Using Wormhole Switching for Networks on Chip: Feasibility Analysis and Microarchitecture Adaptation

Zhonghai Lu

Stockholm 2005

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

Lu, Zhonghai

Using Wormhole Switching for Networks on Chip: Feasibility Analysis and Microarchitecture Adaptation

TRITA-IMIT-LECS AVH 05:05
ISSN 1651-4076
ISRN KTH/IMIT/LECS/AVH-05/05--SE

Royal Institute of Technology
Department of Microelectronics and Information Technology
Laboratory of Electronics and Computer Systems
Electrum 229
S-164 40 Kista, Sweden

Abstract

Network-on-Chip (NoC) is proposed as a systematic approach to address future System-on-Chip (SoC) design difficulties. Due to its good performance and small buffering requirement, wormhole switching is being considered as a main network flow control mechanism for on-chip networks. Wormhole switching for NoCs is challenging from NoC application design and switch complexity reduction.

In a NoC design flow, mapping an application onto the network should conduct a feasibility analysis in order to determine whether the messages' timing constraints can be satisfied, and whether the network can be efficiently utilized. This is necessary because network contentions lead to nondeterministic behavior in message delivery. For wormhole-switched networks, we have formulated a contention tree model to accurately capture network contentions and reflect the concurrent use of links. Based on this model, the timing bounds of real-time messages can be derived. Furthermore, we have developed an algorithm to test the feasibility of real-time messages in the networks.

From the wormhole switch micro-architecture level, switch complexity should be minimized to reduce cost but with reasonable performance penalty. We have investigated the *flit admission* and *flit ejection* problems that concern how the flits of packets are *admitted into* and *ejected from* the network, respectively. For flit admission, we propose a novel coupling scheme which binds a flit-admission queue with an output physical channel. Our results show that this scheme achieves a reduction of up to 8% in switch area and up to 35% in switch power over other comparable solutions. For flit ejection, we propose a p-sink model which differs from a typical ideal ejection model in that it uses only p flit sinks to eject flits instead of $p \cdot v$ flit sinks as required by the ideal model, where p is the number of physical channels of a switch and v is the number of virtual channels per physical channel. With this model, the buffering cost of flit sinks only depends on p, i.e., is irrespective of v. We have evaluated the coupled flit-admission technique and p-sink model in a 2D 4×4 mesh network. In our experiments, they exhibit only limited performance penalties in some cases. We believe that these cost-effective models are promising candidates to be used in wormhole-switched on-chip networks.

Acknowledgments

I would like to express my deep gratitude to my supervisor, Prof. Axel Jantsch. His expertise in electronic system design has helped me position my research topics and enlighten my understandings. He has created a very pleasant research environment for his students to explore personal potentials. He is always an encouraging supervisor whom you can receive firm support from.

I am grateful to Dr. Ingo Sander. He supervised my Master thesis and now co-supervises my Ph.D. studies. He provides me with valuable and in-time discussions on many interesting problems. His insightful knowledge on system modeling and synchronous languages help me understand the subject well.

My work has involved two project groups led by Axel. One is the Nostrum group. Nostrum is the name for the concept of our Network-on-Chip communication platform. This group involves Mikael Millberg, Rikard Thid, Erland Nilsson, Raimo Haukilahti and Johnny Öberg. The other is the ForSyDe (FORmal SYstem DEsign) group where Ingo Sander, Tarvo Raudvere and Ashish Kumar Singh contribute to. I thank all the group members for fruitful discussions and knowledge exchange. Particularly, I thank Rikard for his Network-on-Chip simulator called Semla. The flexibility of the NoC simulator has allowed me to develop other network layer models and perform experiments.

Thanks to Prof. Shashi Kumar, who co-supervised my study before he left KTH. Thanks to Dr. Mattias O'Nils for being my opponent.

Special thanks to my Chinese colleagues in the department.

Finally I am indebted to my wife Yanhong and daughter Lingyi. I have spent too much leisure time with computers, not with my family. Without the support from the family, nothing can be achieved. The last thanks go to my parents for their encouragement and constant bolstering for my studies.

Zhonghai Lu

Stockholm, April 2005

Contents

1 Background **1**
- 1.1 Network-on-Chip . . . 1
- 1.2 On-chip Networks . . . 3
- 1.3 Wormhole Switching . . . 7
- 1.4 Thesis Overview and Author's Contributions . . . 8

2 Feasibility Analysis **13**
- 2.1 Introduction . . . 13
- 2.2 Related Work . . . 15
- 2.3 The Real-time Communication Model . . . 16
- 2.4 The Contention Tree Model . . . 17
- 2.5 The Feasibility Test . . . 22
  - 2.5.1 The feasibility test algorithm . . . 22
  - 2.5.2 The feasibility analysis flow . . . 25

3 Flit Admission and Flit Ejection **27**
- 3.1 Introduction . . . 27
- 3.2 Related Work . . . 28
- 3.3 The Wormhole Switch Architecture . . . 29
- 3.4 Flit Admission . . . 32
  - 3.4.1 The decoupled admission . . . 32
  - 3.4.2 The coupled admission . . . 32
- 3.5 Flit Ejection . . . 34
  - 3.5.1 The ideal sink model . . . 34
  - 3.5.2 The p-sink model . . . 35

4 NoC Simulation **37**
- 4.1 The NoC Simulation Tool . . . 37

4.2 Traffic Configuration 38

5 Summary **41**
5.1 Thesis Summary 41
5.2 Future Work 42

References **43**

List of Figures

1.1 A mesh NoC with 9 nodes 2
1.2 Flits delivered in a pipeline 7
1.3 Virtual channels (lanes) 8

2.1 Feasibility analysis in a NoC design flow 14
2.2 Network contentions and contention tree 17
2.3 Message contention for links simultaneously 19
2.4 Avoided scenario of flit delivery 20
2.5 Message scheduling 21
2.6 A three-node contention tree 23
2.7 Message scheduling and contended slots 23
2.8 A feasibility analysis flow 25

3.1 Flit admission and flit ejection 27
3.2 A canonical wormhole lane switch (no ejection) 29
3.3 Flitization and assembly 30
3.4 Lane-to-lane associations 31
3.5 Organization of packet and flit-admission queues 32
3.6 The coupled admission sharing a (p+1)-by-p crossbar 33
3.7 The ideal sink model 34
3.8 The p-sink model 35

4.1 Network evaluation 38

List of Publications

Papers included in the thesis:

1. 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*, Shanghai, China, January 2005.

2. 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*, Tampere, Finland, November 2004.

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

4. 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.

5. 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.

6. 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*, Alberta, Canada, July 2005.

Publications not included in the thesis:

1. 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*, Kyoto, Japan, October 2002.

2. 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.

3. 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*, Munich, Germany, March 2003.

4. 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.

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

Abbreviations

ASIC	Application-Specific Integrated Circuit
BDG	Blocking Dependency Graph
CAD	Computer Aided Design
CMOS	Complementary Metal Oxide Semiconductor
CT	Contention Tree
DSM	Deep SubMicron
FIFO	First In First Out
FCFS	First Come First Serve
FPGA	Field-Programmable Gate Array
GALS	Globally Asynchronous Locally Synchronous
GUI	Graphical User Interface
IP	Intellectual Property
LCM	Least Common Multiple
LL	Lumped Link
NNSE	Nostrum Network-on-Chip Simulation Environment
NoC	Network on Chip
NT	Non-real Time
PC	Physical Channel
QoS	Quality of Service
QC	QoS Class
RNI	Resource Network Interface
RT	Real Time
RTL	Register Transfer Level
SEMLA	Simulation EnvironMent for Layered Architecture
SoC	System on Chip
TM	Traffic Model
VC	Virtual Channel
XML	Extensible Markup Language

Chapter 1

Background

This chapter serves as a background for the thesis, introducing Network-on-Chip (NoC), on-chip networks, and wormhole switching.

1.1 Network-on-Chip

Following Moore's law [26] which has sustained in semiconductor industry for over 40 years, a single chip is predicted to be able to integrate four billion 50-nm transistors operating below one volt and running at 10 GHz by the end of the decade [1]. Due to its huge capacity the billion-transistor chip can take on very complex functionalities with hundreds of interconnected microprocessor-sized computing resources. Such resources may be programmable like CPUs, dedicated like ASICs, configurable like FPGAs, or passive like memories etc. However, it is challenging to fully exploit the capacity offered by the technology. At the physical level, the Deep SubMicron (DSM) effects are getting severe. The interferences caused by crosstalk, power/ground plane noises etc. affect signal integrity to a larger extent. Inductance has to be taken into consideration more closely. A distributed RLC wire model is necessary to replace a lumped RLC wire model. All these factors make it more difficult to have the physical properties under control. At the logic level, the wire delay will soon dominate the gate delay [16]. Lowering the threshold of a CMOS to enhance performance has to trade off with power. At the Register Transfer Level (RTL), a purely synchronous design approach is challenged because a global synchronous clock simply will be infeasible. The synchronous design is only feasible in a small portion of a future chip. A future SoC is likely to be a multi-rate multi-voltage Globally Asynchronous Locally Synchronous (GALS) system.

