KTH Information and
Communication Technology

Architectural Techniques for Improving Performance in Networks on Chip

Mikael Millberg

KTH Royal Institute of Technology
School of Information and Communication Technology
Dept of Electronic Systems

Doctoral Thesis

Stockholm, Sweden 2011

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

KTH Information and
Communication Technology

Architectural Techniques for Improving Performance in Networks on Chip

Mikael Millberg

KTH Royal Institute of Technology
School of Information and Communication Technology
Dept of Electronic Systems

Doctoral Thesis

Stockholm, Sweden 2011

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

Dept of Electronic Systems (ES)
School of Information and Communication Technology (ICT)
Royal Institute of Technology (KTH)
Forum 120
SE-164 40 Kista, Sweden

Architectural Techniques for Improving Performance
in Networks on Chip

Doctoral Dissertation in Electronic System Design, KTH 2011

Printed by US-AB Stockholm, Sweden 2011

TRITA-ICT/ECS AVH 11:13
ISSN 1653-6363
ISRN KTH/ICT/ECS/AVH-11/13-SE
ISBN 978-91-7501-169-1

Learning is its own reward. Nothing I can say is better than that.

Michael S. Hart

Abstract

The main aim of this thesis is to propose enhancing techniques for the performance in Networks on Chips. In addition, a concrete proposal for a protocol stack within our NoC platform Nostrum is presented. Nostrum inherently supports both Best Effort as well as Guaranteed Throughput traffic delivery. It employs a deflective routing scheme for best effort traffic delivery that gives a small footprint of the switches in combination with robustness to disturbances in the network. For the traffic delivery with hard guarantees a TDMA based scheme is used.

During the transmission process in a NoC several stages are involved. In the papers included, I propose a set of strategies to enhance the performance in several of these stages. The strategies are summarised as follows

Temporally Disjoint Networks is that a physical network, potentially, can be seen to contain a set of separate networks that a packet can enter dependent on when it enters the physical network. This has the consequence that we could have different traffic types in the different networks.

Looped containers provide means to set up virtual circuits in networks using deflective routing. High priority container packets are inserted into the network to follow a predefined, closed, route between source and destination. At sender side the packets are loaded and sent to the destination where it is unloaded and sent back.

Proximity Congestion Awareness reduces the load of the network by diverting packets away from congested areas. It can increase the maximum traffic load by a factor of 20.

Dual Packet Exit increases the exit bandwidth of the network leading to a 50 percent reduction in worst-case latency and a 30 percent reduction in average latency as well as a lowered buffer usage.

Priority Based Forced Requeue prematurely lifts out low priority packets from the network to be requeued. Packets that have not yet entered the network compete with packets inside the network which gives tighter bounds on admission with a reduction of worst case latencies by 50 percent.

Furthermore, *Operational Efficiency* is proposed as a measure to quantify how effective a network is and is defined as the *throughput per buffers used in the system.* An increase of the injection of packets into the network to increase the system throughput will have a cost associated to it and can be optimised to save energy.

Reflection

Researchers have a strong resemblance with ants. In researching something completely different I came across a web page about ants [Greensmith]. When reading it, I was struck by the similarities between these ants and the research community. To make you see what I saw I've equipped their description of ants with more research-ish terms. Especially I've replaced the ants with researches, the queens with professors/senior researches and the workers/larvae with PhD students or PhD where appropriate...

The Researches

Researches are among the most successful people. Experts estimate that there could be 20,000 or more varieties of researches in the world. They have evolved to fill a variety of different scientific niches. They are found from the Arctic Circle to the tip of South America. They are interesting organisms that should be studied to better understand their unique behaviours and their roles in the earth's Societies.

The Research Community and Life cycle

Ants live in cooperative groups called *colonies* – the researchers call their cooperative groups *communities*. For both the ants and the researchers two or more generations overlap in the colony; adults – professors – take care of the young – the PhD students. The ants are divided into castes, specialized groups that take care of certain tasks; the researchers are divided into groups that attack different aspects of that particular community's research problem. Researches do have reproductive castes too, the professors and senior researches – and non-reproductive castes – the PhD students.

New Community Formation

Once a community of researches matures, it can establish new communities through various methods. The most common are budding and swarming. Budding is the breakaway of a group of researches from a mature community to form a new community. The group usually consists of one or more senior researches and some PhD students. Budding is common with species of ants that have multiple senior researches. Most researches establish new communities through swarming. Every now and then, mature research communities generate large numbers of "winged" forms. These are the young researches and PhD students going off to mate. An inseminated queen/senior researcher with funding then rids herself of her wings and attempts to start a new nest. The senior researcher rears her first brood alone funding them with his funding. If successful, the first brood opens up the nest and brings in funding for themselves, the senior researcher, and subsequent broods, and the community/department grows. However, the percentage of senior researches that successfully begin *new communities* is thought to be very small.

And the similarities continue... However, this analogy with the ants is both comforting as well as depressing. The comforting part is that it makes you feel as a part of something bigger where the individuals collectively contribute to the ant-hill of knowledge. This at the same time is the depressing part – if you did not publish that article somebody else will – if the presented idea, or your precious result, is useful and relevant enough for the community that is. I'm a strong believer in that the scientific progress is inevitable. That's why it now and then pops up disputes who were the inventor behind something. Like, for example, the phone – was it Bell or Grey? or could it even have been Meucci? The truth is most likely that all did it, and if they would not have done it somebody else would; because at the point in time the collective ant hill had reached that level and the time was come for the telephone to be invented. So in that sense neither Bell nor Grey is greater than any ant that happens to put a straw on the top of the hill – it is inevitable that somebody will do it!

For the Network on Chip community, it has been the same – the early papers originate from 1999 and can if examined be seen as isolated outbursts of inevitability due to that the anthill had reached a particular height.

This reasoning could be easily extended and end up in the conclusion that due to this inevitability, it is no point whatsoever to get up in the morning and go to the ant-hill to struggle for getting that particular straw towards the top. There are, however, two rewards on your way – first we have the 'Columbi egg' reward that is the immediate reward of getting your submitted paper published; even though many could have written it – *you* did it! And second – the reward of having a part in something bigger – when you go home after a hard day's work you can turn around and watch the silhouette of the ant-hill in sunset...

Acknowledgements

Some say that the process of becoming a PhD is a like a journey. In my case, this particular journey started in another millennium and was never like a two-week ‘all-inclusive’ or a four week trip with a minuscule planning and guide included. My journey has more been like back-packing in the land of Academia...

Since the trip was not bought as a package holiday no pre-booked guides were included. Instead I have had the opportunity to spend some time with ‘local guides’ that have helped me along the way. Hence, I would like to express my sincere gratitude to all my supervisors and co-supervisors that I have had the pleasure to spend time with and learn from. First I would like to thank my *most* local guide for his support and patience – Professor Axel Jantsch but also all my other guides: Ahmed Hemani, Shashi Kumar, Adam Postula and Hannu Tenhunen.

If you are about to make a trip, or is out on a trip, sooner, or later you will need some administrative support in filling out the right forms and solve logistic issues. In the land of Academia these supportive people are represented by secretaries, janitors and computer service personnel. For me the following people have offered me their support and patience over the years: Hans, Peter, Lena and Agneta – thank you!

Along any trip with longer duration, you make acquaintance with “locals”. These locals have offered me friendship and encouraged me as well as offered support during my stay. In addition they have made my daily life fruitful and exciting. Ingo, Christian, Johnny, Thomas, Johan, Lars and all the rest of you – thank you.

In any good adventure, you need to go on some serious excursions to meet the adventure. My travel mates over the years that also became my partners in crime deserves a big acknowledgement – Erland, Rikard, Zhonghai, and Per – thanks for all!!!

Except for going on excursions you need friends in your daily backpacker life as well. Some of you just made a short visit in Academia some took the opportunity to stay longer. Wherever you are today – thanks for the time we spent – Mladen, Mikael C, Mikael L, Per, Wim, Morgan, Stefan, Dan, Raimo, Per B., Kim, Steffen, Abhijit, Dinesh, Andreas, Peeter and all you others.

Furthermore, I would like to thank the opponent and his "cabin crew" of the thesis committee for their efforts, engagement, insightful suggestions and comments on my research. Their professional flight and landing enabled me to arrive home safely at the end. My sincere gratitude to all members of the committee: Dr. Marcello Coppola, Docent Christian Schulte, Professor Thomas Hollstein, and Professor Jari Nurmi.

Finally, and most importantly, I would like to thank my family. It is simply impossible to go away if you have nothing to come home to!

Stockholm, October 2011

Mikael Millberg

Included Publications

Part A. Papers Included in the thesis

Paper I

[Nilsson2003] Erland Nilsson, Mikael Millberg, Johnny Öberg, and Axel Jantsch. *Load Distribution with the Proximity Congestion Awareness in a Network on Chip*. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 2003), pp. 1126-1127, 2003.

Paper II

[Millberg2004a] Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel 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 and Exhibition (DATE 2004), pp. 890-895 Vol.2, 2004.

Paper III

[Millberg2004b] Mikael Millberg, Erland Nilsson, Rikard Thid, Shashi Kumar, and Axel Jantsch. *The Nostrum Backbone – A Communication Protocol Stack for Networks on Chip*. In Proceedings of the 17th International Conference on VLSI Design 2004 (VLSI 2004), pp. 693-696, 2004.

Paper IV

[Millberg2007a] Mikael Millberg and Axel Jantsch. *A Study of NoC Exit Strategies*. In Proceedings of the First International Symposium on Network on Chip (NOCS 2007), pp. 217-217, 2007.

Paper V

[Millberg2007b] Mikael Millberg and Axel Jantsch. *Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy.* In Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007), pp. 511-518, 2007.

Paper VI

[Millberg2009] Mikael Millberg and Axel Jantsch. *Priority Based Forced Requeue to Reduce Worst-case Latency for Bursty Traffic*. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 2009). pp. 1070-1075. 2009.

Paper VII

[Kumar2002] Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johnny Öberg, Kari Tiensyrja, and Ahmed Hemani. *A network on Chip Architecture and Design Methodology.* In Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2002), pp. 105-112, 2002.

Paper VIII

[Thid2003] Rikard Thid, Mikael Millberg, and Axel Jantsch. *Evaluating NoC Communication Backbones with Simulation*. In Proceedings of the IEEE NorChip Conference (NorChip 2003), pp. 27-30, 2003.

Part B. Other Relevant Publications on the Subject by the Author

Publication A

[Hemani2000] Ahmed Hemani, Axel Jantsch, Shashi Kumar, Adam Postula, Johnny Öberg, Mikael Millberg, and Dan Lindqvist. *Network on chip: An Architecture for Billion transistor era.* In Proceedings of the IEEE NorChip Conference (NorChip 2000), pp. 166-173, 2000.

Publication B

[Millberg2002] Mikael Millberg. *The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone*. Research report, LECS, ECS, Royal Institute of Technology, Sweden, TRITA-IMIT-LECSR02:01, ISSN1651-4661, ISRN KTH/IMIT/LECS/R-02/01 SE, 2002.

Publication C

[Pamunuwa2004] Dinesh Pamunuwa, Johnny Öberg, Li-Rong Zheng, Mikael Millberg, Axel Jantsch, and Hannu Tenhunen. *A study on the implementation of 2-D mesh-based networks-on-chip in the nanometre regime*. The VLSI Journal on Integration, vol. 38, No 1, pp. 3-17, 2004.

Publication D

[Lu2005] Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson, and Axel Jantsch. *NNSE: Nostrum network-on-chip simulation environment*. In Proceedings of the Swedish System-on-Chip Conference (SSoCC 2005), 2005.

Publication E

[Lu2009] Zhonghai Lu, Mikael Millberg, Axel Jantsch, Alistair Bruce, Pieter van der Wolf, and Tomas Henriksson. *Flow regulation for on-chip communication.* In Proceedings of the Design Automation and Test in Europe Conference and Exhibition (DATE2009), pp. 578-581, 2009.

Contents

4 Paper Results & Author's Contributions

5 Included Publications

A Nostrum

1 Introduction

This introduction is actually two introductions in one – it is an introduction to the thesis as well as an introduction to the history leading to Networks on Chip. In the first part of the introduction, there is a very brief overview of the thesis and how, and where, my work has contributed. In the second part I will give a coarse historical view on system integration and describe my view on why the Network of Chip concept has emerged and why this has happened now. In addition, I will try to give an explanation why the concept of Network on Chip has not entirely made its break through yet. Even though, it is starting to gain acceptance in industry.

Despite the unorthodoxly large page count of the thesis, it is actually still in the format of a collection. The reason for selecting this format is that the included papers are quite self contained and have already been scrutinized once. Moreover, the papers included have been published over a rather long time, and the early papers should hence be seen in the context and maturity of the NoC research of that time. Even though the thesis is in the format of a collection, I still feel that it deserves a larger Related Work section than usual. The related work section – the *Networks on Chip* chapter – naturally gets this size due to the, already mentioned, relatively large time span of the presented papers. The time span of the papers also called for a separate chapter of the Nostrum platform since none of the papers give a coherent picture of the platform that is valid for all the papers. Hence, *Chapter A – Nostrum* has been included.

Furthermore, during the process of developing the material in the included papers various simulators were developed in order to test and proof different ideas. For this reason, *Chapter B – The Semla Simulator* has been included.

These extra chapters of "information dross" could be seen as a complement appendix to the separate Network on Chip chapter where the background and the surroundings of the thesis are presented. These chapters do hopefully serve the purpose of giving a more coherent picture and better understanding of the presented papers.

In summary, the thesis could be said to have the following outline:

Chapter 1 – Introduction – Part II. As stated above this is my view on the system integration that led to the concept of Networks on Chips. In the second chapter – Networks on Chips – the concept of Networks on Chips are introduced and various aspects of the concept are presented and discussed. In addition to being an introduction chapter to Networks on Chips I also try to give the background to the included papers as well as put them into a context. This chapter is followed by a chapter summarising and delivering some condensed conclusions. Last of the "ordinary" chapters is a separate chapter with a summary of the included papers together with a few words about the author's contribution.

The Nostrum chapter A will present the platform used for all the experiments and implementation. In Chapter B, the simulator used for all the experiments throughout the thesis is described.

With this said I really hope that you enjoy the chapters to come.

1.1 INTRODUCTION – PART I – THESIS CONTRIBUTIONS

The main aim of this thesis is to propose enhancing techniques for performance of the communication aspects in Networks on Chips (NoCs); also a contribution to the layered Network on Chip is made.

The Network on Chip platform that has been developed during the evolution of our ideas we have chosen to call Nostrum. Nostrum is a network of switch-resource pairs organised in a 2D-mesh. Each switch has five input and five output connections. Four in/output connections connect the switch to its nearest neighbours in the direction of the compass and one in/output connection is used to connect the switch to the resource. The resource is an umbrella term for any device that wishes to communicate over the network, e.g. microprocessors, DSPs, memories etc.

The Nostrum platform has the following key features

- Supports best effort and guaranteed throughput traffic delivery
- Reliable – no packet drop
- Deflective routing is employed for best effort traffic delivery which gives a small footprint of the switches
- A TDMA based routing scheme is used for traffic delivery with hard guarantees

Paper III is describing the layered platform Nostrum and paper VII, and VIII give examples where the platform has been used. For the two latter papers the author's contribution is limited to the sections of the papers where the platform is concretely described.

Transmission of messages from a sender to a receiver over a NoC involves several stages

- Packetisation/Segmentation – the messages are split into packets
- Ingress/Downstream Queuing, Arbitration and Network Admission
- Network Transportation – routing through the network
- Network Exit and Egress/Upstream queuing
- Depacketisation/Desegmentation

In the papers [I, II, IV, V, and VI] included in this thesis, I propose a set of *strategies* to enhance the performance in several of these stages. Furthermore I propose a measure to quantify the how effective a network is in paper V – Operational Efficiency.

The concepts presented in the papers [I, III, IV, and V] are quite general and not tightly bound to the deflective routing of our Network on Chip concept platform Nostrum. In this context, it should also be stressed that the contribution of the thesis is to propose strategies to enhance the performance of the network. The contribution is NOT to promote the mesh topology nor the deflective routing strategy employed. In Figure 1.1 the contributions of the performance enhancing strategies are positioned in relation to where in the message transmission sequence they act.

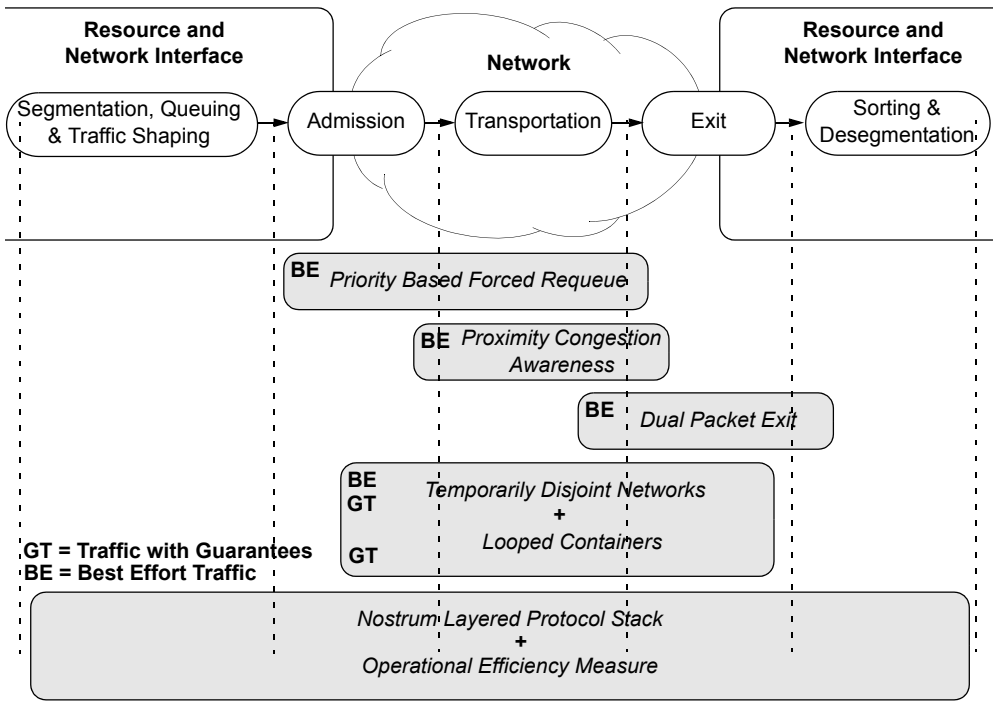


FIGURE 1.1. Thesis Contributions

Below is a sub sectioned list of the main contributions of the thesis and in what paper they appear.

Nostrum Layered Protocol Stack

Contribution appears in: Paper III, VII and VIII

Paper III presents a concrete proposal for a protocol stack within our Network on Chip concept platform Nostrum. The paper motivates and includes a protocol stack with well defined service interfaces within the Nostrum realm.

This paper presents our Network on Chip concept Nostrum. The concept defines a packet switched network with support for best effort traffic packet delivery as well as support for guaranteed bandwidth traffic, using virtual circuits. Furthermore, it includes a layered protocol stack and a corresponding nomenclature for describing the individual layers and their interfaces.

Within the concept, a concrete instance is described – the Nostrum Backbone. The backbone is a mesh based communication architecture defining Resources as hosts for communicating processes. The backbone defines the logical placement of these Resources and how they are connected. The Nostrum consists of Switch-Resource pairs connected in a two-dimensional grid. The relation between Resources, the Resource Network Interface, the Network Interface and the Switches is included. Our layered protocol stack uses a terminology heavily inspired by the OSI reference model. To prove the work of concept a distributed DSP application from industry was simulated. The results showed that the protocol covers the need of the particular example. The concept is further elaborated in *The Layered Approach to NoCs* on page 84 and in *Nostrum Layering* on page 170.

Temporally Disjoint Networks

Contribution appears in: Paper II

The idea behind the *Temporally Disjoint Networks* is that a physical network, potentially, can be seen to contain a set of separate networks that a packet can enter dependent on *when* it enters the physical network. A necessary condition for the existence of these TDNs is that a position in the network can only be reached on a multiple of N hops where N must be greater than 1. As a consequence the number of TDNs that exist, N is equal to the number of hops it takes to leave a switch and then get back to the very same switch in such a network.

In our case, these conditions will be fulfilled in a Manhattan network with logically identical switches that performs no reordering of packets.

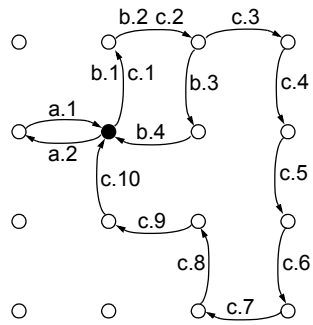


FIGURE 1.2. Temporally Disjoint Networks

If the network is a torus the number and rows and columns must be even to render more than one TDN.

To illustrate the idea three different routes have been layered out in the Manhattan network in Figure 1.2. Their respective path lengths are 2, 4, and 10. Hence, the number of TDNs that exist in this network will be 2. This means that two packets that are sent out consecutively on the network (i.e. in cycle n and cycle n+1) will never be able to collide! This has the consequence that we could have different types of traffic in the different TDNs to enable different guarantees on latency and throughput in the respective TDN. The concept is further elaborated in *Network Transport* on page 178.

Looped Containers

Contribution appears in: Paper II

Looped containers provide a means to set up virtual circuits in a network that employs a deflective routing policy. To give guarantees in latency and throughput for individual packets three separate, but linked, problems must be solved. (1) The packet must be able to enter the network at a given time; the packet must be transported over the network within a certain time, and (3) the packet must be guaranteed an exit from the network to be delivered properly. Failing in solving either of these problems means that limited or no guarantees on latency can be given!

To solve these problems a proxy packet was inserted into the network – bound to follow a predefined, closed, route between the source and the destination. This particular proxy packet was given the highest possible priority to guarantee precedence over any other packet.

Once the packet visited the sender it was loaded and sent to the destination where it is unloaded – hence the name *looped container*.

If some effort is put into the mapping process of senders, receivers, and container routes a set of different virtual circuits can be set up. In the case where the cycling time of different loops shares a smallest common divider they can overlap without risk of collision. In the case where this may not be possible the virtual circuits can be placed in different TDNs to guarantee an interference free behaviour. The concept is further elaborated in *Network Admission, Routing and Exit* on page 175.

Proximity Congestion Awareness

Contribution appears in: Paper I

Proximity Congestion Awareness (PCA) is used to reduce the total load of the network. The bufferless deflective routing policy of our platform Nostrum can create hot-spots in the centre of the network. The reason for this behaviour is that a super-set of all possible routing paths to and from all senders and receivers in the network will have an overrepresentation of potential paths in the centre of the network. The remedy to this is to try to divert packets away from congested areas. The solution we propose is based on the idea that all switches monitor their current load and distribute this information – the stress value – to its neighbouring switches. The stress value is the sum the number of incoming packets averaged over the last four clock cycles.

A comparison of the resulting FIFO load of the different setting is presented in textual form in the paper and shows that the reduced maximum average FIFO load is greatly reduced and that the load has a wider distribution over a larger area. Using the PCA concept results in a substantial improvement with a 20-time load reduction. The concept is further elaborated in *Network Transport* on page 178.

Dual Packet Exit

Contribution appears in: Paper IV and V

Dual Packet Exit is a proposed solution to the problem that there exist an accumulation of packets at the exits of the Network. The problem was identified when the exit process of packets in Nostrum was analysed. Our solution to this is to increase the exit bandwidth. The result is a reduction in the worst case latencies, the average latencies as well as in a lowered use of buffers. From simulation the most beneficial increase in terms of enhanced performance versus cost was to double the exit bandwidth – hence the name *Dual Packet Exit (DPE)*.

Priority Based Forced Requeue

Contribution appears in: Paper VI

The focus is here set on the admission to the network. The main problem is that packets may have to wait indefinitely long to enter the network dependent on the current load of the network. The network may have a fair routing policy that gives priority to "old" packets to keep the worst case latency down. Unfortunately, this does not help the packet that has not yet entered the network. Our solution to this is to make the system more globally fair by introducing *Priority Based Forced Requeue*.

Forced Requeue is to prematurely lift out low priority packets from the network and requeue them outside using priority queues. The first benefit of this approach, applicable to any NoC offering best effort services, is that packets that have not yet entered the network now compete with packets inside the network and hence tighter bounds on admission times can be given. The second benefit that is more specific to deflective routing in Nostrum is that packet reshuffling dramatically reduces the latency inside the network for bursty traffic due to a lowered risk of collisions at the exits.

Operational Efficiency

Contribution appears in: Paper V

Operational Efficiency is a measure to quantify how effective the network is. The cost that we define is the use of buffers. The use of buffers is both in terms of required buffer capacity as well as the average number of buffers actively used. The *required buffer capacity* is the buffers that have to exist in the system to cover for any worst case scenario. The *average buffers actively used* is the average number of buffers that is currently holding a packet in the system. From the average buffers actively used in the system we derive and define the term *Operational Efficiency* that is a measure defined as the *throughput per buffers used in the system.* The greatest benefit of this measure is that a graph plotting the Operational Efficiency vs. the packet injection ratio now has a clear *sweet spot!* This has the concrete impact that an increase of the injection of packets into the network to increase the system throughput will have a cost associated to it and hence can be optimised to save energy. Using this measure we show that the use of our Dual Packet Exit strategy can significantly increase the system bandwidth without increasing the energy used. The concept is further elaborated in *Service Characteristics* on page 72.

1.2 Introduction – Part II – The Road Towards NoC

Today's ever increasing integration of transistors into a single chip has created a demand for intelligent communication on-chip. To understand why this has happened, a historical perspective on integration is needed. The process of integration is a key component in the evolution of both microprocessors and personal computers as well as for the more custom electronic system that today has become a *System on Chip*.

The key is to understand that for both the computer as well as for the System on Chip realm integration is not only restricted to an increase in the sheer number of transistors but also encompasses the system as a whole. The process of integration made what used to be off-chip modules now become to be on-chip modules with much of their functionality intact. In short – what used to be a *system on board* is now a *system on-chip*!

Historical System Integration

This process of integration has been the companion of the hardware designer from the very beginning. During the 1950's discrete transistors connected with wires gave logic gate functionality, and this was "the system". As soon as technology advance made it possible, transistors were integrated in monolithic devices, such as flip-flops, half adders, etc. These monolithic devices were now the modules and could be used to compose a system.

The transistors (and the wires) were integrated and sold as discrete components – logic gates. Now the integrated logic gates could be used as components in building a system giving more advanced functionality such as adders, controllers and complex custom logic functions. The most common of these macro-blocks where considered as systems and turned into components.

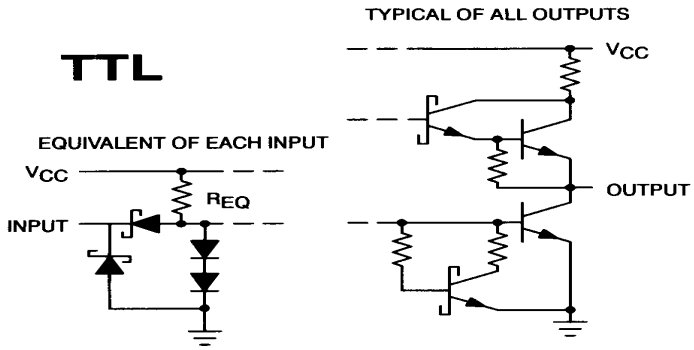


FIGURE 1.3. Electrically Standardised In- and Outputs – Transistor-Transistor Logic (TTL)

The monolithic devices were developed and refined during the 60's with milestones such as the Transistor-Transistor Logic (TTL) [Millman1987] in 1963, which was a standardized *Electric* interface to ease integration and system composability as depicted in Figure 1.3

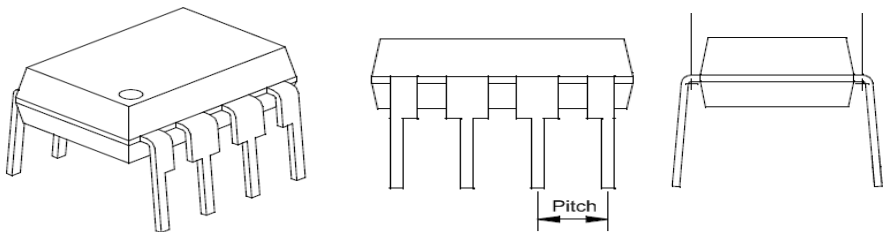


FIGURE 1.4. Physically Standardised – Dual In-line Package

By the 1970s, almost all digital components were TTL compatible; this included all logic parts, memories and micro processors. In 1965 the standardized Dual In-line Package (DIP) [Millman1987] format significantly, which was a *physical* standardised layout, eased printed circuit board layout and reduced the system assembly cost (Figure 1.4).

TABLE 1.1. Scaling Terminology

Name	Year	# Gates
Single Transistor	1959	-
Unit Logic	1960	1
Multi-function	1962	2-4
Complex Function	1964	5-20
Medium Scale Integration	1967	20-200
Large Scale Integration (LSI)	1972§	200-2000
Very Large Scale Integration (VLSI)	1978	2000-20000
Ultra Large Scale Integration (ULSI)	1989	20000-

During the 70's the micro processor were introduced. One of the first micro processors to hit the market was Intel's 4 bit 4004, which contained ~2000 transistors. Now larger systems could be built targeted for the consumer market. Previously, the predominant customer had been the military and big companies using large main frame computers. With the advent of the 4004 the consumer market opened and the Japanese manufacturer, Busicom's desktop printing calculator, became the world's first commercial product to use a microprocessor. The 4004 delivered the same computing power as the pioneer electronic computer, the ENIAC, built in 1946, which filled an entire room and used 18,000 vacuum tubes [Intel4004]. The 4004 had three companion circuits: the 4001 – a Read-Only Memory (ROM) chip for storing software; the 4002 – a Random Access Memory (RAM) chip for data storage and the 4003 – an input-output device.

In April 1973 came the 8 bit 8008 that had 3300 transistors. The instruction set of the 8008 eventually became part of the basis for the X86 architecture behind Intel chips today. The 8008 was to be followed by the 8080 in 1974. The 8080 was 4,500 transistors and became the basis of the Altair 8800 that some people regard as the ancestor to the modern personal computer. Of course Intel was not the only micro processor manufacturer of that time but the process of integration has been similar in any other computer brand.

The 8080 was followed by the 8086 and later by 80186 that had the major function to reduce the number of chips required by including features such as a DMA controller, interrupt controller, timers, and chip select logic – system integration!

The steady increase in the number of transistors that could be placed on one chip has two components: improved process technology (1) gives smaller geometry in combination with larger chips (2). Scaling of transistors sizes down make them operate faster. The consequence of this combined effect was early recognized by the Intel co-founder Gordon Moore who predicted that the semiconductor performance would have an exponential growth. In Table 1.1 this exponential increase is depicted with the corresponding terminology for that particular size

This evolution has progressed over the years and given faster and faster computers and systems. Throughout this development three key items can be identified

- Higher *integration* in terms of transistors
- Systems *off*-chip become Systems *on*-chip
- Standardization in interfaces, that enables *composability* and *reuse*

There is, however, a serious problem that arises in the process of integration. The logic itself becomes too big to handle – both from a designer's perspective as well as from the clock distribution perspective. In order to solve the both these problems a GALS (Globally Asynchronous Locally Synchronous) [Hemani1999, Meincke1999, Muttersbach1999] design style/methodology can be adopted. The basic idea of GALS is to divide the design into blocks, or regions, of approximately up to 100k gates. Each block will have a single synchronous clock but inter communication between blocks are handled asynchronously.

This design style obviously requires means to handle the asynchronous communication. To facilitate this, a bus could be employed. The bus is a shared medium and hence enables multipoint communication. This solution rimes pretty well with the three key items identified since the bus is a common off-chip communication solution. In addition, it encourages and requires standardization of interfaces and protocols. Even so – as we will see – this will take us far but not all the way!

