Design and Analysis of On-Chip Communication for Network-on-Chip Platforms

Zhonghai Lu

Stockholm 2007

Department of Electronic, Computer and Software Systems
School of Information and Communication Technology
Royal Institute of Technology (KTH)
Sweden

Thesis submitted to the Royal Institute of Technology in partial fulfillment of the requirements for the degree of Doctor of Technology

Lu, Zhonghai
Design and Analysis of On-Chip Communication for Network-on-Chip Platforms

ISBN 978-91-7178-580-0
TRITA-ICT/ECS AVH 07:02
ISSN 1653-6363
ISRN KTH/ICT/ECS AVH-07/02 -SE

Royal Institute of Technology
School of Information and Communication Technology
Department of Electronic, Computer and Software Systems
Electrum 229
S-164 40 Kista, Sweden

Abstract

Due to the interplay between increasing chip capacity and complex applications, System-on-Chip (SoC) development is confronted by severe challenges, such as managing deep submicron effects, scaling communication architectures and bridging the productivity gap. Network-on-Chip (NoC) has been a rapidly developed concept in recent years to tackle the crisis with focus on network-based communication. NoC problems spread in the whole SoC spectrum ranging from specification, design, implementation to validation, from design methodology to tool support. In the thesis, we formulate and address problems in three key NoC areas, namely, *on-chip network architectures*, *NoC network performance analysis*, and *NoC communication refinement*.

Quality and cost are major constraints for micro-electronic products, particularly, in high-volume application domains. We have developed a number of techniques to facilitate the design of systems with low area, high and predictable performance. From flit admission and ejection perspective, we investigate the area optimization for a classical wormhole architecture. The proposals are simple but effective. Not only offering unicast services, on-chip networks should also provide effective support for multicast. We suggest a connection-oriented multicasting protocol which can dynamically establish multicast groups with quality-of-service awareness. Based on the concept of a logical network, we develop theorems to guide the construction of contention-free virtual circuits, and employ a back-tracking algorithm to systematically search for feasible solutions.

Network performance analysis plays a central role in the design of NoC communication architectures. Within a layered NoC simulation framework, we develop and integrate traffic generation methods in order to simulate network performance and evaluate network architectures. Using these methods, traffic patterns may be adjusted with locality parameters and be configured per pair of tasks. We propose also an algorithm-based analysis method to estimate whether a wormhole-switched network can satisfy the timing constraints of real-time messages. This method is built on traffic assumptions and based on a contention tree model that captures direct and indirect network contentions and concurrent link usage.

In addition to NoC platform design, application design targeting such a platform is an open issue. Following the trends in SoC design, we use an abstract and formal specification as a starting point in our design flow. Based on the synchronous model of computation, we propose a top-down communication refinement approach. This approach decouples the tight global synchronization into process local synchronization, and utilizes synchronizers to achieve process synchronization consistency during refinement. Meanwhile, protocol refinement can be incorporated to satisfy design constraints such as reliability and throughput.

The thesis summarizes the major research results on the three topics.

Table of Contents

Acknowledgments **vii**

List of Publications **ix**

List of Figures **xiv**

Abbreviations **xvi**

1 Introduction **1**
1.1 Network-on-Chip (NoC) 1
1.1.1 System-on-Chip (SoC) Design Challenges 1
1.1.2 Network-on-Chip as a SoC Platform 4
1.1.3 On-Chip Communication Model 8
1.2 Research Overview 10
1.3 Author’s Contributions 11

2 NoC Network Architectures **17**
2.1 Introduction 17
2.1.1 On-Chip Communication Network 17
2.1.2 Wormhole Switching 25
2.2 Flit Admission and Ejection 26
2.2.1 Problem Description 27
2.2.2 The Wormhole Switch Architecture 28
2.2.3 Flit Admission 31
2.2.4 Flit Ejection 33
2.3 Connection-oriented Multicasting 36
2.3.1 Problem Description 36
2.3.2 The Multicasting Mechanism 37
2.4 TDM Virtual-Circuit Configuration 38
2.4.1 Problem Description 38

2.4.2 Logical-Network-oriented VC Configuration 39
2.5 Future Work 43

3 NoC Network Performance Analysis 45
3.1 Introduction 45
3.1.1 Performance Analysis for On-Chip Networks 45
3.1.2 Practices of NoC Simulation 49
3.2 NNSE: Nostrum NoC Simulation Environment 50
3.2.1 Overview 50
3.2.2 The Simulation Kernel 51
3.2.3 Network Configuration 52
3.2.4 Traffic Configuration 53
3.2.5 An Evaluation Case Study 56
3.3 Feasibility Analysis of On-Chip Messaging 57
3.3.1 Problem Description 57
3.3.2 The Network Contention Model 58
3.3.3 The Feasibility Test 65
3.4 Future Work 69

4 NoC Communication Refinement 71
4.1 Introduction 71
4.1.1 Electronic System Level (ESL) Design 71
4.1.2 Communication Refinement 72
4.1.3 Synchronous Model of Computation (MoC) 73
4.2 The Communication Refinement Approach 76
4.2.1 Problem Description and Analysis 76
4.2.2 Refinement Overview 79
4.2.3 Channel Refinement 81
4.2.4 Process Refinement 84
4.2.5 Communication Mapping 90
4.3 Future Work 91

5 Summary 93
5.1 Subject Summary 93
5.2 Future Directions 94

References 97

Appended papers 111

Acknowledgements

Studying towards Ph.D. takes five years, with 20% teaching workload. It is a long process filled with mixed feelings, pleasure and pressure, satisfaction and disappointment. The pleasure and enjoyment originate from the persistent development of innovative ideas in the frontline of the interesting research area. The pressure may undergo in the face of frequent deadlines for tasks and papers, especially during the tight finance period in the department. The satisfaction comes often for the recognition of a piece of innovative work while the disappointment may occur for the sake of mis-understanding and rejections. My life as a Ph.D. student is a colorful picture, which composes a beautiful and important part of my life.

While this picture is painted, a lot of people have contributed in various ways. It is the right time and place to acknowledge their help. Professor Axel Jantsch is the one I thank most. I thank him far more just because of the fact that he is my supervisor. He is a very respectable person for his personality, knowledge and creativity. I am lucky to be one of his students. In reality, he treats his students very equally. He is not only a supervisor but also a colleague and a collaborator. I am indebted to Dr. Ingo Sander. He has been acting as the co-supervisor for my Ph.D. study, and helping me in all the ways possible. He is concerned with not only my progress in research but also my office and health. We have had many and many small talks and discussions, which are sources of friendship and inspiration. I thank Professor Shashi Kumar, who was my co-supervisor before he left KTH.

I acknowledge valuable discussions and diverse help from all my colleagues in the System, Architecture and Methodology (SAM) group, particularly, the two project teams, *Nostrum* and *ForSyDe*, where I have been involved in. The Nostrum team investigates network-on-chip architectures and associated design techniques. The present and past contributors include Mikael Millberg, Rikard Thid, Erland Nilsson, Raimo Haukilahti, Johnny Öberg, Kim Petersen and Per Badlund. Particularly, I thank Rikard for his original work in the layered NoC simulation kernel. The ForSyDe team aims to develop a formal design methodology for System-on-Chip applications from modeling to implementation and verification. This team involves Ingo Sander, Tarvo Raudvere, Ashish Kumar Singh and Jun Zhu.

I appreciate our system group, Hans Berggren and Peter Magnusson, for their active and patient support in computer and network systems. I thank secretaries Lena Beronius, Agneta Herling and Rose-Marie Lövenstig for their administrative assistance in traveling and other issues.

I thank all other colleagues in the Department of Electronic, Computer and Software Systems for their various help and for the pleasant and encouraging environment we contribute to and share. I thank Roshan Weerasekera for friendship. Special thanks should go to all my Chinese colleagues, particularly to Lirong, Jian Liu, Li Li, Bingxin, and Jinliang, for the great occasions and happiness we share.

During my Ph.D. study period, I have supervised twelve Master theses. These works have deepened my understanding on the corresponding subjects and most of them are excellent. I thank all the students for their hard and fruitful work, particularly, Bei Yin, Mingchen Zhong, Li Tong, Karl-Henrik Nielsen, Jonas Sicking and Ming Liu.

I have taken an internship in Samsung Electronics in the summer of 2005. During the three-month period, I investigated the state-of-the-art interconnect techniques. I thank Mr. Soo Kwan Eo, Dr. Cheung and Dr. Yoo and all others in the system design technology group for their arrangements and assistance.

Finally I give my deepest gratitude to my family, my wife Yanhong and daughter Lingyi. I could not count how many weekends I have spent with my computer, and how many times I have been late back home. Any piece of my achievement has an invisible part of their contribution. I thank my brothers and sisters in China for their endless concerns. I thank my parents for their permanent love and irreplaceable support.

The research presented in the dissertation is financed by the Swedish government within the SoCware program and the European Commission within the Sprint project.

Zhonghai Lu

December 2006, Stockholm

List of Publications

Part A. Papers included in the thesis:

- **NoC Network Architectures**

1. Zhonghai Lu and Axel Jantsch. Flit admission in on-chip wormhole-switched networks with virtual channels. In *Proceedings of the International Symposium on System-on-Chip*, pages 21-24, Tampere, Finland, November 2004.

2. Zhonghai Lu and Axel Jantsch. Flit ejection in on-chip wormhole-switched networks with virtual channels. In *Proceedings of the IEEE NorChip Conference*, pages 273-276, Oslo, Norway, November 2004.

3. Zhonghai Lu, Bei Yin, and Axel Jantsch. Connection-oriented multicasting in wormhole-switched networks on chip. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'06)*, pages 205-210, Karlsruhe, Germany, March 2006.

4. Zhonghai Lu and Axel Jantsch. TDM virtual-circuit configuration in network-on-chip using logical networks. In submission to *IEEE Transactions on Very Large Scale Integration Systems*.

- **NoC Network Performance Analysis**

5. Zhonghai Lu and Axel Jantsch. Traffic configuration for evaluating networks on chip. In *Proceedings of the 5th International Workshop on System-on-Chip for Real-time Applications*, pages 535-540, Alberta, Canada, July 2005.

6. Zhonghai Lu, Mingchen Zhong, and Axel Jantsch. Evaluation of on-chip networks using deflection routing. In *Proceedings of the 16th ACM Great Lakes Symposium on VLSI (GLSVLSI'06)*, pages 296-301, Philadelphia, USA, May 2006.

7. Zhonghai Lu, Axel Jantsch and Ingo Sander. Feasibility analysis of messages for on-chip networks using wormhole routing. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC'05)*, pages 960-964, Shanghai, China, January 2005.

- **NoC Communication Refinement**

8. Zhonghai Lu, Ingo Sander, and Axel Jantsch. Refining synchronous communication onto network-on-chip best-effort services. In Alain Vachoux, editor, *Applications of Specification and Design Languages for SoCs - Selected papers from FDL 2005*, Chapter 2, pages 23-38, Springer, 2006.

9. Zhonghai Lu, Ingo Sander, and Axel Jantsch. Towards performance-oriented pattern-based refinement of synchronous models onto NoC communication. In *Proceedings of the 9th Euromicro Conference on Digital System Design (DSD'06)*, pages 37-44, Dubrovnik, Croatia, August 2006.

Part B. Publications not included in the thesis:

10. Zhonghai Lu, Ingo Sander, and Axel Jantsch. Refinement of a perfectly synchronous communication model onto Nostrum NoC best-effort communication service. In *Proceedings of the Forum on Specification and Design Languages (FDL'05)*, Lausanne, Switzerland, September 2005.

11. Zhonghai Lu, Li Tong, Bei Yin, and Axel Jantsch. A power-efficient flit-admission scheme for wormhole-switched networks on chip. In *Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics*, Florida, U.S.A., July 2005.

12. Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson, and Axel Jantsch. NNSE: Nostrum network-on-chip simulation environment. In *Proceedings of Swedish System-on-Chip Conference*, Stockholm, Sweden, April 2005.

13. Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson, and Axel Jantsch. NNSE: Nostrum network-on-chip simulation environment. In *The University Booth Tool-Demonstration Program of the Design Automation and Test in Europe Conference*, Munich, Germany, March 2005.

14. Ingo Sander, Axel Jantsch, and Zhonghai Lu. Development and application of design transformations in ForSyDe. *IEE Proceedings - Computers & Digital Techniques*, 150(5):313-320, September 2003.

15. Zhonghai Lu and Axel Jantsch. Network-on-chip assembler language (version 0.1). Technical Report TRITA-IMIT-LECS R 03:02, ISSN 1651-4661, ISRN KTH/IMIT/LECS/R-03/02-SE, Royal Institute of Technology, Stockholm, Sweden, June 2003.

16. Ingo Sander, Axel Jantsch, and Zhonghai Lu. Development and application of design transformations in ForSyDe. In *Proceedings of Design, Automation and Test in Europe Conference*, pages 364-369, Munich, Germany, March 2003.

17. Zhonghai Lu and Raimo Haukilahti. NoC application programming interfaces. In Axel Jantsch and Hannu Tenhunen, editors, *Networks on Chip*, Chapter 12, pages 239-260. Kluwer Academic Publishers, February 2003.

18. Zhonghai Lu, Ingo Sander, and Axel Jantsch. A case study of hardware and software synthesis in ForSyDe. In *Proceedings of the 15th International Symposium on System Synthesis (ISSS'02)*, pages 86-91, Kyoto, Japan, October 2002.

19. Zhonghai Lu and Axel Jantsch. Admitting and Ejecting Flits in Wormhole-switched On-chip Networks. In submission to *Journal of Systems Architectures* (under the second round review).

List of Figures

1.1 A mesh NoC with 9 nodes 5
1.2 On-chip communication layers with Application Level Interface (ALI) and Core Level Interface (CLI) 9

2.1 Flits delivered in a pipeline 25
2.2 Virtual channels (lanes) 26
2.3 Flit admission and ejection 27
2.4 Flitization and assembly 27
2.5 A canonical wormhole lane switch (ejection not shown) 29
2.6 Lane-to-lane associations 30
2.7 Organization of packet- and flit-admission queues 31
2.8 The coupled admission sharing a (p+1)-by-p crossbar 33
2.9 The ideal sink model 34
2.10 The p-sink model 35
2.11 The virtual-circuit configuration problem 39
2.12 TDM virtual circuits 40
2.13 Using logical networks to avoid conflict 41
2.14 The view of logical networks 41
2.15 Virtual-circuit configuration approaches 44

3.1 Network evaluation 51
3.2 The communication layers in Semla 51
3.3 Network configuration tree 53
3.4 The traffic configuration tree 54
3.5 Feasibility analysis in a NoC design flow 57
3.6 Network contention and contention tree 60
3.7 Message contention for links simultaneously 62
3.8 Avoided flit-delivery scenario 63
3.9 Message scheduling 64

3.10 A three-node contention tree 65
3.11 Message scheduling and contended slots 66
3.12 A feasibility analysis flow 68

4.1 Computation and communication elements 77
4.2 NoC communication refinement 80
4.3 Processes for synchronization 86
4.4 Wrap a strong process 87
4.5 Wrap a strict process 87
4.6 A strong and a strict process 87
4.7 Two non-strict processes 87
4.8 Feedback loop 89
4.9 A relax-synchronization process 89

Abbreviations

ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
ALI	Application-Level Interface
CAD	Computer Aided Design
CLI	Core-Level Interface
CMOS	Complementary Metal Oxide Semiconductor
CT	Contention Tree
DSM	Deep SubMicron
DTL	Device Transaction Level
ESL	Electronic System Level
FIFO	First In First Out
ForSyDe	FORmal SYstem DEsign
FPGA	Field-Programmable Gate Array
GUI	Graphical User Interface
ITRS	International Technology Roadmap for Semiconductors
IP	Intellectual Property
LN	Logical Network
MoC	Model of Computation
NNSE	Nostrum Network-on-Chip Simulation Environment
NoC	Network on Chip
OCP	Open Core Protocol
PC	Physical Channel
PE	Processing Element
QoS	Quality of Service
RNI	Resource Network Interface
RT	Real Time
RTL	Register Transfer Level
SEMLA	Simulation EnvironMent for Layered Architecture
SoC	System on Chip
TDM	Time-Division Multiplexing
ULSI	Ultra Large Scale Integration
VC	Virtual Channel / Virtual Circuit
VCI	Virtual Component Interface

Chapter 1

Introduction

This chapter highlights System-on-Chip design challenges and introduces the Network-on-Chip concept. We also give an overview of the research presented in the thesis and outline the author's contributions to the enclosed papers.

1.1 Network-on-Chip (NoC)

1.1.1 System-on-Chip (SoC) Design Challenges

Our life has been largely shaped by the exciting developments of modern electronic technologies, such as pervasive and ubiquitous computing, ambient intelligence, communication, and Internet. Today micro-electronic products are influencing the ways of communication, learning and entertainment. The key driving force for the developments during decades is the System-on-Chip (SoC) technologies, where complex applications are integrated onto single ULSI chips. Not only functionally enriched, these products such as mobile phones, notebooks and personal handheld sets are becoming faster, smaller-in-size, larger-in-capacity, lighter-in-weight, lower-in-power-consumption and cheaper. One could favorably think that this trend will persistently continue. Following this trend, we could integrate more and more complex applications and even systems onto a single chip. However, our current methodologies for SoC design and integration do not evenly advance due to the big challenges confronted.

- *Deep SubMicron (DSM) effects* [43, 80, 134]: In early days of VLSI design, signal integrity effects such as interconnect delay, crosstalk, inter-symbol interference, substrate coupling, transmission-line effects, etc. were negligible due to relatively slow clock speed and low integration density. Chip interconnect was reliable and robust. At the scale of 250 nm with aluminum and 180 nm with copper and below, interconnect started to become a dominating factor for chip performance and robustness. As the transistor density is increased, wires are getting neither fast nor reliable [43]. More noise sources due to inductive fringing, crosstalk and transmission line effects are coupled to other circuit nodes globally on the chip via the substrate, common return ground and electromagnetic interference. More and more aggressive use of high-speed circuit families, for example, domino circuitry, scaling of power supply and threshold voltages, and mixed-signal integration combine to make the chips more noise-sensitive. Third, higher device densities and faster switching frequencies cause larger switching-currents to flow in the power and ground networks. Consequently, power supply is plagued with excessive IR voltage drops as wells as inductive voltage drops over the power distribution network and package pins. Power supply noise degrades not only the driving capability of gates but also causes possible false switching of logical gates. Today signal and power integrity analysis is as important as timing, area and power analysis.

- *Global synchrony* [3, 47]: Predominating digital IC designs have been following a globally synchronous design style where a global clock tree is distributed on the chip, and logic blocks function synchronously. However, this style is unlikely to survive with future wire interconnect. The reason is that technology scaling does not treat wire delay and gate delay equally. While gate delay (transistor switching time) has been getting dramatically smaller in proportion to the gate length, wires have slowed down. As the chip becomes communication-bound at 130 nm, multiple cycles are required to transmit a signal across its diameter. As estimated in [3], with the process technology of 35 nm in year 2014, the latency across the chip in a top-level metal wire will be 12 to 32 cycles depending on the clock rate assuming best transmission conditions such as very low-permittivity dielectrics, resistivity of pure copper, high aspect ratio (ratio of wire height to wire width) wires and optimally placed repeaters. Moreover, a clock tree is consuming larger portions of power and area budget and clock skew is claiming an ever larger portion of the total cycle time [94]. Even if we have an unlimited number of transistors on a chip, chip design is to be constrained by communication rather than capacity. A future chip is likely to be partitioned into locally synchronous regions but global communication is asynchronous, so called GALS (Globally Asynchronous Locally Synchronous).

- *Communication architecture* [9, 19]: Most current SoCs have a bus-based architecture, such as simple, hierarchical or crossbar-type buses. In contrast to the scaling of chip capacity, buses do not scale well with the system size in terms of bandwidth, clocking frequency and power. First, a bus system has very limited concurrent communication capability since only one device can drive a bus segment at a time. Current SoCs integrate fewer than five processors and, rarely, more than 10 bus masters. Second, as the number of clients grows, the intrinsic resistance and capacitance of the bus also increase. This means that the bus speed is inherently difficult to scale up. Third, a bus is inefficient in energy since every data transfer is broadcast. The entire bus wire has to be switched on and off. This means that the data must reach each receiver at great energy cost. Although improvements such as split-transaction protocols and advanced arbitration schemes for buses have been proposed, these incremental techniques can not overcome the fundamental problems. To explore the future chip capacity, for high-throughput and low-power applications, hundreds of processor-sized resources must be integrated. A bus-based architecture would become a critical performance and power bottleneck due to the scalability problem. Novel on-chip communication architectures are desired.

- *Power and thermal management* [85, 105]: As circuits run with higher and higher frequencies, lowering power consumption is becoming extremely important. Power is a design constraint, which is no more subordinate to performance. Despite process and circuit improvements, power consumption shows rapid growth. Equally alarming is the growth in power density on the chip die, which increases linearly. In face of DSM effects, reducing power consumption is becoming even more challenging. As devices shrink to submicron dimensions, the supply voltage must be reduced to avoid damaging electric fields. This development, in turn, requires a reduced threshold voltage. However, leakage current increases exponentially with a decrease in the threshold voltage. In fact, a 10% to 15% reduction can cause a two-fold increase in leakage current. In increasingly smaller devices, leakage will become the dominant source of power consumption. Further, leakage occurs as long as power flows through the circuit. This constant current can produce an increase in the chip temperature, which in turn causes an increase in the thermal voltage, leading to a further increase in leakage current.

- *Verification* [107, 111]: Today SoC design teams are struggling with the complexity of multimillion gate designs. System verification runs through the whole design process from specification to implementation, typically with formal methods or simulation-based validation. As the system has become extremely complex, the verification or validation consumes an increasing portion of the product development time. The verification effort has reached as high as 70% of engineering efforts.

- *Productivity gap* [4, 111]: Simply put, productivity gap is the gap between what we are capable of building and what we are capable of designing. In line with Moore's law [83], the logic capacity of a single chip has increased at the rate of 58% per annum compounded. Soon the complexity of the chip enters the billion-transistor era. The complexity of developing SoCs is increasing continuously in order to exploit the potential of the chip capacity. However, the productivity of hardware and software design is not growing at a comparable pace. The hardware design productivity is increased at a rate in the range 20% to 25% per annum compounded. Even worse, the software design productivity improves at a rate in the range from 8% to 10% per annum compounded. As a consequence, the costs of developing advanced SoCs are increasing at an alarming pace and time-to-market is negatively affected. The design team size is increased by more than 20% per year. This huge investment is becoming a serious threshold for new product developments and is slowing down the innovation in the semiconductor industry. As stated in the ITRS roadmap [4], cost of design is the greatest threat to continuation of the semiconductor roadmap.

1.1.2 Network-on-Chip as a SoC Platform

Innovations occur where challenges are present. Network-on-Chip (NoC) was proposed in face of those challenges in around year 2001 in the SoC community [9, 27, 37, 41, 109, 119]. In March 2000, packet-switched networks were proposed in SPIN [37] as a global and scalable SoC interconnection. The term *Network-on-Chip* appeared initially in November 2000 [41] where NoC was proposed as a platform to cope with the productivity gap. In June 2001, Dally and Towles proposed NoC as a structured way of communication to connect IP modules [27]. The GigaScale Research Center (GSRC) suggested NoC to address interconnection woes [119]. In October 2001, researchers from the Philips Research presented a router architecture supporting both best-effort and guaranteed-throughput traffic for Network-on-Silicon [109]. In January 2002, Luca and De Micheli formulated NoC as a new SoC paradigm [9]. While network-on-chip is still in its infancy, the

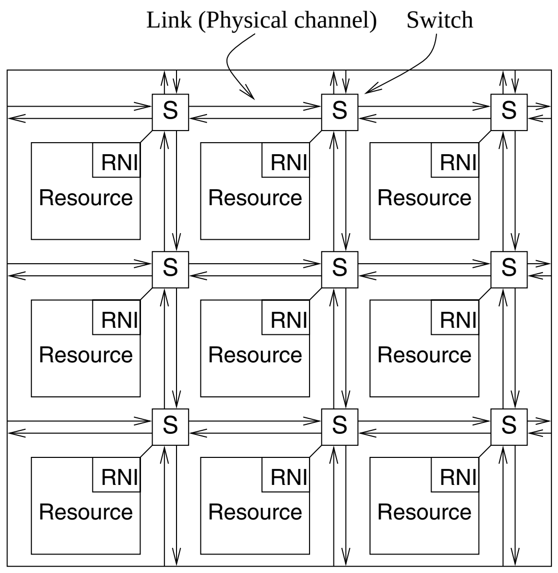


Figure 1.1. A mesh NoC with 9 nodes

concept has spread and been accepted in academia very rapidly. Some big companies, for instance, NXP semiconductors (former part of Philips Semiconductors) and ST Micro-electronics, are also very active in this field [34, 52]. A comprehensive survey on current research and practices of NoC can be found in [13].

Aimed to be a systematic approach, NoC proposes networks as a scalable, reusable and global communication architecture to address the SoC design challenges. As an instance, *Nostrum* [81, 90] is the name of the Network-on-Chip concept developed at the Royal Institute of Technology (KTH), Sweden. It features a mesh structure composed of switches with each resource connected to exactly one switch, as shown in Figure 1.1. A resource can be a processor, memory, ASIC, FPGA, IP block or a bus-based subsystem. The resources are placed on the slots formed by the switches. The maximal resource area is defined by the maximal synchronous region of a technology. The resources perform their own computational, storage and/or I/O processing functionalities, and are equipped with Resource-Network-Interfaces (RNIs) to communicate with each other by routing packets instead of driving dedicated wires.

Communication network is a well-known concept developed in the context of telephony, computer communication as well as parallel machines. On-chip networks share many characteristics with these networks, but also have significant differences. For clear presentation, throughout the thesis, we also call an on-chip network a *micro-network*, a parallel-machine network a *macro-network*, a telephony or computer network a *tele-network*. On-chip networks are developed on a single chip and designed for closed systems targeting perhaps heterogeneous applications. Parallel-machine networks are developed on distributed boards and designed for a particular application which typically executes specific algorithms. Computer networks are geographically distributed and designed for open systems running diverse applications from client-server, peer-to-peer and multicast applications. Telephony networks are also geographically distributed but are designed mainly for the purposes of communicating voice, video and data. The design of a closed system allows for customization in which the network properties including the network-level, link-level and physical-level properties can be propagated to the application level and both communication and computation may be efficiently optimized. Since a micro-network is built on a single chip, it can have wide parallel wires and allows high rate synchronous clocking. On the other hand, it has more stringent constraints in performance, area and power, which are typical trade-off considerations for SoC designs. As communication is to transfer data, timing is the first-level citizen. On-chip networks have the strictest requirement on delay and jitter. The time scale is measured in nano seconds. This requirement precludes many of the software-based sophisticated arbitration, routing and flow-control algorithms. Cost is a major concern for on-chip networks since most SoCs target high-volume markets. The buffering in an on-chip network has very limited space and is expensive in comparison with board-level, local-area and wide-area networks. This means that a NoC allows a limited count of routing tables and virtual-channel buffers in network nodes. Power consumption is important for all kinds of networks. However, on-chip networks are developed also for embedded applications with battery-driven devices. Such applications require extremely low power which is not comparable to large-scale networks. As we also mentioned, on-chip network designs are confronted by the DSM effects. Taming bad physical effects is as important as network design itself. Furthermore, many SoC networks are developed as a platform for multiple use cases, not only for a single use case. Therefore designing micro-networks also need to take reconfigurability into account.

As we view it, Network-on-Chip is a revolutionary rather than evolutionary approach to address the SoC design crisis. It shifts our focus from computation to communication. It should take interconnect into early consideration in the design process, and might favor a meet-in-the middle (platform-based) design methodology against a top-down or bottom-up approach. NoC has the following features:

- *Interconnect-aware* [93]: As the technology scales, the reachable region in one clock cycle diminishes [3]. Consequently, chip design is increasingly becoming communication-bound rather than capacity-bound. Since the size of a single module is limited by the reachable region in one cycle, to exploit the huge chip capacity, the entire chip has to be partitioned into multiple regions. A good partitioning should be regular, making it easier to manage the properties of long wires including middle-layer and top-layer wires. Each module is situated in one partitioned region and maintains its own synchronous region. In this way, the reliance on global synchrony and use of global wires can be alleviated. To guarantee correct operation, registers may be used in wire segments to make the design latency-insensitive [17]. Besides, each IP may be attached to a switch. Switches are in turn connected with each other to route packets in the network. The signal and power integrity issues may be addressed at the physical, link and higher layers. For example, redundancy in time, space and information can be incorporated in transmission to achieve reliability. By physically structuring the communication and successfully suppressing the DSM effects, the design robustness and reliability can be improved.

- *Communication-centric* [10]: Networking distributed IP modules in a partitioned chip results in a naturally parallel communication infrastructure. As long as the chip capacity is not exceeded, the number of cores which can be integrated on a single chip is scalable. The inter-core communications share the total network bandwidth with a high degree of concurrency. The network can be dimensioned to suit the bandwidth need of the application under interest. The parallel architecture allows concurrent processing in computation and communication. This helps to leverage performance and reduce power in comparison with a sequential architecture permitting only sequentialized processing. A protocol stack is typically built to abstract the network-based communication. Each layer has well-defined functionalities, protocols and interfaces. The design space at each layer has to be sufficiently explored. The tradeoffs between performance and cost should be considered in the design, analysis and implementation of the communication architecture. Quality-of-Service (QoS) and system-wide performance analysis are central issues to address predictability.