At the architecture level, buses such as a single bus, bridged-buses, and hierarchical multiple-bridged buses, face even worse scalability problems with respect to performance and power. Limited bandwidth and bus length will make it difficult to interconnect many more resources. At the system level, heterogeneous resources imply various IP blocks, interfaces, protocols and operating systems etc., which are hard to integrate on a single chip. From the design methodology perspective, the gap between the methodology capacity and the chip capacity is not decreasing but increasing. The ad hoc RTL design considers too many implementation details. Its design capacity in terms of the number of transistors and productivity in terms of design time will not be sufficient to design the billion transistor chip. To close the gap, the abstraction level of design has to be increased, and reuse at all levels of abstraction is a must [19].

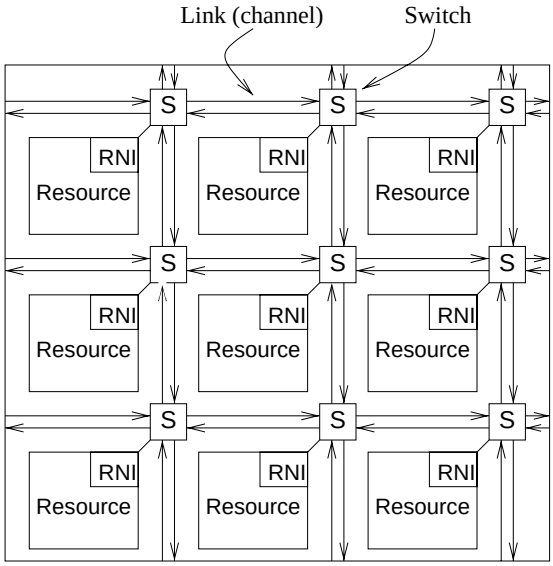


Figure 1.1. A mesh NoC with 9 nodes

In summary, chip design is increasingly becoming communication-bound rather than computation-bound [3, 36]. The interconnection problem will be addressed by communication-based design [35]. As a systematic approach, Network-on-Chip (NoC) [4, 13, 17, 23, 24, 25, 33, 34, 39], called also Network-on-Silicon, proposes networks as a scalable, reusable and global communication architecture to overcome the pains of future System-on-Chip. As an example, *Nostrum* [25, 29] is the Network-on-Chip communication platform developed at Royal Institute of Technology, Sweden. It has a mesh structure composed of switches with each switch

connected to a resource, as shown in Figure 1.1. The resources are placed on the slots formed by the switches. The maximal resource area is defined by the maximal synchronous area of a technology. The resources perform their own computational functionalities, and are equipped with Resource-Network-Interfaces (RNIs) to communicate with each other by routing packets instead of dedicated wires.

Although Network-on-Chip is considered as an inevitable solution for future SoCs [39], the design challenges with NoCs rise from the architecture, design methodology, programming model, and CAD tool support etc. All these difficulties stem from the stringent SoC evaluation criteria: performance, power and cost, as well as time-to-market (productivity) and time-in-market (programmability) pressures.

1.2 On-chip Networks

On-chip network is the core of NoC. A network is characterized by its *topology*, *switching strategy*, *flow control scheme* and *routing algorithm* [27].

- The *topology* refers to the physical structure of the network graph. For NoCs, regular structures are favored since they offer the potential to manage the ever-worsening electrical properties relating to signal and power integrity [13].

- The *switching strategy* determines how the data in 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 is released. 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 typically contains the error-checking code. Since packet-switching routes packets individually in the network, the network resources can be better utilized. Typical packet switching techniques include *store-and-forward*, *virtual cut-through*, and *wormhole switching* [1].

  Store-and-forward: a network node must receive the entire packet before forwarding it to the next downstream node. The non-contentional latency T

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

for transmitting L flits is expressed by Equation 1.1. 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 and either accepted or ejected.

$$T = (L/BW + R) * H \tag{1.1}$$

where BW is the link bandwidth; 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 to the destination node.

Virtual cut-through: a network node receives the portion of a packet and then forwards it downstream, if buffer space at the next switch is available. It does not wait for the reception of the entire packet. The buffers of the downstream node are allocated at the packet level and 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

$$T = L/BW + R * H \tag{1.2}$$

Wormhole switching: a packet is decomposed into flits. Operating like virtual cut-through, wormhole switching delivers flits in a pipelined fashion through the network. The difference is that the buffers of a switch are smaller, and do not hold the entire packet but a portion of the packet (a flit). If blocking occurs, the flits of the packet are blocked in place. Due to pipelined transmission, the non-contentional latency T of transmitting L flits is the same as that for virtual cut-through.

- The *routing algorithm* determines the routing paths the packets may follow through the network graph. It 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* [14]. Deterministic routing chooses always the same path given the source node and the destination node. It ignores path diversity. Oblivious routing, which include 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. An interesting extreme case of non-minimal adaptive routing is deflection routing [7].

Deflection routing: it is also called *hot-potato* routing. Its distinguishing feature is that it does not buffer packets. Instead, packets are always on the run. In fact, deflection routing is a routing algorithm plus a deflection policy. The routing algorithm determines the favored links for packets. The deflection policy governs how the contentions for links will be resolved. The link bandwidth is allocated packet-by-packet. Contentions are resolved by forwarding the packet with the highest priority to its favored link and misrouting packet(s) with a lower priority to unfavored links. As it does not buffer packets, the switch design can be simpler and thus cheaper. Since the routing paths are fully adaptive to the network state, it has higher link utilization and offers the potential to allow resilience for link or switch faults.

- The *network flow control* governs how a packet is forwarded in the network, concerning shared resource allocation and contention resolution. The shared resources are buffers and links (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 and contended. 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.

  Bufferless flow control is the simplest form of flow control. Since there is no buffering in switches, the resource to be allocated is channel bandwidth. It relies on an arbitration to resolve contentions between competing packets. After the arbitration, the winning packet advances over the channel. The other packets are either dropped or misrouted since there are no buffers. The deflection routing uses bufferless flow control.

  Buffered flow control stores blocked packets while they wait to acquire network resources. Store-and-forward, virtual cut-through and wormhole switching 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 channel bandwidth and buffers in units of packets. Wormhole switching allocates both channel 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. This phenomenon is called *back pressure*. Low-level flow control mechanisms are introduced to provide such back-pressure. Since the buffer availability information is passed between switches, these low-level flow control mechanisms can be considered as link-level flow control. There are three types of link-level flow control techniques in common use today: credit-based, on/off, and ack/nack [14]:

- With *credit-based* flow control, the upstream switch keeps track of the number of free flit buffers in each buffer queue downstream. If the upstream switch sends a flit, it decrements the corresponding count. If the downstream switch forwards a flit, it sends back a credit to the upstream switch, causing a buffer count to be incremented.
- With *on/off* flow control, there is a single control bit indicating whether the upstream switch is permitted to send (on) or not (off). A signal is sent upstream only when it is necessary to change the state. An off signal is sent when the control bit is on and the number of free flit buffers falls below the threshold *Foff*. An on signal is sent when the control bit is off and the number of free flit buffers rises above the threshold *Fon*.
- With *ack/nack* flow control, there is no state kept in the upstream switch to represent buffer availability. The upstream switch optimistically sends flits to the downstream switch. If the downstream switch has a free buffer, it accepts the flit and sends an acknowledge (*ack*) to the upstream switch. Otherwise, the downstream switch drops the flit and sends back a negative acknowledge (nack). The upstream switch must hold each flit until it receives an ack. If it receives a *nack*, it retransmits the flit.

Link-level flow control can manage short-term imbalance between packet injection and packet ejection. If congestion persists in the networks, buffers will be filled up and the traffic flow will be controlled all the way back to the packet injection source. This exerts the need for end-to-end flow control

to manage the imbalance between the sending process and receiving process [9]. Link-level flow control can also be used on host-switch links besides switch-switch links.

The *Octagon* on-chip network can select packet and circuit switching modes [33]. Using circuit switching in NoCs is proposed in [23, 24]. The *Proteo* NoC adopts virtual cut-through switching [4]. The Nostrum NoC employs deflection routing [25, 29]. Proposals to use wormhole switching in networks on chip can be found in [2, 10, 17, 34] [2]. Since NoC can be viewed as the future of SoC, an on-chip network will be highly constrained with not only performance but also cost and power. By exploring the traffic predictability of SoC applications, the network may be customized. In addition, the network should be better utilized but not over-designed.

1.3 Wormhole Switching

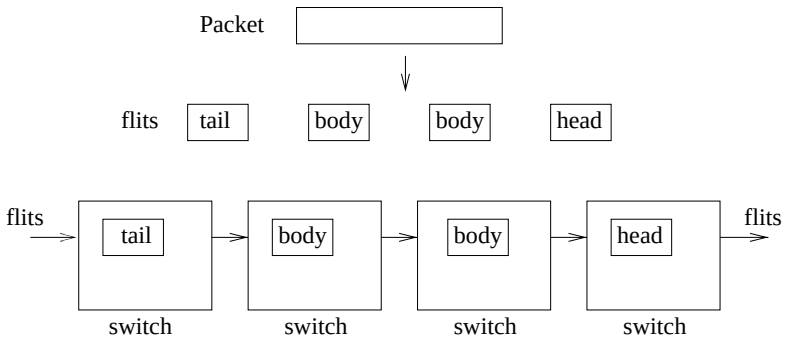


Figure 1.2. Flits delivered in a pipeline

Wormhole switching [12] allocates buffers and physical channels (PCs) to flits instead of packets. A packet is decomposed into one or more flits. 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 1.2, a packet is segmented into four flits, with one head

[2]The Philips $\mathit{Æthereal}$ NoC supports both best-effort and guaranteed services. It uses either wormhole or virtual cut-through switching for the best effort traffic. For the guaranteed throughput traffic, it employs TDM-based (time-division multiplexing) virtual circuit switching.

flit leading two body flits and one tail flit, and then the four flits are transmitted in pipeline via switches. For the same amount of storage, it achieves lower latency and greater throughput.