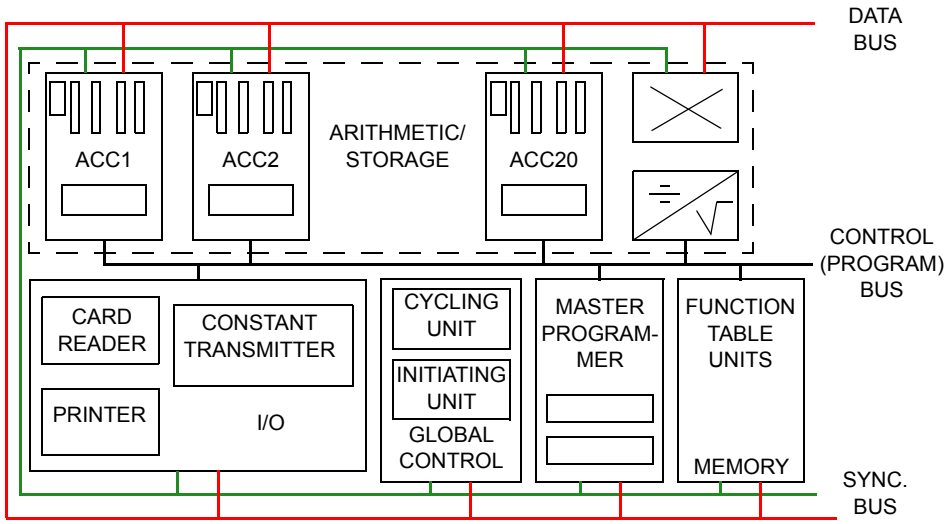


FIGURE 1.5. Eniac Schematics

The Bus

The bus has been present from the very beginning and the need for a shared medium was recognized even in the earliest machines like ENIAC (Figure 1.5) [Goldstine1972]. With the invention of the bus, the need for a standard way of communication was apparent. The bus *has* developed over the years *but still the bus is essentially a set of wires connected to all devices*; and since it is a shared resource – one connection between devices reserves the whole interconnect. In Figure 1.6 the simplest variant of the bus is depicted.

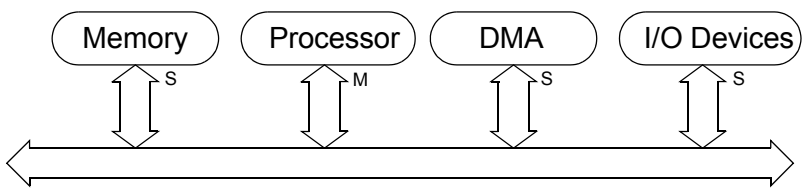


FIGURE 1.6. Simple Bus

Simple, because the processor acts as a *Master (*marked with an M in the figure) exclusively and hence no arbitration is needed. The Master is the only device on the bus that is allowed to initiate any traffic to and from the *Slaves*. In the case of a system where multiple Masters are presents some kind of arbitration is needed. If fairness, bandwidth guarantees are desired the arbitration can become quite complex.

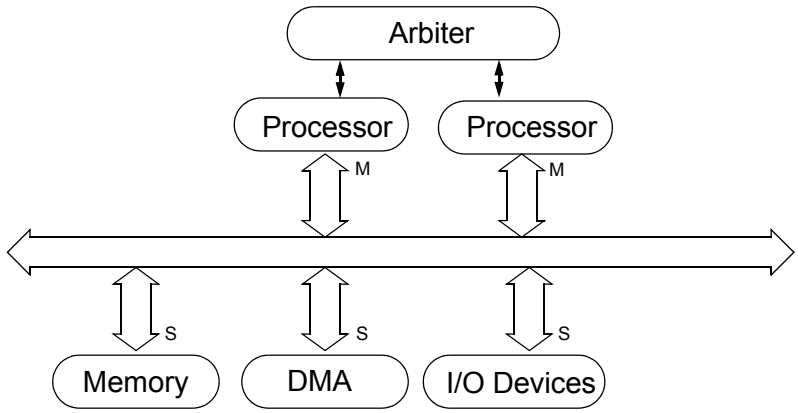


FIGURE 1.7. Arbitrated Bus

In the case of several Masters that wish to utilise the bus, arbitration is required as depicted in Figure 1.7.

The strength of the bus is that it is a shared medium, which makes it capable of connecting any device attached to it to any other device. This is at the same time the buses' weakest point since it has a very limited total system bandwidth. Moreover, the scalability from a physical perspective is limited since attaching more devices requires high fan-out and deep multiplexer logic and hence causes delays [Arifin2004]. In addition, the wires tend to be long, adding even more delay.

The Segmented Bus

One improvement that can potentially boost a bus based system is to divide it into independent segments and hence increase the system bandwidth – Figure 1.8. The precondition to achieve this is of course that the problem is possible to parallelise; otherwise the benefit of any bus-segmentation will only be a reduction of the depth of the multiplexer logic. In order to be an efficient solution, the inter-segment transactions should be quite rare and most of the operation should be done within the same segment. If no buffering is used in the bridges they could be implemented as simple tri-state buffers.

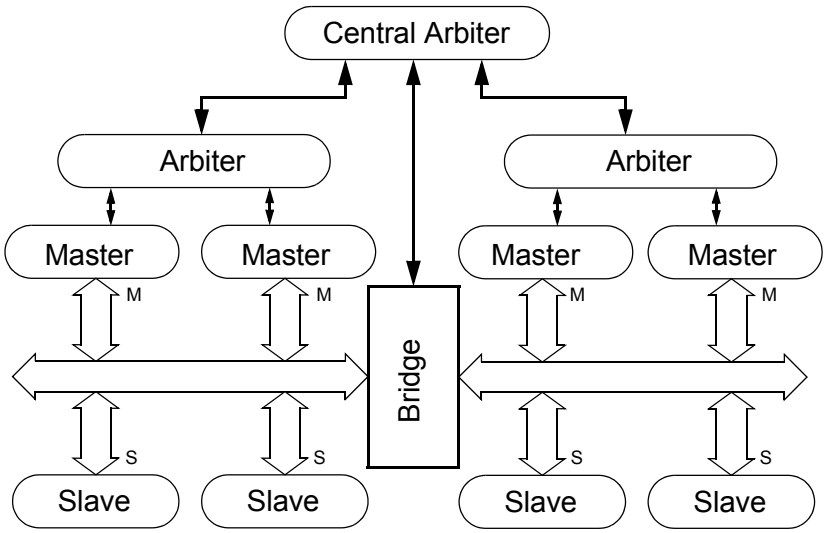


FIGURE 1.8. Segmented Bus

Except for the obvious benefit – that the segmented bus gives fast access as long as the target is in the same segment – the use of bus segmentation gives rise to, at least, three serious objections.

- The Central Arbiter needs a thorough understanding of the system to be able to perform any "fair" arbitration
- Any arbitration will be too complex given the available time
- An inter-segment transaction will not only be slow but also stall or block any other transaction

If these initial objections are ignored, for the moment – What happens with the hierarchical segmented bus as a communication infrastructure when the system (on-chip) becomes bigger? For a system *off*-chip, such as the PC, the PCI (Peripheral Component Interconnect) has been used as a scalable solution [Dandamudi2002].

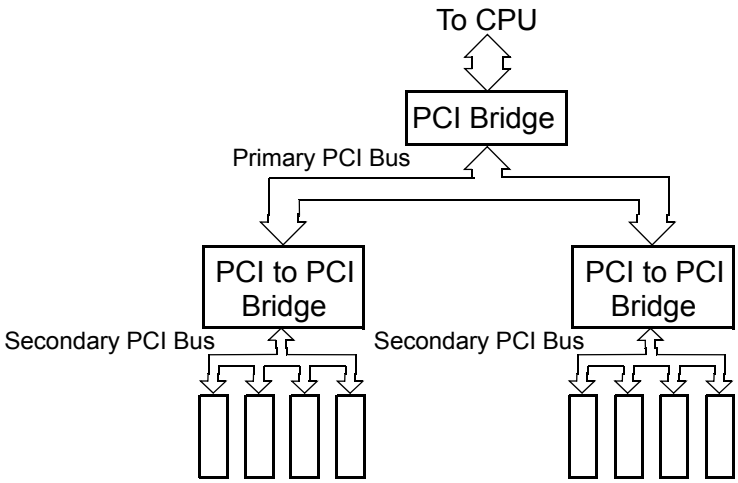


FIGURE 1.9. Hierarchical PCI Bus

The Hierarchical PCI

The hierarchical PCI is typically built using PCI-to-PCI bridges. The bridge connects two independent PCI buses: a primary and a secondary. The bus closer to the CPU is called the primary PCI bus; the other is called the secondary. Each secondary PCI bus supports up to four PCI devices, as shown in Figure 1.9.

The bridge allows concurrent operation of the two PCI buses as well as traffic filtering, which minimizes the traffic crossing over to the other side. The traffic separation along with the concurrent operation improves overall system performance for bandwidth-hungry applications such as multimedia. Each PCI-to-PCI bridge has its own bus arbiter.

Would it be possible to migrate this solution *as is* into the chip to solve the problem of a too complex design? Unfortunately, the answer is no – this since the delay can be 10-20 clock across the chip. So if this is to be implemented on chip any inter segment traffic would have to use a much down scaled clock for communication. One possible solution to run inter-segment communication at high speed is to equip the bridges will have with buffers and run communication in a pipelined circuit set-up fashion. Furthermore, any traffic crossing one or several segments/bridges would need some kind of system wide arbitration together with some header & routing information appended to the messages.

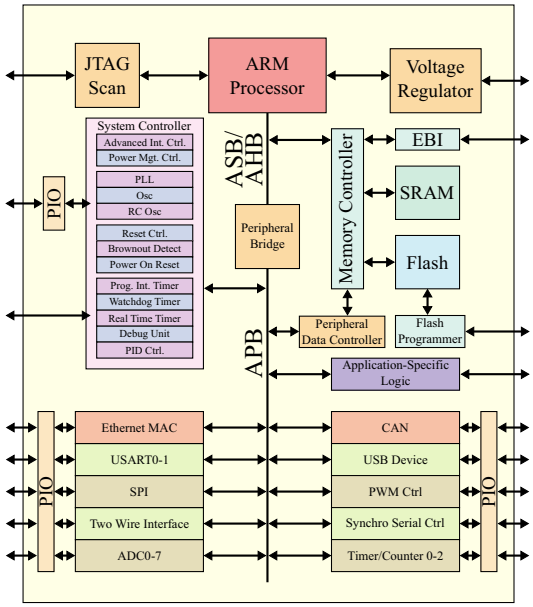


FIGURE 1.10. ARM Based System on Chip

System(s) on Chip

In parallel with the personal computer evolution, the concept of *System on Chip* (SoC) has evolved. Some would claim that a microprocessor could be called a SoC. However, what we today mean when we talk about a system on chip is a system where

- Components are reusable
- Components have already been tested and verified
- Designs specified at the highest possible level of abstraction – but can still be synthesized for implementation

In addition, it is desirable that the platform should be highly programmable.

A typical System on Chip is depicted in Figure 1.10 [SoC-Wiki]. The system is controlled by a micro processor, an ARM Core (Advanced RISC Machine where RISC is an acronym for Reduced Instruction Set Computer) connected to the relatively fast System bus – the ASB (Advanced System Bus).

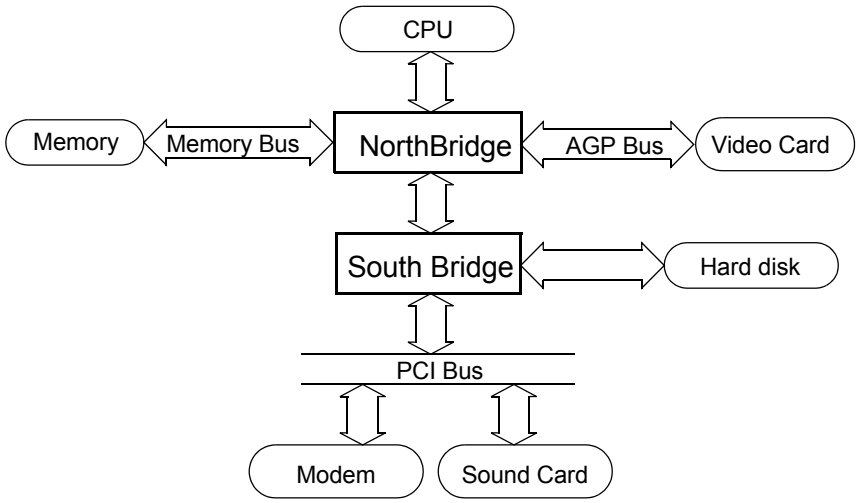


FIGURE 1.11. "Modern" Computer

The main components connected to the system bus are – except for the central micro processor – the memory, the video handling devices, a Digital Signal processor, and the bridge to the peripherals bus. The peripherals are all connected to their own bus since their requirements are different from the components of the system bus. This set-up is a very common set-up and exists for several platforms. If we look into a somewhat old "modern" computer the bus structure will be something like Figure 1.11. Please note the strong resemblance with the bus structure of the "typical" System on chip in Figure 1.10. Both systems have a fast bridging unit/bus connecting the memory and the micro processor. Furthermore, this fast bridge connects a "number crunching" unit. In the case of the personal computer, it is the graphic accelerator/video card. In the case of the system on chip it is usually some sort of DSP as seen earlier. The bridge is also connected to the system bus.

The Multi-core System on Chip

Moore's Law – the observation by the Intel co-founder Gordon Moore that the number of transistors on a chip will approximately double every 18 to 24 months – has enabled a rapid increase in devices that can be fit onto a single chip. This circumstance, in combination with an ever increasing system clock speed has enabled a rapid evolution of performance.

However, the ability to increase the global clock speed is butting up against the laws of physics. The reason is that clock speeds above a couple of GHz develop significant heat due to dissipated power. The power dissipation has two conceptually different components (i) The clock tree drivers needed to drive a clock at that speed to be simultaneous becomes huge. (ii) The high clock speed itself generates/dissipate power when distributed.

In the past, an increase in clock speed was matched with a scale down in supply voltages to keep down the dynamic power dissipation ruled by

$$DynamicPower = CapacitiveLoad \times Voltage^2 \times SwitchingFrequency$$

Today, however, we have reached a point where the supply voltage is around 1V and further decrease is no longer possible due to that the transistors become to leaky. In addition, the lowered voltage also heavily affects the noise margins of the circuit. This can be illustrated by Figure 1.12, depicting of the clock speeds of different Intel processor families [Patterson2008]. Please note the design decision made for the Dual Core processor of 2007 to actually lower the clock speed.

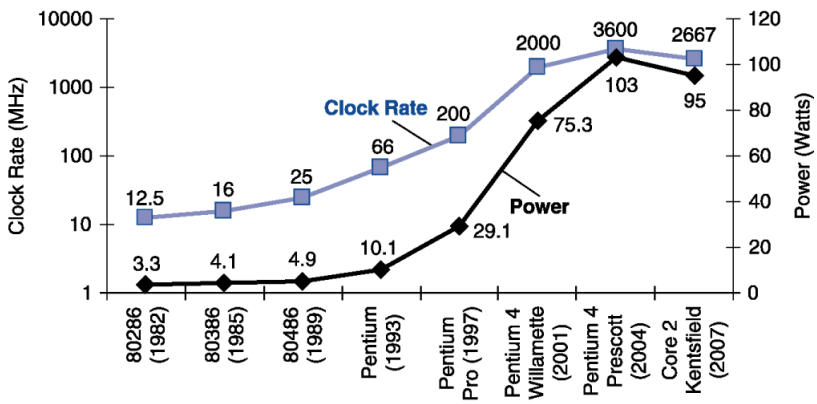


FIGURE 1.12. Intel CPU Trends

This has led to a natural paradigm shift since shrinking is no longer possible with increasing clock speeds and hence chip-global clock distribution is no longer possible. So instead of making the systems faster the strive is now to make them more parallel to utilise the abundance of transistors that the law of Moore yields. The problem is, however, that there exists a limited support for parallelism.

The Shared memory

As we have seen in this chapter the concepts that work *off-chip* are naturally moved *on-chip*. With the concept of System on chip the off-chip bus naturally moved along into the chip.

The main reason why the bus has lasted so long is that communication in a system on chip is done via a shared memory – the main memory. Since the memory is a common resource and – hence – naturally limits the systems' bandwidth *the bus is not the limiting factor on performance alone!*

So why stick to one single memory if that is such a limiting factor? Simplicity! The reason is that the sequential programming model has been the successfully taught in schools and by a majority of the software developer would be considered as the "natural" way to solve a programming task.

In the world of the hardware developers, the relation to parallelism is somewhat ambivalent. The Hardware Descriptive Languages (HDL) like VHDL or Verilog are supporting parallelism, since they are more or less just textual ways of drawing classic schematics; and the art of drawing schematics is inherently a way of describing a massively parallel system. However, when it comes to building a System on Chip the hardware designer falls back on a sequential, single memory model of the system despite the inherent parallelism. Most SoCs that can be found on the web look something like Figure 1.10. A set of different components connected with two busses. One bus for memory traffic and one bus for peripherals. Most of the communication over the processor bus goes via the main memory.

So if evolution is inevitable, why have we dragged along with the bus so long? *Because a paradigm shift takes places due to necessity not of its possibilities.* The one memory makes the bus to stay!

Patrick Gaughan and Sudahakar Yalamanchili early made a prediction in 1993 that large scale parallel architectures would, by necessity, resort to physically distributed memories to provide scalable memory bandwidth [Gaughan1993]. These scalable memories would in turn require an efficient use of high bandwidth interconnections. The target for their projection was the parallel computers, but it will also be valid for the Systems on Chip.

In 1995 William Wulf and Sally McKee realized "the same" thing from another viewpoint. They recognized that rate of improvement in microprocessor speed exceeds the rate improvement in DRAM memory speed and hence will lead to the inevitable – the hit of the memory wall! [Wulf1995]

A paradigm shift takes place due to its necessity NOT as a merit of its potential!

Networks on Chip

The Network on Chip (NoCs) is the natural extension of the on-chip bus. Natural, both, from the system integration evolution perspective as well as from a performance perspective. The system integration evolution makes the off-chip network migrate into the chips. These on-chip networks borrow from wide area computer networks, backplane networks, dedicated parallel computer networks, et cetera.

Performance-wise the increase in communication led to the inevitable – the system grew out the non-hierarchical bus. Furthermore – as mentioned to be one of the driving forces behind Moore's law – the chip sizes are not scaled down with technology – rather the opposite, thus it takes multiple clock cycles for data signals to traverse across a chip. Furthermore – logic will be/is today partitioned into multiple synchronous clock regions that will make data signals only have to traverse a small fraction of the chip area to solve the problem of interconnect effects [Sylvester2001, Kumar2002]. Hence, now the time is right for the introduction of the on-chip network.

Networks on Chip have the key features of

- Scaling well (low order topologies for traffic that is locally dominated)
- Naturally separates communication from computation
- Inherently supports a GALS methodology (Globally Asynchronous, Locally Synchronous) which is a necessity for larger System on Chip
- Potentially predictable electric characteristics
- Better utilisation of wires compared to point-to-point networks
- Inherent support for parallelism

Additional to this some NoCs have

- A packet based delivery scheme
- Guarantees on communication

To further stress the importance of Network on Chips as an *evolutionary step* it can be mentioned that the ITRS (International Technology Roadmap for Semiconductors) [SIA-Design2007] has identified on chip networks as the potential solutions for upcoming system-level design problems like

- SoC reconfigurability – To provide flexible, reconfigurable communication structures
- Design block reuse – Standardised communication structures and interfaces support reuse: IPs with standardized interfaces can be easily integrated and exchanged, and communication structures reused.

The upcoming chapter *Networks on Chip* is intended to cover the major aspects on Networks on Chip and its potential as well as giving an in-depth introduction to the topic.

2 Networks on Chip

Since the thesis is about architecture enhancements for providing communication within a Network on Chip a separate chapter about Networks on Chip (NoC) seems proper. In this chapter I will discuss why NoCs are needed and what is so special about Network on Chip – in comparison to its siblings *off* chip. Furthermore, I will talk about the characteristics of the *Network Layer* in Networks on Chip which I would say is defined by the *Topology* in combination with how packets are delivered by the network. The packet delivery has the steps of Segmentation/Packetisation, Ingress Queuing, Admission, Routing, Exit, Egress Queuing, and Desegmentation in combination with Arbitration. Once this is presented the concept *Quality of Service* will be discussed and the chapter will be finished with a few words about the layered approach to Network on Chip.

2.1 Why NoCs?

The motivation, need, and inevitability of Networks on Chip have been stated in more or less any publication in the area. I have tried to collect the most convincing and representative papers that cover most of the main arguments used.

The problem of scaling the interconnect was first articulated in the "early" NoC publications. In Pierre Guerrier and Alain Greiner's paper *A Generic Architecture for On-chip Packet-switched Interconnections* [Guerrier2000], they stated the "obvious" fact that busses are inherently non-scalable; and this for the two main reasons (1) the bandwidth of the bus is shared by all attached devices (2) the clocking frequency for wiring becomes tightly constrained by the electrical properties of the deep sub-micron processes.

Furthermore, they also pointed out that every device connected adds parasitic capacitance and hence the electrical performance degrades with growth; even the arbitration becomes slower with an increasing number of masters. Despite that the network approach they advocated would consume more area than the corresponding bus, they still claimed it to be better since the aggregated bandwidth scales with network size and the same router could be used as a modular component for any size of network. In addition, they saw the advantage of enabling faster BIST and higher throughput thanks to the pipelined fashion in which packets were distributed.

In William J. Dally and Brian Towels paper *Route Packets, Not Wires: On-Chip Interconnection Networks* [Dally2001] the predictability of the electrical characteristics were pointed out as a main advantage; the predictability results in low crosstalk and opens for high-speed aggressive signalling circuits. Moreover, the advantage of the use of standard interfaces is pointed out as well as the fact that dedicated wires are highly underutilised.

The problem of increasing wire delay was articulated by Luca Benini and Giovanni De Micheli in *Powering Networks on Chips* [Benini2001]. They recognized that global wires spanning a significant fraction of the chip will carry signals whose propagation delay will exceed a clock period. Furthermore, the distribution of the clock signal from a single source will be extremely hard to achieve without introducing large skew. This led them to suggest a GALS methodology for System on Chip. GALS (Globally Asynchronous Locally Synchronous) refers to the idea that the design should be divided in smaller synchronous blocks that communicate asynchronously with other synchronous regions [Meincke1999 and Muttersbach1999]. Jens Muttersbach et al. also articulate that choosing the ideal period for a hypothetical global clock will be difficult since a global clock is used in anything from memory accesses to high-performance pipelined data paths.

Kees Goossens et al. saw four major main advantages of the NoC design scheme in comparison to a system composed of dedicated wires in their paper *Networks on Silicon: Combining Best-Effort and Guaranteed Services* [Goossens2002]. (1) First, a NoC would reduce the wire congestions around the resources since wires/channels now could be reused. (2) Second, the worst case dimensioning could be calculated based on an *average* for the traffic flows *potentially* utilising a channel instead of as a worst case for every single traffic flow. (3) As third comes the benefit from a designer's layout perspective where the design of the router and network could be fixed.

(4) The last and most important benefit was that the *communication* could be decoupled from the *computation* which would enable compositional layout and timing closure of the different resources/IPs of the system. This in concrete means that the NoC infrastructure can be reused and its design cost can be amortized over multiple designs.

The global wire delays in relation to the clock made Paul Wielage and Kees Goossens [Wielage2002] suggest that an architecture that would be future-proof had to be constructed from processing nodes (Resources) that did not grow in complexity with technology – instead the processing nodes would scale in numbers. The reason is that the global wire delay stays at best constant under technology scaling and hence these wires become effectively slower in comparison to the gate delay. As an example they state that for a 50 nm technology crossing a chip takes about between 6-10 clock cycles – this clearly invalidates the idea of a fully synchronous chip. As a feasible size for these processing nodes that would scale with technology 50k-100k gates are suggested [Ho2001 and Sylvester2001]. Even though we could make these global wires – they would be very costly from an energy perspective since repeaters would have to be inserted; this since the propagation delay of wire increases quadratically with its length for a given energy.

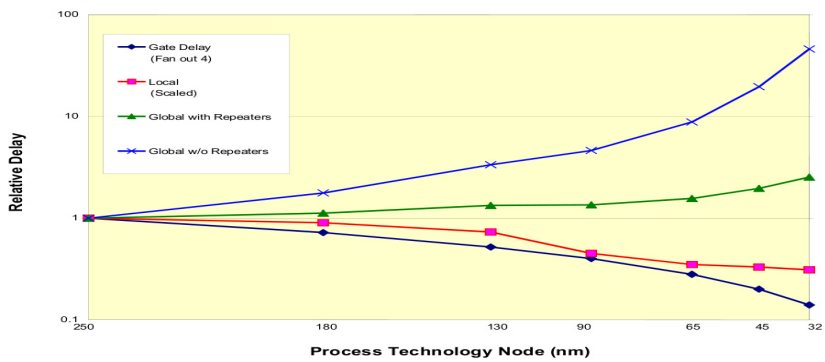


FIGURE 2.1. Delay for Metal 1 and Global Wiring versus Feature Size

This wire scaling problem can be illustrated with Figure 2.1 taken from the ITRS (International Technology Roadmap for Semiconductors) published by SIA (United States Semiconductor Industry Association) [SIA-Interconnect2005] where the gate delay of Metal layer 1 is plotted together with local and global wire delays. As it can be seen the global wiring delay gets worse with shrinking geometries but the gate delay together with the local wire delay get smaller.

From what is stated above it is easy to conclude that the inevitable down-scaling is the root of all problems or as Ron Ho et al. put it *"One interesting view of scaling is that the real problem is not the wire, but rather the increasing complexity that the scaling enables" [Ho2001].*

2.2 SO WHAT'S SO SPECIAL ABOUT NETWORKS *ON* CHIP?

Off chip networks have been around for a long time and come in many flavours. Typically, we have telephone networks, wide area computer networks, dedicated parallel computer networks et cetera. Of these networks, a NoC interconnect topology mostly resembles the interconnect architecture of high performance parallel computer systems since the telephone and wide area networks do not posses local proximity in combination with that they exhibit more nondeterminism [Benini2002]. However, there exist a fundamental difference – in a high performance computer network the energy is usually not a matter of concern but the wires are; for the Network on Chip, it is the opposite. Both the NoC and parallel computer networks provide mechanisms for data transfer between processing nodes or between processors and memory modules [Grama2003]. The NoCs, however, do also possess the ability to facilitate energy-efficient local communication patterns due to their modular structure, which make them, especially suitable for Multi Processor Systems on Chip (MPSoC) solutions [Pande2005]. Unfortunately, the power restrictions of NoCs in comparison to high performance computers make the NoC limited in bandwidth. According to Stefano Santi et al. [Santi2005] many high performance computing networks do not have any support for Quality of Service. The two presumed rationales are (1) that the very high-speed links in high performance networks are providing capacity well above what it expected for any typical application. (2) High performance computing networks are optimised for giving the best *average* performance due to the expected cost of giving hard guarantees.

The two biggest differences between a 'typical' off-chip network and today's on-chip networks are the, aforementioned, power aspect and the requirement of a planar topology. Today, however, the need of planar topologies is somewhat alleviated with the advent of 3D networks [Pavlidis2007]. Another important aspect is the requirement of a limited buffer space. The reason these differences are of significance is that otherwise already established technologies could have been brought – with small or no changes – on board on the System on Chip. Now the situation is somewhat more intricate and calls for special solutions dedicated to NoCs.

The limited buffer space requirement was a point made by William J. Dally and Brian Towles [Dally2001] and refers to that the buffer space in an on-chip network directly impacts the power and are overhead and hence must be kept to a minimum; in traditional off/inter chip networks the area has not been an issue but instead the pin count of the chip has been the bottleneck. This means that the channels used for on-chip communication can be relatively wide in comparison to the 8-16 bits used for the inter-chip networks.

In addition to the power aspect – or *the energy constraint* – Luca Benini et al. [Benini2001] also make the observation that the NoCs differ when it comes to *design time specialization*. With design time specialization, they envision a vertical design process where all layers of the design are specialized and optimised for the target application domain. In comparison with the off-chip network standardization is only employed for specifying the abstract network interfaces for the end-nodes and hence the network itself can be highly customized for the application domain. In a traditional off-chip network, the different sub-components would have been built with standard interfaces to support full modularity.

Another point made by Andrei Rădulescu and Kees Goossens is that for off-chip networks out-of-order delivery is the typical scenario due to reordering and packet drop in the switches [Radulescu2004]. In NoCs, it is possible to circumvent this by using e.g. static routing or with a Virtual Circuit scheme. Further they point out that deadlocks that may occur in off-chip networks could be circumvented in the NoC by using an "intelligent" routing strategies e.g. the *Turn model* [Glass1992].

2.2.1 Parallel Computer Clusters → MPSoC → SoC

Despite that it is a known fact that Networks on Chip are suggested to be used in systems ranging from heterogeneous custom designed ASIC like Systems-on-Chip to quite homogenous and regular Multiprocessor System-on-Chips (MPSoC) as well as in the communication infrastructure in high-end FPGAs this is very rarely articulated in the collected NoC literature; this even though the requirements of a Network on Chip will change quite dramatically in terms of traffic patterns, throughput, and latency as well as in the mapping process. It is hard to come up with a solution that fits all but some groups have done some observations: Srinivasan Murali et al. have articulated the idea that the more predictable the traffic patterns are at design time the more appropriate a custom topology is [Murali2006].

Jerry Tao Ye et al. have analysed the requirements on the on-chip communication infrastructure for the traffic of an MPSoC system employing a shared memory paradigm [Ye2004]. They characterise a typical MPSoC system to consist of several processor nodes connected by an on-chip network. Each node consists of a Central Processing Unit (CPU) – and possibly a Floating Point Unit (FPU) – employing a two level local cache. All nodes logically have their "own" memory but that can physically either be a common memory block or a distributed memory. This means that most of the on-chip memory traffic will be write, read, or cache related operations. Among their findings, they found a mesh based structure employing a contention-look ahead routing scheme to be appropriate. For a SoC based system this may not have been the case.

Another observation that can be made is that the Multiprocessor system is easier to map onto a regular structure due to its homogenous nature. The traditional Systems on chip often have a set of components or building blocks of vastly varying sizes, which make them hard to "squeeze" into e.g. a mesh like structure. For the mapping of an SoC, the Æthereal is a good example of an appropriate architecture, since there are few restrictions when it comes to mapping [Goossens2005].

2.2.2 Message Passing vs. Memory Sharing

Since a "typical" NoC in some sense is a parallel architecture it will inherit concepts from the traditional parallel (computer) architecture field as well. In this field several different parallel programming models exist and *Shared Address Space* together with *Message Passing* are the two most important.

The idea behind the *Shared Address Space* aka the *Shared Memory* model is that parallel instances communicate via a bulletin board where everyone can post information to be shared by peers. The big advantage of a Shared Memory architecture is that no special hardware is required. It inherits the 'simplicity' of sharing data on a memory attached to a bus. The disadvantage is that it requires the programmer to handle the synchronization.

Typical examples of shared memory architecture are the Multiprocessor Systems-on-Chips (MPSoC) that have been used in high performance embedded systems such as network processors and parallel media processors. This kind of architecture will have a traffic behaviour that are very much concerned with the updating and flushing of local caches against some main memory (Cache and memory transactions and Cache coherence operations). This kind of architecture will require "short" packets for notifications and memory request as well as "long" packets for actual data written or read [Ye2003 and Ye2004].

For the *message passing* approach data transactions are performed explicitly. This approach requires primitives for send, receive, etc. together with an API supporting this – preferably in hardware. The best known API for message passing is MPI [MPI1993]. Message passing conceptually inherits its behaviour and implementation from traditional network of cooperating workstations, or clusters. The advantage of message passing is that it is easier to optimise programs since all communication is explicit.