- *Platform-based* [50, 87]: Since the cost of design is the major obstacle for innovative and complex SoCs [46], developing a programmable, reconfigurable and extensible communication platform is essential for SoC designs. To this end, NoC shall serve as a communication and integration platform providing a hardware communication architecture, an associated interconnect interface, as well as a high-level interface for integrating hardware IPs, custom logic and for software programming. This enables the architecture-level reuse. One challenge is to address the balance between generality and optimality. A platform must serve not only one application but also many applications within an application domain. On the other hand, customization to enhance performance and efficiency is needed to make designs competitive. Providing well-defined interfaces at least at the network level and the application level is important, because it enables IPs and functional blocks to be reusable. Interface standardization is one major concern to make IPs from different vendors exchangeable. It must be efficient and also addresses legacy IPs. The concept of interface-based design has been shown successful for IP plug-and-play in the history of software and hardware developments, for example, instruction sets and various interconnect buses or protocols such as Peripheral Component Interface (PCI) and Universal Serial Bus (USB). A NoC design methodology should also favor communication interfaces for the greatest possible IP reuse and integration [112, 129]. Using validated components and architectures in a design flow shrinks verification effort, reduces time-to-market and guarantees product quality, thus enhancing design productivity.

As such, NoC research does not deal with only several aspects of SoC design but creates a new area [50]. The term NoC is used today mostly in a very broad meaning. It encompasses the hardware communication infra-structure, the middleware and operating system, application programming interfaces [64, 101], the design methodology and its associated tool chain. The challenges for NoC research have thus been distributed in all aspects of SoC design from architecture to performance analysis, from traffic characterization to application design.

1.1.3 On-Chip Communication Model

On-chip communication is to provide a means to enable interprocess communication with a set of constraints and properties satisfied. A good view of network-based process-to-process communication is to follow the ISO's OSI model [135]. The seven-layer model was proposed to interconnect open systems, which are heterogeneous and distributed. The layered structure decomposes the communication problem into more manageable components at different hierarchical layers. Rather than a monolithic structure, several layers are designed, each of which solves one part of the problem. Besides, layering provides a more modular design. At each

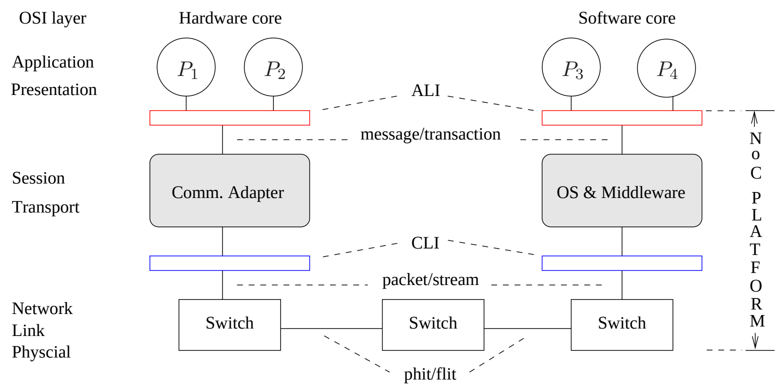


Figure 1.2. On-chip communication layers with Application Level Interface (ALI) and Core Level Interface (CLI)

layer, protocols and services, which are implementation-independent, are well-defined. Peer entities at the same layer can thus communicate with each other transparently. Adding new services to one layer may only need to modify the functionality at one layer, reusing the functions provided at all the other layers. Due to these advantages, several NoC groups [9, 82, 119] have followed this model and adapted it to build a protocol stack for on-chip communication.

As a platform, NoC shall provide well-defined interfaces for application programming and IP integration. Two levels of interfaces can be identified. One is the *Core-Level Interface (CLI)*, which is used to connect hardware cores. At this level, IPs and processors implement interfaces such as AXI [6], OCP [95], VCI [130], CoreConnect [45] and DTL [104]. The other level interface is for integrating hardware logic via a communication adapter and for programming embedded software. The Operating System (OS) [92] and middleware can be part of the platform. This level of interface is *Application-Level Interface (ALI)*. A recent proposal of the two-level interfaces for multiprocessors on chip can be found in [129].

An on-chip communication model combines the two views: *abstract layered communication* and *interface-based communication*. Although having been discussed separately, the two views are coherent, as shown in Figure 1.2. As can be seen, hardware and software processes (illustrated as P_1, P_2, P_3, P_4 in Figure 1.2) representing the application layer use the ALI. The hardware communication adapter for integrating hardware cores and operating system & middleware for integrating software cores realize the session and transport layers, and connect to the CLI. The CLI encapsulates the network. It is worth noting that bypassing one layer is possible, as long as the interfaces match. For example, if a hardware IP implements the CLI, it can be directly connected to the CLI instead of connecting to the ALI, bypassing the communication adapter.

1.2 Research Overview

We have been orienting our NoC research towards three key issues: *on-chip network architectures*, *network performance analysis* and *application design methodology*. The network communication architectures deal with the design of on-chip networks. The performance analysis evaluates the network performance and helps to uncover the impact of network parameters on performance. The design methodology is concerned with how to design applications on a NoC platform. Specifically, we deal with communication refinement that synthesizes the communication in a system model into on-chip communication. Essentially these topics deal with the design and analysis of on-chip communication for NoC platforms.

We have identified and formulated problems related to the three aspects mentioned above. The thesis is based on the research results from these studies. In the following, we give a brief sketch of the main results:

- *NoC network architectures*: We have proposed cost-effective switch architectures, a connection-oriented multicasting scheme, as well as a TDM (Time Division Multiplexing) virtual-circuit configuration method using logical networks. After studying wormhole switch micro-architectures, we propose flit admission and ejection schemes, which are cost-effective with minimal performance penalty. Our multicasting mechanism is also proposed for wormhole-switched networks. It is connection-oriented, and a connection can be established dynamically. Based on the concept of a logical network, we have developed theorems and used a back-tracking algorithm to configure contention-free TDM virtual-circuits.

- *NoC network performance analysis*: We have investigated traffic configuration, carried out network simulation and made feasibility analysis. We propose how to configure synthetic traffic patterns using distribution with controllable locality or channel-by-channel customization. This traffic configuration method has been integrated into our Nostrum NoC Simulation Environment (NNSE). A case study on the deflection networks shows that our simulator enables to explore the architectural design space and helps to make proper decisions on topology, routing schemes and deflection policies. The feasibility analysis aids designers with information about whether the application can fulfill the timing requirements of messages on the network and how efficient network resources can be utilized. It allows one to evaluate the network using algorithm instead of simulation. Hence, it is more efficient but less accurate. This feasibility analysis is performed on wormhole-switched networks.

- *NoC communication refinement*: Based on a synchronous system model, we have proposed a communication refinement approach that refines the abstract communication into network-based communication. During the refinement, synchronization consistency is maintained in order to be correct-by-construction and protocol refinement can be incorporated to satisfy performance constraints.

Next, we summarize the author's contributions in each of the enclosed papers.

1.3 Author's Contributions

The thesis is based on a collection of papers, which are all peer-reviewed except Paper 4 that is under review. The papers are grouped into three blocks, namely, *NoC network architectures*, *NoC network performance analysis*, and *NoC communication refinement*. Each block is dedicated to one chapter in the thesis and we concentrate on introducing the author's contributions in each chapter. The detailed materials, experiments, results and other related work are referred to the papers. In the following, we summarize the enclosed papers highlighting the author's contributions. These papers are also listed in the references.

- **NoC Network Architectures**

Paper 1 [66]. Zhonghai Lu and Axel Jantsch. Flit admission in on-chip wormhole-switched networks with virtual channels. In *Proceedings of the International Symposium on System-on-Chip*, pages 21-24, Tampere, Finland, November 2004.

This paper discusses the flit admission problem in input-buffering and output-buffering wormhole switches. Particularly it presents a novel cost-effective coupling scheme that binds flit admission queues with output physical channels in a one-to-one correspondence manner. The experiments suggest that the network performance is equivalent to the base line scheme which connects a flit admission queue to all the output physical channels.

Author's contributions: The author contributed with the problem formulation, conducted experiments and wrote the manuscript.

Paper 2 [67]. Zhonghai Lu and Axel Jantsch. Flit ejection in on-chip wormhole-switched networks with virtual channels. In *Proceedings of the IEEE NorChip Conference*, pages 273-276, Oslo, Norway, November 2004.

This paper studies flit ejection models in a wormhole virtual channel switch. Instead of the costly ideal flit-ejection model, two alternatives which largely reduce the buffering cost are proposed. Experiments show that the p-sink model achieves nearly equivalent performance with the ideal sink model if the network is not overloaded.

Author's contributions: The author formulated the flit-ejection problem, proposed solutions, conducted experiments and wrote the manuscript.

Paper 3 [75]. Zhonghai Lu, Bei Yin, and Axel Jantsch. Connection-oriented multicasting in wormhole-switched networks on chip. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'06)*, pages 205-210, Karlsruhe, Germany, March 2006.

This paper presents a connection-oriented multicast scheme in wormhole-switched NoCs. In this scheme, a multicast procedure consists of establishment, communication and release phases. A multicast group can request to reserve virtual channels during establishment and has priority on arbitration of link bandwidth. This multicasting method has been effectively implemented in a mesh network with deadlock freedom. Our experiments show that the multicast technique improves throughput, and does not exhibit significant impact on unicast performance in a network with mixed unicast and multicast traffic.

Author's contributions: The author contributed with the idea and protocol design, suggested experimentation methods, and wrote the manuscript. The implementation and experiments were conducted by Bei Yin.

Paper 4 [65]. Zhonghai Lu and Axel Jantsch. TDM virtual-circuit configuration in network-on-chip using logical networks. In submission to *IEEE Transactions on Very Large Scale Integration Systems*.

Configuring Time-Division-Multiplexing (TDM) Virtual Circuits (VCs) on network-on-chip must guarantee conflict freedom for VCs besides allocating sufficient time slots to them. Using the generalized concept of logical networks, we develop and prove theorems that constitute sufficient and necessary conditions to establish conflict-free VCs. Moreover, we give a formulation of the multi-node VC configuration problem and suggest a back-tracking algorithm to find solutions by constructively searching the solution space.

Author's contributions: The author developed and proved the theorems, formulated the problem, wrote the program, conducted experiments and wrote the manuscript.

- **NoC Network Performance Analysis**

Paper 5 [68]. Zhonghai Lu and Axel Jantsch. Traffic configuration for evaluating networks on chip. In *Proceedings of the 5th International Workshop on System-on-Chip for Real-time Applications*, pages 535-540, Alberta, Canada, July 2005.

This paper details the traffic configuration methods developed for NNSE. It presents a unified expression to configure both uniform and locality traffic and proposes application-oriented traffic configuration for on-chip network evaluation.

Author's contributions: The author formulated the unified expression for the regular traffic patterns and defined application-oriented traffic, integrated the methods in NNSE, conducted experiments and wrote the manuscript.

Paper 6 [76]. Zhonghai Lu, Mingchen Zhong, and Axel Jantsch. Evaluation of on-chip networks using deflection routing. In *Proceedings of the 16th ACM Great Lakes Symposium on VLSI (GLSVLSI'06)*, pages 296-301, Philadelphia, USA, May 2006.

This paper evaluates the performance of deflection networks with different topologies such as *mesh*, *torus* and *Manhattan Street Network*, different routing algorithms such as *random*, *dimension XY*, *delta XY* and *minimum deflection*, as well as different deflection policies such as *non-priority*, *weighted priority* and *straight-through* policies. The results suggest that the performance of a deflection network is more sensitive to its topology than the other two parameters. It is less sensitive to its routing algorithm, but a routing algorithm should be minimal. A priority-based deflection policy that only uses global and history-related criterion can achieve both better average-case and worst-case performance than a non-priority or priority policy that uses local and stateless criterion. These findings may be used as guidelines by designers to make right decisions on the deflection network architecture.

Author's contributions: The author formulated the problem, proposed solution schemes, and wrote the manuscript. The implementation and experiments were conducted by Mingchen Zhong.

Paper 7 [69]. Zhonghai Lu, Axel Jantsch and Ingo Sander. Feasibility analysis of messages for on-chip networks using wormhole routing. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 960-964, Shanghai, China, January 2005.

The paper proposes a method for investigating the feasibility of delivering mixed real-time and nonreal-time messages in wormhole-switched networks. Particularly it describes a contention tree model for the estimation of worst-case performance for delivering real-time messages.

Author's contributions: The author formulated the contention tree model, developed the algorithm, wrote the program, performed experiments, and wrote the manuscript.

- **NoC Communication Refinement**

Paper 8 [71]. Zhonghai Lu, Ingo Sander, and Axel Jantsch. Refining synchronous communication onto network-on-chip best-effort services. In Alain Vachoux, editor, *Applications of Specification and Design Languages for SoCs - Selected papers from FDL 2005*. Springer, Chapter 2, pages 23-38, 2006.

The paper proposes a top-down design flow to refine synchronous communication onto NoC best-effort services. It consists of three steps, namely, *channel refinement*, *process refinement*, and *communication mapping*. In channel refinement, synchronous channels are replaced with stochastic channels abstracting the best-effort service. In process refinement, processes are refined in terms of interfaces and synchronization properties. Particularly, we use *synchronizers* to maintain local synchronization of processes and thus achieve *synchronization consistency*, which is a key requirement while mapping a synchronous model onto an asynchronous architecture. Within communication mapping, the refined processes and channels are mapped onto a NoC platform. A digital equalizer is used as a tutorial example and implemented in the *Nostrum* NoC platform to illustrate the feasibility of our concepts.

Author's contributions: The author proposed the design flow for the communication refinement, developed solutions for the synchronization problem, conducted the case study, and wrote the manuscript.

Paper 9 [72]. Zhonghai Lu, Ingo Sander, and Axel Jantsch. Towards performance-oriented pattern-based refinement of synchronous models onto NoC communication. In *Proceedings of the 9th Euromicro Conference on Digital System Design (DSD'06)*, pages 37-44, Dubrovnik, Croatia, August 2006.

This paper is complementary to Paper 8, which mainly discusses how to maintain synchronization consistency while refining the synchronous communication on asynchronous NoC architectures. This paper focuses on how to achieve performance-oriented refinement. Specifically, it deals with *protocol refinement* and *channel mapping* issues. In protocol refinement, we show how to refine communication towards application requirements such as reliability and throughput. In channel mapping, we discuss channel-convergence and channel-merge to make efficient use of shared network resources.

Author's contributions: The author developed the idea, implemented the proposed techniques, conducted experiments, and wrote the manuscript.

The remainder of the thesis is structured as follows. Chapter 2 summarizes our research results on NoC network architectures. In Chapter 3, we describe our work on NoC network performance analysis. We present our NoC communication refinement approach in Chapter 4. Finally we summarize the thesis in Chapter 5.

Chapter 2

NoC Network Architectures

This chapter summarizes our research on NoC network architectures, particularly, cost-effective switch architectures [Paper 1, 2], connection-oriented multicasting [Paper 3], as well as TDM (Time Division Multiplexing) virtual-circuit configuration [Paper 4].

2.1 Introduction

2.1.1 On-Chip Communication Network

A. On-chip network characteristics

As with macro- and tele-networks, on-chip micro-networks share the same characteristics in *topology*, *switching*, *routing*, and *flow control*. Additionally, a micro-network has to provide high and predictable performance with small area overhead and low power consumption. As noted in [27], a micro-network should appear as logical wires for network clients. Quality of Service (QoS) is thus a crucial aspect to distinguish one micro-network from another. Moreover, the design of on-chip systems should take advantage of well-validated legacy or third-party IP cores to shorten time-to-market and to guarantee product quality. To this end, IP reuse, exchange and integration are other critical issues. Addressing these issues demands a standardized hardware interface. The interface wrapping a micro-network can therefore be a distinguishing feature of a NoC proposal.

In the following, we describe the micro-network characteristics, namely, *topology*, *switching*, *routing*, *flow control*, *Quality of Service* and *Interface*, highlighting present NoC practices in these regards.

B. Topology

The *topology* refers to the physical structure of the network graph, i.e., how network nodes (switches or routers) are physically connected. It defines the connectivity (the routing possibility) between nodes, thus having a fundamental impact on the network performance as well as the switch structure, for example, the number of ports and port width. The tradeoff between generality and customization is an important issue when determining a network topology. The generality facilitates the re-usability and scalability of the communication platform. The customization is aimed for performance and resource optimality. Both regular and irregular topologies have been advocated for NoCs. Regular topologies such as k-ary 2-cube meshes [56] and tori [27] are popular ones because their layouts on a two-dimensional chip plane use symmetric-length of wires. The significance of the regularity lies in its potential of managing wire delay and wire-related DSM effects. The k-ary tree and k-ary n-dimensional fat tree [1] are two alternative regular NoC topologies. With a regular topology, the network area and power consumption scale predictably with the size of the topology. The arguments for using irregular topologies are that specific applications require flexible and optimal topology. In [123], the number of ports in switches can be synthesized according to the requirement of connectivity. However, the area and power consumption of an irregular network topology may not scale predictably with the topology size. Other topologies in between regular and irregular ones are also proposed for NoCs. For example, an interesting NoC topology is the Octagon NoC [52] in which a ring of 8 nodes connected by 12 bi-directional chords. Traveling between any pair of nodes takes at maximum two hops. In [99], a butterfly fat-tree topology was proposed in which IPs are placed at the leaves and switches placed at the vertexes. Moreover, regular topology may be customized by introducing application-specific long-range links to improve performance with a small area penalty [96].

C. Switching strategy

The *switching strategy* determines how a message traverses its route. There are two main switching strategies: *circuit switching* and *packet switching*. Circuit switching reserves a dedicated end-to-end path from the source to the destination before starting to transmit the data. The path can be a real or virtual circuit. After the transmission is done, the path reservation with associated resources is released. Circuit-switching is *connection-oriented*, meaning that there is an explicit connection establishment. In contrast to circuit-switching, packet-switching segments the message into a sequence of packets. A packet typically consists of a header, payload and a tail. The header carries the routing and sequencing information. The payload is the actual data to be transmitted. The tail is the end of the packet and usually contains error-checking code. Packet-switching can be either *connection-oriented* or *connection-less*. Connection-oriented communication preserves resources while connection-less communication does not. Connection-oriented communication can typically provide a certain degree of commitment for message delivery bounds. With connection-less communication, packets are routed individually in the network in a best-effort manner. The message delivery is subject to dynamic contention scenarios in the network, thus is difficult to provide bounds. However, the network resources can be better utilized. Typical packet switching techniques[1] include *store-and-forward*, *virtual cut-through* [53], and *wormhole switching* [2].

- *Store-and-forward*: A network node must receive an entire packet before forwarding it to the next downstream node. Both link bandwidth and buffers are allocated at the packet-level. The non-contentional latency T for transmitting L flits is expressed by Equation 2.1. Flit is the smallest unit for the link-level flow control, which is the minimum unit of information that can be transferred across a link.

  $$T = (L/BW + R) * H \tag{2.1}$$

  where BW is the link bandwidth in flits per cycle; R is the routing delay per hop; Hop is the basic communication action from switch to switch. H is the number of hops from the source node to the destination node.

- *Virtual cut-through*: Like store-and-forward, virtual cut-through allocates both link bandwidth and buffers in units of packets. However, in virtual cut-through, a network node does not wait for the reception of an entire packet. It receives a portion of the packet, and then forwards it downstream if the buffer space in the next switch is available. The downstream node must have enough buffers to hold the entire packet. In case of blocking, the entire packet is shunt into the buffers allocated. By transmitting packets as soon as possible, virtual cut-through reduces the non-contentional latency T for transmitting L flits to

  $$T = L/BW + R * H \tag{2.2}$$

[1]Both store-and-forward and virtual cut-through do not divide packets into flits. We show the division here for a consistent presentation of the switching techniques.

[2]In the literature, *wormhole switching*, *wormhole routing* and *wormhole flow control* have been used. In this thesis, we tend to use *wormhole switching*.

- *Wormhole switching*: A packet is decomposed into flits. Operating like virtual cut-through, wormhole switching delivers flits in a pipelined fashion. Due to the pipelined transmission, the non-contentional latency T of transmitting L flits is the same as that for virtual cut-through. Wormhole-switching and virtual cut-through are both *cut-through* switching techniques. They mainly differ in how they handle packet blocking. With wormhole switching, link bandwidth and buffers are allocated to flits rather than packets. The switch buffering capacity is a multiple of a flit. If a packet is blocked, flits of the packet are *stalled* in place. With virtual cut-through, a switch, at which a packet is blocked, must *receive* and *store* all flits of the blocked packet. This enforces that the buffering capacity in switches must be a multiple of a packet. Virtual cut-through utilizes the network's bandwidth more efficiently, achieving higher throughput than wormhole switching but requiring higher buffering capacity.

Circuit-switching for on-chip networks is proposed in [132] to satisfy applications with hard real-time constraints. The majority of on-chip networks is based on packet-switching, and combined packet-switching and circuit-switching. For example, TDM virtual-circuits [34, 81], which preserves time slots to switch packets in a contention-free manner, can be viewed as a circuit-switching technique implemented in a packet-switched network.

D. Routing algorithm

The *routing algorithm* determines the routing paths the packets may follow through the network graph. It usually restricts the set of possible paths to a smaller set of valid paths. In terms of path diversity and adaptivity, routing algorithm can be classified into three categories, namely, *deterministic routing*, *oblivious routing* and *adaptive routing* [28]. Deterministic routing chooses always the same path given the source node and the destination node. It ignores the network path diversity and is not sensitive to the network state. This may cause load imbalances in the network but it is simple and inexpensive to implement. Besides, it is often a simple way to provide the ordering of packets. Oblivious routing, which includes deterministic algorithms as a subset, considers all possible multiple paths from the source node to the destination node, for example, a random algorithm that uniformly distributes traffic across all of the paths. But oblivious algorithms do not take the network state into account when making the routing decisions. The third category is adaptive routing, which distributes traffic dynamically in response to the network state. The network state may include the status of a node or link, the length of queues, and historical network load information. A routing algorithm is termed *minimal* if it only routes packets along shortest paths to their destinations, i.e., every hop must reduce the distance to the destination. Otherwise, it is non-minimal. Both *table-based* and *algorithmic* routing mechanics can be used to realize the routing algorithms [28]. The table-based routing mechanism uses routing tables either at the source or at each hop along the route. Instead of storing the routing relation in a table, the algorithmic routing mechanism computes it. For speed, it is usually implemented as a combinational logic circuit. The algorithmic routing is usually restricted to simple routing algorithms and regular topologies, sacrificing the generality of table-based routing.

In comparison with adaptive routing, deterministic or oblivious minimal routing results in relatively simple switch designs because a routing decision is made independent of the dynamic network state. Though a routing algorithm has different properties in design complexity, adaptivity and load balancing, the performance of a routing algorithm is also topology and application dependent [88]. An interesting extreme case of non-minimal adaptive routing is *deflection routing* [16], also called *hot-potato* routing. Its distinguishing feature is that it does not buffer packets. Instead, packets are always on the run cycle-by-cycle. A deflection policy prioritizes packets on the use of favored links. If there is no contention, packets are delivered via shortest paths. Upon contending for shared links, packets with a higher priority win arbitration and use the favored links while packets with a lower priority are mis-routed to non-minimal routes. Deflection routing has been used in optical networks where buffering optical signals is too expensive [106]. Because of simplicity and adaptivity, it is adopted and implemented in communication networks embedded in massively parallel machines such as the Connection machine [42]. For the same reasons, it has also been proposed for on-chip networks in the Nostrum NoC [81, 91]. Using deflection routing results in faster and smaller switch designs. As projected in [91], a deflection switch with an arity of five can run 2.38 GHz with a gate count of 19370 in 65 nm technology. Deadlock and livelock are the primary concern when designing a routing algorithm in order to ensure correct network operation [30]. As shown in [97], application knowledge can be effectively utilized to avoid deadlock. In [16, 49], maximum delivery bounds are derived for deflection networks. Thus the networks are livelock free.

E. Network flow control

The *network flow control* governs how packets are forwarded in the network, concerning shared resource allocation and contention resolution. The shared resources are buffers and links (physical channels). Essentially a flow control mechanism deals with the coordination of sending and receiving packets for the correct delivery of packets. Due to limited buffers and link bandwidth, packets may be blocked due to contention. Whenever two or more packets attempt to use the same network resource (e.g., a link or buffer) at the same time, one of the packets could be stalled in placed, shunted into buffers, detoured to an unfavored link, or simply dropped. For packet-switched networks, there exist *bufferless flow control* and *buffered flow control* [28].

- *Bufferless flow control* is the simplest form of flow control. Since there is no buffering in switches, the resource to be allocated is link bandwidth. It relies on an arbitration to resolve contentions between contending packets. After the arbitration, the winning packet advances over the link. The other packets are either dropped or misrouted since there are no buffers. The deflection routing uses bufferless flow control. In fact, deflection routing includes an orthogonal concern of routing algorithm and deflection policy. While a routing algorithm determines the favored links for packets, a deflection policy resolves contentions for shared links by forwarding the packet with the highest priority to its favored link and misrouting other packet(s) with a lower priority to unfavored links. As deflection routing does not buffer packets, the switch design can be simpler and thus cheaper because it has no buffer and flow management. Moreover, since the routing paths of packets are fully adaptive to the network state, deflection routing has higher link utilization and offers the potential to allow resilience for link and switch faults.

- *Buffered flow control* stores blocked packets while they wait to acquire network resources. Store-and-forward, virtual cut-through and wormhole switching techniques adopt buffered flow control. The granularity of resource allocation for different buffered flow control techniques may be different. Store-and-forward switching and virtual cut-through switching allocate link bandwidth and buffers in units of packets. Wormhole switching allocates both link bandwidth and buffers in units of flits. Buffered flow control requires a means to communicate the availability of buffers at the downstream switches. The upstream switches can then determine when a buffer is available to hold the next flit to be transmitted. If all of the downstream buffers are full, the upstream switches must be informed to stop transmitting (assuming drop-less delivery). This phenomenon is called *back pressure*. Link-level flow control mechanisms, in which the buffer availability information is passed and propagated between switches, are introduced to provide such back-pressure. Today, there are three types of link-level flow control techniques in common use: credit-based, on/off, and ack/nack [28].

The flow control scheme of a network may be coupled with its switching strategy. For instance, both store-and-forward and virtual cut-through switching use the packet-buffer flow control, and wormhole switching uses the flit-buffer flow control. It is worthwhile to discuss them separately because a flow control scheme emphasizes the movement of packet flows instead of switching individual packets.

F. Quality of Service

Generally speaking, Quality-of-Service (QoS) defines the level of commitment for packet delivery. Such a commitment can be correctness of the result, completion of the transaction, and bounds on the performance [33]. But, mostly, QoS has a direct association with bounds in bandwidth, delay and jitter, since correctness and completion are often the basic requirements for on-chip message delivery. Correctness is concerned with packet integrity (corrupt-less) and packet ordering. It can be achieved through different means at different levels. For example, error-correction at the link layer or re-transmission at the upper layers can be used to ensure packet integrity. A network-layer service may secure that the packets are delivered in order. Alternatively, if a network-layer service cannot promise in-order delivery, a transport-layer service may compensate to do the re-ordering. Completion requires that a flow control method does not drop packets. In case of a shortage of resources, packets can be mis-routed or buffered. In addition, the network must ensure deadlock and livelock freedom.

Roughly classified, NoC researchers have proposed *best-effort*, *guaranteed*, and *differentiated* services for on-chip packet-switched communication. A best-effort service is connectionless. The network delivers packets as fast as it can. Packets are routed in the network, resulting in dynamic contentions for shared buffers and links. A packet-admission policy is usually desired to avoid network saturation. Below the saturation point, the network exhibits good average performance but the worst-case can be more than an order of magnitude worse than the average case. A guaranteed service is typically connection-oriented. It avoids network contentions by establishing a virtual circuit. Such a virtual circuit may be implemented by time slots, virtual channels, parallel switch fabrics and so on. The Æethereal NoC [34] implements a contention-free TDM virtual-circuit in a network employing buffered flow control. The Nostrum NoC [81] also realizes TDM virtual-circuit but in a network using bufferless flow control. Both Æethereal and Nostrum networks operate synchronously. The MANGO network [14] is clockless.

Since the network switches do not share the same notion of time, it uses sequences of virtual channels to set up virtual end-to-end connections. In contrast to TDM, SDM (Space-Division-Multiplexing)-based QoS is achieved by allocating individual wires on the link for different connections [61]. For the guaranteed services, if a virtual circuit is set up dynamically, the setup procedure has to use best-effort packets. This phase is somewhat unpredictable due to the best-effort nature. A differentiated service prioritizes traffic according to different categories, and the network switches employ priority-based scheduling and allocation policies. For instance, the QNoC [15] network distinguishes four traffic classes, i.e., signaling, real-time traffic, read-write and block transfer. The signaling class has the highest and the block-transfer class the lowest priority. Priority-based approaches allow for higher utilization of resources but cannot provide strong guarantees like guaranteed services. To improve resource usage, a best-effort service may be mixed with a guaranteed service using, for example, slack-time aware routing [5].

G. Interface

Wrapping on-chip networks with an interface is essential for NoC designs. The interface is preferably standardized, but domain-specific customization is necessary for optimal and dedicated solutions. An interface-based design approach [112, 129] separates computation from communication. It gives the interface users an abstraction that makes only the relevant information visible. It facilitates the exchange and reuse of IPs as long as the IPs conform to the same interface.

