mathbb{XY} \rightarrow \tilde{\mathbb{XY}} \ , \tag{4.7}$$

where each tile in a sub-mesh is labeled $(\tilde{x}, \tilde{y})$ such that

$$\begin{aligned} \tilde{\mathbb{XY}} &= \left\{ (\tilde{x}, \tilde{y}) \mid \tilde{x} \in \tilde{\mathbb{X}}, \tilde{y} \in \tilde{\mathbb{Y}}, \tilde{\mathbb{X}} \subset \mathbb{X}, \tilde{\mathbb{Y}} \subset \mathbb{Y} \right\} \\ \tilde{\mathbb{X}} &= \left\{ 0, 1, \ldots, \frac{M}{4} - 1 \right\} \\ \tilde{\mathbb{Y}} &= \left\{ 0, 1, \ldots, \frac{N}{4} - 1 \right\} \end{aligned} \quad . \tag{4.8}$$

Using (3.1) and (4.8) together, a labelling for the hybrid interconnect is defined, where each tile can be represented by a tuple $(\tilde{x}, \tilde{y}, s, t)$ such that:

$$\tilde{\mathbb{XY}} \times \mathbb{ST} = \left\{ (\tilde{x}, \tilde{y}, s, t) \mid \tilde{x} \in \tilde{\mathbb{X}}, \tilde{y} \in \tilde{\mathbb{Y}}, s \in \mathbb{S}, t \in \mathbb{T} \right\} \tag{4.9}$$

where $\mathbb{S}$ and $\mathbb{T}$ are the same as defined in (3.1). For example, the bottom-left tile of each sub-mesh, using (4.8), has the label $(0, 0)$. There are 16 sub-meshes in total shown in Figure 4.5. For the bottom-left tile of each sub-mesh to have a unique label, (4.9) can be used to label each one as $(0, 0, s_i, t_j)$, for the i^{th} (local) ring and j^{th} ring-interface.

The worst case hop count for the unmodified mesh network is expressed by (4.3), and for a large enough value of N, the latencies incurred by the network will be too large for application software to support. The worst-case hop count for the hybrid interconnect depends on which tile in the mesh is used to connect to the hierarchical ring interconnect. If a corner tile is used, the worst case hop count can be described by (4.10) whereas if a tile in the middle of the sub-mesh is used, the worst case hop count can be described by (4.11), where H_{rings} is the number of hops through the ring interconnect.

$$H_{worst} = 2\left[\frac{M+N-2}{k}\right] + H_{rings} \tag{4.10}$$

$$H_{best} = 2\left[\frac{M+N-2}{2k}\right] + H_{rings} \tag{4.11}$$

Figure 4.6 shows how the hop counts increase as N increases for the mesh and hybrid architecture, but it does not necessarily imply that the latencies will increase proportionately. There are many parameters that can affect the latency, resulting in a large design space. A property of the hybrid interconnect that has been modeled is that the *xy*-routing algorithm tends to route global traffic away from the center of a sub-mesh towards the edges when the bridge tile is located in a corner. As will be seen in Section 6.2, the overall effect is that the resources in the center of the mesh end up processing less global traffic and local traffic is handled more efficiently.

When global traffic is routed through a bridge component, that component can quickly become flooded, causing the flow control mechanisms to activate, which can have a negative impact on latency. Increasing the input buffer sizes of the bridge tile solves the problem and enables traffic to flow much more smoothly. The required buffer size is very sensitive to the ratio of global to local traffic, as well as the size of the sub-mesh, since in the worst case, the amount of traffic can

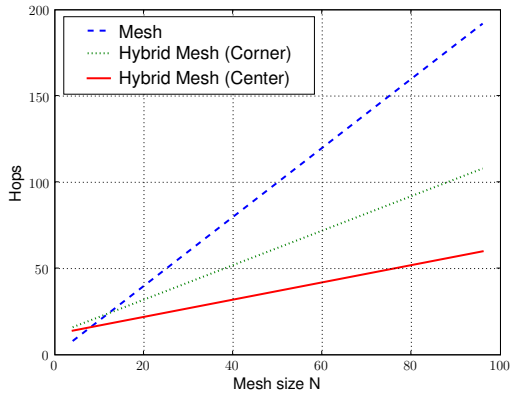


Figure 4.6: Worst case hop counts for the mesh and hybrid-mesh topologies. The worst case hop count for the hybrid topology depends on the placement of the bridge component.

increase quadratically with N.

4.4.0.1 Enhanced Hybrid Interconnect

The hierarchical ring interconnect described in [16, 17] is prone to congestion when back-pressure signals are asserted, resulting in the network being under-utilized. The hierarchical nature of the interconnect and its back-pressure signals can however be exploited to boost performance. It can be seen from Figure 3.1 that the potential for a system bottleneck is largest in the global ring. If the global ring asserts a back-pressure signal, all 4 local rings must stop sending data, which can result in the local rings being under-utilized. In the work presented in [16, 17], the hierarchical ring interconnect was used to connect PEs together, so the situation was acceptable, especially considering that the resources usage of the interconnect is efficient [17]. In the case of the hybrid topologies, that same structure is used to connect multiple NoCs together instead of PEs, so the bandwidth requirement on the hierarchical ring interconnect will be much greater.

Traffic on the hierarchical ring interconnect can be classified as *local* and *global*. Local traffic does not need to travel to the global ring, which is a fact that can be exploited to enable the implementation of an *enhanced* version of the hybrid interconnect. The *stop up* and *stop down* signals described in Section 3.3 can be

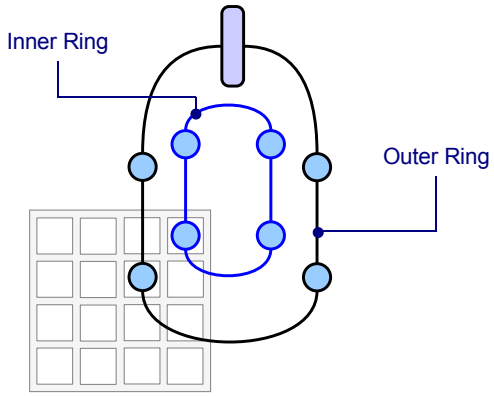


Figure 4.7: Enhanced architecture which uses two tiles to send data through the hierarchical ring interconnect. Since local and global ring traffic are unrelated, an increase in performance can be obtained by using two rings at the local level to route each type of traffic.

used to determine if data can still be injected into the interconnect. If the global ring asserts a *stop down* signal because of congestion, this does not necessarily mean that the local ring is also congested. The routing of global and local data can thus be performed independently, reducing the overall latencies of traffic in the hybrid architecture.

By modifying the implementation of the ring-interface component of the hierarchical rings, logical channels similar to *virtual* channels can be achieved. The advantage of this approach is that the logical channels can share hardware. For every local ring:

- The *inner* channel is used for sending data locally on a ring.
- The *outer* channel is used for sending global data through the global ring.

The splitting of the network into two logical channels has it costs, such as the need to sacrifice a second tile on the mesh to avoid multiplexing the data at the level of the mesh. Figure 4.7 shows how the enhanced architecture uses two tiles to connect the mesh and hierarchical rings together. Since a corner tile has only two input and output ports, it is preferable to route traffic to the inner ring through an interface that is not on an edge.

4.5 Ring/Mesh Bridge Component

To evaluate the proposed architectures, the hardware implications (area, latency, energy consumption) of the dedicate bridge component requires consideration. The bridge component is a unit that enables data transfer to/from the mesh and hierarchical ring interconnect. The major hurdle faced when implementing the bridge component is that it interfaces two fundamentally different network architectures. As already discussed in Section 4.1, the mesh network is a wormhole routed-network, where a collection of flits are logically related to form a packet. In contrast, the hierarchical ring interconnect considers all routed flits as *unrelated*. The bridge component must maintain the ordering of flits at egress points, and also must ensure to not interleave flits belonging to different worms. There are two ways to handle the problem of interfacing these two dichotomous networks that involve different bridge architectures; they will be described in the following subsections.

4.5.1 Narrow Bridge Implementation

At first glance, the most straightforward approach to bridging the mesh and ring networks is to simply tunnel the flits coming from the mesh through the hierarchical rings as was done in [18, 19]. This approach is termed "narrow" because each flit is routed individually, and so the data width of the ring network is equal to the width of a flit from the mesh network plus the header width of the ring network. Figure 4.8 shows the narrow implementation of the bridge component connecting a mesh switch and ring interface. The left side of Figure 4.8 shows how flits are moved upwards from the mesh to the ring interconnect by tunneling mesh flits individually through the ring.

Since the ring interconnect considers flits to be independent, flits from different ring interfaces can be interleaved as the travel through the interconnect. When a flit arrives at an egress point, the bridge component must reassemble the original mesh packet so that it can be injected properly back into the mesh network. It is critical that flits from different packets not be interleaved by the bridge components — the behavior of the mesh network will be non-deterministic.

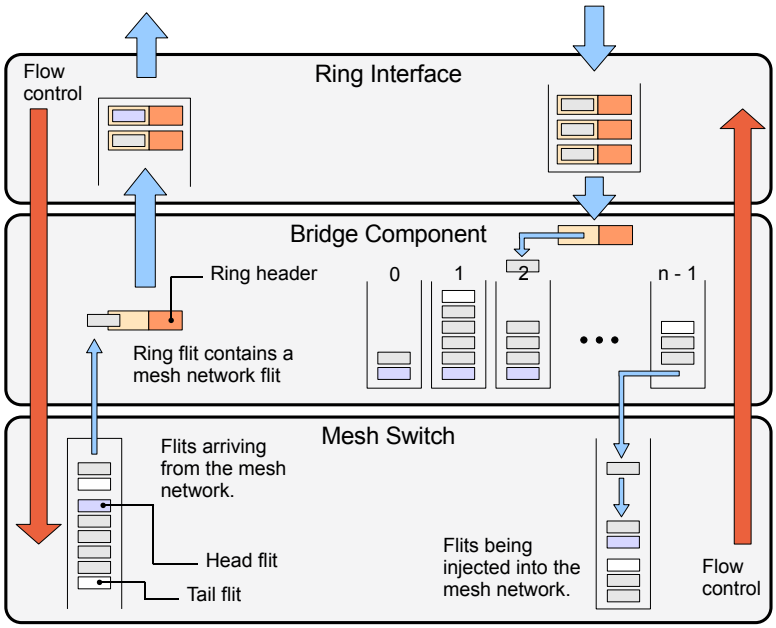


Figure 4.8: The narrow implementation of the bridge component injects flits into the ring network as they arrive from the mesh. The bridge must reassemble packets at the egress point, which requires $n_{\mathrm{ri}} - 1$ buffers large enough to store a complete mesh packet. Only once the entire packet has been received at the egress point can the bridge component inject a packet into the mesh network.

To guard against interleaving packets, the bridge component can reconstruct the original flit at the egress point. For every bridge component connected to the hierarchical rings, there can be at most $n_{\mathrm{ri}} - 1$ different worms arriving simultaneously from the ring interconnect. The reason this holds true is because a packet is known to leave the mesh network intact due to the nature of wormhole routing.

To solve the problem of interleaving, every bridge component stores flits until a tail flit arrives to complete a worm. Only when a full worm has arrived will a bridge component start to transfer it to the mesh. To reconstruct packets, the bridge component requires enough storage to hold $n_{\mathrm{ri}} - 1$ worms. The size of the buffers required is directly related to the maximum packet/worm size allowed on the mesh. If the packet size is large, the buffer requirements for the bridge will be larger. Additionally, large packet sizes will result in a latency penalty while packets are forced to wait to be reassembled at the bridge component before being forwarded to the mesh. The right side of Figure 4.8 shows how the bridge component stores flits from individual worms in W_{count} FIFOs.

A drawback of the narrow implementation is that the latency incurred by each flit due to the reassembly process at the egress point is non-deterministic and depends on the congestion in the ring interconnect. As long as there is 1 outstanding flit remaining, the other flits belonging to the same packet cannot be injected into the mesh. This situation can also have the negative effect on increasing the inter-flit jitter of a packet to an unacceptable degree (depending on the application).

4.5.2 Wide Bridge Implementation

A second approach used here is to interface the two networks by widening the hierarchical rings such that a full worm can be carried by a single ring-flit — termed the "wide" bridge implementation. While this creates a larger bandwidth, the drawback of this approach is that it will require more resources. This trade-off can be justified since the goal of using the hierarchical rings is to reduce latencies and hop-counts for global traffic. A real world example of using a wide bus-width for a ring-based interconnect is the IBM Cell processor, which

uses four unidirectional rings that are each sixteen bytes wide. Since an entire packet (or worm) will arrive intact at the egress bridge, less buffering is needed to store outgoing flits. Recall that for the straightforward or *narrow* approach, each bridge component must be able to store $n_{\mathrm{ri}} - 1$ complete packets. For the wide or *fat* approach, only a single packet needs to be buffered because packets are processed as they arrive from the ring interconnect. The wide approach differs from the narrow approach further in the way incoming flits are processed. In the narrow approach, flits are simply forwarded onto the rings as they arrive, which necessitates packet reassembly and extra buffers at the egress bridge. The wide approach, on the other hand, needs to buffer flits until the entire packet has arrived, at which time it can be forwarded intact to the rings. Figure 4.9 shows an example of the wide bridge implementation, which requires only 2 registers wide enough to store a ring flit. The figure shows how a ring flit is assembled over a period of several clock cycles at the ingress port, and how the flits are subsequently injected back into the mesh. Unlike the narrow approach, latency incurred at the ingress and egress bridges is deterministic. Another advantage to using wide hierarchical rings is that since it takes multiple clock cycles to assemble a packet at the ingress point, a single bridge component will only be able to inject a flit into the ring interconnect every n_f clock cycles (where n_f is the number of flits per packet). This forces a fair sharing of the bandwidth available on each local ring between each bridge component belonging to that ring. The result is that the possibility of starvation is removed and the overall performance of the system is improved[2].

4.6 Resource Requirements

Extending a mesh network by adding hierarchical rings improves performance at the cost of the extra resources required by the rings. In this section, a characterization of the resource overhead for each architecture is presented.

The resource requirement of a FIFO is a function of its width w and depth d,

[2]The simplest implementation of a ring is prone to starvation unless extra logic is added to provide fair sharing of bandwidth.

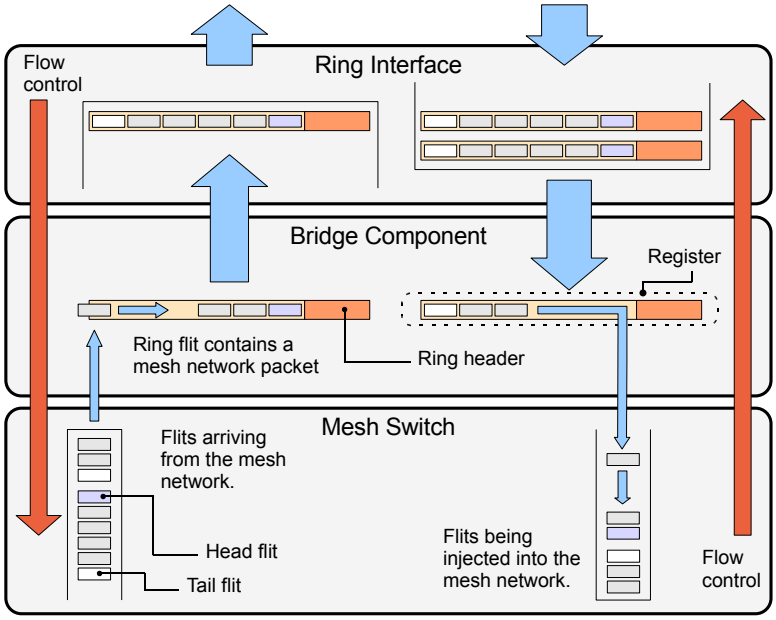


Figure 4.9: Wide implementation of the bridge component. Flits arriving from the mesh network are stored in a register until the entire packet has been received. Once the tail flit arrives, the entire packet can be forwarded through the hierarchical rings as a single ring-flit. At the egress bridge, the packet is injected one flit at a time back into the mesh network.

expressed as

$$F(w,d) = w \cdot d \quad . \tag{4.12}$$

The resource requirement of a mesh router can therefore be expressed as

$$R_{mesh}(w,d) = 6 \cdot F(w,d) + L_{mesh}(w,d) \quad , \tag{4.13}$$

where L_{mesh} represents the resource usage due to combinational and sequential logic (note that a mesh router has 6 FIFOs). Similarly, the resource usage of the ring-interface and inter-ring interface components can be expressed as

$$\begin{aligned} R_{ri}(w,d) &= 3 \cdot F(w,d) + L_{ri}(w,d) \quad , \\ R_{iri}(w,d) &= 2 \cdot F(w,d) + L_{iri}(w,d) \quad . \end{aligned} \tag{4.14}$$

since the number of FIFOs in the ring and inter-ring interfaces are 3 and 2 respectively. Making a simplifying assumption that L is approximately the same for all components, then (4.13) and (4.14) are reduced to being functions of F only.

For the wide implementation of the hierarchical rings, the F term is multiplied by n_f (number of flits per worm), so (4.14) can be written in terms of R_{mesh} as

$$\begin{aligned} R_{ri} &= n_f \cdot \frac{3}{6} \cdot R_{mesh} \quad , \\ R_{iri} &= n_f \cdot \frac{2}{6} \cdot R_{mesh} \quad . \end{aligned} \tag{4.15}$$

Therefore, the extra resources required by the hierarchical rings can be expressed in terms of R_{mesh} as

$$R_{hrings} = n_{ri} \cdot R_{ri} + n_{iri} \cdot R_{iri} \ , \tag{4.16}$$

which evaluates to approximately $7n_f \cdot R_{mesh}$ extra resources required for the hierarchical ring architecture. It should be noted that the assumptions are based on a simple implementation of the mesh router; more complex routers that use virtual channels would have a higher buffer requirement.

For the hybrid architecture, resources are actually saved by splitting a large mesh into sub-meshes since edge and corner tiles require less resources. When a mesh is split into smaller meshes, all the links that are cut result in resource

savings because the associated FIFOs are no longer needed. A normal mesh router has 6 FIFOs, whereas an edge tile has 4, and a corner tile has 3. The difference in required buffers between the normal mesh and hybrid architectures is related to the number of tiles that are converted to edge and corner tiles and can be expressed as

$$\Delta R_{hybrid} = \Delta R_{edge} + \Delta R_{corner} \ , \tag{4.17}$$

where ΔR_{edge} is the difference in resources required by edge tiles, and ΔR_{corner} is the difference in resources required by corner tiles.

For a normal mesh, the number of corner tiles n_{corner} is 4, and the number of corner tiles in the hybrid architectures is $16n_{corner}$. Therefore, the difference in resource requirement due to an increased number of corner tiles can be expressed using (4.15) as

$$\Delta R_{corner} = (16n_{corner} - n_{corner})\frac{2}{6} \cdot R_{mesh} \ . \tag{4.18}$$

Similarly, the number of edge tiles in a mesh is given by

$$n_{edge}(N, M) = 2(N-1) + 2(M-1) - n_{corner} \ , \tag{4.19}$$

or

$$n_{edge}(N) = 4(N-1) - n_{corner} \tag{4.20}$$

when $M = N$. For the hybrid interconnect, each sub-mesh is of size $N_{sub} = \frac{N}{4}$, for which (4.20) evaluates to

$$n_{edge}\left(\frac{N}{4}\right) = N - 8 \ . \tag{4.21}$$

Since there are 16 sub-meshes, the total number of edge tiles in the hybrid interconnect can be expressed as

$$n_{edge,hybrid} = 16 \cdot n_{edge}\left(\frac{N}{4}\right) \ . \tag{4.22}$$

Therefore, the difference in resource requirement due to an increase in the number

of edge tiles in the hybrid architecture can be expressed as

$$\Delta R_{edge} = (n_{edge,hybrid} - n_{edge}) \cdot \frac{1}{6} \cdot R_{mesh} \ . \tag{4.23}$$

For example, using a mesh size of $N = 16$, (4.17) evaluates to

$$\begin{aligned} \Delta R_{hybrid} &= 74 \cdot \frac{1}{6} \cdot R_{mesh} + 60 \cdot \frac{2}{6} \cdot R_{mesh} \\ &= 32 \cdot R_{mesh} \end{aligned} \ . \tag{4.24}$$

Therefore, when a mesh of width 16 is converted to the hybrid architecture, the resources saved by splitting the large mesh into 16 smaller sub-meshes is roughly equivalent to 32 normal routers. A mesh of size 16 contains 256 tiles, so a savings of 32 routers represents an approximate reduction of 12.5%, Thus, a good alternative to having a single large mesh is to use several smaller meshes that are interconnected by a global interconnect as is the case for the hybrid interconnect. The estimate can be seen as being conservative because the cost (i.e. area) of each router is linearly dependent on the number of input FIFOs. However, the area of the routers usually grows quadratically with the number of inputs due to the poor scalability of the crossbar switches commonly used in NoC implementations.

Chapter 5

NoCsim Simulation Platform and Software

Certain properties such as resource requirements, maximum operating frequency, energy usage, and routing complexity can only be extracted from RTL level implementations. However, hardware description languages are not as flexible as high-level programming languages for quickly prototyping new architectures. Furthermore, creating complex test-benches using HDLs is time consuming and difficult because the designer is ultimately limited by the expressiveness of the language being used. The architectures presented in Chapter 4 would have proven time consuming to implement at the RTL level; architectural modeling was performed using behavioral SystemC models instead. In order to model and simulate the various hybrid architectures, the following application/libraries were created:

- *NoCsim*: A custom library of SystemC components that are used to model the different architectures. The entire source tree totals approximately 18k *lines of code* (LOC).
- *Gengraph*: A custom class C++ library and application that provides data structures for manipulating and generating graphs (3k LOC).
- *SAgraph*: Implementation of the simulated annealing algorithm that makes use of the Gengraph library to generate optimized task assignments for the mesh and hybrid topologies (1k LOC).

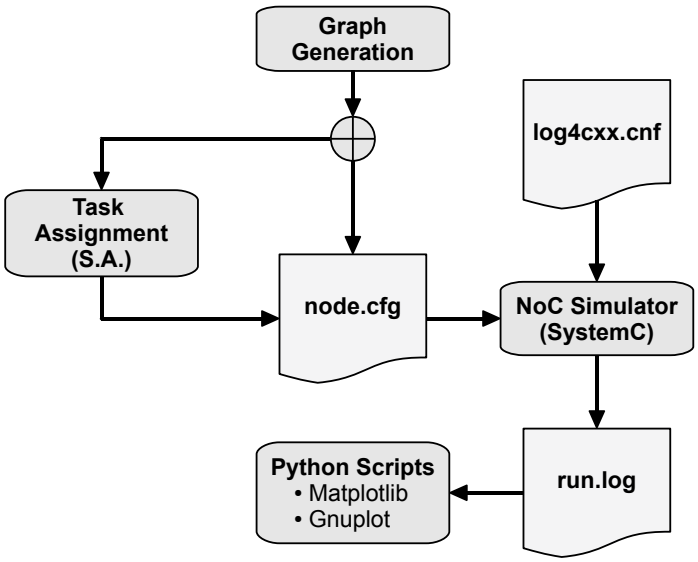


Figure 5.1: Custom Network-on-Chip simulation platform tool flow.

Figure 5.1 shows how each tool fits within the simulation tool flow used to generate the simulation results presented in Chapter 6. First, the Gengraph program is used to generate a task graph which is written to a configuration file. On startup, the NoC simulator configures each node on the network based on the generated task graph configuration file. Once a simulation run completes, the log files are processed by custom Python scripts that generate plots using either Matplotlib or Gnuplot.

The following sections will describe how each component of the simulation environment works.

5.1 NoC Simulation Platform and Library

The NoC simulation platform (NoCsim) consists of a library of components that were designed to leverage the expressiveness of C++. Specifically, *object oriented* (OO) design concepts were used so that the components could be easily combined in a "plug-and-play" manner to facilitate architectural exploration. Figure 5.2 shows the *unified modeling language* (UML) Class Diagram of the NoCsim

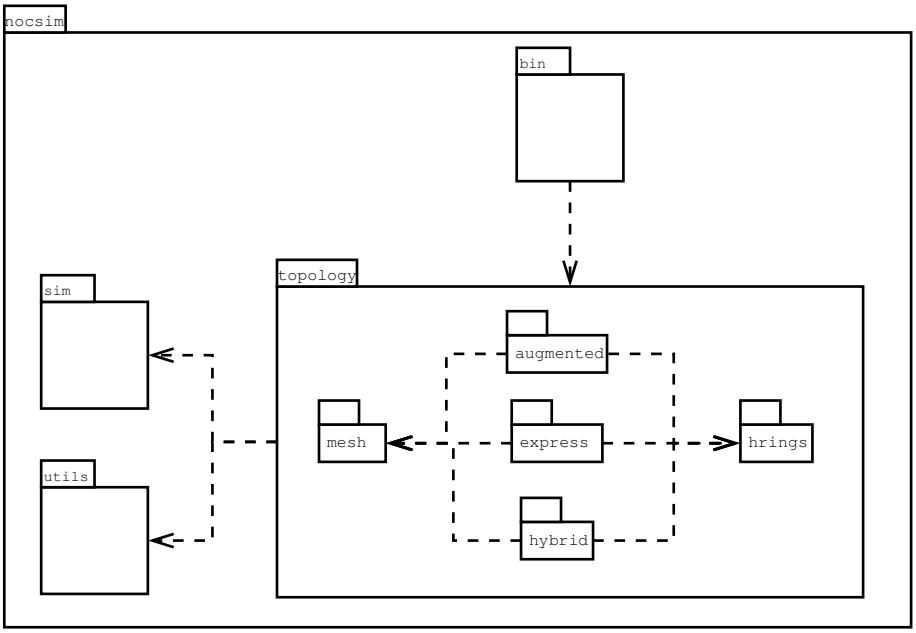


Figure 5.2: UML class diagram of NoC simulation platform architecture.

platform. The `topology` *package*[1] contains sub-packages that in turn contain the components specific to each topology being modeled. For example, the `mesh` package contains all classes related to modeling a mesh architecture.

One of the main features that makes constructing a network topology relatively easy is the way that components are connected using OO concepts. For example, ports are grouped together using special utility classes, and components can be connected together using specially designed methods. To illustrate the mechanisms used to connect components together, the `hrings` package, which is used to model the hierarchical rings, is explained in the next subsection.

5.1.1 The `hrings` package

The class diagram of the `hrings` package is shown in Figure 5.3, where the classes can be categorized as:

- *Container* classes represent high-level abstractions and are composed of several component classes:

[1] A package is the UML term for a module or library.