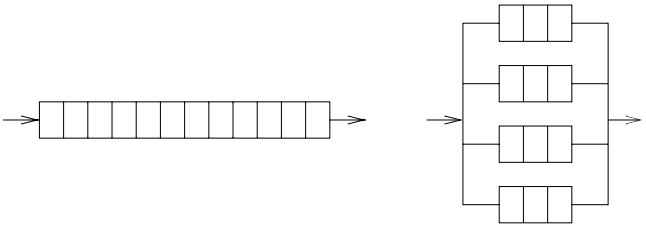


Figure 1.3. Virtual channels (lanes)

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 [11]. Several parallel lanes, each of which is a flit buffer queue, share a PC (Figure 1.3). 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 lanes is being advocated for on-chip networks [2, 10, 17, 34].

1.4 Thesis Overview and Author's Contributions

In this thesis, we mainly discuss two issues: *feasibility analysis* for NoC application design and *complexity reduction* for switch design. We have investigated the two problems in wormhole-switched networks on chip. The feasibility analysis aids designers with information about whether the application can fulfill the timing requirements on the NoC and how efficient network resources will be utilized. For the complexity reduction, we will explore the switch micro-architecture in order to reduce the number of logic gates and buffers. Specifically, we look into the *flit admission* and *flit ejection* problems. We will present in detail the feasibility analysis in Chapter 2, flit admission and flit ejection in Chapter 3. In Chapter 4, we briefly introduce NoC simulation. Finally we summarize the thesis in Chapter 5.

Since the thesis is based on a collection of papers, we concentrate on introducing the author's contributions in each chapter. The detailed materials, experiments

and results are referred to the papers. In the following, we summarize the enclosed papers highlighting the author's contributions.

- **Feasibility Analysis**

  Paper 1. 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*, Shanghai, China, January 2005.

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

  Author's contributions: The author formulated the contention tree model, developed algorithms and experiments to illustrate the use of the feasibility analysis, and wrote the manuscript.

- **Flit Admission and Flit Ejection**

  Paper 2. 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*, 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 idea, solution, experiments and wrote the manuscript.

  Paper 3. Zhonghai Lu and Axel Jantsch. Flit ejection in on-chip wormhole-switched networks with virtual channels. In *Proceedings of the IEEE NorChip Conference*, 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 performance penalty is tolerable with the p-sink

model, which achieves nearly equivalent performance with the ideal sink model if the network is not overloaded.

Author's contributions: The author initially noted the cost problem with the ideal flit ejection model, proposed solutions, conducted experiments and wrote the manuscript.

Paper 4. 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.

This paper is a follow-up paper on the coupled flit-admission scheme. It presents the power study of this scheme in comparison with a typical flit admission scheme, by using multiplexer-based crossbar and tristate-gate-based crossbar. An RTL implementation of the switch shows up to 8% reduction in the number of logic gates. The experimental results show that the coupled admission achieves up to 35% less switch power than other comparable solutions.

Author's contributions: The author contributed with the idea, power analysis methods and wrote the manuscript. Li Tong provided the power analysis results based on the RTL wormhole switch model designed by Bei Yin. Bei Yin also provided the switch area and performance data.

- **NoC Simulation and Traffic Configuration**

Paper 5. Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson, and Axel Jantsch. NNSE: Nostrum network-on-chip simulation environment. In *Swedish System-on-Chip Conference*, Stockholm, Sweden, April 2005.

This paper gives an overview of the Nostrum Network-on-chip Simulation Environment called NNSE by describing the NoC simulation kernel in SystemC [37, 38], the traffic and network configuration methods, and performance evaluation strategy. This tool has been demonstrated in the university booth program of the Design, Automation and Test in Europe Conference, Munich, Germany, March 2005.

Author's contributions: The author proposed the methods to configure network, traffic and the process to evaluate networks, programmed the network layer model for wormhole switching, coded the Graphical User Interface (GUI) of the tool in Python, and wrote the manuscript.

During the development of the tool, the author receives feedbacks from the other authors. Rikard Thid developed the layered simulation framework and programmed the simulation kernel in SystemC with contributions from Mikael Millberg and Erland Nilsson.

Paper 6. 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*, 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 regular traffic patterns and the application-oriented traffic configuration, integrated the methods in NNSE, conducted experiments and wrote the manuscript.

Chapter 2

Feasibility Analysis

This chapter covers the contention tree model and its algorithm for the feasibility test of real-time messages in wormhole-switched networks on chip [Paper 1].

2.1 Introduction

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 instance. With a mapping, application tasks running on these resources load the network with messages, and impose timing requirements. Timely delivery of these messages is essential for performance and predictability. However, routing messages in a network is inherently nondeterministic because messages experience various contention scenarios due to sharing resources such as buffers in switches and links between switches. These contentions cause indeterminate delay and jitter, leading to the possible violation of timing constraints for the messages. It is therefore important to conduct a feasibility test to determine if the messages can be delivered in time.

Figure 2.1 illustrates the feasibility analysis in a NoC design flow. A NoC design may be simply viewed as mapping a system specification onto a NoC architecture. 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 or cost. In our context, we concentrate the feasibility analysis on performance. We follow the feasibility definition in [5]:

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

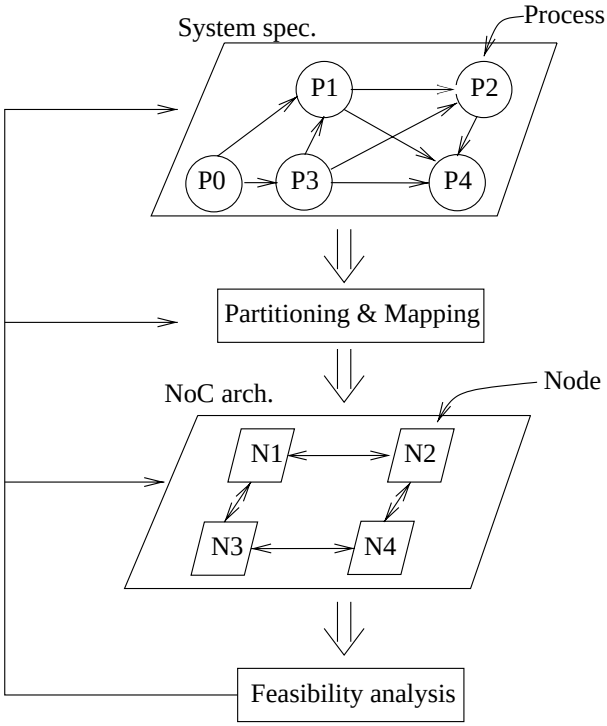


Figure 2.1. Feasibility analysis in a NoC design flow

in the set, and it does not prevent any message in the set from meeting its timing property.

With a feasibility test, we can obtain the pass ratio, i.e., the percentage of feasible messages, and the network utilization of the feasible messages. This information could be used in the iterative NoC design process to calibrate the partitioning of the specification, the mapping from processes to network nodes, and even the network architectural decisions such as topology and routing algorithm.

We distinguish messages with two different classes of timing requirements [34]. Messages with a deterministic performance bound, which must be delivered predictably even under worst case scenarios, are *critical or real-time* (RT) messages. Messages with a probabilistic bound, which ask for an average response time, are *non-critical or nonreal-time* (NT) messages. The two classes of messages coexist in on-chip networks. By estimating the *worst-case latency* for RT messages and average-case latency for NT messages, we can determine their fea-

sibility, respectively. Our focus in this thesis is on the performance bounds of real-time messages.

Feasibility analysis of real-time messages usually resorts to a *static* approach. This means that the characteristics of messages are known in advance. This is possible since SoC design is application specific, for example, executing a relatively stable body of code. For many SoC applications, a static analysis approach is preferred over a dynamic one for the sake of efficiency and predictability [40]. Exhaustive simulation of a NoC for each partitioning and mapping is more accurate, but difficult, if not impossible, to cover all the system states and very time-consuming. Meanwhile, NoC design is constrained by time and cost [6, 13], demanding an efficient means to help designers with information about the feasibility of messages and implications on performance/cost tradeoffs.

In the sequel, the related work is outlined in Section 2.2. Section 2.3 describes the communication model assumed by the contention tree (CT) model. In Section 2.4, the CT model is described in detail. Based on the CT model, a feasibility test algorithm and a feasibility analysis flow are described in Section 2.5.

2.2 Related Work

Kandlur *et al.* [18] developed a scheme to guarantee the maximum end-to-end message delivery time and a schedulability test to ensure that real-time messages meet their deadlines in multihop networks using store-and-forward switching. Few previous studies have been performed on the feasibility analysis of messages for wormhole-switched networks. Lee [21] presented a feasibility test algorithm for a simplex virtual circuit with real-time guarantees in wormhole networks. In the literature, there exist two worst-case performance models that do not assume a special communication model such as a virtual circuit. One is the lumped link (LL) model [5, 15], the other the blocking dependency graph (BDG) [20]. Both models assume deterministic routing and priority-based link arbitration in switches.