Some groups (us included) have implemented primitives for message passing in hardware since it enables a rapid development of adopters to other communication standards, e.g. AXI [Millberg2004b]. Andrei Rădulescu and Kees Goossens did, however, point out that existing protocols such as OCP, AMBA are not *directly* applicable; the reason is that these protocols assume an ordered transport of data. If one Master initiates requests in a specific order they will automatically arrive in that order – for NoCs special care needs to be taken if this is to be guaranteed [Radulescu2004]. In a later publication Andrei Rădulescu at al. suggest that a shared memory abstraction via transactions could be used to provide a smooth transition from traditional buses to NoCs [Radulescu2005]. These transactions, or *request messages*, are issued by master IP modules and handled by the addressed slave modules which reply with response messages. This not only provides backward compatibility with existing bus protocols such as AXI [AXI] or OCP [OCP] it also allows efficient implementations of future protocols.

Even though there are several ways to implement memory sharing on a message passing architecture, and vice versa, it is important to understand the mechanisms used and provided by a particular NoC to be able to utilise it in the most effective way. Furthermore, this distinction is of high importance when two NoCs are to be compared – a NoC providing primitives for Message Passing will most likely not perform well in a benchmark written in a Shared Memory style (and the other way around).

2.3 NETWORK ON CHIP TOPOLOGIES

The topology is defined by how the switching elements of the network are connected as well as how the network interfaces are connected to the network. By topology, we hereby mean the *logical* interconnection structure of the network graph. The logical structure often has a one to one mapping to the physical layout.

NoC Lingua

In order to be able to discuss the topology issue I feel that some NoC lingua could be of use. The terminology used within the NoC community may have converged over the years but still there is no consensus of what expression to use, e.g. *switch* or *router*? This, in combination with, that any document referring to older documents has to tackle the problem of how to denote things in a consistent way. To avoid confusion when discussing, and referring to, various articles I have chosen to adopt to the currently used lingua in the article. This, on the other hand, comes with the penalty that the terminology used throughout this thesis will not be consistent. In the upcoming pages, I will try to shed some light around the NoC lingua.

A typical System on Chip (SoC) of today contains *Resources* like Processors, Digital Signal Processors (DSPs), Memory Controllers, I/O units, Memories, etc. These resources are centred around one or more communication infrastructure devices – typically bridged busses. Back in 1997 Rajesh K. Gupta and Yervant Zorian defined these *resources* or *cores* as pre-designed, pre-verified hardware pieces that can be used as composable parts of a larger system on chip [Gupta1997]. If the context is a System on Chip or a Network on Chip the resources may be referred to as *IP*s (Intellectual Properties) or *Processing Nodes* [Wielage2002] as well. Some authors have chosen to further divide these resources into *Processing Elements* (PEs) and *Memories*.

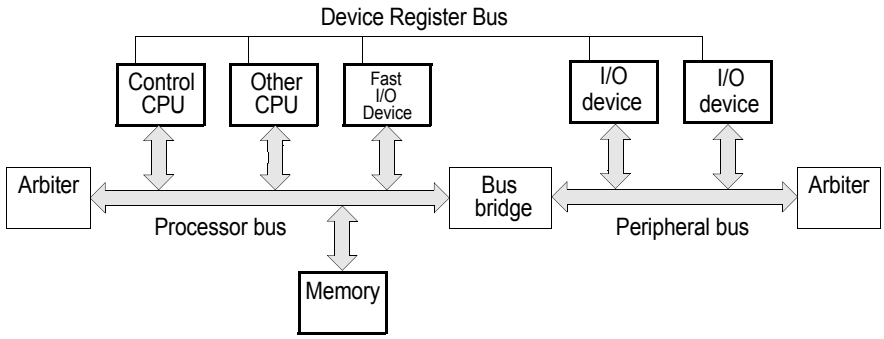


FIGURE 2.2. "Typical" SoC – The IBM Core Connect

In Figure 2.2 a "typical" SoC is represented by a system utilising the IBM Core Connect [CoreConnect]. If this scenario would be moved into the "typical" NoC realm the bus based communication infrastructure would be replaced with a network and the resources would be equipped with *Network Interfaces (NIs)* to access the network. A cluster of a resource/Network Interface/Switching element would be called a *Node* as depicted in Figure 2.3. The Nodes are further connected to each other with *links* – basically a set of wires.

In the traditional bus based system participants are defined as either *Masters* or *Slaves* where the masters are the initiators of communication. The actual transfer of data in communication is called a *transfer.*

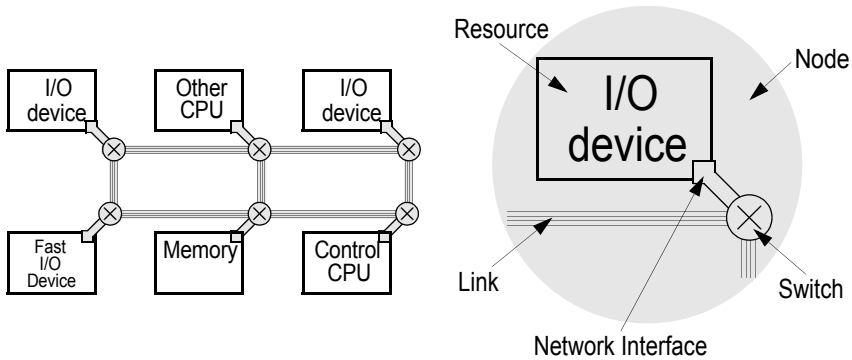


FIGURE 2.3. The Resources, Nodes, Switches, Links, and their Relation

In Networks on Chips these transfers take the form of *packets* or *messages* where messages are the containers from the application's point of view.

The packets are the entities of communication at the network level and a message may be composed of several packets that can be sent independently over the network (See *Store-and-Forward or Packet Switching* on page 64). Packets (or Messages) could also be divided into *flits* (**fl**ow control digi**ts**) that are sent through the network in a *pipelined* fashion (See *Virtual Cut-through* on page 65 or *Wormhole switching* on page 66).

Communication Mechanisms – Switching and Routing

In the collected NoC literature, the concepts of *Routing* and *Switching* are used interchangeably even though they are different things. The reason is a clash in how to define things within the NoC realm and how they are used in the internet realm. Traditionally switching (or *bridging*) is an activity that is handled by the second layer, (L2 – data link), within the OSI model and involves moving packets between devices by using the physical address. The concept of routing is handled by the network layer (L3). This is how the concepts of routing and switching are used within the internet [Cisco2003]. An internet switch can decide where a packet should be sent by examining the MAC address (Media Access Control) within the data link header of the packet (the MAC address is the hardware address of a network adapter). A switch maintains a database of MAC addresses and what port they are connected to. This means that a switch only can forward a packet to a port in its database; a packet with unknown address is simply dropped – no intelligence here!

A router, on the other hand, can decide where to send a packet using the Network ID within the Network layer header. It then uses the routing table to determine the route to the destination host. Routers communicate with one another and maintain their routing tables through the transmission of a variety of messages. The routing update message is one such message that generally consists of all or a part of a routing table. By analysing routing updates from all other routers, a router can build a detailed picture of network topology.

On the internet, the process of routing involves two basic activities: determining optimal routing paths and the transportation of the packets that contains the real data from source to destination. In the context of the routing process, the latter of these *is referred to as packet switching*. Although packet switching is relatively straightforward, path determination can be very complex.

Given this information above clearly most NoC architectures are having routers that are performing a very basic variant of routing close to the border of switching. For instance, a communication mechanism like Wormhole switching (see Section "Wormhole Switching," on page 57) is actually performing *routing of the head flit but switching on the body flits*! So to avoid hair-splitting in the process of describing different aspects of the Networks on chips I have chosen to stick with the respective terminology of the paper currently discussed, and hence I am using the terms routing and switching interchangeably. Consequently, the same applies for the terms *router* and *switch* as well!

Communication Infrastructure Classifications

When classifying communication infrastructures and networks three basic topology types – together with a hybrid superset – can be identified as depicted in Figure 2.4

- *Shared Medium – e.g. busses*
- *Direct Network – e.g. meshes and trees*
- *Indirect Network – e.g. multistage interconnection networks*
- *Hybrid Network*

The definition of a shared medium is straight forward – if the "network interfaces" are sharing the medium, i.e. no switching element is present then the communication infrastructure is classified as *Shared Medium*.

If all switching elements in the network are connected to at least one network interface the network is said to be a *direct network;* if the network has more switching elements than network interfaces it is classified as *indirect*. Any mix of the aforementioned topologies would be a *hybrid*.

In the parallel computer network community, a direct network is a network that consists of point-to-point communication links between processing nodes; whereas the indirect network is a network utilising communication links connected by switches [Grama2003].

All classes of topologies, of course, have their benefits and weaknesses. The main components that decide a particular topology's suitability are the number of nodes in combination with the traffic pattern.

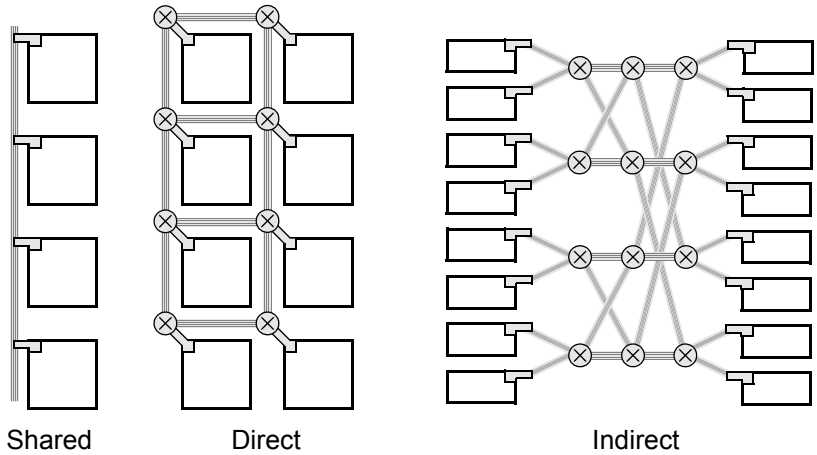


FIGURE 2.4. The Shared Medium, Direct, and the Indirect Network

Shared Medium

If the number of nodes is small and/or there is no parallelism in the system a "shared" topology is an excellent choice; typical example is the traditional SoC with one memory and no utilised parallelism. The shared medium requires very little hardware, i.e. no switching or routing elements. For a traditional bus, an arbitrator is however required. Furthermore, the shared medium offers a very natural way of broadcasting information. The shared medium's two major drawbacks are the lack of scalability – both from a bandwidth perspective as well as from an electrical perspective; long wires mean high capacitance means power loss/slow speed.

Direct Network

Direct networks can be seen as a collection of small point-to-point networks between subsets of nodes in a network. The nodes are, as mentioned earlier, the collective names for a switch, a network interface and a resource "bundled" together. The immediate benefit of the direct network in comparison to the shared network is that the total bandwidth of the network scales with the increased number of nodes. I addition, the direct network offers a natural path diversity that adds robustness to the network. Typical examples of direct networks are the mesh, the torus and the hypercube.

Indirect Network

In the indirect network, some of the switches are not connected to any network interfaces and have the sole purpose of being intermediate switches to alleviate communication. The indirect network has their history in telephone networks where connections between peers in a system are set up and torn down. In the world of SoC, they have been used as high performance interconnect structures for advanced bus based systems e.g. AMBA (Advanced Microcontroller Bus Architecture) where several outstanding transactions can be performed in parallel. In NoCs indirect networks have been used in *SPIN* [Guerrier2000], which is a fat-tree implementation, the *Butterfly Network* used by Partha Pratim Pande et al. [Pande2003] and the *Hierarchical Star Topology* used by Se-Joong Lee et al. [Lee2005b].

Hybrid Network

The hybrid networks are an umbrella term for combinations of the three network types above. An example of a hybrid network could be a "standard" NoC topology – e.g. a mesh – that is used for packet based global interconnections connecting local buses for nearest-neighbour communications between e.g. a micro processor and a local memory [Kumar2002]. In a strict sense most networks could be called hybrid since the switches themselves internally is small indirect networks and hence a typical NoC composed of switches would fall into this category.

Network Characteristics

When characterising a network several different metrics can be used. The relative importance of these metrics highly depends on the intended application. Some metrics can more or less easily be analytically determined from the topology of the network. Other important characteristics need to be derived from simulation e.g. throughput, latency, etc.

These characteristics, however, tend more to be *consequences of the particular service employed* than real characteristics that stem from the topology and hence will be referred to as *Service Characteristics* – see *Service Characteristics* on page 72.

Below is a summary of the most common network characteristics used:

- Diameter – largest distance between any pair of nodes in the network. This will heavily affect applications with global traffic patterns whilst applications with mostly local communication will be unaffected.
- Bisection bandwidth – the smallest bandwidth across any bisection of the network. The bisection must divide the network into two almost equal halves. The two opposites in this "contest" are the shared bus that possesses the minimal bisection bandwidth of 1 and the fully connected network has a bandwidth that scales with the number of nodes. In general for traffic patterns without locality the network's bisection bandwidth will become a limiting factor unless the network scales with both the number of nodes and growing average distance. If the bisection bandwidth is divided by the number of nodes in the network it gives an interesting measure of how well the bandwidth of the network scales with an increasing number of nodes – bisection bandwidth per node.
- Redundancy/Path diversity – a redundant network has the potential of being tolerant of faulty links as well congestions in the network; this under the condition that it that also facilitates a flexible routing scheme.
- Aggregated bandwidth/throughput – the total bandwidth of the whole network. This is an upper bound of how well a network will suit an application with a lot of parallel communication. This measure is interesting in combination with other metrics; for instance, a network with no path diversity and/or a low bisection band width may not be able to be used at its full potential.
- Regular vs. Irregular – a regular network topology is defined in terms of some sort of regular graph structure (such as rings, meshes, hypercubes, etc.)
- Link Bandwidth – the rate at which data can be transferred over a separate link/node to node connection.

- Blocking vs. non-blocking – A non-blocking network will support that N source nodes can connect to N destination nodes in any one-to-one combination. A non-blocking network will always be able to make a connection between any pair of non-busy nodes regardless of any already existing connections. Example: a 2×2 mesh will be non-blocking since any pair-wise combination of three sources to three destinations can never block a fourth connection pair to be set up. This is obviously only true if the path set-up mechanism is intelligent enough to find the "free" path. On the other hand; a 4×4 *will be blocking* due to a limited bisection bandwidth. In other words: if the network is cut in half, there are eight nodes on the right side and eight on the left side. The number of links connecting these two halves are only four – hence connecting more than four nodes from one side to any four nodes on the opposite side will not be possible. This is, on the other hand, obviously true at all times regardless of how intelligent the path set-up mechanism is since no "free" path can exist.
- Switch degree – how many links/pairs of in- and out-puts every switch has. The higher degree the more flexible routing decisions – but at the price of a higher implementation cost per switch.
- Symmetry – does the network looks the same from every node? A typical example of a symmetric network is the torus.
- Homogeneity – all the nodes and links are identical. As it can be seen above the list of different characteristics is rather long, and it may be very hard to foresee all the possible implications when the choice of the network type is done. However – to articulate the characteristics early in the design process may rule out the worst surprises.

2.3.1 Network / Communication Infrastructure Types

Traditionally, systems on chip have either employed a bus structure or a custom wiring scheme (point-to-point). Qualitatively this was identified as a bottleneck by Pierre Guerrier & Alain Greiner [Guerrier2000] and can be seen as the starting signal for the NoC research community. Later Evgeny Bolotin et al. quantitatively made a comparison of the bus, the segmented bus, the point-to-point, and the mesh [Bolotin2004a]. Their findings can be summarized in Table 2.1 where asymptotic cost functions of the different architectures are presented.

TABLE 2.1. Asymptotic Limits for Area, Power and Frequency

Topology	Total Area	Power Dissipation	Operation Frequency
Bus	$O(n^3\sqrt{n})$	$O(n\sqrt{n})$	$O\left(\frac{1}{n^2}\right)$
Segmented Bus	$O(n^2\sqrt{n})$	$O(n\sqrt{n})$	$O\left(\frac{1}{n}\right)$
Mesh	$O(n)$	$O(n)$	$O(1)$
Point-to-Point	$O(n^2\sqrt{n})$	$O(n\sqrt{n})$	$O\left(\frac{1}{n}\right)$

In the analysis, they assume that the resources are of equal size and arranged in a grid. Further they assume a uniform traffic distribution for the traffic between the resources; to simplify the analysis the load capacitance of the interconnection architecture is assumed to depend only on the link length. From this they derive a set of analytical expressions for area, power and operating frequency for each interconnection scheme. In Table 2.1 I have chosen to show only the asymptotic limits of these functions. The surprisingly large area for the bus and the segmented bus is explained by with *"... the NS-bus requires an excessive bus width [...] to compensate for the lack of parallelism and for the low operating frequency due to its larger load capacitance" [Bolotin2004a].*

One substantial difference between traditional off chip networks and the Network on Chip is the assumed "restriction" on a planar topology [Santi2005]. Since the NoC is restricted to a 2D layout – due to the silicon mapping constraint – low order topologies are preferable since they have natural planar mappings [Culler1999]. Even though we are constrained by a planar topology, there are still many options when the network topology is to be chosen.

In order to characterize the communication infrastructures (point-to-point, busses, and NoCs) that is used, or proposed, the subsections below are devoted to that. The motive for choosing the variants below are that they are either traditionally – or contemporarily – used in SoC design, or they are the most common NoCs [Salminen2008, Bjerregaard2006, and Moraes2004]. Since the mesh is the topology that has been used in my work the mesh is also dealt with in most depth.

- Point to point
- Shared and Bridged Bus
- Fat-tree
- Extended Bidirectional Ring
- Custom Topologies
- Mesh and Torus

Point to point

Together with the bus the point-to-point connection scheme has been the most common in traditional SoC design. For a moderately sized design, it works well but since every new design needs a completely different layout for its wiring scheme the point-to-point becomes expensive in terms of design effort over several designs. The reason is that a point-to-point wiring scheme by necessity must be ad-hoc. With shrinking geometries the two design phases of logic synthesis and placement can no longer be separated due do non-neglectable wire delays and hence it is crucial to be able to estimate wire delays already during synthesis [Atienza2008]

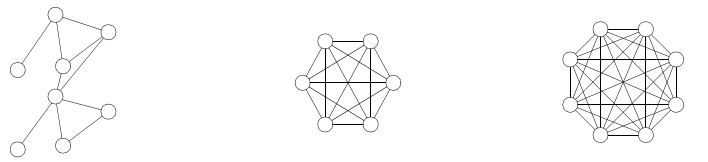


FIGURE 2.5. Point-to-point Networks of Varying Density

The more dense/complete the point-to-point graph is the higher aggregate bandwidth. For the fully connected graph, the diameter is as low as one but the wiring cost grows as N(N-1) – see Figure 2.5. As pointed out by Kees Goossens et al. [Goossens2002] dedicated point-to-point wires suffer from three major problems. (1) These wires must be dimensioned for worst case conditions both from the electrical perspective (cross talk and wire spacing) as well from the perspective of the sheer volume of traffic. (2) Since some resources will be connected to multiple other resources they will suffer from wire-congestion. (3) This type of architecture is inherently non-scalable since it exists an implicit dependency between the resources from a mapping perspective which is layout specific. One further point made by Christian Grecu et al. is that the propagation delay of long wires will exceed the limit of one clock cycle – this can be overcome by the insertion of FIFO buffers. However, the insertion tends to be very ad hoc and hence will make the design hard to generalize or scale [Grecu2005].

Shared and Bridged Bus

As mentioned above – the bus is traditionally the most commonly used platform for communication for inter and external chip communication. The bus comes with some nice features and some less nice properties.

On the positive side – the bus is very simple and requires little hardware except for a mechanism handling arbitration. In addition, the bus offers a very natural way of broadcasting information.

Since the bus is a shared medium it also shares the shared medium's major drawback – lack in scalability. Neither does it support multiple concurrent communications and can therefore, not utilise any inherent parallelism of the system.

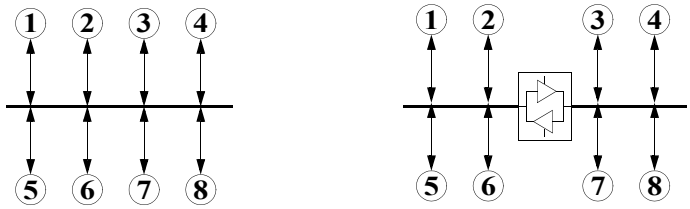


FIGURE 2.6. The Shared Bus and the Bridged Bus

One way to reduce the length of the bus – and hence the capacitance – is to split it as depicted on the left side in Figure 2.6. By segmenting the bus not only the length is shortened, transfers of data are now possible to perform in parallel in the different bus-segments. This, however, comes at a cost – the more segments the more control (hardware) is needed to route the data to the right destination; the segmented bus will more and more resemble a network! The problem with this approach is that it tends to become very ad hoc and this means that solutions are locally optimised and hence become hard to automate, port, or scale [Grecu2005]. Paul Wielage and Kees Goossens suggest that the bus could serve as a first, local layer of communication infrastructure [Wielage2002]. This local bus should be equipped a local bridge to a global network connecting all the busses. The gain is twofold: the network interfaces are often over-dimensioned for any IP connected to it – several IPs could beneficially communicate locally before the need for a network actually arises. Moreover, a bus interface is normally much cheaper than a network interface.

Fat-Tree

The most famous fat-tree architecture in NoC history is the SPIN architecture (Scalable Programmable Integrated Network). SPIN origins from one of the first NoC papers – Pierre Guerrier & Alain Greiner's paper *A Generic Architecture for On-Chip Packet-Switched Interconnections* [Guerrier2000]. The fat-tree is a typical example of an *indirect network*; the topmost layer of switching elements are only indirectly connected to the resource elements and have the sole purpose of being "bridges" between different parts of the network as shown in Figure 2.7; the path "upwards" is dynamic but fixed "downwards".

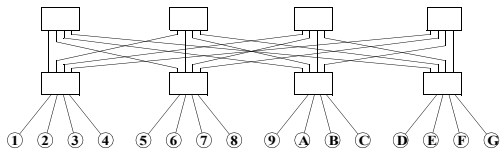


FIGURE 2.7. 16 Node Fat-tree

This particular network of eight nodes has a diameter of three and a bisection bandwidth per node of 1, which is relatively high – hence the name; a traditional tree network would have a bisection bandwidth per node that rapidly decreases with network size since the bisection bandwidth is constant. In general fat-tree networks have a diameter that is proportional to log_2N-1. The choice of architecture was motivated by that Charles E. Leiserson had formally proven that the fat-tree would be nearly the most cost efficient for VLSI realisations from a routing perspective [Leiserson1985]; this decision was, however, re-evaluated in their sequel architecture *DSPIN* (Distributed, Scalable, Programmable, Integrated Network) where they decided to move to a mesh-based architecture. The major shortcomings of SPIN were (1) that it was a very inflexible architecture since it was hard to modularise; (2) also it was difficult to synthesise and (3) the routing decision was centralised and hence did not work in within a GALS paradigm [Panades2006].

Extended Bidirectional Ring

The traditional bidirectional ring is actually the simplest form of a torus, that is, an N-ary 1-cube. For the ring the process of routing is straightforward – the clockwise or counter-clockwise direction are simply selected depending on the shortest path. The benefits of the ring is a small footprint thanks to a very uncomplicated routing process in combination with being a sparse network with low degree (2) – the wiring cost hence grows as N. Also the physical mapping process can be simple in an SoC with processing elements of varying sizes in the same way the bus is/was. On the down side is a large diameter and a limited bisection bandwidth.

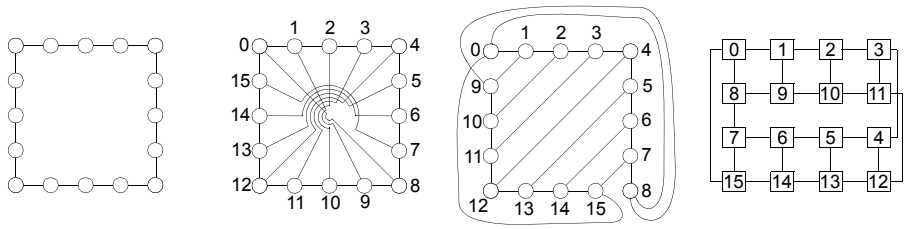


FIGURE 2.8. Point-to-point Networks of Varying Density

Extended Bidirectional Ring – The Spidergon

An interesting architecture that tries to overcome the weaknesses of the traditional bidirectional ring whilst keeping its benefits is the Spidergon NoC of ST Microelectronics [Coppola2008]. In the Spidergon the degree of the nodes have been slightly increased (3) with a bidirectional "shortcut" to the opposite side of the ring. The obvious gain is a lowered diameter (N/4) in combination with a slightly higher bisection bandwidth. At a glance the extended ring appears hard to map onto a 2D surface due to the many crossing connections in the centre as depicted in Figure 2.8b. However, if the network is "cut in half and twisted" the many crossing connections are straightened out so that only one crossing remains (Figure 2.8c). This simple change of viewpoint now enables even the Extended ring to be mapped in a similar manner to the traditional ring.

Even though the switches are simpler than the traditional mesh thanks to the lower degree (3 in comparison to 4) the wiring cost is of the same magnitude if the Spidergon NoC is used to connect resources of a traditional mesh layout.

Custom topologies

In the literature, there are several examples of custom topologies used [Bertozzi2005, Jalabert2004, and Murali2006]. Groups that have implemented custom topologies report that in comparison to the "standard" topologies (mesh, torus, etc.) they have achieved improved performance and less power and area overhead when used for SoCs.

In Antoine Jalabert et al.'s paper about ×pipes they put forth the following reasons for choosing an irregular topology [Jalabert2004]; Since many SoCs involve heterogeneous cores – having varied functionality, size, and communication requirements – a regular interconnect, designed to match the requirements of a few communication-hungry components will be largely over-designed with respect to the needs of the remaining components. As a concrete example they put forward their implementation of an MPEG4 decoder. In this decoder, the embedded memory (SDRAM) is much larger than all other cores and is also the critical communication bottleneck. As seen in Figure 2.9 the block sizes are highly non-uniform they would not match a regular, tile-based floorplan [Murali2006]. Also, many neighbouring blocks do not *need* to communicate and hence the regular structure will over-provision the communication facilities; that is, there is a significant risk of under-utilizing many tiles and links.

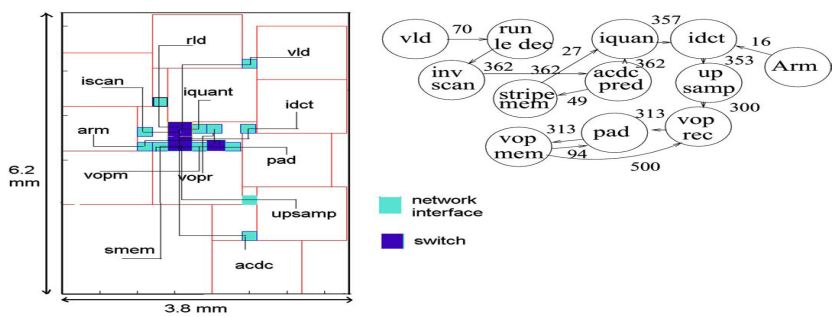


FIGURE 2.9. Custom topology mapped Video Object Plane Decoder of Srinivasan Murali et al.

Srinivasan Murali et al. – on the other hand – argue that regular and redundant topologies are required for on-chip systems where the traffic characteristics of the system cannot be predicted statically, as in multiprocessor systems. For most SoCs the system is designed with static (or semi-static) mapping of tasks to processors and hardware cores and hence the communication traffic characteristics of the SoC can be obtained statically.

Among other reasons for using NoCs is the fact that the interconnect structure and wiring complexity can be well controlled. When the interconnect is structured, the number of timing violations that occur during the physical design phase is decreased. Such design predictability is critical for today's SoCs for achieving timing closure. Early works on NoC topology design assumed that using regular topologies (such as the mesh) would lead to regular and predictable layouts. While this may be true for designs with homogeneous processing cores and memories, this is not true for most SoCs as they are typically composed of heterogeneous cores. This because the core sizes of the SoC vary and the floorplan of the design does not match the regular, tile-based floorplan of standard topologies.

Srinivasan Murali et al. implemented a set of six different SoC benchmarks consisting of e.g. a video processor, an MPEG decoder, a Video Object plane decoder, etc. The six designs were implemented on (1) a traditional mesh, (2) a mesh where the "unnecessary" links have been removed and (3) a custom NoC. An excerpt of their findings is presented in Table 2.2, and as it can be seen it is in favour of the custom NoC in terms of area, power and performance.

TABLE 2.2. Topology Comparison for VPOC, MPEG, and VODP of Srinivasan Murali et al.

Application	Topology	Power (mW)	Average Hops	Area (mm^2)
VPROC	custom	79.64	1.67	47.68
	mesh	301.8	2.58	51.0
	opt-mesh	136.1	2.58	50.51
MPEG4	custom	27.24	1.5	13.49
	mesh	96.82	2.17	15
	opt-mesh	60.97	2.17	15.01
VOPD	custom	30.0	1.33	23.56
	mesh	95.94	2.0	23.85
	opt-mesh	46.48	2.0	23.79

Davide Bertozzi et al. did a similar exercise to Srinivasan Murali et al. in 2005 and their conclusions agree. They also report significant area and power improvements with an automatically synthesised custom NoC [Bertozzi2005]. The main reason for the improvement was that fewer switches were used and the switches that *were* used had a smaller size in comparison to the corresponding mesh switches.

As driver application they used a Video Object Plane Decoder (VOPD) – as depicted in Figure 2.10 – where only about half of the cores communicated to more than a single core. This motivates the configuration of this custom NoC, having less than half the number of switches than the mesh NoC; also the custom NoC has lower packet latency as the number of switch and link traversals is lower.

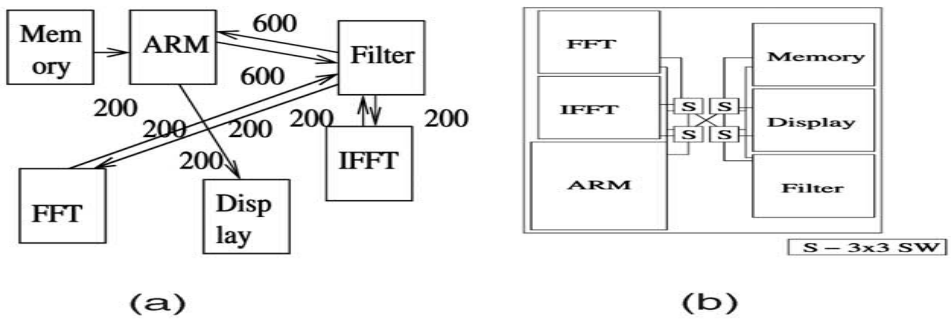


FIGURE 2.10. Custom topology mapped Video Object Plane Decoder of Bertozzi et al.

Mesh and Torus

Most NoCs presented in literature have a physical layout that resembles a mesh or a torus as seen in Figure 2.11 [Salminen2008, Bjerregaard2006, and Micheli2006].

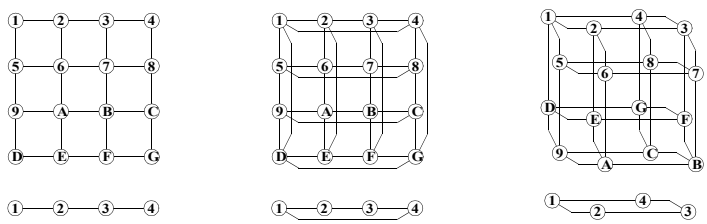


FIGURE 2.11. The Mesh, the Torus, and the Folded Torus

The main reasons for choosing a mesh are:

- Fits a planar chip
- Short and predictable links
- Easy routing – addresses become positions in the grid!
- Scalable
- Intuitively Appealing

Nikolay Kavaldjiev and Gerald Smit did a study in 2003 where they analysed how well different topologies were suited for NoC purposes [Kavaldjiev2003]. Their conclusion was that low dimensional topologies like the mesh and torus are the best candidates thanks to that they scale, have an efficient layout, and are energy-efficient as well.