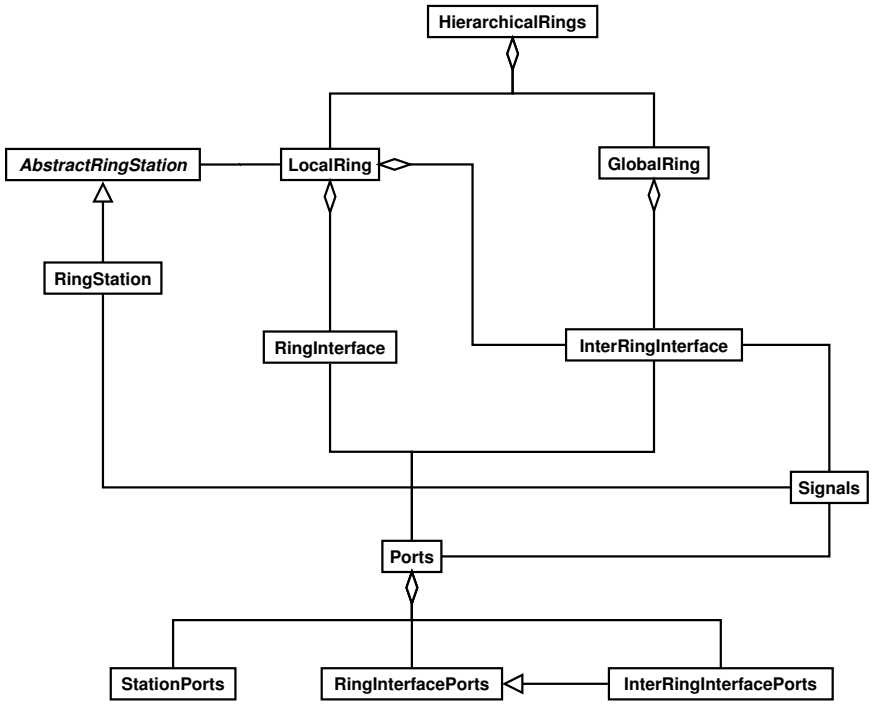


Figure 5.3: Hierarchical rings package class diagram.

  - `HierarchialRings` is the top-most class that when instantiated, represents the entire hierarchical ring interconnect.
  - `LocalRing` and `GlobalRing` represent instances of a local or global ring, repectively.

- *Components* are classes that model hardware components such as routers:
  - `RingInterface` and `InterRingInterface` model the IR and IRI components discussed in Chapter 3.
  - `RingStation` models a terminal node that sends and receives data over the interconnect.

- *Port/signal* classes are used to group logically related ports and signals together similar to VHDL *records*[2].
  - Any class name ending in `Ports` or `Signals`.

[2] A VHDL record is loosely equivalent to a *struct* in C.

The straightforward method of connecting components together in SystemC is to define individual input and output ports that will need to be connected one-by-one. In SystemC, a *signal* represents a medium over which two components can communicate. Therefore, connecting two components requires that a signal be connected to an output port of the first component, and then to the input port of the second component. Connecting ports individually can lead to errors when dealing with complex components that have many ports, and detecting a wrongly connected port can be difficult and time consuming. A better approach, which is employed by NoCsim, is to use OO design techniques to group port objects into classes.

Before going into the SystemC/C++ implementation details, it is instructive to first consider the block diagram in Figure 5.4, which shows how components can be connected by using ports and signals graphically. For example, since each RI needs to connect to a processing element (in this case, the `RingStation`) and other RIs, it needs to have 3 ports:

1. An input port that connects to the previous ring-interface on the ring.
2. An output port that connects to the next ring-interface on the ring.
3. A bi-directional port that connects to the processing element[3].

Since a port is nothing more than a C++ object, multiple port objects can be grouped together within specialized `Port` classes. As shown in Figure 5.3, there are several port-type classes used by the hierarchical rings model. Listing 5.1 shows a code snippet taken from the `RingInterfacePorts` class, which itself has internal classes for input and ouput ports. For example, the `OutToRing` class shown in Listing 5.1, groups all output ports that communicate with the ring. All output ports that communicate with the `RingStation` are grouped together in the `OutToStation` class (not shown). Furthermore, `OutPorts` groups all output ports that belong to the `RingInterface` together in a single class. Lastly, the input and output port classes can be instantiated by the `RingInterface` class, and all of the ports can be accessed through the `in` and `out` objects.

[3]Note that a bi-directional port is implemented as two unidirectional ports.

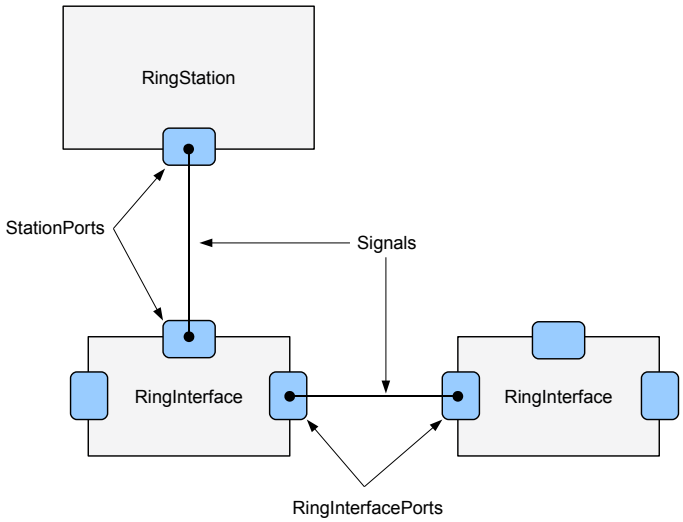


Figure 5.4: Block diagram showing how components are connected using ports and signals. Each component has 1 or more ports that can be used to connect to other components.

The classes shown in Listing 5.1 enable the port-to-port connections shown in Figure 5.4 to be made. What is missing is an efficient way to connect the ports that is less error prone than performing the connections manually. The reason the "one-by-one" approach is error prone is because the connections must be done manually by the programmer, who must be aware of how all the ports and signals connect to each other. The approach taken when designing the NoCsim libraries was to centralize the port connection code by using `connect` methods. Listing 5.2 shows an example of the connect method for the ring interface. Note the method signature, which provides an added layer of security; if the person calling the function passes an incorrect object type, the compiler will catch the error. Manually connecting the ports and signals would require the code in the `connect` method to be repeated often, which can result in errors due to typos, copy/paste errors, or programmer error. Conversely, centralizing the connection by putting it in a function makes instantiating components easier and less prone to errors.

Once all the port, signal, and connect methods have been defined, a network topology can be easily constructed by the programmer. Listing 5.3 shows how a `for` loop can be used to connect several ring-stations to the appropriate station

Listing 5.1: Partial code listing showing how the output ports of the `RingInterface` are grouped together using separate classes. Also shown is how the input and output ports are instantiated by the `RingInterface` class.

```
1  template<class DataT, int ADDR_BITS = 4>
2  class OutToRing
3  {
4    public:
5
6      sc_out<bool> stopUp;
7      sc_out< std::bitset<ADDR_BITS> > stopDown;
8      sc_out<DataT> data;
9
10     OutToRing() : stopUp("stopUp"), stopDown("stopDown"), data("data") { };
11 };
12
13 template<class DataT, int ADDR_BITS = 4>
14 class OutPorts
15 {
16   public:
17
18     OutToRing<DataT, ADDR_BITS>    ring;
19
20     OutToStation<DataT>            station;
21 };
22
23
24 template<class RingFlitT, int ADDR_BITS>
25 class RingInterface : public sc_module
26 {
27   public:
28
29     InPorts<RingFlitT, ADDR_BITS>  in;
30
31     OutPorts<RingFlitT, ADDR_BITS> out;
32
33   // ...
34 };
```

objects using repeated calls to a single function. The effect is to hide (or abstract) from the programmer the underlying code needed to connect all the ports and signals together.

The major advantage achieved by abstracting away the details of connecting components is to promote *reuse*, where connecting components together to form new topologies can be done relatively easily, as long as each component respects the port/signal/connect paradigm. The hybrid interconnects described in Section 4, where the hierarchical rings and mesh topologies were combined, were constructed in this manner.

5.2 Traffic Generation

This section describes the applications used to generate traffic on the interconnect by creating task-graphs using a custom written application. Furthermore, a second application can be used to optimize the task distribution of the randomly

Listing 5.2: The signatures for the `connect` method of the ring interface

```
/**
 * Connect two ring interfaces together.
 */
static void connect(RingInterfaceT *left, RingInterfaceT *right, RingSignalsT *sigs){
  // connect backpressure signals.
  left.in.ring.stopUp(sigs.stopUp);
  left.in.ring.stopDown(sigs.stopDown);

  right.out.ring.stopUp(sigs.stopUp);
  right.out.ring.stopDown(sigs.stopDown);

  // connect data channel
  left.out.ring.data(sigs.data);
  right.in.ring.data(sigs.data);
};

/**
 * Connect a station to a ring interface.
 */
static void connect(AbstractRingStationT *station, RingInterfaceT *ri,
    StationSignalsT *sigs){
  // ...
};
```

Listing 5.3: Using the `connect` method of the ring-interface in a loop to connect the ports of `RingStation` to `RingInterface` objects.

```
// Connect Station objects to RingInterface objects.
for(int i = 0; i < nodes; i++)
{
  RingInterfaceT::connect(
      stations[i],
      riArray[i],
      &ri_signals[i]->station);
};
```

generated task graphs in order to improve the locality of the generated traffic. Also, in order to compare the performance of each architecture, a category partitioning of traffic types based on the distances traveled in the interconnect is made.

5.2.1 Graph Generation

A standalone utility called *Gengraph* was implemented in C++ to generate pseudo-random and biased pseudo-random task graphs and map them onto the interconnect under study. Each vertex in the graph corresponds to a task, and each *directed edge* between two vertices indicates that the *source* vertex will send data to the *sink* vertex during simulation. The complexity of the task graph is controlled by several input parameters. The process of generating a task graph and mapping consists of the following steps:

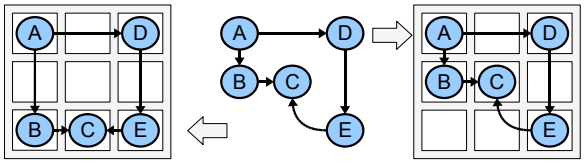


Figure 5.5: Example of two possible mappings of a graph for a mesh size of 3×3.

1. A task graph is generated based on input parameters.
2. The task graph is mapped to a mesh.
3. The final mapping is written to a configuration file.
4. The configuration file is read by the simulator.

In order to generate application-like traffic on the interconnect using the task-graph generation program, some simplifying assumptions have been made in the simulation models:

- Only a single task is mapped to a node on the interconnect.
- Each task can send to (*out-degree*) and receive from (*in-degree*) zero or more tasks.
- The maximum *in/out-degrees* of a task can be constrained.

Since we are interested in generating traffic on the network, mapping a single task is sufficient. However, the approach can be expanded to handle more complex task graphs (i.e. tasks can be annotated with priorities, execution times, resource type, etc).

The simulator constructs a *send schedule* for each node based on the *out-degree* of the task that has been mapped to it. A simple example of a task graph and subsequent mapping is shown in Figure 5.5. When the configuration file is read by the simulator, the node that has been assigned task A will construct a send schedule consisting of two entries, namely B and D. During the simulation, task A will send data to both tasks at randomly spaced intervals.

5.2.2 Classification of Traffic

Since the composite architectures are comprised of several levels arranged in a hierarchy, it is necessary to categorize traffic according to the levels in the hierarchy that it traverses. Therefore, a classification of traffic is made as follows:

- C_0 traffic is termed *local* because the distances travelled are short. Furthermore, this type of traffic does not require routing through the ring interconnect in the composite architectures.
- C_1 traffic must travel an intermediate distance and may be routed through the global interconnect.
- C_2 traffic must travel long distances and will always[4] traverse the global interconnect.

The partitioning of traffic into the three categories enables the unmodified mesh network and the composite architectures to be compared. Due to the architectural differences of each architecture, the category partitions must be performed differently for each. The following subsections describe how the different traffic types are categorized for each architecture.

5.2.2.1 Wormhole-Routed Mesh

For the unmodified mesh network, the category partitioning of traffic corresponds to the distances between any pair of sender and receiver nodes on the network. The categories are described by

$$f_{cat}(p_i) = \begin{cases} C_0: & 0 & < f_{hops}(p_i) <= \frac{M+N-2}{4} \\ C_1: & \frac{M+N-2}{4} & < f_{hops}(p_i) <= \frac{M+N-2}{2} \\ C_2: & \frac{M+N-2}{2} & < f_{hops}(p_i) <= M+N \end{cases} . \tag{5.1}$$

In (5.1), p_i corresponds to a path between two tiles on the mesh. By classifying traffic using (5.1), data can be collected on local and global traffic for the purpose

[4] If adaptive routing is being used, C_2 traffic may be routed differently without traversing the global interconnect.

of comparing the performance of the normal mesh topology to the composite architectures.

5.2.2.2 Augmented and Express Architectures

Since the augmented and express architectures consist of a normal wormhole mesh with either a hierarchical or normal ring network added, the traffic classifications for the new architectures can be made using (5.1). In the augmented architecture, C_0 traffic will travel only through the mesh network at the lowest level of the hierarchy because the distances are short. The intermediate distances of C_1 traffic requires that it be routed up to the second hierarchical level and through a local ring of the hierarchical ring network. Lastly, C_2 traffic, which must travel the longest distances, will be routed up to the third hierarchical level through the global ring of the hierarchical ring network.

The three traffic categories are represented graphically in Figure 4.4. Short paths between two sender nodes such as $T \rightarrow U$ are of type C_0 traffic. The paths $P \rightarrow Q$ and $R \rightarrow S$ shown in Figure 4.4 are of types C_2 and C_1, respectively. Since the goal of the augmented and express architectures is to reduce the latency incurred by global traffic, traffic types C_1 and C_2 are of primary interest because they are the types most likely to traverse the hierarchical ring interconnect.

5.2.2.3 Hybrid Architecture

The classification of traffic types for the hybrid topology cannot be made using (5.1) because two nodes that are logically close to each other using (4.1) may actually belong to different sub-meshes, requiring that the packet travel through the hierarchical ring interconnect. For the hybrid topology, the traffic types correspond to the layers in the hierarchy shown in Figure 3.1. Using (4.9), traffic can be categorized using

$$f_{cat}(P \rightarrow Q) = \begin{cases} C_0: & s_p = s_q, t_p = t_q \\ C_1: & s_p \neq w_q, t_p = t_q \\ C_2: & s_p \neq w_q, t_p \neq t_q \end{cases} . \tag{5.2}$$

The classifications given by (4.9) can be explained in plain language as:

- C_0 traffic corresponds to traffic that is sent between tiles at the lowest level of the hierarchy, namely between stations belonging to the same sub-mesh.
- C_1 traffic travels through a single local ring to reach a tile on another sub-mesh which is part of the same local mesh.
- C_2 corresponds to traffic that travels between tiles belonging to different local meshes and thus must go through the global ring.

5.2.3 Optimized Task Assignment

A companion application called *SAgraph* that uses the Gengraph data structures was implemented to performs optimized task assignment using *simulated annealing* [126]. The motivation for performing optimized task assignment is because randomly assigning tasks to nodes on the composite architectures will not make the best use of available resources because non-optimal assignment will generate a large amount of traffic which will be routed through the global interconnect. In [20], simulated annealing was used to perform optimized task assignments. The algorithm analyzes a task graph and assigns tasks to nodes to minimize path lengths between communicating nodes.

Simulated annealing is a heuristic algorithm that can be used to find a good approximation of the global optimum in a reasonably small number of iterations (e.g. 10^6). The algorithm starts at a random initial state that has an initial *energy* (cost). A neighboring state is chosen at random and its energy is calculated using the cost function. The current energy is compared to the energy of the neighboring state, and a decision is made to stay at the current state, or to transition to the neighboring state. The goal of the algorithm is to iteratively move from a high energy state to a low energy state, which corresponds to a local minimum. The algorithm terminates when the maximum number of iterations has been reached.

The selection of a neighboring state is performed in an application specific way. For the task-assignment problem described here, the neighboring state is chosen by randomly exchanging the assignment of a pair of tasks. Ideally, transitions would only be made to states with a lower energy/cost (downhill move), but

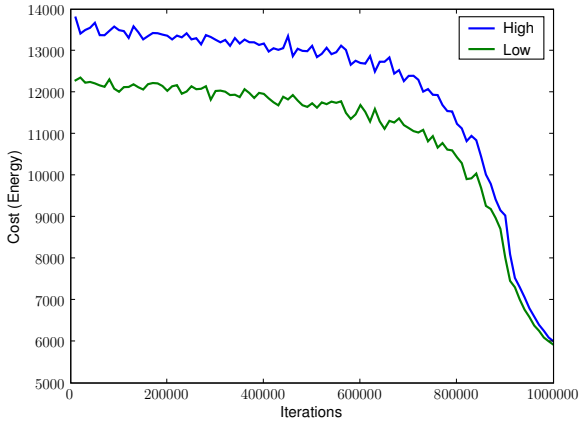


Figure 5.6: The progression of the simulated annealing algorithm over 10^6 iterations for a task-graph consisting of 20^2 vertices and 10^2 edges being mapped onto a 20^2 mesh. The space between the two lines shows how the tolerance for uphill moves decreases with time.

restricting the algorithm to downhill only moves may prevent the algorithm from finding a good solution because it converges quickly to find a local minimum.

To enable the algorithm to search a larger portion of the solution space, uphill moves (transitions to states with higher energy) are allowed based on a probabilistic function. The probability that a uphill move is allowed decreases with time, as does the size of the allowed move. Figure 5.6 shows the progression of the simulated annealing algorithm for a task-graph consisting of 20^2 vertices and 10^2 edges being mapped onto a 20^2 mesh over 10^6 iterations. For clarity, the 10^6 data points have been separated into 10^3 blocks, and the maximum and minimum values of each block have been plotted. The area between the two lines shows the range of values of the evaluated cost function. At the start of execution, when the probability of uphill moves is greatest, it can be seen that the different between the "High" and "Low" lines in Figure 5.6 is greatest. Hence, the algorithm is allowed to search through a larger numbering of neighboring states. As the number of iterations increases, less uphill moves are allowed. As the probability of an uphill move is decreased, the algorithm is restricted to neighboring states with lower cost, thereby resulting in a narrowing in the gap between the two plots in Figure 5.6.

For the normal mesh architecture, the sum of the distance given by (4.2) can be minimized to find a near-optimal solution. However, a different approach must be taken for the composite architectures because traffic that must travel up through the global interconnect has a higher associated cost than traffic that does not. For the composite architecture, (4.2) can be rewritten as

$$\tilde{f_{hops}}(p_i) = \alpha f_{hops}(p_i) \ , \tag{5.3}$$

where

$$\alpha = \begin{cases} 1 : f_{cat}(p_i) = C_0 \\ 2 : f_{cat}(p_i) = C_1 \\ 6 : f_{cat}(p_i) = C_2 \end{cases} \ . \tag{5.4}$$

The multiplier α in (5.4) is used to increase the penalty associated with a task assignment that requires traffic to go through the hierarchical rings. The value of α is greatest for traffic that would have to travel through the system bottleneck, namely the global ring.

5.3 Blocking Aware Task Assignment

The two-dimensional mesh network is a popular topology used for NoC interconnects. It consists of multi-port switches arranged in a rectilinear grid, where each switch is connected to its associated core and bi-directionally to its top, bottom, left and right neighbors. Each core-switch unit is called a *node*. The most popular routing method used in on-chip mesh networks is deterministic *xy* (dimension) routing because it is simple and guarantees minimum path length. However, blocking (contention) can occur when multiple packets are routed along intersecting paths. In fact, wormhole networks have poor link utilization under heavy loads because they saturate from contention [127] well before the available bandwidth has been exhausted. In an attempt to reduce the latency penalties associated with blocking, various *adaptive routing* schemes (which are non-deterministic) have been suggested as improvements over *xy* routing, but it has been shown that deterministic routing often yields superior performance [128].

In this section, a methodology for assigning tasks to nodes in a wormhole-routed mesh that takes contention into account is presented, such that the overall performance is optimized. The algorithm accepts a task graph where vertices and edges represent respectively the tasks and the communications between them. The algorithm analyzes the task graph and assigns tasks to nodes to minimize path lengths and the amount of potential blocking during concurrent communications.

As stated in [129], only a few approaches from real-time systems have been applied to SoC design. Much of the work done in *real-time scheduling* (RTS) treats the communication medium as a resource that can be locked by only one process at a time. This treatment does not apply well to a wormhole routed mesh, because no single task can explicitly lock the communication path between any two nodes on the network. To meet *quality-of-service* (QoS) constraints such as those imposed by real-time systems, resources must be dedicated to high-priority traffic on the network as discussed in [53]. The work presented here aims to reduce latencies for a normal mesh network by avoiding contention and making more efficient use of available resources without explicitly reserving resources for high priority traffic. This approach is acceptable for systems which do not have stringent timing constraints such as *soft* real-time systems or general purpose *chip-multiprocessors* (CMPs).

5.3.1 Communication Costs

In *xy* routing, a deterministic communication path is established between two cores on the mesh. The path begins at the core that initiates the communication and always starts by traversing along the mesh horizontally. As it reaches the core that is on the same column as the targeted core, the path turns vertically until the destination is reached. The total *communication cost* of communication channel c_o in the mesh can be represented by the length of its path and the potential blockage it may cause when it occupies a number of routing channels

$$f_{comm}(c_o) = \alpha f_{lat}(c_o) + \beta f_{blk}(c_o) \ , \tag{5.5}$$

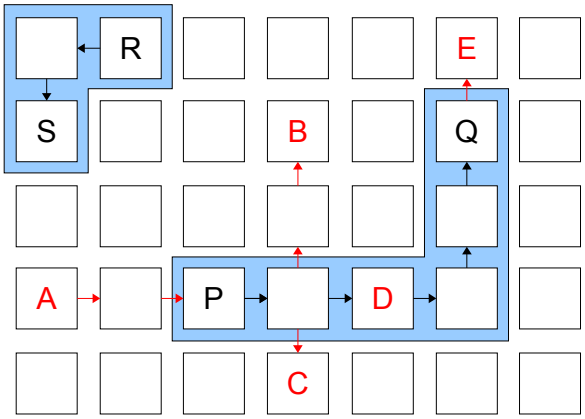


Figure 5.7: Contention for communication paths in a mesh. Communication between $P \rightarrow Q$ can potentially block communication between $A \rightarrow \{B, C, D, E\}$ Communication between $R \rightarrow S$ has less probability of causing blocking than $P \rightarrow Q$.

where α and β are the normalization factors of latency and blockage costs respectively. The magnitudes of each term in (5.5) may be different by orders of magnitude. The normalization factors are needed in order to control the weight that $f_{lat}(c_o)$ and $f_{blk}(c_o)$ have on the overall cost calculation.

Blocking occurs when multiple packets are routed along the same path and contention occurs at an output port of a switch. Figure 5.7, shows example wormhole routes which are blocked by the communication route between nodes P and Q.

5.3.2 Latency Cost Function

One of the major reasons for using NoCs is to minimize communication latencies between communicating cores. It is thus important to minimize the *latency cost* of a wormhole communication c_o from node locations (x_p, y_p) to (x_q, y_q). Using *xy* routing, the path taken is deterministic and provably shortest, given by (5.6) and is thus an important metric to be minimized.

$$f_{lat}(c_o) = |x_p - x_q| + |y_p - y_q|. \tag{5.6}$$

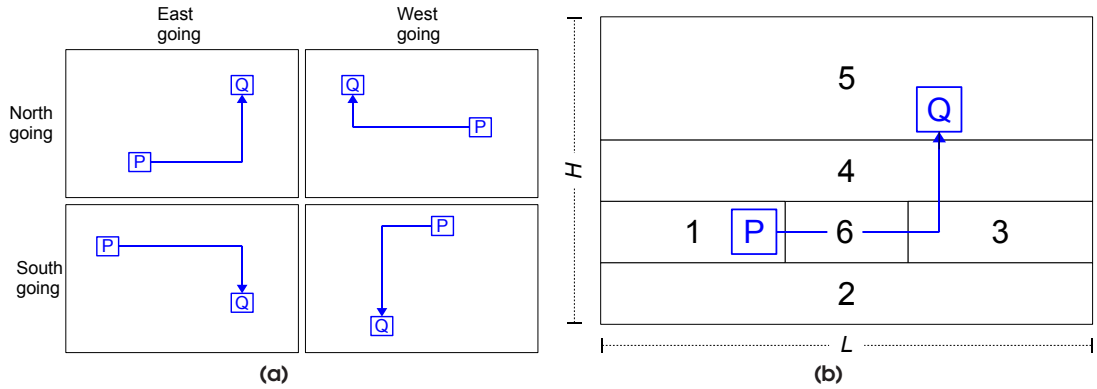


Figure 5.8: Possible communication channels and blockage patterns on a mesh. (a) Possible communication channels from node P to Q on the mesh. (b) 6 zones correspond to different blockage patterns due to given communication channel $c_o : P \rightarrow Q$.