To make a huge number of legacy IPs reusable and integrable, an on-chip network interface has to follow standard interfaces. Current network interconnects implement interfaces such as AXI [6], OCP [95], VCI [130] and DTL [104]. AXI (Advanced eXtensible Interface) is AMBA's highest performance interface developed by ARM to support ARM11 processors. The configurable AXI interconnection is optimized for the processor-memory backplane and has advanced features such as split transactions (address and data buses are decoupled), multiple outstanding transactions, and out-of-order data. OCP (Open Core Protocol) is a plug-and-play interface for a core having both master and slave interfaces. The OCP specification defines a flexible family of memory-mapped, core-centric protocols for use as a native core interface in on-chip systems. OCP addresses both data-flow signaling and side-band control-flow signaling for error, interrupt, flag, status and test. The VCI (Virtual Component Interface) specification includes three variants: PVCI (peripheral), BVCI (basic) and AVCI (advanced). The DTL (Device Transaction Level) interconnection interface is developed by Phillips Semiconductors to interface existing SoC IPs. It allows easy extension to other future interconnection standards. The Æethereal NoC [113] provides a shared-memory abstraction to the cores and is compatible to standard interfaces such as AXI, DTL and OCP. The SPIN [37] and Proteo [122] NoCs support the VCI interface. The OCP interface is used in the MANGO NoC [14]. Nonetheless, the cost of adopting standard socket-type interfaces is nontrivial. The HERMES NoC [84] demonstrates that the introduction of OCP makes the transactions up to 50% slower than the native core interface. Therefore domain-specific interfaces will be an option for optimization.

Next, in Section 2.2, we investigate the design complexity of a canonical wormhole switch from the perspective of admitting and ejecting flits, proposing the *coupled admission* model for flit admission (Paper 1) and the p-*sink* model for flit ejection (Paper 2). Section 2.3 suggests a multicasting service (Paper 3). In Section 2.4, we discuss the construction of TDM virtual-circuits using logical networks (Paper 4). Since both Section 2.2 and Section 2.3 consider wormhole switching, we introduce wormhole switching further in the next subsection.

2.1.2 Wormhole Switching

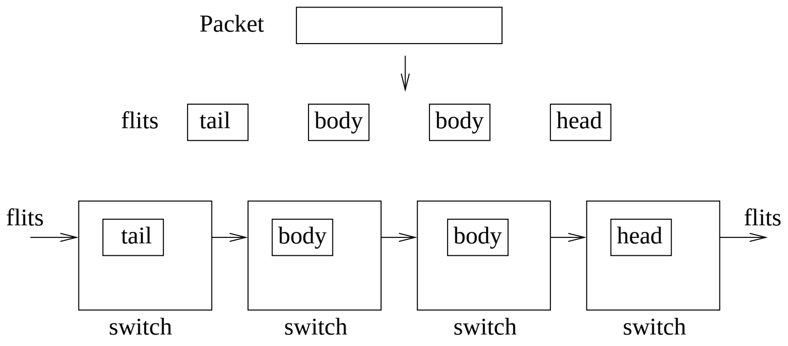


Figure 2.1. Flits delivered in a pipeline

Wormhole switching [26] allocates buffers and physical channels (PCs, links) to flits instead of packets. A packet is decomposed into a head flit, body flit(s), and a tail flit. A single-packet flit is also possible. We call this decomposition *flitization*. Flitization is named following packetization, i.e., encapsulate a message into one or more packets. A flit, the smallest unit on which flow control is performed, can advance once buffering in the next switch is available to hold the flit. This results in that the flits of a packet are delivered in a pipeline fashion. As illustrated in Figure

2.1, a packet is segmented into four flits, with one head flit leading two body flits and one tail flit, and then the four flits are transmitted in a pipeline via switches. For the same amount of storage, it achieves lower latency and greater throughput.

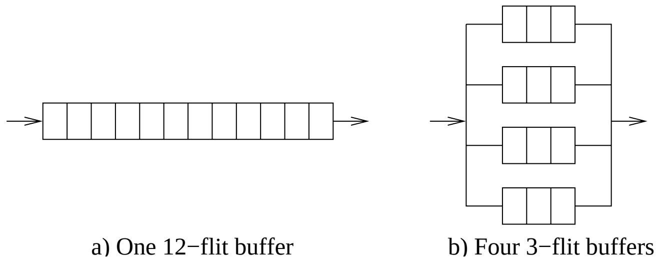


Figure 2.2. Virtual channels (lanes)

However, wormhole switching uses physical channels (PCs) inefficiently because a PC is held for the duration of a packet. If a packet is blocked, all PCs held by this packet are left idle. To mitigate this problem, wormhole switching adopts *virtual channels* (*lanes*) to make efficient use of the PCs [25]. Several parallel lanes, each of which is a flit buffer queue, share a PC (Figure 2.2). Therefore, if a packet is blocked, other packets can still traverse the PC via other lanes, leading to higher throughput. Because of these advantages, namely, *better performance*, *smaller buffering requirement* and *greater throughput*, wormhole switching with virtual-channel flow control is the prevailing switching scheme advocated for on-chip networks [1, 24, 44, 110]. In addition, virtual-channel has a versatile use in optimizing link utilization, improving throughput, avoiding deadlock [26], increasing fault tolerance [18] and providing guaranteed services [14]. Nonetheless, in order to maximize its utilization, the procedure to allocate virtual channels is critical in designing routing algorithms [128].

Note that wormhole switching is not without problems. First, it incurs flit-type overhead to distinguish head, body, tail, and single-packet flits. Second, the flits of a packet may be distributed in flit buffers of multiple switches. The intermediate buffers between a head flit and a tail flit may be under-utilized, resulting in lower buffer utilization [120]. Third, due to the flit distribution, wormhole switching is more prone to deadlock.

2.2 Flit Admission and Ejection

This section summarizes the research in Paper 1 (Flit admission) and Paper 2 (Flit ejection).

2.2.1 Problem Description

Despite the aforementioned advantages, using wormhole switching for on-chip networks has to minimize the switch design complexity. First, since an on-chip network is an interconnect using shared wires instead of dedicated wires to pass signals, its cost must be reasonable [27]. Second, embedded applications often have very stringent requirements on power. For future complex SoC integration, the communication bandwidth is achievable but the energy consumption will probably be the bottleneck that has to trade off the performance [98]. Therefore, in order to shrink energy dissipation, it is important to reduce the switch design complexity so as to decrease the number of gates and switching capacitance.

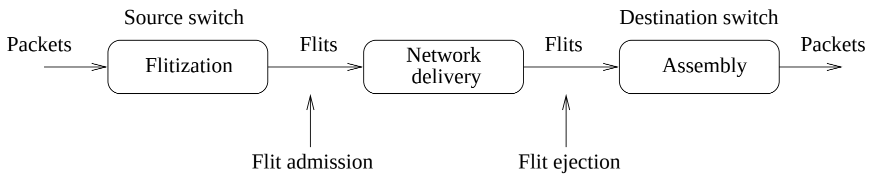


Figure 2.3. Flit admission and ejection

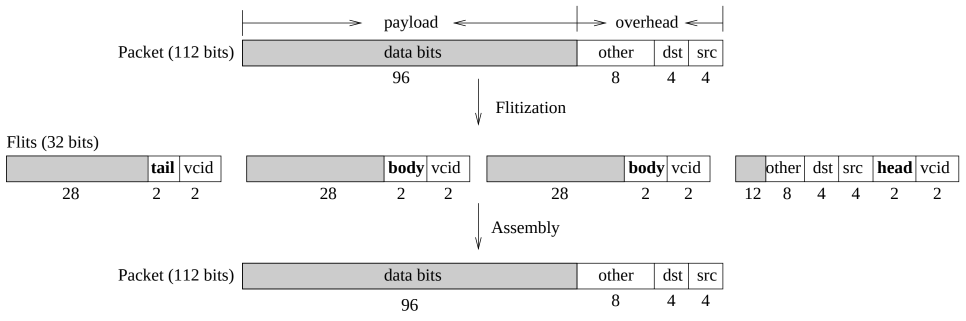


Figure 2.4. Flitization and assembly

We examine the problem of flit-admission and flit-ejection in a wormhole-switched network. As depicted in Figure 2.3, the delivery of packets passes through three stages: *flitization*, *network delivery*, and *assembly*. The flitization is performed at a source node, and the assembly, which decapsulates flits into packets, is conducted by a destination node. Figure 2.4 illustrates the flitization and assembly of a packet. As can be seen, the packet is encapsulated into four flits (one head flit, two body flits and one tail flit), where vcid is the identity number of the lane the flit occupies. We assume 4 lanes per port in a switch, thus vcid takes 2 bits.

Since flits are both the workload of switches and the source of network contentions for shared Virtual Channels (VCs)[3] and Physical Channels (PCs, links), their admission and ejection are as important as delivery. As the transmission time of a flit comprises admission time, delivery time plus ejection time, the network performance is the function of *flit-admission*, *flit-delivery* and *flit-ejection*. Intuitively, to achieve good network utilization and throughput, flits should be admitted as fast as possible. However, flits to be advanced (after admission) may contend not only with each other, but also with flits to be admitted. Flit-admission and flit-delivery interfere with each other. This implies that a fast admission mechanism may speed up the admission but slow down the delivery. If the network is too loaded, the overall transmission time may get worse. For the ejection process, a faster ejection frees flit buffers quicker, thus the faster the better. A slower ejection of flits may slow down the flit delivery and eventually affect the flit delivery and admission through back-pressure. However, an ideal ejection, which ejects flits immediately once they reach destinations, may over-design the switch. Finally the interplay between flit-admission and flit-ejection influences the tradeoff between performance and switch complexity. A practical cost-effective ejection model may actually tolerate a slower but simpler admission model with reasonable performance penalty.

In the rest of this section, we first explain the operation of a canonical wormhole lane switch in Section 2.2.2. Then we discuss flit-admission and flit-ejection models in Section 2.2.3 and Section 2.2.4, respectively. Particularly, we introduce the *coupled* admission and the p-*sink* model.

2.2.2 The Wormhole Switch Architecture

Figure 2.5 illustrates a canonical wormhole switch architecture with virtual channels at input ports [25, 102, 110]. It has p physical channels (PCs) and v lanes per PC. It employs the credit-based link-level flow control to coordinate packet delivery between switches.

A packet passes the switch through four states: *routing*, *lane allocation*, *flit scheduling*, and *switch arbitration*. In the routing state, the routing logic determines the routing path the packet advances over. Routing is only performed with the head flit of a packet and only when the head flit becomes the earliest-come flit in the FIFO lane. After routing, the packet path and output PC are determined. In the state of lane allocation, the lane allocator *associates* the lane the packet occupies with an available lane in the next switch on its routing path, i.e., to make a

[3]In Section 2.2 and Section 2.3, we use the shorthand *VC* for *Virtual Channel*.

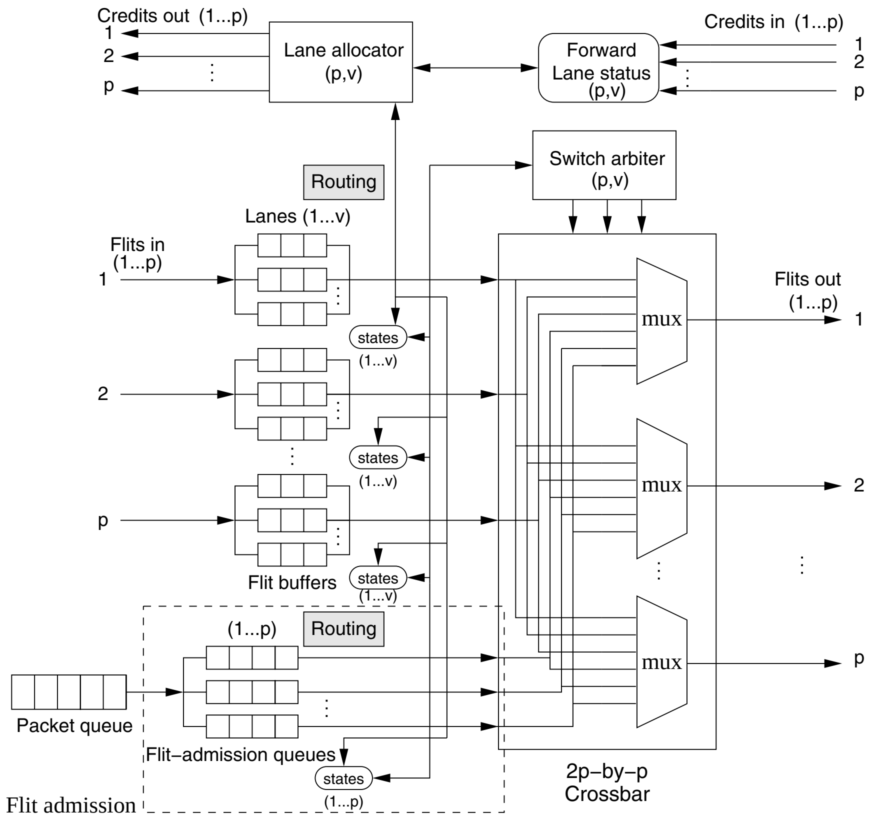


Figure 2.5. A canonical wormhole lane switch (ejection not shown)

lane-to-lane association. A lane is available if it is not currently being allocated to an upstream lane. A lane-to-lane association fails when all requested lanes are already associated to other lanes in directly connected switches, or the lane loses arbitration in case multiple lanes in the switch try to associate with the same downstream lane. Note that it is not necessarily required here that there is an empty buffer in the lane in order to make a successful association. If the lane-to-lane association succeeds, the flit vcid is determined and the packet enters the scheduling state. If there is a buffer available in the associated downstream lane, the lane enters the state of switch arbitration. This can be done with a two-level arbitration scheme. The first level of arbitration is performed on the lanes sharing the same physical channel. The second level of arbitration is for the crossbar traversal. If the lane wins the two levels of arbitration, the flit situated at the head of the lane is switched out. Otherwise, the lane stays in the arbitration state. The lane-to-lane association is released after the tail flit is switched out. Then the allocated lane is available to be reused by other packets. Credits are passed between adjacent switches in order to keep track of the status of lanes, such as if a lane is free and a count of available buffers in the lane.

A flit differs from a packet in that (1) a flit has a smaller size; (2) only the head flit carries the routing information such as source/destination address, packet size, priority etc. As a consequence, the routing and lane allocation can only be performed with the head flit of a packet. Once a lane-to-lane association is established by the head flit of the packet, the rest of flits of the packet inherit this association. After the tail flit leaves, the lane-to-lane association is released. Thus, a lane is allocated at the packet level, i.e., *packet-by-packet* while a link is scheduled at the flit level, i.e., *flit-by-flit* since the flit scheduling as well as the switch arbitration is performed per flit. As the head flit advances, lanes are associated like a chain along the routing path of the packet, the rest of flits are pipelined along the chain path. Carrying routing information only in the head flit of a packet leaves more space for payload. However, flits belonging to *different* packets can not be interleaved in associated lane(s) since only head flits contain routing information. To guarantee this, a lane-to-lane association must be *one-to-one*, i.e., *unique* at a time.

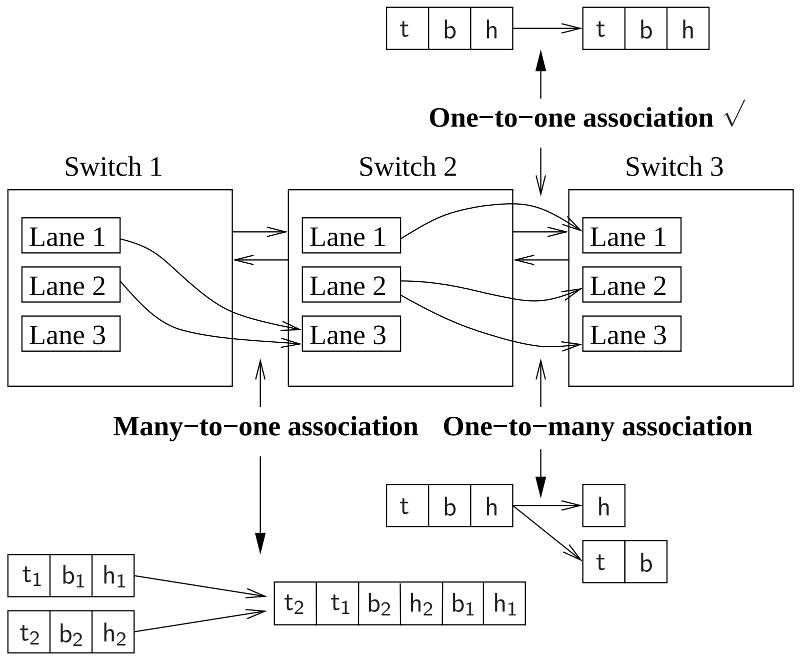


Figure 2.6. Lane-to-lane associations

Figure 2.6 illustrates lane-to-lane associations. The one-to-many association leads to that the flits from lane 2 in switch 2 are delivered to lane 2 and lane 3 in switch 3. The many-to-one association results in that lane 3 in switch 2 will receive flits from lane 1 and lane 2 from switch 1. Obviously the one-to-many and many-to-one associations result in that the integrity of a worm (the flit sequence of a packet) is destroyed. It becomes impossible either to route the flits of a packet or assemble the flits into a packet. Therefore, both one-to-many and many-to-one associations must be forbidden, and only one-to-one association is permissible.

2.2.3 Flit Admission

A. The decoupled admission

We assume that a switch receives packets injected via a packet FIFO. A packet is first flitized into flits that are then stored in flit FIFOs, called *flit-admission queues*, before being admitted into the network. There are various ways of organizing the packet queue and the flit-admission queues. In Figure 2.7(a), flit-admission queues are organized as a FIFO. In Figures 2.7(b) and 2.7(c), they are arranged as p parallel FIFO queues (p is the number of PCs). Figures 2.7(a) and 2.7(b) permit at maximum one flit to be admitted to the network at a time while Figure 2.7(c) allows up to p flits to be admitted simultaneously. We adopt the organization of flit-admission queues in Figure 2.7(c) for our further discussions since it allows potentially higher performance while the other two may lead to under-utilize the network.

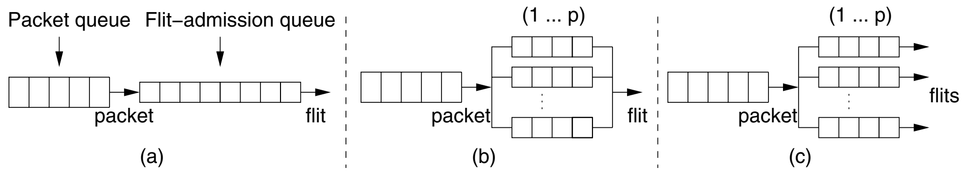


Figure 2.7. Organization of packet- and flit-admission queues

The organization of p flit-admission queues is also illustrated in the switch architecture in Figure 2.5. Initially, packets are stored in the packet queue. When a flit-admission queue is available, a packet is split into flits which are then put into an admission queue. Similarly to a lane, a flit-admission queue transits states to inject flits into the network via the crossbar. Note that flits to be admitted (in admission) contend with flits already admitted (in delivery) for VCs in the lane-to-lane association state and PCs in the *crossbar arbitration* state. This interference makes a flit-admission model nontrivial. Our study shows that when the network is nearly saturated, a faster admission model actually begins to slow down the delivery, worsening the network performance. Furthermore, *the routing is performed after flitization*. By this scheme, each flit-admission queue is connected to p multiplexers. Flits from a flit-admission queue can be switched to anyone of the p output PCs. To implement this scheme, the crossbar must be fully connected, resulting in a port size of $2p \times p$. Since the flit-admission queues are decoupled from the output PCs, we call this flit-admission scheme *decoupled admission*.

B. The coupled admission

Although the decoupled admission allows a flit to be switched to anyone of the p output ports, this may not be necessary since a flit is aimed to one and only one port after routing. Based on this observation, we propose a coupling scheme that can sharply decrease the crossbar complexity, as sketched in Figure 2.8. Just like the decoupled admission, it uses p flit-admission queues, but one queue is bound to one and only one multiplexer dedicated for a particular output PC. Due to this coupling, flits from a flit-admission queue are dedicated to the output PC. Consequently, an admission queue only needs to be connected to one multiplexer instead of p multiplexers. The size of the crossbar is sharply decreased from $2p \times p$ to $(p+1) \times p$, as shown in Figure 2.8. The number of control signals per multiplexer is reduced from $\lceil log(2p) \rceil$ to $\lceil log(p+1) \rceil$ for any $p > 1$ [4].

In order to support the coupling scheme, *the routing must be performed before flitization* instead. By a routing algorithm, the output physical channel that a packet requests can be determined. Hence, the corresponding admission queue is identified. One drawback due to the coupling is that the head-of-line blocking may be worse if the packet injection rate is high. Specifically, if the head packet in the packet queue is blocked due to the bounded number and size of the flit-admission queues, the packets behind the head packet are all unconditionally blocked during the head packet's blocking time. In the decoupled admission, the head-of-line blocking occurs when the four flit-admission queues are fully occupied. With the coupled admission, this blocking occurs when the flit-admission queue, which the present packet targets, is full.

As the crossbar is a power-hungry component in a switch [131], the coupled admission saves also power in comparison with the decoupled admission due to the reduction in the gate count and switching capacitance. The study on the power assumption of the flit admission schemes [74] shows that the coupled admission

[4] $\lceil x \rceil$ is the ceiling function which returns the least integer that is not less than x.

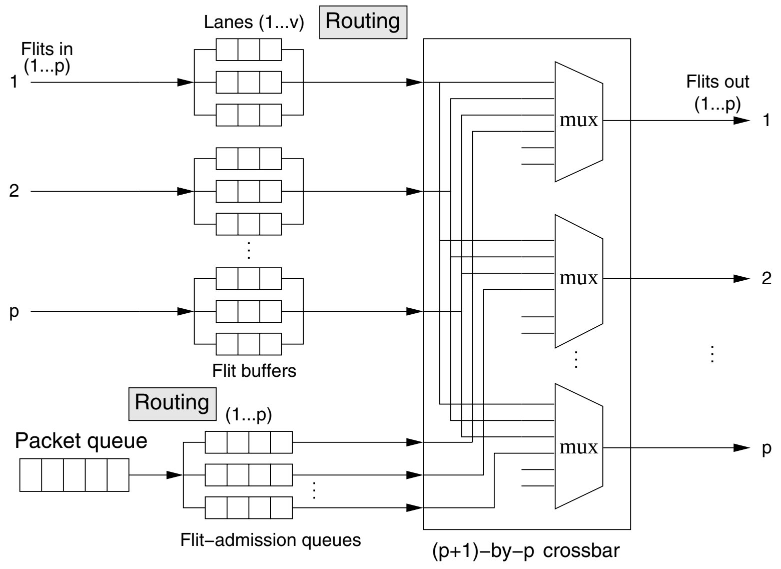


Figure 2.8. The coupled admission sharing a (p+1)-by-p crossbar

decreases the switch power by about 12% on average with the uniform traffic with random-bit payloads.

2.2.4 Flit Ejection

A. The ideal sink model

An ideal sink model is typically assumed for a wormhole lane switch. With such a model, flits reaching their destinations are ejected form the network immediately, emptying the lane buffers they occupy. An ideal flit-ejection model is drawn in Figure 2.9. A *flit sink* is a FIFO receiving the ejected flits. Each lane is connected to a sink and the crossbar (for packet forwarding) via a de-multiplexer.

To incorporate ejection, the lane state is extended with a *reception* state in addition to the four states. If the routing determines that the head flit of a packet reaches its destination, the lane enters the reception state immediately by establishing a *lane-to-sink* association. Since flits from different packets can not interleave in a sink queue, there must be $p \cdot v$ sink queues, each of them corresponding to a lane, in order to realize an immediate transition to the reception state. Assuming that one sink takes the flits of one packet, the depth of a sink is the maximum number of flits of a packet. After the lane transits to the reception state, the head flit bypasses the crossbar and enters its sink. The subsequent flits of the packet are

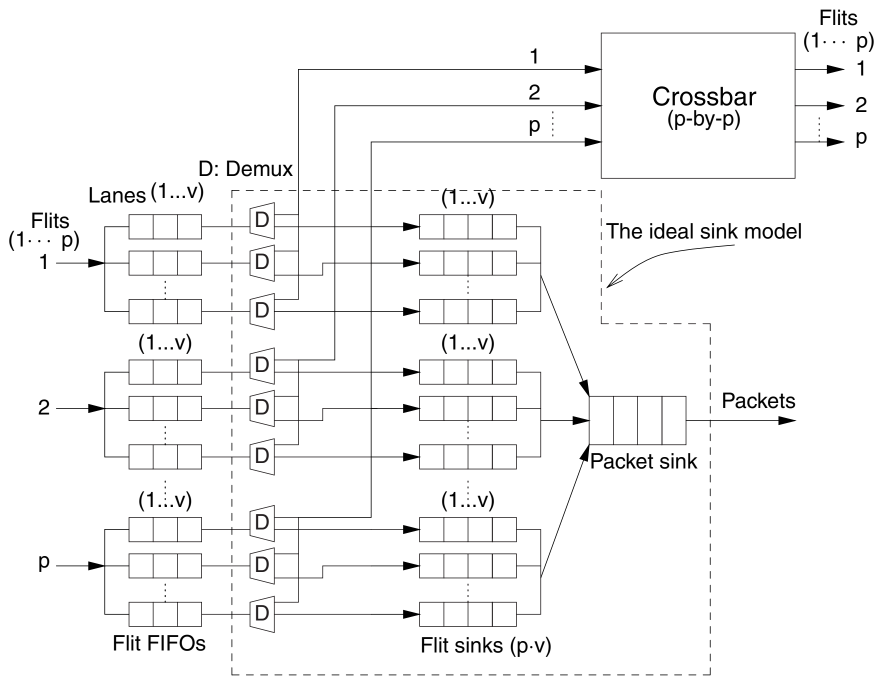


Figure 2.9. The ideal sink model

ejected into the sink immediately upon arriving at the switch. When the tail flit is ejected, the lane-to-sink association is freed. This model is beneficial in both time and space. Although a head flit may be blocked by flits situated in front of it in the same lane, a non-head flit neither waits to be ejected (time) nor occupies a flit buffer (space) once the lane is in the reception state. Moreover, it does not interfere with flits buffered in other lanes from advancing to next switches downstream (because the v demultiplexers of a PC share one input of the crossbar, one PC allows one flit from a lane to be switched via the crossbar without interference with sinking flits from other lanes.). Upon receiving all the flits of a packet, the packet is composed and delivered into the packet sink. If the packet sink is not empty, the switch outputs one packet per cycle from it in a FIFO manner.

B. The p-sink model

Implementing the ideal sink model requires $p \cdot v$ flit sinks, which can eject $p \cdot v$ flits per cycle. This may over-design the switch since there are only p input ports, implying that at maximum p flits can reach the switch per cycle. Since the maximum number of flits to be ejected per switch per cycle is p, we can use p sink queues instead of $p \cdot v$ sink queues to eject flits to avoid over-design. Moreover, in order to have a more structured design, we could connect the p sink queues to the crossbar, as illustrated in the dashed box of Figure 2.10.

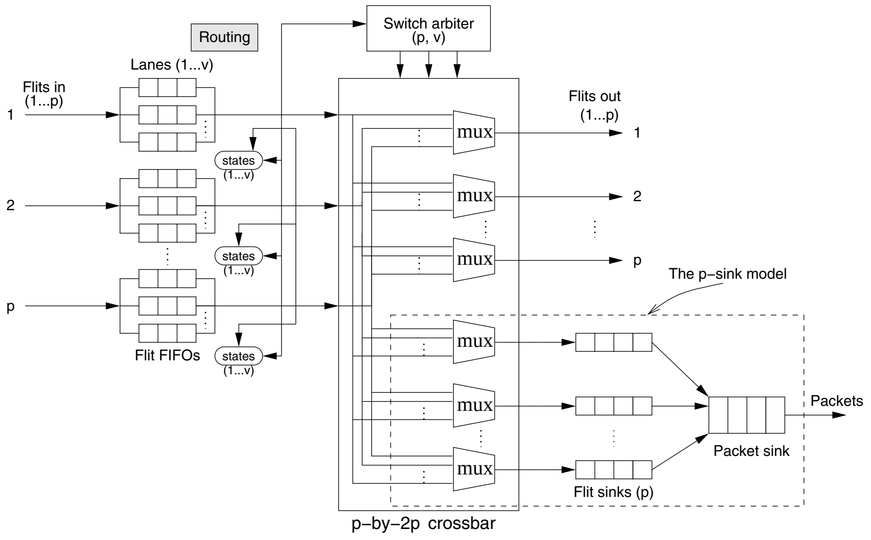


Figure 2.10. The p-sink model

To enable ejecting flits in the p-sink model, we now extend the four lane states with two new states: an *arriving* and a *reception* state. If a head flit reaches its destination, the lane the flit occupies transits from the routing to the arriving state. Then it will try to associate with an available sink, i.e., to establish a *lane-to-sink* association. If the association is successful, the lane enters the *reception* state. Subsequently the other flits of the packet follow this association exactly like flits advancing in the network. Upon the tail flit entering the sink, the association is torn down. If the lane-to-sink association fails (when all sinks have already been allocated), the head flit is blocked in place holding the lane buffer. To speed up flit ejection, the contentions for the crossbar input channels and crossbar traversal are arbitrated on priority. A lane in a reception state has a higher priority than a lane in a state for forwarding flits. The drawback in this sink model is the increase of blocking time when flits reach their destinations. First, the lane-to-sink association may fail since all sink queues might be in use. In contrast, an ideal sink model guarantees an exclusive sink for each lane. Second, only one lane per PC can win arbitration to an input channel of the crossbar due to sharing the input channel for both advancing flits and ejecting flits. In case of more than one lane of a PC are in the reception state, only one lane can use the channel.