The 2D mesh and torus networks are the most common topologies and have been implemented by various groups within the NoC community [Bolotin2004b, Felicijan2004, Hu2005, Kumar2002, Liang2000, Mello2005, Millberg2004b, Moraes2004, Sgroi2001, Soteriou2006, and Wolkotte2005] (mesh) [Dally2001, and Marescaux2002] (torus).

If the torus and the mesh are to be compared it has been shown that the torus both has a higher throughput as well as a lower average latency [Yang2008]. On the other hand, Partha Pratim Pande et al. [Pande2005] came to the conclusion that even though the torus may have a higher throughput the mesh is more effective from an energy perspective (lower bit energy) due to the folded torus doubled link length.
William J. Dally and Brian Towles came to a similar conclusion – if the wire transmission power dominates over the power in the routers meshes are more power efficient [Dally2001].

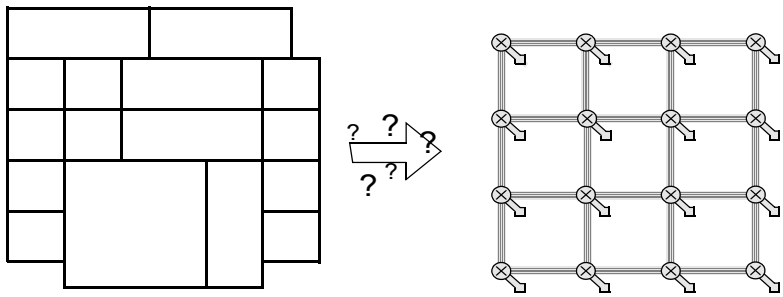


FIGURE 2.12. How to Make an Efficient Mapping?

A Few Words about Mapping onto a Mesh based NoC

An effective process of mapping is essential for the performance of the NoC. From our observations based on different simulated traffic patterns the mapping has a profound impact on performance; in comparison with routing strategy chosen the impact of a reasonable mapping is dominating. With this said I do not claim the routing strategy to be of no/small importance – rather highlight two observations.

- A clever mapping is essential for good performance
- If two NoCs are to be compared e.g. to evaluate, which routing strategy is the most efficient this becomes very hard. This, despite that the two NoCs utilizing the same workload/benchmark the impact of mapping will be so dominant that it makes it very difficult to analyse and make any claims about the benefits of either routing strategy

Mapping is not only restricted to the spatial mapping of cores onto a chip; it does also involve the routing of the communication between these cores as well as mapping of the individual traffic flows into appropriate Guaranteed or Best Effort channels that the platform provides so that requirements on latency, throughput can be fulfilled.

Lap-Fai Leung and Chi-Ying Tsui argue that to satisfy the hard dead-line requirements under worst case conditions, the network resource allocation and scheduling of guaranteed services *have* to be done off-line, i.e. under the static mapping phase to minimize the network resource usage [Leung2006].

When it comes to the process of mapping a detailed study is out of the scope of this thesis but to make any statement here I fully agree with Andreas Hansson et al. in their claim that the process of mapping must be a unified approach in that the different aspects of mapping have to be considered simultaneously [Hansson2005]. In their paper, they present a holistic mapping process that also involves the TDMA slot allocation for the provision of latency and throughput guaranteed services

The idea of a regular mesh (torus) based NoC is intuitively appealing but the more "*SoC-ish"* a system becomes, i.e. the more heterogeneous the cores are the less suitable the mesh seems due to different sizes of the cores. This was correctly pointed out in Radu Marculescu et al.'s survey of *Outstanding Research Problems in NoC Design.* Regarding topology the size and shape of the cores vary widely hence a regular topology will waste area [Marculescu2008]. A CMP/MPSoC system is more easily adapted.

To attack this problem of heterogeneity in size, sacrifices have to be made. If the logical structure of the full mesh does not have to be kept and the switches in the network can make intelligent choices and route packets around missing/removed links a more clever design can be made. A NoC that adopts this scheme is the QNoC of Technion (Israel) depicted in Figure 2.13 [Bolotin2004b].

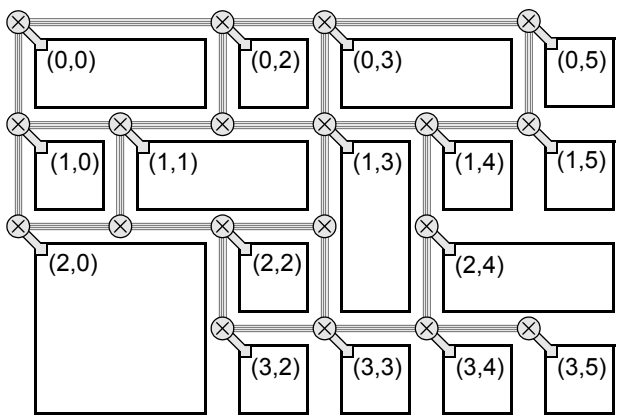


FIGURE 2.13. QNoC Mapping with "missing" links

The potential cost of this approach is obviously that the switches need to be explicitly aware of how to route the packets. In the traditional, complete, mesh, the routing is – for natural reasons – not a matter of concern since the routing is simply based on the position in the grid [Bolotin2005 and Bolotin2007]. For a network where some links are missing routing tables are needed to avoid that packets are routed into "dead ends". These routing tables are potentially costly, since they grow with network size – this regardless if they are located in the routers or in the sources. The QNoC hence employs a hardware-efficient routing technique that is based on a combination of a fixed routing function and reduced routing tables with entries that are created only for destinations whose routing decisions differ from the output of the routing function.

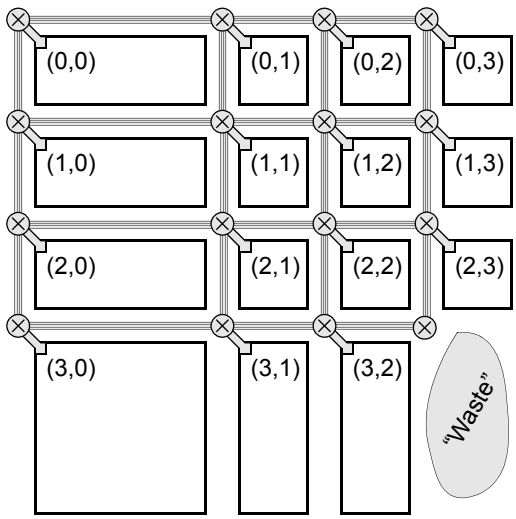


FIGURE 2.14. Regular Mapping with Waste Areas

If the logical structure of the mesh is to be kept with an acceptable waste of silicon area the mapping can be done in such a way that the links no longer have to be of the same length – see Figure 2.14. This approach will keep most of the benefits of the mesh or torus with (maybe) the exception of the predictable characteristics of having links of equal length. This does, however, add one further dimension to the mapping process and the waste area might not be of significant size dependent on (1) how efficiently the resources can be mapped into a grid with given sizes of rows and columns (2) how well the other requirements of the mapping process can be fulfilled.

The Energy Perspective of Meshes vs. Tori

Since meshes and tori are common topologies a small comparison could come handy. In 2005 Hangsheng Wang et al. suggested a framework for the study on the energy consumption of different topologies [Wang2005]. Their "simple" assumption was that the average energy consumed by transporting a single flit/packet – E_{flit} – could be expressed as either (1) or (2). These equations are variants of the one that William J. Dally used to motivate the choice of a Torus [Dally2001]

$$E_{flit} = H_{avg}(E_R + E_L) \quad (1)$$

Where H_{avg} is the average number of switches/routing elements a flit has to traverse in the system; E_R the average routing traversal energy and E_L is the average link traversal energy per channel, i.e. the cost of transferring a flit between two switches. Moreover, D_{avg} is the average distance from source to destination and E_{L0} is the average link traversal energy per unit length and determined by signalling technique and process technology.

$$E_{flit} = H_{avg} \cdot E_R + D_{avg} \cdot E_{L0} \quad (2)$$

In their study, they evaluated meshes and tori of different dimensions as well as hypercubes, hierarchical meshes (aka hypercubes). In hierarchical meshes and tori, channels not only connect adjacent nodes but also connect v-node away neighbours in each dimension. These connections are called *express channels* and the distance v is referred to as the *express interval*.

Their findings could be summarized in that

- From an energy standpoint, high dimensional tori should never be selected over hierarchical tori or express cubes.
- As the process geometry shrinks the express interval of the hierarchical tori should be increased to keep the hierarchical tori the most energy-efficient.

In both conclusions above the assumption is made that the average hop distance H_{avg} scales with the size of the network, i.e. that the application running on the system is fully parallelisable to present an even load to the network. From a SoC perspective, this is however doubtful. Despite this objection, I have chosen to use the H_{avg} as defined by Hangsheng Wang et al. and by William J. Dally.

If the n×n 2D mesh is compared with the folded 2D torus equation (2) gives

Mesh with H_{avg}=2n/3

$$E_{flit}^{M} = \frac{2n}{3}(E_R + E_L) = \frac{2n}{3}E_R + \frac{2n}{3}E_L \qquad (3)$$

Torus with H_{avg} = n/2 and the channel length doubled

$$E_{flit}^{T} = \frac{n}{2}(E_R + 2E_L) = \frac{n}{2}E_R + nE_L \qquad (4)$$

As it can be seen the torus consumes *25 percent less* routing energy, E_R, but *50 percent more* link energy, E_L, in comparison to an equally sized mesh. This means that the ratio between link and routing energy determines which alternative that is the most energy-efficient. However – this is under the assumption that the traffic pattern is uniform and this is most likely not the case for an average SoC. If locality in the traffic pattern is assumed – and it should be if a low order topology is to be chosen – then the mesh becomes more advantageous since the H_{avg} no longer scales with the size of the network. Davide Bertozzi et al. came to a similar conclusion in their work but with "realistic" traffic patterns [Bertozzi2005]. They applied their design methodology `NetChip` onto four different video processing applications: a Video Object Plane Decoder (VOPD) mapped onto 12 cores, an MPEG4 decoder mapped onto 14 cores, a Picture-In-Picture (PIP) application mapped onto eight cores (PIP), and a Multi-Window Display (MWD) application mapped onto 14 cores. The traffic patterns of these applications are rather local – as seen in Figure 2.15 – and hence the mesh outperforms the torus in terms of both area and power for all four cases despite that the average hop count is about 10 percent smaller for the torus.

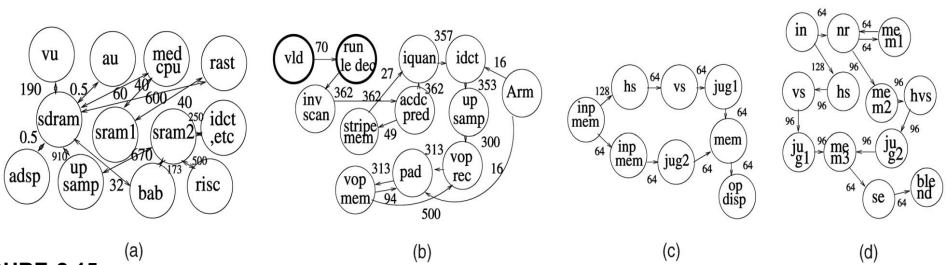


FIGURE 2.15.
Core graphs of video processing applications. (a) MPEG4 core graph, (b) VOPD core graph, (c) PIP core graph, and (d) MWD core graph.

Bottom Line

In the collected literature on Network on Chips, there is no "obvious" winner when it comes to the choice of topology [Marculescu2008]. If the commonly available NoC proposals are investigated at least some have an articulated reason behind their choice of topology.

Despite that higher order networks may have theoretical advantages over the low order networks they are ruled out due to the layout constraints that exist on a planar space such as the chip.

Fortunately, the communication pattern of a traditional SoC exhibit locality [Bertozzi2005] and hence low order topologies could be advantageously used [Culler1999]. In particular accesses to memory displays a high degree of both temporal and spatial locality of data accesses; this in combination with that since SoCs traditionally heavily rely on systems communicating via memory the low dimensional topology NoC is a natural choice [Kavaldjiev2003].

Despite what is said about how appropriate different topologies are in terms of the architecture's ability to scale and of layout and energy-efficiency I would say that the two most important factors when making the choice of topology still is how well it matches the traffic pattern of the system in combination with the possibility to efficiently map the system in question to this particular topology.

2.4 PACKETISATION, ADMISSION, ROUTING AND EXIT

Since the purpose of this chapter is to put the papers and concept presented in my publications into a context it will only be a brief introduction to network characteristics and in no sense a complete description. The reader is encouraged to study (at least) the introduction chapters of the book *Principles and Practices of Interconnection Networks* by William J. Dally and Brian Towles and/or the *Interconnection Networks: An Engineering Approach* by Jose Duato et al. for a good overview and a thorough treatment of the topic [Dally2003 and Duato1997]. In *Networks on Chip,* Giovanni De Micheli and Luca Benini treat the concept of Networks with the focus set on Network on Chips [Micheli2006]. These three books will greatly improve the reading experience and understanding of this chapter.

During the transmission of messages from a sender to a receiver over a NoC several stages are involved as shown in Figure 2.16

- Packetisation/Segmentation – the messages are split into packets
- Ingress/Downstream Queuing, Arbitration and Network Admission
- Network Transportation – routing through the network
- Network Exit and Egress/Upstream queuing
- Depacketisation/Desegmentation

Even though all of these stages are essential, the network transportation is the part that has been given the most attention. Innumerous papers have been devoted to how to efficiently transport data over the network in terms of Topology selection, Routing algorithms, Router micro-architectures, Arbitration, Throughput guarantees, Power and energy issues, Fault tolerance and reliability issues, Timing and so forth.

Most of these aspects are analysed from the perspective of data transportation with or without guarantees. Very little is, however, written about how to guarantee network *Admission* and *Exit* for Best Effort traffic efficiently.

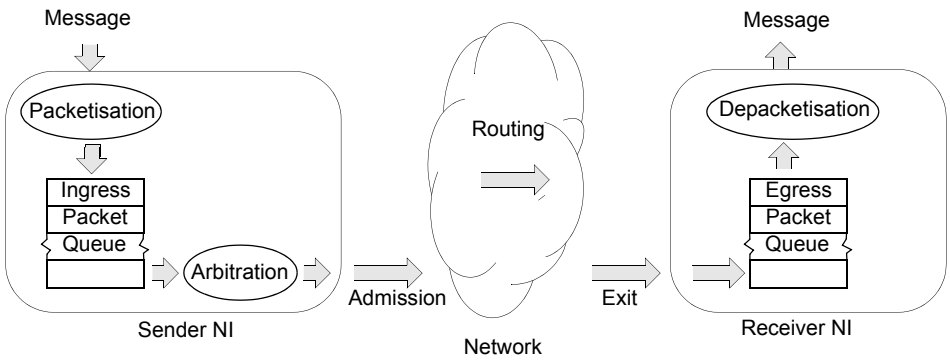


FIGURE 2.16. Packetisation, Admission, Routing and Exit

Even though the stages are the same the situation is somewhat different for traffic allocated to Best Effort versus traffic that is assigned to a service class with some given guarantees (for a more elaborative description of these concepts see Section 2.5, "Quality of Service – QoS," on page 70 ff.). For traffic with "hard" guarantees the messages are treated as first class citizens and hence will be winners in the arbitration process in the Network Interfaces as well as in the network.

For the transportation through the network different schemes can be used such as Virtual Circuits [Mello2005], TDMA based schemes [Goossens2005 and Millberg2004a], etc. One potential problem is that if the guarantees given, only apply to the transportation – but *not* the access to the recipient (Master – Slave data rate mismatch). In these cases end-to-end flow control might be needed.

For the Best Effort traffic, the situation is very much dependent on the traffic situation in the rest of the network, and it is very hard to achieve "global" fairness in the system.

By *global fairness,* I mean that all packets in the system, that is, *in the network and in all Network Interfaces (NI)* of the system are competing on the same premises. The reason is that the arbitration is local to the network interfaces; hence packets that reside in different Network Interfaces are not competing on the same premises. As a consequence all decisions inside the respective network interfaces only (at best) guarantee *local fairness*. Neither are the packets in the NIs competing on the same conditions with the packets inside the network. In our publication from 2009 we introduced the concept of *Priory Based Forced Requeue* to make the fairness more global [Millberg2009]. The idea is to make the packets inside the NI queues compete with packets in the switch attached to that Network Interface. By doing this the global worst case latencies are significantly reduced.

Another problem that may arise for Best Effort traffic is that the packets inside the network may have difficulties to get out, due to contention at the exit node. The contention not only affects the packet that currently wants out but may cause congestion and the formation of a saturation tree. The simple but effective solution that we introduced was the *Dual Packet Exit* concept which basically means that the exit bandwidth is doubled, which results in a performance boost of the system [Millberg2007a and Millberg2007b].

Except for our publications, I have not been able to find any relevant publication attacking the problem of admission and exit from the network for Best Effort traffic.

2.4.1 Packetisation / Segmentation

During the *Packetisation stage,* the message is segmented into packets and routing information is appended to the packet. Depending on routing strategy the scenario is a little different. In the case of a virtual cut-through or wormhole switching strategy the packets consist of a *header*, *payload* and a *tail*; these parts are further divided into *flit* as seen in Figure 2.17 and in Figure 2.18. In the case of a deflective routing strategy, the flits are the same as packets and all packets have their own header – Figure 2.19.

The routing information added in the packetisation stage is further dependent on whether *source routing* or *distributed routing* is employed. For the source routing the explicit routing path is appended to the packet and for the distributed routing only the destination address is needed – for more information see Section 2.4.3, “Routing,” on page 59.

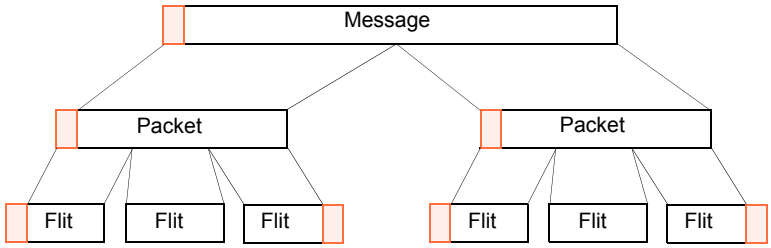


FIGURE 2.17. Wormhole switching – Message Split into 2 Packets Split into 3 Flits

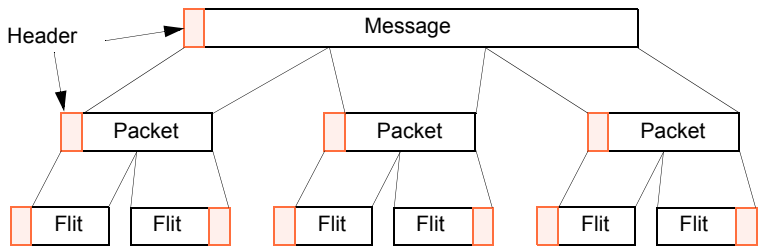


FIGURE 2.18. Wormhole switching – Message Split into 3 Packets Split into 2 Flits

Wormhole Switching

In 2004 Aline Vieira de Mello et al. did a survey on switching algorithms on mesh based NoCs where they, among other things, analysed the trade-off between packet size and the average time to deliver these packets.

In their study, they used a 5×5 NoC with an XY routing algorithm where they varied the number of flits that contained one packet (10, 100, 1000 and 10000 flits respectively). The average time it took to deliver one flit was 13.2, 4.6, 79.4 and 105.47 clock cycles, respectively. The conclusion they draw from this was that *"Small packets have the advantage to induce a small number of blocked paths, with a corresponding reduction in packet delivery times. Larger packets present the opposite behaviour. However, small packets impose larger overheads for segmentation and reassembly in the IP core wrappers. Also, arbitration/routing is executed more frequently in this case. As a conclusion, medium size packets (up to 500 flits) represent a good trade-off between packet size and the time required to deliver these packets with the XY routing algorithm." [Mello2004].*

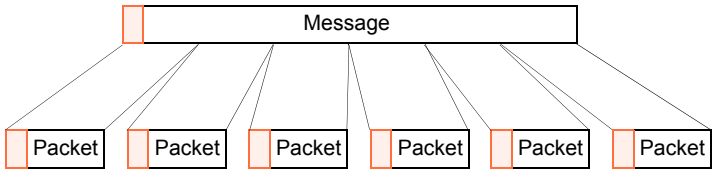


FIGURE 2.19. Deflective Routing – Message Split into 6 Packets

Deflective Routing

A deflective routing algorithm will, naturally, be more effective the smaller a message is, that is, the less packets that are to be assembled at the recipient. This, since the deflective routing strategy is non-minimal and hence, most likely, introduces a packet reordering. Due to this drawback of packet reordering all packets have to be equipped with sequence numbers to enable the message assembly at the recipient. For this reason, a deflective routing strategy will strongly benefit from having a small worst case latency. We have attacked this problem both at the sender side with an admission policy based on the *Priority Based Forced Requeue*-concept as well at the receiver side with the *Dual Packet Exit*-strategy [Millberg2009 and Millberg2007b]. Despite our efforts, the deflective routing will most likely have to be accompanied with some sort of data transport facility for long messages, e.g. cache transfers as well as for streaming media like multimedia application. Our proposed solution to this problem is based on *looped containers* that utilise a TDM based scheme and hence can coexist with the Deflective Routing transport scheme that is primarily intended for short and instant message delivery with Best Effort characteristics [Millberg2004a].

2.4.2 Ingress and Egress queuing

The queues (or buffers) in the network interfaces have two purposes: (1) they absorb the differences in speed and burstiness between the resource and the NoC; (2) they hide the network internals such as the packetisation, arbitration, and end-to-end flow control from the application. The size of the buffers must, of course, be sufficiently large but not too large since buffers are a major contributor to the energy consumption and area in the Network Interfaces [Coenen2006]. The sizing of these buffers could be based upon extensive simulation or with analytical methods [Hansson2008].

2.4.3 Routing

Routing is the process of selecting paths in a network from a source to destination along which to send network traffic. The route can be predetermined – *static routing* – or a function of the network traffic – *dynamic routing*. If only the shortest possible paths are allowed the routing is said to be *minimal* otherwise *non-minimal*. If the routing decision is made in the sender node of the packet it is referred to as *source routing* alternatively the decision is taken locally in the switches along the way of the packets, and hence it is called *distributed routing.* Regardless whether the routing is predetermined, minimal or where the switching decision is taken the process of routing should be both free of *livelocks* as well as *deadlocks*. Deadlock is a circular dependency between packets that prohibit further progress, i.e. the packets are waiting for each other in a cycle. Livelock means that packets proceeds indefinitely, but never arrive – this is only possible for an adaptive non-minimal routing.

Given the characteristics above several different *communication strategies* are possible. The most common communication strategy in the realm of Network in Chips is the *Wormhole switching* strategy followed by Circuit switching and Deflective Routing. It should also be mentioned that *wormhole switching* is referred to as *wormhole routing* in the literature as well.

Static vs. Dynamic / Deterministic vs. Adaptive

The properties of a Network on Chip will very much depend on the routing scheme it uses. Routing falls into the two categories *static* or *dynamic*. Static routing must take the worst case scenario into account, whereas the dynamic routing can get away with dimensioning for the average traffic volume [Goossens2002].

Depending how a path is defined, routing can be classified as *deterministic* or *adaptive*. In deterministic routing, the path is completely specified from the relative position of source and target addresses. If the decision of the path is made without any information about the current status of the network it is called *oblivious*. In adaptive routing on the other hand, the path is a function of the network traffic. Routing in irregular topologies is typically accomplished using routing tables. These tables can be located in the routers or in the sources.

Global routing, that uses static routing, has the advantage of being able to avoid both contention – and hence congestion. Contention means that two packets want to utilise the same link/router at the same time – this may result in congestion. Congestion is that the network "clogs" up which leads to that other packets in the system gets affected and results in a degraded performance of the system. Traditionally global system routing means that connections are set up statically – circuit switching. Dynamic routing is more flexible and is implemented with packets.

Jingcao Hu and Radu Marculescu argue that the most appropriate routing techniques for NoC should be static since a NoC would only be used on a small class of applications and hence a designer will have a good understanding of the traffic characteristics and can use that information to avoid congestion by wisely mapping the IP cores and routing paths. Therefore, a static routing technique is the obvious choice [Hu2004]. Further they claim that the implementation of a dynamic scheme requires *far more resources* and that the packet reordering will require *huge* buffering space. Unfortunately, they back up their claims with neither numbers nor references [Hu2003].

Luca Benini and Giovanni De Micheli take the opposite standpoint and claim that favour adaptive routing is the best choice for special-purpose SoCs. The reasons for this point of view are that future on-chip micro-network designs will emphasize speed and decentralisation of routing decisions. Furthermore, properties such as robustness and fault tolerance will also be highly desirable. These factors, and the observation that traffic patterns for special-purpose SoCs tend to be irregular, seem to favour adaptive routing. But, as a saving clause they also state that when traffic predictability is high and nondeterminism is undesirable, deterministic routing *may* be the best choice [Benini2002].

Aline V. de Mellos et al.'s publication from 2004 makes a comparative study on a deterministic routing scheme vs. a set of three adaptive routing schemes. The packet switching technique was wormhole switching based (see *Wormhole switching* on page 66) and the topology was a mesh. Their results indicate that, the *total time* to deliver *all* packets was in favour of the deterministic routing. However, the partially adaptive algorithms can potentially speed up the time to deliver *individual packets*, but from a global point the deterministic routing was superior. The reason behind the shortcoming is the tendency of the adaptive protocol to concentrate the traffic in the centre of the network, increasing the number of blocked paths.

Worth mentioning is that the size of the network was comparably small (5×5) [Mello2004]. One way to circumvent the problem of the packet concentration in the centre of the network is to employ a scheme where the routers are communicating their current workload to their neighbours and hence enables better routing decisions like our *Proximity Awareness* concept [Nilsson2003].

In Jingcau Hu's thesis, he moves the decision of using a dynamic or static protocol into the router [Hu2005]. The concept he calls DyAD (**Dy**namically switching between **A**daptive and **D**eterministic modes) and means that each router in the network continuously monitors its local network load to dynamically decide which protocol to use; if the network is uncongested a low latency deterministic mode is chosen. For the congested network, the DyAD router switches to the adaptive routing mode and thus avoids the congested links by exploiting other routing paths to ensure a higher network throughput. The area overhead reported is a modest 7 percent increase.

Source Routing vs. Distributed Routing

Source routing means that the entire path is decided before the packet is sent and the routing information is appended to the packet header. This has the advantage of using very simple routers, since they can be stateless and identical and hence do not require any configuration [Radulescu2005]. The drawback is that the path cannot be changed after the packet has been sent plus that the routing information will be present in the packet headers and therefore, makes the packet headers larger.

Distributed Routing means that the router has to take the decision how the packet should be routed. The potential benefit is that this decision can be dynamically changed dependent on the status of the network. The drawback is that the router either (1) has to have routing tables or (2) implement a more or less complex decision logic.

Deadlock and Livelock

One potential problem with the wormhole switching protocol – or rather – any adaptive protocol, is the possible risk of a *livelock* or *deadlock*. *Deadlock* means that two, or more, packets are having a cyclic dependency so that no forward progress can be made. *Livelock* means that one or more packets are circulating the network without ever making any progress towards their destination.

The potential for a locking situation, can, however, be avoided if the right measures are taken. In 1994 Christopher J. Glass and Lionel M. Ni presented the *Turn Model* which was proven to be both live and deadlock free [Glass1992]. The "trick" they used was that some paths/routing decisions were declared illegal and hence the routing algorithm went from *fully adaptive* to *partially adaptive*. Variants of this have later been implemented in several NoCs e.g. the *odd-even routing algorithm* [Hu2003].

Another way to solve the deadlock problem is to use virtual channels [Duato1997]. In this approach, one physical channel is split into several Virtual Channels (VCs). By making the channel dependency graph acyclic via routing restrictions deadlock can be avoided. By using VCs high performance can be achieved. However, this scheme requires larger buffer space for the waiting queue of each VC and hence can make it costly.

However, the solutions above does not fully solve the problem; the deadlock identified above only means that there does not exist cycles *in the dependency graphs of the network*. Even though the dependent graphs of a NoC and an IP utilising it may be cycle free in isolation the system as a whole is not guaranteed to be deadlock free when put together as Andreas Hansson et al. point out [Hansson2007b]. They tackle this problem by making a holistic analysis of message dependences to detect these cycles.

Switching

For network based communication there exist three commonly used conceptually different main approaches: *Circuit switching*, *packet switching* (or *store-and-forward*), and *virtual cut-through switching*. For completeness both the store-and-forward and virtual cut-through strategies are presented even though they have been shown to be inappropriate for Networks on Chip. The most common switching strategy for *on chip* communication is *Wormhole switching* that is a variant of virtual cut-through switching. In general the choice of a source-destination path can be separated from communication mechanism.

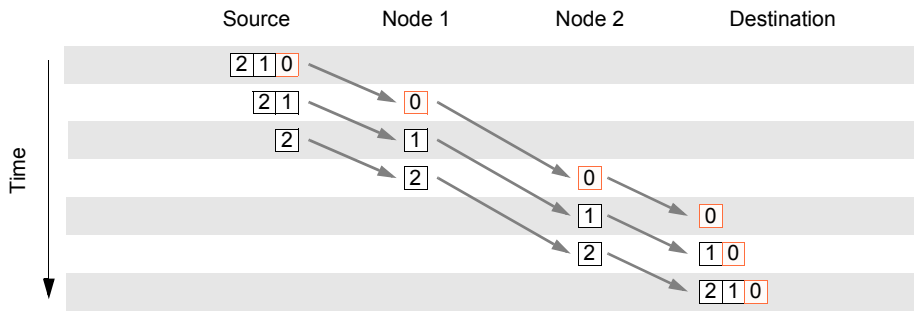


FIGURE 2.20. Circuit Switching

Circuit-switching

Circuit-switching implies that a path from the source to the destination is initially established. The circuit will be kept alive, at least, until the entire message is transmitted. Circuit switching can use *physical circuits*, where each physical link is reserved for the duration of the message, or *virtual circuits*, where only virtual links are reserved. In pure virtual circuit switching the message will only be sent when a circuit acknowledgment has been received by the sender. Circuit-switching is most effective when messages are long. Minimum message latency is proportional to the sum of the message length and some constant multiple of the path length plus the time it takes to set up the circuit. In the world of NoCs this method is quite uncommon and represented by e.g. the containers of Nostrum [Millberg2004a] or the circuit switched network of Pascal T. Wolkotte et al. In their work, they explicitly target streaming applications where they statically set up the circuits [Wolkotte2005].