5.3.3 Blockage Cost Function

The communication costs due to the network congestion (blocking), among different paths must be accounted for. To quantify the potential amount of network congestion on the mesh due to a given wormhole communication c_o, the ratio of communications among cores elsewhere on the mesh that are blocked by the given communication to those that are not is considered. Hence, the *blocking cost* for c_o on the entire $H \times L$ mesh is

$$f_{blk}(c_o) = \sum_{all\ \ nodes} \frac{N_{blocked}}{N_{pass}}\ , \tag{5.7}$$

where $N_{blocked}$ and N_{pass} are respectively the number of blocked and permissible destination cores initiated by the i^{th} core on the mesh. Figure 5.8a shows that blocking depends on the relative locations of the given communication route $c_o : (P, Q)$ and the initiation location R. c_o could either be *East*- or *West*-going, *North*- or *South*-going. To evaluate (5.7), the mesh is divided into 6 zones denoting the possible locations of R relative to c_o, as shown in Figure 5.8b. The number of nodes, either reachable and unreachable (as blocked by c_o) from R in each of the zones are determined. Thus (5.7) can be restated as (5.8) and (5.9), where $N_{zone}(i)$ and N_{mesh} are the number of cores belonging to zone i and the total number of

Table 5.1: N_{zone} and $N_{blocked}$ values for zones 1 to 3 for $P : (x_p, y_p) \rightarrow Q : (x_q, y_q)$

i	$N_{zone}(i)$	$N_{blocked}(i)$
1	E: x_p W: $L - x_p + 1$	E: $(L - x_p)H$ W: $(x_p - 1)H$
2	N: $(y_p - 1)L$ S: $(H - y_p)L$	N: $H - y_p$ S: $y_p - 1$
3	E: $L - x_q + 1$ W: x_q	N: $H - y_p$ S: $y_p - 1$

cores on the mesh respectively.

$$f_{blk}(c_o) = \sum_{i=1}^{6} f_{blk}(c_o, i) \tag{5.8}$$

$$f_{blk}(c_o, i) = \frac{N_{blocked}(i)}{N_{pass}(i)} N_{zone}(i) = \frac{N_{blocked}(i) N_{zone}(i)}{N_{mesh} - N_{blocked}(i)} \tag{5.9}$$

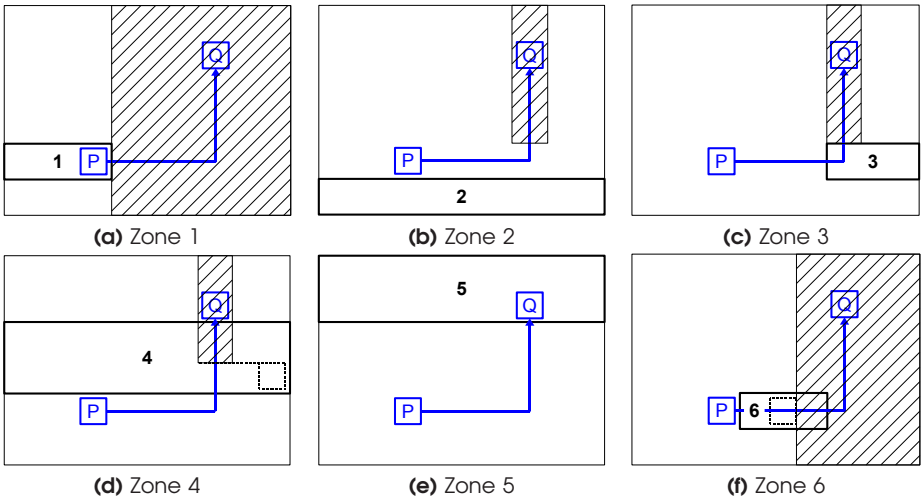
Let the origin of a $H \times L$ mesh be at the lower-left, denoted $(1, 1)$. Suppose a channel c_o established from *origin core* $P : (x_p, y_p)$ to *destination core* $Q : (x_q, y_q)$ is active, we define:

- c_o is *East*-going if $x_q \geq x_p$; *West*-going if $x_p > x_q$.
- c_o is *North*-going if $y_q \geq y_p$; *South*-going if $y_p > y_q$.

The following subsections consider the blockage of *xy* routing in each zone for all relative locations of P and Q.

If c_o is East-going, and if core R in the x_p^{th} row would like to establish a new communication channel to cores elsewhere on the mesh, then only cores located in columns left of P are reachable. Figure 5.9 illustrates the reachable and blocked regions on the mesh. In general, values of N_{zone} and $N_{blocked}$ for R in zone 1 to 3 are listed in Table 5.1, according to the orientation of c_o. Thus, $f_{blk}(c_o, i)$ for $i = 1, 2, 3$ can be determined.

If R is in zone 4, and c_o is North-going, then all cores except those in column x_q above the row of R are reachable. Figure 5.9d reveals that the blockage pattern



(a) Zone 1 **(b)** Zone 2 **(c)** Zone 3

(d) Zone 4 **(e)** Zone 5 **(f)** Zone 6

Figure 5.9: Channels initiated from various zones to shaded regions being blocked by channel in use.

is dependent on the row location of R. In general,

$$f_{blk}(c_o, 4) = \begin{cases} \sum_{j=y_p+1}^{y_q-1} \frac{(H-j-1)L}{HL-(H-j-1)}, & c_o \text{ North-going} \\ \sum_{j=y_q+1}^{y_p-1} \frac{(j-1)L}{HL-(j-1)}, & c_o \text{ South-going} \end{cases} \tag{5.10}$$

Since c_o does not block any new channel from zone 5, $N_{blocked}(5) = 0$ and thus, $f_{blk}(c_o, 5) = 0$. In zone 6 (Figure 5.9f), if c_o is East-going, all nodes to the right of R are unreachable. In general,

$$f_{blk}(c_o, 6) = \begin{cases} \sum_{j=x_p+1}^{x_q-1} \frac{L-j-1}{j+1}, & c_o \text{ East-going} \\ \sum_{j=x_q+1}^{x_p-1} \frac{j-1}{L-j+1}, & c_o \text{ West-going} \end{cases} \tag{5.11}$$

5.3.4 NoC Communication Optimization

Using the heuristic optimization algorithm called *simulated annealing* (SA) [126], the task assignment on homogeneous meshes using deterministic wormhole routing is optimized. The goal of the optimization is to minimize the blocking cost associated with a solution (task assignment). Let p_i be the i^{th} trial task assign-

ment; its associated *cost* f_{comm} is given by

$$f_{sch}(p_i) = \sum_{n=1}^{N} \frac{1}{\sigma(c_n)} f_{comm}(c_n) \ , \tag{5.12}$$

where c_n, $n = 1, ..., N$ are communication links in p_i, and σ is its priority.

The SA algorithm iteratively solves (5.12), searching the solution space for a task assignment with minimal cost. The simulated annealing implementation described in Section 5.2.3 has been modified to include the cost associated with contention when performing task-assignment optimization.

Chapter 6

NoCsim Simulation Study

The architectures presented in Chapter 4 were simulated using the NoCsim simulation platform and companion tools discussed in Chapter 5. The simulation platform was used to perform architectural exploration and to compare the performance of the different architectures relative to each other. The traffic distributions, latencies, and hop counts were compared, as well as the effects of bridge placement and adaptive routing (for the augmented architecture). Also, the effect of blocking aware task assignment on the normal mesh architecture is presented. The simulation results obtained are presented in the following sections.

6.1 Express Rings

As previously mentioned, the *tiled* architecture shown in Figure 4.2 consists of several instances of a mesh which has had a single ring added to it (Figure 4.1). Before presenting simulation results for the more complex architectures, the results obtained when simulating the simpler architecture of Figure 4.1 are first discussed. A mesh size of $N = 10$ was simulated using a randomly generated task graph to generate traffic on the network. The resulting traffic patterns for the normal mesh and a mesh with a single ring consisting of 4 bridge components are shown in Figure 6.1a and Figure 6.1b, respectively. Interestingly, while the mesh exhibits congestion in the center, the center region of the express-ring

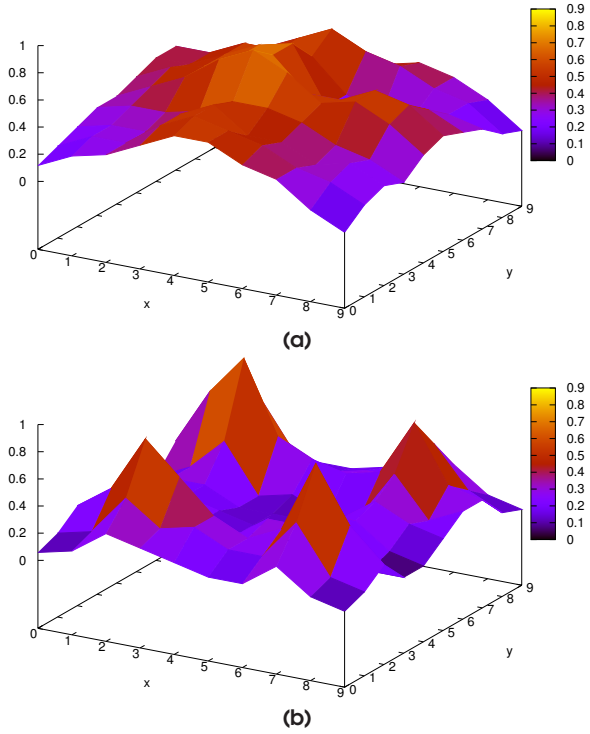


Figure 6.1: Distribution of traffic for a network size of $N = 10$. (a)The mesh architecture exhibits congestion in the center. (b) The single express ring topology exhibits spikes that correspond to bridge components which route traffic upwards onto the ring.

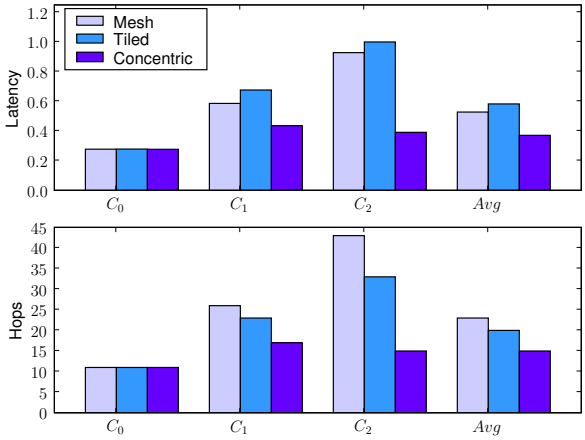


Figure 6.2: Simulation results showing latencies (normalized) and hop-counts for the normal mesh, tiled express rings, and concentric express rings architectures for $N = 36$. The concentric architecture performs best, and the tiled architecture has longer average hop-counts which can be attributed to difficulty finding an optimal routing strategy.

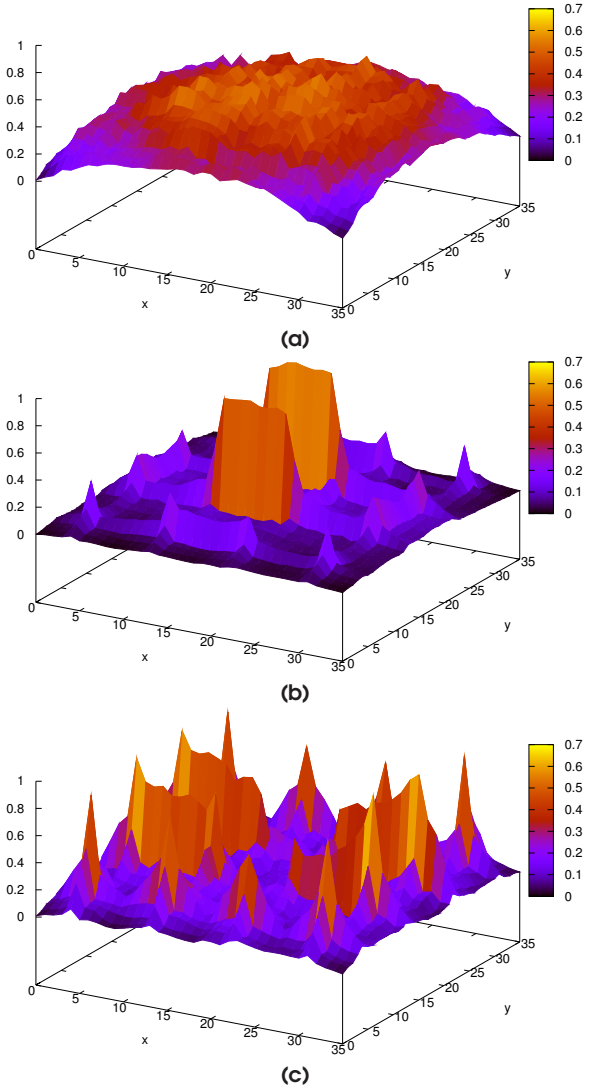


Figure 6.3: Traffic distribution for the mesh, tiled, and concentric architectures for $N = 36$. (a) The mesh architecture exhibits congestion in the center. (b) The tiled architecture has a high level of congestion at bridge locations in the center of the mesh. (c) The concentric architecture shows congestion points evenly distributed through the mesh, and the resources in the mesh are relatively uncongested.

architecture is relatively free of congestion. Since the mesh size used was relatively small, the hop-counts and latencies for both architectures were similar, but the traffic patterns are quite different. As the mesh size is scaled up, one would expect that the performance would follow the trends shown in Figure 4.6.

Figure 6.2 shows the hop counts and latencies for the mesh, tiled and concentric architectures for $N = 36$. The tiled architecture has a slightly higher latency for C_1 and C_2 traffic while exhibiting lower hop counts. This can be attributed to delays due to congestion in the center of the mesh. Figure 6.3b shows that there is a large amount of traffic which passes through the center of the mesh. This can be attributed to the fact that when global traffic travels between rings, the tiles in the center of the mesh must route traffic between bridge components belonging to different rings. Traffic from the outer portions of the mesh is quickly routed through the individual rings to the center of the mesh, effectively concentrating traffic in the center of the mesh. On the other hand, Figure 6.3c shows congestion points evenly distributed through the mesh, and the resources in the mesh are relatively uncongested. This results in lower hop-counts and latencies when compared to the normal mesh architecture because localized traffic (short distance) traffic encounters relatively little congestion. The increased performance of the concentric architecture can be attributed to the fact that global traffic does not tend to accumulate in one area, as with the tiled architecture.

6.2 Comparison of the Hybrid, Augmented, and Mesh Architectures

To investigate the suitability of the hierarchical ring interconnect for global routing, simulations were performed for increasing sizes of N, and the results for all three architectures were collected. Figure 6.4 shows that as the size of the mesh increases, the latencies and hop-counts of all the architectures also increase. An interesting point in the figure is that when $N = 44$, the average latency of the hybrid interconnect becomes larger than for the mesh or augmented mesh architectures. This can be attributed to the fact that the amount of traffic passing through the ring interconnect increases quadratically with N, and at some point,

Table 6.1: Latencies (normalized to 100) for the Hybrid, Augmented, and Mesh architectures for different values of N.

	Hybrid			Augmented			Mesh		
N	C_0	C_1	C_2	C_0	C_1	C_2	C_0	C_1	C_2
20	10	22	23	17	23	23	16	31	47
28	10	27	29	20	28	29	20	41	66
36	15	36	37	25	34	35	24	52	82
44	20	78	79	33	49	51	29	62	100

the hierarchical rings will reach a saturation point where offered throughput will degrade due to congesting at the ingress and egress points (bridge components). As N becomes larger, the hierarchical rings can be scaled up in several ways. First, the cardinality k can be increased, which will have the effect of reducing the sub-mesh size. A second approach is to simply add more local rings or to even add a second hierarchical ring interconnect, for large N.

Table 6.1 shows how the latencies varied for both architectures. Contrary to expectation, the latencies for local traffic (type C_0) passing through the hybrid interconnect is actually less than that observed in the normal mesh topology. This phenomenon can be explained by the fact that routing global traffic *upwards* through the hierarchical rings frees up resources at the lowest level of the network hierarchy. Conversely, the augmented architecture exhibits similar latencies for C_0 traffic as compared to the mesh because less traffic is routed upwards as compared to the hybrid topology, since there always exists two paths between a sender-receiver pair in the augmented architecture. Even though the hybrid topology outperforms the augmented architecture for C_0 traffic, it can be seen from Figure 6.4 that the augmented architecture actually scales better than the hybrid. This is due to the fact that there is less congestion on the global rings because less traffic needs to be routed upwards as can be seen in Figure 6.5, where the percentage of C_2 traffic is much higher for the hybrid than for the other two topologies. This can be attributed to a sub-optimal mapping of the task-graph, which will be discussed further in Section 6.6.

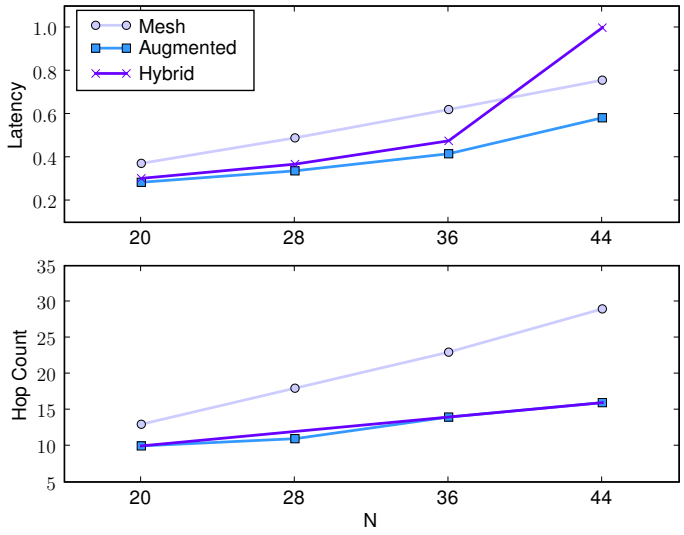


Figure 6.4: Average latencies (normalized) and hop counts for increasing values of N. The performance of the hybrid architecture decreases dramatically for $N = 44$ due to the quadratic increase in traffic being sent over the global interconnect.

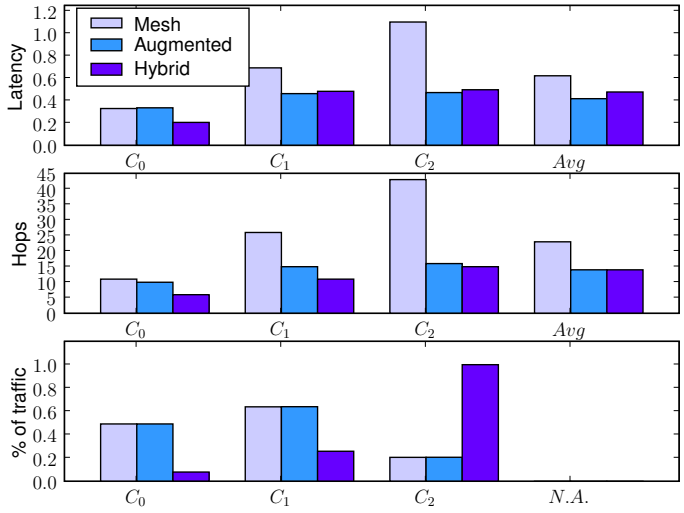


Figure 6.5: Latencies and hop counts for each traffic type for a mesh size of $N = 36$. The bottom graph shows the percentage of the total traffic for each category.

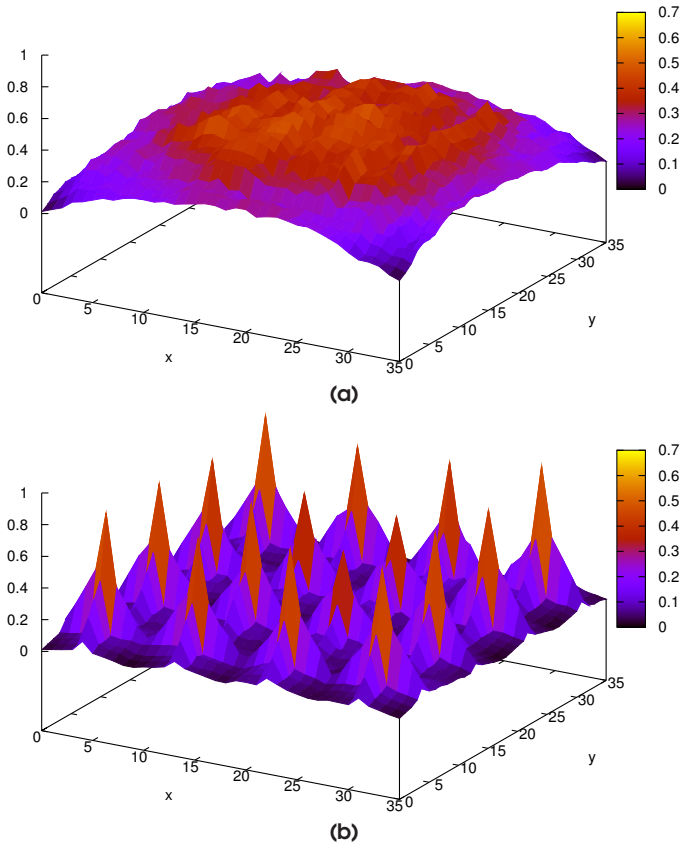


Figure 6.6: Distribution of traffic for a network size of $N = 36$ (a) The mesh architecture exhibits congestion in the center. (b) The hybrid architecture exhibits spikes that correspond to bridge components which route traffic upwards onto the hierarchical ring network. The augmented architecture (not shown), exhibits a similar traffic pattern to that of the hybrid.

Table 6.2: Hop counts for the Hybrid, Augmented and Mesh architectures for different values of N

	Hybrid			Augmented			Mesh		
N	C_0	C_1	C_2	C_0	C_1	C_2	C_0	C_1	C_2
20	3	7	12	7	11	12	6	15	24
28	3	9	14	8	13	14	9	20	34
38	6	11	15	10	15	16	11	26	43
44	7	13	17	13	17	18	14	32	53

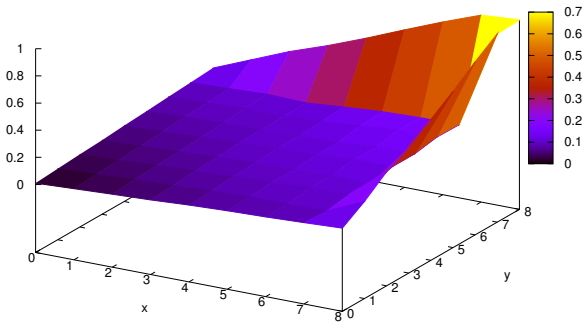


Figure 6.7: Traffic patterns for a sub-mesh of size $N_{sub} = 8$ and for a mesh size of $N = 36$ where the bridge component is located on a corner tile. Congestion is highest at the corner where the bridge component has been placed.

6.3 Bridge Placement in the Hybrid Topology

The placement of the bridge component relative to the edges of a sub-mesh can have a significant impact on performance. When the placement is made at a corner tile, the number of input ports from the mesh is limited to two. Moving the location of the bridge component away from the edge increases the number of available input ports to four, thereby decreasing congestion at the ingress and egress points of the global interconnect. In [18], it was observed that when the bridge was placed in a location away from the edge but still in the relative corner of the sub-mesh, the xy-routing algorithm caused global traffic (types C_1 and C_2) to be routed away from the center of the sub-mesh and towards the edges. This resulted in less congestion in the center of the sub-mesh thereby resulting in lower latencies for local traffic. Figure 6.7 shows how the majority of traffic is routed along towards the edges and the corner where the bridge component is located. The figure shows that most of the traffic is concentrated in the corner of the sub-mesh, leaving the center of the mesh relatively free from congestion. The drawback is that the few corner tiles near the bridge component see a disproportionate amount of traffic. Moving the bridge away from the corner will yield better results as shown in Figure 6.8, where the latencies and hop-counts have been reduced.

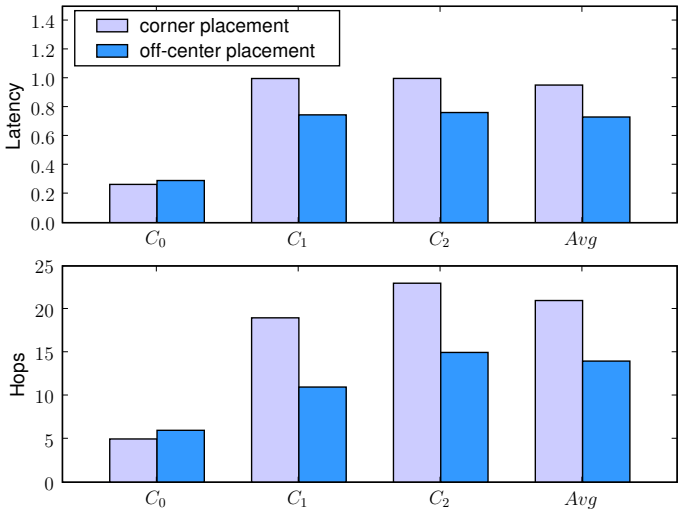


Figure 6.8: Effect of bridge placement on hop-counts and latencies for a mesh size of $N = 36$. Moving the bridge away from the corner reduces latencies and hop counts.

6.4 Enhanced Hybrid Architecture

Using the SystemC library of components, the enhanced architecture described in Section 4.4.0.1 was implemented. It is possible to implement the hybrid interconnect by making some modifications to the architecture from [17], but for simulation purposes, it was simpler to instantiate an extra ring-interface component for every sub-mesh and to connect them together to form the inner ring as shown in Figure 4.7.

Simulations for a mesh size of 32 were performed, and the results obtained are shown in Figure 6.9. It can be seen that the latency for the hybrid interconnect has improved for types C_1 and C_2 traffic. What is interesting is that the enhanced hybrid interconnect shows a significant latency reduction for type C_2 traffic. Since traffic types C_1 and C_2 are unaffected by each other's stop signals, an overall decrease in both their latencies is observed.

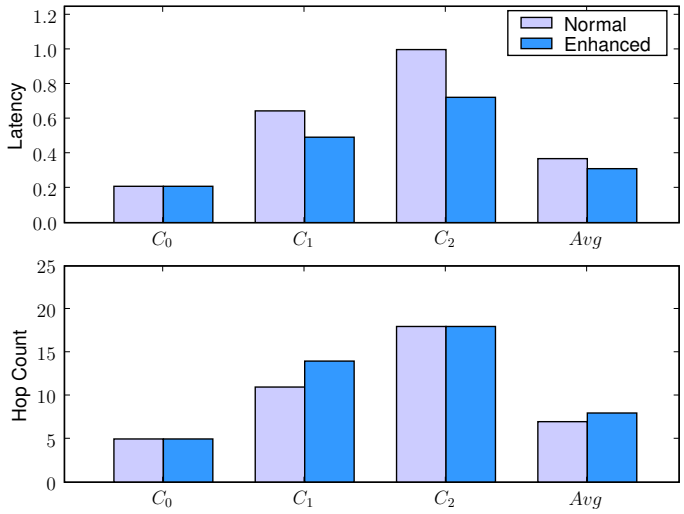


Figure 6.9: Performance improvement of the enhanced hybrid over the normal hybrid architecture for $N = 32$. A reduction in the latencies for both C_1 and C_2 are observed for the enhanced hybrid architecture.

6.5 Applying Adaptive Routing to the Augmented Architecture

In addition to comparing the effects that the hierarchical ring interconnect has on the latency for global traffic, the impact of different routing algorithms are investigated. The simulation platform described in [18] used *xy-routing* to route traffic. Several routing alternatives have been added to the architecture in order to further increase performance.

The wormhole-routed mesh uses *xy* routing to route data. Its advantage is that it is deterministic and optimal in that packets always take the shortest path. More complex *adaptive* routing methods have been shown to not perform as well as *xy* routing [128], which is counter-intuitive because they try to route traffic around congestion and reduce latencies. It was shown in [128] that this can have the opposite effect.

When the hierarchical rings are combined with the wormhole routed mesh, a modification of the *xy* routing strategy is needed such that the global traffic can be routed to a bridge component and through the hierarchical ring interconnect. A deterministic shortest path algorithm is used, where packets are routed towards

the nearest bridge component if the packet is classified as a global packet. The modified *xy* routing strategy is termed *xyb* routing.