	$p \times 1$ **Mux**	1×2 **Demux**	**Flit sink**
The ideal model	-	$p \cdot v$	$p \cdot v$
The p-sink model	p	-	p

Table 2.1. Cost of the sink models

To implement this p-sink model, the crossbar must double its capacity from p-*by*-p (p $p \times 1$ multiplexers) to p-*by*-$2p$ ($2p$ $p \times 1$ multiplexers), as illustrated in Figure 2.10. The number of control ports of the crossbar is doubled proportionally. Table 2.1 summarizes the number of each component to implement the sink models. As can be seen, the ideal sink model requires $p \cdot v$ flit sinks while the p-sink model uses only p flit sinks. With the p-sink model, the number of flit-sinks becomes independent of v, implying that the buffering cost for flit sinks is only $1/v$ as much as the ideal ejection model.

2.3 Connection-oriented Multicasting

This section summarizes the research in Paper 3.

2.3.1 Problem Description

As discussed previously, a bus and its variants (segmented, cross-bar and hierarchical buses) do not scale well with the system size in bandwidth and clocking frequency. However, a bus is very efficient in broadcasting since all clients are directly connected to it. A unicast is in fact broadcasted to all clients in the bus segment. In a NoC, IP blocks are distributed and communicate through multi-hop connections. This allows many more concurrent transactions, but does not directly support multicast. In NoC systems, it often desires to maintain a consistent view on the system state among the distributed cores, for example, in the case of implementing cache coherency protocols, of passing global states for barrier synchronization, and of managing and configuring the network. These communication patterns involve one source but multiple recipients. This type of pattern distinguishes from one-to-one communication in that the same message from one source has to be transmitted to multiple destinations. Particularly, real-time constrained, throughput-oriented embedded applications for multi-media processing exhibit such patterns, for instance, forking one data stream into multiple identical streams to be processed by multiple processing elements. Providing an efficient support for such one-to-many communication patterns is desirable.

Implementing multicast by sending multiple unicast messages is intuitive but neither efficient nor scalable because of excessive link and buffer consumption. Secondly, these messages are delivered in a best-effort manner without QoS. Our purpose is to provide an efficient multicast support from the network layer. We have taken a connection-oriented approach in aware of QoS. This allows dynamic multicast setup and release, thus consuming resources only if necessary. Our multicast scheme is realized in wormhole-switched networks. The resulting wormhole switch supports both unicast and multicast.

2.3.2 The Multicasting Mechanism

Our multicasting mechanism consists of three phases: *group setup*, *data transmission*, and *group release*. It is connection-oriented, meaning that a multicast connection must be established before one-to-many data transmission can start. The member visiting order of a multicast group is computed off-line and the multicast path is set up conforming to the unicast routing algorithm in the group setup phase. After data transmission, a multicast connection has to be explicitly released. A multicast connection means that

- There is a group master who owns the connection. The group master is the source node who has the group member information and determines the member visiting sequence. It initiates the establishment by sending a multicast setup packet using unicast. The last node in the sequence is responsible for acknowledging the establishment. In case of setup failure, a negative acknowledgment is sent from the node where the failure occurs. After data transmission, the group master sends a multicast release packet to release the connection.

- A simplex path is defined from the group master to the last member, passing other member nodes. Data transmission will deterministically follow this path from upstream to downstream. In addition to the group master, any upstream node is allowed to send multicast packets downstream.

- Each switch along the multicast delivery path has stored information about how to deal with a multicast packet (*copy*, *forward* or *sink*) and about the connection status. The copy means that the multicast packet has to be forwarded besides being locally sunk. The record of a multicast connection includes $\{\mathsf{MultiID}, \mathsf{GroupType}, \mathsf{Sadr}, \mathsf{VCID}, \mathsf{VCID\ downstream}, \mathsf{Output\ PC}, \mathsf{Next\ member\ addr.}\}$, where MultiID is the multicast group identity number, which is unique for each multicast group; GroupType is the type of the multicast group which informs the switches whether the multicast group requires reserving a lane or not; Sadr is the group master address; VCID is the identity number of the lane that the multicast packets use in the current switch; VCID downstream is the identity number of the lane allocated in the next downstream switch; OutputPC is the output physical channel over which the multicast packets are to be switched; Next member address is the address of the next member in the multicast group.

During the setup phase, multicasting can be aware of QoS in the sense that a multicast group may request to *reserve a lane*. The GroupType indicates if the group reserves a lane or not. To speed up multicasting, multicast packets enjoy higher priority than unicast packets for link bandwidth arbitration. After a connection is established, a multicast is realized by sending a single copy of multicast packets to multicast group members along the pre-established path. Multicast packets carry multiID in their headers. This results in low packet overhead and efficient use of link bandwidth. The drawback is the setup and release overhead.

The multicasting protocol is designed seamlessly with the unicasting protocol. The unicast packet format is extended to include different types of packets. In the implementation, the controller of the unicast switch is enriched to identify and act according to the different packet types. The data path of the switch remains the same. In this way, the resulting wormhole switch supports both unicast and multicast. The network allows mixed unicast and multicast traffic.

2.4 TDM Virtual-Circuit Configuration

This section summarizes the research in Paper 4.

2.4.1 Problem Description

A Virtual Circuit (VC)[5] is a set of pre-allocated resources to enable performance guarantees. Since the pre-allocation involves a setup phase, a VC is connection-oriented. A TDM (Time-Division Multiplexing) VC [34, 81] is a VC that shares

[5]In Sections 2.4 and 2.5, we use the shorthand *VC* for *Virtual Circuit*.

buffers and link bandwidth in a time-division fashion. Each node along the VC's path is equipped with a time-sliced routing table which reserves time slots for input packets to use output links. The routing table partitions link bandwidth and avoids the simultaneous use of shared links. A VC is simplex. In general, it may comprise multiple source and destination nodes (multi-node). As long as a VC is established, packets delivered on it, called *VC packets*, encounter no contention and thus have guarantees in latency and bandwidth. Unlike connection-less Best-Effort (BE) packet delivery that starts as soon as possible, VC packet delivery can not start until the VC is successfully set up. Therefore, VC configuration is an indispensable process. Moreover, well-planned VC configurations can make a better use of network resources and achieve better performance.

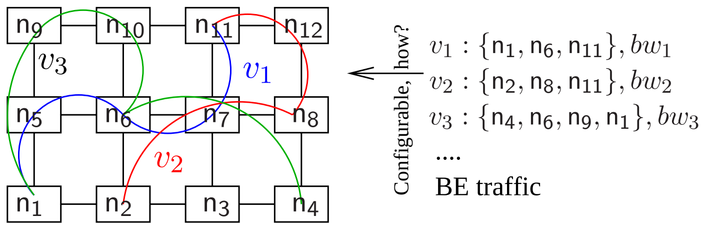


Figure 2.11. The virtual-circuit configuration problem

Figure 2.11 illustrates the multi-node VC configuration problem. It shows a partial mesh network and a specification of three VCs, v_1, v_2 and v_3, to be configured in the network. The network also delivers BE traffic. Each VC comprises multiple source and destination nodes and is associated with a bandwidth requirement. Configuring VCs involves (1) *path selection*: This has to explore the network path diversity. It turns out that there exists a huge design space to explore. Suppose that the size of a VC specification set is m, each VC has p alternative paths, we have p^m solution possibilities; (2) *slot allocation*: Since VC packets can not contend with each other, VCs must be configured so that an output link of a switch is allocated to one VC per slot. Both steps together must ensure that VCs are set up free from contention and allocated with sufficient slots. The network must be deadlock-free and livelock-free.

2.4.2 Logical-Network-oriented VC Configuration

A. TDM virtual circuits

Figure 2.12 shows two VCs, v_1 and v_2, and the respective routing tables for the switches. An output link is associated with a buffer or register. v_1 passes switches

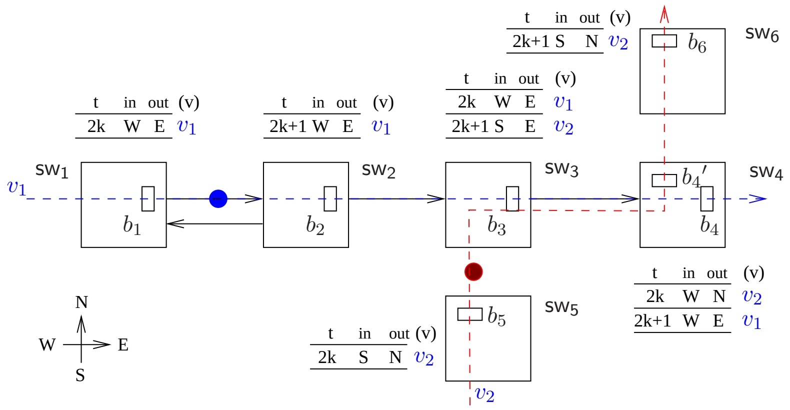


Figure 2.12. TDM virtual circuits

sw$_1$, sw$_2$, sw$_3$ and sw$_4$ through $\{b_1 \to b_2 \to b_3 \to b_4\}$; v_2 passes switches sw$_5$, sw$_3$, sw$_4$ and sw$_6$ through $\{b_5 \to b_3 \to b_4' \to b_6\}$. v_1 and v_2 only overlap in buffer b_3, i.e., $v_1 \cap v_2 = \{b_3\}$. A routing table entry (t, in, out) is equivalent to a routing function $\mathcal{R}(t, in) = out$, where t is time slot, in an input link, and out an output link. For example, $(2k, W, E)$ in sw$_1$ means that sw$_1$ reserves its E (East) output link at slots $2k$ ($k \in \mathbb{N}$) for its W (West) inport ($\mathcal{R}(2k, W) = E$). As can also be observed, sw$_3$ configures its even slots $2k$ for v_1 and its odd slots $2k+1$ for v_2. As v_1 and v_2 interleavely use the shared buffer b_3 and its associated output link, v_1 and v_2 do not conflict.

B. Using logical networks to avoid conflict

We draw a simplified picture of Figure 2.12 in Figure 2.13, where a bubble represents a buffer. VC packets on v_1 and v_2 are fired once every two cycles. Their bandwidth is $bw_1 = bw_2 = 1/2$ packets/cycle. Suppose that both VCs start admitting packets at slot $t = 0$. v_1 packets visit the shared buffer b_3 at even slots with an initial latency of two slots; v_2 packets visit b_3 at odd slots with an initial latency of one slot. This also means that, at even slots, v_1 packets hold buffers b_1 and b_3 while v_2 packets hold buffers b_5 and b_4'; At odd slots, v_1 packets hold buffers b_2 and b_4 while v_2 packets hold buffers b_3 and b_6. Thus v_1 and v_2 never conflict with each other. Figure 2.13 shows a snapshot of VC packets at even slots.

From the local perspective of buffer b_3, the alternate use of this shared buffer by v_1 and v_2 virtually partitions its time slots into two disjoint sets, the odd set and the even set. The two sets can be regularly mapped to the slot sets of other buffers on

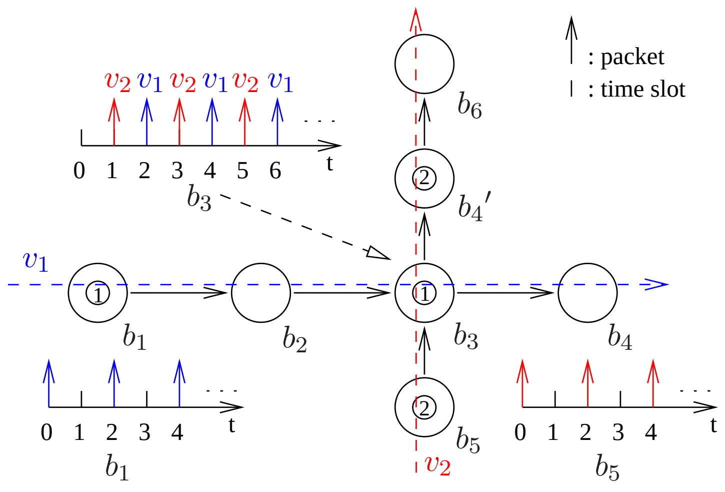


Figure 2.13. Using logical networks to avoid conflict

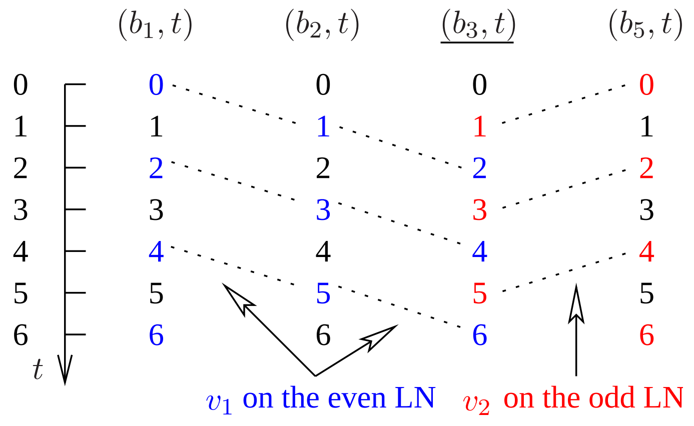


Figure 2.14. The view of logical networks

a VC in an unambiguous way. The reason is that, due to the synchronous network operation and contention-free VC-packet transmission, a VC packet advances one step per slot and never stalls, thus a packet visiting b_1 at even slots will visit b_2 at odd slots, visit b_3 at even slots, and so on. Therefore the partitioned slots are networked, as illustrated in Figure 2.14. We can view that v_1 and v_2 stay in the same physical network but in different logical networks. We define a *logical network (LN)* as a composition of associated sets of time slots in a set of buffers of a VC with respect to a *reference buffer*. We call them LNs because the logically networked slots comprise disjoint networks over time. The overlapping of v_1 and

v_2 results in two LNs, the even and odd LNs; b_3 is the reference buffer. If v_1 and v_2 subscribe to different LNs, they are conflict-free. In the example, v_1 subscribes to the even LN and v_2 to the odd LN.

C. Formal conflit-free conditions

The logical-network concept generalizes the concepts of admission classes [16] and temporally disjoint networks [81]. In Paper 4, we have given formal definitions on the VC and LN. Formally, we have addressed the key questions for the LN-oriented VC configuration. Suppose that v_1 and v_2 are two overlapping VCs,

- The maximum number T of LNs, which both VCs can subscribe to without conflict, equals to $GCD(D_1, D_2)$, the Greatest Common Divisor (GCD) of their admission cycles D_1 and D_2. The *admission cycle* D of a VC v is the length (in number of time slots) of its packet-admission pattern. The bandwidth that a LN possesses equals $1/T$ packets/cycle.

- Assigning both VCs to different logical networks is the sufficient and necessary condition to avoid conflict between them.

- If they have multiple shared buffers, these buffers must satisfy *reference consistency* in order to be free from conflict. If so, any of the shared buffers can be used as the reference buffer to construct LNs. Two shared buffers b_1 and b_2 are termed *consistent* if it is true that "v_1 and v_2 packets do not conflict in buffer b_1" if and only if "v_1 and v_2 packets do not conflict in buffer b_2". The sufficient and necessary condition for them to be consistent is that the distances of b_1 and b_2 along the two VCs, denoted $d_{b_1\vec{b}_2}(v_1)$ and $d_{b_1\vec{b}_2}(v_2)$, respectively, satisfy $d_{b_1\vec{b}_2}(v_1) - d_{b_1\vec{b}_2}(v_2) = kT,\ k \in \mathbb{Z}$. Furthermore, instead of pair-wise checking, the reference consistency can be linearly checked.

D. The VC configuration method

We have used the theorems to guide the construction of VCs. We use a backtracking algorithm to constructively search the solution space while exploring the path diversity. The algorithm is a recursive function performing a depth-first search. The solution space in a tree structure is generated while the search is conducted. The backtracking algorithm trades runtime for memory consumption. At any time during the search, only the route from the start node to the current expansion node is saved. As a result, the memory requirement of the algorithm is $O(m)$, where m is the number of VCs. This is important since the solution space organization needs excessive memory if stored in its entirety. Whenever two VCs overlap, the assignment of LNs to VCs is performed. If they can be assigned to two different LNs with sufficient bandwidth, the assignment is done successfully. Otherwise, the assignment fails. Other path alternatives have to be considered. This VC-to-LN assignment serves as a bounding function by which, if it fails, the algorithm prunes the current expansion node's subtrees, thus making the search efficient.

With a feasible solution, for each VC v_i ($1 \leq i \leq m$) with a normalized bandwidth requirement $b\bar{w}_i$, our VC configuration program returns $(\vec{P}_i, D_i, \vec{A}_i, R_i)$, where $\vec{P}_i$ is the sequence of buffers visited by v_i, representing its delivery path; D_i is the admission cycle by which v_i repeats its packet-injection pattern; $\vec{A}_i$ is the allocated slot vector whose size $|\vec{A}_i|$ is the number of slots in $\vec{A}_i$, and $\forall A_{i,j} \in \vec{A}_i$ ($1 \leq j \leq |\vec{A}_i|$), $A_{i,j} \in [0, D_i)$ and $|\vec{A}_i| \in (0, D_i]$; R_i is the reference buffer for which the time slots are referred to. The LN(s) that v_i subscribes to is reflected in $\vec{A}_i$ explicitly, or implicitly if v_i uses only a portion of bandwidth in the allocated LN(s). In addition, they satisfy the bandwidth constraint:

$$\frac{|\vec{A}_i|}{D_i} \geq b\bar{w}_i$$

As an example, for the two VCs, v_1 and v_2, in Figure 2.13, $D_1 = D_2 = 2$. The number T of logical networks is $GCD(2,2) = 2$. The result of configuring v_1 is $(< b_1, b_2, b_3, b_4 >, 2, \{0\}, b_3)$, which is equivalent to $(< b_1, b_2, b_3, b_4 >, 2, \{0\}, b_1)$. This means that v_1 packets are fired from b_1 at even slots, once every two cycles. The result of configuring v_2 is $(< b_5, b_3, b'_4, b_6 >, 2, \{1\}, b_3)$, which is equivalent to $(< b_5, b_3, b'_4, b_6 >, 2, \{0\}, b_5)$. This means that v_2 packets are fired from b_5 at even slots, once every two cycles.

2.5 Future Work

NoC communication architectures need to offer various services with different guarantees and efficient support for different communication patterns such as multicast, peers and client-server, to provide robust and reliable communication, to enable re-configuration, and to reduce area and power budget. In the future, our research may be complemented along the following threads:

- *Contract-oriented virtual-circuit configuration*: While configuring virtual-circuits (VCs), the network can satisfy their requirements in two ways. One is to meet their demand on its own. The other is to eventually generate contracts through possible negotiation, one for each VC. According to the

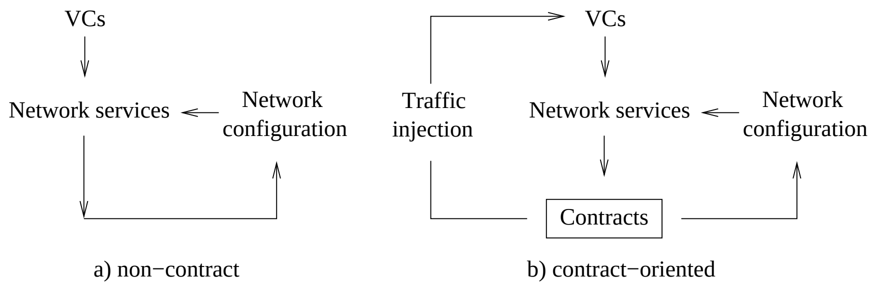


Figure 2.15. Virtual-circuit configuration approaches

contracts, the network nodes configure slot tables. Moreover, VC traffic is injected obligating to the contracts. We illustrate the two approaches in Figure 2.15. Figure 2.15a is non-contract oriented while Figure 2.15b is contract-oriented. As we can see, there is an additional feedback loop introduced in the contract-oriented approach. This loop defines the obligation of VCs in terms of traffic injection pattern. In this way, both the network and VCs must fulfill their obligations. Our VC configuration approach generates $(\vec{P}, D, \vec{A}, R)$ for each VC. This in fact constitutes a *contract*. The network configures slot tables along the VC delivery path using the contracts, and the VCs regularly launch packets using the allocated slots. We expect that a contract-oriented method can facilitate predictable IP integration and the formal validation of QoS guarantees in comparison with a noncontract-oriented one. These benefits as stated have not yet been substantiated.

- *Reconfigurable QoS network architectures*: SoC applications are becoming extremely functionally rich. For example, a personal handheld set does telephoning, multimedia processing, gaming, and may execute diverse web-based utilities. A network designed for such systems must be reconfigurable because different use cases require different configurations. Satisfying all use cases concurrently may over-design the network, leading to unacceptable cost. To have a dynamically re-configurable communication platform is most cost- and power-efficient since it allows us to allocate and use resources only if it is necessary. The challenge is not to sacrifice performance, efficiency and predictability when allowing adaptivity. Efficient protocols, micro-architectures and methods are in a great need to support network reconfigurability. Error-resilient and self-healing mechanisms can be further incorporated to provide fault tolerance and robustness to cope with the nano-regime uncertainties.

Chapter 3

NoC Network Performance Analysis

This chapter summarizes our simulation-based and algorithm-based NoC performance analysis [Paper 5, 6, 7]. The simulation-based analysis [Paper 5, 6] is performed within the Nostrum Network-on-chip Simulation Environment (NNSE). The algorithm-based approach addresses the feasibility test of delivering real-time messages in wormhole-switched networks [Paper 7].

3.1 Introduction

3.1.1 Performance Analysis for On-Chip Networks

A. On-chip network performance analysis

Network-on-chip provides a structured communication platform for complex SoC integration. However, it aggravates the complexity of on-chip communication design. From the network perspective, there exists a huge design space to explore at the network, link and physical layers. In the network layer, we need to investigate topology, switching, routing and flow control. In the link layer, we can examine the impact of link capacity and link-level flow control schemes on performance. In the physical layer, we could inspect wiring, signaling, and robustness issues. Each of the design considerations (parameters) also has a number of options to consider. From the application perspective, the network should not only be customizable but also be scalable. To design an efficient and extensible on-chip network that suits a specific application or an application domain, performance analysis is a crucial and heavy task. The impact of the design parameters at the different layers and the performance-cost tradeoffs among these parameters must be well-understood. The customization on optimality and extensibility can sometimes be in conflict with each other. For instance, a customized irregular topology may be optimal but not easy to scale. In addition, the analysis task is very much complicated because of the un-availability of domain-specific traffic models. Due to the separation between computation and communication, a communication platform may be designed in parallel with the design of computation. The concurrent development speeds up time-to-market, but leaves the development of the communication platform without sufficiently relevant traffic knowledge. Therefore we must be able to evaluate network architectures and analyze their communication performance with various communication patterns extensively so as to make the right design decisions and trade-offs. Once a network is constructed in hardware, it is difficult, time-consuming, and expensive to make changes if performance problems are encountered.

Design decisions include both architecture-level decisions such as topology, switching, and routing algorithm, and application-level decisions such as task-to-node mapping, task scheduling and synchronization etc. While evaluating network architectures and analyzing their performance, we can embed design decisions in experiments during the evaluation and analysis process. In turn, this helps to seek for optimal network and application constructions. Making design decisions is likely to be an iterative process. The feedback information in such a process includes functional and nonfunctional measures. Functional criteria are typically bandwidth, latency, jitter, and reliability, which can be broadly classified into quality-related metrics. Nonfunctional criteria are network utilization, area and power consumption, which are all cost-related.

B. Network performance analysis methods

We may classify network performance analysis methods (before prototyping and implementation) into three categories: *simulation-based*, *algorithm-based* and *mathematics-based*. Below, we give a brief account on them to the extent that is adequate for the introduction.

The *simulation-based* approach builds network and traffic models and then the network operation is simulated by loading the traffic into the network. The network models can be constructed in detail or in an abstract way. For example, a switch model can model all its functional components such as buffers, crossbar and control units in detail. Alternatively, a switch can only model the packet shuffling behavior without modeling each component. A switch may model the internal pipeline stages, leading to a multiple-cycle model. Or it can be just a single-cycle model in which packet switching completes in one cycle. More abstractly, a network may be modeled without building detailed switch models. But the network behavior such as routing and arbitration is modeled so that the net effect of packet delivery such as delay and jitter is reflected in the results. In a simulation-based approach, both synthetic and realistic traffic models can be applied. Furthermore, it allows us to perform system-wide simulation where the interaction between the network and traffic sources/sinks may be captured and the performance-cost tradeoff is examined [103, 114]. The evaluation of the network performance is conducted after simulation statistics are collected. The simulation speed can be different depending on the modeling details [78].

The *algorithm-based* approach makes assumptions on network communication models. In the communication model, the network delivery characteristics and switch arbitration behavior are captured. Additional models may be created to reflect network contention. The network behavior can thus be approximated. Based on the models, an algorithm is then developed to conduct the performance evaluation without resorting to detailed simulation. An algorithm-based approach usually assumes that traffic has certain properties, for example, periodicity and independence. Examples using the algorithm-based approach can be found in [7, 40, 55, 69].

The *mathematics-based* approach builds mathematical models for network and traffic. The performance figures are calculated through formal derivation. For instance, two basic analytic tools for network performance evaluation are queuing theory and probability theory [28]. Queuing theory [36] is useful for analyzing a network in which packets spend much of their time waiting in queues. Probability theory is more useful in analyzing networks in which most contention time is due to blocking rather than queuing. Another example is the use of the network calculus [22, 23] to compute the end-to-end delivery bounds. The mathematics-based approach is most efficient but limited in capability. It can model many aspects of a network, but there are some situations that are simply too complex to express under the mathematical models. Besides, it often simplifies the real situations by making a number of approximations that may affect the accuracy of results.

The performance analysis methods, as described above, are not isolated. They can be used to validate against each other. To validate a model, we need to compare its results against known good data at a representative set of operating points and network configurations. They may be composed to take the advantages of each method. For instance, simulation and formal methods may be combined to speed up the simulation-based performance analysis [57].

C. A comparison of network performance analysis methods

	Simulation	**Algorithm**	**Mathematics**
Com. model	Detailed/Abstracted	Simplified	Accurate/Simplified
Evaluation	Cycle-true/Behavior sim.	Run algorithms	Formal derivation
Execution time	Slow/Medium	Medium	Fast
Accuracy	High/Medium	Medium	High/Medium
Capability	High	Low	Medium

Table 3.1. Network performance analysis methods

All the performance analysis methods require building network and traffic models. They mainly differ in modeling details, efficiency, quality-of-result (accuracy) and capability. We compare the three methods in Table 3.1[1]. The simulation-based method can offer the highest accuracy but may be very time-consuming. Each simulation run can take considerable time and evaluates only a single network configuration, traffic pattern, and load point. It is difficult, if not impossible, to cover all the system states. Depending on the details simulated, runtime and accuracy trade off with each other. However, simulation, in contrast to emulation and implementation, is flexible and cheap. It can also model complex network designs for which mathematical or other analytical models are difficult to build. A simulation tool usually enables to explore the architectural design space and assess design quality regarding performance, cost, power and reliability etc. The algorithm-based scheme does not run network simulation, but the network behavior is captured in an algorithm. It is generally faster than simulation-based schemes, but only approximates the simulated results. The mathematical analysis [2, 35] is the most efficient one. It provides approximate performance numbers with a minimum amount of effort and gives insight into how different factors affect performance. It also allows an entire family of networks with varying parameters and configuration to be evaluated at once by deriving a set of equations that predict the performance of the entire family. The accuracy of results depends on the accuracy of the mathematical models for the traffic and network. It can be rough but gives an initial and quick estimation. A performance bound may be also tight enough.

[1]Note that the qualitative assessments on run-time, accuracy and capability emphasize the differences between the methods. They are relative, and should not be considered absolute.

As simulation is most powerful, once it is verified, it is typically used to validate the algorithm-based and formalism-based approaches. In the next subsection, we outline current practices of NoC simulation.

3.1.2 Practices of NoC Simulation

NoC researchers have used general-purpose network simulators and NoC-specific simulators to simulate the network behavior. OPNET is a commercial network simulator used in [15, 133]. It provides a tool for hierarchical modeling and includes processes, network topology description and supports different traffic scenarios. However, to simulate an on-chip network, it has to be adapted by explicitly modeling synchronous operations and distribution [133]. OMNET is an open-source C++-based network simulation engine. It is used in [89] to validate a network contention model proposed in [69]. As with OPNET, additional modules are needed to model synchronous network operations in OMNET. Semla [125, 126] is a dedicated NoC simulator written in SystemC [20]. It implements five layers of the OSI seven-layer model (without the presentation and session layers), and is equipped with transaction-level primitives to communicate messages between application processes. The SystemC kernel provides the concurrent and synchronous operation semantics, thus a SystemC-based network simulator can take this advantage. In [8], a VHDL-based RTL model is created for evaluating power and performance of NoC architectures. It can model dynamic and leakage power at the system-level. The Orion [131] performance-power simulator models only the dynamic power consumption.

OCCN (On Chip Communication Network) [21] models on-chip network communication in SystemC using high-level modeling concepts such as transactions and channels. In [77], an on-chip communication network is treated as a *communication processor* to reflect servicing demands. The network is modeled using allocators, schedulers and synchronizers. The allocator decides the resource requirements such as bandwidth and buffers along a message's path while minimizing resource conflict. The scheduler executes the message transfer accordingly, minimizing the resource occupation. The synchronizer performs synchronization according to dependencies among messages while allowing concurrency. In [32], network communication is defined as a multiport blackbox communication structure. A message can be transmitted from an arbitrary port to another but the actual implementation of the NoC may not be considered.