The lumped link model is a path-based model in which all the links along a message M_i's path are lumped into a single link. The message is scheduled on this link together with other competing messages. The feasibility test algorithms based on this model are efficient [5, 15]. However, due to lumping, all the competing messages must be scheduled in sequence. As a result, direct and indirect contentions are treated in the same way. Also, no concurrent use of the links on M_i's path can be taken into account.

In [20], Kim et al. analyzed network contentions to determine the feasibility of RT messages on wormhole-switched networks. They used a blocking depen-

dency graph to express the contentions a message may meet. In the graph, direct contention is distinguished from indirect contention. However, this graph does not either reflect the possible concurrent use of links.

2.3 The Real-time Communication Model

In wormhole switching, a message is divided into a number of flits for transmission[1]. Wormhole switching manages two types of resources: the lanes and the physical link bandwidth. Lanes are flit buffers organized into several independent FIFOs instead of a single FIFO. Such an organization can provide deadlock-free routing, as well as to improve the network throughput [11]. Lane allocation is made at the packet 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. This model is fair and produces average-case latency results, which is suitable to deliver NT messages. NT messages do not associate with a priority, and message delivery experiences average contention scenario. This communication model can not directly support real-time messages because there is no guarantee that messages are delivered before deadlines. In order to enable guarantee on message delivery, real-time messages must be served with other disciplines, for instance, priority-based arbitrations [22].

Similarly to the LL model, we assume a real-time (RT) message delivery model with a conventional switch architecture. Special RT communication services generally require special architectural support which potentially complicates 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 [15, 22], takes a small number of flits. With this RT model, the worst-case latency T^{rt} of delivering a message of L flits is given by :

$$T^{rt} = (L + L_{pri})/B^{rt} + HR + \tau = T + \tau \tag{2.1}$$

where B^{rt} is the minimum link bandwidth allocated to the RT message along its route; H is the number of hops from the source 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 per hop is assumed to be the same for a head flit and a body/tail flit. The first term counts for the transmission time of all the message

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

flits including that occupied by the priority; 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 contentions.

2.4 The Contention Tree Model

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 bandwidth of all the links 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. 2.2a 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.

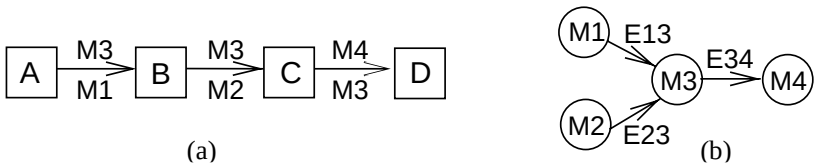


Figure 2.2. Network contentions and contention tree

To capture both direct and indirect contentions and to reflect concurrent scheduling on disjoint links, we have formulated a *contention tree* 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 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. Form 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. 2.2a is shown in Fig. 2.2b, 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.

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 [5, 15], we assume that there is 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 block message(s) with a higher priority. As discussed in [5, 15], this problem can be alleviated by packetization.

- By the communication model, messages are allocated time slots depending on their priorities and contentions. Whenever there is a contention for a

link, a message with a higher priority will be allocated first, even if it is blocked elsewhere. 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 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 2.3, 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 for slots [1, 7], i.e., for its entire base latency period.

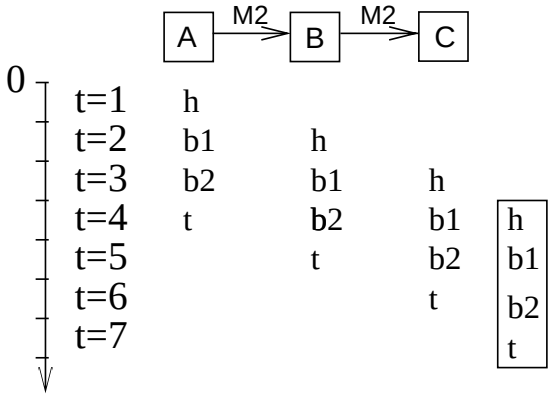


Figure 2.3. Message contention for links simultaneously

- 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 2.4, the scenario in time slot 3 is avoided, when the head flit **h** advances from node B to C but the first body flit **b1** is blocked in node

A. Clearly, if flits are individually routed via links, the contention period becomes unpredictable and larger than its base latency.

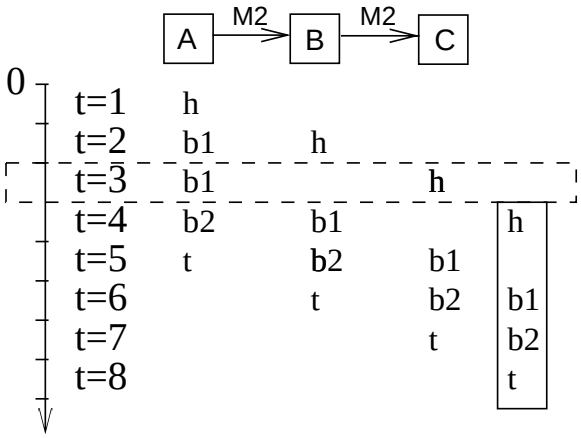
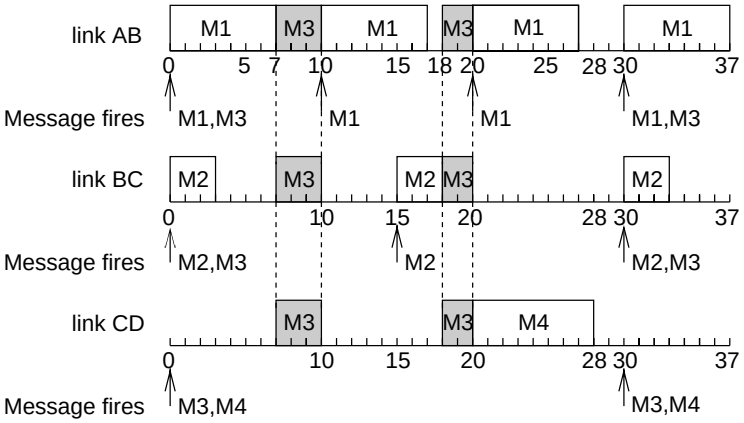


Figure 2.4. Avoided scenario of flit delivery

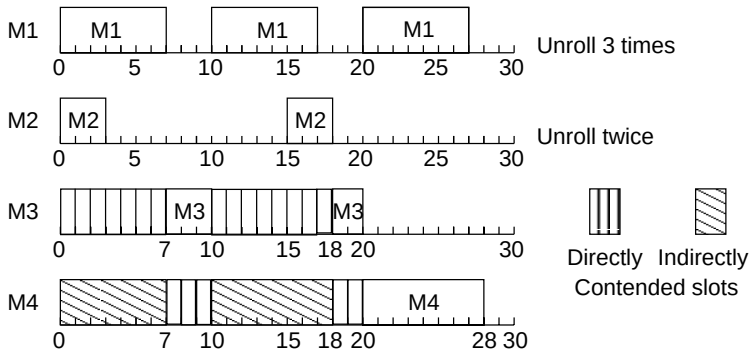
Table 2.1. Message parameters and latency bounds

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

Table 2.1 shows an example of message parameters for Fig. 2.2, 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. 2.5a. 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



(a) Link schedules of the messages



(b) Global schedules of the messages

Figure 2.5. Message scheduling

transmission by holding slots $[19, 20]$. After M_3 finishes its first delivery, M_4 is allocated 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 time for the four messages are 0, 0, 15, 20. The latency bounds for the four messages are also listed in Table 2.1. 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. 2.2b are disjoint, therefore their schedules do not interfere with each other;

(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. 2.5b. If direction contention is not distinguished from indirect contention, 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 captured, M_3 and M_4 would 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 might be pessimistic since many of them are not likely to occur at the same time. Also, a lower priority message can use the link bandwidth if a competing message with a higher priority is blocked elsewhere.

The validation of the CT model as well as the comparisons with the LL and BDG models are provided in [28].

2.5 The Feasibility Test

2.5.1 The feasibility test algorithm

Based on the contention tree created, 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 is feasible. If a node has feasible parent(s), we must first mark the contended slots as occupied and then schedule the node.

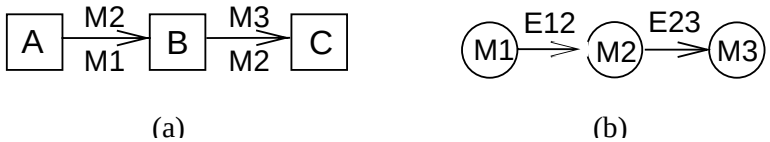


Figure 2.6. A three-node contention tree

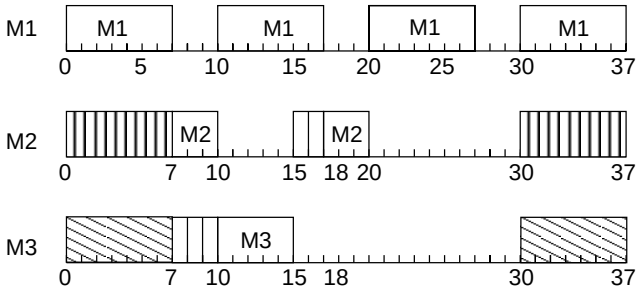


Figure 2.7. Message scheduling and contended slots

Note that a slot occupied by a higher priority message is not necessarily a *contended* slot. Consider the contention tree in Figure 2.6 where the three messages use the parameters in Table 2.1. The message schedules are depicted in Figure 2.7. 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. 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 direct 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 2.7, 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.