The augmented architecture can suffer from congestion at the bridge components if there are too many packets arriving at once. This is a phenomenon common to all hierarchical architectures, but unlike a tree-like structure, the augmented architecture is not limited to a single path between nodes. Figure 4.4 shows that there are two possible paths between any two nodes on the mesh:

1. Packet routed through the mesh normally
2. Packet routed through the hierarchical ring interconnect.

In an effort to reduce the latencies due to blocking at the bridge components, a routing scheme has been implemented, termed *adaptive-xyb* routing (which is a form of deflection routing). If a packet is blocked at a bridge due to congestion, the bridge component will forward the packet through the mesh normally instead of letting it wait. By using a flag bit in the header flit, the next switch to route the packet will be informed to forward it normally instead of sending it back to the bridge station that it came from.

To improve the latencies incurred by global traffic, simulations comparing the *xyb* and *adaptive-xyb* routing were performed. The goal of applying the adaptive routing is to relieve congestion at the bridge nodes by forwarding global packets through the mesh normally. The decision to change the routing strategy is made locally by the bridge component. When a packet is waiting in a buffer, the bridge component keeps track of how long the packet has been blocked. After some threshold value has been exceeded, the bridge component marks the packet and forwards it locally instead of up through the hierarchical rings. The threshold value will have a large impact on the system performance:

- As the threshold value increases, the performance will converge towards that of the normal *xyb* algorithm.
- As the threshold value decreases, the performance will converge towards that of the normal *xy* routing used by the mesh as very many packets get adaptively routed.

The value of the threshold is related to the aggressiveness; the lower the threshold value, the more aggressive the algorithm. Figure 6.10 shows the results obtained when comparing the *xyb* and *adaptive-xyb* routing strategies. Figure 6.10a shows that when using a less aggressive adaptive routing approach, there is a reduction in latencies for C_1 and C_2 traffic, while their associated hop counts are marginally increased. Conversely, Figure 6.10b shows that using the adaptive routing strategy more aggressively results in a larger improvement in latency at the expense of a marked increase in hop counts, which can be attributed to a large amount of traffic being routed through the hierarchical rings to avoid congestion. Although the adaptive-xyb algorithm has resulted in an increase in hop counts, they are still smaller than the counts obtained for the normal mesh architecture in Section 6.2. The results in Figure 6.10 show that the adaptive-xyb algorithm can achieve a compromise between hop count and latency by tuning its aggressiveness.

As previously mentioned, the application of adaptive routing does not always lead to a performance increase [128]. As is evident from the simulation results presented, the aggressiveness of the algorithm must be tuned on a per-application basis because each application will exhibit a different traffic pattern. Since deterministic routing is more general, and usually performs well enough, it is the most often used routing strategy in the NoC literature. However, the extra effort required to tune an adaptive routing strategy may be practical if a reasonably large performance gain can be achieved as shown in Figure 6.10.

6.6 Improved Task Assignment

As previously discussed in Section 6.2, a non-optimal mapping of a task graph can lead to a large proportion of the total traffic being routed through the hierarchical rings. As can be seen in Figure 6.4, the system performance will degrade at some large enough value of N because of congestion at the ingress and egress points of the ring network. Using an adapted version of the simulated annealing in [20] where (5.3) was minimized, a more efficient mapping of a task graph was made for a mesh size of $N = 44$. Figure 6.11 shows how the latencies and hop counts are affected for each architecture. The mesh architecture experienced the

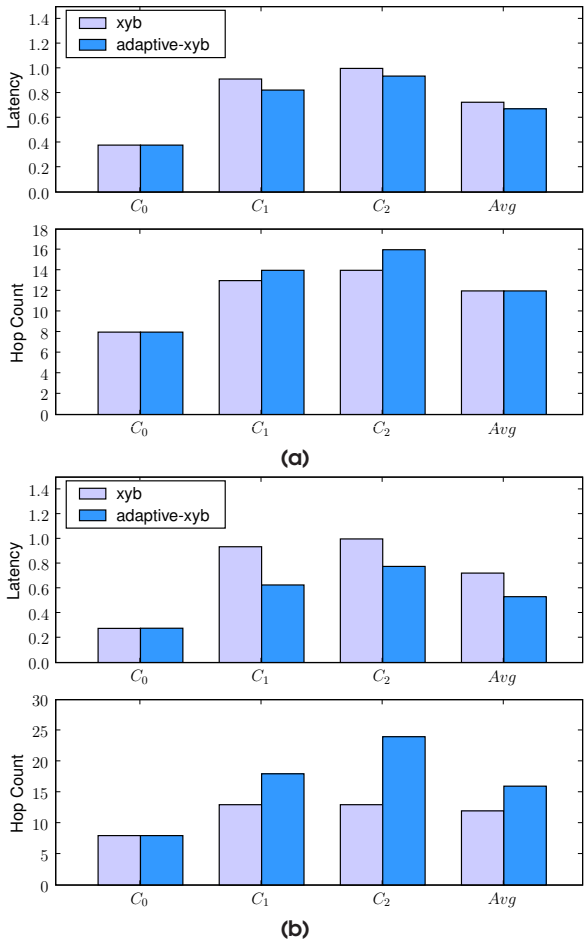


Figure 6.10: Comparison of the *xyb* and *adaptive-xyb* routing for the augmented architecture ($N = 28$). (a) Less aggressive adaptive routing, (b) More aggressive approach.

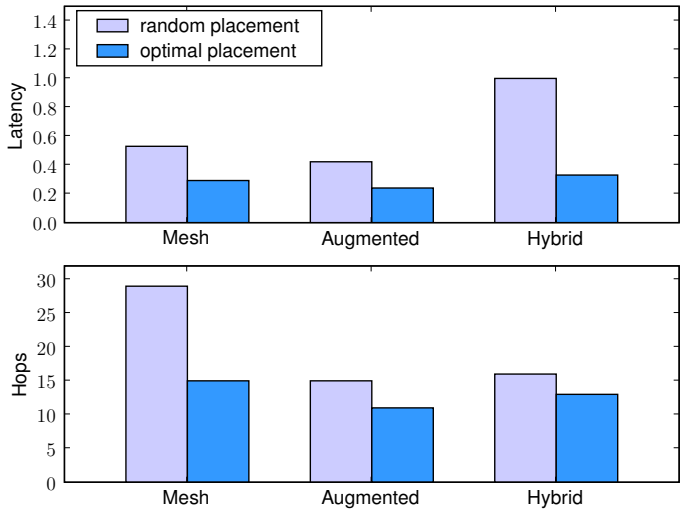


Figure 6.11: Performance improvement of all three architectures when a more efficient task assignment is used. The average latencies and hop counts have been decreased for each architecture.

largest decrease in average hop counts, but most significant is the 67% reduction in latency achieved by the hybrid interconnect. From Figure 6.4, we saw that for $N = 44$, the performance of the hybrid architecture was degrading because of the quadratic increase in traffic being sent over the interconnect. The efficient mapping achieved by the simulated annealing algorithm has yielded performance characteristics comparable to the mesh and augmented mesh. It can therefore be concluded that the scalability of the hybrid interconnect can be increased by intelligent task assignment.

6.7 Blocking Aware Task Assignment

Experimental results were obtained by running the simulated annealing algorithm for one million iterations for a task graph consisting of 80 tasks and communication events respectively. The task graph was mapped onto a 10×10 mesh. The communication events between tasks were randomly generated such that each task was constrained to initiating one communication but could itself be the target of multiple communications. The results summarized in Figure 6.12 shows that the lowest combined communication cost (i.e. the sum of the latency and

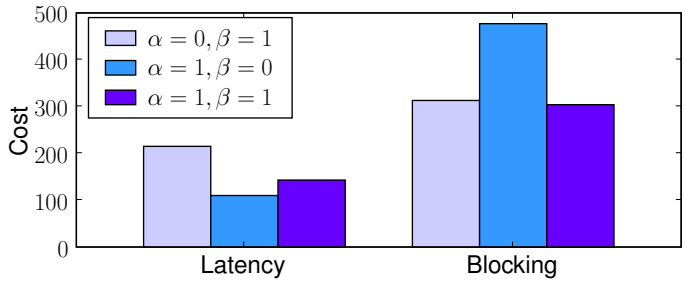


Figure 6.12: Latency and blocking costs for optimization results after 10^6 iterations of applying simulated annealing to (5.5) for blocking only $(\alpha, \beta) = (0, 1)$, latency only $(\alpha, \beta) = (1, 0)$, and both $(\alpha, \beta) = (1, 1)$ for a 10×10 mesh.

blocking costs) is obtained when both latency and blocking are minimized, that is $\alpha = 1$ and $\beta = 1$ in (5.5).

6.7.1 Effect on Latency

A task graph consisting of 36 vertices and 144 edges was generated using the *Gengraph* utility described in Chapter 5. The graph was mapped onto a 6×6 mesh which was then processed by the simulated annealing application for $(\alpha, \beta) = \{(1, 0), (1, 1)\}$. As discussed in Section 5.3.1, setting the normalization factor β to zero means that the optimization performed by the simulated annealing application minimizes only the length of communication while disregarding blocking. When $(\alpha, \beta) = (1, 1)$, the optimization considers both criteria and attempts to find a compromise between minimizing communication distances while also reducing blocking.

The results of the experiments are shown in Figure 6.13, where Figure 6.13a shows the magnitude of the cost function for each term in (5.5) for the optimized solution. The figure also shows the percent improvement of the solution over the cost of the initial random mapping of the task graph onto the mesh.

Figure 6.13b shows that the average latency for type T_1 traffic is reduced at the expense of an increase in the latency of T_2 traffic when $(\alpha, \beta) = (1, 1)$. While the increase in the average latency for T_2 is large, the effect on the total average latency is negligible because the total amount of type T_2 is small compared to the other types. Conversely, the latency reduction for both T_0 and T_1 traffic results in a 5% decrease in the total average latency because they represent a larger

percentage of the total traffic, as can be seen in the bottom part of Figure 6.13b.

6.7.2 Network Utilization

A well known problem with mesh networks is that similar to large cities, congestion develops in the center while the edges are under-utilized. The effect of minimizing (5.5) with respect to communication distance *and* blocking cost results in some traffic being routed along less used paths in the network.

A task graph consisting of 144 vertices and 600 edges was mapped onto a 12×12 mesh for values of $(\alpha, \beta) = \{(1, 0), (1, 1)\}$. The number of flits appearing at the input ports of each switch were counted and the totals are shown in Figure 6.14, where the x and y directions correspond to the location of a switch on the mesh, and the z direction corresponds to the number of flits normalized to 1. When only communication distance was minimized, Figure 6.14a shows that traffic was concentrated in the center of the mesh, leaving the switches along the edges under-utilized. When blocking was taken into consideration, Figure 6.14b shows that the network utilization was more evenly spread over the entire mesh, including the edges.

As (5.7) does not take the communication distance between tasks into account, minimizing blocking alone can result in increased latencies due to larger communication distances. The trade-off is that congestion in high-traffic areas is relieved and otherwise under-utilized resources can be better utilized. The challenge lies in selecting proper values for the normalization constants α and β in (5.5) such that the solution achieves the desired balance while maintaining acceptable performance metrics.

6.8 Chapter Summary

Simulation results for several topologies that use rings for global routing to address the scalability issues of mesh networks were presented. Simulation results for the *express ring* topologies, which use one or more unidirectional rings for global traffic were also presented. It was observed that the *tiled* approach resulted in congestion in the center of the mesh when global traffic gets routed

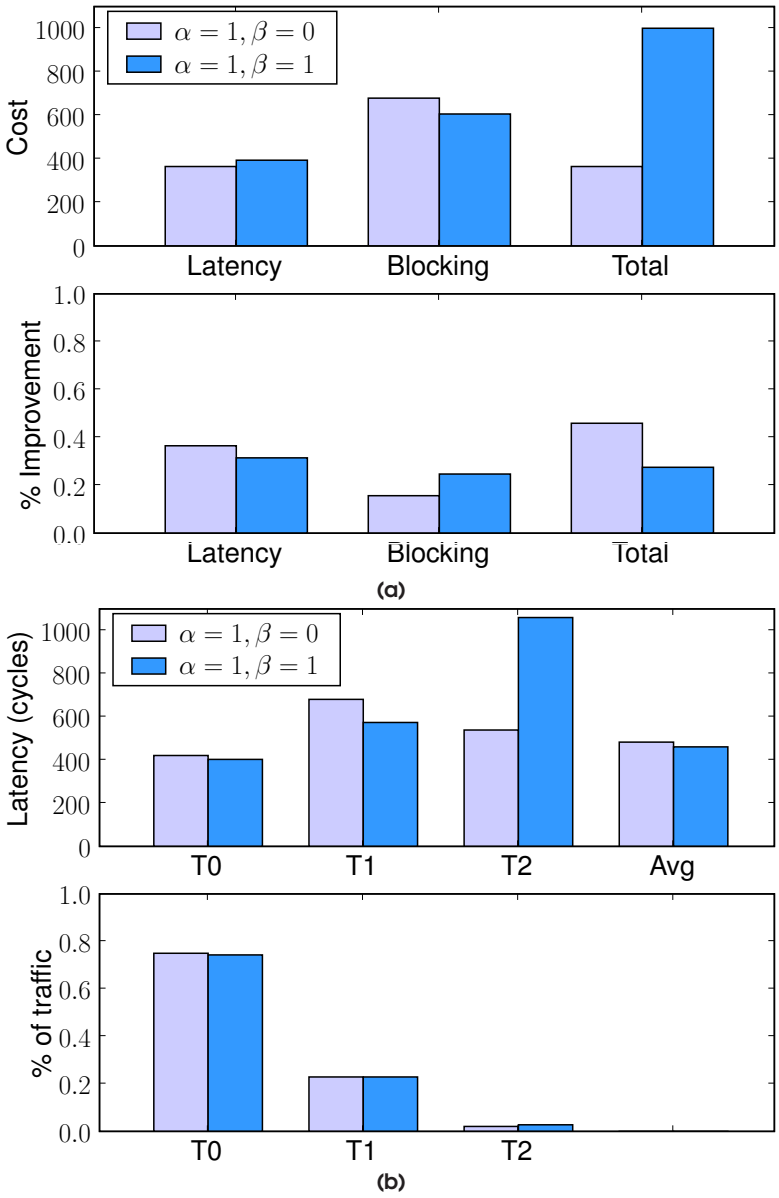


Figure 6.13: Results obtained for a task graph consisting of 36 vertices and 144 edges mapped onto a 6×6 mesh. (a) Cost and percent improvement of (5.12) obtained via simulated annealing over the cost of the initial random mapping. (b) Average latencies for each traffic type $\{T_0, T_1, T_2\}$ obtained by SystemC simulation.

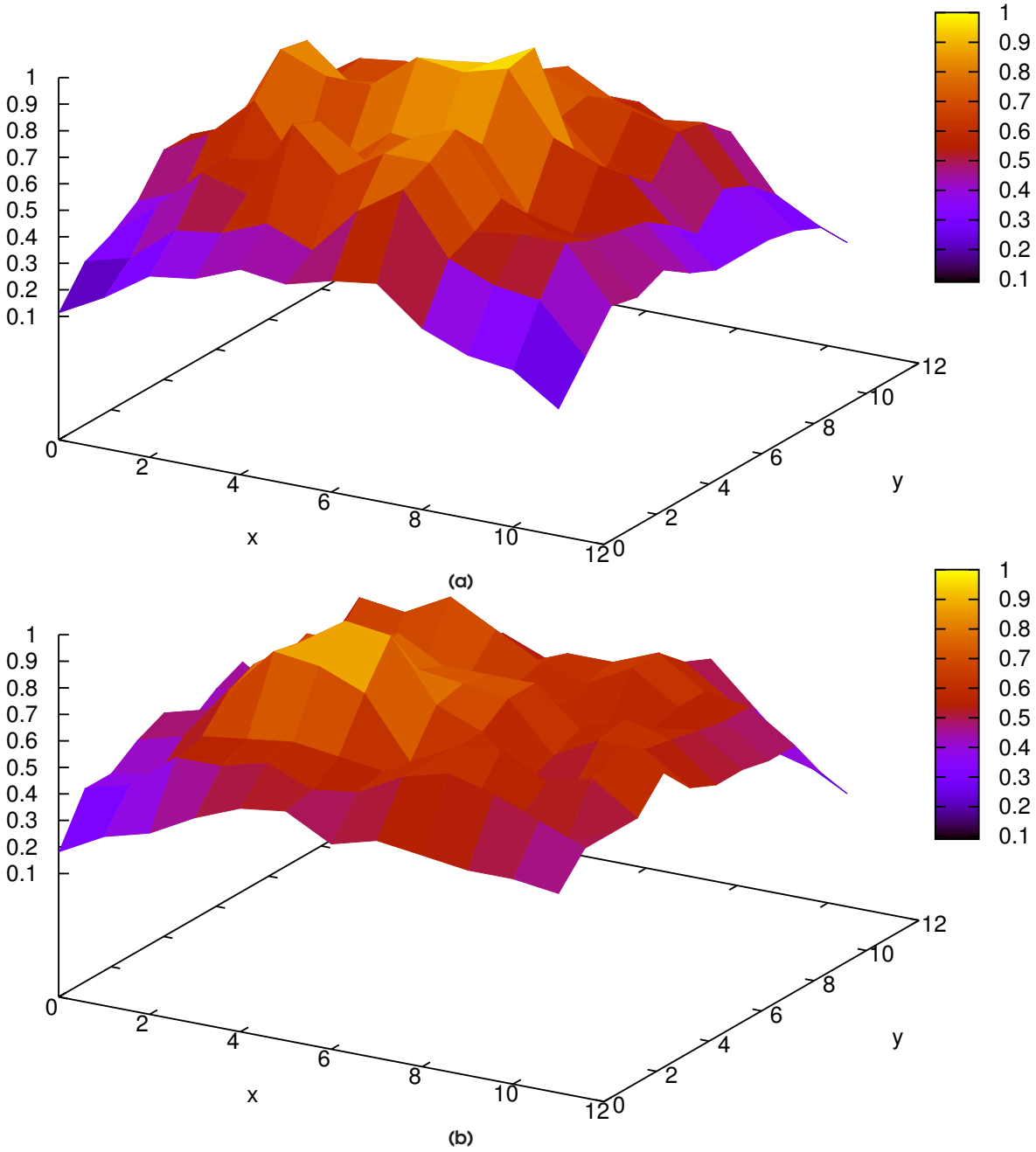


Figure 6.14: Traffic characteristics obtained for a task graph consisting of 144 vertices and 600 edges mapped onto a 12×12 mesh. (a) When $(\alpha, \beta) = (1, 0)$, switches in the center of the mesh are more heavily utilized than edge switches. (b) When $(\alpha, \beta) = (1, 1)$, the network utilization is more evenly spread out.

across the central tiles. The *concentric* rings topology performed better, exhibiting an even distribution of congestion points, and also showed reduced congestion in the center of the mesh.

Simulation results for the *augmented* and *hybrid* network topologies, which use hierarchical rings for global routing, were presented. Combining the hierarchical rings with the mesh topology leverages the strengths of each topology, where the mesh exhibits low latencies for local traffic, and the hierarchical rings exhibited lower hop counts for global traffic. The results showed that the use of hierarchical rings for routing global traffic can have a positive impact on the performance of the interconnect, whereby average latencies and hop-counts are reduced.

For the hybrid architecture, the partitioning of a large mesh into smaller sub-meshes leads to some interesting results. A well known problem with the mesh topology is the fact that traffic hotspots develop in the center of the mesh. In the hybrid topology, the global traffic is routed towards the edges of the sub-mesh thereby reducing the congestion in the center of the mesh, resulting in reduced latencies for local traffic. By placing resources which need to send global traffic closer to the bridge station and taking advantage of locality, latencies and hop counts can be decreased. Furthermore, it was shown that placing the bridge location away from the absolute corner of a sub-mesh, a further gain in performance could be achieved for global traffic without negatively affecting the latencies of local traffic. Lastly, an enhanced version of the hybrid architecture can decrease latencies by providing separate virtual paths through the hierarchical-ring interconnect. As the size of the mesh is increased, the sub-meshes also get larger, which may results in a quadratic increase in traffic that can go through the hierarchical ring interconnect. It can therefore be concluded that there is some optimal size for the sub-meshes before performance will degrade to unacceptable levels due to congestion. The problem could be alleviated by simply scaling up the hierarchical rings, but since a mesh size of approximately 36 resulted in similar performance characteristics for both the mesh and hybrid networks, it is preferable to keep the sub-mesh sizes fixed while increasing the size of the hierarchical rings by adding either an extra level to the hierarchy or more bridge components per local ring. Furthermore, by minimizing the amount of traffic that travels through the ring interconnect, a significant reduction in latency was

achieved. Specifically, the performance of a $N = 44$ hybrid mesh was increased significantly by using simulated annealing to find a more optimal task assignment.

Unlike the hybrid topology, the augmented topology has the property that two paths exist between any two nodes on the network when using deterministic *xy* routing. This property presents the opportunity to consider the application of adaptive routing algorithms in an effort to further improve performance. The application of adaptive routing was shown to yield a more efficient use of network resources by lowering the average latencies by keeping the hop counts lower than the mesh alone. It was shown that the performance of the adaptive routing algorithm is affected by the aggressiveness with which it is applied and that the trade-off between latency and hop-counts can be tuned.

The simulated annealing application was used to obtain task-graph mappings that minimize the amount of contention (blocking) on a wormhole routed mesh network. Simulations comparing shortest path mappings to ones that take blocking into consideration were performed. The results showed that minimizing both metrics can yield improved results over minimizing for lowest latencies (shortest path) only. Furthermore, the simulation results showed that the blocking-aware task assignment approach results in a more even network utilization as compared to a shortest path only mapping, which is characterized by high network utilization in the center of the mesh.

Chapter 7

Energy Modeling and Optimization of the Hierarchical Rings

The energy consumption of the system interconnect is becoming increasingly important. As such, an evaluation of the energy consumption of the hierarchical rings has been included for study, and is presented in this chapter. Previous to the *NoCsim* platform, a high level transactional model [130] was implemented using the StepNP [131] platform. To evaluate the effect of dynamic frequency scaling on the hierarchical ring interconnect, an energy model was integrated into the interconnect model [16]. The energy model adds the capability to monitor the simulated energy usage of the interconnect network for the execution of a particular application program. The practical energy models are in general not intended to provide an exact representation of energy consumption; instead, they are intended to provide a mechanism whereby the effect of varying design parameters on energy consumption can be compared. Hence, the requirement for the energy consumption model is that the various alternatives are considered relative to each other.

7.1 Energy Model

For modeling energy consumption in the hierarchical ring, the main contributors are the ring interfaces with their buffering and more complex logic, and the wire connections between points on all rings. The energy characteristics for the ring

interfaces were obtained by first synthesizing a representative Verilog model of the interface using *Synopsys Design/Power Compiler* to extract an energy estimate for the sequential logic of the interface that is denoted by E_{if}. Note that we assume the energy consumption of a ring-interface and an inter-ring interface are approximately the same. The energy consumption (per clock cycle) of the point-to-point connections of parallel wires between adjacent interfaces is:

$$E_{p2p} = \frac{1}{2} C_L V^2 N_{(0\rightarrow 1, 1\rightarrow 0)} \tag{7.1}$$

where C_L is the sum of wire capacitance C_w and input capacitance C_{in} of a ring interface, and N is the total number of wires experiencing $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions. The total energy per clock cycle of the entire interconnect is therefore a function of the switching activity caused by network traffic. The energy consumed per cycle by all 20 ring and inter-ring interfaces in the architecture shown in Figure 3.1 is

$$E_{sum} = \sum_{1 \leq i \leq 20} \left(E_{if,i} + E_{p2p,i} \right) . \tag{7.2}$$

The utility of the energy model is for approximating the energy consumption of the interconnect with a fair degree of accuracy in order to understand how much energy the interconnect consumes with respect to the entire system, and to evaluate the effectiveness of optimization schemes.

It was decided to use frequency scaling *without* voltage scaling in the model because it was felt that using an arbitrary quadratic equation would make the results look more impressive than actually possible. Unless a full low-level simulation of the circuit is performed using *HSpice*, it cannot be known for certain how the circuit will react to varying voltage levels. A high-level model that uses voltage scaling could very well assume voltage values that would cause the circuit to fail. For these reasons, it was felt that keeping voltages constant while varying the frequency of operation is a more realistic and conservative approach.

Listing 7.1: Optimization pseudocode showing how the ring speed is governed by the FIFO occupancies.

```
MAXFREQ = GLOBAL_CLOCK_FREQUENCY
OCCUPANCY = occupancy of fullest FIFO on the ring
if(OCCUPANCY > MAX_THRESHOLD) {
    set ring speed to MAXFREQ
} else if(OCCUPANCY > MED_THRESHOLD){
    set ring speed to MAXFREQ/2
} else if(OCCUPANCY > MIN_THRESHOLD){
    set ring speed to MAXFREQ/3
} else {
    the traffic on the ring is very low
    set ring speed to MAXFREQ/4
}
```

7.2 Energy Optimization

The energy optimization scheme varies the ring speeds according to buffer occupancies. If the majority of buffer space is unused, the interconnect clock rate can be reduced to conserve energy. As the buffers begin to fill, the clock rate is increased in order for the rings to consume buffered packets more rapidly and reduce the buffer occupancy. The FIFOs of all the interfaces on a ring are examined and the occupancy level of a ring is taken to be the maximum occupancy of any single FIFO on the ring. The pseudo-code for the optimization algorithm is shown in Listing 7.1 (note that each ring is controlled independently). The threshold constants need to be fine-tuned in order to avoid being over-aggressive which will cause many backpressure signals and will ultimately impact execution times negatively.

The parameters that relate the ring speeds to the global clock are the *local divisor* and *central divisor* parameters, which are the factors by which the respective clocks are slowed down. The energy consumption of an interface is linearly dependent on the frequency f, and is given by

$$\tilde{E}_{if} = \frac{E_{if}}{k_f} \ , \tag{7.3}$$

where k_f is the central or local divisor used to lengthen or shorten the clock period of the rings. For example, multiplying the period by $k_f = 2$ will lead to

a reduction of 50% in the energy consumption of a ring interface. (7.2) can now be expressed as

$$\tilde{E}_{sum} = \sum_{1 \leq i \leq 20} (\tilde{E}_{if,i} + E_{p2p,i}). \tag{7.4}$$

It is important to realize that slowing the rings down by some divisor constant k does not simply result in the simulation time being lengthened by the same constant, which would result in the same overall energy consumption. When no data is being sent through the rings, energy is still being consumed by the ring interfaces, so scaling down the clock signals will result is energy savings. The optimization scheme being used is meant to take advantage of the communication characteristics of many parallel applications which often start with a period of intense communication between nodes where data is being sent for processing. Once all the nodes have sent/received their data, there will be a period of time where the traffic on the interconnect is low while the nodes process their data.

The addition of the energy model to the StepNP model provides a facility to study the energy consumption of the system over time and the effect of different optimization algorithms with respect to energy consumption and average power.