Next, we present our NoC simulation tool NNSE in Section 3.2. In Section 3.3, we present our algorithm-based network performance analysis, focusing on the feasibility test of delivering real-time messages, i.e., whether their timing constraints can be met or not.

3.2 NNSE: Nostrum NoC Simulation Environment

3.2.1 Overview

NNSE stands for Nostrum NoC Simulation Environment in which Nostrum is the name of our NoC concept [81]. NNSE is aimed to be a tool for full NoC system simulation so that designers can use it to explore the architecture-level and application-level design space. Currently, it is capable of

- constructing network-based communication platforms [125],
- generating synthetic and semi-synthetic Traffic Patterns (TPs) [68],
- simulating the communication behavior with the various TPs [76], and
- mapping application tasks onto the platform [71].

The first three functions have been automated and the last function is so far a manual step. The automation is achieved through parameterizing network and synthetic traffic configurations. One can configure these parameters, recompile the program if the parameters are compile-time, and invoke simulations with the specified network and traffic configurations. This procedure can be conducted in a Graphical User Interface (GUI). With the GUI, the tool allows us to easily explore different network architectures and different traffic settings. Network architectures can thus be efficiently and extensively evaluated. In addition to using synthetic traffic, the manual application mapping creates realistic traffic scenarios in the communication platform. The evaluation may be iterative by applying the configured or created traffic on the configured networks, as illustrated in Figure 3.1. The evaluation criteria can be performance, power and cost. The current version evaluates only the network performance in terms of packet latency, link utilization and throughput. Since it simulates the network behavior at the flit-level cycle-by-cycle, the performance estimates are accurate.

NNSE logically comprises a NoC simulation kernel [124, 125] wrapped with a GUI. The kernel is developed in SystemC and the GUI written in Python. Following the ISO's OSI seven-layer model [135], the simulation kernel called *Semla* (Simulation EnviornMent for Layered Architecture) implements five of the seven layers except for the representation and session layers. The simulation tool presently

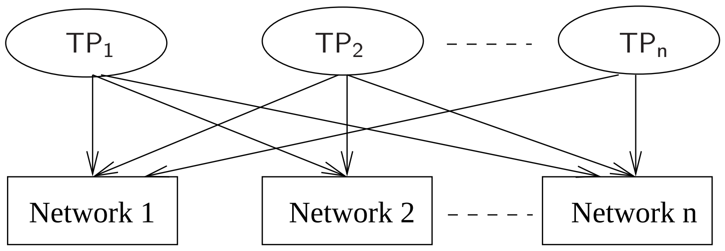


Figure 3.1. Network evaluation

supports the configuration of the network and application layers in the GUI. The configuration of the application layer refers to the traffic configuration. In the GUI, all the network and traffic configurations can be stored and thus reusable. To facilitate data exchange, they are stored as eXtensible Markup Language (XML) files. The simulation results can be shown graphically or in a text format.

3.2.2 The Simulation Kernel

The simulation kernel Semla [124, 125] implements the five communication layers, namely, the physical layer (PL), the data link layer (LL), the network layer (NL), the transport layer (TL) and the application layer (AL). The upper three layers are shown in Figure 3.2, where TG/S stands for Traffic Generator/Sink, and Glue is the TL component which does packetization/packet-assembly, message queuing, multiplexing, de-multiplexing and so on.

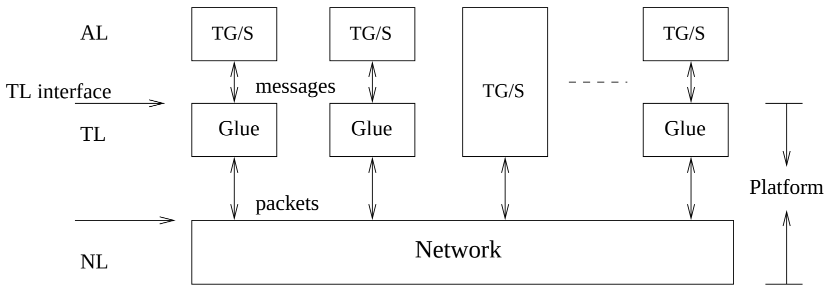


Figure 3.2. The communication layers in Semla

The transport layer provides transaction-level communication primitives as an interface to enable communication via *channels* between application processes. A channel, similar to a SystemC channel [20], is a transaction-level modeling entity which allows simplex communication from a source process to a destination process. In Semla, a compact set of message passing primitives for using the best-effort service is defined and implemented:

- *int open_channel(int src_pid, int dst_pid)*: it opens a simplex channel from a source process *src_pid* to a destination process *dst_pid*. The method returns a positive integer as a unique channel identity number *cid* upon successfully opening the channel. Otherwise, it returns a negative integer for various reasons of failure, such as invalid source and destination processes. The current implementation opens channels statically during compile time and the opened channels are never closed through simulation.

- *bool nb_write(int cid, void msg)*: it writes *msg* to channel *cid*. The size of messages is finite. It returns the status of the write. The write is nonblocking.

- *bool nb_read(int cid, void *msg)*: it reads channel *cid* and writes the received protocol data unit to the address starting at *msg*. It returns the status of the read. The read is nonblocking.

Application tasks use the set of communication primitives to communicate messages with each other. While mapping tasks onto the NoC platform, the network topology is visible. The communication part of the tasks must be written in or adapted to the communication primitives. The interaction between the tasks creates realistic workload in the platform, and the system behavior can be simulated.

Thanks to the layering, one can design and implement different structures and protocols in a layer without modifying other layers as long as one complies with the interfaces. For instance, Semla originally developed the network layer for deflection routing. In order to perform experiments on flit-admission and flit-ejection schemes in Chapter 2, the network layer for wormhole switching was developed and integrated into the simulator. The physical layer was skipped because the interest was on the flit-level not the phit-level activities. While the compilation and simulation were invoked, only the network layer entity was replaced while the upper layers remained the same.

3.2.3 Network Configuration

We parameterize a network according to topology, switching mode and routing algorithm. The network configuration is thus straightforward, as illustrated by the tree in Figure 3.3. The topology is for a 2D regular structure, which can be dimensioned along the number of nodes on the X axis, the number of nodes on the Y axis. The structure may be chosen from one of the options (mesh, torus, tree,

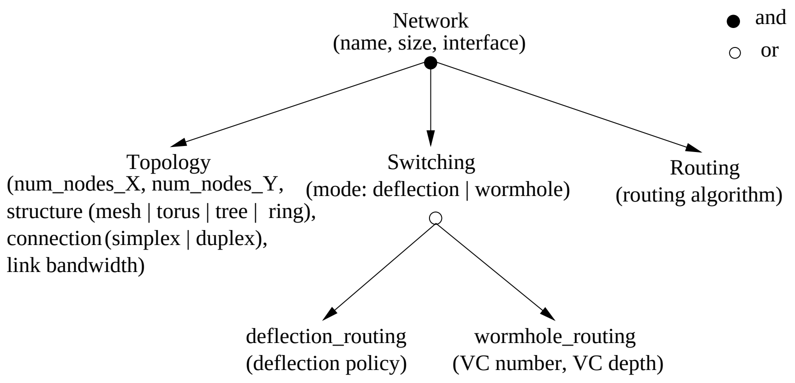


Figure 3.3. Network configuration tree

ring). The link connection may be simplex or duplex with different data width. The network parameters may be further elaborated resulting in the next level in the tree since each of them has a number of choices on its own. As can be seen, with the deflection scheme, different deflection polices may be chosen; with the wormhole scheme, the number and depth of virtual channels (VCs) may be specified.

3.2.4 Traffic Configuration

This subsection summarizes the research in Paper 5.

A. Traffic configuration approaches

Network evaluation typically employs *application-driven* and *synthetic* traffic [28]. Application-driven traffic models the network and its clients simultaneously. This is based on full system simulation and communication traces. Full system simulation requires building the client models. Application-driven traffic can be too cumbersome to develop and control. In NNSE, application-driven traffic is created by mapping application tasks onto the communication platform. Synthetic traffic captures the prominent aspects of the application-driven workload but can also be easily designed and manipulated. Because of this, synthetic traffic is widely used for network evaluation.

In NNSE, two types of traffic can be configured. One is purely *synthetic traffic*, the other *application-oriented traffic*. For synthetic traffic, we proposed a unified formal expression for both uniform and locality traffic. With this expression, we can control the locality of traffic distribution by setting locality factors for the traffic. The application-oriented traffic is semi-synthetic, which can be viewed as a traffic type between application-driven traffic and synthetic traffic. It statically defines the spatial distribution of traffic on a per-channel basis according to application, and the temporal and size distributions of each channel may be synthetic or extracted from communication traces.

B. The traffic configuration tree

Traffic can be characterized and constructed via its distributions over three dimensions: *spatial distribution*, *temporal characteristics*, and *message size specification*. The spatial distribution defines the communication patterns between sources and destinations. The temporal characteristics describe the message generation probability over time. The size specification gives the length of generated messages. We use a traffic configuration tree to express the elements and their attributes in Figure 3.4.

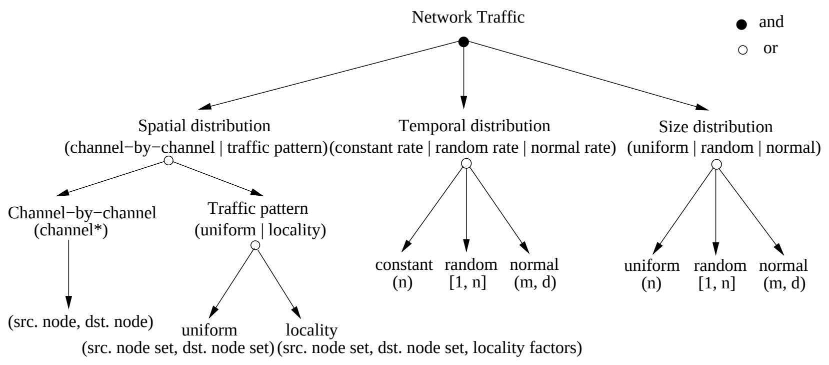


Figure 3.4. The traffic configuration tree

By the spatial distribution, traffic is broadly classified into two categories: *traffic pattern* and *channel-by-channel* traffic. In a traffic pattern, all the channels share the same temporal and size parameters. In contrast, channel-by-channel traffic consists of a set of channels, and each channel can define its own temporal and size parameters. The temporal distribution has a list of candidates such as constant rate (periodic), random rate, and normal rate etc. The size distribution has a list of choices such as uniform, random, and normal. As can be observed, these lists are just examples of possible distributions. Other useful distributions can be integrated into the tree with their associated parameters. According to the tree, configuring a traffic pattern is to select a set of parameters on the three axes. Note that the axes may not be independent. For instance, scale-invariant burstiness traffic and scale-variant burstiness (self-similar) traffic [100] involve the variation in the time scale and message size, thus requiring the synergy between the temporal distribution and the size distribution.

C. Representation of traffic patterns

As shown in Figure 3.4, the traffic patterns consist of *uniform* and *locality* traffic. They can be uniformly expressed in a formal representation.

We first define *communication distribution probability* DP from node i to node j $DP_{i \to j}$ as the probability of distributing messages from node i to node j while node i sends messages to the network. Suppose, there are N nodes in the network, Equation 3.1 means that all messages from node i are aimed to the N destination nodes.

$$\sum_{j=1}^{N} DP_{i \to j} = 1 \tag{3.1}$$

Next, we relate DP to the minimal distance between nodes. Let the shortest distance between a source node i and a destination node j be d, we define communication distribution probability $DP_{i \to j}$ as a relative probability to a common probability factor P_c ($0 \leq P_c \leq 1$) in Equation 3.2.

$$\begin{aligned} DP_{i \to j} &= coef(\alpha, d) \cdot P_c \\ where\ coef(\alpha, d) &= 1 + \frac{\alpha}{d+1} \end{aligned} \tag{3.2}$$

In the equation, *coef* is the *distribution coefficient* and α called *locality factor*. Since $DP_{i \to j} \geq 0$, $\alpha \geq -(d+1)$. Particularly when $\alpha = -(d+1)$, $DP_{i \to j} = 0$; when $\alpha = 0$, $DP_{i \to j} = P_c$. Besides, when $-(d+1) < \alpha < 0$, $DP_{i \to j}$ is proportional to distance d; When $\alpha > 0$, $DP_{i \to j}$ is inversely proportional to distance d. In addition, $\alpha(d)$ can be defined for each possible value of distance d.

Using the traffic expression, the locality of traffic distribution can be easily controlled by setting $\alpha(d)$ for each possible distance value d. For instance, if $\alpha(d) = -(d+1)$, $coef(-(d+1), d) = 0$ meaning that no traffic is generated between sources and destinations if their shortest distance is d; if $\alpha(0) = -1$ for $d = 0$, $coef(-1, 0) = 0$, meaning that no self-loop traffic is created. If we set "$\alpha(d) = 0$" for all possible values of d in the network, their distribution coefficients $coef(0, d) = 1$. Then for any source node i, it has an equal probability to distribute traffic to any node j. In this case, the traffic distribution is independent of distance d, meaning that the traffic is uniform. After setting $\alpha(d)$, we can calculate $coef(\alpha, d)$ and P_c using Equations 3.1 and 3.2. Then $DP(d)$ can be derived. An example of the calculation is given in Paper 5.

D. Channel-by-channel traffic

For the traffic patterns, we control the traffic generation and locality by setting a locality factor α for each possible distance d. Since one distance may cover a number of pairs of source and destination nodes, we avoid specifying the communication distribution probabilities for each source node to each and every possible destination node. For channel-by-channel traffic, as the name suggests, we set traffic parameters for each individual channel. The set of traffic parameters of a channel is $\{s\_proc, d\_proc, \mathcal{T}, \mathcal{S}\}$, where s_proc represents the source process, d_proc the destination process, $\mathcal{T}$ its temporal characteristics, and $\mathcal{S}$ its message size specification. For each channel, we can determine the source node for s_proc and the destination node for d_proc after the application task graph is mapped onto the network nodes. The temporal characteristics $\mathcal{T}$ and the message size specification $\mathcal{S}$ can be synthetically configured using the same set of options in the tree or approximated using analysis or communication traces [68].

Channel-by-channel traffic differs from the traffic patterns mainly in that the traffic's spatial pattern is statically built on a per-channel basis according to an application task graph. Since the communication pattern in the task graph is captured, this type of traffic is used to construct application-oriented workloads.

3.2.5 An Evaluation Case Study

As a case study (Paper 6), we have evaluated deflection networks in NNSE [76]. A deflection-routed network (see Section 2.1 of Chapter 2) has three orthogonal characteristics: *topology*, *routing algorithm* and *deflection policy*. It is crucial to explore the alternatives of the three aspects since the decisions on these aspects may be hardwired and may not be dynamically configurable or too costly to permit dynamic configuration. Therefore identifying the significance of each factor and evaluating their alternatives play a vital role in the decision-making.

In the evaluation, we have considered 2D regular topologies such as *mesh*, *torus* and *Manhattan Street Network*, different routing algorithms such as *random*, *dimension XY*, *delta XY* and *minimum deflection*, as well as different deflection policies such as *non-priority*, *weighted priority* and *straight-through* policies [76]. Our results suggest that the performance of a deflection network is more sensitive to its topology than the other two parameters. It is less sensitive to its routing algorithm, but a routing algorithm should be minimal. A priority-based deflection policy that uses global and history-related criterion can achieve both better average-case and worst-case performance than a non-priority or priority policy that uses only local and stateless criterion. These findings are important since they can guide designers to make right decisions on the network architecture, for instance, selecting a routing algorithm or deflection policy which has potentially low cost and high speed for hardware implementation.

3.3 Feasibility Analysis of On-Chip Messaging

This section summarizes the research in Paper 7.

3.3.1 Problem Description

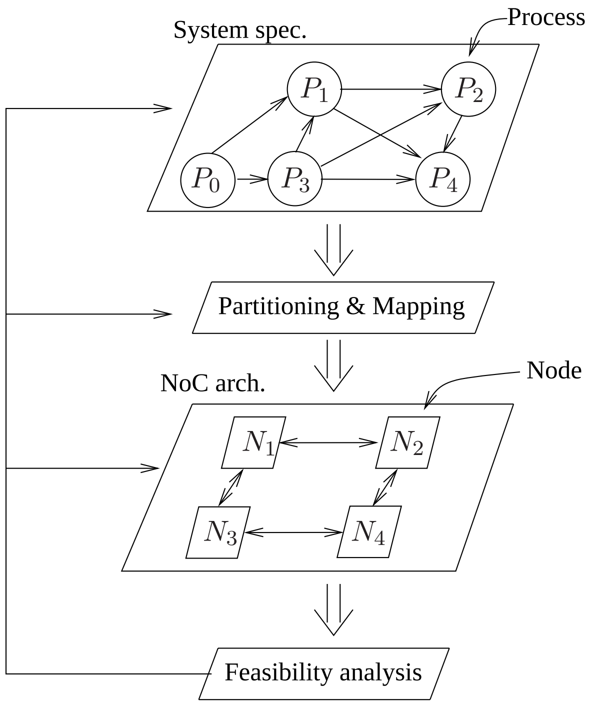


Figure 3.5. Feasibility analysis in a NoC design flow

As illustrated in Figure 3.5, NoC design starts with a system specification which can be expressed as a set of communicating tasks. The second step is to partition and map these tasks onto the resources of a NoC. With a mapping, application tasks running on these resources load the network with messages, and impose timing constraints for delivering messages. The feasibility analysis is performed on the resulting NoC instance. Feasibility analysis could, on its own, cover a wide range of evaluation criteria such as performance, power and cost. In our context, we concentrate on the timely delivery of messages, which is essential for performance and predictability.

Following [110], we distinguish *real-time* and *nonreal-time* messages in on-chip networks. Messages with a deterministic performance bound, which must be delivered predictably even under worst case scenarios, are *real-time* (RT) messages. Messages with a probabilistic bound, which ask for an average response time, are *nonreal-time* messages. Our focus in the thesis is on the feasibility analysis of delivering RT messages in a wormhole-switched network. We follow the feasibility definition in [7]: *Given a set of already scheduled messages, a message is termed* feasible *if its own timing property is satisfied irrespective of any arrival orders of the messages in the set, and it does not prevent any message in the set from meeting its timing property.* We resort to an algorithm-based instead of simulation-based approach in the analysis to avoid cycle-by-cycle simulations. Since it is the network contention that makes the message delivery non-deterministic, we formulate a *contention tree* model that captures direct and indirect network contentions and reflects concurrency in link usage. Based on this model, we investigate message scheduling to estimate the worst-case performance for RT messages and develop an algorithm to conduct the feasibility analysis. The analysis returns the pass ratio, i.e., the percentage of feasible messages, and the network utilization of the feasible messages.

In the following, we first describe the contention tree model, message scheduling on a contention tree, and then the feasibility analysis flow.

3.3.2 The Network Contention Model

A. The real-time communication model

Wormhole switching divides a message into a number of flits for transmission[2]. During the delivery, it manages two types of resources, the lanes and the link bandwidth. Lanes are flit buffers organized into several independent FIFOs instead of a single FIFO. Lane allocation is made at the message level while link bandwidth is assigned at the flit level. In conventional wormhole switches, the shared lanes are arbitrated on First-Come-First-Serve (FCFS), and the shared link bandwidth are multiplexed by the lanes. Messages are not associated with a priority and they are equally treated. This model is fair and produces average-case performance results.

[2]The effect of packetization is not considered here.

It is suitable to deliver nonreal-time messages, which do not require guarantees. But, it can not directly support real-time messages because there is no promise that messages are delivered before deadlines. In order to enable guarantees, real-time messages must be served with other disciplines, for instance, priority-based arbitrations [62].

We assume a conventional wormhole switch architecture and a priority-based delivery model for RT messages. Special RT communication services generally require special architectural support which may potentially complicate the switch design. All messages are globally prioritized, and priority ties are resolved randomly. This model arbitrates shared lanes and link bandwidth on priority. The priority, which may be assigned according to rate, deadline or laxity [40, 62], takes a small number of flits. With this RT communication model, the worst-case latency T^{rt} of delivering a message of L flits is given by :

$$T^{rt} = (L + L_{pri})/B^{rt} + HR + \tau = T + \tau \tag{3.3}$$

where B^{rt} is the link bandwidth allocated to the RT message along its route; H is the number of hops from the source node to the destination node; R is the routing delay per hop; L_{pri} is the number of flits used to express the message priority. The routing delay R is assumed to be the same for head flits and body/tail flits. The first term counts for the transmission time of all the message flits; the sum of the first two terms is the non-contentional or base latency T, which is the lower bound on T^{rt}; the last term τ is the worst-case blocking time due to network contention.

B. Network contention

To estimate the worst-case latency T^{rt} of an RT message M, we have to estimate the worst-case blocking time τ. To this end, we first determine all the contentions the message may meet.

In flit-buffered networks, the flits of a message M_i are pipelined along its routing path. The message advances when it receives the link bandwidth along the path. The message may directly and/or indirectly contend with other messages for shared lanes and link bandwidth. M_i has a higher priority set S_i that consists of a *direct contention* set S_{D_i} and an *indirect contention* set S_{I_i}, $S_i = S_{D_i} + S_{I_i}$. S_{D_i} includes the higher priority messages that share at least one link with M_i. Messages in S_{D_i} directly contend with M_i. S_{I_i} includes the higher priority messages that do not share a link with M_i, but share at least one link with a message in S_{D_i}, and $S_{I_i} \cap S_{D_i} = \emptyset$. Messages in S_{I_i} indirectly contend with M_i. As an example, Fig. 3.6a shows a fraction of a network with four nodes and four messages. The messages M_1, M_2, M_3 and M_4 pass the links AB, BC, AB→BC→CD, and CD, respectively. A lower message index denotes a higher priority. The message M_1 has the highest priority, thus $S_1 = \emptyset$. For the message M_2, it directly contends with M_3, but it has a higher priority, thus $S_2 = \emptyset$. The message M_3 has a higher priority message set $S_3 = S_{D_3} = \{M_1, M_2\}$, $S_{I_3} = \emptyset$. For the message M_4, $S_{D_4} = \{M_3\}$ and $S_{I_4} = \{M_1, M_2\}$ because M_1 or M_2 may block M_3 which in turn blocks M_4.

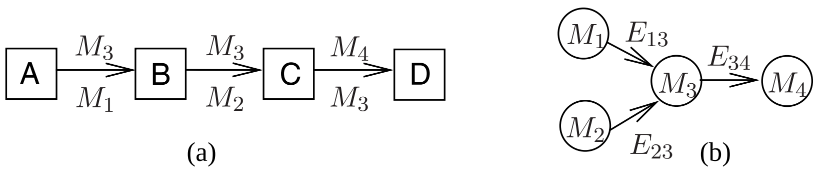


Figure 3.6. Network contention and contention tree

C. The contention tree

To capture both direct and indirect contentions and to reflect concurrent scheduling on disjoint links, we have formulated a *contention tree* model that is defined as a directed graph $G : M \times E$. A message M_i is represented as a node M_i in the tree. An edge $E_{ij}(i < j)$ directs from node M_i to node M_j, representing the direct contention between M_i and M_j. M_i is called *parent*, M_j *child*. Given a set n of RT messages, after mapping the messages to the target network, we can build a contention tree with the following three steps:

Step 1. Sort the message set in descending priority sequence with a chosen priority assignment policy.

Step 2. Determine the routing path for each of the messages.

Step 3. Construct a tree, starting with the highest priority message M_1, and then $M_2 ... M_n$. If M_i shares at least one link with M_j where $i < j \leq n$, an edge E_{ij} is created between them. Each node in the tree only maintains a list of its parent nodes.

In a contention tree, a direct contention is represented by a directed edge while an indirect contention is implied by a *walk* via parent node(s). A walk is a path following directed edges in the tree. The contention tree for Fig. 3.6a is shown in Fig. 3.6b, where the three direct contentions are represented by the three edges E_{13}, E_{23} and E_{34}, and the two indirect contentions for M_4 are implied by the two walks $E_{13} \rightarrow E_{34}$ and $E_{23} \rightarrow E_{34}$ via M_4's parent node M_3. Since determining the routing path is a priori, creating a contention tree is more suitable for deterministic routing. For adaptive routing, it is difficult to figure out the worst-case routing path.

D. Assumptions and simplifications

The estimation of latency bounds are based on messages' schedules on links. A schedule is a timing sequence where a time slot is occupied by a message or left empty. The latency bound of a message is the earliest possible completion time for delivery under the worst case. Before introducing schedules of messages, we list the assumptions, limitations and simplifications as follows:

- The messages we consider are periodic and independent. There is no data dependency among messages so that each message can be periodically fired or activated, meaning that the messages are sent to the network and start to compete for shared resources, i.e., buffers and links.

- We focus on link contentions. Similarly to [7, 40], we assume that there is a sufficient number of Virtual Channels (VCs) so that priority inversion due to VC unavailability does not occur. Priority inversion happens when a message with a lower priority holds shared resources, leading to blocking messages with a higher priority. As discussed in [7, 40], this problem can be alleviated by packetization.

- In this communication model, messages are allocated with time slots depending on their priorities and contentions. Whenever there is a contention for a link, a message with a higher priority will be scheduled first. In addition, a higher priority message can preempt a lower priority message.

- The worst case is assumed to occur when all the messages are fired into the network at the same time.

- The bandwidth of a link is assumed to transmit one flit in one time slot. The routing delay per hop takes one time slot. We simplify the pipeline latency on links so that the flits of a message are available to compete *all* the link bandwidth along the message's path simultaneously for the duration of its communication time. To explain this, we illustrate a message transmission in Figure 3.7, where M_2 passes through three hops (A, B, C) and two links (AB, BC). M_2 contains four flits (one head `h` flit, two body `b` flits and one tail `t` flit). It has a base latency of 7 $(1 \cdot 3 + 4)$. If M_2 fires at time instant 0, by the assumption, it will compete for *both* links AB and BC from slot 1 to 7, i.e., for its entire base latency period.

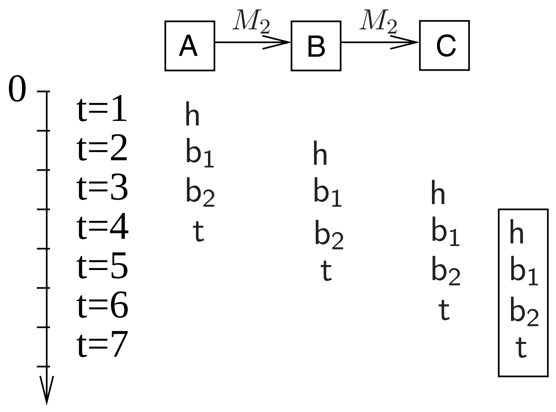


Figure 3.7. Message contention for links simultaneously

- We assume that a message advances only if it simultaneously receives all the link bandwidth along its path. This means that the flits are delivered either concurrently via the links or blocked in place. As a result, a message competes for links only for its base latency period. It does not happen that a flit advances via a link while another flit is blocked in place. As shown in Figure 3.8, at time slot 3, the head flit h has advanced from node B to C but the first body flit b_1 is blocked in node A. As a consequence, the pipeline latency is increased by one slot. According to our assumption, this scenario in time slot 3 is avoided and thus not considered. Apparently, if flits are individually routed via links, the contention period may become larger than its base latency and unpredictable.