Algorithm 1 CT-based Feasibility Test for Real-Time Messages

```
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 = φ
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 ≤ T_i ≤ N_Di)
        or (M_i is deadline-constrained  and T_i ≤ 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);
```

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 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 D constraint or a jitter J constraint. 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.

We formulate this contention tree-based feasibility test in Algorithm 1. The input to the algorithm is a contention tree and message parameters, and the out-

put is the feasibility for each of the n messages on the tree, 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.

2.5.2 The feasibility analysis flow

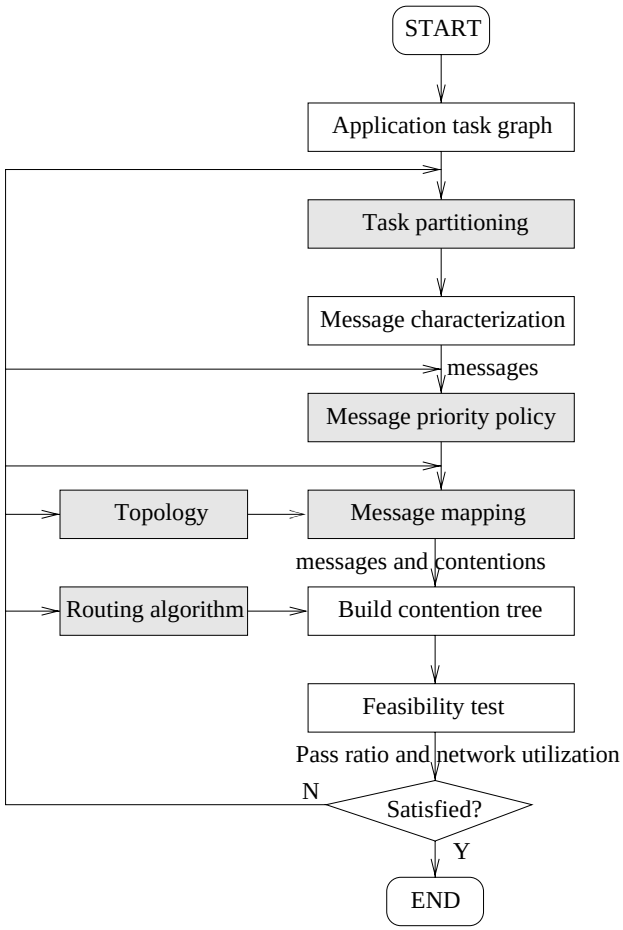


Figure 2.8. A feasibility analysis flow

By using the feasibility test, we can efficiently conduct feasibility analysis by exploring the application-level, partitioning/mapping-stage and architectural-level design space. Figure 2.8 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 path etc., we can build different contention trees by simply exploring these possibilities. After creating a contention tree, the feasibility test algorithm can perform the feasibility 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 make the design decisions. Clearly, this procedure is iterative until satisfaction.

Chapter 3

Flit Admission and Flit Ejection

This chapter presents the coupled admission model for flit admission and the p-sink model for flit ejection in wormhole-switched networks on chip [Paper 2, 3, 4].

3.1 Introduction

Because of the aforementioned advantages, namely, *better performance*, *smaller buffering requirement* and *greater throughput*, wormhole switching with lanes is proposed for NoCs. Nevertheless, using wormhole switching for on-chip networks has to minimize the switch design complexity. Since a network on chip is an interconnect scheme using shared wires instead of dedicated wires to pass signals, its cost should be reasonable [13]. Another reason can be seen from the power perspective. Although the communication bandwidth is achievable for future complex SoC integration, the energy consumption will probably be the bottleneck that has to trade off the performance [30]. Therefore, it is crucial to reduce the switch design complexity to decrease the number of gates and switching capacitance in order to shrink energy dissipation.

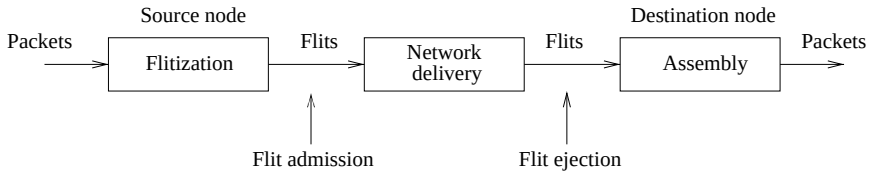


Figure 3.1. Flit admission and flit ejection

In this chapter, we examine the problem of flit admission and flit ejection in a wormhole-switched network. As illustrated in Figure 3.1, 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. Since flits are both the workload of switches and the source of network contentions for shared Virtual Channels (VCs) and Physical Channels (PCs) (links), the admission speed, the ejection speed and the balance between admission and ejection are important. To achieve good network utilization and system throughput, flits should be admitted as fast as possible. However, the more flits admitted in the network, the higher contentions may occur, leading to performance degradation. Similarly, a slower ejection of flits will make the ejection become performance bottleneck. An ideal ejection, which ejects flits immediately once they reach their destinations, may over-design the switch.

The remainder of this chapter is organized as follows. Section 3.2 outlines the related work. In Section 3.3, we explain the operation of a canonical wormhole lane switch. We discuss flit admission and flit ejection models in Section 3.4 and Section 3.5, respectively. Particularly, we detail the *coupled* admission and the p-sink model.

3.2 Related Work

The performance model of a wormhole switch that considers implementation complexity was first noted by Chien [8]. A more efficient canonical wormhole lane switch architecture and its performance model was presented in [32]. In general, the design complexity of a wormhole lane switch is the function of p and v, where p is the number of PCs of the switch and v is the number of lanes per PC. To gain further performance, flit-reservation flow control [31] was proposed which utilizes control flits to reserve bandwidth and buffers before transferring data flits.

To our knowledge, no prior work has been reported in the literature specifically discussing flit-admission and flit-ejection models. All of the above work assumes an ideal flit-ejection model while evaluating the network performance. Our motivation is to reduce the switch complexity to achieve cost-effective designs in silicon by exploring the design space of the switch micro-architecture. In line with this idea, Rijpkema et al. proposed to customize the lane buffers as dedicated hardware FIFOs instead of register-based or RAM-based FIFOs to reduce the area and thus achieve reasonable buffering cost [34]. To reduce the control complexity of the switches, deterministic routing is favored against adaptive routing. This may also be justified by exploiting the traffic predictability of specific applications [17],

which NoCs target. Moreover, regular low-dimension network topologies are considered for NoCs to further simplify the control and keep the electrical properties of wires under control [13, 25].

3.3 The Wormhole Switch Architecture

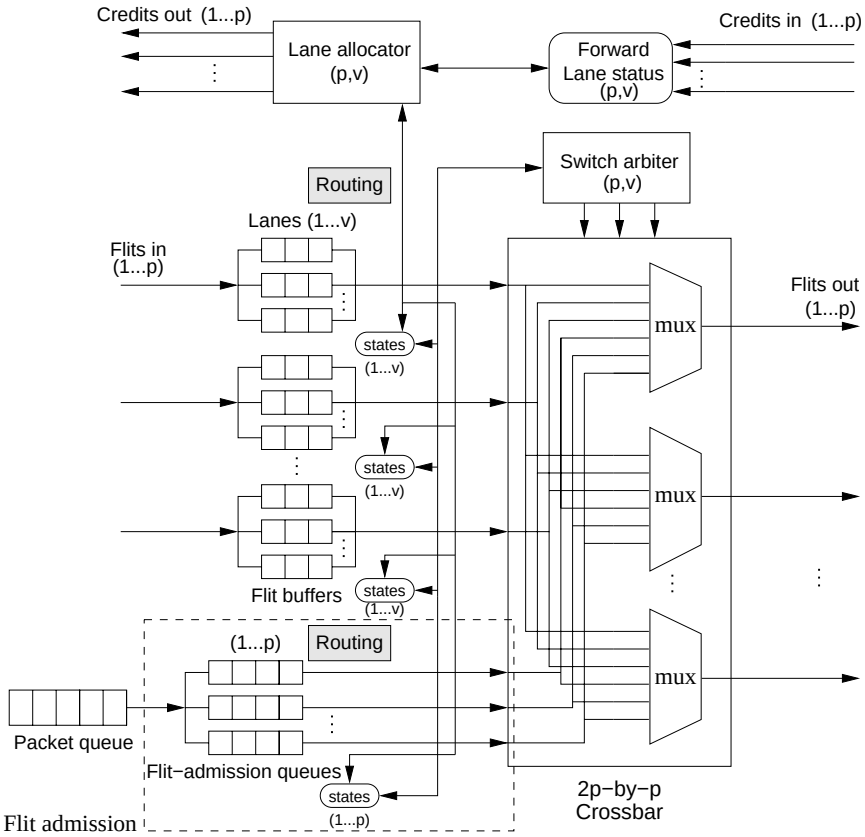


Figure 3.2. A canonical wormhole lane switch (no ejection)

Figure 3.2 illustrates a canonical wormhole switch architecture with virtual channels at inputs [11, 32, 34]. It has p physical channels (PCs) and v lanes per PC. It employs 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. 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 *lane-to-lane* association. If the lane-to-lane association succeeds, the packet enters into the scheduling state. If there is a buffer available in the associated downstream lane, the lane enters into the switch arbitration. 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 returns back to the scheduling state. The lane-to-lane association is released after the tail flit is switched out. Credits are passed between adjacent switches in order to keep track of the status of lanes.