7.3 Simulation Results

The StepNP model provides an ARM processor core *instruction set simulator* (ISS)[1], which is used as the PE connected to each ring-interface. Application code written in C can thus be executed on the ARM cores after compilation using a cross-compiler. A memory mapped *direct memory access* (DMA) facility enables data to be sent and received over the interconnect. For the interconnect, all FIFO depths were configured to be 16 flits deep unless specified otherwise.

To generate traffic on the interconnect, three synthetic test-benches and a parallel matrix transposition application were used. The three synthetic test-benches are:

- *Synthetic Low*: each station transmits a stream of data to one corresponding station in the farthest ring.

[1]The provided ISS is *not* cycle accurate.

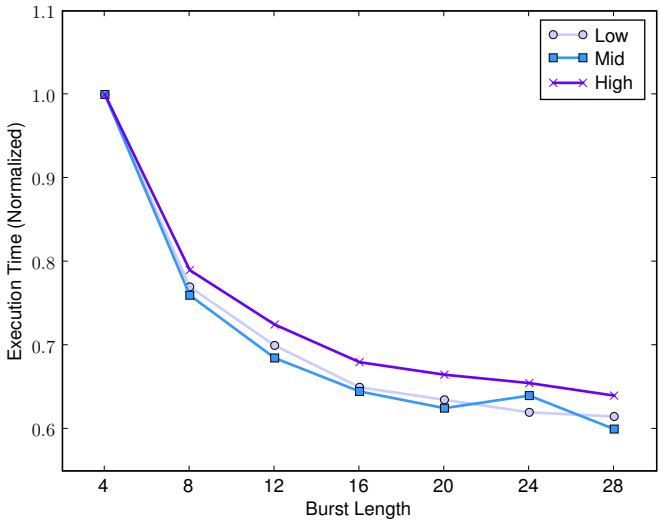


Figure 7.1: Execution time (normalized) for different burst length (words) values. Increasing has a large impact on the execution time, but the reduction diminishes as the length is increased.

- *Synthetic Mid*: each station multi-casts a stream of data to all stations in the farthest ring.
- *Synthetic High*: broadcasts a stream of data to all stations in the system.

7.3.1 Data Burst Length and FIFO Depths

The burst length is the number of words a station could transmit at one time with each DMA send operation. A small burst length can cause sub-optimal network utilization whereas a large burst length can potentially flood the network and cause the assertion of many backpressure signals. Because each transmit operation is matched with a receive operation in message-passing systems, data burst length also affects the amount of memory that must be allocated to stage data for transmission or to store the received data. Selecting a large burst length would mean that a large amount of memory may be reserved unnecessarily. Figure 7.1 illustrates the effect of different data burst lengths for the three synthetic test-benches. It can be observed that increasing the burst length can have a large impact on the execution time, but the reduction diminishes as the length

is increased. For the three test-benches, the reduction in execution times becomes relatively constant for burst sizes larger than 16. Further increases in burst length have negligible effect on performance, but they introduce higher memory requirements for staging the data before transmission and for storing the received data. Interestingly, the optimal burst length corresponds roughly to the size of the FIFOs in the ring-interfaces (16).

The depth of FIFOs located on the switches can affect both the performance and the design area of the system. Table 7.1 summarizes the effect of the FIFO depth on the communication performance for the synthetic testbenches, as well as the matrix transpose program. The performance improvement achieved through increasing FIFO depth is limited for most of the testbenches. The simulation results have indicated that the FIFO depth does not have a significant impact on communication performance in the two-level hierarchical ring network for the matrix transposition application. This phenomenon can be explained by noting that the total execution time is dominated by the time it takes to process the data at each node rather than the communication latencies over the interconnect. Based on the simulation results, good performance can be achieved using the relatively small FIFO size of approximately 16 words.

7.3.2 Energy Optimization

The matrix transpose program was used to obtain the execution time and the energy consumption of the interconnect network. The first part of the experiment finds the optimal burst length for best performance. The second part of the experiment finds the best configuration for the clocking scheme to provide optimal energy characteristics for the architecture.

The results of simulations performed with different data burst lengths for a 128x128 matrix transpose are shown in Table 7.2. Results indicate that short data burst lengths have longer execution times. The execution time is minimized with a burst length of 16 words. Increasing the burst above 16 increases execution time from congestion caused by higher utilization of the interconnect network. Simulations were performed with a burst size of 16, which was previously established to give the best performance. The results of simulations with varying ring

Table 7.1: Effect of FIFO depth on performance using a burst length of 16.

FIFO Depth	Execution Time (ns)			
(words)	High	Mid	Low	Trans
12	644670	466860	412530	740610
16	646290	463500	414420	740970
20	647760	469170	413790	741150
24	647610	466860	409860	740970
28	647550	462180	411690	740970
32	645450	467280	405240	746370
36	645510	472690	410910	746190
40	644670	442290	407760	746190
44	645660	442290	406290	746190
48	645240	442290	406290	746190
Max	647760	496860	414420	746370
Min	644670	442290	405240	740610
% difference	0.47	6.87	2.27	0.78

Table 7.2: Interconnect performance of the 128×128 matrix transpose program for different burst lengths using a FIFO size of 16.

Burst Length (words)	Execution Time (ms)	Energy (uJ)	Power (mW)
1	20.72	884.25	42.67
2	15.17	647.70	42.67
4	12.49	533.39	42.68
8	11.26	481.05	42.69
16	10.96	467.85	42.68
32	11.31	483.06	42.68
64	12.60	537.76	42.67

speeds are listed in Table 7.3. It can be observed that the ring speeds have less impact on the execution time, but can achieve substantial energy savings with slower operating frequencies. The overall execution time can be longer because the latency increases with lower ring operating frequencies. The execution time, however, is not affected significantly by the ring operating frequencies due to the fact that the interconnect network is still able to transmit data with good throughput.

We observe a large reduction in energy usage by simply slowing down the local rings. The energy savings from reducing the local ring clock rate diminishes with a higher local divisor because the static energy consumption becomes dominant. From results listed in Table 7.3, we believe that a local divisor of 2 and a central divisor of 1 is the optimal clocking scheme for the hierarchical-ring multiprocessor system. This result makes sense, as one would expect the system bottleneck to be the central ring.

7.3.3 Dynamic Power Optimization

Simulation results obtained with dynamic power optimization enabled are shown in Table 7.3. Optimization "off" means that dynamic frequency scaling is disabled. Similarly, optimization "on" means that dynamic frequency scaling is enabled. The local/central divisor values set the base (maximum) interconnect speed for dynamic optimization. The optimization algorithm increases the cycle time multiplication factor k_f by integer multiples in order to reduce the interconnect speed when the traffic intensity is low. The optimization scheme takes advantage of the bursty communication characteristics of the matrix transposition program. When the program is performing calculations, the ring clock rate is reduced to conserve energy while traffic is low. When traffic begins to fill the buffers, the ring clock rate gradually increases to accommodate the requirements of the sending nodes. For the matrix transposition program, the dynamic optimization algorithm reduces power consumption by 30–70% over the unoptimized case where the speed remains constant for the same local/central divisor value settings. The largest improvement, not surprisingly, is when the base clock rate for the interconnect is the same as the processor clock rate, i.e., when both

Table 7.3: Effect of varying ring speeds and dynamic optimization for burst length and FIFO size of 16 words. Optimization "on" and "off" means that dynamic optimization is enabled and disabled, respectively.

		Optimization Off		Optimization On	
Local Divisor	Central Divisor	Exec Time (ms)	Power (mW)	Exec Time (ns)	Power (mW)
1	1	10.96	42.68	11.03	11.30
2	1	10.97	23.57	11.07	9.27
2	2	11.05	21.47	11.12	9.07
2	3	11.13	20.80	11.20	8.96
2	4	11.15	20.44	11.29	8.90
3	1	10.99	17.21	11.11	7.94
3	2	11.06	15.09	11.15	7.72
3	3	11.17	14.39	11.23	7.59
3	4	11.15	14.07	11.32	7.52
4	1	11.04	14.03	11.15	6.98
4	2	11.10	11.91	11.20	6.76
4	3	11.19	11.21	11.23	6.62
4	4	11.29	10.86	11.28	6.53
5	1	11.07	12.12	11.20	6.26
5	2	11.12	10.00	11.23	6.06
5	3	11.20	9.29	11.29	5.91
5	4	11.25	8.95	11.29	5.81

divisors are one. The improvement is diminished as the base clock rate is set lower with divisors that are larger than one, although the reduction in energy consumption is still significant.

7.4 Chapter Summary

This chapter has outlined techniques in the modeling and evaluation of a hierarchical ring interconnect for SoC multiprocessor systems using a high-level transactional SystemC and StepNP [131] model. The model has enabled energy/performance tradeoffs of different design parameters to be evaluated, thus guiding design decisions for the low-level RTL implementation discussed in Chapter 8.

Specifically, a simulation model has been used to explore how the FIFO sizes affect system performance. It was shown that increasing FIFO sizes does not necessarily increase performance, and that the sizes can be kept relatively small while still achieving a good network performance. For the chosen architecture and applications, a FIFO size of 16 was determined to yield a good trade-off between size and performance. Furthermore, results also show that dynamically adjusting the speeds of the rings based on buffer occupancies yields significant energy savings without adversely affecting performance, thus validating the effectiveness of the approach.

Chapter 8

RTL Implementation of the Hierarchical Ring Interconnect

Chapter 6 presented simulation results obtained using high-level models written in SystemC. While high-level modeling facilitates architectural exploration, the results obtainable from such abstract representations are limited. On the other hand, a low-level RTL implementation enables designers to study the physical properties of a design such as resource requirements and speed. The effort required to create a RTL implementation is much greater than a high-level SystemC model, but without performing synthesis and place-and-route, it is difficult to verify that the design meets performance and resource requirements, or even if the design is actually feasible. In order to ascertain the suitability for on-chip implementation, the hierarchical rings have been implemented at the *register transfer level* (RTL) level. The implementation consists of approximately 8.5k lines of VHDL code, and required greater effort to implement than the behavioral SystemC models presented earlier.

This chapter presents the RTL implementation details, simulation, synthesis, and place-and-route results. The results will confirm that the architecture is indeed suitable for on-chip implementation. Section 8.3 will show that the RTL implementation of a modestly sized Leon3 based NoC can fit on current FPGAs, and simulation results confirm that relatively small storage buffers still achieve high throughput.

8.1 On-Chip Suitability of a Hierarchical Ring Interconnect

As discussed in Chapter 3 and Section 2.2.2, ring networks have relatively simple routers and require less resources than other, more complex, architectures. Furthermore, the point-to-point connectivity between low-degree nodes results in an implementation devoid of global routing; as will be shown, place-and-route will be more efficient than mesh topologies when targeting *field programmable gate array* (FPGA) implementations. The unidirectional nature of the rings reduces the overhead associated with routing, which results in low latencies and high throughput. Furthermore, the structure of the rings is suitable for multimedia applications where a pipelined approach is often used in the processing of data streams.

Another property that makes rings attractive from an implementation standpoint is the fact that they can be easily partitioned. In the early days of networking, when fabrication technologies were more limited, rings were attractive because the switches had low pin-counts (lower packaging cost) when compared to more complex architectures. The same property can be exploited to efficiently partition and distribute a large design over several FPGAs. This is important because current FPGAs cannot accommodate a SoC consisting of 16 processors connected by the hierarchical rings, let alone a SoC consisting of tens of cores. Section 8.3 will show that even though the RTL implementation is area efficient, the system size that will fit on a relatively large FPGA (Xilinx Virtex II family) is still limited. It is therefore necessary to look at alternative architectures that use several interconnected FPGAs[1].

Partitioning an interconnect requires cutting links, and for each link cut, resources are required to route the signals off-chip. There is therefore a direct correlation with the overhead required to partition a topology and the number of links that must be cut. Partitioning unidirectional rings requires cutting only 2 links, which will require a relatively small overhead. On the other hand, partitioning a large mesh is difficult because of the number of links that must be cut.

[1] A planned multi-FPGA implementation will be discussed in Section 9.1.

If a mesh of width N is cut in half, the number of unidirectional links cut would be $2N$. It can therefore be concluded that the mesh topology is more difficult to partition than a ring topology. In the case of a large mesh network that will not fit on a single chip, the hierarchical rings present an interesting alternative. Instead of using a large mesh, the hybrid topology presented in Section 4.4 can instead be used, such that the partitions are made by cutting the links of the hierarchical rings, while the sub-meshes can be distributed over a number of FPGAs. The hierarchical rings would thus provide the inter-chip (long distance) communication, while still using the mesh for local communication.

As previously discussed, the on-chip interconnection network will be a limiting factor for performance and energy consumption, and in some cases accounting for over 50% of the total energy requirement [9]. Buffers account for a large part of the area and energy requirement of an interconnect, so minimizing the size and number of buffers is an important part of the design process. There is an obvious correlation between the complexity of the interconnect and the resource requirements. It can therefore be concluded that the interconnect architecture should be as simple as possible while still meeting performance requirements. The hierarchical ring implementation discussed here requires less buffers when compared to a four-way mesh router. Furthermore, since the switching logic is simple, the clock rate of the interconnect can be higher when compared to other implementations as will be shown in Section 8.3.

For a SoC implementation using the hierarchical rings for the system interconnect, the critical path will be in the processing elements (PEs) and not the switches that constitute the network. This is attributable to the very few levels of logic used in processing data in the interconnect implementation. Since the complete interconnect resides on the same die and never has to exit the chip, the clock rate supported by the interconnect will be on par with L1 cache. Furthermore, a full clock cycle is allocated in the timing budget for the interconnect routing delays by registering all the input signals and having no shared asynchronous signals that could create a critical path in the interconnect.

As previously discussed, hierarchical rings are well suited to the application of DVS techniques. In [16], it was shown that applying dynamic frequency scaling can result in a significant reduction (30–70%) in power consumption with negli-

gible impact on the execution time for the benchmark presented. Furthermore, supporting a higher clock rate is desirable for the application of DVS techniques because a larger range transfer rates can be used, which will yield better performance characteristics.

The unidirectional point-to-point connectivity of the interconnect has certain desirable characteristics for real-time embedded applications. Most important is the fact that packets are guaranteed to be delivered (i.e. no dropped packets) and they will arrive at the destination in the same order as they were sent. This property reduces the buffering needed as data does not need to be re-ordered by the destination (a CPU and Memory intensive operation). In addition, as will be described in Section 8.2, a simple but effective flow control mechanism guarantees that packets will not be dropped, eliminating the need for acknowledgement packets and associated protocol overhead.

A hierarchical ring structure is also efficient at routing multicast/broadcast packets [91]. Sending broadcast packets in a mesh topology is difficult because there exists many paths to any single destination. Care must be taken so that packets are not needlessly duplicated. In contrast, hierarchical rings can easily handle broadcast packets without wasting resources due to the way in which packets are forwarded, making them efficient for implementing cache-coherency protocols.

In [91], the authors compare mesh to hierarchical rings and showed that for shared memory applications they outperformed by 20–40% for certain applications. They also demonstrated that increasing the bandwidth of the global-ring (i.e. increasing the clock rate) has a significant impact on performance. In short, the hierarchical rings provide a flexible interconnection strategy that can be tuned depending on the application requirements to meet energy, area, and performance constraints.

8.2 Architecture

The system architecture is shown in Figure 8.1, where each processing element connected to a ring-interface consists of a Leon3 processors, a AMBA bus and ar-

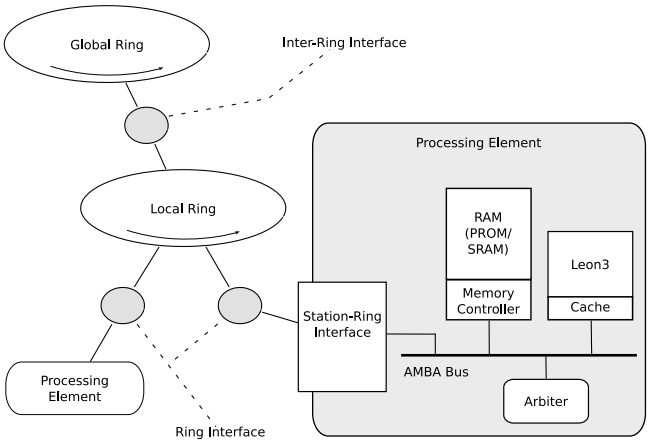


Figure 8.1: System architecture of the RTL implementation. Each processing element consists of a Leon3 processor, AMBA bus and arbiter, memory, and a station-ring interface.

biter, memory and controller, plus a station-ring interface component that enables the AMBA bus to interface with the rings.

8.2.1 Interconnect

The architecture is based on the design presented in [91], which was used in a shared memory multiprocessor system. The implementation of the two-level hierarchy, as shown in Figure 3.1, consists of 4 local rings connected to a single global ring.

In order to keep buffering and latencies to a minimum, each packet is actually a *flit/phit* and can be forwarded in 1 clock cycle. As shown in Figure 8.1, a local ring is connected to the central ring by an inter-ring interface (IRI) and each station is connected to a local ring by a station-ring interface (SRI).

Flits are routed through the interconnect via the (I)RIs shown in Figure 3.2, which are connected to form the 2D hierarchical ring structure. Each (I)RI contains *bi-synchronous*[2] FIFOs that buffer incoming and outgoing packets. The bi-synchronous FIFOs enables the partitioning of the rings into separate clock do-

[2]A bi-synchronous FIFO is commonly referred to as *asynchronous*, which is technically incorrect, but semantically correct because the FIFO in question has two clock inputs. A strictly asynchronous implementation has no clock.

mains. The links connecting the interfaces together to form the rings are bit-parallel signals that represent one packet of data or *flit*.

The fact that the clock rate of the different rings can be independent allows for increased flexibility when tuning the interconnect for specific applications. For example, the clock rate of the global ring can be higher than that of the local rings in order to reduce the latency of global traffic [91]. Furthermore, multiple clock domains provide the facility for the eventual introduction of dynamic clock throttling [16] which can allow rings to be slowed down or sped up as needed to accommodate changing bandwidth requirements while reducing energy consumption.

8.2.1.1 Routing and Flow Control

As discussed in Section 3.3.3, the interconnect implements *lossless* communication that uses a relatively simple backpressure mechanism for handling network congestion to prevent packets from being dropped. At each hierarchical level, a backpressure signal can be propagated to prevent injection of new packets. When enough outstanding packets have been removed from the interconnect, freeing resources, the backpressure signal is de-asserted. The implementation is *deadlock* free because outstanding traffic is *always* allowed to drain, therefore guaranteeing that the network never stops due to congestion.

The NoC interconnect was designed to be realistically used on a FPGA to link together CPU and DSP cores. The interconnect had to be area efficient yet support both point-to-point addressing, multicast and broadcast. This was solved by virtue of the hierarchical configuration of the rings coupled with a one-hot encoding of the addresses as described in Section 3.3.2. The use of one-hot encoding simplifies the steps needed to route packets throughout the interconnect. One can infer that the interconnect can process packets quickly and that the switching logic will be small; this is supported by synthesis results shown in Section 8.3.

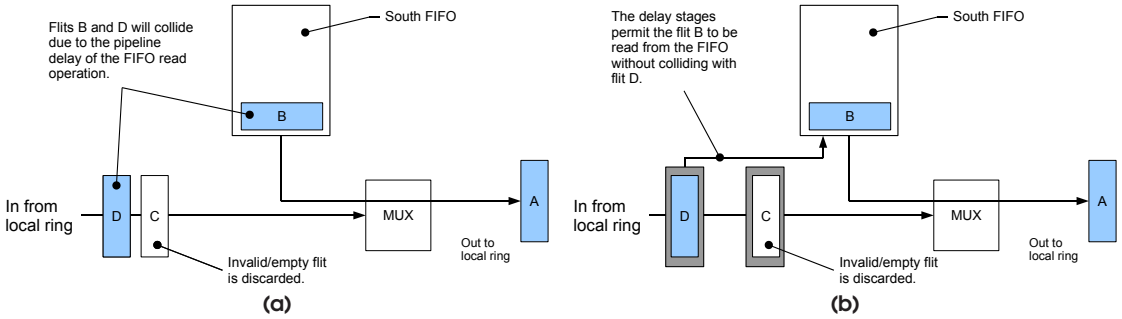


Figure 8.2: (a) Due to the pipelined read delay of the FIFO, flit B will collide with flit D. (b) The addition of 2 input registers delays flit D by one clock cycle, thus enabling flit B to be routed to the output without a collision occurring.

8.2.1.2 Improved Inter-Ring Interface Implementation

The simplicity of the hierarchical rings leads to area-efficient hardware implementation and simple routing logic. However, the very same simplicity can lead to sub-optimal network utilization under certain circumstances. Specifically, when there are several flits in the north or south FIFOs of the inter-ring interface, the current architecture does not allow them to be read in consecutive clock cycles. Instead, the architecture is limited to reading 1 flit every 2 cycles from the FIFOs. This section will present a modification of the IRI such that the network utilization is improved and average latencies are reduced.

When performing *pipelined* read operations[3] from the north and south FIFOs, the last flit read may collide with an incoming flit. The situation is illustrated in Figure 8.2a:

1. During the first clock cycle, flit *A* is read from the FIFO and placed at the output port, while the empty flit *C* is discarded (the "read enable" signal is kept high).

2. When flit *D* is detected at the output port, the read operation is terminated (the "read enable" signal is unset) and flit D will be forwarded to the output port.

3. When the "read enable" signal is de-asserted, the pipeline delay will require

[3] A *pipelined* read from a FIFO is when the "read" signal is kept high for consecutive clock cycles, enabling 1 flit to be read per clock cycle.

an extra clock cycle to complete, thereby resulting in an extra flit, flit *B*, being read. Since IRI will forward flit *D* during this cycle, the extra flit read from the FIFO due to the pipeline delay will be dropped.

Therefore, the architecture shown in Figure 3.2b cannot perform pipelined read operations due to the possibility of flits being dropped. Consequently, flits stored in the FIFO may be delayed unnecessarily when there is no data circulating on the local ring[4].

In order to support pipelined read operations, the architecture of the inter-ring interface shown in Figure 3.2b can be improved by adding two extra registers, or *delay stages*, at the inputs in order to provide a 2 cycle delay. The delay afforded by the added registers provides the IRI with the extra time needed to turn off the pipelined read operation, and to handle the extra flit that is read from the FIFO due to the pipeline delay. The modified architecture is shown in Figure 8.2b, where flits *C* and *D* must pass through two registers before being forwarded to the output port. When the IRI detects a valid flit at the input, flit *D* in this case, it will disable the "read enable" signal of the FIFO. The pipeline delay will result in flit *B* being read from the FIFO and forwarded to the output port. Instead of a collision, flit *D* will be stored in the second register (flit *C* will be discarded), resulting in no lost flits.

Simulation results comparing the improved and normal architectures will be presented in Section 8.3.2.1. The results obtained show a reduction in average latency due to the better network utilization afforded by the pipelined read capability.

8.2.2 Processing Element

As shown in Figure 8.1, a processing element consists of a Leon3 processor, a memory controller and a *station-ring interface* (SRI), which are all connected to an AMBA bus. The SRI acts as an AMBA slave which can be accessed from the Leon3 through memory mapped IO.

In order to enable more efficient use of the interconnect, the SRI was modified to be an AMBA master as well as a slave so that it could initiate direct memory

[4]The case where there is *no* data circulating on the local ring is the worst-cast.

access (DMA) and transfer incoming flits from the interconnect to memory without needing processor intervention. A program running on the processor can write data to memory and then configure the SRI by writing to configuration registers. When the SRI has been triggered to start sending data in DMA mode, it will request the AMBA bus and wait until the arbiter grants the bus. Once the bus has been granted, the SRI will start reading data from memory in *incremental Hburst* [41] mode, which will enable the SRI to send 1 word/cycle over the interconnect. By efficiently removing incoming data from the interconnect, the DMA mechanism results in better interconnect utilization due to the flow control mechanisms being triggered less often.

8.2.3 Applications

Multimedia applications often involve a series of processing steps where data is passed from one station to another in a pipelined fashion. As can be seen in Figure 3.1, PEs on the local ring are connected in a unidirectional manner. The implementation of sequential data processing is efficient because multiple simultaneous transfers can occur between pairs of stations. Furthermore, it is often the case in parallel applications that a node is designated as *head-node* and controls the operation of the other nodes in the network by broadcasting/multicasting messages and/or data to them. As discussed, the hierarchical ring architecture is efficient at sending multicast packets and will thus be efficient for applications exhibiting these kinds of communication characteristics. Furthermore, applications that generate less global traffic exhibit better performance characteristics than mesh networks [91] . In mesh-architectures, complex routing and flow-control mechanisms are needed in order to ensure these properties. This overhead can negatively affect the performance of the interconnect. For real-time applications, and especially multimedia applications, low latencies, guaranteed delivery and in-order arrival are critical for correct operation that meets quality-of-service constraints. The hierarchical ring interconnect implementation exhibits these characteristics while still maintaining low-latencies and high speeds.

8.3 Results

The hierarchical ring interconnect was initially chosen for a multi-processor SoC implementation on FPGA targeting multimedia applications. It is difficult to properly analyze a high level transactional model because they are not constrained by real world considerations such as energy consumption or implementation area. Although high level models are useful in early stages of development and can be used to guide design decisions as was done in [16], the only way to accurately analyze the performance characteristics is by studying an actual RTL level description. With that in mind, the following subsections present some RTL level simulation results as well as the synthesis and place-and-route results generated by EDA software.

8.3.1 Experimental Setup

Traffic generation was accomplished through the implementation of stochastic processes that inject packets into the interconnect based on several random variables:

- local traffic probability (LTP),
- burst length,
- read and write delay.

The manner in which the aforementioned variables affect the traffic patterns is described in the following subsections.

8.3.1.1 Local Traffic Probability

The architectural characteristics of the hierarchical ring interconnect should be exploited by the application. Specifically, the application should be designed so that it takes advantage of the high node-to-node bandwidth characteristics of the local rings. Furthermore, since the system bottleneck is the global ring, the amount of traffic sent over it should be minimized.

The *local traffic probability* (LTP) of the sender stations is used to control the ratio of global to local traffic being sent by each station. For example, when the local traffic probability is 0.6, there is a 60% chance that the sender will send traffic to a station on its local ring[5]. The LTP can therefore be used to study the performance of the interconnect for several LTP values, simulating ideal and non-ideal traffic loads.

8.3.1.2 Burst Length

A *burst* operation is when a station sends multiple flits in consecutive clock cycles, without any pauses/delays between individual flits. When the destination address for the current send operation has been selected as described in Section 8.3.1.1, the number of packets (in this case *flits*) to be sent in 1 burst operation is determined by a random variable that can range from 1–16[6]. Since the RTL implementation of the processing element support DMA transfer (specifically the SRI), it was deemed necessary to model burst transfers.