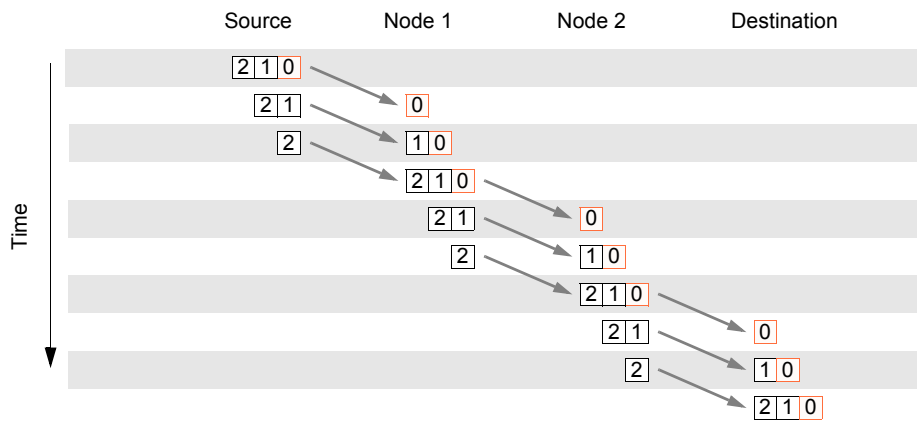


FIGURE 2.21. Store-and-Forward or Packet-Switching

Store-and-Forward or Packet Switching

Store-and-forward means that a message is segmented and sent as packets over the network. As the name suggests all the packets belonging to the same message are stored in the switch before they are forwarded. Packet-switched methods are generally advantageous when messages are short and/or infrequent [Gaughan1993]. The potential benefit is that the switch can make content aware decisions. The drawback – that rules out multi-packet message store-and-forward to be used in Networks on Chip is that it introduces a minimal delay that is proportional to the product of the number of switches the packets have to traverse and the message length; also it requires an unacceptable amount of buffers in the switches since capacity must be reserved for storing multiple complete messages [Tota2006]. For a network *off chip* this strategy is viable since buffer space is relatively cheap but wires are expensive and hence packets/messages need to be sent in a serialised fashion. In the world of NoC the packet switching technique is represented by e.g. “pure” store-and-forward and deflective routing – see below.

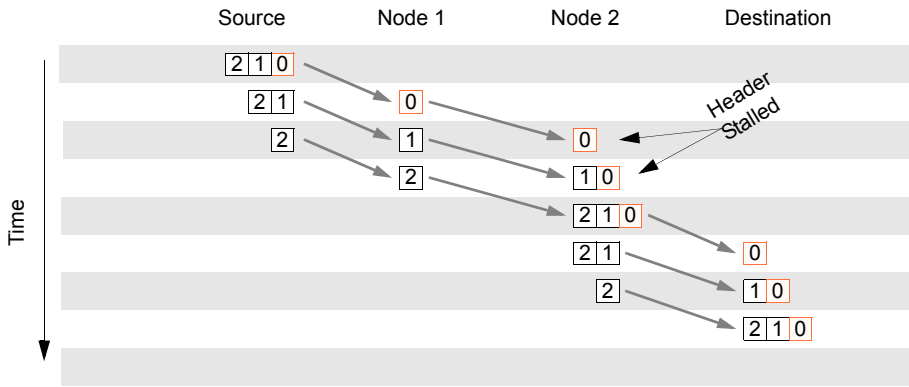


FIGURE 2.22. Virtual Cut-Through

Virtual Cut-through

The virtual cut-through could be seen as a variant of the store-and-forward approach [Kermani1979] or as a combination of packet switching and circuit switching. The message is broken into small pieces called **fl**ow control digi**ts** – or flits – that are *pipelined* through the network. The requirement of having the complete message being stored before it can be forwarded has been relaxed with the option that it is legal to start delivering packets to the sequent router if a route exists and buffer space is available. If either of these conditions is unfulfilled the virtual cut-through is reduced to store-and-forward with its inherent weaknesses; this since only the header contains routing information and therefore, each incoming data flit is simply forwarded along the same output channel as its predecessor. Thus, transmission of different packets cannot be interleaved or multiplexed over one physical channel. In addition, this also has the implication that *all* routers must have the capacity to buffer the entire message since a router containing a stalled header flit have no means of stopping subsequent incoming body flits.

At heavy loads, virtual cut-through routing tends to behave like packet-switching protocols; at light loads, it behaves more like circuit-switching protocol [Gaughan1993]. Despite its known weaknesses, it has been used for NoC purposes – even though just for evaluation by e.g. Jingcao Hu and Radu Marculescu [Hu2004]

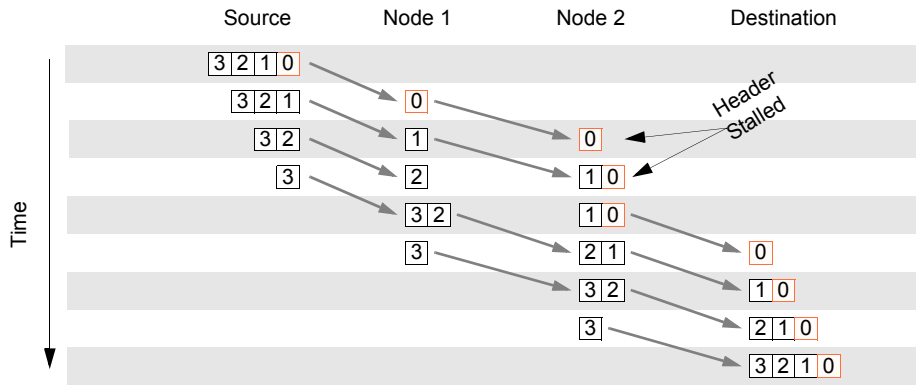


FIGURE 2.23. Wormhole switching – Buffer size of 2

Wormhole switching

Wormhole switching was developed by William J. Dally and Charles L. Seitz and is the evolution of *the virtual cut-through strategy* [Dally1986]. Originally, it was designed to be used in parallel computer clusters because of its comparable small delay and reduces the buffer requirement in contrast to virtual cut-through. Minimum latency is proportional to the sum of the message length and the path length.

Each packet/message is divided into flits (**fl**ow control digi**ts**); the header flit sets up the routing path at each router. The buffers at the intermediate nodes are the size of a message header. The body flits will when follow the path set up by the head flit; the tail flit will finally release the path set up. Hence the name wormhole switching since all flits will follow the same path in a consecutive way. The wormhole switching could be seen as a variant of cut-through routing since flits can be forwarded by the switch, as soon as they are available but this is under the condition that there exist a free link to the next switch *and* buffer space in that very switch. If no output link is disposable, flits are buffered at intermediate nodes. If the buffers are large enough to hold the entire message, the message is buffered at the blocked intermediate node and no links are held. If the buffers are *not* large enough, the message will be buffered across several intermediate nodes. This has the negative consequence that a worm/flit that does not fulfil either of these last two conditions will get stalled and lock up network resources (buffers).

In the presence of *hot modules*, the hop-by-hop backpressure mechanism of wormhole switching cause buffers to be filled up and become stalled blocking new arrival to this router. This creates a domino effect that spreads over the network creating a saturation tree [Pfister1985]. Hot modules in this context are modules that are bandwidth limited and in high demand of other modules in the network. Isask'har Walter et al. have recognized this problem and suggest a solution based of credit based distributed access regulation scheme [Walter2007]. Another solution suggested by Jose Duato et al. involves dynamically allocated separate buffers for the congested flows [Duato2005]. Srinivasan Murali and Giovanni De Micheli acknowledge the problem with the statement that the use of a wormhole flow control results in a non-linear increase in latency (due to blocking of paths in case of contention, creating a domino-effect) with decreasing link bandwidth [Murali2004]. As Jennifer Rexford and Kang G. Shin put it "... *a blocked wormhole stalls in the network, effectively dilating its length until its outgoing channel becomes available. As a result wormhole network typically utilise only a fraction of the available network bandwidth" [Rexford1994]*. As a consequence packet based routing outperforms wormhole switching at higher network loads at a cost of increased packet delays and buffers. Still wormhole switching is well suited for Best Effort switching due to its low latency and small buffer requirements.

The wormhole switching scheme is by far the most commonly used in NoC design as seen in Erno Salminen's survey of 2008 [Salminen2008]. The performance of the wormhole switching is, however, heavily affected by the number of available buffers [Leung2006]. But buffers are expensive from an energy perspective and cannot be increased without significant cost [Ye2004]. In the HERMES switch paper, Fernando G. Moraes et al. report that in a switch utilising a flit-size of 32 the buffers occupy 96 percent of the total switch area [Moraes2004]. This in combination means that a switch utilising wormhole switching is a relatively expensive in terms of area – and hence energy – if good performance is desired.

Deflective Routing (or Hot potato)

Deflective routing is an adaptive routing protocol where the route of individual packets is not only restricted to the shortest path. That is – if a switch is unable to route a packet in the most preferable direction a switching decision that routes a packet in any other direction is legal [Feige1992]. This has the implication that the switch *does not have to have* explicit buffers for packets that currently cannot be forwarded. There is, however, no restriction that a deflective routing scheme automatically means that the switches cannot have buffers. In the case that the switches lack explicit buffering the deflective routing can be called *Hot potato routing* since the packets now have to be treated as potatoes too hot to hold. The two obvious benefits of this scheme are that (1) the switches can be made very small since no explicit buffers have to be implemented; (2) the routing scheme becomes robust since packets potentially can be routed around hot-spots and congestions in the network.

As with anything the deflective routing scheme has its drawbacks as well – the most commonly articulated in literature is (1) the potential cost of reordering since packets belonging to the same message eventually have to be reordered; (2) Deflection means that the path length is no longer minimal. On the other hand, Smaragda Konstantinidou and Lawrence Snyder showed that these misroutes have a small impact on the overall performance [Konstantinidou1994]. The deflective routing scheme they used in their *Chaos router* employed virtual cut-through routing with a randomised deflection scheme (3) Also there exist a potential risk of livelock, this is, however, not an issue if the packets are equipped with e.g. hop-counters to guide the routing process. In a wormhole switching scheme the livelock situation is normally handled by restricting the routing decisions to minimal paths only – clearly this is not an option for deflective routing.

To further avoid hot-spots and congested areas of the network we suggested a scheme for the switches to broad cast their current load to their neighbours – we called the concept Proximity awareness [Nilsson2003]. By averaging the load over time for neighbouring switches, we managed to cut the number of packets caught in local “traffic jams”. Later Jerry Tao Ye et al. got inspired by this scheme and made variant for wormhole switching – they did, however, choose to call it Contention-look-ahead routing [Ye2004]. In their paper from 2004 they report saving in terms of both buffers, network latency as well as in total execution times.

One interesting, recent, variant of the deflection routing scheme is the bufferless routing concept BLESS of Thomas Moscibroda and Onur Mutlu. They present two variants of their routing scheme – one, which is the traditional single flit based and one, which is a mix of a deflective and wormhole switching [Moscibroda2009]. Their main motives behind their approach are that the buffers consume significant energy and the increased complexity of the router. Among other findings, they report energy savings up to 40 percent compared with other existing routing algorithm utilising buffers. Within their study, the incorporated an energy model capturing the energy consumption of additional hardware required by BLESS. They paid special attention to accurately model the energy consumed by the extra buffers needed on the receiver side. In addition they also incorporated the increased link width to transmit header information together with the logic to reorder flits of individual packets in the receiver. To make a comparative study they divided of the energy consumption into network energy, buffer energy, router energy and link energy; the energy model was capturing the dynamic as well as static components. The buffer energy included, both, the input buffers of the routers and the receiver-side buffers needed to reorder packets for in-order delivery. The router energy included, both, routing and arbitration energy components. In the case of buffer energy, dynamic buffer energy is consumed whenever a flit is written to or read from a buffer.

Regarding the reordering problem at the receiver side they start out by making the observation that this likely increases the number of flits/packets that need to be buffered. In addition, in-order delivery of packets requires buffering of packets that arrive out-of-order, both in bufferless and buffered routing. They finish their treatment of the topic with the statement: "*The increased receiver-side buffering requirements of BLESS and the additional logic to reorder flits reduces the energy reductions obtained by eliminating input buffers in routers. However, our evaluations show that the energy reduction due to eliminated router buffers outweighs the energy increase due to increased receiver-side buffers.*" *[Moscibroda2009].*

2.5 QUALITY OF SERVICE – QOS

Quality of Service (QoS) is often used synonymously with *Guaranteed Services* i.e. services with strictly defined properties in comparison to the *Best Effort* services with only statistical properties; this even though the Quality in QoS – in my humble opinion – may refer to statistical properties as well. Without risking a too serious clash with the terminology used by the NoC community, I'd like to use the term *Quality* in the same way that its synonyms *Inherent feature* or *property* is used. Quality of Service is hence just a recognition that *a certain service will possess some properties*. Further I'd like to define the *Guaranteed Services* as services that have *known and definite values/properties with 100 percent certainty within a given time interval*. Best Effort services are consequently, the services that have *known and definite value/property with less than 100 percent certainty within a given time interval*. Given this rather loose definition it is possible to make the transition between the Guaranteed and Best Effort by changing the time interval.

With this said I'd like to leave this semantic hair-splitting and simply state that the concept of Quality of Service is of essential importance if a network on chip is going to work as an interconnection platform that inherently supports composability.

Kees Goossens et al. formulated a set of good reasons why guaranteed services are beneficial/necessary for the building of a composable system [Goossens2002]. The first three reasons concern the programmability of the IPs, whereas the two latter deals with the composability. (1) Some IPs have strict requirements when it comes to data throughput and/or timing. (2) Guaranteed services make the dependencies of the interconnect explicit, which eases the structuring, design and programming. (3) The IPs can be made simpler since no negotiation with the interconnect is needed – either the service request is granted or not! (4) Services granted to an IP are unaffected by the traffic from other resources in the network. (5) The implementation cost of the Network becomes known early in the design process since accurate models of the traffic behaviour of the IP are required to give the aforementioned guarantees. Hence a good estimate in terms of resources required by the interconnect can be given. Edwin Rijpkema et al. also argue that a flexible and efficient solution to the communication problem requires at least two service classes by the network; the *guaranteed throughput* and the *Best Effort* [Rijpkema2001].

The problem with guaranteed services is that they are costly to realize; either an extremely detailed knowledge is required about the system and its traffic behaviour so that the traffic can be orchestrated in such a way that given guarantees really *can* be guaranteed. The problem with this approach is that systems of today are far from predictable and hence very hard to orchestrate in any reasonable way. The alternative to detailed orchestration is that the interconnect system has to be grossly over dimensioned to give guarantees on latency and throughput. Any possible scenario must be covered for in terms of hardware. One noteworthy observation made by Stefano Santi et al. is that high performance, state of the art, computing interconnects lack support for QoS; this despite that this kind of architectures, often, require high throughput as well as low latency [Santi2005]. The reason behind this design decision is that the network is over dimensioned in such a way that network congestions will not occur. For Network on Chips this kind of luxury is, however, not affordable due to limited silicon area as well as a constrained power budget.

To ease these requirements on the interconnect two major approaches can be taken: Traffic scenarios/Use-cases and diversification/classification of the traffic streams. The use of Use-cases (or Traffic scenarios) means that early in the design process the different subset of possible traffic streams originating from various applications in the system that can coexist in the system at a given point in time are identified. By grouping the traffic streams into these subsets an estimate of the network resources needed can be given. This reduces the complexity of the problem of orchestrating the traffic at the expense of reducing the freedom in which way the system can be used. The use of use-cases may make it easier to guarantee QoS *within* the use-case. The transitions between different use-cases are, however, problematic. As Andreas Hansson et al. point out – all running applications are disrupted during use-case transitions, even those continuing operation. One solution proposed by the authors is to use a *partial re-configuration* to guarantee an undisrupted QoS for applications that have a life time that spans over one or more use-case transition [Hansson2007a].

The diversification, or classification, of the traffic streams aims to identify the different needs and requirements that the individual streams may have. Some streams, e.g. live video may have requirements on throughput and others may have it in latency.

Since a high (useful) utilisation of the network is desired, any over dimensioning of the network should be avoided. Utilisation at this degree comes at a price. If the network starts to saturate the behaviour of traffic, which is packet based and not circuit based will start to show variations in latency and throughput hence it is very important to identify the requirements of the individual streams that *really* need guaranteed services. Any assignment of a stream that could do with a Best Effort like service to a guaranteed service means that the network potentially has to be over dimensioned.

The problem, with combining best effort service with guarantee services has been recognised by e.g. Jennifer Rexford and Kang G. Shin as inherently hard [Rexford1994]. The reason put forward is that the low-level policies of the routing will highly influence how the traffic classes interact.

Luca Benini et al. even suggest that the concept of *Quality* not only should include performance but reliability as well since Systems on chip will be increasingly involved in different control applications [Benini2001].

2.5.1 Service Characteristics

When discussing Quality of Service or Guaranteed Services it is important to articulate: what is it that is guarantee? That is – what characteristics could be said to be part of a guaranteed service? From literature *latency* and *throughput* are the most obvious and common but there are others. If we superset over what several research groups have suggested the following characteristics crystallises [Rijpkema2003, Vellanki2004, Bolotin2004a, and Goossens2005]:

Data integrity

Data is delivered uncorrupted.

Lossless Data Delivery

All data is delivered by the interconnect.

For the GT services, Lossless Data Delivery is a quite obvious property but Edwin Rijpkema et al. also claim that this is a desired property for the BE service as well. The reason is that since the BE service normally has no time to go through a connection set-up phase – before the real data is transmitted – lost data cannot be detected at the receiver side and hence the connection should be lossless [Rijpkema2001].

In-Order Data Delivery

This means that the data is delivered in the same order that they were sent. A service can have the property of in-order data delivery for the *individual* data that belong to the same message but at the same time provide out-of-order data delivery for different messages e.g. AXI [AXI].

Latency

By latency, we mean how long it takes for a packet from the time it is sent to the time it is received. However, latency could further be divided into system and network latency. By *system latency,* we mean the total time from a sender to a recipient. With network latency, we mean the time the packet spends in the network – NI queuing time excluded. If we have a guaranteed latency then that is an *upper* bound on the worst case latency. The average latency is the average latency for a set of packets over a – fixed – time interval. Some situations will, however, require a fixed latency; one example is reconfiguration of a part of the chip. During reconfiguration a bit-stream may be read from an internal source, e.g. an off-chip flash memory to be used in the process of updating the re-configurable part of the chip [Goossens2008].

Jitter Delay

Variation in latency; unregulated platforms usually have a delay pattern as depicted in Figure 2.24 where a majority of the packets has roughly the same delay with some variation and some packets suffer from an "arbitrarily" long delay – the heavy tail of the distribution.

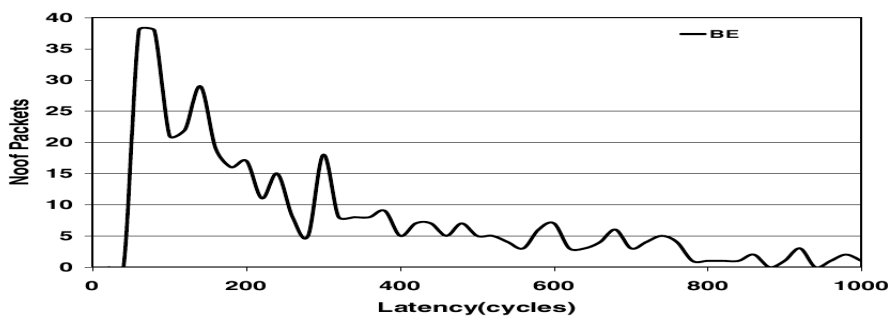


FIGURE 2.24. "Typical" Best Effort Traffic Latency Distribution with Heavy Tail

Figure 2.24 is taken from Praveen Vellanki et al.'s paper and plots the variation in packet latency for destinations that are uniformly 3 hops away in a 4×4 mesh based NoC architecture at an injection rate of 0.05 packets/cycle/node [Vellanki2004]. As it can be seen the worst latency deviates rather much from the average latency. From my experience, this type of heavy tail latency distribution is quite "typical" for Best Effort traffic and is unacceptable for many NoC implementations.

Throughput (or bandwidth)

Throughput is the amount of data that is transferred from a sender – receiver pair during a finite time interval. The system throughput is the amount of data that is transferred from a set of sender – receiver pairs (possibly all). If we have a guaranteed throughput then that is a *lower* bound on the latency over a *finite* time period.

Operational Efficiency

In our publication from 2*007 – Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy,* we defined the term *Operational Efficienc*y [Millberg2007b]. As stated above – the network throughput is the average number of packets the system can deliver per clock cycle. If the injection rate is increased the throughput increases accordingly until the network is saturated. Above saturation a stable network continues to deliver the peak throughput; however, if a network is loaded above saturation the packet delay becomes potentially unbounded and hence renders the network unusable. For an unstable network, the throughput drops beyond saturation, e.g. a network that is allowed to drop packets [Tanenbaum2003].

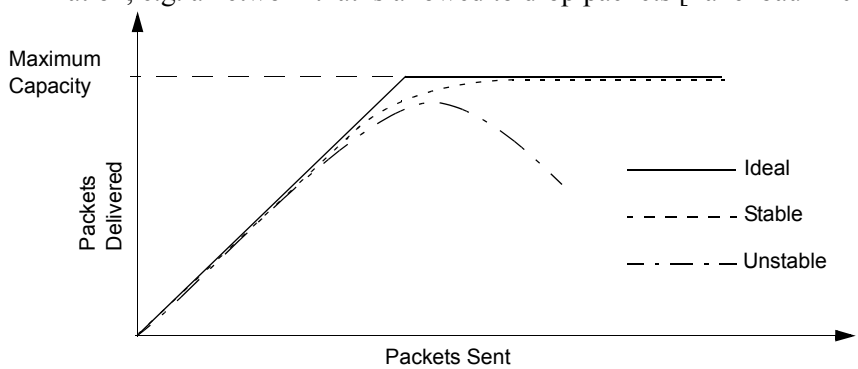


FIGURE 2.25. Stable vs. Unstable Network Operation

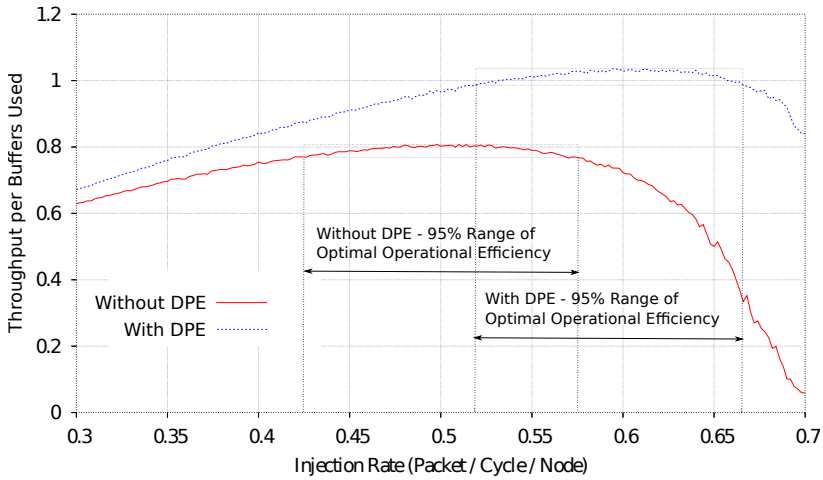


FIGURE 2.26. Operational Efficiency

Given this fact it is easy to jump to the conclusion that the best performance is achieved by driving a *stable* network as close to the saturation point as possible! However, loading the network to this extent must come with a cost. The cost is the increase in average buffers needed to transfer a packet through the system, together with a potentially higher worst case latency. Ignoring the worst case latency, we propose the measure of *Operational Efficiency* to capture the *Throughput per Buffers Used* in the network.

The idea is that an underutilised as well as an over-utilised network will give a bad ratio between the throughput delivered and number of buffers used. By making this definition it is now possible to find a 'sweet-spot' in the graph. In Figure 2.26 the concept of Operation Efficiency is illustrated with a comparison between a mesh network utilising our Dual Packet Exit (DPE) concept and one without. The graph shows the packet injection rate versus the Operational Efficiency. The *Buffers Used per Packet* is defined as *the average number of buffers needed to transfer a packet from source to destination*; this could also be interpreted as the average packet latency. Since the Operational Efficiency is derived from

$$OperationalEfficiency = \frac{SystemThroghughput}{BufferUsedPerPacket}$$

The immediate observation that can be made in Figure 2.26 is that with the DPE the throughput of the network can be increased while still using the same number of buffers.

Furthermore, it can be seen the network utilising the DPE approach is more effective. If we define a region where the network operates within 95 percent of its optimum that region is considerably moved in a higher throughput area ([0.44..0.59] vs. [0.53..0.67]) and the drop-off of in efficiency also happens much closer to the saturation point of the DPE network.

Partha Pratim Pande et al. were thinking along the same lines when they defined the *Bit energy per throughput* [Pande2005]. This does, however, not give any 'sweet-spot' in the graph and served the purpose of being a comparative measure between different topologies.

Traffic Classes

The measures above, however, do not exist in isolation – often the traffic (or flows) utilising a guaranteed service could be characterised to belong to one of the aforementioned *traffic classes* where one or several of the guaranteed characteristics above are components. This means that even though a service that is characterised as *Best Effort* may as well posses Guaranteed service characteristics such as *Data integrity, In-Order Data Delivery, etc*. In QNoC (Quality of Service NoC) the traffic is divided into four *Service Levels* (SLs) according to different types of communication requirements [Bolotin2004b]. By making this division it eases the life for the system architect since it may be easier to sort the traffic into these service levels than to have to be explicit in a detailed traffic characterisation. They suggest the following traffic levels:

Signalling covers urgent messages like interrupts and control signals and very short packets that are given the highest priority in the network to assure the shortest latency.

Real-Time service level guarantees bandwidth and latency to real-time applications, such as streamed audio and video processing. This service is packet based; a maximal level of guaranteed bandwidth is allocated to each real-time link and should not be violated.

Read/Write (RD/WR) service level provides bus semantics and is designed to support short memory and register accesses.

Block-Transfer service level is used for the transfer of long messages and blocks of data, such as cache refill and DMA transfers.

In QNoC a priority ranking is established, where **Signalling** is given the highest priority and **Block-Transfer** the lowest. The QNoC employs preemptive communication scheduling where data of a higher priority packet is always transmitted before that of a lower service level (a round-robin is employed within service levels). This gives a service that has soft (statistical) guarantees.

How to Measure?

Without going too deep into the realm of measurement I would like to make some remarks on this rather big topic. For the interested reader, I could point to the simulation chapter of *Principles and Practices of Interconnection Networks* by William J. Dally and Brian Towles for a more thorough introduction to interconnection network simulation [Dally2003]. Some of the service characteristics above are "obvious" in their nature and given by the implementation of the network e.g. *Data integrity, Lossless Data Delivery, In-Order Data Delivery*, etc. Other characteristics are determined by the implementation of the network in combination with the current traffic e.g. *Throughput, Latency, Jitter delay*, etc. Often it is possible to give some theoretical upper/lower bounds on these characteristics, but they tend to be over pessimistic. The measurements that can be done are often not only restricted to a single number – rather the results tend to be distributions from where numbers such as average, worst case, best case, confidence intervals, variances, etc. could be derived. Sadly, it does not end with these distributions, packets within a traffic flow tend to have temporal correlations as well, that is, it may be more likely that, given a particular packet has a certain latency, a consecutive packet may have a similar latency.

Useful vs. Background Traffic

To characterise and understand the networks behaviour it is necessary to make measurements – but what/how to measure? One simple set up could be to inject traffic/a flow into a loaded network, i.e. a network with some background traffic, and simply make some easy measurement on that particular flow. This may give some useful information about that very flow but does not capture whether that particular flow impairs the "background" traffic. This means the traffic is actually divided into two sets – the traffic that is currently under measure – the useful traffic – and the background traffic. By varying the amount of traffic belonging to the respective traffic sets a landscape of interesting measurement points can be defined.

The first and easiest thing to measure on is the "empty" network – no background traffic. This will give some lower bound on the best case scenario for any traffic flow. By gradually increasing the useful traffic the saturation point of the network will be found in terms of latency, throughput and jitter delay.

Spatial Characteristics

Not only the amount of traffic injected into the network will be of importance – the points where the traffic is injected and ejected will greatly affect the result as well. The spatial characteristics will give heaps of useful information how well the particular network distributes the traffic and how it responds to congestion.

Temporal Characteristics

The most natural temporal characteristic is the traffic bursts. A network will most likely respond very differently to short and long messages – that is – consecutive packet sequences that belong to the same message. If the network lacks e.g. virtual channels or the possibility to utilise non-minimal paths the result of long messages may be that the network stalls and a saturation tree is formed that drastically reduces the overall performance of the network. On the other hand – too many short messages may be costly for a network that requires that a connection is set up before transmission.

Synthetic vs. "Real" Traffic Scenarios

Moreover, the choice of traffic scenario will affect the behaviour. A purely synthetic random traffic generator may give valuable insights in early stages of development but turns out to be completely useless in the process of analysing the network suitability for a given application. On the other hand – analysing a real application trace might not say anything about the characteristics of a particular network just how a specific trace behaves on this particular network.

Half Full or Half Empty?

The conclusion to draw from what is said above is that measurements are easy to perform but to choose the *right* measurement seems extremely hard. Not only that the universe spanned by the combinations of all the scenarios that could potentially be used for measurements are huge.

The results from any single simulation could potentially be analysed in a multitude of ways. So – dependent on the mind-set of the network developer the possibly enormous set of simulation scenarios could either be viewed as an asset or as a hinder. Within the NoC community there has been attempts to tackle these and similar problems of characterisation of networks and applications e.g. by Vassos Soteriou et al. [Soteriou2006]

2.5.2 Best Effort

What comes to mind when discussing Best Effort performance is why providing a service that comes with no guarantees? The existence of the Best Effort actually stems more from consequences than from the need. First of all – the Best Effort is cheap – virtually indefinitely cheap since no promises are made! More practically it does have a cost, but, it is low. If a platform for communication exists it can be employed directly – without changes – to deliver Best Effort performance. The second reason for the existence of Best Effort is that it is a possibility to use unreserved or unused capacity of a guaranteed service. This, since, the guaranteed services have a strong tendency to become expensive in terms of over-allocation of resources.

When discussing the Best Effort service the characteristic that is assumed to be of a Best Effort nature is often throughput or latency – implying that, e.g. the *Data integrity* and the *Lossless Data Delivery* characteristics are guaranteed. Once again, it is of importance to articulate *what we actually mean with guarantees*. A service that claims to have the characteristic of e.g. *Lossless Data Delivery,* in reality, also have implied that there exists an upper bound on the worst case latency – otherwise such claim is meaningless!

The characteristics of the Best Effort services are often based on average cases in comparison to the Guaranteed Services that has to take the Worst case into account. The worst case is very likely much lower that the average case, and hence we get the aforementioned over-allocation of capacity for the guaranteed services.

Some groups have implemented a ‘Bastard’ variant of Best Effort that they call a *Guaranteed Service* where they offer a “guarantee’ that a packet is delivered with a latency less than a fixed number with a *probability* of 99.5 percent [Santi2005]. In relation to what I “defined” in the opening of this chapter I would rather refer to this as a Best Effort Service.

In their experiment, they implemented a simple priority mechanism that gave priority to the Guaranteed Service over the plain effort service. The ratio in terms of traffic devoted to the different service classes was 10 percent respectively 90 percent. The idea is appealing but would rather be sold as two classes of BE with different characteristics where the high priority class is *guaranteed* to pre-empt the other.

Platforms that both offer BE and "true" GT services and utilises the "spare" capacity for the Best Effort services is our Nostrum, the Æthereal platforms and the NoC of Praveen Vellanki et al. In Nostrum the guaranteed traffic utilises a TDMA scheme and any unused capacity is devoted to a deflection routing scheme [Millberg2004a]. In Æthereal and the NoC of Vellanki et al. both services is handled by a TDMA based scheme with the Best Effort utilising un-allocated time slots [Goossens2005 and Vellanki2004].

2.5.3 Guaranteed Services

There exist a number of platforms of today that implement some sort of Guaranteed Service. There are several different options; Æthereal and our Nostrum uses a TDMA (Time Division Multiple Access) based scheme, the MANGO (Message-passing, Asynchronous Network-on-Chip) NoC of Tobias Bjerregared et al. [Bjerregaard2005a] and the NoC router presented by Tomaz Felicijan and Steve B. Furber [Felicijan2004] have chosen to implement Virtual Circuits to serve the purpose – both platforms are clockless.