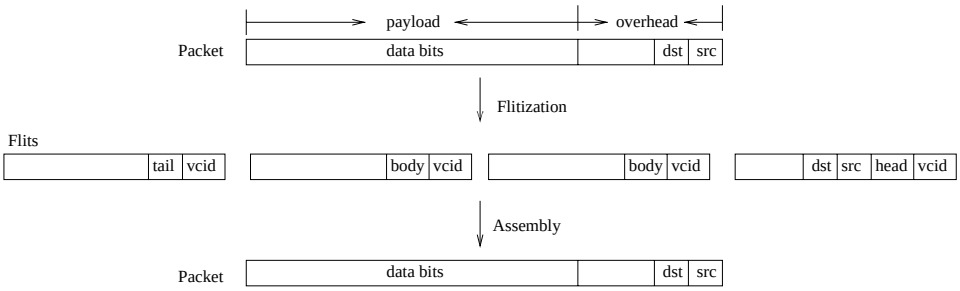


Figure 3.3. Flitization and assembly

With wormhole switching, a packet is decomposed into a head flit, zero or more body flit(s), and a tail flit. A single-packet flit is also possible. In Figure 3.3, a packet is encapsulated into four flits, where *vcid* is the identity number of a virtual channel or 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 torn down. This is to say, 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 on per flit basis. 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 flits other than head flits do not contain routing information. To guarantee this, a lane-to-lane association must be *one-to-one*, i.e., *unique* at a time. An upstream lane can not associate with two or more downstream lanes simultaneously (one-to-many association). A downstream lane can not be allocated to two or more upstream lanes at the same time (many-to-one association). Both one-to-many and many-to-one associations must be forbidden. Figure 3.4 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 3 in switch 3. The many-to-one association results in that lane 3 in switch 2 will receive flits from lane 1 and 2 from switch 1. Obviously the one-to-many association and many-to-one association 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. In a word, only one-to-one association is permissible.

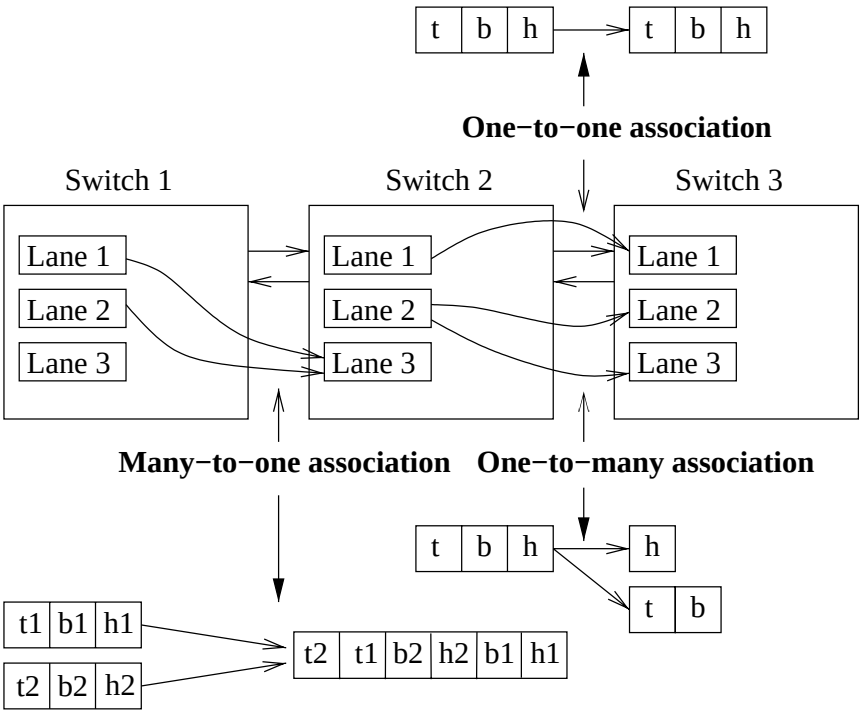


Figure 3.4. Lane-to-lane associations

3.4 Flit Admission

3.4.1 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 *admission queues*, before being admitted into the network. There are various ways of organizing the packet queue and the admission queues. In Figure 3.5.(a), flit-admission queues are organized as a FIFO. In Figure 3.5.(b) and 3.5.(c), they are arranged as p parallel FIFO queues (p is the number of PCs). Figure 3.5.(a) and 3.5.(b) allow at maximum one flit to be admitted to the network at a time while Figure 3.5.(c) allows up to p flits to be admitted simultaneously. We adopt the organization of flit-admission queues in Figure 3.5.(c) for our further discussions since it allows potentially higher performance.

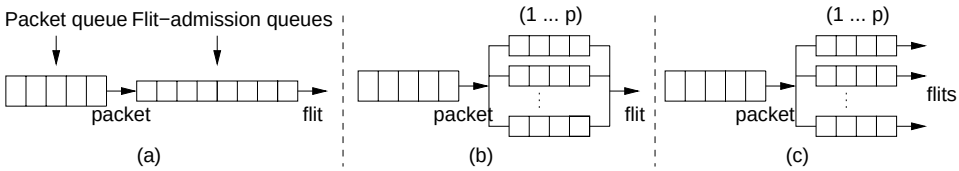


Figure 3.5. Organization of packet and flit-admission queues

Figure 3.2 also illustrates the organization of p flit-admission queues in the switch architecture. 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 the admission queue. A flit-admission queue transits states similarly to a lane to inject flits into the network via the crossbar. 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 admission scheme *decoupled admission*.

Observation: *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*.

3.4.2 The coupled admission

We propose a coupling scheme for the flit-admission that can sharply decrease the crossbar complexity, as sketched in Figure 3.6. Just like the decoupled admission,

it uses p admission queues, but one queue is bound to one and only one multiplexer 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 3.6. The number of control signals per multiplexer is reduced from $\lceil log(2p) \rceil$ to $\lceil log(p+1) \rceil$ for any $p > 1$ [1].

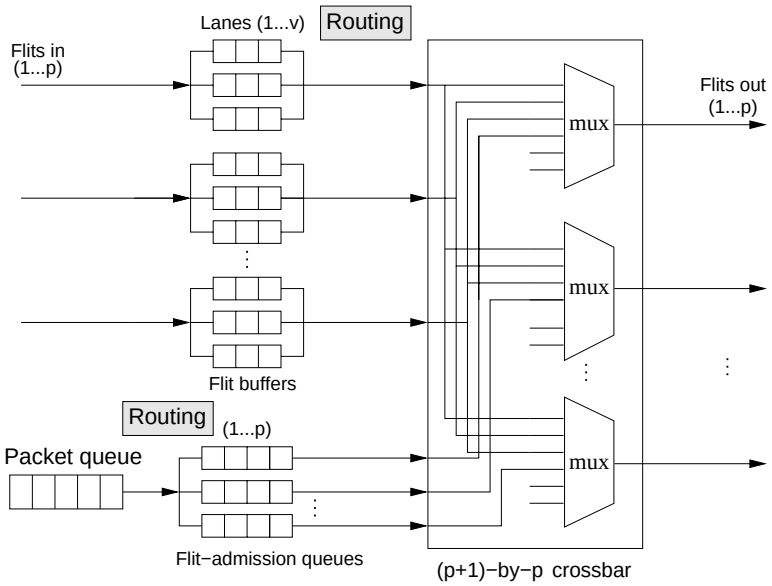


Figure 3.6. The coupled admission sharing a (p+1)-by-p crossbar

In order to support the coupling scheme, the routing must be performed before flitization instead. By a routing algorithm, the output physical channel which 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 higher. Specifically, if the head packet in the packet queue is blocked due to the bounded number and size of the admission queues, the packets behind the head packet are all unconditionally blocked during the head packet's blocking time. By 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 the present packet is aimed to is full.

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

3.5 Flit Ejection

3.5.1 The ideal sink model

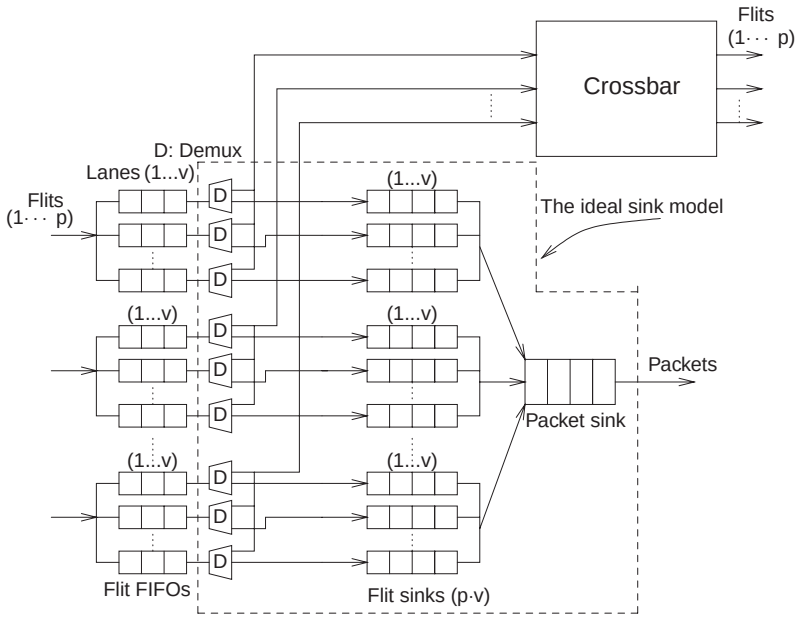


Figure 3.7. The ideal sink model

An ideal sink model is typically assumed for a wormhole switch. With an ideal ejection, 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 3.7. A *flit sink* is a FIFO receiving the ejected flits. Each lane is connected to a sink and the crossbar 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 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 reaching its destination 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 of the use of the demultiplexers, one PC always 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 the packet sink in a FIFO manner.