8.3.1.3 Read and Write Delay

In general, an application will access the interconnect and then do something else before the next access. Two random variables are used to model this behavior, namely the *read* and *write* delays. The *read* delay is the amount of time the receiver station will wait before emptying data from the incoming FIFO. This delay models the fact that SRI may have to wait before being able to offload the flits stored in the incoming FIFO to memory. Similarly, the *write* delay is the amount of time a sender will wait before sending again. The write delay attempts to model the time between successive send operations initiated by the PE.

8.3.2 Simulation Results

Figure 8.3 shows the FIFO usage of a high load simulation. The simulation case is one example where all stations send words to all other stations. The simulation

[5]Note that the destination address is also a random variable.

[6]The FIFO sizes of the RI components have been set to 16, so sending more will simply result in a backpressure signal from the RI when its outgoing FIFO is full.

Table 8.1: Effect of local traffic probability (LTP) on latency.

LTP	Global Traffic (cycles)			Local Traffic (cycles)		
	Min	Max	Avg	Min	Max	Avg
0.4	27	413	118	16	261	81
0.5	25	366	115	16	346	78
0.6	26	342	103	16	359	74
0.7	25	404	109	16	347	73
0.8	28	265	90	16	258	75

setup is comprised of four local rings connected by a global ring. The IRI FIFOs (two traces at the bottom of each Global Ring block) are saturating due to the high amount of inter-ring (global) traffic generated. The simulation comprised of approximately 5000 packets (each station sending and receiving approximately 300 packets).

Depending on the value used for the LTP variable, the traffic going through the global ring causes the buffers in the IRIs to remain relatively full during most of the simulation. Table 8.1 shows the effect of the LTP variable on latencies. It can be seen that the latencies experienced by local traffic is not as affected as the global traffic latencies. This phenomenon can be attributed to the fact that the global ring becomes a bottleneck when the amount of global traffic is high (ie. the LTP is lower). For applications which have higher probabilities of local traffic, a LTP > 0.6 shows good performance results for the high-load testbench. One would expect the average latency to be lower when there is less traffic on the interconnect.

What is interesting is that the global ring flow-control signals are not triggered very often. It can therefore be concluded that using buffer sizes of 32 for the IRIs, the global ring is able to keep up with the demands made on it by the local rings. The right edge of Figure 8.3 shows that the buffer occupancies decrease while the backpressure signals are asserted. This shows that the flow control mechanisms are working, where outstanding flits are drained from the interconnect without packet loss. Furthermore, the behavior of the system respects the criteria of guaranteed and in-order delivery.

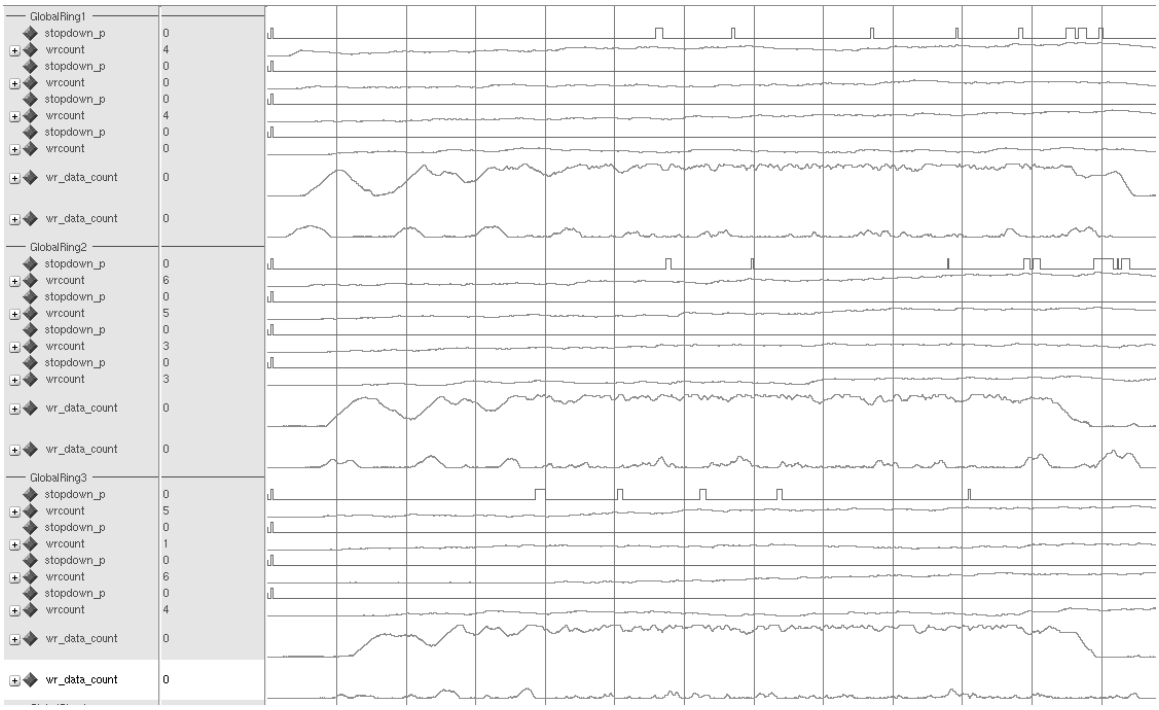


Figure 8.3: ModelSim trace of the FIFO count in a highly loaded interconnect simulation.

Interestingly, the authors of [132] report that the overhead imposed by network routing severely limits the throughput of the system. In fact, they report that when many stations send data simultaneously, the throughput of the system can drop by as much as 80%, supporting the previous claim that routing and protocol overhead can severely hamper interconnect performance. As can be seen in Figure 8.3, the RTL implementation of the hierarchical rings implementation does not suffer as much under high loads.

8.3.2.1 Improved Inter-Ring Interface

The improved implementation of the inter-ring interface described previously in Section 8.2.1.2 was implemented in VHDL, and simulations that compare the performance of the unmodified versus the improved architecture were performed. The LTP used for the experiment was 0.6, and approximately 10^5 total flits were injected into the interconnect for the simulation run. Figure 8.4 shows the latency reduction achieved when the pipelined read is used; the plot shows the average

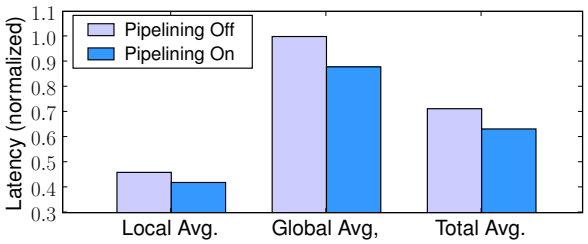


Figure 8.4: Simulation results comparing the performance of the normal and pipelined architectures of the IRI. Results show that the average latency for all traffic has be reduced by approximately 11% due to the addition of the delay stages at the input of the IRI.

latencies for local and global traffic, as well as the average of all traffic. The latency reductions for local and global traffics were 8.7% and 12.1% respectively, yielding a average latency reduction for all traffic of 11.3%. Furthermore, 2.3% more flits were injected due to better network utilization when pipelined reads were enabled. Therefore, the addition of 2 delay stages to the input of the IRI has yielded a performance improvement of approximately 10%, which is a good improvement considering the relatively small cost of the extra hardware. Furthermore, the overhead associated with the delay stages can be mitigated by reducing the size of the FIFOs slightly to compensate, which would result in little-to-no overhead associated with the improved architecture.

8.3.3 Synthesis and Timing Analysis Results

To substantiate the claims of high performance in FPGAs, the performance of each module in the NoC interconnect was analyzed after performing place and route in a *xc2v4000-bf957-5* Xilinx FPGA using the ISE 8.1i FPGA software. The routing delay in large FPGA has a significant impact on the final timing. This aspect is often overlooked in the design of NoC interconnects. It is also impossible to account for these effects in high-level simulation models.

The design and implementation of the rings is relatively insensitive to the routing due to two factors, namely the locality of routing signals and the pipelined structure adopted in the design of the modules. Since no global connection strategies (i.e. links that span multiple hops) are used, as sometimes proposed

Table 8.2: Synthesis results targeting Xilinx Virtex2 FPGA.

Component	Area (LUT) Virtex 2	Clock Rate (MHz) Virtex 2
Ring Interface	708	250 Ring / 200 CPU
Inter-Ring Interface	336 + 4 Block RAM	214
Station-Ring Interface	171	169
Complete Interconnect	14153	200

in torus and mesh based networks [94], the resulting implementation is efficient in terms of routing, area, and performance.

The results obtained from synthesis shown in 8.2 compare quite favorably to other interconnect implementations targeting the same technology. For example, the switch used to implement a mesh network in [133] requires 631 lookup tables (LUTs) and can run at 25Mhz with a bus width of 10 bits (8 data and 2 control). The switch implementation in [132] uses approximately 900 LUTs and can run at 40Mhz with a bus width of 19 bits (16 data and 3 control). The results shown in Table 8.2 shows that the RTL implementation of the hierarchical rings requires a comparable number of LUTs for a much larger bus width of 48 bits (32 for data and 16 for addressing since each flit carries both source and address masks). The simple routing logic results in relatively small area requirements, especially for the inter-ring interface which requires 336 LUTs.

The hypothesis that the simple routing logic would result in a fast interconnect also holds true. The synthesis results for a complete 2 level hierarchical interconnect with 4 stations per local ring and 4 local rings connected to a global ring are shown in the last row of Table 8.2. The entire interconnect requires 14153 LUTs and can run at 200Mhz, yielding a peak sustainable transfer rate of 6.4Gbps between adjacent nodes. The transfer rates reported by [133] and [132] are 320Mbps and 500Mbps respectively.

8.3.4 Routing Structure

The interconnect generates a very clean routing structure as shown in Figure 8.5. The figure highlights *Local Ring 1* (upper left corner) and *Local Ring 2* (upper

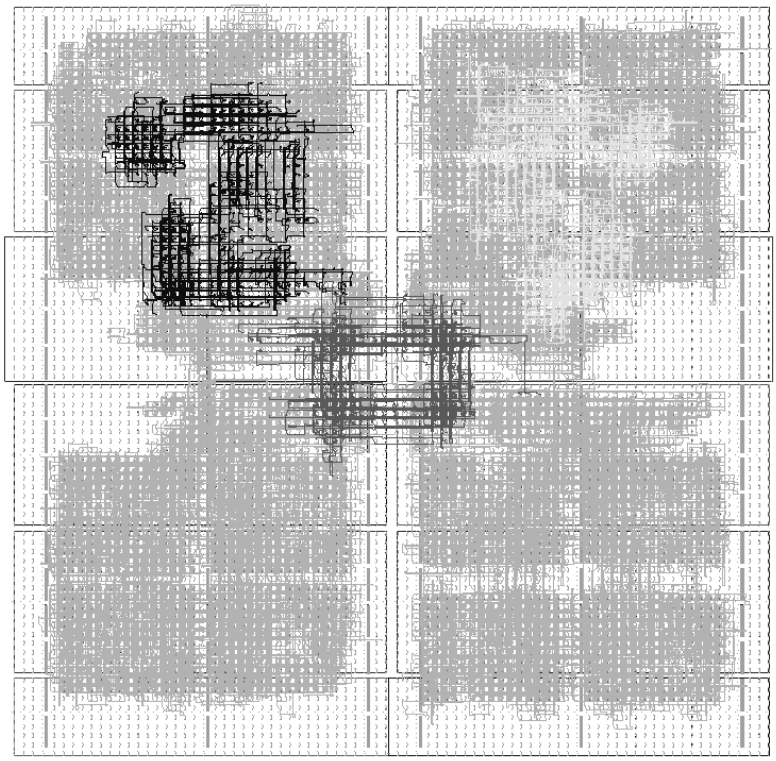


Figure 8.5: Routing Efficiency of the NoC Implementation for 16 stations.

right corner) showing the ring data buses localized in their respective areas. The central ring data bus is highlighted in the center of the figure and also shows the use of short and local routing resources. This contributes to a large extent to the good performance of the NoC interconnect meeting a 200 MHz performance goal on all of the clock domains (16 CPU domains, 4 local rings and 1 global ring).

8.4 Chapter Summary

An investigation into the properties of a hierarchical ring interconnect for a multiprocessor NoC implementation using Leon3 processors has been presented. The motivation behind this work was to demonstrate the suitability of the hierarchical ring interconnect for FPGA implementations. The results obtained during simulation and synthesis clearly demonstrate that the properties of the interconnect compare favorably to other RTL NoC implementations. In fact, synthesis

results show that for a larger bus width of 48, compared to 10 and 19, the area requirements are comparable and the clock speeds are much higher. The results of place-and-route show that the point-to-point connectivity of the architecture yields an area-efficient layout with minimal global routing. Workload simulations also showed that the hierarchical ring implementation can handle high levels of traffic without a significant degradation in performance due to the fact that the routing logic does not require any additional protocol overhead. Furthermore, a low overhead (in terms of hardware) improvement of the inter-ring interface, which enables pipelined read operations, resulted in a better network utilization and an 11% reduction of the average latency.

Lastly, the RTL implementation makes use of several clock domains which will enable the tuning of the clock speeds of each ring to adapt to the data requirements of individual algorithms. The presence of multiple clock domains also lends itself to energy efficiency by eventually integrating dynamic power optimization techniques.

Chapter 9

Conclusions and Future Work

The role of the system interconnect is gaining in importance as increasing numbers of cores are being integrated onto a single chip. Due to the performance, area, and energy constraints, it is necessary to explore alternative interconnection methodologies for large SoC designs. The most promising alternative is the network-on-chip (NoC) approach, which uses an on-chip network to connect a large number of cores. The NoCs are more scalable than shared-medium (bus) architectures, and also provide the ability to use a layered design approach, similar to the OSI protocol stack.

The topology most studied in NoC research is the two-dimensional mesh, which exhibits good performance for localized traffic. However, it suffers from increased latencies and hop counts for long distance traffic as the network size increases. Furthermore, the mesh network routers are relatively large and complex because of the number of input ports and buffers. For on-chip implementation, where energy and area are important metrics to be minimized, it is necessary to explore alternative architectures to the mesh network. This thesis explored the viability of a hierarchical ring network for on-chip implementation through the use of behavioral simulation models and a low-level RTL implementation. In addition, to overcome the scalability limitations of the mesh topology, several composite architectures that use rings for routing global traffic were proposed and simulated using an original simulation platform.

The hierarchical rings were chosen for study because of their simplicity, which results in routers that support a high clock rate, and are area- and energy-efficient. Furthermore, the planarity of the hierarchical rings results in a low wiring complexity — a desirable characteristic for ASIC and FPGA implementation. The RTL synthesis results presented in Chapter 8 shows that the area and speed of the rings compared favorably to other RTL implementations in the literature. In fact, the RTL implementation of the hierarchical rings require a comparable area for a much larger bus width of 48 bits. In addition, the clock rate of 200 MHz reported after performing timing analysis was much higher than those reported by other NoC RTL implementations (e.g. 25 MHz). When compared to other implementations in the literature, it can be concluded that the hierarchical rings are in fact well suited for on-chip implementation.

An interesting property of the hierarchical rings is that their structure permits each ring to be partitioned into its own clock domain, which enables the application of dynamic frequency scaling. In Chapter 7, an energy model was used to evaluate the performance of the rings. It was shown that a significant energy savings of approximately 30–70% could be achieved by varying the operating frequencies of individual rings based on their buffer occupancies without adversely affecting performance. Thus, in addition to the promising RTL synthesis results, the architecture is amenable to the application of frequency scaling that can reduce the energy consumption of the already resource efficient interconnect.

Chapter 4 presented several composite ring/mesh topologies that use rings for global routing to help mitigate the scalability issues associated with mesh interconnects, namely large hop-counts and latencies for long distance communications. The simulation results presented in Chapter 6 show that the hierarchical rings can be successfully employed to improve the performance of the mesh network for long-distance traffic. The addition of the rings to the mesh interconnect resulted in reduced latencies and hop counts for global traffic.

The mesh topology also suffers from congestion in the center portion of the network. It was shown that the addition of the hierarchical rings as a global interconnect alleviates congestion by routing traffic up through the rings, thereby freeing resources in the mesh. In addition to architectural considerations, the routing strategy employed can also affect the performance of the network. An

adaptive routing strategy was employed in Section 6.5 to alleviate congestion at the bridge components in the augmented architectures. Simulation results showed that the application of adaptive routing can improve performance, and that tuning the aggressiveness with which the strategy is applied impacts the performance improvement achieved by the algorithm.

Task assignment is when an application, usually represented as a task-graph, is physically mapped onto the interconnect. The task assignment has a large impact on performance because it determines how far or near communicating tasks will be relative to each other. For example, a non-optimal mapping will require many long distance communications, which will degrade performance due to large hop counts and congestion. The most often used task-assignment method is to minimize the distances between communicating pairs of nodes. This method usually results in an uneven distribution where traffic concentrates in the center of the mesh causing delays. To achieve a more even traffic distribution, a novel blocking-aware task assignment scheme is presented in Section 5.3. The methodology is suitable for reducing the amount of blocking/contention as well as the average latency experienced by traffic in a wormhole routed mesh. To accomplish this task, metrics that help to gauge the effects of blocking in communication infrastructure on a SoC were developed. The simulation results presented in Section 6.6 showed that the network utilization can be more evenly spread out while achieving similar performance in terms of latencies and hop counts when compared to the minimal-path task assignment. The methodology presented in Section 5.3 can be adapted to the composite architectures by modifying the blockage cost function to take into account contention for the bridge components.

To summarize, the work presented in this thesis shows the hierarchical rings to be a viable alternative to the popular mesh topology for NoC implementation. The simplicity of the unidirectional rings results in good performance and resource efficiency. Lastly, the scalability issues of the mesh network can be attenuated by the addition of the rings for routing global traffic to reduce average latencies and hop counts.

9.1 Future Work

As discussed in Section 2.1, the network-on-chip (NoC) paradigm has been proposed relatively recently, and there are still many open problems that require investigation. One of the most challenging problems will be to write application code that will exploit the bandwidth afforded by the on-chip network. To accomplish this task, the necessary application programming interfaces (APIs) and device drivers will need to be developed.

The RTL implementation of the hierarchical rings in VHDL was the first step in implementing a full FPGA SoC. The next step will consist of integrating 16 processing elements with the hierarchical rings, where each node will run the Linux operating system. To enable software development, custom Linux device drivers will need to be implemented, which will enable applications to interface with the interconnect. Layered on top of the device drivers, an API that provides a high-level interface for sending and receiving messages will be written, thus facilitating the development of distributed application software.

A drawback of the hierarchical ring interconnect is the bisection bandwidth of the global ring, which can saturate under non-ideal traffic loads. To increase the bandwidth, bi-directional rings can be used instead of unidirectional ones (see Appendix B). The use of counter-rotating rings would result in multiple paths between any two nodes on the network, thereby introducing the possibility of using adaptive routing strategies to improve performance. In addition, the hyper ring configuration, which has more than a single global ring, can be used to increase the bisection bandwidth of the interconnect (see Appendices A and B). Another possibility is to explore a combination of the hyper and multi ring topologies to form as yet unexplored topologies that would exhibit increased bisection bandwidth and path-diversity. Lastly, more complex switch architectures can be considered in an effort to boost performance. For example, multi-flit packets similar to those used in wormhole-routed mesh networks can be used; or the effect of adding virtual-circuits to the rings can be evaluated. Each approach would require a certain amount of overhead, but depending on the application requirements, the extra cost may be justified.

The composite architectures presented in Chapter 4 were shown to improve the scalability of mesh networks. The interface between the mesh and hierarchical ring interconnect can affect the performance of the interconnect since the bridge component can become a bottleneck. A thorough exploration of different bridging strategies would be a worthwhile exercise, as the potential performance improvement can be significant. In Section 4.5, so-called "narrow" and "wide" bridge implementation were presented. A third option, which would increase the bandwidth of the bridge component, would be to provide 4 output ports to the hierarchical ring interconnect. This architecture would enable the bridge component to read packets from all 4 input ports of the mesh router simultaneously (i.e. north, south, east, and west), which would reduce the latencies due to contention for the input port of the bridge.

Further, the importance of the task assignment cannot be overstated as it can greatly affect performance and energy usage. The presented blocking-aware task assignment strategy is designed to reduce contention and to distribute traffic more evenly. The current implementation does not afford fine-grained control over the task distribution. For example, a designer may wish to constrain certain low priority tasks to be assigned to edge tiles, or high priority ones to central tiles, etc. This strategy can be extended to consider task priorities, as well as custom-defined zones such that a more fine-grained control of the optimization can achieved, ultimately yielding better performance.

In addition to the open research problems discussed in Section 2.1.5, a significantly more comprehensive effort will be needed that accounts for difficulty in designing and manufacturing correct large scale SoCs. As large SoC systems are becoming increasingly complex, the task of verifying correct system behavior is becoming difficult. The integration of online debugging mechanisms [134] that enable the system operation to be monitored in real-time are of great importance. In that work, the debug interfaces and protocols were devised that facilitate the instrumentation of a SoC by assertion checkers using modern assertion languages such as PSL. The communication of checker results, together with the necessary time-stamping and synchronization information, is achieved through the NoC.

As well as increased complexity, designers will have to cope with manufacturing errors that will be unavoidable in future manufacturing processes [11]. It is

therefore necessary to incorporate the test access and the debugging mechanisms within future designs that will enable errors to be detected and bypassed. Rudimentary debugging mechanisms have already been incorporated into the RTL implementation of the hierarchical rings [134]. The aforementioned FPGA SoC implementation will incorporate more complex debugging mechanisms, which will aid in the development effort by enabling real-time monitoring and data collection to be performed during system operation.

Appendix A

The Hyper-Ring Architecture

This chapter presents a comparative analysis of hierarchical ring and hyper ring interconnects for network-on-chip (NoC). The two-level hierarchical ring architecture has been shown to be resource efficient [17]. However, the hierarchical ring topology has a constant bisection bandwidth that does not scale with network size. To address this issue, an alternate architecture called a two-dimensional hyper ring [123] is considered. The hyper ring topology is presented as an improvement over the hierarchical ring topology, whereby an additional global ring is used to double the bisection bandwidth. A comparison of the simulation and synthesis results of the architectures is presented, which show that the cost (in terms of area and power) associated with the performance increase achieved by the addition of the second ring is acceptable. Finally, the second global ring also improves the path diversity of the topology, enabling the addition of debug capabilities for detecting and diagnosing error conditions.

A.1 Architectures

While the simplicity of the unidirectional rings makes it an attractive architecture, it has the drawback of limited scalability. That is, the diameter of the network grows linearly with network size, hence hop-counts and latencies can become unacceptably large. To attenuate the scalability issues of the single ring, it is

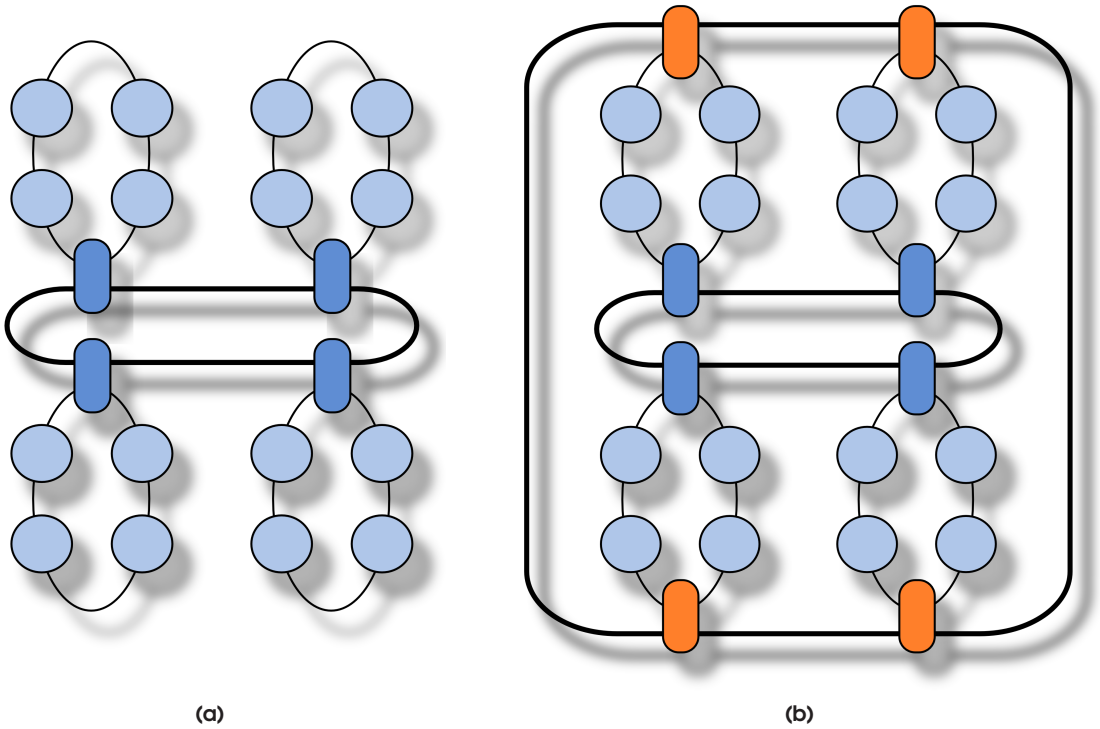


Figure A.1: (a) Two-level hierarchical ring architecture consisting of a single global ring and four local rings. Each local ring has four terminal nodes. (b) A hyper ring architecture consisting of two global rings and four local rings.

therefore appropriate to consider a hierarchical topology of rings, which results in greater scalability and smaller network diameters. In fact, the diameter of a hierarchical ring topology grows logarithmically with the number of nodes.

An example of a hierarchical ring topology is shown in Figure A.1a, which is a two-level architecture consisting of 4 local rings and 1 global ring for routing traffic between local rings. The topology of unidirectional rings connected in a hierarchical manner share the same characteristics of the single unidirectional ring that are of importance for NoC implementations. Importantly, the low degree of the switches results in simple, fast, and area efficient routers. For example, the degree of the terminal nodes shown in Figure A.1a is 3 (the local connection to the PE is not shown), and the degree of the inter-ring routers is 4. The low degree of the routers used in the hierarchical rings results in a planar topology, which is well suited for efficient 2D layout. The topologies discussed in this chapter

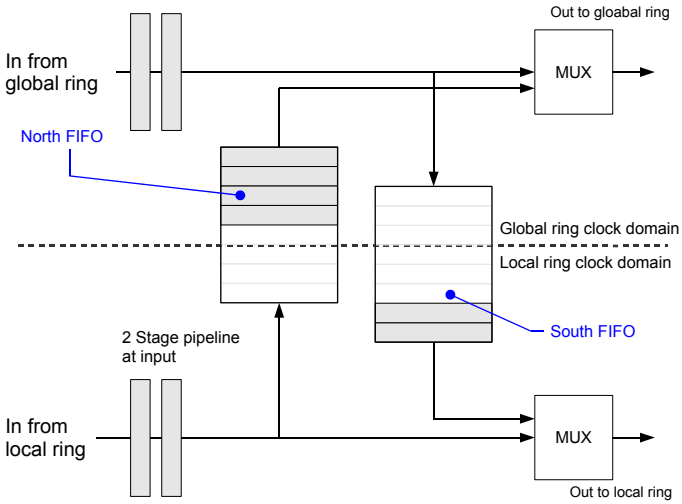


Figure A.2: The pipelined implementation of the inter-ring interface (IRI) routes data between local and global rings. FIFOs that are intersected by the dotted line, that indicates the clock boundary, are required to be bi-synchronous.

have no global routing, consequently, place-and-route will be efficient [17]. The unidirectional nature of the rings reduces the overhead associated with routing and thus results in low latencies and high throughput.