To provide enough bandwidth to honour committed network service guarantees often means over allocation of network resources. For the "traditional" GT this seems inevitable since the user of the service may not use its given guarantees fully. One way to get around the problem is to introduce a "new" service that mixes the properties of the Best Effort with the traditional GT. One example of this is the SuperGT of Théodore Marescaux and Henk Corporaal [Marescaux2007]. Their idea is to provide a packet based in-order delivery service with a guaranteed base-level, of throughput. On top of the base-level, it is possible to inject traffic on a particular SuperGT connection but this has in-order Best Effort guarantees. Since their SuperGT is based on Æthereal the GT part is done by pre-allocating slots in the TDMA based scheme. The BE part is then handled by claiming "free"/un-allocated slots [Goossens2005]. Marescaux and Corporaal report an increased throughput *up to* 35 percent at a cost of 6 percent extra area. How their solution would affect other traffic in the network is, however, unclear.

2.5.4 The Process of Allocation

When providing Best Effort Services and Services with some guarantees the process of allocating capacity for the service and the process of putting the service into use deserves a few words. In a paper from 2004 we have chosen to make the following distinction [Millberg2004a]

- *Static Allocation* – The mechanisms behind the service are hard-coded, decided at design time or at least in the start-up phase of the system. In our case, with our looped containers, the routes of the containers would be hard-coded in the switches and all the containers would be launched in the start-up phase of the system
- *Semi-static Allocation* – Same as above with the exception that the containers would be launched when needed. This implies that any "guaranteed" service would only provide guarantees *after* all the necessary containers have been successfully launched.
- *Dynamic Allocation* – The set-up of the mechanisms for the service would be programmed on 'the fly'. In our case, the container packet routes in the switches would have to be programmed once needed as well as the launch of the containers. Such complex mechanism would have to be orchestrated by some central operating systems that would either analyse any request during runtime or have to be responsible for reprogramming the network utilising a use-case based approach.

Radu Marculescu et al. make a similar division of how to administer guaranteed services into three distinct groups [Marculescu2008]. (1) Virtual Circuits can be pre-reserved and allocated statically off-line. (2) Multiple priority levels can be used to favour high priority traffic. (3) The Network interfaces can be made aware of current traffic load and hence regulate traffic to ensure the guaranteed services.

Axel Jantsch and Zhonghai Lu have chosen to group the resource allocation techniques into three main categories: *Circuit switching*, *Time Division Multiplexing*, and *Aggregate Resource Allocation* [Jantsch2009]

- *Circuit switching* – All necessary resources are allocated for the entire life time of a connection. In every switch, there is a table that defines the connections between input ports and output ports. The output port is exclusively reserved for packets from that particular input port. With few exceptions such as SoCBuS [Wiklund2003] and Crossroad [Chang2006] circuit switching has not widely been used in NoCs because only few applications justify the exclusive assignment of resources to each connection.

- *Time Division Multiple Access (TDMA)* – Resources are exclusively allocated to a specific user during well-defined time periods or *time slots*. The time slots often have the granularity of clock cycles. The allocated time slots are encoded in a slot allocation table with one table for each shared resource, e.g. a link. The authors identified two major drawbacks with this scheme. First, there is a trade-off between detail of bandwidth allocation and table size since a finer granularity for bandwidth requires larger slot tables. Second, it exists a direct relation between granularity of allocatable bandwidth and maximum delay. If the bandwidth allocated is k/n with k out of n slots allocated, a packet has to wait n/k − 1 cycles, in the worst case, for the next slot to appear. In the word of NoCs the TDMA is employed in e.g. our Nostrum platform, in Æthereal and in the NoC of Vellanki et al. [Millberg2004a, Goossens2005, and Vellanki2004].

- *Aggregate Resource Allocation* – Each resource is assigned a traffic budget for both sent and received traffic. The reasoning is that, if all resources comply with their budget bounds, the network is not overloaded and can guarantee minimum bandwidth and maximum delay properties for all the flows. Traffic budgets can be defined per resource or per flow, and they have to take into account the communication distance to – correctly – reflect the load in the network. Aggregate allocation schemes are flexible but give looser delay bounds and require larger buffers than finer grained control mechanisms such as TDMA and circuit switching.

By statically allocating capacities to each flow better utilisation and tighter QoS bound can be achieved. Zvika Guz et al. analysed the link capacities off-line in their wormhole based NoC platform and came up with an implementation that meets all dead lines for the individual flows [Guz2006].

The analytical model of their NoC was shown to be very close to the simulated results. Offline is obviously to prefer if the SoC has a predictable behaviour that *can* be analysed off-line. The same methodology used may be employed while allocating capacity for the aforementioned use-cases (or traffic scenarios).

The statical assignment process does, however, assume a very detailed knowledge about the system that can be used in the analysis process. This assumption about the statically known (and schedulable) system by Guz has been assumed by many others [Hu2004 and Liang2000]

2.5.5 Packet Reordering

The GT can involve guarantees on the different measures like the ones listed above, latency, throughput, etc. but can also include other *properties*. One such property is *in-order delivery*. The reason this may be of importance is the cost associated with the potential reordering of packets. The problem of packet reordering, and the assumed associated cost of reorder buffers in the NI has been articulated by many authors – Israel Cidon et al., Huimin Hu et al., Mieszko Lis et al., Andrei Rădulescu et al., Erno Salminen et al., Radu Stefan and Kees Goossens [Cidon2005, Hu2003, Lis2009, Radulescu2005, Salminen2008, and Stefan2009]. There is, however, to my knowledge, only one analysis with numbers presented on this assumed cost in the context of NoCs. The paper is *A Case for Bufferless Routing in On-Chip Networks*, of Thomas Moscibroda and Onur Mutlu and gives a detailed quantitative analysis of the associated cost of reordering [Moscibroda2009]. Their conclusion is that the gain of a bufferless approach is higher than the cost – see Section "Deflective Routing (or Hot potato)," on page 68 for further information.

Sergio Tota et al. identify the problem and think it is manageable but that further analysis is needed [Tota2006 and Tota2007]. In our article about *Priory Based Forced Requeueing,* we made an estimate on the implementation cost of a reorder buffer [Millberg2009]. The presented numbers on the implementation cost are most probably over pessimistic due to that the implementation assumed a continuously sorting reorder buffer of a depth of an observed worst case delay. In a reorder buffer in the NI (on the receiver side) the message lengths are most likely smaller; this because the stream based data transfers are more suitable to be sent over a Virtual Circuit not as Best Effort traffic!

2.6 The Layered Approach to NoCs

To master the increasing complexity of today's billion transistor Systems on Chip the orthogonalisation of concerns are essential. Kurt Keutzer et al. recognised this in their article from 2000 [Keutzer2000]. In this article, they articulate the necessity of decoupling *computation* from *communication*. This decoupling enables the computing components like micro processors, DSPs, etc. to be designed independently from the communication part – the NoC. This decoupling, however, requires that the services that could be expected from the network on chip are well-defined and independent of the real network implementation – the decoupling of *function* from *implementation*. Further the NoC itself may need to be divided into more or less autonomous parts that hide implementation details. For a deeper discussion see Drew Wingard's text, that gives a good introduction [Wingard2005].

In order to separate the different concerns that exist when implementing a Network on Chip a layered approach can be taken. This has been articulated by us [Millberg2004b] as well as by others [Sgroi2001, Goossens2002, Benini2002, and Benini2001]. In general the upper layers of the stack are done in software and the lower layers are in hardware. The lower layers manifests as the Network Interface, the switches and links of the Network.

If networks are to be compared with buses the added functionality of the networks will require a protocol stack to manage the complexity of the network as well as being able to offer differentiated services. The pressure to keep the protocol stacks small is, however, higher on chip than off chip due to the relatively modest size of the attached IP blocks [Radulescu2005].

Since there was no real reason for reinventing the wheel the OSI protocol stack [Zimmermann1980] was early identified as a strong candidate to use as a reference model when describing the different layers needed in a NoC protocol/hardware stack. The Open Systems Interconnection Reference Model (OSI Reference Model or OSI Model) is a conceptual, abstract description for layered communications and computer network protocol design. The original OSI model protocol stack is primarily not used as a "real" protocol stack – it works more like a reference used to position other protocol stacks. Nevertheless, there exists a very precise definition of the responsibilities of the different layers defined by ISO in a series of documents where e.g. the X.200 standard describes the OSI – Basic Reference Model [ISO1994].

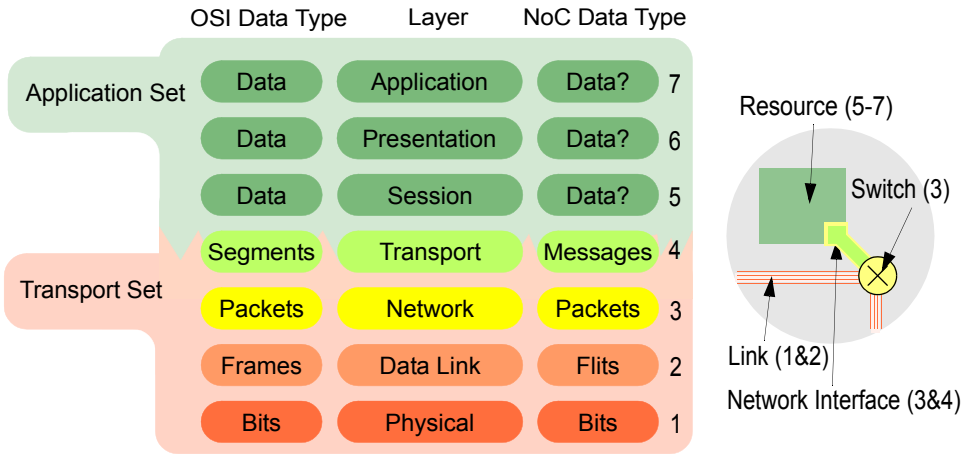


FIGURE 2.27. OSI Model

Within the original OSI model, there are seven layers defined as depicted in Figure 2.27. The three lowest layers of the OSI protocol stack (Physical, Data Link & Network) are the ones that are the most concerned with the transportation of data and are very much dictated by the hardware architecture. The transport layer will work as an interface between the "pure" hardware world and the potential software oriented application set, whereas the topmost ones function as wrappers and formatters of data for the resource.

Within the collected NoC literature most papers that could be discussed within the context of the OSI would fall into the *Data Link*, *Network* or *Transport* layers. I will exclude the Physical layer since it mainly defines the electrical and physical characteristics of devices, which are more of layout issues than a specific NoC problem. The interested reader is referred to the article of Luca Benini et al. [Benini2001].

Layer 2 – Data Link

"The Data Link Layer provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the Physical Layer." [Jennings1993]

In our NoC realm, this concretely means guaranteeing the transport of flits from one switch to another. This may involve handshaking and distribution of back pressure signalling. Potentially, error detection and correction are encompassed by the data link later as well. If an error is detected and not corrected, upper layers must be informed so that measures can be taken.

Layer 3 – Network

"*The Network Layer provides the functional and procedural means of transferring variable length data sequences from a source to a destination, via one or more networks while maintaining the quality of service requested by the Transport Layer. The Network Layer performs network routing, flow control, segmentation/desegmentation, and error control functions.*" *[Jennings1993].*

The network layer is hiding the network internals such as topology routing scheme for the communication IP modules. As identified by us and Kees Goossens et al. this can be divided into two parts [Millberg2002 and Goossens2002]. The first, and the most essential part, involves packet delivery, routing congestion control as well as scheduling. The second part has more of a maintaining character and involves network management and collection of statistical data for diagnosis of the network functionality. Even issues of fault tolerance could be characterized as a responsibility of the Network Layer. The two extremes of switching policy that could be implemented by the Network layer is circuit switching versus packet switching. If the traffic flows of the network are persistent, that is, that they are likely not to change, or if they have very specific demands on latency or throughput the circuit switching is the technique to use. On the other hand, if the traffic pattern changes during run-time packet switching have to be considered due to the potentially very high cost of setting up circuits that will be underutilised. These issues are a central part in the process of the system design to minimize the energy consumption. In networks on chip the Network Layer is implemented in the Network Interface (NI). To keep the cost of the NI as low as possible Andrei Rădulescu et al. suggest that it should be fully implemented in hardware. In their implementation – Æthereal – the Network layer of the protocol stack is called the *NI-core* [Radulescu2005].

Layer 4 – Transport

"*The Transport Layer provides transparent transfer of data between end users, providing reliable data transfer services to the upper layers. The Transport Layer controls the reliability of a given link through flow control, segmentation/desegmentation, and error control. Some protocols are state and connection oriented. This means that the Transport Layer can keep track of the segments and retransmit those that fail.*" *[OSI-Wiki].*

The Transport layer in the world of NoC will work as the boundary between the resource and the network; also it is a divider between the pure hardware oriented network and the software based upper part of the protocol stack. This layer will be responsible for the packetisation/depacketisation of the messages/data that is sent over the network. If a multipath, and/or a reordering strategy on routing, is being employed the packet reordering will potentially be an issue for this layer as well. The packet size employed can be application specific within one SoC and has been identified as having an impact on the energy consumption of the network [Ye2003]. Further this layer will be responsible for the flow control of the network and the administration of Best Effort vs. guaranteed services. In the world of our Nostrum platform, the Transport layer is implemented in the RNI (Resource Network Interface) which could be both implemented in both hard and /or software dependent on what services it is supposed to deliver to the upper layers. In Æthereal this is handled by something that the authors call a *shell* which is connected to one more of the Ports of the Network interface [Goossens2008].

Further the Transport layer will serve as an interface in that it enables synchronisation between the different clock domains of the network and the resource. This enables a GALS (Globally Asynchronous Locally Synchronous) design style. In its purest sense, it has been implemented in the MANGO platform since that NoC is asynchronous or clockless [Bjerregaard2005b]. For other platforms employing a clocked network it is simply a place for synchronisation and buffering to compensate for variations in the data up- and downstream flows. These buffers of the Network interfaces have been identified as a major contributor to the area of the NoC [Coenen2006]. In addition, these buffers constitute a boundary between the conceptual realms of communication and computation. Some work has been done in trying to reduce the sizes of these buffers. For example, Martijn Coenen et al. suggest that a credit based system could be used for end-to-end traffic control [Coenen2006].

The purpose of the software oriented top most layers is to provide the resources with an abstraction of the underlying hardware platform and offer system oriented maintenance services. Luca Benini et al. [Benini2001] envisioned that the software layers of the protocol stack would play the role of system software and should implement some sort of Dynamic Power Management in order to offer different service levels. Furthermore, they suggested that this system software should handle dynamic information flow management that could re-configure the network at run-time to meet changing demands of throughput, etc.

Since NoC design and its accompanying complexity will not automatically be solved by a clever platform and a set of layers this also has to be complemented with a methodology. Marco Sgroi et al. identified this at an early stage and hence proposed that methodologies and tools for NoC development must be able to handle and avoid protocol implementations that are incorrect (e.g. due to deadlocks and race conditions), or sub-optimal (e.g. is power-hungry or introduce unacceptable latency) [Sgroi2001]. To tackle this, they suggest that a methodology:

- When applied should add discipline to on-chip communication design and hence enable the transition from *ad-hoc* SoCs to *disciplined* IC platforms.
- It should be based on formal Models of Computation and support a correct-by-construction synthesis design flow and a set of analysis tools for broad design exploration.
- Maximize reuse within the definition of a set of interfaces between layers
- Provide an application programmer with a set of APIs abstracting architecture details.

2.7 SUMMARY

The concept of Networks on Chips (NoCs) has been proposed as the future communication platform for large Systems on Chip with potentially hundreds of communicating peers as well as for Multi Processor Systems on Chip. NoCs possess attractive properties as being inherently scalable, regular and predictable. Furthermore, NoCs provide standardised interfaces that encourages reuse and a structured design style. This, in combination with a natural decoupling between the computation and communication in a layered fashion eases product development as well as reducing timing closure issues.

Even though Networks on Chips borrows from parallel computer networks as well as from traditional off-chip networks the constraints of the on-chip networks differ. On chip networks of today have a limited power budget, limited buffer capabilities as well as having requirements on a planar topology and switches of low degree.

Several topologies have been proposed for Networks on chip, most of them having simple layouts due to the two-dimensional mapping that a on-chip layout requires. The most common topologies are variants of meshes and custom topologies. Meshes are favoured for their predictability in layout and routing; custom topologies for their ability cope with an irregular layout.

Except for the topology, care must be taken when the routing and switching policy is chosen. Routing is the process of selecting paths in a network from a source to destination along which to send network traffic; switching involves the actual transportation of data. Within the NoC community the routing and switching policy and the corresponding hardware solutions have been the inspiration for numerous publications.

To be able to cope with the varying demands that different communication peers may have support for Quality of Service characterised communication has been developed. If this scenario is to be painted with the broadest strokes the services provided by the NoC could be classified as Best Effort (BE) and Guaranteed Throughput/bandwidth (GT) where GT is mainly used for communication that have specific demands on bandwidth, throughput and/or latency and a set-up time for the communication link can be accepted. Best effort is used for "the rest" of the traffic that requires a quick means for instant communication and the actual performance is of less importance.

To enable separation of different concerns a layered approach is taken. The layered approach help in separating *communication* and *computation* as well as *function* from *implementation*.Withing the layered approach the OSI reference model has been successfully utilised as a source of inspiration and the result is presented in many publications and is represented in many reference designs

Today the Network of Chip research community is a mature community with a history that spans more than a decade. Several companies – Arteris, Sonics, Inocs to name a few – are today providing full on chip communication infrastructure solutions with tool suits that support entire design flows from early system designs pre-simulations to post synthesis verification. Many commercial platform inherently have support for the most common communication protocols. like AXI, OCP, AHB etc.

3 Conclusions

Contemporary System-on-chip platforms usually contain bus based interconnection infrastructures, where a designer can create a new system by configuring and programming the cores connected to the busses. However, global on-chip communication is becoming a problem as future silicon chips become larger, technology scales down, and the clock frequency increases. Bus based platforms suffer from limited scalability and poor performance for large systems. Network-on-Chips (NoCs) are the network based communication solution for SoCs. They inherently encourage reuse of the communication infrastructure across many products thus reducing design-and-test effort as well as time-to-market.

However, if these NoCs should be useful, different traffic classes must be offered due to the various requirement from the SoC. Typically, these traffic classes fall into two categories – Best Effort and traffic classes with Guarantees. These guarantees include bandwidth, throughput, latencies, jitter, etc. To be able to offer these traffic classes a Network-on-Chip communication platform must be both flexible and efficient.

To meet these specifications we have developed a Network on Chip platform – Nostrum. Nostrum is a layered architecture that inherently supports both Best Effort as well as Guaranteed Throughput traffic delivery. The layout of Nostrum is based on a planar Manhattan structure which gives a straight forward mapping process. Planar topologies can with advantage be used if the traffic pattern possess a certain locality.

During the development of Nostrum the concept of a modular, layered design with clear interfaces has shown to be of utter importance. Any changes – like extensions to the protocol, routing strategy etc. – have relatively easy been incorporated in the design. The modular structure also enabled and simplified cooperation in the design process. I believe that the benefits are even greater when a full system is to be designed due to the inherent separation between the communication and computation realms.

Transmission of data from a sender to a receiver over a NoC involves several stages. The data has to be segmented into packets and put in the downstream queue waiting for the arbitrator controlled admission to the network. Once out on the network the packet needs to be transported in a safe and effective way over the network to finally be delivered to the designated destination. To deliver good performance attention needs to be paid to all of these stages.

In Nostrum the traffic classes with guarantees are handled by Virtual Circuits. The Virtual Circuit gives guarantees on latency and throughput and employs a variant on Time Division Multiple Access (TDMA). The concept – we call it looped containers – utilises proxy packets that are going back and forth between the sender and receiver. In this way a certain amount of the networks capacity is always claimed for this virtual circuit since the container packet guarantees an unbroken connection between the sender and receiver. This looped container solution covers – and guarantees – Admission, Transport as well as Exit from the network. The cost of implementation is very small with very little extra hardware and an extra packet payload of only 2 percent.

For the for Best Effort traffic delivery Nostrum employs a deflective routing scheme that gives a small footprint of the switches in combination with robustness to disturbances in the network. The deflective routing does however introduce a potential reordering of packets within a message as a natural side effect to its flexible routing scheme. To minimise the cost of this effect the best effort traffic delivery should first and foremost be used for short messages. Stream based data should be assigned to the Virtual Circuit based services. Hence, the deflection routing policy is a competitive policy thanks to its modest hardware requirements and flexibility; but it should be complemented with a traffic delivery scheme that has support for stream based data delivery.

The non-reordering packet policy in the switches[1] of the deflection routing scheme creates something that we call Temporary Disjoint Networks (TDN). The TDNs make the network to be separated into several independent networks. These separate networks can be utilised for different traffic classes with different guarantees, e.g. a network with little traffic naturally has a lower bandwidth but tighter bound on delivery for instance. Once again – any platform that aspire to be a competitive platform the Systems on Chip communication of the future must inherently support several traffic classes due to the varying demands and diverse requirements of the applications. Since the TDN is a characteristic of the platform and not an implemented feature no extra hardware is required.

For the Admission phase of the Best Effort I have proposed the concept of Priority Based Forced Requeue (PBFR). PBFR increases the fairness in the system by making the packets waiting for admission to the network compete with the packets already in the network. The concept gave a 50 percent reduction in worst case latencies in the system which gives tighter bounds on latency in the communication process.

To enhance the Transport phase for Best Effort traffic we suggest the Proximity Congestion Awareness (PCA) as part of the solution. PCA further enhances the flexible capabilities of the deflective routing scheme to incorporate the possibility to actively avoid congestion if possible. The PCA make the individual switches aware of the load in their immediate surroundings of the network. The neighbouring switches communicate their current load by sending out a *stress value* to be used in the routing process. Utilisation of PCA was shown to reduce the maximum average load in the downstream queues of the network with up to 20 times.

Finally, to enhance the Exit phase performances from the network I propose Dual Packet Exit (DPE) as a remedy. DPE simply means that two packets per clock cycle are let out from the network from every switch in the network instead of one. The DPE gives a 50 percent reduction in terms of worst case latency and a 30 percent reduction in terms of average latency as well as an increased throughput both from a system and network perspective. These very significant improvements stems from the fact that the network should not be used as a storage space for packets but only for transportation. The cost associated with DPE is that packets now need to be stored outside the network instead of inside.

1.The switches does not reorder packet but due to the flexibility in routing paths different packets may take different paths and hence arrive out of order.

Despite that the thesis to a large extent is abut how to enhance performance in different parts of the network a still feel that I must emphasise the importance of a good mapping and a well carried out system analysis. During the development of the presented strategies in the thesis the importance of the mapping has been evident many times. An ill-mapped system often worsens the performance to a much larger extent than a badly chosen routing strategy. So, to utilise the network at its full potential both a good mapping and a well functioning packet delivery service is of utter importance!

In conclusion – the Networks on Chips will make their major break through when the chip sizes become too big and the bus based system no longer can cope with the traffic size and the various demands of different traffic scenarios. However, even though the thesis may advocate the Networks on Chips as the communication platform of the future I still believe that the bus will stay to handle *local* traffic of chip. Hence, it will not be *replaced* by the Networks on chips but rather be *complemented* with NoCs handling the *global* traffic on chip.

4 Paper Results & Author's Contributions

4.1 Paper I – Load Distribution with the Proximity Congestion Awareness in a Network on Chip

Design Automation and Test in Europe – DATE 2003

Authors: Erland Nilsson, Mikael Millberg, Johnny Öberg and Axel Jantsch

This is the full length version of the paper that was submitted to DATE 2003. A shorter version of this paper was accepted to the poster session.

This paper describes the concept of *Proximity Congestion Awareness (PCA)* that is used to reduce the total load of the network. The paper also introduces the average FIFO usage of a network. The main result of the paper is the substantial improvement with about 20-time load reductions when using the PCA concept.

In the paper, a discussion is carried out how the bufferless deflective routing policy of our platform Nostrum can create hot-spots in the centre of the network. The reason for this behaviour is that a super-set of all possible routing paths to and from all senders and receivers in the network will have an overrepresentation of potential paths in the centre of the network. The remedy to this is to try to divert packets away from congested areas. The solution we propose is based on the idea that all switches monitor their current load and distribute this information – the stress value – to its neighbouring switches. The stress value is the sum of the number of incoming packets averaged over the last four clock cycles. The related hardware cost is reported to be very moderate in relation to the benefit. Synthesis of the switch was made using lsi10k technology. When optimised on speed, an area of 21 029 and a gate depth of 48 was achieved.

Further, the switch design is described to explain the experiments on the three modes of operation that were tried out. The three settings are: no stress value, one cycle stress value, and averaged stress value over four cycles.

A comparison of the resulting FIFO load of the different setting is presented in textual form in the paper and shows that the reduced maximum average FIFO load is greatly reduced and that the load has a wider distribution over a larger area. We report a 20-time average load reduction when using the PCA concept.

Thesis author contributions: In this paper, I'm a major contributor with a heavy role in the development of the concept, experimental set-up as well as in the initial VHDL development even though Erland Nilsson did most of the writing.

4.2 Paper II – Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks

Design Automation and Test in Europe – DATE 2004

Authors: Mikael Millberg, Erland Nilsson, Rikard Thid and Axel Jantsch

This paper addresses the problem on how to implement services with guarantees such as throughput and latency in a bufferless deflective network. In the paper two separate concepts are presented (1) the *Temporally Disjoint Networks* (TDNs) and (2) the *looped containers*.

The idea behind the *Temporally Disjoint Networks* is that a physical network, potentially, can be seen to contain a set of separate networks that a packet can enter dependent on *when* it enters the physical network. A necessary condition for the existence of these TDNs is that a position in the network can only be reached on a multiple of N hops where N must be greater than 1. As a consequence the number of TDNs that exist, N is equal to the number of hops it takes to leave a switch and then get back to the very same switch in such a network.

In our case, these conditions will be fulfilled in a Manhattan network with logically identical switches that performs no reordering of packets. If the network is a torus the number and rows and columns must be even to render more than one TDN.

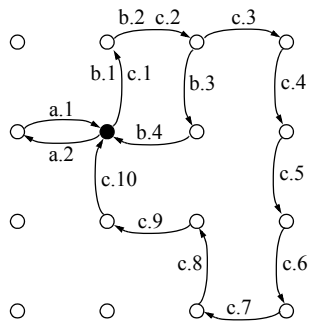


FIGURE 4.1. Temporally Disjoint Networks

To illustrate the idea three different routes have been layered out in the Manhattan network in Figure 4.1. Their respective path lengths are 2, 4, and 10. Hence, the number of TDNs that exist in this network will be 2. This means that two packets that are sent out consecutively on the network (i.e. in cycle n and cycle n+1) will never be able to collide! This has the consequence that we could have different types of traffic in the different TDNs to enable different guarantees on latency and throughput in the respective TDN.

The second *concept* that was presented in the paper – the *looped containers* – provides a means to set up virtual circuits in a network that employs a deflective routing policy. To give guarantees in latency and throughput for individual packets three separate, but linked, problems must be solved. (1) The packet must be able to enter the network at a given time, the packet must be transported over the network within a certain time, and (3) the packet must be guaranteed an exit from the network in order to be delivered properly. Failing in solving either of these problems means that limited or no guarantees on latency can be given!

To solve these problems a proxy packet is inserted into the network – bound to follow a predefined, closed, route between the source and the destination. This particular proxy packet is given the highest possible priority in order to guarantee precedence over any other packet. Once the packet visits the sender it is loaded and sent to the destination where it is unloaded – hence the name *looped container*. If some effort is put into the mapping process of senders, receivers, and container routes a set of different virtual circuits can be set up. In the case where the cycling time of different loops shares a smallest common divider they can overlap without risk of collision.

In the case where this may not be possible the virtual circuits can be placed in different TDNs to guarantee an interference free behaviour. This is the second use of the Temporally Disjoint Networks presented in the paper.

The increased relative hardware cost of implementing virtual circuits using looped containers is less than 2 percent in term of additional gates. The effective relative payload for a packet with 128 bits is more than 98 percent due to the additional two extra bits to implement the concept. Simulations also show that background traffic in the network is very little affected by the VCs; but for the assigned to the VCs, the VCs give a tremendous boost in guaranteed latency and bandwidth. The average bandwidth of the traffic assigned to the VCs is not changed – but now it is guaranteed! As expected, the latency of AB goes from being exponential to become constant.

Thesis author contributions: In this paper, I'm a major contributor of the Nostrum backbone concept as well as in the concepts of the Temporally Disjoint network and the looped containers. Furthermore, I've contributed to a large extent in the experimental set-up. I also did most of the writing.

4.3 Paper III – The Nostrum Backbone – A Communication Protocol Stack for Networks on Chip

The Seventeenth International Conference on VLSI Design 2004

Authors: Mikael Millberg, Erland Nilsson, Rikard Thid, Shashi Kumar and Axel Jantsch

This paper presents our Network on Chip concept Nostrum. The concept defines a packet switched network with support for best effort traffic packet delivery as well as support for guaranteed bandwidth traffic, using virtual circuits. Furthermore, it includes a layered protocol stack and a corresponding nomenclature for describing the individual layers and their interfaces.

Within the concept, a concrete instance is described – the Nostrum Backbone. The backbone is a mesh based communication architecture and defines Resources as hosts for communicating processes. The backbone defines the logical placement of these Resources and how they are connected.

The Nostrum mesh consists of Switch- Resource pairs connected in a two-dimensional grid. Moreover, the relation between Resources, the Resource Network Interface, the Network Interface and the Switches is included. Our layered protocol stack uses a terminology heavily inspired by the OSI reference model. To prove the work of concept a distributed DSP application from industry was simulated. The results showed that the protocol covers the need of the particular example.

A standard protocol stack allows the separate, independent development and validation of resources, communication network and applications as long as they comply with the defined protocols.

This clear separation enables systematic reuse of resources, communication infrastructure and application features. We have defined such a protocol stack within the concept of Nostrum. The protocol stack is defined from the physical to the transport layer and, based on this; the layered Nostrum simulator is developed. The simulator allows experiments with different protocol variants because individual layers can be replaced without affecting other parts.

Thesis author contributions: This paper has to a big extent been based on my technical report on the subject [Millberg2002]. I'm a major contributor of the Nostrum concept as well as the backbone. In addition, I've contributed, to a large extent, in the experimental set-up. I also did most of the writing.

4.4 Paper IV – A Study of NoC Exit Strategies

First International Symposium on Networks-on-Chips – NOCS 2007

Authors: Mikael Millberg and Axel Jantsch

In this paper the exit process of packets in our Network on Chip platform Nostrum is analysed. The analysis gave that there exist an accumulation of packets at the exits of the Network. Our solution to this is to increase the exit bandwidth. The result is a reduction in the worst case latencies as well as in the average latencies. This paper was accepted as a poster and could be considered as a pre-study to the sequel paper *Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy.*

Thesis author contributions: In this paper, I'm the major contributor and did most of the writing.

4.5 Paper V – Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy

10th Euromicro Conference on Digital System Design – DSD 2007

Authors: Mikael Millberg and Axel Jantsch

In this publication we further stresses the benefit of an increased exit bandwidth. From simulation the most beneficial increase in terms of enhanced performance versus cost is to double the exit bandwidth – hence the name *Dual Packet Exit (DPE)*. In addition to the previous reported reduction in worst case latencies as well as in average latencies this paper also stresses the lowered use of buffers. The lowered use is in terms of required buffer capacity as well as in the average number of buffers actively used. The *required buffer capacity* is the buffers that have to exist in the system to cover for any worst case scenario. The *average buffers actively used* is the average number of buffers that is currently holding a packet in the system. From the average buffers actively used in the system we derive and define the term *Operational Efficiency* that is a measure defined as the *throughput per buffers used in the system.* The greatest benefit of this measure is that a graph plotting the Operational Efficiency vs. the packet injection ratio now has a clea*r sweet spot!* This has the concrete impact that an increase of the injection of packets into the network to increase the system throughput will have a cost associated to it and can be optimised to save energy. Using this measure we show that the use of our Dual Packet Exit strategy can significantly increase the system bandwidth without increasing the energy used.