Observation: *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*.

3.5.2 The p-sink model

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 3.8.

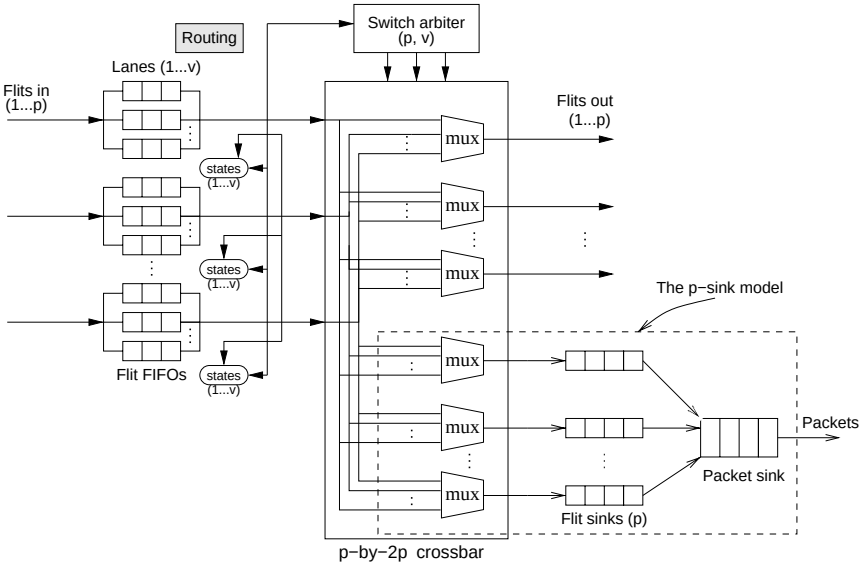


Figure 3.8. The p-sink model

To enable ejecting flits by 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, 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. 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 3.1. 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 3.8. The number of control ports of the crossbar is doubled proportionally. Table 3.1 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 sink number becomes independent of v, implying that the buffering cost for flit sinks is only $1/v$ as much as the ideal ejection model.

Chapter 4

NoC Simulation

This chapter introduces the NoC simulation tool called NNSE (Nostrum Network-on-Chip Simulation Environment) and the traffic configuration methods integrated in NNSE [Paper 5, 6].

4.1 The NoC Simulation Tool

Network-on-chip offers the potential to overcome the design challenges of future SoC designs such as the synchronization problem, the scalability problem of buses, the heterogeneity of systems etc. However, it greatly enhances the complexity of on-chip communication design. One major difficulty is to select a communication network that suits a specific application or application domain. From the network perspective, there exists a huge design space to explore with regard to topology, routing and flow control. From the application perspective, the network should serve as a customized platform and be relatively stable.

Simulation is a fast and cheap approach to help designers to make design decisions before resorting to emulation and implementation. NNSE is aimed to be a tool for full NoC simulation. Currently, it is capable of evaluating network architectures by parameterizing network and traffic configurations. It enables to configure a network, traffic models (TMs) and then evaluate the network with various traffic models. The evaluation is iterative by applying the traffic on the configured networks, as illustrated in Figure 4.1. The evaluation criteria can be performance, power and cost. The current version evaluates only network performance in terms of packet latency, link utilization and throughput.

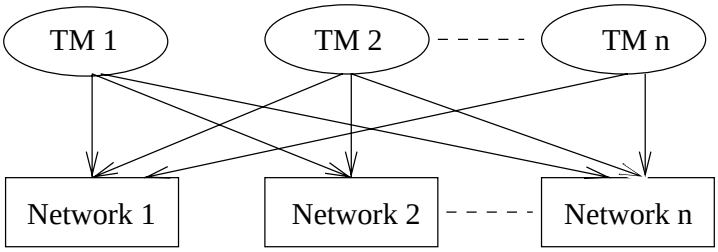


Figure 4.1. Network evaluation

NNSE logically comprises a NoC simulation kernel wrapped with a graphical user interface (GUI) written in Python. The simulation kernel is called Semla (Simulation EnviornMent for Layered Architecture) [37, 38] which implements the physical layer, the data link layer, the network layer, the transport layer and the application layer in SystemC following the ISO's OSI seven-layer model. Semla is cycle-accurate. The salient advantage of layering is that the complexity can be decomposed and dealt with independently. One can change the implementation of a layer without the need to change the layer above provided the interface between the two layers maintains. For instance, Semla originally developed the network layer for deflection routing. In order to perform experiments for flit admission and flit ejection in Chapter 3, the network layer for wormhole switching was developed and integrated into the Semla framework. With the GUI, all the network and traffic configurations can be stored and reused. To facilitate data exchange, they are stored as EXtensible Markup Language (XML) files.

4.2 Traffic Configuration

Network evaluation typically employs *application-driven* traffic and *synthetic* traffic [14]. 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. 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.

To evaluate on-chip networks, traffic configuration should also capture the application characteristics. In NNSE, two types of traffic can be configured. One is synthetic traffic, the other *application-oriented traffic*. For synthetic traffic, we

proposed a unified expression to configure both uniform and locality traffic. With this expression, changing the traffic locality is realized by changing its locality factor(s). The application-oriented traffic 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, and the temporal and size distribution of each channel may be synthetic or extracted from communication traces.

Chapter 5

Summary

This chapter summarizes this thesis followed by future research directions.

5.1 Thesis Summary

In this thesis, three important issues, *feasibility assessment for NoC application design*, *complexity reduction for switch design* and *NoC simulation*, have been discussed. The significance of the topics lies in that (1) there is a huge design space to explore, for instance, task partitioning, architectural choices and process-to-node allocation etc. Such an exploration should be efficient and achieve a good utilization of network resources; (2) although a future chip is abundant in transistors, the incremental cost and power consumption due to using networks as shared resources (switches and links) of interconnections should be minimal; (3) flexible and efficient NoC simulation is essential in a NoC design flow.

Throughout the text, the thesis focuses on the author's contributions to the three topics.

- The contention tree model accurately captures network contentions and reflects concurrency in link utilization, providing a simple model to estimate the worst-case performance for real-time messages on wormhole-switched networks. With this model, feasibility analysis with respect to design decisions in a NoC design process can be performed efficiently. The outcome of such a feasibility analysis can help designers to calibrate design decisions in order to achieve design goals.

- We have taken the initiative to look into the flit admission and flit ejection problem. The proposed coupled flit-admission technique and p-sink model shrink the switch complexity with only limited performance penalties in some cases. The coupled flit-admission largely simplifies the switch crossbar. With an RTL implementation, it shows a reduction of 41% in the crossbar logic gates and up to 8% in the total switch logic gates. In our experiments, the coupled admission scheme achieves up to 35% switch power reduction due to the decrement in logic gates and switching activities. The p-sink model uses only $1/v$ flit sinks as much as the ideal sink model. Although these models are equally applicable to macro-networks in parallel computing, we have conducted experiments in a wormhole-switched network using smaller amount of buffering resources and with a lower dimension topology. These models can serve as potential alternatives for cost-effective on-chip network design. Moreover, they demonstrate the importance and possibility to reduce the design complexity of canonical switches.

- The traffic configuration methods integrated in NNSE allows one to change the traffic locality by simply modifying the locality factor(s), and to create application-specific traffic.

5.2 Future Work

There are some interesting directions for further studies on feasibility assessment and complexity reduction. The feasibility analysis flow highlights the possibilities with the contention-tree-based feasibility analysis algorithm. Many more experiments from application-level decisions to architecture-level decisions could be performed and results can be studied in-depth.

We have reported preliminary results on the track of complexity reduction. As demonstrated, the reduction in design complexity leads to both cost reduction and power saving. We believe that the room for complexity decrement is large since networks on chip probably employ regular topologies and designers have good knowledge of the traffic characteristics. If combining flit admission together with packet admission, we could efficiently dimension the packet buffers to satisfy the performance constraints without buffer overrun for specific applications.

References

[1] *The International Technology Roadmap for Semiconductors*. Semiconductor Industry Association, 2001.

[2] 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.

[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] M. Alho and J. Nurmi. Implementation of interface router IP for Proteo network-on-chip. In *Proceedings of the 6th IEEE International Workshop on Design and Diagnostics of Electronics Circuits and Systems*, April 2003.

[5] 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.

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

[7] 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.

[8] A. A. Chien. A cost and speed model for k-ary n-cube wormhole routers. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):150–162, Feb. 1998.

[9] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1998.

[10] 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.

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

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

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

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

[15] 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.

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

[17] 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.

[18] D. D. Kandlur, K. G. Shin, and D. Ferrari. Real-time communication in multihop networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(10):1044–1056, Oct. 1994.

[19] 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.

[20] 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.

[21] S. Lee. Real-time wormhole channels. *Journal of Parallel and Distributed Computing*, 63(3):299–311, 2003.