A.1.1 Switch Implementation

Data is routed through the interconnect via the inter-ring and ring interface components previously discussed in Chapter 3. Figure A.2 shows the pipelined implementation of the inter-ring interface (IRI), which differs from the implementation shown in Figure 3.2b. When flits are read from one of the IRI FIFOs in consecutive clock cycles (pipelined read operation), care must be taken to avoid collisions when a new flit arrives at the input because of the 1 cycle delay associated with halting a read operation. To avoid collisions, a two stage input pipeline has been added to the IRI. When a new flit arrives at the input, an active pipeline read operation is halted. On the next clock cycle, the received flit will advance to the 2 stage of the input pipeline, while the last flit of the read operation is read from the FIFO and written to the output. On the next clock cycle, the flit is read from the second stage register and written to the output, thus having avoided a collision.

A.1.2 Diagnostic and Monitoring

A further motivation for moving from the hierarchical rings to the 2D hyper ring is that it provides a more suitable platform for debugging and monitoring. The higher complexity brought forward by NoC implementations requires an integrated approach to the debugging and monitoring of the NoC while in operation. A debug methodology was proposed [134] to extend the network such that it could carry maintenance, status and detected faults information on a hierarchical ring implementation.

A limitation of the hierarchical rings is that only one path exists between any two nodes on the network. Therefore, the time-sensitive monitoring and debugging information can be delayed if there is congestion on the network. On the other hand, the hyper rings have higher path diversity due to the second global ring (i.e. more than one path exists between any two nodes on the network). The hyper rings extends the debug architecture by allowing the debug information to follow a different routing path than the regular traffic. The routing tables can be modified such that the debugging and monitoring information temporarily have exclusive use of one of the global rings for their tasks. While one global ring is reserved for debug traffic, the NoC can still be in an operational state, albeit at a reduced global traffic carrying capacity. In the described scenario, the time-sensitive debugging information is provided with a low latency path from sender to receiver, enabling the system to be more easily monitored and error conditions diagnosed.

A.2 Simulation Results

A cycle accurate SystemC model of the hierarchical and hyper rings was used to compare the performance of the two architectures. The reason SystemC was used was because of the much faster simulation times. However, it should be noted that the accuracy of the SystemC model has been verified by comparing it with RTL simulation results.

At each input node (ring interface), an input terminal has a probability P_I of generating a flit. The variable P_I denotes the *injection rate* per clock cycle. For

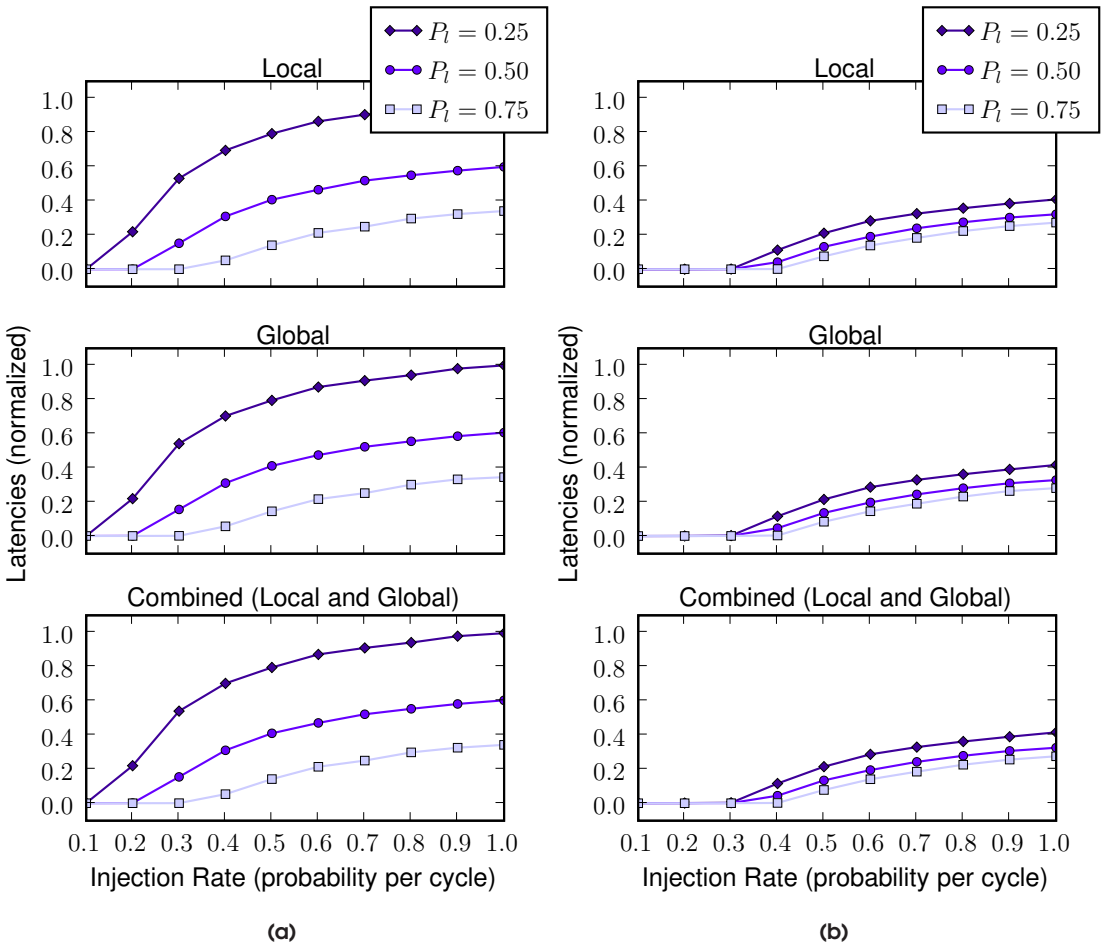


Figure A.3: (a) Simulation results for the hierarchical ring architecture. (b) Simulation results for the hyper ring architecture.

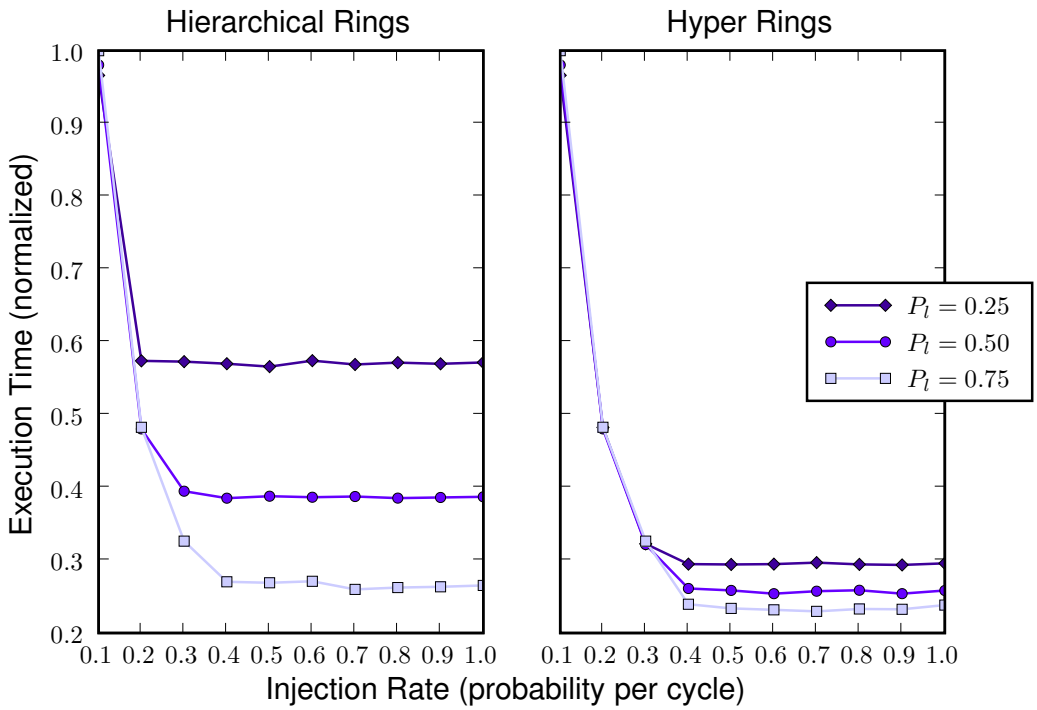


Figure A.4: Execution times for the hierarchical and hyper rings architectures.

example, and injection rate of 0.1 means that each input node has a 10% chance of generating a new flit, which averages out to 1 flit every 10 clock cycles. When an input node generates a new flit, it is stored in an *infinite* input queue [59], and the latency of the flit is calculated as the time from when a flit is created, to when a flit is ejected from the network. The use of the infinite input queue is important because the time a flit spends in the input queue correctly accounts for delays in the network due to congestion.

The system bottleneck of the hierarchical rings is the global ring. If too much traffic is sent over the global ring, the network will saturate due to congestion. The bisection bandwidth of the interconnect is defined as the bandwidth of the minimum number of links that must be cut in order to separate the network into two equal parts. In the case of the hierarchical ring architecture, the number of bisection links is 2. As there are 16 nodes in the architectures, the expected saturation point of the topology is when the injection rate is approximately 10–20% under uniform traffic distribution. The goal of the hyper rings architecture is to improve upon this number; in fact, the added global ring doubles the expected bisection bandwidth.

When two nodes on the same local ring communicate with each other, the traffic being exchanged is said to be *local*. Similarly, if the two nodes belong to different local rings, the traffic must travel through the global ring, and the traffic is said to be *global*. As the two architectures being studied are hierarchical in nature, a variable P_L has been introduced that denotes the probability of a generated flit being local. Using P_L, the ratio of local to global traffic being generated by the simulator can be controlled. As previously discussed, the hierarchical nature of the topology makes it sensitive to the amount of global traffic being sent over the global ring. To maximize performance, communicating tasks should be assigned to nodes on the same local ring if possible, thus maximizing *locality* of communication and improving performance.

To model sub-optimal and optimal application mappings, three values of P_L were used during simulation: 0.25 models a poor mapping where 75% of traffic is global. A P_L value of 0.50 also models a sub-optimal mapping for a hierarchical topology because it implies that half of all traffic will travel through the bisection links. Finally, a P_L value of 0.75 represents a good mapping that exploits the

architecture well. Figure A.3 shows the results obtained for both architectures while varying the injection rate from 0.1 to 1.0, where each node was configured to send 5000 flits. The normalized latencies show that as expected, hierarchical rings saturate at a lower injection rate than the hyper rings. Furthermore, the flits travelling through the hyper rings experience a marked reduction in average latencies, ranging from approximately 60% for $P_L = 0.25$, to approximately 10% for $P_L = 0.75$. The difference in average latency for $P_L = 0.75$ is less than for other values because the generated traffic maps well onto both architectures. We can conclude from Figure A.3 that latency reduction experienced by the hyper rings over the hierarchical rings is greater as the amount of global compared to local traffic increases. However, the hyper rings still outperforms the hierarchical rings in all cases.

Figure A.4 shows the completion times for each simulation run. The completion time is defined as the time it takes for the last flit to reach its destination and be ejected from the network. It can be seen that the completion times for the hyper rings are significantly lower on average.

A.3 Synthesis Results

The synthesis of the hardware modules was performed using Synopsys DC Ultra (Version X-2005.09) to highlight the performance of the hyper ring in a CMOS ASIC. The technology target was TSMC 0.18μm using the Artisan standard-cell library [135] and operating at 1.8V.

The synthesis was performed for typical operating conditions (1.8V operation, room temperature, normal process). The wireload model was selected to be relatively small (average wire length 66.7 μm). This is a reasonable assumption since the PE interface logic and the ring interface logic are very localized. Their logic cone depend only on internally registered values. The module architectures are registered at the output and input, so the full clock cycle is available for signal propagation between nodes. This adds a slight area penalty due to the flip-flops and additional buffering for the flow control, but will ensure that the layout is unaffected by the interconnect delay.

Table A.1: Area estimates details for the NoC building blocks

Operating Frequency	Comb. Area (k μm^2)	Non Comb. Area (k μm^2)
Interface to the processing element		
625 MHz	95.4	176.6
500 MHz	85.8	176.3
250 MHz	79.5	175.4
100 MHz	78.4	175.2
Interface between rings		
625 MHz	74.1	157.5
500 MHz	66.7	157.1
250 MHz	61.6	156.4
100 MHz	61.1	156.2

Table A.2: Power estimates details for the NoC building blocks

Operating Frequency	Cell Power (mW)	Switching Power (mW)	Cell Leakage (nW)
Interface to the processing element			
625 MHz	120	26.7	1308
500 MHz	93.2	22.8	1030
250 MHz	46.1	10.8	919
100 MHz	18.6	4.3	912
Interface between rings			
625 MHz	105	23.4	1070
500 MHz	81.8	19.1	935
250 MHz	40.1	9.2	801
100 MHz	16.1	3.5	817

Table A.3: Area and power estimates for the two architectures synthesized for different frequency of operation

Operating Frequency	Total Cell Area (mm^2)	Total Power (W)	Cell Leakage mW
hierarchical ring			
625 MHz	5.28	2.85	25.2
500 MHz	5.10	2.26	20.2
250 MHz	4.95	1.11	17.9
100 MHz	4.93	0.45	17.9
hyper ring			
625 MHz	6.20	3.37	29.5
500 MHz	5.98	2.66	23.9
250 MHz	5.82	1.31	21.1
100 MHz	5.80	0.52	21.1

The synthesis was performed for four different target frequencies, namely 625 MHz, 500 MHz, 250 MHz and 100 MHz. Table A.1 shows the maximum frequency of operation at 625 MHz. The performance limitation comes from the internal memory buffers; the critical path is the *FIFO Full* signal in the bi-synchronous FIFO that passes data across rings.

At full speed, the NoC interconnect does consume a fair amount of power (up to 3.37 W, see Table A.3), but dynamic clock scaling is inherently supported by the design so less-loaded rings could run at a reduced clock speed and save a considerable amount of power. The processing elements in the NoC are expected to consume a more significant amount of power.

One element that is interesting to note is that 93.5% and 92.2% of the cell area of the NoC interconnect is occupied by the various buffers in the hierarchical-ring and hyper ring respectively. In this design, FIFOs were implemented using arrays of flip-flops, which are less efficient than using an optimized RAM cell. The size of the FIFO memories are all between 640 and 768 bits which could benefit from the generation of RAM cells to reduce their area overhead.

For a similar target frequency of 500 MHz, the hyper rings use 17.7% more power and occupy 17.2% more area. Those results are quite positive, since the hyper ring offers new beneficial architectural properties such as path redundancy

and higher bisection bandwidth that the hierarchical-ring topology was lacking.

The localized clock domains allow the two rings to be clocked at different frequencies of operation. A chip floorplan mirroring a topology like the one shown in Figure 3.1 would benefit from a higher frequency of operation in the center ring due to the smaller parasitic capacitance of its interconnect. The effect of increasing the speed of the global ring would be to increase the bisection bandwidth, thereby increasing the saturation point. It is therefore possible to tune the architecture bandwidth depending on performance requirements. The outer ring can use a lower frequency of operation and carry some of the excess traffic that would create congestion in the inner ring or it can be used as a lower-bandwidth debug or synchronization channel. Latency-sensitive traffic will benefit from the shorter latencies offered by the faster ring.

A.4 Chapter Summary

A comparative analysis of the hierarchical and hyper rings topology has been presented. The advantages of the hyper rings over the hierarchical rings include increased bisection bandwidth and path diversity, and improved debugging and monitoring capabilities. Simulation results have shown that the hyper rings perform better for all types of traffic distributions, as shown by the higher saturation points, particularly for more uniform traffic distributions. Moreover, the throughput of the architecture is increased, as shown by the faster completion times of the simulation runs.

While the performance improvements of the hyper rings is interesting, it comes at the cost of the higher energy and area requirements. The performance/-cost trade-off has been evaluated by comparing synthesis results for both architectures. The hardware synthesis has shown the hyper ring to be an advantageous proposition as the extra hardware cost is within acceptable parameters (approximately 17%). The low additional area and power cost would be offset in many applications by the increased performance, flexibility, reliability and overall quality of the resulting topology.

Appendix B

Fat Hierarchical and Hyper Ring Architectures

This chapter presents two "fat" wormhole routed multistage ring architectures that use bidirectional global rings to increase the bisection bandwidth of the original architectures, which are composed of unidirectional rings. First, a hierarchical ring topology which has a single global ring is considered. Second, a hyper ring topology is presented as an improvement over the hierarchical ring topology, whereby an additional global ring is used to double the bisection bandwidth. Furthermore, the bisection bandwidth of both architectures can be doubled by using bidirectional instead of unidirectional global rings, an approach similar the one used by the "fat tree" [63] architecture. Behavioral level SystemC models are used to compare the performance of the hierarchical and hyper ring architectures, as well as the relative improvement gained by making the architectures *fat*. The architectures use wormhole routing, and virtual channels are used to increase link utilization and to avoid deadlock. Results will show that the added global bandwidth improves performance and increases the scalability of the architectures under study.

B.1 Architectures

The hierarchical and hyper ring architectures used here are the same as shown in Figure A.1. However, their implementations differ from the ones used in Sec-

tion A.1 in that they use wormhole routing and also support virtual channels. While providing more flexibility, the wormhole implementation is more complex and will require more area than the slotted implementation described in Chapter 3. However, if greater performance is needed, the wormhole routed implementation is a suitable alternative.

B.1.1 Router Architecture

Wormhole routing is often used for NoC implementations because it requires less buffers than store-and-forward approaches, and it exhibits shorter latencies than virtual circuit switching. Also, the use of virtual channels [136] can improve link utilization, and be used to avoid deadlock in ring networks [60].

The router architecture used to construct the hierarchical and hyper ring topologies is the generic input queued virtual channel router [59, 136]. Our implementation does not have output queues, and the output ports are registered. Each input port has an input unit consisting of FIFOs for each virtual channel, and state information used for routing. Packets are assigned output ports by a shared router component, and a virtual channel by a shared virtual channel allocator. As full crossbars do not scale well, a multiplexed crossbar switch [136] is used to connect input queues to the output ports. Since the routing function used in the architecture is simple, and the virtual channel allocation policies are fixed, the route computation and virtual channel allocation is assumed to require a penalty of 1 clock cycle. Hence, the switch traversal time for a header flit encountering no contention is 3 cycles (body flits take 2 cycles).

B.1.1.1 Virtual Channel Allocation

In wormhole routed ring networks, care must be taken to avoid deadlock due to the possibility of cyclic channel dependencies. A minimum of two virtual channels is needed to avoid deadlock in a ring network [60], however, in the hyper ring network, 3 virtual channels are needed because of the added possibility of cycles forming due to the 2 global rings. The virtual channel allocation scheme used is as follows: If the current node is denoted by n_i, and the destination node by n_j, if $i < j$ virtual channel 0 is used, and if $i > j$, virtual channel 1 or 2 is

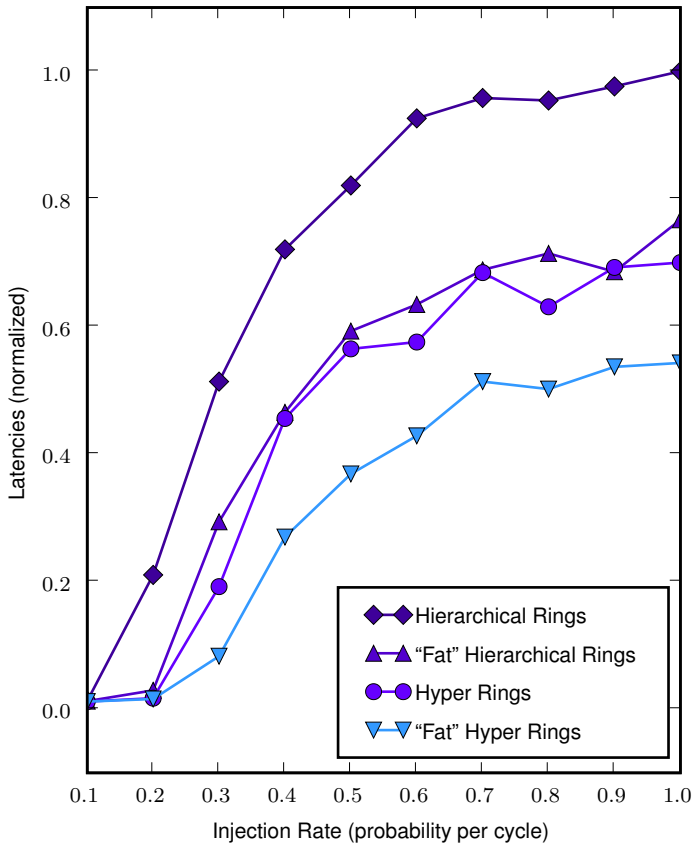


Figure B.1: SystemC simulation results for the hierarchical and hyper ring architectures using uniform random traffic ($P_L = 0.5$).

used depending on whether or not the destination node resides on the same ring. Specifically, if the packet is destined to the same ring, virtual channel 1 is used, otherwise channel 2 is used to break any cycles that can form through the global rings.

B.2 Simulation Results

Behavioral level SystemC models were used to compare the performance of the different architectures. At each input node (ring interface), an input terminal has

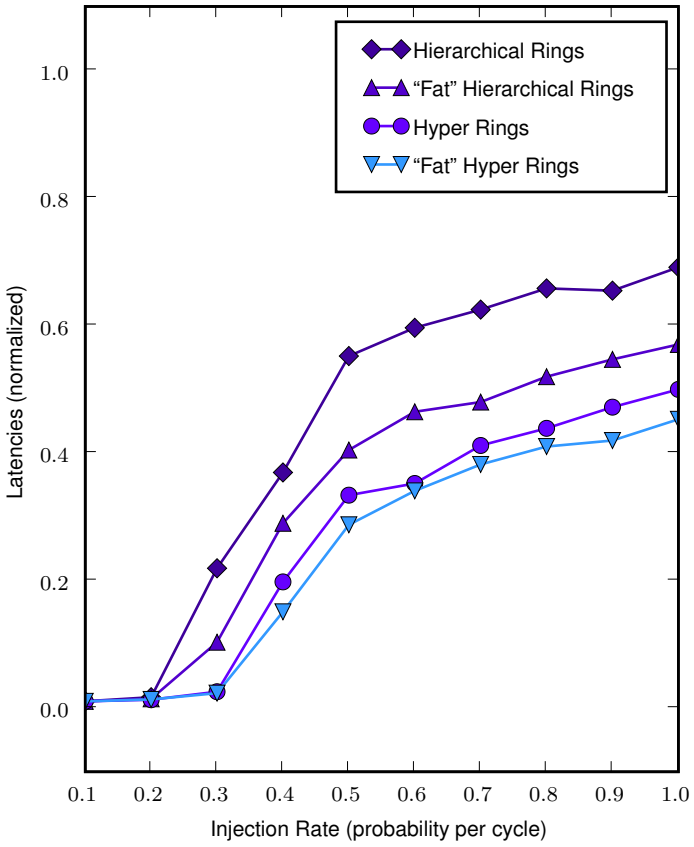


Figure B.2: SystemC simulation results for the hierarchical and hyper ring architectures using skewed random traffic ($P_L = 0.7$).

a probability P_I of generating a flit. As in Section A.2 the variable P_I denotes the *injection rate* per clock cycle. When an input node generates a new flit, it is stored in an *infinite* input queue [59], and the latency of the flit is calculated as the time from when a flit is created, to when a flit is ejected from the network. The use of the infinite input queue is important because the time a flit spends in the input queue correctly accounts for delays in the network due to congestion.

Reiterating previous discussions about the hierarchical rings, the global ring is the system bottleneck. If too much traffic is sent over the global ring, the network will saturate due to congestion. In the case of the hierarchical ring architecture, the number of bisection links is 2. As there are 16 nodes in the architectures, the expected saturation point of the topology is when the injection rate is approximately 10–20% under uniform traffic distribution. The goal of the hyper rings architecture is to improve upon this number; in fact, the added global ring doubles the expected bisection bandwidth. In addition, making the architectures fat further increases the bisection bandwidth and the expected saturation point.

As in Section A.2, the variables P_L is used to denote the probability of sending local traffic. To model optimal application mappings, two values of P_L were used during simulation: 0.5 models a poor mapping for a hierarchical topology because half of all traffic will travel through the bisection links. Second, a P_L value of 0.7 represents a good mapping that exploits the architecture well. Figure A.3 shows the results obtained for both architectures while varying the injection rate from 0.1 to 1.0, where each node was configured to send 5000 flits. The normalized latencies show that as expected, the hierarchical rings saturate at a lower injection rate than the hyper rings. Furthermore, the flits travelling through the hyper rings experience a marked reduction in average latencies. Lastly, the fat architectures show that the average latencies are reduced for both architectures. Interestingly, the performance improvement of the fat architectures is greatest for $P_L = 0.5$ (Figure B.1), showing that the added global bandwidth afforded by the bidirectional rings considerably improves the performance of the architectures under sub-optimal application mappings. The simulation results confirm that the use of bidirectional rings does indeed improve the performance of the two architectures, making them less sensitive to sub-optimal application mappings.

B.3 Chapter Summary

The hierarchical ring network, having a single global ring, can suffer from congestion at the global ring under non-ideal traffic loads. To address this, the two-dimensional hyper ring, which has an additional global ring, can be considered. The bisection bandwidth of both the hierarchical and hyper ring topologies can be improved by using bidirectional global rings to create so-called "fat" architectures. While the fat topologies require extra resources for the more complex routers of the global ring, the node degree of the routers is still less than that of the typical mesh router. Furthermore, the resource efficient unidirectional rings are still used at the lowest level of the hierarchy for local traffic.

The performance of all 4 topology variations have been compared using behavioral level SystemC models. Simulation results have shown that the hyper rings perform better than the hierarchical rings, and the fat hyper ring architecture performed the best. The use bidirectional global rings serves to improve the performance of both architectures, making them less sensitive to traffic patterns exhibiting a high proportion of global traffic. In conclusion, depending on the application characteristics and resource constraints, one of the presented multistage ring topologies can be considered for NoC implementation due to the relatively low resource requirements (i.e. simplicity of the routers) when compared to the mesh architecture.