To prove the work of the DPE concept extensive simulations were carried out. For a 4×4 mesh the average system latency is reduced from 14 to 9 clock cycles and the observed worst case latency is reduced from 85 to 45 clock cycles at an injection rate of 0.63. In concrete this means a 50 percent reduction in terms of worst case latency and a 30 percent reduction in terms of average latency as well as an increased throughput both from a system and network perspective. If we set for an average latency of 10 cycles the network with DPE gives roughly 25 percent higher injection rate for the same latency compared to a system without. The validity of the chosen approach is not restricted to uniformly random traffic patterns on meshes but also applicable to "any" topology where the traffic pattern involves potential network exit congestions due to multiple sources having the same destination or where multiple routing paths are possible.

Thesis author contributions: In this paper, I'm the major contributor and did most of the writing

4.6 Paper VI – Priority Based Forced Requeue to Reduce Worst-case Latency for Bursty Traffic

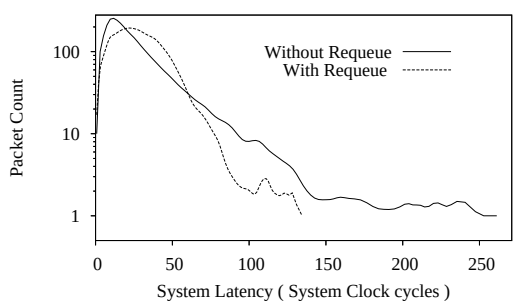
Design Automation and Test in Europe – DATE 2009

Authors: Mikael Millberg and Axel Jantsch

The previous papers in the thesis have very much been focused on the transport and the exit from the network. In this paper, the focus is set on the admission to the network. The main problem that we solve here is that packets may have to wait indefinitely long to enter the network dependent on the current load of the network. The network may have a very fair routing policy that gives priority to "old" packets to keep the worst case latency down. Unfortunately, this does not help the packet that has not yet entered the network. Our solution to this is to make the system more globally fair by introducing a concept called Priority Based Forced Requeue.

Forced Requeue is to prematurely lift out low priority packets from the network and requeue them outside using priority queues. The first benefit of this approach, applicable to any NoC offering best effort services, is that packets that have not yet entered the network now compete with packets inside the network and hence tighter bounds on admission times can be given. The second benefit which is more specific to deflective routing as in the Nostrum NoC is that packet reshuffling dramatically reduces the latency inside the network for bursty traffic due to a lowered risk of collisions at the exit of the network.

Utilisation of the Priority Based Forced Requeue changes the characteristics of the observed system latencies in the system. To illustrate a graph is provided which is a histogram that depicts the latency.



As seen in the graph, and evident from simulation, data the experimental results show a 50 percent reduction in worst-case latency from a system perspective thanks to the reshaped latency distribution Noteworthy here is that the average latency is kept the same.

Thesis author contributions: In this paper, I'm the major contributor and did most of the writing.

4.7 Paper VII – A Network on Chip Architecture and Design Methodology

IEEE Computer Society Annual Symposium on VLSI 2002

Authors: Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johnny Öberg, Kari Tiensyrja and Ahmed Hemani

In this paper a scalable Network on Chip platform is proposed. The platform includes both the architecture and the design methodology. The architecture consists of a mesh populated by switch and resource pairs and communication between the switches is packet based. A resource is assumed to be a processor core, memory, an FPGA, or a custom hardware block, which fits into the available slot and complies with the interface of the NoC. The NoC architecture essentially is the on-chip communication infrastructure comprising the physical layer, the data link layer and the network layer of the OSI protocol stack. The protocols within these layers must be implemented in the resource to network interface (RNI) for every resource in the NoC.

Within the architecture the concepts of regions is defined. A region is an area inside the NoC, which is insulated from the network and which may have different internal topology and communication mechanisms. The concept of region allows for resources of larger size than the atomic slots in the mesh. Regions are connected to the ordinary NoC by special communication arrangements.

The NOC design methodology consists of two phases. In the first phase a concrete architecture is derived from the general NoC template. The concrete architecture defines the number of switches and shape of the network, and the number and kind of resources. The second phase maps the application onto the concrete architecture to form a concrete product

Thesis author contributions: Even though the writing of the paper was coordinated and done mainly by the first authors of the paper the author of the thesis would still claim to be a contributor as being part of the discussion and contributing with central ideas of this paper. The central ideas I felt I strongly contributed to and developed – in text as well as in discussions – are mainly in Section 3 – Network on Chip Architecture. In particular my work heavily contributed to the Network on Chip Architecture Section (3.1) and the Network Protocol Stack Section (3.3) in shaping them to their final form.

4.8 Paper VIII – Evaluating NoC Communication Backbones with Simulation

NorChip Conference 2003

Authors: Rikard Thid, Mikael Millberg and Axel Jantsch

This paper describes a Network on Chip simulator that was developed to evaluate our NoC architecture Nostrum.

The simulator is divided into an application domain and a communication domain. The application domain contains Resource Models (RM) and a Resource mapper. The purpose of the RMs is to, as the name suggests, model the resources of the system. The RMs interact by sending and receiving messages over the network so that the behaviour of the network for a given workload can be studied. The placement of the Resources is managed by the Resource mapper. A designer can easily change the mapping of Resources since all mapping is done in the Resource mapper, and no other part of the simulator is directly affected by the mapping.

The communication domain consists of models of entities that implement the various layers of Nostrum together with a topology generator that instantiate and connect the entities. Four layers are represented in the simulation environment – Transport, Network, Data link, and Physical layer.

To investigate how efficiently our Nostrum architecture can perform some small experiments were set up to compare how Nostrum performs in relation to a bus-based architecture. Nostrum, as well as the bus, was modelled within the simulator. Both Nostrum and the bus were exposed to the same workload model and it was shown that our Nostrum platform can operate at a much lower clock frequency than a shared bus platform. For the bus architecture, a bus clock faster than 1.8 GHz is required, and the Nostrum only needs 200 MHz to handle the workload.

Thesis author contributions: Even though the writing of the paper was done mainly by the first author, Rikard Thid, I would still claim being a contributor as being part of the discussion and contributing with central ideas of this paper. The paper is based on the Nostrum platform and a simulation setup combined. The author of the thesis is a heavy contributor to the Nostrum platform and the concept of layers as well as a main contributor to the concepts manifested in the simulator. Even the experimental setup was a topic for discussion and could hence be classified as a joint effort even though Rikard Thid is the responsible for the actual writing of code.

5 Included Publications
Paper 1

Load distribution with the Proximity Congestion Awareness in a Network on Chip

Erland Nilsson, Mikael Millberg, Johnny Öberg, and Axel Jantsch
Laboratory of Electronics and Computer Systems / Royal Institute of Technology (KTH)
Email: {erlandn, micke, johnny, axel}@imit.kth.se

Pages: 1126-1127 Reference in Thesis – [Nilsson2003]

Note: To give more detail on the concept and provide the full information the submitted full length version of the paper have been included in the Appendix for reference

Appeared in Proceedings of DATE 2003

Load distribution with the Proximity Congestion Awareness in a Network on Chip

Erland Nilsson, Mikael Millberg, Johnny Öberg, and Axel Jantsch
Laboratory of Electronics and Computer Systems / Royal Institute of Technology (KTH)
Email: {erlandn, micke, johnny, axel}@imit.kth.se

Abstract

In Networks on Chip, NoC, very low cost and high performance switches will be of critical importance. For a regular two-dimensional NoC we propose a very simple, memoryless switch. In case of congestion, packets are emitted in a non-ideal direction, also called deflective routing. To increase the maximum tolerable load of the network, we propose a Proximity Congestion Awareness, PCA, technique, where switches use load information of neighbouring switches, called stress values, for their own switching decisions, thus avoiding congested areas. We present simulation results with random traffic which show that the PCA technique can increase the maximum traffic load by a factor of over 20.

1 Introduction

On-chip communication becomes a challenge as the number of transistor functions increases on a single silicon die. Clock and data distribution over large distances is impossible to accomplish in a simple manner. Several research groups propose packet switched Network on Chip [1] [2] to address the problem with communication between Intellectual Properties, where each IP-block is a Resource or a part of a Resource. We propose the use of Nostrum, a two-dimensional Network on Chip using hot-potato routing algorithm as switching policy for datagram distribution [3].

Other ways to organise this mesh are for example the flattened torus model [4], or a plain two-dimensional mesh, the latter is also the model we have chosen to implement Nostrum.

In the plain two-dimensional mesh, the number of packets passing through the centre of the mesh is significantly higher compared to packets traveling along the edges. As every Resource may transmit to any other Resource in the mesh with equal probability, most packets will pass through the centre, which becomes a hot-spot. Such a hot-spot is not desirable.

However, hot-spots can be avoided by distributing control information over the network. We call this concept Proximity Congestion Awareness, PCA. The simulations made in this paper are based on random traffic where each Resource has the same probability of communication to any other Resource in either way.

2 Switch load distribution

The load can spread over a larger area by using different routing rules. For example Round Robin [5], local active deflection, and non-local deflection.

PCA can be used to make the load distribution more uniform. Information to help the Switches in their routing decision is sent between the Switches. The informative value PCA is using is called stress value and is sent from one Switch to its neighbours in all directions. The stress value relates to the load level in that Switch. The surrounding Switches receives four such values from its closest neighbours; this helps each Switch to get a picture of the surroundings.

To avoid oscillations, i.e. a situation when two neighbouring switches get a high stress value every other cycle, causing a packet to bounce between the two switches, the stress value should be averaged over a number of switch cycles.

The incoming packets are sorted in priority order, which in the current implementation is the number of switch cycles the packet has been traveling for; the packet with the highest priority gets to make the first choice of output and the following packets in descending order. The packets which preferred output is occupied by a higher priority packet must be forced in a direction that contradicts to the desired.

3 Simulation results & Conclusion

Packets sent from the Resource are first put in a FIFO-buffer. The study of all FIFOs in the network mesh is a method to see how the load is distributed around a hot-spot.

Appeared in Proceedings of DATE 2003

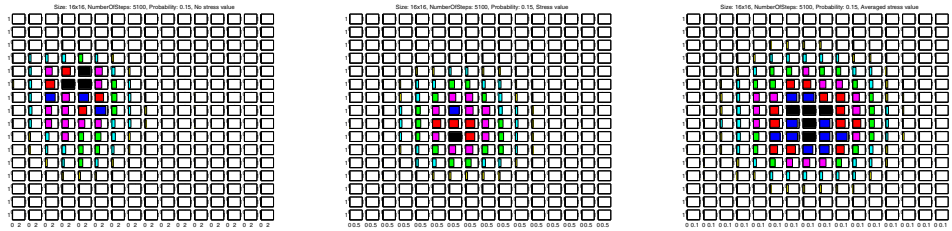


Figure 1. a) Average load i FIFOs without using stress value, largest average number is 3.2 packets. b) Same simulation using stress value, largest average number is 0.9 packets. c) Average load using four cycles averaged stress value, 0.1 packets waiting in FIFO.

The first model acts as a reference where PCA is not used.

The second model is to use PCA with stress values. The stress values are updated every switch cycle.

The third model is one where the stress values are averaged over four cycles. Although, the third model is the most advanced, the stress value averager is fairly small, about 200[1] gates.

In the three cases, the same input data has been used with a mesh size of 16×16. The packet probability is for the first model close to the maximum possible on purpose, shown by simulations [6], to create as much congestion as possible.

An average load of 0.01 tells us that the FIFO is occupied by one packet every 100 switch cycle, which is fairly low. The intention is to show the influence of PCA between the three cases presented.

In figure 1.a, the hot-spot is pushed to the north-west corner, if north is up in the figure. The reason for the non-centered load is a result of the routing decisions in the Switch. The output for the deflected packet is fixed, in this case primarily to the West, secondary North, etc.

The stress value notifies the surrounding Switches about how many packets the Switch handles during that cycle. Using stress values in this manner increases the performance of the Switch since the outputs are ordered in the most preferable order. Compare the maximum value on the x-axis in figure 1.b, which has a maximum value[2] of 0.9, to the value in figure 1.a, which is 3.2.

In figure 1.c, it can be seen that load is distributed over a larger area than before. With experience of the visualised data, the maximum average load is estimated to be 0.15. Compared to the implementation using stress value with no averaging, the average load now achieved is six times less compared to the non averaged stress value. In relation to the most basic implementation, it is enhanced with a factor of more than 20.

It can clearly be seen from the previous discussion that the implementation of a more balanced load using stress values increases the network throughput and decreases the packet delivery time. All simulations in this paper has been made using random traffic. However, it is shown in further experiments in preliminary results that the positive effect of using PCA is also valid for a traffic model where the probability for communication between nearby Resources is higher compared to Resources on far distance.

Synthesis of the third model was made using lsi10k-technology. Optimised on size resulted in an area of 13 964 with a critical path of 79. When optimised on speed, an area of 21 029 and a gate depth of 48 was achieved.

[1] 218 gates using the lsi10k-library.

[2] Observe the scaling of the x-axis since every bar in the whole figure is scaled from the Switch with the maximum average load.

References

[1] L. Benini and G. De Micheli. *Network on Chips: A new SoC Paradigm*. Stanford University, IEEE, 2002.

[2] W. Dally and B. Towles. *Route Packets, Not Wires: On-Chip Interconnection Networks*. Design Automation Conference, IEEE, Las Vegas, USA, 2001.

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

[4] W. Dally and C. Seitz. *The Torus Routing Chip*. California Institute of Technology, 1986.

[5] W. Stallings. *Data and Computer Communications*. Prentice Hall International Editions, 5^{th} edition, 1997.

[6] E. Nilsson. *Design and Implementation of a hot-potato Switch in a Network on Chip*. Master's thesis, Royal Institute of Technology, IMIT/LECS 2002-11, Sweden, June 2002.

Included Publications
Paper 2

Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the *Nostrum* Network on Chip

Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel Jantsch

Laboratory of Electronic & Computer Systems, Royal Institute of Technology (KTH)
LECS/IMIT/KTH, Electrum 229, 164 40 Kista, Sweden
{micke, erlandn, thid, axel}@imit.kth.se

Pages: 890-895 Vol.2 Reference in Thesis – [Millberg2004a]

Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the *Nostrum* Network on Chip

Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel Jantsch

Laboratory of Electronic & Computer Systems, Royal Institute of Technology (KTH)
LECS/IMIT/KTH, Electrum 229, 164 40 Kista, Sweden
{micke, erlandn, thid, axel}@imit.kth.se

Abstract

In today's emerging Network-on-Chips, there is a need for different traffic classes with different Quality-of-Service guarantees. Within our NoC architecture Nostrum, we have implemented a service of Guaranteed Bandwidth (GB), and latency, in addition to the already existing service of Best-Effort (BE) packet delivery. The guaranteed bandwidth is accessed via Virtual Circuits (VC). The VCs are implemented using a combination of two concepts that we call 'Looped Containers' and 'Temporally Disjoint Networks'. The Looped Containers are used to guarantee access to the network – independently of the current network load without dropping packets; and the TDNs are used in order to achieve several VCs, plus ordinary BE traffic, in the network. The TDNs are a consequence of the deflective routing policy used, and gives rise to an explicit time-division-multiplexing within the network. To prove our concept an HDL implementation has been synthesised and simulated. The cost in terms of additional hardware needed, as well as additional bandwidth is very low – less than 2 percent in both cases! Simulations showed that ordinary BE traffic is practically unaffected by the VCs.

1 Introduction

Current core based *System-on-Chip (SoC)* methodologies do not offer the required amount of reuse to enable the system designer to meet the time to market constraint. A future SoC methodology should have potential of not only reusing cores but also reusing the interconnection and communication infrastructure among cores.

The need to organise a large number of cores on a chip using a standard interconnection infrastructure has been realised for quite some time. This has led to proposals for platform based designs using standardised interfaces, e.g. the VSI initiative [1]. Platforms usually contain bus based interconnection infrastructures, where a designer can create a new system by configuring and programming the cores connected to the busses. A concrete example of this is manifested in Sonic's µ-networks [2]. Due to the need for a systematic approach for designing on-chip communication Benini and Wielage [3, 4], have proposed communication centric design methodologies. They recognise the fact that interconnection and communication among cores for a SoC will captivate the major portion of the design and test effort.

As recognised by Guerrier [5], bus based platforms suffer from limited scalability and poor performance for large systems. This has led to proposals for building regular packet switched networks on chip as suggested by Dally, Sgroi, and Kumar [6, 7, 8]. These *Network-on-Chips (NoCs)* are the network based communication solution for SoCs. They allow reuse of the communication infrastructure across many products thus reducing design-and-test effort as well as time-to-market. However, if these NoCs should be useful, different traffic classes must be offered, as argued by Goossens [9]. One of the traffic classes that will be requested is the *Guaranteed Bandwidth (GB)* that has been implemented in, e.g. Philips's Æthereal [9].

Our contribution is the service of GB, to be used within our NoC architecture *Nostrum*, in addition to the already existing service of Best-Effort (BE) packet delivery [10]. *Nostrum* targets low overhead in terms of hardware and energy usage in combination with tolerance against network disturbances, e.g. congestions. In order to achieve these goals deflective routing was chosen as switching policy. In comparison to the switch of Rijpkema [11], and in Philips's Æthereal, the need for hardware is reduced by the absence of routing tables as well as in and output packet queues.

The service of GB is accessed via *Virtual Circuits (VC)*. These VCs are implemented using a combination of the two concepts called *Looped Containers* in *Temporally Disjoint Networks (TDN)*. The solution is cheap, both in terms of header information in the packets, hardware need, and bandwidth used for providing the service.

The rest of the paper is organised as follows. In section 2, we briefly describe the *Nostrum* NoC. Section 3 explains the theory behind our concept. Section 4 presents how the concept can be used and what possibilities this usage gives. The section also includes synthesis and simulation results. The last section is used for conclusions.

2 *Nostrum*

We have developed a concept that we call *Nostrum* [12] that is used for defining a concrete architecture – the *Nostrum Mesh Architecture*. The communication infrastructure used within the concept is called the *Nostrum Backbone*.

2.1. The *Nostrum* Concept

Nostrum is our concept of network based communication for 'System on Chip's (SoCs). *Nostrum* mixes traditional mapping of applications to hardware with the use of the communication infrastructure offered by Network-on-chip (NoCs). Within *Nostrum*, the '*System*' in SoC can be seen as a system of applications. An application consists of one or more processes that can be seen as functional parts of the application. In order to let these processes communicate, the *Nostrum* concept offers a packet switched communication platform and it can be reused for a large number of SoCs, since it is inherently scalable.

To make the packet switched communication practical for on-chip communication, the protocols used in traditional computer networks cannot be employed directly; the protocols need to be simplified so that the implementation cost as well as speed/throughput performance is acceptable. These simplifications are made from a functional point of view and only a limited set of functions are realised.

2.2. The *Nostrum* Backbone

The purpose of the backbone is to provide a reliable communication infrastructure, where the designer can explore and chose from a set of implementations with different levels of reliability, complexity of service etc.

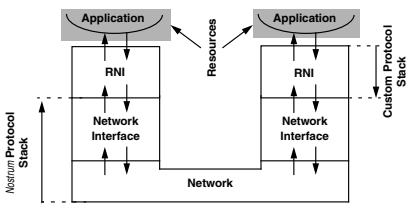


Fig. 1. The Application/RNI/NI

In order to make the resources communicate over the network, every resource is equipped with a *Network Interface (NI)*. The NI provides a standard set of services, defined within the *Nostrum* concept, which can be utilised by a *Resource Network Interface (RNI)* or by the resource directly. The role of the RNI is to act as glue (or adaptor) between the resource's internal communication infrastructure and the standard set of services of the NI. Dependent on the functionality requested from the *Nostrum Backbone,* the *Nostrum* protocol stack can be more or less shallow. However, the depth of the custom protocol stack, which may include the RNI, is not specified within the concept.

2.3. Communication Services

The backbone has been developed with a set of different communication protocols in mind e.g MPI [16]. Consequently, the backbone can be used for both BE using single-message passing between resources (datagram based communication) as well as for GB using stream oriented data distribution (VC). The message passing between the resources is packet based, i.e. the message is segmented and put into packets that are sent over the network. The ordering of packets and de-segmentation of messages is handled by the NI. In order to cover the different needs of communication two different policies are implemented:

A. Best-Effort

In the BE implementation, the packet transmission is handled by datagrams. The switching decisions are made locally in the switches on a dynamic/non-deterministic basis for every individual datagram that is routed through the network. The benefit is low set-up time for transmission and robustness against network link congestion and failure. The policy is described in [10] and will not further be discussed.

B. Guaranteed Bandwidth

The GB is the main topic of the paper and is implemented by using a packet type, which we call container. A container packet differs from the datagram packets in two ways. They follow a pre-defined route and they can be flagged as empty.

2.4. The *Nostrum* Mesh Architecture

The NoC *Nostrum* Mesh Architecture [13] is a concrete instance of the *Nostrum* concept and consists of *Resources (R)* and *Switches (S)* organised, logically and physically in

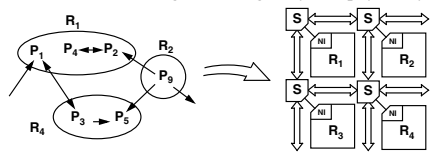


Fig. 2. *Nostrum* Process to Resource mapping

a structure where each switch is connected to its four switch neighbours and to its corresponding resource as depicted in Figure 2. From an implementation point of view, the resources (Processor cores, DSPs, Memories, I/O etc.) are the realisation of the *Processes (P)*. A resource can host single or multiple processes, potentially the processes can belong to one or several different applications. However, the *Nostrum* Concept is not inherently dependent of the

mesh topology, other possibilities might include folded torus, fat-trees [14] etc. The reason why the mesh topology was chosen stems from reasons of three types.

First, higher order dimension topologies are hard to implement. As analysed by Culler [15], low dimension topologies are favoured when wiring and interconnects carry a significant cost, there is a high bandwidth between switches, and the delay caused by switches is comparable to the inter-switch delay. This is the case for VLSI implementations on the 2-dimensional surface of a chip and practically rules out higher dimension topologies. The torus topology was rejected in favour of a mesh since the folded torus has longer inter-switch delays.

Second, there is no real need for higher order dimension topologies. We assume that all applications we have in mind, e.g. telecom equipment and terminals, multi-media devices, and consumer electronics etc. exhibit a high degree of locality in the communication pattern. This is in stark contrast to the objective of traditional parallel computers; designed to minimise latency for arbitrary communication patterns.

Third, the mesh inhibits some desirable properties of its own, such as a very simple addressing scheme and multiple source-destination routes, which give robustness against network disturbances.

3 Theory of Operation

The switching of packets in *Nostrum* is based on the concept of deflective routing [17], which implies no explicit use of queues where packets can get reordered, i.e. packets will leave a switch in the same order that they entered it. This is possible since the packet duration is only one clock cycle, i.e. the length of packets is one *flit*. This means that packets entering a switch at the same clock cycle will suffer the same delay caused by switching and therefore leave the switch simultaneously. However, if datagram packets are transmitted over the network they may arrive in another order than they were sent in; since they can take different routes, this can result in different path lengths. The reason for packets taking different routes is that *the switching decision is made locally in the switches on a dynamic basis for every individual datagram that is routed through the network* – as stated earlier.

3.1. The Temporally Disjoint Networks

The deflective routing policy's non-reordering of packets creates an implicit time division multiplexing in the network. The result is called Temporally Disjoint Networks (TDNs). The reasons for getting these TDNs are *The Topology* of the network and *The Number of Buffer Stages* in the switches.

A. The Topology

Packets emitted on the same clock cycle can only collide, i.e. will only be 'in the same net', if they are on a multiple distance of the smallest round-trip delay. Intuitively this can be explained by colouring the nodes so that every second node is black and every second is white. Since all the white nodes are only connected to black nodes and all the black nodes are only connected to white nodes, any packet routed on the network will visit black and white nodes interchangeably. Naturally, this means that two packets residing in nodes of different colour, at a point in time, will never meet! That is, these two packets will never affect each others switching decisions. This is illustrated in Figure 3 (A); the network of a 4x4 mesh is unfolded and displayed as a bipartite graph in (B) where the left-side nodes only have contact with the right-side nodes and the opposite ditto. Please note that all the edges are bidirectional.

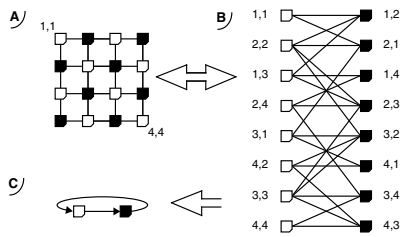


Fig. 3. Disjoint networks due to topology

This bipartite graph can further be collapsed into the lower left graph (C) of Figure 3 where all the black and white nodes are collapsed into one node respectively and the edges now are unidirectional. Logically packets residing in neighbouring time/space-slots could be seen as being in different networks, i.e. in Temporally Disjoint Networks. The contribution to the number of TDNs that stems from the topology is called the *Topology Factor.*

B. The number of buffer stages in the switches.

In the previous case where the topology gave rise to two disjoint nets, implicit buffering in the switches was assumed, i.e. a switching decision was taken every clock cycle. If more than one buffer is used in the switches, e.g. input and output buffering is used, this also creates a set of TDNs. In Figure 4, this is illustrated by taking the graph of

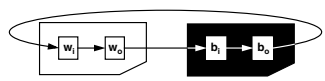


Fig. 4. Disjoint networks due to buffer stages in switches

Figure 3 (C) and equip it with buffers. The result is that every packet, routed on the network, must visit buffers in the following order: white input (w_i) -> white output (w_o) -

> black input (b_i) -> black output (b_o), before the cycle repeats. The result is a smallest round-trip delay of four clock cycles and hence four TDNs exist; where both the Topology Factor and the Buffer Stages contributes with a factor of two each. So in general

$$\text{TDN} = \text{Topology Factor} \times \text{Buffer Stages}$$

A clever policy when dealing with these multiple disjoint networks will give the user the option of implementing different priorities, traffic types, load conditions etc. in the different TDNs.

3.2. The Looped Container Virtual Circuit

Our Virtual Circuit is based on a concept that we call the *Looped Container*. The reason for this approach is that we must be able to guarantee bandwidth and latency for any VC that is set up. The idea is that a GB is created by having information loaded in a container packet that is looped between the source and the destination resource. The reason for this approach is the fact that it is very hard to guarantee admittance to the network at a given point in time as we shall see. This stems from two chosen policies

- Packets already out on the network have precedence over packets that are waiting to be launched out on the network.
- At a certain point in time the difference in the number of packets entering a switch, and the packets coming out after being switched, is always zero; that is, packets are neither created, stored, nor destroyed in the switches.

In Figure 5 (A), the consequence of these two policies is illustrated. The packet that wants to get out on the network never gets the chance since all the outgoing links are occupied. The switching policy, illustrated in Figure 5 (A), of letting the incoming packets be deflected, instead of properly routed, is not sufficiently for a proper network operation; but the sum of incoming/outgoing packets are the same, i.e. a deflected packet is occupying the same number of outputs as a packet routed to any other output!

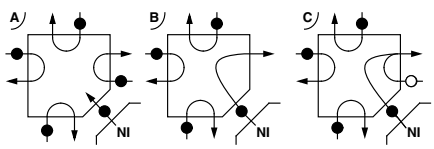


Fig. 5. Launching a packet out on the network

In Figure 5 (B), one link is unoccupied and the packet can therefore immediately get access to the network.

In Figure 5 (C), the principle behind our VC using containers as information carriers is illustrated. One 'empty' container arrives from the east, information from the resource is loaded, and the container is sent away.

In order to further illustrate the principle, Figure 6 depicts a VC going from the Source (1) to the Destination (3); a container belonging to this VC is tracked during four clock cycles. It is, in this example, assumed that the container already exists. In the first clock cycle, the container

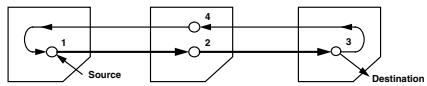


Fig. 6. The looped container

arrives to the switch connected to the Source. The container is loaded with information and sent off to the east. The reason why the information could be loaded instantly was that the container already was there and occupied one of the inputs. As a result of this, it is known that there will be an output available the following clock cycle.

In the second clock cycle, the container and its load is routed along its predefined path with precedence over the ordinary datagram packets originating from the BE traffic.

In the third cycle, the container reaches its destination, the information is unloaded and the container is sent back. Possibly with some new information loaded, but now with the original source as destination.

The fourth cycle is similar to the second.

3.3. Bandwidth Granularity of the Virtual Circuit

If the Looped Container and the Temporally Disjoint Networks (TDN) approaches are combined, we get a system where a limited set of VCs can share the same link. The number of simultaneous VCs, routed over a certain switch, is equal to the number of TDNs. This means that on-chip we can have many VCs, but only a limited set of VCs can be routed over the same switch – this since only one VC can subscribe to the same TDN on a switch output. To illustrate

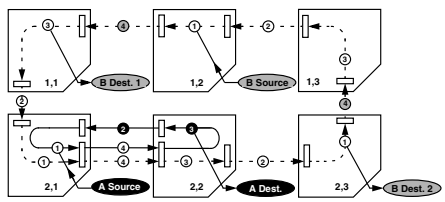


Fig. 7. BW granularity example

the concept, Figure 7 depicts two VCs; VC_A with black container packets and VC_B (path dashed) with grey ditto. In switch [2,1] and [2,2] the containers of both VCs will share the same links (and switch). The numbers inscribed in the packets, denotes which TDN the respective packet belong to; the numbers range from one to four since the number of TDNs, in Figure 7, is four since we have a bipartite topology and two buffer stages in every switch. As seen in Figure 7, VC_A have subscribed to TDN_2 and TDN_3, whereas VC_A only uses TDN_4.

The smallest bandwidth, the $BW_{Granularity}$, that is possible to acquire, for any VC, is dependent on the $VC_{Round\text{-}trip\ delay}$. The $VC_{Round\text{-}trip\ delay}$ is the length of the circular VC path in terms of buffers. In VC_A the $VC_{Round\text{-}trip\ delay}$ is four and in VC_B twelve. The $VC_{Round\text{-}trip\ delay}$ is the same as the number of containers a VC can have in all existing TDNs. Since the containers represent a fraction of the maximum BW over one link, the $BW_{Granularity}$ becomes

$$BW_{Granularity} = \frac{BW_{Max}}{VC_{Round\text{-}trip\ delay}}$$

The BW_{Max} is the switching frequency times the payload in the system, usually in terms of Gbit/s. The BW_{Max} that exist within one TDN is

$$BW_{Max(TDN)} = \frac{BW_{Max}}{TDN}$$

Of course several containers can be launched on a network if more than the initial $BW_{Granularity}$ is desired. The $BW_{Aquired}$ then naturally becomes

$$BW_{Acquired} = Container \times BW_{Granularity}$$

If the VC only subscribe to one TDN, the total number of containers is limited to

$$Container \leq \frac{VC_{Round\text{-}trip\ delay}}{TDN}$$

Regarding the individual characteristics of VC_A and VC_B they are presented in Table 1.

	VC_A	VC_B
$BW_{Granularity}$ of BW_{Max}	1/4	1/12
Launched containers	2	2
Used TDNs	2	1
$BW_{Aquired}$ of BW_{Max}	1/2	1/6

Table 1. Summary of VC characteristics

4 Use of Concept

Accessing the VC is done from the NI. The set up of VCs is, in the current implementation, semi-static, this means that the route for the respective VC is decided at design time but the numbers of containers used by every VC is variable. That is – the bandwidth, for the different VCs, can be configured at start-up of the network. To set up the VC, i.e. to get the containers in the loop, the containers are launched during a start-up phase of the network where no ordinary datagram packets are allowed to enter the network. If more bandwidth is needed during run time, this can be achieved by launching more containers. However, in this case the set-up time can not be guaranteed since "new" container packets are not guaranteed access to the network. Naturally, if less bandwidth is needed some containers can be taken out of the loop.