E. Scheduling on the contention tree: an example

Table 3.2 shows an example of message parameters for Fig. 3.6, where the priority is assigned by rate, and the deadline D equals period p. The worst-case schedules for the three links are illustrated separately in Fig. 3.9a. Initially, all messages are fired. M_1 is allocated 7 slots on link AB. M_2 is allocated 3 slots on link BC. M_3 is blocked by M_1 and M_2. M_4 is blocked by M_3. After M_1 and M_2 complete transmission, M_3 is allocated 3 slots concurrently on link AB, BC and CD. At time slot 10, M_1 fires again and holds slots $[11, 17]$ on link AB, preempting M_3. At time slot 15, M_2 fires the second time and holds slots $[16, 18]$ on link BC. After M_1 and M_2 complete their second transmission, M_3 continues its first transmission by holding slots $[19, 20]$. After M_3 finishes its first delivery, M_4 is allocated

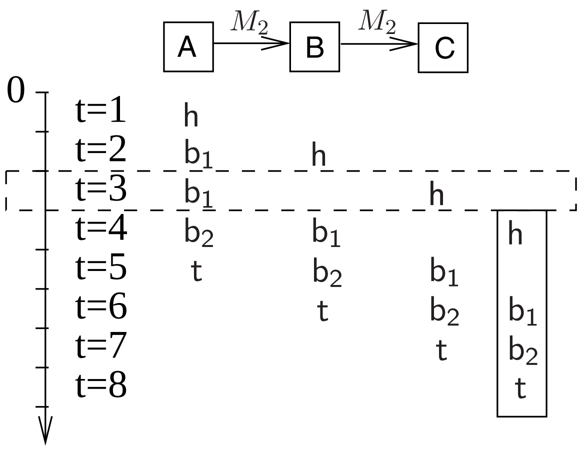


Figure 3.8. Avoided flit-delivery scenario

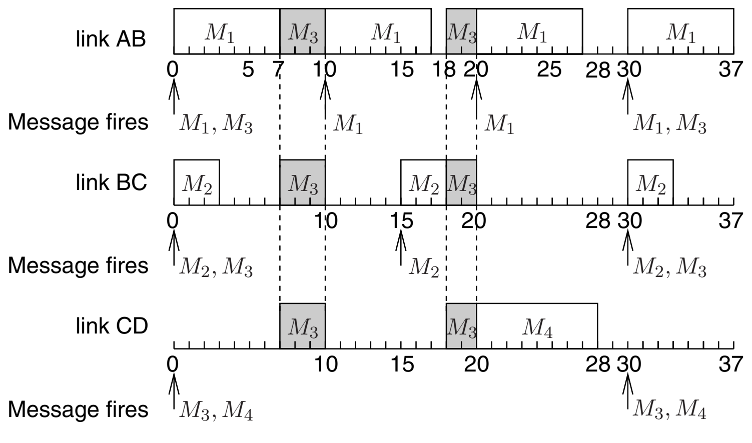
Table 3.2. Message parameters and latency bounds

Message	**Period** p	**Deadline** D	**Base latency** T	**Latency bound** T^{rt}
M_1	10	10	7	7
M_2	15	15	3	3
M_3	30	30	5	20
M_4	30	30	8	28

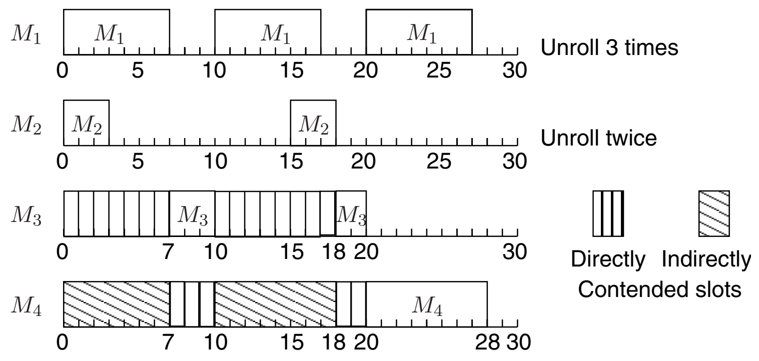
slots $[21, 28]$ on link CD. M_1 starts its third round and holds slots $[21, 27]$ on link AB. Since the four messages have a Least Common Multiple (LCM) period of 30, the four messages are scheduled in the same way at each LCM period. From the schedules, we can find that the latency bounds for M_1, M_2, M_3, M_4 are 7, 3, 20, 28, respectively. Equivalently, the worst-case blocking times for the four messages are 0, 0, 15, 20. The latency bounds for the four messages are also listed in Table 3.2. We can see that all the four messages are feasible.

Looking into the schedules, we can observe that

(1) M_1 and M_2 are scheduled in parallel. This concurrency is in fact reflected by the *disjoint* nodes in the tree. We call two nodes *disjoint* if no single walk can pass through both nodes. For instance, M_1 and M_2 in Fig. 3.6b are disjoint, therefore their schedules do not interfere with each other.



(a) Link schedules of the messages



(b) Global schedules of the messages

Figure 3.9. Message scheduling

(2) M_3 is scheduled on the overlapped empty time slots [8, 10] and [19, 20] left after scheduling M_1 and M_2. This is implied in the tree where M_3 has two parents, M_1 and M_2. The *contended slots* [1,7] and [11,18] are occupied by M_1 or M_2. A contended slot is a time slot occupied by a higher priority message when the contention occurs. A contention occurs only when two competing messages are fired.

(3) M_4 is scheduled only after M_3 completes transmission at time 20. The indirect contentions from M_1 and M_2, which are reflected via slots [1,7] and [11,18], *propagate* via its parent node M_3. For M_3, these slots are directly contended

slots. For M_4, they become indirectly contended slots.

The four message schedules are individually depicted in Fig. 3.9b. If the direction contention is not distinguished from the indirect contention as the lumped-link model [7] does, M_3 and M_4 would be considered infeasible since M_2 would occupy the slots [8, 10] and [18, 20], leaving only three slots [28, 30] for M_3 and M_4. If the concurrent use of the two links, AB by M_1 and BC by M_2, was not properly captured as the blocking-dependency model [55] does, M_3 and M_4 would also be considered infeasible since M_2 would occupy the slots [8, 10] and [18, 20] before slot 30.

In a contention tree, all levels of indirect contentions propagate via the intermediate node(s). This is pessimistic since many of them are not likely to occur at the same time. Also, a lower priority message can actually use the link bandwidth if a competing message with a higher priority is blocked elsewhere.

The validation of the contention-tree model as well as the comparisons with other proposed contention models are provided in [89].

3.3.3 The Feasibility Test

A. The feasibility test algorithm

Based on the contention tree and priority-based message scheduling, each feasible message obtains a global schedule. A message schedule is based on its parents' schedules. If a node has no parent or feasible parent, it is scheduled whenever it fires, thus it is always feasible. If a node has feasible parent(s), we must first mark the contended slots as occupied and then schedule the node.

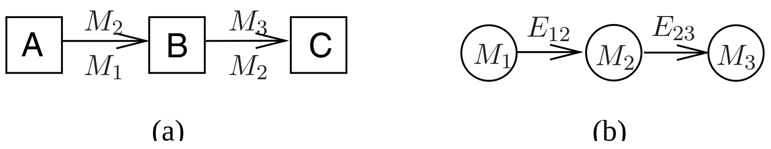


Figure 3.10. A three-node contention tree

Note that a slot occupied by a higher priority message is not necessarily a *contended* slot. Consider the contention tree in Figure 3.10 where the three messages use the parameters in Table 3.2. The message schedules are depicted in Figure 3.11. M_1 has the highest priority and schedules whenever it fires. Consider the LCM period for the three messages, which is 30 in this case, M_1 fires three times and occupies slots [1, 7], [11, 17] and [21, 27]. M_2 fires twice at time 0 and 15.

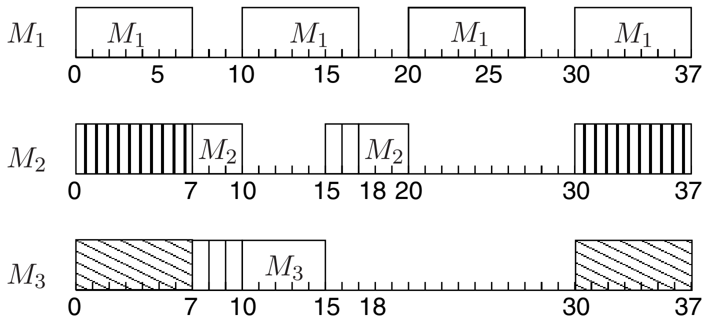


Figure 3.11. Message scheduling and contended slots

Although the slots [10, 15] and [21, 27] are occupied by M_1, M_1 does not contend with M_2 during these time slots since M_2 is not fired or has already been scheduled. The directly contended slots with M_1 are slots [1, 7] and [16, 17], implying that M_2 can not be scheduled on these slots. Hence, M_2 schedules on slots [8, 10] and [18, 20]. M_3 fires once at time 0. The contended slots are [1, 7] (indirectly with M_1) and [8, 10] (directly with M_2). Hence, M_3 is scheduled on slots [11, 15]. In summary, a slot is regarded as a contended slot only if two conditions are true: (1) it is occupied by a higher-priority message; (2) competing messages must fire at the time slot. Particularly, for indirectly contended slots, the intermediate message(s) must also fire in order to pass the contention downwards; otherwise, the slots are not contended. As illustrated in Figure 3.11, for M_3, slots [11, 15] are occupied by M_1 but not contended slots, since M_2 are not fired during these slots. Therefore M_3 is scheduled on these slots.

The indirect contentions propagate via parent nodes. Disjoint nodes are scheduled concurrently. If a node M has k feasible parents, M can only be scheduled on the overlapped empty or free (non-contended) slots of the k parents' schedules. The feasibility of a message can be determined by comparing the number N of empty slots available for scheduling M with its non-contentional or base latency T. We distinguish messages with a deadline constraint D or a jitter constraint J. For a deadline constrained message, its latency bound T^{rt} must satisfy $T^{rt} \leq D$; For a jitter constrained message, its latency bound T^{rt} must satisfy $D - J \leq T^{rt} \leq D$. For a message M with a base latency T, we denote that the number of available slots for scheduling M before its jitter range $D - J$ and before its deadline D is N_J and N_D, respectively. If M is deadline-constrained and $T \leq N_D$, M is feasible (feasible(M)=1); otherwise, M is infeasible (feasible(M)=0). If M is jitter-constrained and $N_J \leq T \leq N_D$, M is feasible; otherwise, M is infeasible.

Algorithm 1 Contention-Tree-based Feasibility Test for Real-Time Messages

```
Input: A sorted set of $n$ messages and a contention tree for the messages;
Output: Feasible$(M_i) = 1/0$, for $i = 1, 2, \cdots n$;
1   Find the LCM for the periods of all $n$ messages;
2   For a message $M_i$, initially $i = 1$, do {
3     Feasible$(M_i) = 0$;
4     find $M_i$'s feasible parent(s) $F_P$;
5     if $F_P = \phi$
6       fire $M_i$ and schedule it to the length of LCM; Feasible$(M_i) = 1$;
7     else
8       do {
9         fire $M_i$ once;
10        mark $M_i$'s contended slots as occupied and the rest as empty within
          $M_i$'s deadline $D_i$;
11        compute the length $N_{Ji}$ and $N_{Di}$, which are the overlapped empty slots
          on $F_P$'s schedules within $M_i$'s jitter range $D_i - J_i$ and deadline $D_i$,
          respectively;
12        if ($M_i$ is jitter-constrained and $N_{Ji} \leq T_i \leq N_{Di}$)
          or ($M_i$ is deadline-constrained  and $T_i \leq N_{Di}$), Feasible($M_i$)=1;
          and schedule $M_i$ on these free time slots;
13        else Feasible$(M_i) = 0$; release the scheduled slots for $M_i$;
14      } while ($M_i$ fires not reaching LCM) and (Feasible$(M_i) = 1$);
15    $i = i + 1$;
16  } while $(i <= n)$;
```

We formulate this contention-tree-based feasibility test in Algorithm 1. The input to the algorithm is a sorted set of messages with parameters and constraints, and a contention tree for these messages. The output is the feasibility for each of the n messages, either *pass* (feasible, Feasible$(M_i) = 1$) or *miss* (infeasible, Feasible$(M_i) = 0$). After obtaining the feasible messages, we can further estimate the link utilization of the feasible messages. Finding the LCM of the messages' periods is the necessary and sufficient condition in order to terminate the algorithm since the rest of a feasible schedule can be repeated after the LCM.

B. The feasibility analysis flow

Using the feasibility test, we can efficiently conduct feasibility analysis by exploring the application-level, partitioning/mapping-stage and architecture-level design

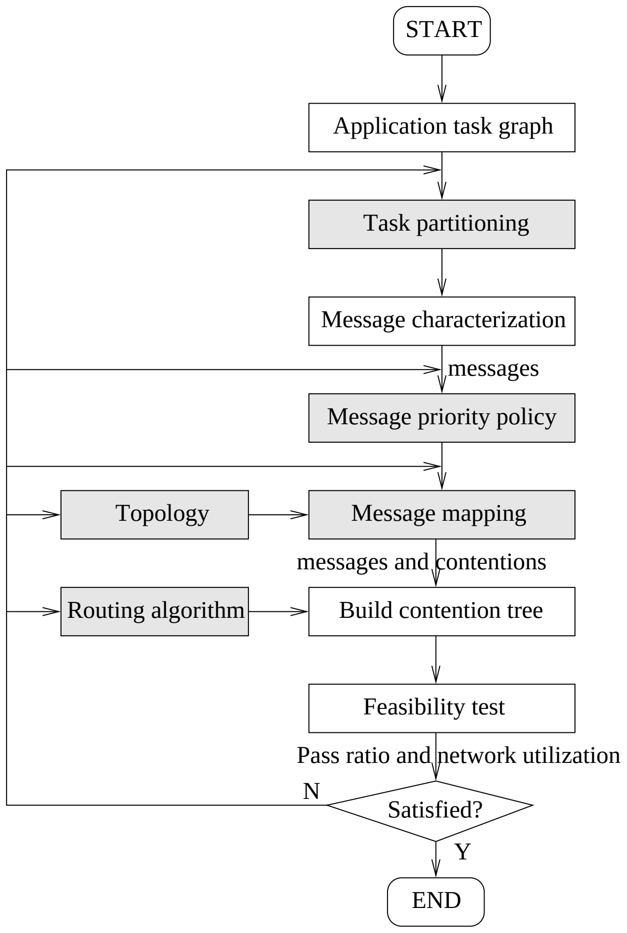


Figure 3.12. A feasibility analysis flow

space. Figure 3.12 shows a feasibility analysis flow. First, we partition the tasks and then characterize the messages from the application task graph. Then we build a contention tree. Since the contention tree is affected by several design decisions such as task partitioning, priority policy, message mapping strategy and the routing algorithm etc., we can build different contention trees by exploring these possibilities. After creating a contention tree, the feasibility test algorithm can perform the analysis. The outcome of the test is the pass ratio and network utilization of feasible messages. These two measures may serve as the criteria to calibrate the design decisions. Clearly, this procedure is iterative until satisfaction.

3.4 Future Work

NNSE has been demonstrated in the University Booth EDA (Electronic Design Automation) Tool Program of DATE 2005 [73]. After publicity, it has been requested for research use by a number of NoC research groups in Europe, U.S.A. and Asia. In the future, we plan to improve it in the following directions:

- *Parameterize more layers*: Current tunable parameters include topology, routing, and switching schemes. Each of the parameters may be extended with more options. These are all network-layer parameters. In NNSE, the layered structure allows us to orthogonally consider other layers' parameters. In the physical layer, we can build wire, noise and signaling models to examine the reliability and robustness issues. We may consider the link layer parameters such as the link capacity, link-level flow control schemes etc. The upper layer like the transport layer allows us to investigate buffer dimensioning and buffer sharing schemes, as well as end-to-end flow control methods.

- *Configure dependent traffic*: We have so far configured independent traffic, both synthetic and semi-synthetic. This means that traffic from different channels is independent from each other. This is easy to control and generate, but realistic traffic exhibits dependency and correlation. The way to generate traffic with various dependencies such as data, control, time, causality etc. is worth investigating. For example, traffic with the requirement of lip-synchronization shows correlated delivery requirements on video and audio traffic streams.

- *Support Quality-of-Service (QoS)*: This requires the implementation of QoS in the communication platform, and accordingly QoS generators and sinks. Monitoring service may be necessary to collect statistics on whether the performance constraints of a traffic stream have been satisfied or not.

- *Integrate application mapping*: A tool that only explores communication performance is not sufficient. System performance is the result of interactive involvement of both communication and computation. Therefore, supporting application-mapping onto NoC platforms is surely desirable. To this end, we need to build and/or integrate resources models for cores, memories and I/O modules.

- *Incorporate power estimation*: As power is as sensible as performance for a quality SoC/NoC product, NNSE should incorporate the estimation of power consumption so that the performance and power tradeoffs can be better investigated and understood.

Extending further the traffic generation for performance evaluation ends up with benchmarking different on-chip networks. The diverse NoC proposals necessitate standard sets of NoC benchmarks and associated evaluation methods to fairly compare them.

Chapter 4

NoC Communication Refinement

This chapter presents our NoC communication refinement approach [Paper 8, 9]. We start with a system model specified in the synchronous model of computation. Through a top-down procedure, we refine the communication in the system model into NoC communication via the communication interface of a NoC platform.

4.1 Introduction

4.1.1 Electronic System Level (ESL) Design

The rapid advancement of technology constantly fuels the SoC revolution [79]. As we mentioned previously, the state-of-the-art SoC design methodologies cannot sufficiently exploit the abundant transistor capacity. An on-going trend to shrink the productivity gap is Electronic System Level (ESL) design. This trend is mixed with the platform-based design concept [54] and the promotion of using formal models for system specification and verification.

Traditional Register Transfer Level (RTL) for hardware design, which was introduced in the 90s, allows synthesized standard cell design. A synthesizable RTL description is presently often the starting point for an ASIC/FPGA design flow. The design productivity cannot keep pace with the exponential expansion of the number of transistors on a chip. Traditional C-based design for embedded software development shows even slower enhancement in design productivity. To shrink the gap between the design capability and the chip capacity, raising the design abstraction-level is an essential step forward. The current activities in Electronic System Level (ESL) [29] is consistent with this direction. The ITRS [46] defined ESL to be a level above RTL, that consists of "a behavioral (before hardware/software partitioning) and architectural level (after)". The ESL raises the abstraction level in which systems are expressed. A system-level design allows larger function-architecture co-exploration [63], which is more than traditional hardware-software codesign. The final implementation can benefit in performance and cost. Using system-level models, hardware and software design can be developed in parallel. This breaks the sequential flow of hardware-first-software-second, thus compressing the design cycle. Besides, the benefits of ESL include enabling new levels of design reuse and offering design chain integration across tool flows and abstraction levels. Using formal models is also advocated for system-level design [54, 116, 118]. As noted in [54], using formal models and transformations in system design is promoted so that verification and synthesis can be applied to advantage in the design methodology. Verification, which is a key design activity, is effective if complexity is handled by formalization, abstraction and decomposition. Besides, the concept of synthesis can be applied only if the precise mathematical meaning of a system specification is defined.

A formal model is associated with Models of Computation (MoCs). As defined in [118], a MoC refers to mathematical models that specify the semantics of computation and of concurrency. Loosely defined, MoC specifies the operational semantics governing how processes interact with each other. There are a variety of MoCs that exist for embedded system design, such as finite state machines [39], Petri nets [86], Kahn process networks [51], and synchronous models [11, 12] etc. A comprehensive digest of the various models can be found in [31, 118]. The tagged-signal model [59] defines a denotational, semantic framework of signals and processes within which models of computation can be studied and compared. In [48], a formal classification and description of these models is presented comprehensively. Essentially, how time and concurrency are expressed distinguishes one MoC from another.

4.1.2 Communication Refinement

Communication refinement is a key step in a system-level design approach. It is a top-down process of synthesizing abstract communication in the system model into concrete communication in the system implementation architecture [31, 54]. Abstraction defines the type of information present in a model. Unlike hierarchy, abstraction is not concerned with the amount of information visible, but with the semantic principles of a model. In general, the movement from high to low abstraction levels involves a decision-making process. By making design decisions and increasing information about implementation details, we replace more abstract models with less abstract models, until the system is manufacturable. Through the refinement process, system properties and application constraints must be incorporated and satisfied.

Communication refinement may be conducted in the functional domain or in the implementation domain and usually comprises well-defined steps. A system model, after steps of refinement, is derived into a refined model. The three key issues for refinement are *correctness, constraint and property satisfaction* and *efficiency*. As the refined model is an elaborate version of the original model, they must be functionally equivalent. This is achieved by preserving semantics during refinement, i.e., a refinement step should not introduce semantic deviation. The second requirement means that the refined, correct model must satisfy design constraints for performance and ensure properties to achieve design objectives. The third one here refers to resource consumption in the system implementation architecture. It can be very specific, depending on whether our application is aimed for low power or low cost.

In the NoC case, the communication architecture is preferably predefined as a platform and the Application Level Interface (ALI), which provides primitives for inter-process communication, is the only way to access the communication services. The NoC communication refinement is therefore to refine the abstract communication in a system specification onto the NoC platform via the ALI. We have proposed a three-step top-down procedure to refine the communication of a system model specified in the synchronous MoC into NoC communication. Before we present the refinement steps, we introduce the synchronous MoC.

4.1.3 Synchronous Model of Computation (MoC)

A. Synchronous modeling paradigm

The synchronous modeling paradigm [11, 12] is based on an elegant and simple mathematical model, which has been shown successful and is the ground of synchronous languages [38] such as Esterel, Signal, Argos and Lustre. The basis is the perfect synchrony hypothesis, i.e., both computation and communication take non-observable time. The critical requirement from specification to implementation is that the implementation has to be *fast enough* both in communication and computation. This means that the implementation phase has to take worst-case into account. Synchronous MoC was initially introduced for reactive and safety-critical embedded control systems where reasoning about the functional correctness is supreme. It has to verify that at each tick over time the system works properly.

In a synchronous MoC, a system is modeled as a set of fully concurrent communicating processes via signals. Processes use ideal data types and assume infinite buffers. By following the tagged-signal model [59], a signal can be defined as a set of ordered events, with each event taking a value and a tag. The value is the informative data to be communicated, and the tag indicates a time slot. This means that each event is conceptually and explicitly accompanied by a time slot to convey data. If the data contains a useful value, the event is $present$ and called a $token$; otherwise, the event is $absent$ and modeled as a $\sqcup$ [1] representing a clock tick. With the introduction of $\sqcup$, multi-rate systems can be modeled since every nth event in one signal aligns with the events in another. A synchronous MoC is a timed MoC where events are globally and totally ordered. Each signal can be related to the time slots of another signal in an unambiguous way. The output events of a process occur in the same time slot as the corresponding input events. Moreover, they are instantaneously distributed in the entire system and are available to all other processes in the same slot. Receiving processes in turn consume the events and emit output events again in the same time slot. Processes can thus be viewed as communicating events via an *ideal channel*, which is delay-free. In addition, the ideal channel is buffer-less and has unlimited bandwidth because any type of event values passes through it instantaneously. This communication channel is in contrast to that of other MoCs. For example, the Kahn and dataflow process networks [58] assume unbounded FIFO channels between actors (processes).

Two events are *synchronous* if they have the same tag. Two signals are *synchronous* if each event in one signal synchronous with an event in the other signal and vice versa. A process is *synchronous* if every signal of the process is synchronous with every other signal of the process. A system is *synchronous* if all processes are synchronous locally and globally (synchronous with each other). A system specified in the synchronous paradigm is a synchronous system. For feedback loops, the perfect synchrony leads to cyclic dependency between an input signal and an output signal. If such cyclic communication is allowed in system behavior, some mechanism must be used to resolve it. One possibility is to introduce a delay in the output signal. Another possibility is to use fixed-point semantics, where the system behavior is defined as a set of events that satisfy all processes. The third possibility is to leave the results undefined, resulting in nondeterminism or infinite computation within one tick. If only one precise result is defined for a feedback loop using the delayed time tag, a synchronous model is determin-

[1]In Paper 9, we used $\perp$ (pronounced “bottom”) to represent *absent*. Since $\perp$ has been used in dataflow process networks to represent *don’t-care* [58], we later used $\sqcup$ in Paper 8 to represent *absent* in order to distinguish it from $\perp$. This notation is also consistent with [48].

istic, i.e., given the same input sequence of events, it generates the same output sequences of events.

B. The ForSyDe methodology

ForSyDe stands for FORmal SYstem DEsign. It is a system-level design methodology for SoC applications developed in the Royal Institute of Technology, Sweden. The ForSyDe methodology [116] is based on the synchronous MoC. It uses *process constructors* to cleanly separate communication from computation. Communication is captured by the process constructors and computation by the function of the processes. It employs transformations in the functional domain to refine a system model into a less-abstract model optimized for implementation [115, 117]. The transformations, which are conducted step by step, can be either semantic-preserving or a design decision. Semantic-preserving transformations are correct by construction while design decision is not. But, formal verification of design decisions is possible by defining an appropriate notion of equivalence [108]. After refinement, the refined model is partitioned into hardware and software and mapped onto the implementation architecture [70]. In ForSyDe, the zero-delay feedback is forbidden. A delay is introduced in the feedback loop. ForSyDe uses the functional language Haskell [127] to express its system models. The models are executable.

Our refinement approach has been conducted in the ForSyDe framework in order to experiment and validate our concepts with executable models, but our refinement approach applies also to a general synchronous model.

C. Modeling NoC applications with the synchronous MoC

The reason to start our refinement from a synchronous model is two fold. One is to adopt a formal MoC for the specification of system function. A synchronous model is formal and purely functional. This highest abstraction level leaves the greatest design space to explore, and the advantage of formalism can be used for well-defined refinement, synthesis and verification. The second reason lies in the appropriateness of the synchronous MoC. To model a NoC application, one can ask which MoC is more appropriate? In general the answer depends on which kind of NoC applications to be designed. Considering the strength and weakness of various MoCs, most probably there is no such a one-size-fits-all MoC but different MoCs find their own roles for different applications. However, as the days of cheap communication are gone, expressing communication in a system specification is necessary. A model for a NoC application has to capture communication properly. Besides, as NoC is a concurrent-processing platform, capturing concurrency in the system model is also necessary. We believe that the synchronous MoC is a good candidate to specify some NoC applications because it captures concurrent computation and communication, and explicitly expresses them in the simplest form possible.

In the following, we first formulate and analyze the communication refinement problem in order to identify the exact sub-problems to be addressed, and then we summarize our solutions presented in Papers 8 and 9.

4.2 The Communication Refinement Approach

4.2.1 Problem Description and Analysis

Our task is to refine synchronous communication into on-chip communication. Specifically, the problem can be formulated as follows: *Given is a synchronous system model, refine the system communication onto a network-based communication platform. During the refinement, design constraints should be satisfied and the network should be efficiently utilized.*

A synchronous model provides globally synchronous, concurrent and instant communication for inter-process communication. The properties of synchronous communication can be summarized as follows:

- *Global synchrony*: There is logically a global clock triggering the consumption and generation of events. Since computation takes non-observable time, input and output events are distributed synchronously in each and every tick.

- *Instancy*: Events pass via an ideal communication channel. The channel provides zero-delay (instantaneous), unlimited bandwidth, ordered, and lossless delivery. The unlimited bandwidth is due to that a signal can have a value type of any kind, such as any primitive and compound types, demanding an arbitrary communication bandwidth. Consequently, if an output signal of a process is connected to an input signal of another process (as long as they have the same value type), the two signals are identical. Thus we can use a single signal to represent both. Together with the global synchrony, ideal channels maintain a global event order.

- *Full-scale concurrency*: In the topology of a process network, all events are communicated in parallel. Each communication channel is point-to-point and dedicated. No serialization on the use of the channels is necessary. Full-scale communication concurrency makes the full-scale process computation concurrency possible because input events are available each tick and processes can evaluate the input events and generate output events each tick.

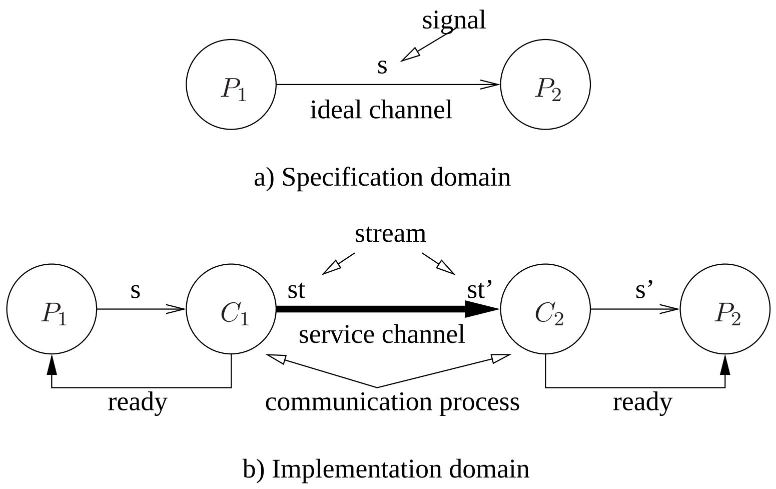


Figure 4.1. Computation and communication elements

The computation elements in the specification domain are *processes*, which produce and/or consume events. The communication elements are *signals*, which are ordered sequences of events, and conceptually *ideal channels*. We illustrate these elements with two processes, P_1 and P_2, in Figure 4.1a.

In the implementation domain, NoC has a very different communication model and associated properties. We consider a message-passing NoC platform where each core has its own local memory. As we discussed in Chapter 1, NoC communication can be represented using the layered and interfaced model. The inter-process communication is offered by the Application-Level Interface (ALI). The application processes use communication primitives such as *open_channel()*, *read()* and *write()* to communicate messages. Logically we can view that process-to-process communication is conducted through a dedicated, point-to-point *service channel*. The service is mapped directly to an underlying network communication service. We simplify the consideration of the session/transport layer, which performs packetization/de-packetization, interleaving for using shared buffers and bandwidth, and re-ordering for maintaining the message causality, if necessary. We assume that the net effect of the session/transport layer is the addition of delay and in-order message delivery. This delay contributes to the delay in the service channel model.

The service-channel communication model differs drastically from the synchronous communication model.

- *Multiple clock domain communication*: There is no a global clock triggering system computation and communication. Instead the cores and the network reside in different clock domains. We assume that the network itself is clocked by a single clock, which has the same phase as the core clocks. The core frequencies can be different from each other. The communication behavior of the cores and network can be modeled in their own clock domains following the synchronous model. But the cross-domain time structures must be arbitrated.

- *Bandwidth-limited and delay-variant channel*: Although we can abstract the inter-process communication as logically a point-to-point service channel at the application layer, these channels share physical communication resources such as buffers and links in the session/transport and network layers. The service channel has a capacity limitation and in general introduces delay and delay variation (jitter). It provides in-order message delivery within a service channel, but there is no message ordering between service channels.

- *Conditional concurrency*: As a service channel is bandwidth-limited, it is impossible to send and receive arbitrary amount of data (any kind of data structure) within a fixed-length time window. The communication concurrency is restricted by available bandwidth. This limitation leads to conditional computation concurrency, i.e., computation concurrency is communication dependent.

In the implementation domain, we must introduce a communication process in order to glue a signal to a service channel. The communication elements in the implementation domain are *communication processes*, *streams* and *service channels*. Streams are ordered sequences of messages. The elementary communication processes either generate messages by consuming events or produce events by consuming messages. Service channels are where the streams are transported. The computation processes must be stallable if the required input tokens for computation are not available. We illustrate the three communication elements with the two computation processes, P_1 and P_2, in Figure 4.1b.