Bibliography

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998.

[2] "The international technology roadmap for semiconductors 2005 edition," International Technology Roadmap for Semiconductors, Tech. Rep., 2005.

[3] E. Y. Chou and B. Sheu, "System-on-a-chip design for modern communications," *IEEE Circuits and Devices Magazine*, vol. 17, no. 6, pp. 12–17, Nov. 2001.

[4] M. Oka and M. Suzuoki, "Designing and programming the Emotion engine," *IEEE Micro*, vol. 19, no. 6, pp. 20–28, Nov./Dec. 1999.

[5] H. P. Hofstee, "Power efficient processor architecture and the Cell processor," *2005. HPCA-11. 11th International Symposium on High-Performance Computer Architecture*, pp. 258–262, Feb. 12–16, 2005.

[6] D. C. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P. M. Harvey, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa, "Overview of the architecture, circuit design, and physical implementation of a first-generation Cell processor," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 179–196, Jan. 2006.

[7] L. Benini and G. D. Micheli, "Powering networks on chips," in *Proceedings of The 14th International Symposium on System Synthesis*, 2001, pp. 33–38.

[8] H. Wang, X. Zhu, L. Peh, and S. Malik, "Orion: A power-performance simulator for interconnection networks," in *Proceedings of the* 35^{th} *Annual IEEE/ACM Symposium on Microarchitecture*, Istanbul, Nov. 2002, pp. 294–305.

[9] V. Raghunathan, M. B. Srivastava, and R. K. Gupta, "A survey of techniques for energy efficient on-chip communication," in *Proceedings of the Design Automation Conference*, Jun. 2–6, 2003, pp. 900–905.

[10] L. Benini and G. D. Micheli, "Networks on chips: a new Soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[11] G. D. Micheli and L. Benini, "Network architecture: Principles and examples," in *Networks on Chips: Technology and Tools*, ser. The Morgan Kaufmann Series in Systems on Silicon, G. D. Micheli and L. Benini, Eds. Morgan Kaufmann, Jul. 2006, ch. 5, pp. 1–22.

[12] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proc. of DATE*. ACM Press, 2000, pp. 250–256.

[13] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proceedings of the Design Automation Conference*, 2001.

[14] H. Zimmermann, "OSI Reference Model–the ISO Model of Architecture for Open Systems Interconnection," *Communications, IEEE Transactions on [legacy, pre - 1988]*, vol. 28, no. 4, pp. 425–432, Apr. 1980.

[15] Wikipedia, "Network On Chip — Wikipedia, The Free Encyclopedia," 2007, [Online; accessed 10-December-2007]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Network_On_Chip&oldid=176233618

[16] S. Bourduas, B. Kuo, Z. Zilic, and N. Manjikian, "Modeling and evaluation of an energy-efficient hierarchical ring interconnect for System-on-Chip multiprocessors," in *NEWCAS*, 2006.

[17] S. Bourduas, J.-S. Chenard, and Z. Zilic, "A RTL-level analysis of a hierarchical ring interconnect for Network-on-Chip multi-processors," in *Proc. International SoC Design Conference (ISOCC)*, October 2006.

[18] S. Bourduas and Z. Zilic, "A hybrid ring/mesh interconnect for network-on-chip using hierarchical rings for global routing," in *Proceedings of the 1st International Symposium on Networks-on-Chip*, May 2007.

[19] ——, "Latency reduction of global traffic in wormwhole-routed meshes using hierarchical rings for global routing," in *Proc. 18th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Jul. 2007.

[20] S. Bourduas, H. Chan, and Z. Zilic, "Blocking-aware task assignment for wormhole routed network-on-chip," in *Proceedings of 50th IEEE Int'l Midwest Symposium on Circuits and Systems/5th IEEE Int'l Northeast Workshop on Circuits and Systems (MWSCAS/NEWCAS 2007)*, Aug. 2007.

[21] L. Benini and G. D. Micheli, "Networks on chip: a new paradigm for systems on chip design," in *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, Paris, France, 2002, pp. 418–419.

[22] R. Ho, K. Mai, and M. Horowitz, "The future of wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.

[23] T. N. Theis, "The future of interconnection technology," *IBM Journal of Research and Development*, vol. 44, no. 3, pp. 379–390, May 2000.

[24] A. Ivanov and G. D. Micheli, "Guest Editors' Introduction: The Network-on-chip Paradigm in Practice and Research," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 399–403, Sep./Oct. 2005.

[25] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems (performance, reliability, digital)," Ph.D. dissertation, Stanford University, 1985.

[26] Wikipedia, "Integrated circuit — Wikipedia, The Free Encyclopedia," 2007, [Online; accessed 13-September-2007]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Integrated_circuit&oldid=156829142

[27] ——, "System-on-a-chip — Wikipedia, The Free Encyclopedia," 2007, [Online; accessed 13-September-2007]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=System-on-a-chip&oldid=156506450

[28] P. J. Aldworth, "System-on-a-chip bus architecture for embedded applications," in *International Conference on Computer Design (ICCD '99)*, Austin, TX, USA, 1999, pp. 297–298.

[29] B. Cordan, "An efficient bus architecture for system-on-chip design," in *Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999*, San Diego, CA, USA, 1999, pp. 623–626.

[30] K. Lahiri, A. Raghunathan, and S. Dey, "Evaluation of the traffic-performance characteristics of system-on-chip communication architectures," in *Proceedings of the 14th International Conference onVLSI Design*, Bangalore, Jan. 3–7, 2001, pp. 29–35.

[31] ——, "Design space exploration for optimizing on-chip communication architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 6, pp. 952–961, Jun. 2004.

[32] A. Boxer, "Where buses cannot go," *IEEE Spectrum*, vol. 32, no. 2, pp. 41–45, Feb. 1995.

[33] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, p. 23, 2007.

[34] C. A. Zeferino, M. E. Kreutz, L. Carro, and A. A. Susin, "A study on communication issues for systems-on-chip," in *Proceedings of the 15th Symposium on Integrated Circuits and Systems Design*, 2002, pp. 121–126.

[35] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1132953

[36] D. Sylvester and K. Keutzer, "A global wiring paradigm for deep submicron design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 2, pp. 242–252, Feb. 2000.

[37] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," in *ASPLOS-VII: Proceedings of the 7th international conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: ACM Press, 1996, pp. 2–11.

[38] B. A. Nayfeh and K. Olukotun, "A single-chip multiprocessor," *Computer*, vol. 30, no. 9, pp. 79–85, Sep. 1997.

[39] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Proceedings of the 20th ACM International Conference on Supercomputing (ICS)*, Jun. 2006. [Online]. Available: http://cva.stanford.edu/publications/2006/jbalfour_ICS.pdf

[40] I. Walter, I. Cidon, R. Ginosar, and A. Kolodny, "Access regulation to hot-modules in wormhole NoCs," in *Proc. International Symposium on Networks-on-Chip*, May 2007.

[41] D. Flynn, "AMBA: enabling reusable on-chip designs," *IEEE Micro*, vol. 17, no. 4, pp. 20–27, Jul./Aug. 1997.

[42] Wikipedia, "OSI model — Wikipedia, The Free Encyclopedia," 2007, [Online; accessed 20-September-2007]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=OSI_model&oldid=158980814

[43] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli., "Addressing The System-on-a-Chip Interconnect Woes through Communication-based Design," in *Proceedings of Design Automation Conference*, Jun. 2001, pp. 667–672.

[44] U. Y. Ogras, J. Hu, and R. Marculescu, "Key research problems in NoC design: A holistic perspective," in *Proc. of the Intl. Conf. on Hardware/Software Codesign and System Synthesis*, Sep. 2005. [Online]. Available: http://www.ece.cmu.edu/~sld/pubs/papers/f175-ogras.pdf

[45] S. Murali and G. D. Micheli, "SUNMAP: a tool for automatic topology selection and generation for Nocs," in *Proceedings of the 41st Design Automation Conference*, 2004, pp. 914–919.

[46] M. E. Kreutz, L. Carro, C. A. Zeferino, and A. A. Susin, "Communication architectures for system-on-chip," in *Proceesings of the 14th Symposium on Integrated Circuits and Systems Design*, Pirenopolis, Brazil, 2001, pp. 14–19.

[47] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "Xpipes: a latency insensitive parameterized network-on-chip architecture for multiprocessor SoCs," in *Proceedings of the 21st International Conference on Computer Design*, Oct. 13–15, 2003, pp. 536–539.

[48] D. Bertozzi and L. Benini, "Xpipes: A Network-on-Chip Architecture for Gigascale Systems-on-Chip," *IEEE Circuits and Systems Magazine*, vol. 4, 2004.

[49] A. Jalabert, S. Murali, L. Benini, and G. D. Micheli, "xpipesCompiler: A tool for instantiating application specific Networks on Chip," in *Design, Automation and Test in Europe (DATE)*, Paris, France, Feb. 2004.

[50] C. Grecu, A. Ivanov, R. Pande, A. Jantsch, E. Salminen, U. Ogras, and R. Marculescu, "Towards Open Network-on-chip Benchmarks," in *Proceedings of the 1st International Symposium on Networks-on-Chip(NOCS)*, May 7–9, 2007, pp. 205–205.

[51] E. Salminen, T. Kangas, J. Riihimaki, and T. D. Hamalainen, "Requirements for network-on-chip benchmarking," in *NORCHIP Conference, 2005. 23rd*, Nov. 21–22, 2005, pp. 82–85.

[52] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proceedings of the Sixth International Workshop on Hardware/Software Codesign (CODES/CASHE '98)*, Seattle, WA, USA, Mar. 15–18, 1998, pp. 97–101.

[53] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *J. Syst. Archit.*, vol. 50, no. 2-3, pp. 105–128, 2004.

[54] P. R. Panda, "Systemc: a modeling platform supporting multiple design abstractions," in *Proceedings of the 14th International Symposium on Systems Synthesis (ISSS '01)*. New York, NY, USA: ACM Press, 2001, pp. 75–80.

[55] *1666 IEEE Standard SystemC Language Reference Manual*, 2005. [Online]. Available: http://standards.ieee.org/getieee/1666/index.html

[56] "Boost random number library." [Online]. Available: http://www.boost.org/libs/random/index.html

[57] "Log4cxx: Library of C++ classes for flexible logging." [Online]. Available: http://logging.apache.org/log4cxx

[58] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Oct. 1–3, 2003, pp. 19–24.

[59] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[60] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach (revised printing)*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2003.

[61] D. Bertozzi, "Network architecture: Principles and examples," in *Networks on Chips: Technology and Tools*, ser. The Morgan Kaufmann Series in Systems on Silicon, G. D. Micheli and L. Benini, Eds. Morgan Kaufmann, Jul. 2006, ch. 5, pp. 147–202.

[62] W. Bux, F. Closs, K. Kuemmerle, H. Keller, and H. Mueller, "Architecture and design of a reliable token-ring network," *IEEE Journal on Selected Areas in Communications*, vol. 1, no. 5, pp. 756–765, Nov. 1983.

[63] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, 1985.

[64] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong-Chan, S.-W. Yang, and R. Zak, "The network architecture of the Connection Machine CM-5," *Journal of Parallel and Distributed Computing*, vol. 33, no. 2, pp. 145–158, 1994. [Online]. Available: citeseer.ist.psu.edu/leiserson94network.html

[65] F. Petrini and M. Vanneschi, "k-ary n-trees: high performance networks for massively parallelarchitectures," in *Proceedings of the 11th International Parallel Processing Symposium*, Genva, Switzerland, Apr. 1–5, 1997, pp. 87–93.

[66] S. R. Ohring, M. Ibel, S. K. Das, and M. J. Kumar, "On generalized fat trees," in *Proceedings of the 9th International Parallel Processing Symposium*, Santa Barbara, CA, USA, Apr. 25–28, 1995, pp. 37–44.

[67] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation," *Communications of the ACM*, vol. 24, no. 5, pp. 300–309, 1981.

[68] S. Bhattacharya, Y. H. Choi, and W. T. Tsai, "Unidirectional cube connected cycles," *IEE Proceedings — Computers and Digital Techniques*, vol. 140, pp. 191–195, Jul. 1993.

[69] W. J. Dally, "Express cubes: improving the performance of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 40, no. 9, pp. 1016–1023, Sep. 1991.

[70] B. W. Arden and H. Lee, "Analysis of Chordal Ring Network," *Transactions on Computers*, vol. 30, no. 4, pp. 291–295, Apr. 1981.

[71] W. Aiello, S. N. Bhatt, F. R. K. Chung, A. L. Rosenberg, and R. K. Sitaraman, "Augmented ring networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 598–609, Jun. 2001.

[72] A. Proestaki and M. C. Sinclair, "Design and dimensioning of dual-homing hierarchical multi-ring networks," *IEE Proceedings — Communications*, vol. 147, pp. 96–104, Apr. 2000.

[73] P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Switch-based interconnect architecture for future systems on chip," in *Proceedings of SPIE, VLSI Circuits and Systems*, 2003, pp. 228–237.

[74] C. Grecu, P. P. Pande, A. Ivanov, and R. Saleh, "A scalable communication-centric SoC interconnect architecture," in *Proceedings of the 5th International Symposium on Quality Electronic Design*, 2004, pp. 343–348.

[75] W. J. Dally and C. L. Seitz, "Deadlock-free Message Routing in Multiprocessor Interconnection Networks," *Transactions on Computers*, vol. 36, no. 5, pp. 547–553, May 1987.

[76] P. Mohapatra, "Wormhole routing techniques for directly connected multicomputer systems," *ACM Computing Surveys*, vol. 30, no. 3, pp. 374–410, 1998.

[77] T. Ye, L. Benini, and G. Micheli, "Packetization and routing analysis of on-chip multiprocessor networks," *Journal of Systems Architecture*, vol. 50, no. 2-3, pp. 81–104, 2004.

[78] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *IEEE Transactions on Computers*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[79] Teraflops research chip. [Online]. Available: http://techresearch.intel.com/articles/Tera-Scale/1449.htm

[80] P. Pratim *et al.*, "Design of a switch for network on chip applications." in *ISCAS (5)*, 2003, pp. 217–220.

[81] S. Kumar *et al.*, "A Network on Chip architecture and design methodology," *isvlsi*, vol. 00, p. 0117, 2002.

[82] D. Sigüenca-Tortosa and J. Nurmi, "Proteo: A new approach to network-on-chip," in *Proceedings of IASTED International Conference of Communication Systems and Networks, CSN'02*, 2002.

[83] "Scalable Coherent Interfafce," IEEE Standard 1596-1992, 1992.

[84] H. Samuelsson and S. Kumar, "Ring road network on chip architecture," in *Proc. IEEE Norchip Conference*, 2004.

[85] H. Singh, M.-H. Lee, G. Lu, N. Bagherzadeh, F.-J. Kurdahi, and E.-M. C. Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, 2000.

[86] A. Brinkmann, J.-C. Niemann, I. Hehemann, D. Langen, M. Porrmann, and U. Ruckert, "On-chip interconnects for next generation System-on-Chips," in *ASIC/SOC Conference*, 2002, pp. 211–215.

[87] C. A. Zeferino, F. G. M. E. Santo, and A. A. Susin, "ParIS: a parameterizable interconnect switch for networks-on-chip," in *Proceedings of the 17th Symposium on Integrated Circuits and Systems Design*, Sep. 7–11, 2004, pp. 204–209.

[88] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: Towards the ideal interconnection fabric," in *International Symposium on Computer Architecture (ISCA), 2007*, June 2007. [Online]. Available: http://www.gigascale.org/pubs/1004.html

[89] A. Vahdatpour, A. Tavakoli, and M. H. Falaki, "Hierarchical graph: A new cost effective architecture for network on chip." in *EUC*, ser. Lecture Notes in Computer Science, L. T. Yang, M. Amamiya, Z. Liu, M. Guo, and F. J. Rammig, Eds., vol. 3824. Springer, 2005, pp. 311–320. [Online]. Available: http://dblp.uni-trier.de/db/conf/euc/euc2005.html#VahdatpourTF05

[90] R. Grindley, T. Abdelrahman, S. Brown, S. Caranci, D. DeVries, B. Gamsa, A. Grbic, M. Gusat, R. Ho, O. Krieger, G. Lemieux, K. Loveless, N. Manjikian, P. McHardy, S. Srbljic, M. Stumm, Z. Vranesic, and Z. Zilic, "The NUMAchine multiprocessor," in *Proceedings of the International Conference on Parallel Processing*, Toronto, Ont., Canada, 2000, pp. 487–496.

[91] G. Ravindran and M. Stumm, "A performance comparison of hierarchical ring- and mesh-connected multiprocessor networks," in *Proceedings of the 3rd International Symposium on High-Performance Computer Architecture*, San Antonio, TX, USA, Feb. 1997, pp. 58–69.

[92] ——, "On topology and bisection bandwidth of hierarchical-ring networksfor shared-memory multiprocessors," in *Proceedings of the 5th International Conference OnHigh Performance Computing*, Madras, India, Dec. 17–20, 1998, pp. 262–269.

[93] N. Manjikian, "Prototyping a hierarchical ring interconnect for system-on-chip multiprocessor implementations," in *NEWCAS*, 2004, pp. 85–88.

[94] H. Wang, L.-S. Peh, and S. Malik, "A technology-aware and energy-oriented topology exploration for on-chip networks," in *Proceedings of Design, Automation and Test in Europe*, 2005, pp. 1238–1243.

[95] V. Soteriou, N. Eisley, H. Wang, B. Li, and L.-S. Peh, "Polaris: A system-level roadmap for on-chip interconnection networks," in *Proceedings of the 24th International Conference on Computer Design (ICCD)*, October 2006. [Online]. Available: http://www.gigascale.org/pubs/930.html

[96] L.-S. Peh and N. Eisley, "High-level power analysis for on-chip networks," in *7th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. Princeton University, September 2004. [Online]. Available: http://www.gigascale.org/pubs/551.html

[97] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 4, pp. 407–420, Apr. 2006.

[98] B. K. Penney and A. A. Baghdadi, "Survey of computer communications loop networks: Part 1," *Computer Communications*, vol. 2, no. 4, pp. 165–180, 1979.

[99] R. M. Metcalfe and D. R. Boggs, "Ethernet: distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, no. 7, pp. 395–404, 1976.

[100] L. Barroso and M. Dubois, "Performance evaluation of the slotted ring multiprocessor," *IEEE Transactions on Computers*, vol. 44, no. 7, pp. 878–890, Jul. 1995.

[101] ——, "Cache coherence on a slotted ring," in *Proceedings of the IEEE International Conference on Parallel Processing (ICPP)*, 1991, pp. 230–237.

[102] K. Loveless, "The implementation of flexible interconnect in the NUMAchine multiprocessor," Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, 1996.

[103] D. Sokolov and A. Yakovlev, "Clock-less circuits and system synthesis," in *System On Chip: Next Generation Electronics*, B. Al-Hashimi, Ed. IEE Press, january 2006, pp. 3–28.

[104] A. Iyer and D. Marculescu, "Power and performance evaluation of globally asynchronous locally synchronous processors," in *Proceedings of the 29th International Symposium on Computer Architecture (ISCA)*, May 2002, pp. 158–168. [Online]. Available: citeseer.ist.psu.edu/iyer02power.html

[105] K. Niyogi and D. Marculescu, "System level power and performance modeling of gals point-to-point communication interfaces," in *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*. New York, NY, USA: ACM Press, 2005, pp. 381–386.

[106] X. Jia and R. Vemuri, "Using GALS architecture to reduce the impact of long wire delay on fpga performance," in *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*. New York, NY, USA: ACM Press, 2005, pp. 1260–1263.

[107] M. Amde, T. Felicijan, A. Efthymiou, and D. E. amd Luciano Lavagno, "Asynchronous on-chip networks," in *System On Chip: Next Generation Electronics*, B. Al-Hashimi, Ed. IEE Press, january 2006, pp. 3–28.

[108] Wikipedia, "Metastability in electronics — wikipedia, the free encyclopedia," 2008, [Online; accessed 10-May-2008]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Metastability_in_electronics&oldid=202166405

[109] R. Ginosar, "Fourteen ways to fool your synchronizer," in *ASYNC '03: Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*. Washington, DC, USA: IEEE Computer Society, 2003, p. 89.

[110] A. Lines, "Asynchronous interconnect for synchronous soc design," *IEEE Micro*, vol. 24, no. 1, pp. 32–41, Jan./Feb. 2004.

[111] F. Gray, "Pulse code communication," Mar. 1953, U.S. patent number 2,632,058.

[112] Wikipedia, "Gray code — wikipedia, the free encyclopedia," 2008, [Online; accessed 10-May-2008]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Gray_code&oldid=211073009

[113] L. Benini and D. Bertozzi, "Network-on-chip architectures and design methods," in *System On Chip: Next Generation Electronics*, B. Al-Hashimi, Ed. IEE Press, january 2006, pp. 3–28.

[114] I. M. Panades and A. Greiner, "Bi-synchronous FIFO for synchronous circuit communication well suited for network-on-chip in GALS architectures," in *Proceedings of the 1st International Symposium on Networks-on-Chip (NOCS)*, Princeton, NJ,, May 2007, pp. 83–94.

[115] E. Beigne, , and P. Vivet, "Design of on-chip and off-chip interfaces for a GALS NoC architecture," in *2006. 12th IEEE International Symposium on Asynchronous Circuits and Systems*, Mar. 2006.

[116] H. de Fraysseix and P. O. de Mendez, "PIGALE: Public Implementation of a Graph Algorithm Library and Editor," http://pigale.sourceforge.net. [Online]. Available: http://pigale.sourceforge.net

[117] H. de Fraysseix and P. Rosenstiehl, "A depth-first search characterization of planarity." *Annals of Discrete Mathematics*, vol. 13, pp. 75–80, 1982.

[118] ——, "A characterization of planar graphs by trémaux orders." *Combinatorica*, vol. 5, no. 2, pp. 127–135, 1985.

[119] J. M. Boyer, P. F. Cortese, M. Patrignani, and G. D. Battista, "Stop minding your p's and q's: Implementing a fast and simple dfs-based planarity testing and embedding algorithm." in *Graph Drawing*, 2003, pp. 25–36.

[120] V. C. Hamacher and H. Jiang, "Hierarchical ring network configuration and performance modeling," *IEEE Transactions on Computers*, vol. 50, no. 1, pp. 1–12, Jan. 2001.

[121] J. W. Kwak and C. S. Jhon, "Performance evaluation of modified hierarchical ring by exploiting link utilization and memory access locality," in *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology*, Dec. 2005, pp. 82–87.

[122] T. Altman, Y. Igarashi, and K. Obokata, "Hyper-ring connection machines," in *TENCON '94. IEEE Region 10's Ninth Annual International Conference. Theme: 'Frontiers of Computer Technology'. Proceedings of 1994*, Aug. 22–26, 1994, pp. 290–294.

[123] F. N. Sibai, "The hyper-ring network: a cost-efficient topology for scalable multicomputers," in *SAC '98: Proceedings of the 1998 ACM symposium on Applied Computing*. New York, NY, USA: ACM Press, 1998, pp. 607–612.

[124] ——, "Performance of the hyper-ring multicomputer," in *SAC '98: Proceedings of the 1998 ACM symposium on Applied Computing*. New York, NY, USA: ACM Press, 1998, pp. 598–606.

[125] J. Wang, W. Yurcik, Y. Yang, and J. Hester, "Multiring techniques for scalable battlespace group communications," *IEEE Communications Magazine*, vol. 43, no. 11, pp. 124–133, Nov. 2005.

[126] N. Sherwani, *Algorithms for VLSI Physical Design Automation*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[127] K. G. Shin and S. W. Daniel, "Analysis and implementation of hybrid switching," *IEEE Transactions on Computers*, vol. 45, no. 6, pp. 684–692, 1996. [Online]. Available: citeseer.ist.psu.edu/shin95analysis.html

[128] A. V. de Mello, L. C. O. F. G. Moraes, and N. L. V. Calazans, "Evaluation of routing algorithms on mesh based NoCs," FACULDADE DE INFORMÁTICA – PUCRS, Brazil, Tech. Rep., 2004.

[129] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis – the SymTA/S approach," in *IEE Proceedings — Computers and Digital Techniques*, Mar. 2005, pp. 148–166. [Online]. Available: citeseer.ist.psu.edu/jersak05system.html

[130] B. S. Kuo, "Modeling and evaluation of a hierarchical ring interconnect for system-on-chip multiprocessing." Master's thesis, McGill University, Montreal, Canada, 2004.

[131] P. Paulin, C. Pilkington, and E. Bensoudane, "StepNP: A system-level exploration platform for network processors," *IEEE Design and Test of Computers*, vol. 19, no. 6, pp. 17–26, Nov. 2002.

[132] F. Moraes, A. Mello, L.M., L. Ost, and N. Calazans, "A low area overhead packet-switched network on chip: Architecture and prototyping." in *VLSI-SOC*, 2003, pp. 318–323.

[133] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69–93, 2004.

[134] J.-S. Chenard, S. Bourduas, N. Azuelos, M. Boule, and Z. Zilic, "Hardware assertion checkers in on-line detection of faults in a hierarchical-ring network-on-chip," in *Workshop on Diagnostic Services in Network-on-Chips*. DATE, Apr. 2007, pp. 371–375. [Online]. Available: http://www.date-conference.com/conference/2007/digest

[135] "TSMC 0.18μm Process 1.8-Volt SAGE-X Standard Cell Library Databook," Artisan Components Inc., Sunnyvale, California, USA.

[136] W. J. Dally, "Virtual-channel flow control," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, no. 2, pp. 194–205, 1992.

Glossary

AMBA
advanced microcontroller bus architecture. 16

ASIC
application specific integrated circuit. 2

CMP
Chip Multi-Processor. 2

DMA
direct memory access. 138

ECC
error correcting codes. 5

EDA
electronic design automation. 5

FPGA
field programmable gate array. 146

GALS
globally asynchronous, locally synchronous. 11

HDL
high-level description language. 19

IP
intellectual property. 2

IRI
inter-ring interface. 48

ISS
instruction set simulator. 138

LOC
lines of code. 95

LSI
large scale integration. 12

LTP
local traffic probability. 154

NA
network adapter. 15

NI
network interface. 15

NoC
Network-on-Chip. 3

OO
object oriented. 96

PE
proccessing element. 2

PSL
property specification language. 167

QoS
quality of service. 19

RI
ring interface. 48

RTL
register transfer level. 145

RTS
real-time system. 19

SA
simulated annealing. 113

SoC
System-on-a-Chip. 2

SRI
station-ring interface. 152

SSI
small scale integration. 12

TCP
transmission control protocol. 4

TLM
transaction-level model. 22

ULSI
ultra large scale integration. 12

UML
unified modeling language. 96

VLSI
very large scale integration. 12