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

[23] D. Liu, D. Wiklund, E. Svensson, O. Seger, and S. Sathe. SoC BUS: The solution of high communication bandwidth on chip and short ttm. In *Proceedings of the Europe Real-Time & Embedded Computing Conference*, September 2002.

[24] J. Liu, L.-R. Zheng, and H. Tenhunen. Interconnect intellectual property for network-on-chip. *Journal of System Architectures, Special issue on networks on chip*, 50(2):65–79, February 2004.

[25] 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*, 2004.

[26] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 1965.

[27] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26(2):62–76, February 1993.

[28] 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.

[29] 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.

[30] 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.

[31] L. S. Peh and W. J. Dally. Flit-reservation flow control. In *Proceedings of High Performance Computer Architecture*, pages 73–84, Jan. 2000.

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

[33] R. Rao, A. Nguyen, and F. Karim. On-chip communication architecture for OC-768 network processors. In *Proceedings of the 38th Design Automation Conference*, 2001.

[34] 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.

[35] 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.

[36] M. Taylor, J. Kim, J. Miller, D. Wentzla, F. Ghodrat, B. Greenwald, H. Ho, m Lee, P. Johnson, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Frank, S. Amarasinghe, and A. Agarwal. The Raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 22(2):25–35, 2002.

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

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

[39] P. Wielage and K. Goossens. Networks on silicon: Blessing or nightmare? In *Proceedings of the Euromicro Symposium On Digital System Design*, Sept. 2002. Keynote speech.

[40] T.-Y. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(11):1125–1136, Nov. 1998.

Paper 1

Feasibility analysis of messages for on-chip networks using wormhole routing

Proceedings of the Asia and South Pacific Design Automation Conference, Shanghai, China, January 2005.

Paper 2

Flit admission in on-chip wormhole-switched networks with virtual channels

Proceedings of the International Symposium on System-on-Chip, Tampere, Finland, November 2004.

Paper 3

Flit ejection in on-chip wormhole-switched networks with virtual channels

Proceedings of the IEEE NorChip Conference, Oslo, Norway, November 2004.

Paper 4

A power-efficient flit-admission scheme for wormhole-switched networks on chip

Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics, Florida, U.S.A., July 2005.

Paper 5

NNSE: Nostrum network-on-chip simulation environment

Proceedings of Swedish System-on-Chip Conference, Stockholm, Sweden, April 2005.

Paper 6

Traffic configuration for evaluating networks on chip

Proceedings of the 5th International Workshop on System-on-Chip for Real-time Applications, Alberta, Canada, July 2005.

List of Licentiate and Doctoral Theses from ESD Lab:

Licentiate Theses

- Johnny Öberg. **An Adaptable Environment for Improved High-level Synthesis**, ISRN KTH/ESD/AVH–99/14-SE, 1996.
- Bengt Oelmann. **Design and Performance Evaluation of Asynchronous Micropipeline Circuits for Digital Radio**, ISRN KTH/ESD/R–96/17–SE, 1996.
- Henrik Olson. **ASIC Implementable Synchronization and Detection Methods for Direct Sequence Spread Spectrum Wideband Radio Receivers**, ISRN KTH/ESD/R–97/11-SE, 1997.
- Mattias O'Nils. **Hardware/Software Partitioning of Embedded Computer Systems**, ISRN KTH/ESD/R–96/17–SE, 1996.
- Daniel Kerek. **Design of A Wideband Direct Sequence Spread Spectrum Radio Transceiver ASIC**, ISRN KTH/ESD/AVH–99/5-SE, 1999.
- Bengt Svantesson. **Dynamic Process Management in SDL to VHDL Synthesis**, ISRN KTH/ESD/AVH–2000/4-SE, 2000.
- Johnny Holmberg. **On Design and Analysis of LDI and LDD Lattice Filters**, ISRN KTH/ESD/AVH–2000/8-SE, 2000.
- Bingxin Li. **Design of Sigma Delta Modulators for Digital Wireless Communications**, ISRN KTH/ESD/AVH–2001/2-SE, 2001.
- Chuansu Chen. **Reusable Macro Based Synthesis for Digital ASIC Design**, ISRN KTH/IMIT/LECS/AVH–2002/1-SE, 2002.
- Li Li. **Noise Analysis of Mixers for RF Receivers**, ISRN KTH/IMIT/LECS/AVH–2002/2-SE, 2002.
- Tomas Bengtsson. **Boolean Decomposition in Combinational Logic Synthesis**, ISRN KTH/IMIT/LECS/AVH-03/08–SE, 2003.
- Meigen Shen. **Chip and Package Co-Design for Mixed-Signal Systems: SoC versus SoP**, ISRN KTH/IMIT/LECS/AVH-03/11-SE, 2004.
- Jimson Mathew. **Design and Evaluation of Fault Tolerant VLSI Architectures**, ISRN, KTH/IMIT/LECS/AVH-04/03–SE, 2004.
- Petra Färm. **Advanced Algorithms for Logic Synthesis**, ISRN KTH/IMIT/LECS/AVH-04/01–SE, 2004.
- René Krenz. **Graph Dominators in Logic Synthesis and Verification**, ISRN, KTH/IMIT/LECS/AVH-04/04–SE, 2004.
- Andrés Martinelli. **Advanced Algorithms for Boolean Decomposition**, ISRN KTH/IMIT/LECS/AVH-04/05–SE, 2004.

- Tarvo Raudvere. **Verification of Local Design Refinements in a System Design Methodology** , ISRN KTH/IMIT/LECS/AVH-04/09–SE, 2004.
- Maxim Teslenko. **Decomposition Algorithms for Efficient Logic Synthesis**, ISRN KTH/IMIT/LECS/AVH-04/08–SE, 2004.
- Xinzhong Duo. **Design and Implementation of RF System-on-Package Modules for Short-Range Wireless Communications**, ISRN KTH/IMIT/LECS/AVH-03/12-SE, 2004.
- Wim Michielsen. **Chip and Package Co-Design for Colpitts Oscillators**, ISRN KTH/IMIT/LECS/AVH-04/14–SE, 2004.
- Yi-Ran Sun **Nonuniform Bandpass Sampling in Radio Receivers**, ISRN KTH/IMIT/LECS/AVH-04/13–SE, 2004.

Doctoral Theses

- Tawfik Lazraq. **Design Techniques and Structures for ATM Switches**, ISBN 91-7170-703-4, 1995.
- Bengt Jonsson. **Switched-Current Circuits: from Building Blocks to Mixed Analog-Digital Systems**, ISRN KTH/ESD/AVH–99/1–SE, 1999.
- Johnny Öberg. **ProGram: A Grammar-Based Method for Specification and Hardware Synthesis of Communication Protocols**, ISRN KTH/ESD/AVH–99/3–SE, 1999.
- Mattias O'Nils. **Specification, Synthesis and Validation of Hardware/Software Interfaces**, ISRN KTH/ESD/AVH–99/4–SE, 1999.
- Peeter Ellervee. **High-Level Synthesis of Control and Memory Intensive Applications**, ISRN KTH/ESD/AVH–2000/1–SE, 2000.
- Henrik Olson. **Algorithm-to-Architecture Refinement for Digital Baseband Radio Receivers**, ISRN KTH/ESD/AVH–2000/2–SE, 2000.
- Bengt Oelmann. **Asynchronous and Mixed Synchronous/Asynchronous Design Techniques for Low Power**, ISRN KTH/ESD/AVH–2000/6–SE, 2000.
- Yonghong Gao. **Architecture and Implementation of Comb Filters and Digital Modulators for Oversupplying A/D and D/A Converters**, ISRN KTH/ESD/AVH–2001/1–SE, 2001.
- Lirong Zheng. **Design, Analysis and Integration of Mixed-Signal Systems for Signal and Power Integrity**, ISRN KTH/ESD/AVH-2001/3-SE, 2001.
- Per Bjuréus. **High-Level Modeling and Evaluation of Embedded Real-Time Systems**, ISRN KTH/IMIT/LECS/AVH–02/3–SE, 2002.

- Imed Ben Dhaou. **Low Power Design Techniques for Deep Submicron Technology with Application to Wireless Transceiver Design**, ISRN KTH/IMIT/LECS/AVH–02/4–SE, 2002.
- Ingo Sander. **System Modeling and Design Refinement in ForSyDe**, ISRN KTH/IMIT/LECS/AVH-03/03–SE, 2003.
- Andreas Göthenberg. **Modeling and Analysis of Wideband Sigma-Delta Noise Shapers**, ISRN KTH/IMIT/LECS/AVH-03/04–SE, 2003.
- Dinesh Pamunuwa. **Modelling and Analysis of Interconnects for Deep Submicron Systems-on-Chip**, ISRN KTH/IMIT/LECS/AVH-03/07–SE, 2003.
- Bingxin Li. **Design of Multi-bit Sigma-Delta Modulators for Digital Wireless Communications**, ISRN KTH/IMIT/LECS/AVH-03/10–SE, 2003.
- Li Li. **Modelling, Analysis and Design of RF Mixed-Signal Mixer for Wireless Communications**, ISRN KTH/IMIT/LECS/AVH-04/12–SE, 2004.
- Abhijit Kumar Deb. **System Design for DSP Applications with the MASIC Methodology**, ISRN KTH/IMIT/LECS/AVH-04/10–SE, 2004.