As we can observe from the above analysis, the ideal communication in the synchronous model does not exist at all in the implementation domain. The immediate questions to answer while refining the synchronous communication into NoC communication are:

- How to compromise global synchrony into multiple-clock synchrony?
- How to refine ideal communication into shared communication?
- How to refine fully concurrent computation and communication into conditionally concurrent computation and communication?
- How to satisfy performance constraints and communication properties?
- How to make a good utilization of network resources during the refinement?

These questions may not be addressed in isolation because they are inherently correlated. For example, refining ideal communication into shared communication results in reducing the concurrency level in the system model. Throughout the communication refinement, the system behavior can not be changed. Maintaining the correct system behavior during the refinement is the first concern because of the violation of the ideal communication assumption in the implementation domain. This task is burdened with the requirements of satisfying constraints such as processing n samples/second and of not over-dimensioning the underlying network.

4.2.2 Refinement Overview

In Papers 8 and 9, we have proposed a three-step approach for the NoC communication refinement problem. Paper 8 mainly focuses on refinement for correctness and Paper 9 for performance and efficiency. In this section, we unify the concepts presented in the two papers in order to give a coherent view of the proposed communication refinement approach.

Our refinement approach consists of three steps, namely, *channel refinement*, *process refinement* and *communication mapping*, as illustrated in Figure 4.2.

Step 1. *Channel refinement* (Section 4.2.3): An ideal channel is refined into a service channel. The service channel models the characteristics of the underlying network communication service. We also model the interfaces between different clock domains assuming that the network resides in a clock domain different from cores.

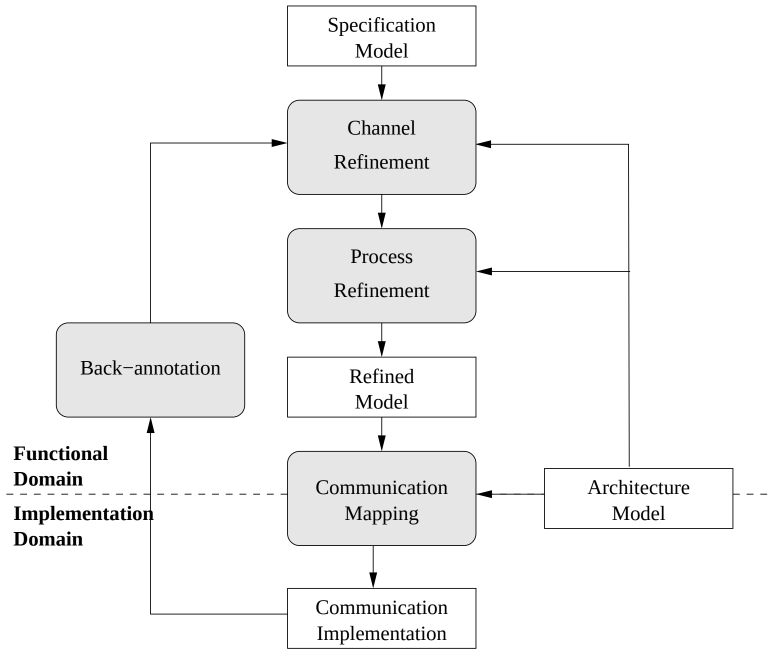


Figure 4.2. NoC communication refinement

Step 2. *Process refinement* (Section 4.2.4): In the specification model, a process may be viewed as only performing functionality since communication is ideal (unlimited bandwidth and zero-delay). This will not be the case once the ideal channel is replaced by a service channel. To reuse the original computational process, we leave it untouched. But we need to encapsulate the process with a *communication process*. This communication process (a) interfaces with the service channel; (b) fulfills the computation synchronization requirement of the process, which we call *process synchronization property*. This synchronization property must be consistent during the refinement. With the introduction of the service channel, the process cannot fire automatically with the clock. Instead, it requires additionally a *synchronization ready* signal from the communication process to control its execution; (c) satisfies communication property and performance constraints by refining communication protocols, for example, performing end-to-end flow control for reliability and overlapping computation with communication to hide the communication latency; (d) deals with feedback loops, if any. A zero-delay feedback loop is resolved by introducing an initial delay in the loop to break the cyclic dependency in our specification model. However, in implementation, it results in excessive delay when the feedback is looped through service channels. If the process sticks to this synchronization point, the feedback loop becomes a serious performance bottleneck.

Step 3. *Communication mapping* (Section 4.2.5): After the above two steps, we obtain a refined model. To further optimize for the use of shared network resources, two or multiple service channels (a) may be converged to share one implementation channel and (b) may be merged into one service channel so as to use one implementation channel. After the optimization, we move from the functional domain to the implementation domain. With a process-to-core mapping plan, we map the service channels and the communication processes in the refined and optimized model onto the NoC platform.

In Step 1, ideal channels are replaced by service channels, which involve multiple clock-domain communication. We use communication processes in Step 2a and Step 2b to refine the ideal communication into shared communication and to refine full-scale concurrency into conditional concurrency. Particularly, we have focused on Step 2b of synchronization consistency, which is proposed for correctness. Step 2c aims to satisfy communication property and performance requirements by communication protocol refinement. Step 2d deals with the feedback problem. It aims also to enhance system performance. In Step 3a and Step 3b, we consider channel-convergence and channel-merge to make efficient use of network resources. In summary, we have taken into account correctness, performance as well as resource utilization during the refinement.

4.2.3 Channel Refinement

A. The clock-domain interface

A synchronous model is very simplified in the sense that a single clock drives the system computation and communication. We assume that NoC communication is partitioned into multiple clock domains. While each clock domain is synchronous, the time structures of cross-domain communications have to be correctly arbitrated. Our assumption is that the cores and the network have their own clock domains. In addition, we assume that all clock phases are aligned.

To refine a single clock domain into multiple clock domains, we introduce clock sub-domains into the system's main domain. Each sub-domain is modeled synchronously, and a clock domain interface arbitrates the time structures of different clock domains. Introducing a synchronous sub-domain into the system model was presented in [115] where the event rate of the sub-domain is $\frac{1}{n}$ (n is a positive integer) of the main domain. The main domain is interfaced to the sub-domain by a single *down-sampling* process $P_{dn}(n)$. The sub-domain is interfaced back to the main domain by a single *up-sampling* process $P_{up}(n)$. We extend this work by considering a *generic domain interface* that connects a clock domain with event rate f_1 to another clock domain with event rate f_2. The simplest form of the fraction $\frac{f_1}{f_2}$ is $\frac{m}{n}$, where m and n are coprime.

The generic interface from domain f_1 to domain f_2 is constructed by using two processes as $I_{f_1 \to f_2} = P_{dn}(m) \circ P_{up}(n)$, where $\circ$ is the composition operator. The processes, $P_{up}(n)$ and $P_{dn}(m)$, are formally defined as follows:

$$P_{up}(n)(\{x_1, x_2, \dots\}) = \{\underbrace{\perp, \dots, \perp}_{n-1}, x_1, \underbrace{\perp, \dots, \perp}_{n-1}, x_2, \dots\}$$
$$P_{dn}(m)(\{\underbrace{x_1, x_2, \dots, x_m}_{m}, \underbrace{x_{m+1}, \perp, \dots, \perp}_{m}, \dots\}) = \{x_m, x_{m+1}, \dots\}$$

The *up-sampling* process $P_{up}(n)$ samples out n times of the input events, and does not result in event loss. The *down-sampling* process $P_{dn}(m)$ samples out $\frac{1}{m}$ times of the input events. At each down-sampling cycle, $m-1$ events are discarded and only the last token (non-absent value) is kept. The interface first does up-sampling and then down-sampling. If $f_1 \leq f_2$, no event drops, hence no token is lost. If $f_1 > f_2$, events are cyclically dropped. But this may or may not lead to the loss of tokens because the token rate may be less than the event rate. To guarantee that there is no data loss at the clock domain interface, the token rate of domain f_1 can not be faster than the event rate of domain f_2. This is to say, that a producer in domain f_1 can not use bandwidth (by generating tokens) more than the consumer domain's capacity. In our further analysis, we assume that this condition is satisfied and there is no data loss at the clock domain interfaces.

B. The service channel model

As we mentioned previously in the analysis, we consider a generic service channel that provides inter-process communication using message passing. In our context, we are interested in that different processes are distributed in different cores. Thus, an inter-process communication corresponds to an inter-core communication. Such kind of communication involves the session/transport layer and the network layer. As having discussed previously, we focus on the network services and simplify the session/transport layer effects.

A service channel is logically a simplex point-to-point channel, offering in-order, lossless and bounded-in-time communication between two end-processes. A service channel is mapped to a communication service in the underlying network. A basic distinction of network services is the guaranteed service and best-effort service. The guaranteed service requires the establishment of a virtual circuit before data transmission can start. Once a virtual circuit is set up, the message delivery is bounded in time. The best-effort service delivers messages as fast as possible. As long as the network does not drop packets and is free from deadlock and livelock, the delivery completion property is honored. Since no resources are pre-allocated, there is no guarantee on a delivery bound in general. This nondeterminism is due to that message delivery experiences dynamic contentions in the network. However, if such a bound does not exist, further analysis may be meaningless since the system performance becomes unpredictable. Therefore we assume that the best-effort service provides a delivery bound, but with an additional condition. The condition can be that the processes (the network clients) and the network interact on a contract basis. Processes inject traffic into the network in a controlled manner according to a traffic specification. Such a traffic specification may conform to, for example, the regulated (σ, ρ) flow model [22, 23]. The network performs a disciplined arbitration on resource sharing. In this way, the network saturation is avoided and the delivery bound can be derived. But, in our current analysis, this regulated traffic admission as well as traffic discipline has not been modeled. Instead we have assumed that such a scheme exists and even the best-effort network service can guarantee the bounded-in-time delivery.

With this assumption, we concentrate on considering the net effect of message delivery, i.e., the delay and its variation (jitter) by resorting to a stochastic approach. Formally, we develop a unicast service channel as a point-to-point *stochastic* channel: given an input stream of messages $\{m_1, m_2, \cdots, m_n\}$ to the service channel, the output stream is $\{d_1, m_1, d_2, m_2, \cdots, d_n, m_n\}$, where d_i denotes the delay of m_i ($i \in [1, n]$), which may be expressed as the number of absent ($\sqcup$) values and is subject to a distribution with a minimum d_{min} and maximum d_{max} value. The actual distribution, which may differ from channel to channel, is irrelevant here. We do not make any further assumptions about this. If $d_i = k$ (k is a positive integer), it means that there are k absent values between m_{i-1} and m_i. We can identify two important properties of the generic service channel: (1) d_i may be varying; (2) d_i is bounded. This behavior is purely viewed from the perspective of application processes and the implementation details are hidden.

Together with clock-domain interfaces, a service channel provides transparent communication for processes in different clock domains. Since the effect of clock-domain interfaces can be modeled by the delay distribution in a service channel, we do not explicitly consider them further.

4.2.4 Process Refinement

A. Interfacing with a service channel

Once an ideal channel is replaced by a service channel, the original process can not be directly connected to the service channel because a service channel uses a different data unit, *message*, and has limited bandwidth. A communication process must be introduced as an interface to connect the original process with the service channel. This communication process implements necessary data conversion and handshake-like control functionality, detailed as follows:

- *Data conversion*: The input/output data type of a service channel is a message that is of a bounded size. But a signal in the specification assumes an ideal data type, whose length is finite but arbitrary, e. g., a 32/64-bit integer, a 64-bit floating point or a user-defined 512-bit record type. Matching the data types requires data conversion, such as decomposition and composition.
- *Bandwidth-regulated control*: The service channel has limited bandwidth while a signal uses unlimited resources. The sending and receiving of messages using the service channel is subject to the available bandwidth. A control function is needed in the communication process to co-ordinate the message sending and receiving.

These adaptations are achieved by writer and reader processes. Specifically, to interface with the service channel, a producer needs to be wrapped with a *writer*, a consumer with a *reader*. As shown in Figure 4.1, P_1 is a producer and P_2 a consumer. C_1 implements the *writer* function and C_2 the *reader* function.

B. Synchronization consistency

Replacing the ideal channel (zero delay and unlimited bandwidth) with a stochastic channel (varying delay and limited bandwidth) leads to the violation of the synchrony hypothesis. Consequently, two synchronous events in the specification model may not have the same time tag any more because they may experience different delays in the service channels. Two synchronous signals in the specification model may no longer be synchronous. Furthermore, the synchronous system becomes globally asynchronous. This leads to possible behavior deviation from the specification. The entire system may not function properly. Correctness becomes the first issue to address in refinement.

We restrict our discussions to *continuous processes* [48]. Informally, we say a process *continuous* if, given partial input events, it generates partial output events.

In addition, adding more input events, more output events are generated but it will not affect the previously generated results. For a continuous process to work correctly, two conditions for delivering its input signals must be satisfied: (1) the event order of each signal must be maintained; (2) the synchronization requirement on the input events, called *process synchronization property*, must be satisfied before the process can fire. In the synchronous model, events are delivered in order and fully concurrent, the two conditions are satisfied cycle by cycle. However, with the NoC service channel model, the first condition is met but the second is not guaranteed. Our objective is to satisfy the process synchronization property, i.e., to maintain *synchronization consistency*. Our approach is to refine the system-level global synchronization into process-level local synchronization. We first classify the process synchronization properties and then use synchronizers to achieve synchronization consistency during refinement.

The synchronization in the system model requires all signals are synchronous and all processes are synchronous. This might over-specify the whole system, limiting implementation alternatives. We derive the synchronization property of a process according to its *evaluation conditions*. This is similar to firing rules that are used to discuss dataflow processes in [58]. By using evaluation conditions, we are able to decouple local computation synchrony from global computation synchrony. In effect, this refines the computation concurrency in the system model from being fully concurrent into being conditionally concurrent.

For a synchronous process with n input signals, PI is a set of N input patterns, $PI = \{I_1, I_2, \cdots, I_N\}$. The input patterns of a synchronous process describe its *firing rules*, which give the conditions of evaluating input events at each event cycle. I_i ($i \in [1, N]$) constitutes a set of event patterns, one for each of n input signals, $I_i = \{I_{i,1}, I_{i,2}, \cdots, I_{i,n}\}$. A pattern $I_{i,j}$ contains only one element that can be either a token wildcard $*$ or an absent value $\sqcup$, where $*$ does not include $\sqcup$. Based on the definition of firing rules, we define four levels of process synchronization properties as follows:

- *Strict synchronization*: All the input events of a process must be present before the process evaluates and consumes them. The only rule that the process can fire is $PI = \{I_1\}$ where $I_1 = \{[*], [*], \cdots, [*]\}$.

- *Nonstrict synchronization*: Not all the input events of a process are absent before the process can fire. The process can *not* fire with the pattern $I = \{[\sqcup], [\sqcup], \cdots, [\sqcup]\}$. This also includes cases where the process can not fire if one or more particular input events are $\sqcup$.

- *Strong synchronization*: All the input events of a process must be either present or absent in order to fire the process. The process has only two firing rules $PI = \{I_1, I_2\}$, where $I_1 = \{[*], [*], \cdots, [*]\}$ and $I_2 = \{[\sqcup], [\sqcup], \cdots, [\sqcup]\}$.

- *Weak synchronization*: The process can fire with any possible input patterns. For a 2-input process, its firing rules are $PI = \{I_1, I_2, I_3, I_4\}$ where $I_1 = \{[*], [*]\}$, $I_2 = \{[\sqcup], [\sqcup]\}$, $I_3 = \{[*], [\sqcup]\}$ and $I_4 = \{[\sqcup], [*]\}$.

Apparently, for processes with a strict or strong synchronization, their synchronization properties can not be satisfied if any of their input signals passes through a service channel since the delays via the channel are stochastic. Although globally asynchronous, the processes can be locally synchronized by using synchronization processes, called *synchronizers*, to satisfy their synchronization properties.

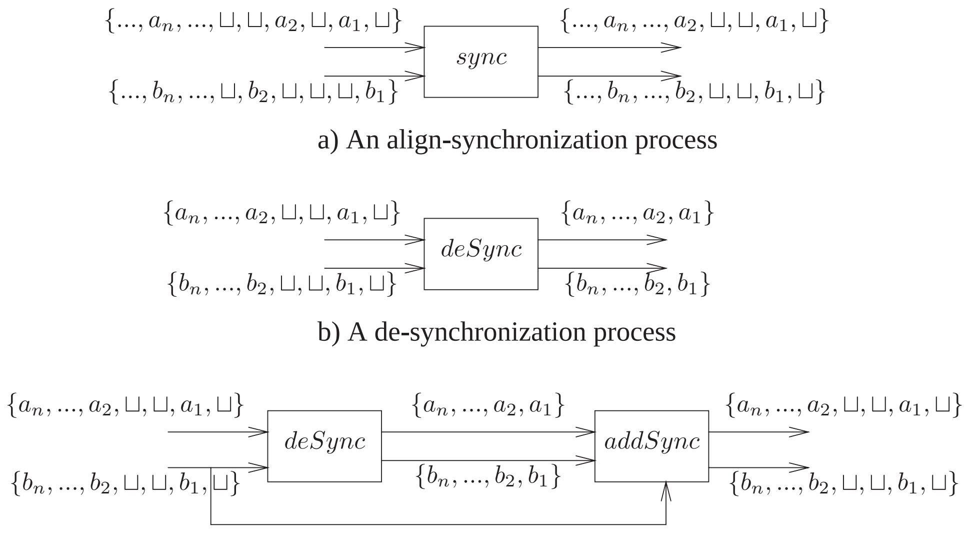

$\{..., a_n, ..., \sqcup, \sqcup, a_2, \sqcup, a_1, \sqcup\}$
$\{..., b_n, ..., \sqcup, b_2, \sqcup, \sqcup, \sqcup, b_1\}$
$sync$
$\{..., a_n, ..., a_2, \sqcup, \sqcup, a_1, \sqcup\}$
$\{..., b_n, ..., b_2, \sqcup, \sqcup, b_1, \sqcup\}$
a) An align-synchronization process
$\{a_n, ..., a_2, \sqcup, \sqcup, a_1, \sqcup\}$
$\{b_n, ..., b_2, \sqcup, \sqcup, b_1, \sqcup\}$
$deSync$
$\{a_n, ..., a_2, a_1\}$
$\{b_n, ..., b_2, b_1\}$
b) A de-synchronization process
$\{a_n, ..., a_2, \sqcup, \sqcup, a_1, \sqcup\}$
$\{b_n, ..., b_2, \sqcup, \sqcup, b_1, \sqcup\}$
$deSync$
$\{a_n, ..., a_2, a_1\}$
$\{b_n, ..., b_2, b_1\}$
$addSync$
$\{a_n, ..., a_2, \sqcup, \sqcup, a_1, \sqcup\}$
$\{b_n, ..., b_2, \sqcup, \sqcup, b_1, \sqcup\}$
c) An add-synchronization process

Figure 4.3. Processes for synchronization

We use a two-input process to illustrate these synchronizers in Figure 4.3. In the figure, we follow the direction of the signals and place the earlier events in the right side of a signal, i.e., $\{\cdots, x_n, \cdots, x_2, x_1\}$. An align-synchronization process *sync* aligns the tokens of its input events, as illustrated in Figure 4.3a. It does not change the time structure of the input signals. A desynchronizer *deSync* removes the absent values, as shown in Figure 4.3b. All its input signals must have the same token pattern, resembling the output signals of the *sync* process.

Removing absent values implies that the process is *stalled*. The desynchronizer changes the timing structure of the input signals, which must be recovered in order to prevent from causing unexpected behavior of other processes that use the timing information. An add-synchronizer *addSync* adds the absent values to recover the timing structure, as shown in Figure 4.3c. It must be used in relation to a *deSync* process. If the input events of the *deSync* is a token, the *addSync* reads one token from its internal buffers for each output signal; otherwise, it outputs a ⊔ event. The two processes *deSync* and *addSync* are used as a pair to assist processes to fulfill strictness.

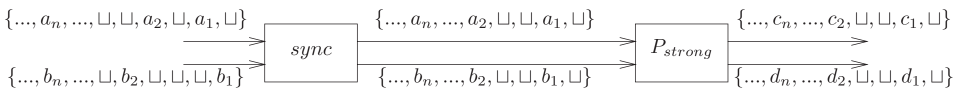

$\{..., a_n, ..., \sqcup, \sqcup, a_2, \sqcup, a_1, \sqcup\}$
$\{..., b_n, ..., \sqcup, b_2, \sqcup, \sqcup, \sqcup, b_1\}$
$sync$
$\{..., a_n, ..., a_2, \sqcup, \sqcup, a_1, \sqcup\}$
$\{..., b_n, ..., b_2, \sqcup, \sqcup, b_1, \sqcup\}$
P_{strong}
$\{..., c_n, ..., c_2, \sqcup, \sqcup, c_1, \sqcup\}$
$\{..., d_n, ..., d_2, \sqcup, \sqcup, d_1, \sqcup\}$

Figure 4.4. Wrap a strong process

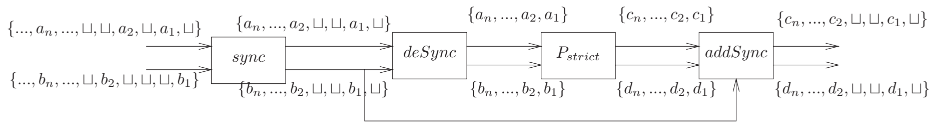

$\{..., a_n, ..., \sqcup, \sqcup, a_2, \sqcup, a_1, \sqcup\}$
$\{..., b_n, ..., \sqcup, b_2, \sqcup, \sqcup, \sqcup, b_1\}$
$sync$
$\{a_n, ..., a_2, \sqcup, \sqcup, a_1, \sqcup\}$
$\{b_n, ..., b_2, \sqcup, \sqcup, b_1, \sqcup\}$
$deSync$
$\{a_n, ..., a_2, a_1\}$
$\{b_n, ..., b_2, b_1\}$
P_{strict}
$\{c_n, ..., c_2, c_1\}$
$\{d_n, ..., d_2, d_1\}$
$addSync$
$\{c_n, ..., c_2, \sqcup, \sqcup, c_1, \sqcup\}$
$\{d_n, ..., d_2, \sqcup, \sqcup, d_1, \sqcup\}$

Figure 4.5. Wrap a strict process

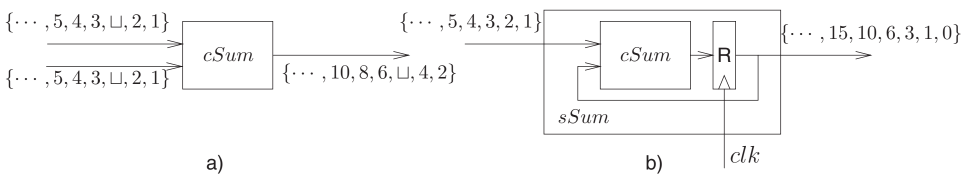

$\{\cdots, 5, 4, 3, \sqcup, 2, 1\}$
$\{\cdots, 5, 4, 3, \sqcup, 2, 1\}$
$cSum$
$\{\cdots, 10, 8, 6, \sqcup, 4, 2\}$
a)
$\{\cdots, 5, 4, 3, 2, 1\}$
$cSum$
R
$sSum$
$\{\cdots, 15, 10, 6, 3, 1, 0\}$
clk
b)

Figure 4.6. A strong and a strict process

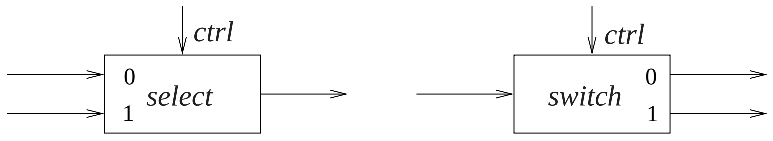


Figure 4.7. Two non-strict processes

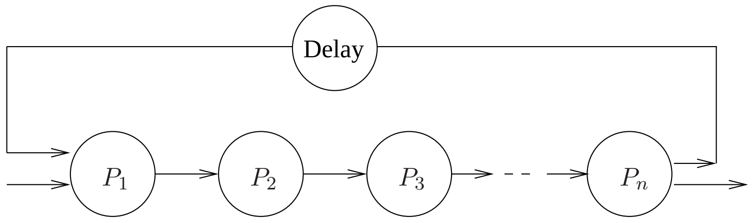
To achieve strong synchronization, we use an align-synchronization process *sync* to wrap a strong process, as shown in Figure 4.4. To achieve strict synchronization, we use three processes, *sync*, *deSync* and *addSync*, to wrap a strict process, as illustrated in Figure 4.5. A strong process is typically a *combinational* process, which is state-less. As long as both input tokens and delays are aligned, the delays on the input signals do not change the behavior of the process. For example, a combinational sum process $cSum$ in Figure 4.6a consumes two input events, one from each signal, adding them together. Delays on both input signals are tolerable as long as they are aligned. A strict process is typically a *sequential* process, which has states and thus is sensitive to the delay on its input signals. For instance, the sequential process $sSum$ in Figure 4.6b calculates the running sum of its input events by adding its state and the token value on the input signal. Its initial state is 0. Any delay on its input signal changes the output sequence. A non-strict process is often a control process, which can not fire if a control token is not available. For example, as shown in Figure 4.7, the *select* and *switch* processes can not fire if the control signal *ctrl* is neither 0 nor 1. Feeding control tokens is particularly important while refining non-strict processes. The refinement of processes with weak synchronization should be individually investigated. Practical examples of using synchronizers are given in Paper 8.

C. Protocol refinement

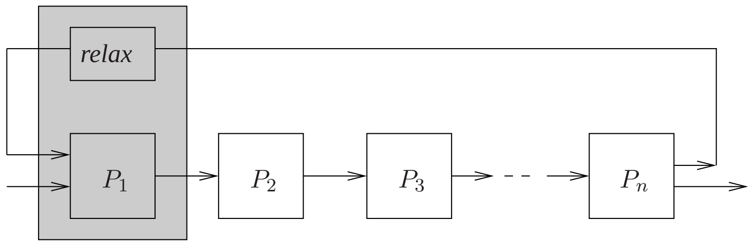
Message passing between a producer process and a consumer process is essentially a process of moving data from the producer side buffer to the consumer side buffer. This requires a co-ordination between the producer process and the consumer process in order to guarantee some properties, such as reliability, completion and buffer-overflow freedom. As the communication via a service channel introduces variable delay, it is important to overlap computation with communication in order to hide the communication latency. Protocol refinement is to refine the communication protocol for various reasons, for example, coordinated and improved communication. We have shown in Paper 9 that our refinement approach can formally incorporate different communication protocols in the step of process refinement to satisfy reliability and to improve throughput. For reliability, we have introduced acknowledgment in the protocol. For throughput, we have shown that data pipelines may be elaborated to hide communication latency and thus increase concurrency in computation and communication.

D. Feedback loop

Figure 4.8a illustrates a feedback loop in the specification. The loop passes through n processes, $P_1, P_2, \cdots, P_n$. In the synchronous model, we insert a single *Delay*



a) Specification model



b) Refined model

Figure 4.8. Feedback loop

process (can be viewed as a register) to break the zero-delay loop. In an implementation domain as shown in Figure 4.8b, even if one process has one cycle delay, it will take n cycles for the feedback signal to loop back. If process P_1 sticks to this synchronization point, the system throughput can never be faster than $1/n$ samples/cycle. A similar and even worse situation occurs in the NoC refinement case. If a feedback signal passes through a best-effort service channel, the delays are nondeterministic. Depending on the length of such a loop, the varying delays may be very long. If strictly observing the dependency, the process has to wait for the availability of the feedback events and cannot fire. The entire system is slowed down and becomes unpredictable due to the feedback loop.

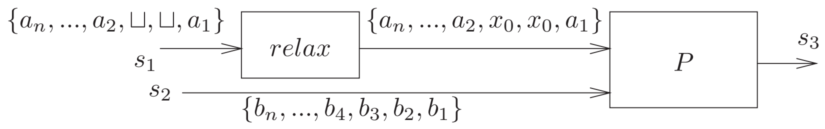

$\{a_n, ..., a_2, \sqcup, \sqcup, a_1\}$
s_1
$relax$
$\{a_n, ..., a_2, x_0, x_0, a_1\}$
P
s_3
s_2
$\{b_n, ..., b_4, b_3, b_2, b_1\}$

Figure 4.9. A relax-synchronization process

In our current proposal, we have used a *relaxed synchronization* method to force the synchronization satisfied. The synchronizer for this purpose is the relax-synchronization process *relax*, as illustrated in Figure 4.9. If the input event is a token, it outputs the token; otherwise, a token x_0 is emitted. The exact value of x_0 is application dependent. Note that relaxing synchronization is a design decision leading to behavior discrepancy between the specification model and the refined model. Care must be taken to validate the resulting system.

An example of using *relax* is given in Paper 8 where an equalizer regulates the volume of an input audio stream to protect the speaker by preventing the audio bass from exceeding a threshold. It analyzes the power spectrum of the resulting output audio stream after the regulation. In case a certain level is reached, it feeds a control signal back to adjust the amplification level of the audio bass amplifier. In the specification, the equalizer has an immediate response whenever surpassing the threshold occurs. However, after mapping the signals to best-effort service channels, the feedback signal takes long and nondeterminate delays to reach the amplifier. If the amplifier sticks to this over-specified synchronization point by following exactly the specification model, the equalizer may not be able to process enough audio samples per second. The system performance might become unacceptable. In this case, a *relax* is inserted to generate an amplification level when a token is not available. The equalizer can thus fulfill the requirement on throughput. The side-effect of this refinement is that the response to the audio-volume control buttons is delayed by some cycles. We can validate that the small amount of delay is acceptable for users. Therefore this design-decision is justified.

4.2.5 Communication Mapping

A. Channel convergence and channel merge

In the mapping phase, a refined channel, i.e., a service channel, is to be mapped to an implementation channel. A simple way of doing this is to map one service channel to one implementation channel. This one-to-one channel mapping may lead to inefficient use of network resources. For example, considering a guaranteed service using TDM virtual-circuits, for low-delay low-bandwidth traffic, bandwidth has to be reserved to satisfy the low-delay requirement. This results in extra time slots reserved but under-utilized. Therefore, an implementation channel should allow multiplexing, i.e., shared by more than one service channel, if possible. Considering these effects, the refined model can be further optimized for efficiency. In Paper 9, we have introduced *channel convergence* and *channel merge*. Channel convergence is for two or multiple service channels to share one implementation channel provided the implementation channel can support their total bandwidth requirement. Channel merge means that two or multiple service channels may be merged into one service channel so as to use one implementation channel by packing messages at the sender side and splitting them at the receiver side.

B. Channel and communication process mapping

The inputs to this task are the refined and optimized model as well as a process-to-core allocation scheme; the output is a communication implementation. We have used the NoC simulator *Semla* [125] as our implementation platform. The Application-Level Interface (ALI) is the set of message-passing primitives introduced in Chapter 3. The mapping stage involves *channel mapping* and *communication process mapping*.

With channel mapping, each pair of processes communicating via a service channel in the refined model results in its dedicated unicast implementation channel, which is mapped to the open channel primitive *open_channel()*.

In communication-process mapping, communication processes for interfacing service channels (*reader* and *writer*), maintaining synchronization consistency (synchronizers such as *sync*, *deSync*, *addSync* and *relax*), elaborating protocol and optimizing resource usage, are mapped onto cores. The original computation processes do not change, but their executions are controlled by their respective communication processes. Besides, with a single-thread implementation on a core, a static schedule has to be found to sequentialize the process executions and coordinate write and read operations [70]. With a multi-thread implementation on a core, processes may be dynamically fired when their synchronization requirements are met according to their evaluation conditions [121]. The reader and writer processes access the ALI by directly calling the communication primitives *read()* and *write()* defined in the NoC simulator. The resulting implementation is executable in the simulation framework.

4.3 Future Work

Our refinement proposal has sketched a way of refining synchronous communication into on-chip network-based communication. Through the refinement, correctness, constraint and efficiency have been taken into account. This approach has been validated in the ForSyDe framework and with our NoC simulator. The concept of synchronization consistency is independent of a particular communication implementation scheme. It can be applied to pure hardware, software and bus-based mixed hardware/software architectures. The proposed synchronizers have been implemented in hardware, software and mixed hardware/software [121].

To make our refinement approach fully-fledged, we realize that the research can be further carried on along the following three tracks:

- *Embed formal semantics in the refinement approach*: This is to give formal definitions for the synchronization issues and develop transformation rules for using synchronizers to represent and check the equivalence between the specification model and the refined model.

- *Conduct performance analysis and optimization*: The refined model allows us to derive performance figures. This requires that a stochastic process must be annotated with good-enough values. So far we just consider the effect of varying delay of the stochastic process. How to estimate the delay/jitter values and further system performance analysis are not addressed yet. Particularly, we will treat feedback using TDM virtual-circuits [34, 81] in order to obtain strong guarantees on delay and jitter bounds.

- *Automate the design flow*: With the well-defined synchronizers and well-controlled use of the synchronizers, automation is possible. Currently a process's synchronization property is annotated manually. But this can be done automatically. The reason is that, in the system model, a process can be defined using *pattern matching* evaluation [116], which nicely matches the process synchronization property. We are building synchronizer libraries in hardware, software and mixed hardware/software, and plan to develop programs that can automatically instantiate synchronizers for processes to maintain synchronization consistency. Optimization for performance and efficiency will be part of the automation.

Chapter 5

Summary

This chapter summarizes the thesis and outlines future directions.

5.1 Subject Summary

Moore's law has sustained in the semiconductor industry for 42 years. Following this law, the process technology has been ever-advancing. Meanwhile, the desire to exploit the technology capacity is ever-aggressive. However, the advancement of the chip capacity and the system integration capability is not evenly developed. The slower development of SoC integration is due to the extremely high level of complexity in system modeling, design, implementation and verification. As communication becomes a crucial issue, NoC is advocated as a systematic approach to address the challenges. NoC problems span the whole SoC spectrum in all domains at all levels. This thesis has focused on *on-chip network architectures*, *NoC network performance analysis*, and *NoC communication refinement*.

- Research on wormhole-switched networks has traditionally emphasized the flit delivery phase while simplifying flit admission and ejection. We have initiated investigation of these issues. It turns out that different flit-admission and flit-ejection models have quite different impact on cost, performance and power. In a classical wormhole switch architecture, we propose the coupled flit-admission and p-sink flit-ejection models. These optimizations are simple but effective. The coupled admission significantly reduces the crossbar complexity. Since the crossbar consumes a large portion of power in the switch, this adjustment is beneficial in both cost and power. The network performance, however, is not sensible to the adjustment before the network reaches the saturation point. The p-sink model has a direct impact on decreasing buffering cost, and has negligible impact on performance before network saturation. As the support for one-to-many communication is necessary, we design a multicasting protocol and implement it in a wormhole-switched network. This multicast service is connection-oriented and QoS-aware. For the TDM virtual-circuit configuration, we utilize the generalized logical-network concept and develop theorems to guide the construction of contention-free virtual circuits. Moreover, we employ a back-tracking algorithm to explore the path diversity and systematically search for feasible configurations.

- On-chip networks expose a much larger design space to explore when compared with buses. The existence of a lot of design considerations at different layers leads to making design decisions difficult. As a consequence, it is desirable to explore these alternatives and to evaluate the resulting networks extensively. We have proposed traffic representation methods to configure various workload patterns. Together with the choices of the traffic configuration parameters, the exploration of the network design space can be conducted in our network simulation environment. We have suggested a contention-tree model which can be used to approximate network contentions. Using this model and its associated scheduling method, we develop a feasibility analysis test in which the satisfaction of timing constraints for real-time messages can be evaluated through an estimation program.

- As communication is taking the central role in a design flow, how to refine an abstract communication model onto on-chip network-based communication platform is an open problem. Starting from a synchronous specification, we have formulated the problem and proposed a refinement approach. This refinement is oriented for correctness, performance and resource usage. Correct-by-construction is achieved by maintaining synchronization consistency. We have also integrated the refinement of communication protocols in our approach, thus satisfying performance requirements. By composing and merging communication tasks of processes to share the underlying implementation channels, the network utilization can be improved.

5.2 Future Directions

With only a short history, Network-on-Chip (NoC) has become a very active research field. Looking into the future, we believe that NoC will continue to be vivid.

We list some key issues that have not been sufficiently addressed or emphasized in the community as follows:

- *Heterogeneous modeling*: In current SoC design flows, a number of modeling techniques have been used ranging from sequential to concurrent models, from untimed to timed models. Application complexity and heterogeneity have driven the need to model a system using heterogeneous models. The Ptolemy project [60] is such an example. This is particularly true for NoC since it also targets highly complex and heterogeneous applications. To which extent to model the underlying architecture characteristics is one issue. While a model itself does not necessarily reflect the detailed characteristics, refinement may be facilitated if the architecture characteristics such as concurrency, time and adaptivity are captured in the model properly. Since there does not exist a one-size-fits-all Model-of-Computation (MoC), multi-MoC modeling will be highly necessary. Based on our understanding on the various MoCs, one challenge is the cross-MoC-domain modeling, i.e., from untimed domain to timed domain, from discrete time to continuous time, from a sequential model to a concurrent model, and vice versa. The follow-up challenges include multi-MoC refinement, synthesis, and verification.

- *Programmability*: To reduce cost, making a NoC *soft* is essential. This requires the support of operating systems that offer various services such as I/O handling, memory management, system monitoring, process scheduling and migration, and inter-process communication, and provide programming models balancing ease-of-programming and efficiency. Efficient application-level interfaces and standardized core-level interfaces are the *hard* part. As NoC is a distributed (not centralized) system in nature, investigating parallel computing models beyond von Neumann models for NoC systems to achieve high performance will become *hot*. For example, the MultiFlex system [101] supports an object-oriented message passing model.

- *Composability*: To build complex systems, we are moving away from creating individual components from scratch towards methodologies that emphasize composition of re-usable components via the network paradigm. NoC systems should allow one to plug new validated components and upgrade old components with linear design efforts and without compromising performance, reliability and verifiability. This feature makes a NoC easy-to-integrate and easy-to-extend, leveraging the reuse to the system level and shrinking the time-to-market.

- *Autonomy*: There are several reasons to hope for an autonomous NoC. A nano-chip is an extremely condensed device where transient and permanent faults on wires and nodes are increasingly possible. Power consumption is workload-dependent and performance-sensible. System optimization involves the re-organization and orchestration of its computation and communication components to tradeoff power and performance and to balance the thermal distribution on the chip. These reliability, performance, power and thermal issues call for an intelligent way like human self-healing, self-vaccinating and self-adjusting systems to dynamically and autonomously adapt the NoC to suit its application demands and operating environments. Along this thread, a simulation tool may be aimed to be *intelligent* in, for example, pinpointing performance bottlenecks and suggesting hints on buffer dimensioning.

- *Design flow integration*: Present design flows for SoC/NoC are not seamlessly integrated. From application specification down to chip fabrication, there exist a number of concerns from physical issues (electrical and thermal), clocking, power, performance, verification, manufacturability and testability. A design flow usually targets one or a small subset of the design aspects. To enable a truly automated design flow, all relevant issues are preferably handled in an integrated design flow to leverage efficiency and overcome the inconsistency between different tools which may come from different vendors.

Technically, NoC has a huge potential to expand. It would come no surprise when yesterday's 1000-node supercomputers become tomorrow's 1000-node networks-on-chips. In addition, NoC will be driven not only for application-specific applications but also for general-purpose applications. Finally, SoC/NoC technology will be combined with other technologies, such as sensor-technology, nanochemistry, biotechnology, micro-mechanics etc., into a multi-disciplinary technology. Innovative application domains will be further inspired by the needs of improving our life quality such as health care, entertainment, safety, information production and exchange, non-restricted communications and of improving our living, developing and ecological environment.

References

[1] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino. SPIN: A scalable, packet switched, on-chip micro-network. In *Design, Automation and Test in Europe Conference and Exhibition - Designers' Forum*, March 2003.

[2] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[3] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger. Clock rate versus IPC: the end of the road for conventional microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 248 – 259, 2000.

[4] A. Allan, D. Edenfeld, J. W. Joyner, A. B. Kahng, M. Rodgers, and Y. Zorian. 2001 technology roadmap for semiconductors. *IEEE Computer*, 35(1):42–53, January 2002.

[5] D. Andreasson and S. Kumar. Slack-time aware routing in NoC systems. In *IEEE International Symposium on Circuits and Systems*, May 2005.

[6] ARM. AMBA advanced extensible interface (AXI) protocol specifcation, version 1.0. http://www.amba.com, 2004.

[7] S. Balakrishnan and F. Özgüner. A priority-driven flow control mechanism for real-time traffic in multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems*, 9(7):664–678, July 1998.

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

[9] L. Benini and G. D. Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–78, January 2002.

[10] L. Benini and G. D. Micheli, editors. *Networks on Chips: Technology and Tools*. Morgan Kaufmann, 2006.

[11] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, 1991.

[12] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, and R. D. Simone. The synchronous languages 12 years later. *Proceedings of The IEEE*, 91(1):64–83, January 2003.

[13] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Survey*, 38(1):1–54, 2006.

[14] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 1226–1231, 2005.

[15] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *The Journal of Systems Architecture*, December 2003.

[16] J. T. Brassil and R. L. Cruz. Bounds on maximum delay in networks with deflection routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(7):724–732, July 1995.

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

[18] S. Chalasani and R. V. Boppana. Fault-tolerant wormhole routing algorithms for mesh networks. *IEEE Transactions on Computers*, 44(7):848–864, 1995.

[19] T. Claasen. An industry perspective on current and future state-of-the-art in system-on-chip (SoC) technology. *Proceedings of the IEEE*, 94(6):1121–1137, June 2006.

[20] M. Coppola, S. Curaba, M. Grammatikakis, and G. Maruccia. IPSIM: SystemC 3.0 enhancements for communication refinement. In *Proceedings of Design Automation and Test in Europe*, 2003.

[21] M. Coppola, S. Curaba, M. Grammatikakis, and G. Maruccia. OCCN: A network-on-chip modeling and simulation framework. In *Proceedings of Design Automation and Test in Europe*, 2004.

[22] R. L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[23] R. L. Cruz. A calculus for network delay, part II: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991.

[24] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini. Xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor SoCs. In *Proceedings of the 21st International Conference on Computer Design*, September 2003.

[25] W. J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–204, March 1992.

[26] W. J. Dally and C. L. Seitz. The torus routing chip. *Journal of Distributed Computing*, 1(3):187–196, 1986.

[27] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference*, 2001.

[28] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufman Publishers, 2004.

[29] D. Densmore, R. Passerone, and A. Sangiovanni-Vincentelli. A platform-based taxonomy for ESL design. *IEEE Design and Test of Computers*, 23(5):359– 374, September-October 2006.

[30] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Network - An Engineering Approach*. IEEE Computer Society Press, 1997.

[31] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli. Design of embedded system: Formal models, validation and synthesis. *Proceedings of the IEEE*, 85(3):366–390, March 1997.

[32] A. Gerstlauer, S. Dongwan, R. Domer, and D.D.Gajski. System-level communication modeling for network-on-chip synthesis. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 45–48, 2005.

[33] K. Goossens, J. Dielissen, J. Meerbergen, P. Poplavko, A. Rădulescu, E. Rijpkema, E. Waterlander, and P. Wielage. *Networks on Chip*, chapter Guaranteeing The Quality of Services. Kluwer Academic Publisher, 2003.

[34] K. Goossens, J. Dielissen, and A. Rădulescu. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):21–31, Sept-Oct 2005.

[35] A. G. Greenberg and J. Goodman. Sharp approximate models of deflection routing in mesh networks. *IEEE Transactions on Communications*, 41(1):210–223, January 1993.

[36] D. Gross and C. M. Harris. *Fundamentals of Queueing Theory*. Wiley, 1998.

[37] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 250–256, March 2000.

[38] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, 1993.

[39] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[40] S. L. Harry and F. Özgüner. Feasibility test for real-time communication using wormhole routing. *IEE Proceedings of Computers and Digital Techniques*, 144(5):273–278, September 1997.

[41] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg, and D. Lindqvist. Network on chip: An architecture for billion transistor era. In *Proceeding of the IEEE NorChip Conference*, November 2000.

[42] W. D. Hills. The Connection machine. *Scientific American*, 256(6), June 1987.

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

[44] J. Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *Proceedings of the Design Automation and Test in Europe Conference*, 2003.

[45] IBM. CoreConnect bus architecture - A 32-, 64-, 128-bit core on-chip bus structure. http://www-03.ibm.com/chips/products/coreconnect/.

[46] ITRS. International technology road map for semiconductors 2004 update: Design, 2004, wwww.itrs.net.

[47] A. Iyer and D. Marculescu. Power and performance evaluation of globally asynchronous locally synchronous processors. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 158–168, 2002.

[48] A. Jantsch. *Modeling Embedded Systems and SoCs*. Morgan Kaufmann Publishers, 2004.

[49] A. Jantsch. Models of computation for networks on chip. In *Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, June 2006.

[50] A. Jantsch and H. Tenhunen, editors. *Networks on Chip*. Kluwer Academic Publisher, 2003.

[51] G. Kahn. The semantics of a simple language for parallel programming. *Information Processing*, 1974.

[52] F. Karim, A. Nguyen, and S. Dey. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5):36–45, Sep/Oct 2002.

[53] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, January 1979.

[54] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transaction on Computer-Aided Design of Integrated Circuits*, 19(12):1523–1543, December 2000.

[55] B. Kim, J. Kim, S. Hong, and S. Lee. A real-time communication method for wormhole switching networks. In *Proceedings of International Conference on Parallel Processing*, pages 527–534, Aug. 1998.

[56] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, K. Tiensyrjä, and A. Hemani. A network-on-chip architecture and design methodology. In *IEEE Computer Society Annual Symposium on VLSI*, 2002.

[57] S. K. Kunzli, F. Poletti, L. Benini, and L. Thiele. Combining simulation and formal methods for system-level performance analysis. In *Proceedings of Design, Automation and Test in Europe Conference*, pages 1–6, March 2006.

[58] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–799, May 1995.

[59] E. A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, December 1998.

[60] E. A. Lee and Y. Xiong. A behavioral type system and its application in Ptolemy II. *Formal Aspects of Computing*, August 2004.

[61] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, and D. Verkest. Spatial division multiplexing: a novel approach for guaranteed throughput on NoCs. In *Proceedings of the 3rd International Conference on Hardware/-Software Codesign and System Synthesis*, pages 81–86, 2005.

[62] J.-P. Li and M. W. Mutka. Real-time virtual channel flow control. *Journal of Parallel and Distributed Computing*, 32(1):49–65, 1996.

[63] P. Lieverse, T. Stefanov, P. van der Wolf, and E. Depretter. System level design with SPADE: an M-JPEG case study. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2001.

[64] Z. Lu and R. Haukilahti. *Networks on Chip*, chapter NoC Application Programming Interfaces. Kluwer Academic Publisher, 2003.

[65] Z. Lu and A. Jantsch. TDM virtual-circuit configuration in network-on-chip using logical networks. *In submission to IEEE Transactions on Very Large Scale Integration Systems*.

[66] Z. Lu and A. Jantsch. Flit admission in on-chip wormhole-switched networks with virtual channels. In *Proceedings of International Symposium on System-on-Chip (ISSoC'04)*, pages 21–24, Tampere, Finland, November 2004.

[67] Z. Lu and A. Jantsch. Flit ejection in on-chip wormhole-switched networks with virtual channels. In *Proceedings of the IEEE Norchip Conference (Norchip'04)*, pages 273–276, Oslo, Norway, November 2004.

[68] Z. Lu and A. Jantsch. Traffic configuration for evaluating networks on chips. In *Proceedings of the 5th International Workshop on System on Chip for Real-time applications (IWSOC'05)*, pages 535–540, July 2005.

[69] Z. Lu, A. Jantsch, and I. Sander. Feasibility analysis of messages for on-chip networks using wormhole routing. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC'05)*, pages 960–964, Shanghai, China, January 2005.

[70] Z. Lu, I. Sander, and A. Jantsch. A case study of hardware and software synthesis in ForSyDe. In *Proceedings of the 15th International Symposium on System Synthesis (ISSS'02)*, pages 86–91, Kyoto, Japan, October 2002.

[71] Z. Lu, I. Sander, and A. Jantsch. *Applications of Specification and Design Languages for SoCs - Selected papers from FDL 2005*, chapter Refining synchronous communication onto network-on-chip best-effort services, pages 23–38. Springer, 2006.

[72] Z. Lu, I. Sander, and A. Jantsch. Towards performance-oriented pattern-based refinement of synchronous models onto NoC communication. In *Proceedings of the 9th Euromicro Conference on Digital System Design (DSD'06)*, pages 37–44, Dubrovnik, Croatia, August 2006.

[73] Z. Lu, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch. NNSE: Nostrum network-on-chip simulation environment. In *The University Booth Tool-Demonstration Program of the Design, Automation and Test in Europe Conference*, March 2005.

[74] Z. Lu, L. Tong, B. Yin, and A. Jantsch. A power-efficient flit-admission scheme for wormhole-switched networks on chip. In *Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics*, July 2005.

[75] Z. Lu, B. Yin, and A. Jantsch. Connection-oriented multicasting in wormhole-switched networks on chip. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 205–210, Karlsruhe, Germany, March 2006.

[76] Z. Lu, M. Zhong, and A. Jantsch. Evaluation of on-chip networks using deflection routing. In *Proceedings of the 16th ACM Great Lakes Symposium on VLSI (GLSVLSI'06)*, pages 296–301, Philadelphia, USA, May 2006.

[77] J. Madsen, S. Mahadevan, K. Virk, and M. Gonzalez. Network-on-chip modeling for system-level multiprocessor simulation. In *International Real-Time Systems Symposium*, 2003.

[78] S. Mahadevan, F. Angiolini, M. Storgaard, R. Olsen, J. Sparsø, and J. Madsen. A network traffic generator model for fast network-on-chip simulation. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 780–785, March 2005.

[79] G. Martin and H. Chang, editors. *Winning the SoC Revolution*. Kluwer Academic Publishers, 2003.

[80] J. W. McPherson. Reliability challenges for 45nm and beyond. In *Proceedings of the 43rd Design Automation Conference*, pages 176– 181, July 2006.

[81] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proceedings of the Design Automation and Test in Europe Conference*, February 2004.

[82] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum backbone - a communication protocol stack for networks on chip. In *Proceedings of the VLSI Design Conference*, Mumbai, India, January 2004.

[83] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(5), 1965.

[84] F. G. 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*, 38(1):69–93, 2004.

[85] T. Mudge. Power: A first-class architectural design constraint. *IEEE Computer*, 34(4):52–58, April 2001.

[86] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[87] M. D. Nava, P. Blouet, P. Teninge, M. Coppola, T. Ben-Ismail, S. Picchiottino, and R. Wilson. An open platform for developing multiprocessor SoCs. *IEEE Computer*, 38(7):60–67, July 2005.

[88] C. Neeb, M. Thul, and N. Wehn. Network-on-chip-centric approach to interleaving in high throughput channel decoders. In *IEEE International Symposium on Circuits and Systems*, pages 1766–1769, 2005.

[89] K.-H. Nielsen. Evaluation of real-time performance models in wormhole-routed on-chip networks. Master's thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, Sweden, 2005.

[90] E. Nilsson, M. Millberg, J. Öberg, and A. Jantsch. Load distribution with the proximity congestion awareness in a network on chip. In *Proceedings of the Design Automation and Test in Europe Conference*, 2003.

[91] E. Nilsson and J. Öberg. Reducing peak power and latency in 2D mesh NoCs using globally pseudochronous locally synchronous clocking. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, September 2004.

[92] V. Nollet, M. Marescaux, and D. Verkest. Operating-system controlled network on chip. In *Proceedings of the 41st Design Automation Conference*, pages 256–259, Los Alamitos, CA, USA, 2004.

[93] J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors. *Interconnect-Centric Design for Advanced SoCs and NoCs*. Kluwer Academic Publisher, 2004.

[94] J. Öberg. *Networks on Chip*, chapter Clocking Strategies for Networks on Chip. Kluwer Accademic Publisher, 2003.

[95] OCP International Partnership. Open core protocol specification, version 2.0. http://www.ocpip.org, 2003.

[96] U. Y. Ogras, R. Marculescu, H. G. Lee, and N. Chang. Communication architecture optimization: making the shortest path shorter in regular networks-on-chip. In *Proceedings of Design, Automation and Test in Europe Conference*, March 2006.

[97] M. Palesi, R. Holsmark, and S. Kumar. A methodology for design of application specific deadlock-free routing algorithms for NoC systems. In *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis*, pages 142–147, October 2006.

[98] D. Pamunuwa, J. Öberg, L.-R. Zheng, M. Millberg, A. Jantsch, and H. Tenhunen. A study on the implementation of 2D mesh based networks on chip in the nanoregime. *Integration - The VLSI Journal*, 38(2):3–17, October 2004.

[99] 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*, 54(8):1025–1040, August 2005.

[100] K. Park and W. Willinger, editors. *Self-Similar Network Traffic and performance Evaluation*. New York: Wiley, 2000.

[101] P. G. Paulin, C. Pilkington, M. Langevin, E. Bensoudane, D. Lyonnard, O. Benny, B. Lavigueur, D. Lo, G. Beltrame, V. Gagne, and G. Nicolescu. Parallel programming models for a multiprocessor SoC platform applied to networking and multimedia. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(7):667–680, July 2006.

[102] L. S. Peh and W. J. Dally. A delay model for router microarchitectures. *IEEE Micro*, 21(1):26–34, Jan.-Feb. 2001.

[103] S. G. Pestana, E. Rijpkema, A. Radulescu, K. Goossens, and O. P. Gangwal. Cost-performance trade-offs in networks on chip: A simulation-based approach. In *Proceedings of the Design, Automation and Test in Europe Conference*, 2004.

[104] Philips Semiconductors. Device transaction level (DTL) protocol specification, version 2.2, 2002.

[105] V. Raghunathan, M. B. Srivastava, and R. K. Gupta. A survey of techniques for energy efficient on-chip communication. In *Proceedings of Design Automation Conference*, June 2003.

[106] R. Ramaswami and K. N. Sivarajan. *Optical Networks: A Practical Perspective*. Morgan Kaufmann Publishers, 1998.

[107] P. Rashinkar, P. Paterson, and L. Singh. *System-On-A-Chip Verification: Methodology and Techniques*. Kluwer Academic Publishers, 2001.

[108] T. Raudvere, I. Sander, A. K. Singh, and A. Jantsch. Verification of design decisions in ForSyDe. In *Proceedings of CODES+ISSS*, California, USA, October 2003.

[109] E. Rijpkema, K. Goossens, and P. Wielage. A router architecture for networks on silicon. In *Proceedings of Progress 2001, 2nd Workshop on Embedded Systems*, Veldhoven, The Netherlands, Oct. 2001.

[110] E. Rijpkema, K. G. W. Goossens, A. Rădulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *Proceedings of Design Automation and Test in Europe Conference*, Mar. 2003.

[111] C. Rowen. *Engineering the Complex SoC*. Prentice Hall PTR, 2004.

[112] J. A. Rowson and A. Sangiovanni-Vincentelli. Interface based design. In *Proceedings of the 34th Design Automation Conference*, 1997.

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

[114] E. Salminen, T. Kangas, V. Lahtinen, J. Riihimäki, K. Kuusilinna, and T. D. Hämäläinen. Benchmarking mesh and hierarchical bus networks in system-on-chip context (in press). *Journal of System Architectures*, 2007.

[115] I. Sander and A. Janstch. Transformation based communication and clock domain refinement for system design. In *Proceedings of the 39th Design Automation Conference*, pages 281 – 286, June 2002.

[116] I. Sander and A. Jantsch. System modeling and transformational design refinement in ForSyDe. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(1):17–32, February 2004.

[117] I. Sander, A. Jantsch, and Z. Lu. Development and application of design transformations in ForSyDe. In *Proceedings of Design, Automation and Test in Europe Conference*, pages 364–369, Munich, Germany, March 2003.

[118] M. Sgroi, L. Lavagno, and A. Sangiovanni-Vincentelli. Formal models for embedded system design. *IEEE Design & Test of Computers*, pages 2–15, April-June 2000.

[119] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vencentelli. Addressing the system-on-a-chip interconnect

woes through communication-based design. In *Proceedings of the 38th Design Automation Conference*, 2001.

[120] K. G. Shin and S. W. Daniel. Analysis and implementation of hybrid switching. In *Proc. of the 22nd International Symposium on Computer Architecture*, pages 211–219, 1995.

[121] J. Sicking. Implementation of asynchronous communication for ForSyDe in hardware and software. Master's thesis, Royal Institute of Technology, Sweden, IMIT/LECS/[2005-73].

[122] D. Siguenza-Tortosa, T. Ahonen, and J. Nurmi. Issues in the development of a practical NoC: the Proteo concept. *Integration, the VLSI Journal*, 38(1):95–105, 2004.

[123] K. Srinivasan, K. S. Chatha, and G. Konjevod. Linear programming based techniques for synthesis of network-on-chip architectures. *IEEE Transactions on VLSI Systems*, 14(4):407–420, 2006.

[124] R. Thid. A network on chip simulator. Master's thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, Sweden, 2002.

[125] R. Thid, M. Millberg, and A. Jantsch. Evaluating NoC communication backbones with simulation. In *Proceedings of the IEEE NorChip Conference*, November 2003.

[126] R. Thid, I. Sander, and A. Jantsch. Flexible bus and NoC performance analysis with configurable synthetic workloads. In *Proceedings of the 9th Euromicro Conference on Digital System Design*, August 2006.

[127] S. Thompson. *Haskell - The Craft of Functional Programming*. Addison-Wesley, 2 edition, 1999.

[128] A. S. Vaidya, A. Sivasubramaniam, and C. R. Das. Impact of virtual channels and adaptive routing on application performance. *IEEE Transactions on Parallel and Distributed Systems*, 12(2):223–237, 2001.

[129] P. van der Wolf, E. de Kock, T. Henriksson, W. Kruijtzer, and G. Essink. Design and programming of embedded multiprocessors: an interface-centric approach. In *International Conference on Hardware/Software Codesign and System Synthesis*, pages 206–217, 2004.

[130] VSI Alliance. Virtual component interface, standard version 2. http://www.vsi.org, 2000.

[131] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In *Proceedings of the 35th International Symposium on Microarchitecture (MICRO)*, November 2002.

[132] D. Wiklund and D. Liu. SoCBUS: Switched network on chip for hard real time embedded systems. In *International Parallel and Distributed Processing Symposium*, 2003.

[133] J. Xu, W. Wolf, J. Henkel, S. Chakradhar, and T. Lv. A case study in networks-on-chip design for embedded video. In *Proceedings of the Design Automation and Test in Europe Conference*, 2004.

[134] L.-R. Zheng. *Design, Analysis and Integration of Mixed-Signal Systems for Signal and Power Integrity*. PhD thesis, Royal Institute of Technology, 2001.

[135] H. Zimmermann. OSI Reference Model – the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980.