Since the set-up of the VCs is based on a mutual agreement between the source and the destination regarding the information to be sent, no buffer overflow is assumed. That is, the source knows at what rate it can send data/packets to the NI and the destination knows what data rate it has to be able to cope with. If several applications reside in the same resource and need to be able to acquire bandwidth this could be handled by setting up several Virtual *Channels* residing in the same Virtual *Circuit*.

4.1. Multi-cast and other functionality

By the use of VC, several services, except for the obvious sending of data from a source to a destination at a guaranteed rate, can be implemented.

Multi-cast can easily be implemented by having multiple destinations along the VC path, as illustrated by VC_B of Figure 7, which has destinations in [1,1] and [2,3]. Even several source/destination pairs can be formed along a VC path subscribed to the same TDN as long as they are aligned so that the source is followed by the destination.

Even busses might be implemented quite effectively using the service of multi-cast. The sheer distribution of data is not of any problem but what might become a bottleneck is the bus master implementation. The delay/latency caused by the VC itself may reduce the bus master's capability of granting/denying access to the bus due to latency. However, if latency is acceptable, nothing hinders an effective implementation of a bus structure.

4.2. Implementation

All services possible to implement using the VC container based concept, e.g. source – destination data distribution, multi-cast, or busses, utilises a combination of four standard switch functions

- **Source** Loads an incoming container with data from the appropriate NI output queue. Flags the packet as non-empty. Sends the container along the VC path
- **Destination (Final)** Read the data from the container and put it in the appropriate NI input queue. Flags the packet as empty. Sends the container along the VC path
- **Destination (Multi-cast)** Same as Destination (Final) but the container is not flagged as empty
- **Bypass** Sends the container along the VC path

Internally the VC path is handled by a small look-up table for every VC in the switch. In the current implementation, the VCs are set up semi-statically and the only extra HW needed in the switches is the one of giving a container packet the highest priority in the direction of its VC path. Also extra HW is needed to set/clear the empty bit dependent on the role of the switch (Source, Multi-cast Destination etc.) and whether to load/unload information. A switch with only BE functionality uses 13695 equivalent NAND gates for combinatorial logic (control), buffers excluded; for the same switch with the added functionality of VCs the

gate count is 13896. So the relative extra HW cost is less than 2 percent! The number of gates is derived from Synopsys Design Compiler.

The additional cost, for implementing the VCs, in terms of bandwidth is very low; only two bits are used as packet header. The first bit identifies the packet as a container and the second flags the packet/container as empty or not. This means that the effective relative payload for a packet with 128 bits is more than 98 percent!

4.3. Simulation Results

Simulations carried out so far extend to HDL simulations with artificial, but relevant, workload models. The workload models used, implements a two-way process communication between A and B. In the first example AB uses BE for communication and in the second the VCs of the GB are employed. In both cases, the communication is disturbed by having random BE traffic in the rest of the network. As a vehicle for the simulation a 4x4 network was chosen. The processes were placed so that A got position [3,1] and B [2,4] in the 4x4 mesh. Both the background traffic as well as the traffic between A and B was created with the same probability, p. In the simulation p ranges from [0 .6], above that the network becomes congested due to fundamental limitations in capacity of the network.

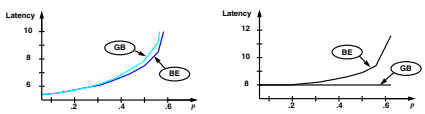


Fig. 8. The background traffic and the AB traffic

In Figure 8 the average latency is plotted against the probability of the packet generation, p. The left graph shows the background traffic and the right graph the AB traffic. BE and GB in the figure relates the respective graph to the traffic pattern used for AB traffic in the simulation.

As seen in Figure 8, the random background traffic in the network is very little affected by the VC; but for the AB traffic, the VC gives a tremendous boost in guaranteed latency and bandwidth for increased traffic in the network. The average bandwidth of the AB traffic is not changed – but now it is guaranteed! and as expected, the latency of AB goes from being exponential to become constant.

Of course, if more VCs were utilised, it would be theoretically possible to construct such traffic patterns and VC route mapping combinations so that network congestions are irreparable, but we found no interest in these artificial corner cases.

5 Conclusions

We have implemented a service of guaranteed bandwidth to be used in our NoC platform *Nostrum*. The GB uses Virtual Circuits to implement the two concepts that we call *Looped Containers* in *Temporally Disjoint Networks*. The VCs are set up semi-statically, i.e. the route is decided at design time, but the bandwidth is variable in run-time. The implementation of the concept was synthesised and simulated. The additional cost in HW, compared to the already existing BE traffic implementation and the cost in terms of header information were both less than 2 percent.

Simulations showed that the VCs did not affect BE traffic in the network significantly but gave a guaranteed bandwidth and a constant latency to the user of the GB. Also the cost of setting up the VC was very low.

Possible drawbacks are the potential waste of bandwidth in the returning phase of the container in the loop, since the container might travel empty if the BE traffic is one-way. Also, the limited granularity of bandwidth possible to subscribe to, might become a problem. Future work includes a method for clever traffic planning to avoid the possible waste of bandwidth when the VCs are set up.

REFERENCES

[1] Virtual Socket Interface Alliance, http://www.vsi.org
[2] Sonics Inc., http://www.sonicsinc.com
[3] L. Benini and G. DeMicheli, Networks on chip: A New SoC Paradigm. IEEE Computer, 35(1): p. 70 ff., January. 2002.
[4] P. Wielage and K. Goossens, Networks on Silicon: Blessing or Nightmare?. In Proc. of Euromicro Symposium on Digital System Design. Architectures, Methods and Tools, p 196-200, 2002
[5] P. Guerrier and A. Greiner, A Generic Architecture for On-Chip Packet-Switched Interconnections. In Proc. of DATE 2000, March 2000.
[6] W. J. Dally and B. Towles, Route packets, not Wires: On-Chip Inter-Connection Networks. In Proc. of DAC 2001, June 2001.
[7] M. Sgroi et al., Addressing the System-on-a-Chip Interconnect Woes through Communication-Based design. In Proc. of DAC 2001, June 2001.
[8] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, K.Tiensyrjä, and A. Hemani, A Network-on-Chip Architecture and Design Methodology. In Proc. of IEEE Comp. Society, April 2002.
[9] K. Goossens et al., Networks on Silicon: Combining Best-Effort and Guaranteed Services, DATE 2002, March 2002.
[10] E. Nilsson, M. Millberg, J. Öberg, and A. Jantsch, Load Distribution with Proximity Congestion Awareness in a NoC, DATE 2003
[11] Trade-offs in the Design of a Router with Both Guaranteed and Best-Effort Services for NoC. E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, E. Waterlander, DATE 2003.
[12] M. Millberg, The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone, Technical Report TRITA-IMIT-LECS R 02:01, LECS, IMIT, KTH, Stockholm, Sweden, 2003.
[13] M. Millberg, E. Nilsson R. Thid and A. Jantsch, The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip, In Proc. of VLSI Design India, January 2004
[14] C. E. Leiserson, Fat Trees: Univ. Networks for Hardware Efficient Supercomputing. IEEE Computer, p 892 ff. vol. c-34, No 10, Oct. 1985.
[15] D. E. Culler and J. P. Singh, “Parallel Computer Architecture - a Hardware Software Approach”, Morgan Kaufmann Publishers, Inc., ISBN 1-55860-343-3, 1999
[16] A Message Passing Interface Standard. http://www.mpi-forum.org
[17] U. Feige, P. Raghavan, Exact Analysis of Hot-Potato Routing. In Proc. of Foundations of Computer Science, p. 553 -562, 1992

Included Publications
Paper 3

The *Nostrum* Backbone - a Communication Protocol Stack for Networks on Chip

Mikael Millberg, Erland Nilsson, Rikard Thid, *Shashi Kumar, and Axel Jantsch

Laboratory of Electronic & Computer Systems
Royal Institute of Technology (KTH)
LECS/IMIT/KTH, Electrum 229, 164 40 Kista, Sweden
{micke, erlandn, thid, axel}@imit.kth.se

**SchoolofEngineering*
Jönköping University, Sweden
Shashi.Kumar@ing.hj.se

THE SEVENTEENTH INTERNATIONAL CONFERENCE ON VLSI DESIGN
JANUARY 5-9, 2004, MUMBAI, INDIA

Pages: pp. 693-696 Reference in Thesis – [Millberg2004b]

The *Nostrum* Backbone - a Communication Protocol Stack for Networks on Chip

Mikael Millberg, Erland Nilsson, Rikard Thid, *Shashi Kumar, and Axel Jantsch

Laboratory of Electronic & Computer Systems
Royal Institute of Technology (KTH)
LECS/IMIT/KTH, Electrum 229, 164 40 Kista, Sweden
{micke, erlandn, thid, axel}@imit.kth.se

**SchoolofEngineering*
Jönköping University, Sweden
Shashi.Kumar@ing.hj.se

Abstract

We propose a communication protocol stack to be used in Nostrum, our Network on Chip (NoC) architecture. In order to aid the designer in the selection process of what parts of protocols, and their respective facilities, to include, a layered approach to communication is taken. A nomenclature for describing the individual layers' interfaces and service definitions of the layers in the protocol stack is suggested and used. The concept includes support for best effort traffic packet delivery as well as support for guaranteed bandwidth traffic, using virtual circuits. Furthermore an application to NoC adapter is defined, as part of the Resource to Network Interface, and is used to communicate between the Nostrum protocol stack and the application. An industrial example has been implemented, simulated, and the results justifies the suggested layered approach.

1 Introduction

Current core based *System-on-Chip (SoC)* methodologies do not offer the required amount of reuse to enable the system designer to meet the time to market constraint. A future SoC methodology should have potential of not only reusing cores but also reusing the interconnection and communication infrastructure among cores.

The need to organise a large number of cores on a chip using a standard interconnection infrastructure has been realised for quite some time. This has led to proposals for platform based designs using standardised interfaces, e.g. the VSI initiative [1]. Platforms usually contain bus based interconnection infrastructures, where a designer can create a new system by configuring and programming the cores connected to the busses. Due to the need for a systematic approach for designing on-chip communication communication centric design methodologies have been proposed in [2, 3] where interconnection and communication among cores for a SoC will captivate the major portion of the design and test effort.

As recognised in [4], bus based platforms suffer from limited scalability and poor performance for large systems. This has led to proposals for building regular packet switched networks on chip as proposed in [5, 6, 7]. These *Network-on-Chips (NoCs)* are the network based communication solution for SoCs. They allow reuse of the communication infrastructure across many products thus reducing design-and-test effort and time-to-market.

In order to enhance reusability and to ease programmability, all NoC proposals recommend standardised and layered protocols for communication among cores. While some of these papers discuss the protocol layering on a conceptual and abstract level [2, 6, 8] others have only elaborated and implemented the lower protocol layers [5, 12].

In this paper, a concrete instance of a full communication protocol stack for *Nostrum*, our NoC architecture, is proposed. The protocol stack ranges from physical to transport layer and offers the designer the possibility to customise the, respective, layers' functionality with respect to the actual needs of the application. One important aspect of our approach is the strong connection to geometry and implementation, which will help us to find efficient solutions.

2 Nostrum

2.1. The *Nostrum* Concept

We have developed a concept called *Nostrum* [13] that is used for defining a concrete architecture – the *Nostrum Mesh Architecture*. The communication infrastructure used within the concept is called the *Nostrum Backbone*. *Nostrum* mixes traditional mapping of applications to hardware with the use of the communication infrastructure offered by Network-on-chip (NoCs). Within *Nostrum*, the '*System*' in SoC can be seen as a system of applications consisting of one or more processes. In order to let processes communicate, *Nostrum* offers a packet switched communication platform that can be reused for a large number of SoCs.

To make the packet switched communication practical for on-chip communication, the protocols used in traditional computer networks cannot be used directly; they need to be simplified so that the implementation cost as well as speed/throughput performance is acceptable. These simplifications are made from a functional point of view and only a limited set of functions are realised.

2.2. The *Nostrum* Backbone

The purpose of the backbone is to provide a reliable communication infrastructure, where the designer can explore and chose from a set of implementations with different levels of reliability, complexity of service etc.

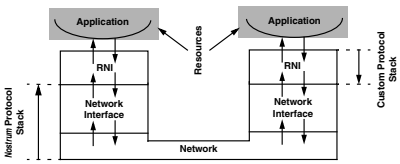


Figure 1. The Application/RNI/NI

In order to make the resources communicate over the network, resources are equipped with a *Network Interface (NI)*. The NI provides a standard set of services that can be utilised by the *Resource Network Interface (RNI)* or by the resource directly. The role of the RNI is to act as glue (or adaptor) between the resource's internal communication infrastructure and the standard set of services of the NI. Dependent on the functionality requested from the *Nostrum backbone,* the *Nostrum* protocol stack can be more or less shallow, but the depth of the custom protocol stack, which may include the RNI, is not specified within *Nostrum*.

The backbone can be used for both best-effort traffic using single-message passing between resources where switching decisions are made locally in the switches on a dynamic basis for every individual datagram routed through the network, as well as for guaranteed bandwidth traffic using virtual circuits.

2.3. The *Nostrum* Mesh Architecture

The NoC *Nostrum* Mesh Architecture (described in [14]) is a concrete instance of the *Nostrum* concept and consists of *Resources (R)* and *Switches (S)* organised, logically and physically in a structure where each switch is connected to

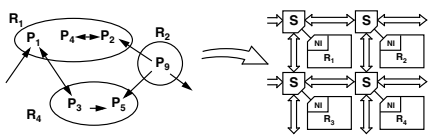


Figure 2. *Nostrum* Process to Resource mapping

its switch neighbours and to its resource as depicted in Figure 2. From an implementation point of view, the resources (Processor cores, DSPs, Memories, I/O etc.) are the realisation of the *Processes (P)*. A resource can host one or many processes, potentially the processes can belong to one or several different applications. However, the *Nostrum* concept is not inherently dependent on the mesh topology, other possibilities include folded torus, fat-trees [9] etc.

The mesh topology was chosen for three reasons:

First, higher order dimension topologies are hard to implement. As analysed in [10], low dimension topologies are favoured when wiring and interconnects carry a significant cost, there is a high bandwidth between switches, and the delay caused by switches is comparable to the inter-switch delay. The torus topology was rejected in favour of a mesh since the folded torus has twice as long inter-switch delays.

Second there is no real need for higher dimensional topologies since it has been shown that two the dimensional mesh topology is quite efficient for a large number of important applications in the area of signal and multimedia processing.

Third, the mesh inhibits some desirable properties of its own, such as a very simple addressing scheme and multiple source-destination routes, which give robustness against network disturbances.

3 The Layered Communication Approach

In order to make different processes communicate a standard needs to be defined for the format of communication. The requirements are quite diverse; the process communication mechanism needs to be able to deal with different data formats, different priorities, the interaction with the underlying network etc.

In order to implement this functionality a vertical set of layers has been defined, with each layer providing the layer above and below with a set of services. This layered approach to communication enhances to great extent the possibilities of performing simulations and implementations of the different layers in a multitude of design languages. It also alleviates changes in the protocol stack.

3.1. Terminology

In order to describe the functionality of the different layers we use a terminology where similar functionalities are grouped into layers and a service is defined as an agreed functionality, which the particular layer has to provide. The services a layer offers could further be divided into functions implemented in that particular layer. The functions within a layer are collected into groups called entities. These entities, within the same layer communicate

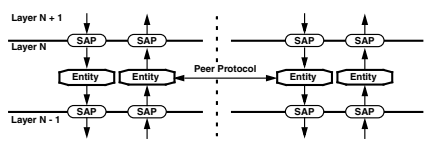


Figure 3. Layer Definitions

with their peers using one or more peer protocols as seen in Figure 3. These units of information used to implement the protocol are called Protocol Data Units (PDUs), i.e. the PDU is the agreed data format, used by peers. In order to provide these services, an interface to the upper and lower layer has to be defined. These interfaces are called Service Access Points (SAPs).

4 The *Nostrum* Protocol Stack

As a starting point for the layers employed in the *Nostrum* backbone protocol stack, the abstraction levels of the OSI reference model is used as a basis. However, these layers only exist as a conceptual aid in the design process. Once the design is set, the layers might be collapsed at a logic level in order to enhance the possibility of hardware implementation optimisations. Within the concept of the *Nostrum* backbone resides the three "compulsory" layers; Network, Data Link, and Physical Layer. These three layers provide the service of delivering packets with a Destination Process Identifier (DPID) as the destination address.

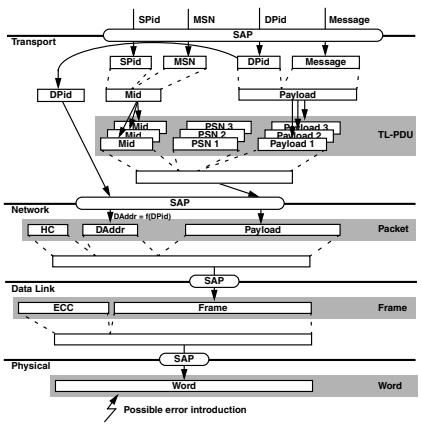


Figure 4. The *Nostrum* Down-Stream Protocol Stack

4.1. Physical Layer

The lowest layer is the Physical Layer with the purpose of moving a word from the output of a switch to the corresponding input of the next. From a simulation point of view, this gives the possibility of error introduction.

4.2. Data Link Layer

The Data Link Layer (LL) provides the reliable transfer of information across the physical link. The LL is responsible for the transmission of frames with the necessary synchronisation, error control, and flow control.

4.3. Network Layer

The Network Layer's (NL) responsibility is to provide the delivery of packets from the transmitting resource's NI through the network to the receiving resource ditto and the intermediate packet routing needed. The NL also has the responsibility of managing and mapping the resource addresses to the process identifiers associated with the processes in the different resources. This management involves the mapping/re-mapping of the process-resource pair, either at system set-up or during run-time.

At the NL two different services can be requested. The first service is the datagram service delivery; any packet sent to the NL will get routed dynamically through the network using a variant of deflection routing [11, 14].

The second service is the Virtual Circuit (VC) service. On request, a VC between the transmitter and the receiver is set up. This VC is implemented as a modified TDMA where the VC can allocate fractions of the total bandwidth.

4.4. Transport Layer

The Transport Layer (TL) handles the establishment of communication. In the case of VC it issues an establishment of a connection and handles the transfer of data by ensuring a virtual point-to-point connection. It also provides traffic control, i.e. the load of the network is detected and overload of the network is hence avoided.

5 The *Nostrum* Simulator

A simulator has been developed in order to prove the *Nostrum* concept. The simulator implements the different layers of the protocol stack "independently", i.e. different layers are implemented as separate plug-ins. The choice of the plug-ins is a part of the design process. Within one design, several different plug-ins can be used for implementing the same layer dependent on the communication requirements. To test the *Nostrum* simulator a relatively large industrial example in the form of a distributed DSP core was employed. The purpose of a real-life example is twofold; it justifies *Nostrum* as a concept and it justifies the use of the simulator as a platform to demonstrate the working of our ideas.In the current simulation set-up only the Application and the Network Layer (NL) were fully simulated. Due to this, packets are passed directly to the NL, ignoring the

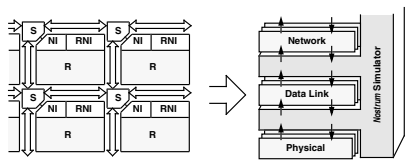


Figure 5. The Simulated *Nostrum* Protocol Stack

intermediate layers. In a full simulation, the diversity in message sizes would force us to split and merge the messages in the TL.

5.1. The Distributed DSP Application Model

Ericsson Radio Systems provided a typical application from the real world. It is a relatively large industrial example in the form of a distributed DSP application.

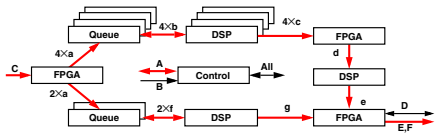


Figure 6. The Distributed DSP Application

The application has 12 serial inputs and 12 serial outputs. The input streams are de-serialised, characterised and sent to the appropriate *Queue*. After these respective data streams have been processed by FPGAs and DSPs they are re-serialised and sent out on one of the outgoing channels. The *Control Processor*, which loads all the system components with data at start-up hosts the monitor and control facilities, after start-up the traffic that it generates is negligible. The characteristics of the messages between the FPGAs and DSPs are shown in the Table 1.

Table 1. Channel Characteristics

Type	Bit- width	Rate (MHz)	Class
a	128	32	Continuous
b, c	16	32	Burst
d	64	32	Burst
e	32	16	Burst
f	16	8	Burst
g	16	4	Burst

5.2. Conclusions drawn from Simulation

The simulations performed show that:

- The protocol stack fully covers the need of the application when simulating different traffic classes.
- The simulator correctly implements the protocol stack.
- We can simulate fairly complex applications on a very high abstraction level, which only captures the traffic pattern between processes.

In addition to the experiment with the protocol stack and the validation of the simulator, several conclusions about requirements imposed by applications on the *Nostrum.* could be drawn. The two most important are: First, the traffic is indeed highly local, provided that we have a sensible mapping of tasks into resources. Second, multi-cast messages have the potential of significantly decreasing the communication load and improve performance, because, data streams are often fed to several consumers which process data in parallel. Since these streams constitute high traffic loads it is beneficial to split the streams into multiple copies as close to the consumers as possible.

6 Conclusions

A standard protocol stack for a Network-on-Chip platform allows the separate, independent development and validation of resources, communication network and applications as long as they comply with the defined protocols. This clear separation enables systematic reuse of resources, communication infrastructure and application features.

We have defined such a protocol stack within the concept of *Nostrum*. The protocol stack is defined from the physical to the transport layer and, based on it, the layered *Nostrum* simulator is developed. The simulator allows experiments with different protocol variants because individual layers can be replaced without affecting other parts. Furthermore, a real application have been modelled, and simulated, on a very high level of abstraction only capturing the traffic pattern between tasks and resources. This enables analysis of application requirements in order to optimise the protocol stack and the *Nostrum* architecture. Also the assumption of the feasibility of using a 2D-mesh as topology for a complex real life example holds!

REFERENCES

[1] Virtual Socket Interface Alliance, http://www.vsi.org
[2] L. Benini and G. DeMicheli, Networks on Chip: A New SoC Paradigm, IEEE Computer, 35(1): p. 70 ff., Jan. 2002.
[3] P. Wielage and K. Goossens, Networks on Silicon: Blessing or Nightmare?, In Proc. of Euromicro Symposium on Digital System Design, Architectures, Methods and Tools, p 196-200, 2002.
[4] P. Guerrier and A. Greiner, A Generic Architecture for On-Chip Packet-Switched Interconnections, In Proc. of DATE 2000, March 2000.
[5] W. J. Dally and B. Towles, Route Packets, Not wires: On-chip Interconnection Networks, In Proc. of DAC 2001, June 2001.
[6] M. Sgroi et al., Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design, In Proc. DAC 2001, June 2001.
[7] S. Kumar et al., A network on Chip Architecture and Design Methodology, IEEE Computer Society, Annual Symposium on VLSI, April 2002.
[8] K. Goossens et al., Networks on Silicon: Combining Best-Effort and Guaranteed Services, In Proc. of DATE 2002, March 2002.
[9] C.E. Leiserson, Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing, IEEE Computer, p. 892 ff. vol c-34, No 10, Oct. 1985.
[10] D. E. Culler and J. P. Singh, Parallel Computer Architecture - A Hardware Software Approach, Morgan Kaufmann Publishers, Inc., ISBN 1-55860-343-3, 1999.
[11] U. Feige and P. Raghavan, Exact Analysis of Hot-Potato Routing, In Proc. of Foundations of Computer Science, p. 553 -562, 1992.
[12] D. Wingard, MicroNetwork-Based Integration of SoCs, In Proc. of DAC 2001, June 2001.
[13] M. Millberg, The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone, Technical Report TRITA-IMIT-LECS R 02:01, LECS, Dept. of IMIT, KTH, Stockholm, Sweden, 2003.
[14] E. Nilsson et al., Load distribution with the Proximity Congestion Awareness in a NoC, In Proc. of DATE 2003, March 2003.

Included Publications
Paper 4

A study of NoC Exit Strategies

Mikael Millberg & Axel Jantsch
KTH - Royal Institute of Technology, Sweden
{micke, axel}@imit.kth.se

Pages: 217-217 Reference in Thesis – [Millberg2007a]

A study of NoC Exit Strategies

Mikael Millberg & Axel Jantsch
KTH - Royal Institute of Technology, Sweden
{micke, axel}@imit.kth.se

The throughput of a network is limited due to several interacting components. Analysing simulation results made it clear that the component that was worth attacking was the exit bandwidth between the network and the connected resources. The obvious approach is to increase this bandwidth; the benefit is a higher throughput of the network and a significant lowering of the buffer requirements at the entry points of the network; this because worst case scenarios now happens at a higher injection rate. The result we present shows significant differences in throughput as well as in average and worst case latency.

Offering services with best effort performance, naturally, gives no hard guarantees due to the dynamic behaviour of any general purpose system. In order to make use of such services statistical performance measures are instead utilised. This leads to that the traffic often has to be below a certain threshold for which the desired statistical properties can be given. These properties can be derived, either, from a rigid reasoning based on the implementation or from an analysis of simulations. From this analysis the desired properties can be given with a safety margin.

Given that we are bound to offer services with statistical characteristics on performance, how do we do this to a low cost? The cost in this context is the required buffers needed to guarantee no packet losses together with a safety margin in terms of injection rate to "guarantee" a worst, and average, case latency.

The approach that we have chosen within, Nostrum [1], for giving the service of *Best effort*, is by utilising deflective routing, with no explicit buffering, to keep the size of the switches small[2]. Through simulations with uniform random traffic patterns, we observed that there seems to exist a "hard" limitation on the network. On a 4x4 mesh this limit is reached for an injection rate of 0.63 packets/node/cycle for the best performing routing strategy tried out. Once this limit is reached packets start queuing up at the entry points of the network and the worst case latencies grows exponentially.

Analysing simulation results made it obvious that an increased bandwidth between the network and the connected resources would make the network perform better. The price for this solution is that packets now need to be buffered at the exits of the network, but from an overall perspective the total buffers required is lowered for a moderately to heavily loaded network. At the limit injection rate the average latency was reduced from 25 to 15 clock cycles and the observed worst case latency from 280 to 180 clock cycles. This will give better margins before the network saturates or a higher throughput with the previous margin kept. All this assumes that there no single node or bisection cut of the network are exposed to a static over-utilisation.

The validity of the chosen approach is not restricted to uniformly random traffic patterns on meshes but also applicable to "any" topology where the traffic pattern involves potential network exit congestions due to multiple sources having the same destination and/or multiple routing paths are possible.

The network exit strategy has not received as much attention as other parts of network design. Most work that in detail analyse cost and performance of a router and the network as a whole, e.g. [3, 4, 5] assume an ideal packet ejection model, which means that packets are absorbed by the receiving node as soon as they are delivered by the network. In [6] an ejection policy is studied that reduces the cost and complexity of the router while minimizing the impact on performance. However, to our knowledge no study about the trade-offs involved in increasing the network exit bandwidth has been reported.

[1] M Millberg et al. The Nostrum backbone - a communication protocol stack for networks on chip. VLSI Design Conference, January 2004.
[2] E. Nilsson et al. Load distribution with the proximity congestion awareness in a network on chip. DATE 2003.
[3] 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.
[4] L. S. Peh and W. J. Dally. A delay model for router micro-architectures. IEEE Micro, 2001.
[5] E. Rijpkema, et al. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. DATE, 2003.
[6] Z. Lu and A. Jantsch. Flit ejection in on-chip wormhole-switched networks with virtual channels. NorChip, 2004.

Included Publications

Paper 5

Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy

Mikael Millberg & Axel Jantsch
ECS - KTH - Royal Institute of Technology, Sweden
{micke, axel}@imit.kth.se

DSD 2007
10th EUROMICRO CONFERENCE on DIGITAL SYSTEM DESIGN
Architecture, Methods and Tools

August 29 – 31, 2007, Lübeck in Germany

Pages: 511-518

Reference in Thesis – [Millberg2007b]

Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy

Mikael Millberg & Axel Jantsch
ECS - KTH - Royal Institute of Technology, Sweden
{micke, axel}@imit.kth.se

When designing a network the use of buffers is inevitable. Buffers are used at the entry point, inside and at the exits of the network. The usage of these buffers significantly changes the performance of the system as a whole. In order to enhance the buffer utilisation the concept of letting more than one packet exit the network at every switch each clock cycle is introduced - *Dual Packet Exit (DPE)*. The approach is tried on a 4×4 and a 6×6 mesh. We demonstrate the buffers used in combination with different routing strategies for best effort performance. The result we present shows a 50% reduction in terms of worst case latency and a 30% reduction in terms of average latency as well as an increased throughput both from a system and network perspective. We define the term *Operational Efficiency* as a measure of the network efficiency and show that it increases by roughly 20% with the *DPE* technique.

1. Introduction

When offering services with best effort performance, naturally, no hard guarantees can be given due to the dynamic behaviour of any general purpose system. In order to make use of such services statistical performance measures are instead utilised. As a consequence the traffic has to be kept below a certain threshold for which the desired statistical properties can be given. These properties can be derived from a rigid reasoning based on the current implementation *or* from observed simulation results where the offered services can be given with certain properties within a safety margin.

If *real* guarantees are to be given the cost is often high since the capacity of the network has to be allocated in such way that the network is inherently bound to be over-dimensioned for any general scenario. If the traffic patterns are static and known prior to network setup the hard guarantees often offer a good alternative, but due to the dynamic behaviour of a general purpose system static traffic patters are rare.

Given that we are bound to offer services with statistical characteristics on performance, how do we do this at lowest possible cost? The cost in this context is the required buffers needed to guarantee a *no-packet-drop* policy together with a safety margin in terms of injection rate to "guarantee" a certain worst, and average, case latency.

The approach that we have chosen, within the mesh based NoC Nostrum [1], for giving the service of *Best Effort* at a low cost, is by utilising deflective routing in order to keep the size of the switches small [2]. The small size is a consequence of not employing explicit buffering. Through simulations with uniform traffic patterns, we can conclude that there seems to exist an upper bound on the performance of the network. On networks of the sizes 4×4 and 6×6 this upper bound is reached for an injection rate of 0.63, and 0.45, respectively, for the routing strategies tried out. Injection rate is defined as *packets per node and clock cycle*. Once this limit is reached packets start queuing up at the entry points of the network and the worst case system latency grows exponentially.

In the course of extensive simulation and performance analysis it became clear that the exit point from the network is a severe bottleneck that keeps packets unnecessarily long in the network. The obvious approach taken is to increase this bandwidth; we call this solution *Dual Packet Exit*. The benefit of the Dual Packet Exit is a higher throughput of the network and a significant lowering of the buffer requirements at the entry points to the network because the worst case scenario now happens at a higher injection rate.

The price for this solution is that packets now need to be buffered at the exits of the network, but the need for buffers at the entry of the network is reduced so that from an overall perspective the buffers required in total is kept constant (and even lowered) for a moderately to a heavily loaded network. For the 4×4 mesh the average system latency is reduced from 14 to 9 clock cycles and the observed worst case latency is reduced from 85 to 45 clock cycles at an injection rate of 0.63. This will give better

margins before the network saturates *or* a higher throughput with the previous margin kept. All this, of course, assumes that there exists a balanced load in the network in the sense that no single node or bisection cut of the network are exposed to a static over-utilisation.

The validity of the chosen approach is not restricted to uniformly random traffic patterns on meshes but also applicable to "any" topology where the traffic pattern involves potential network exit congestions due to multiple sources having the same destination or where multiple routing paths are possible.

The network exit strategy has not received as much attention by researchers as other parts of network design. Most work that in detail analyse cost and performance of a router and the network as a whole, e.g. [2, 3, 4] assume an ideal packet ejection model, which means that packets are absorbed by the receiving node as soon as they are delivered by the network. In [5] an ejection policy is studied that reduces the cost and complexity of the router while minimizing the impact on performance. However, to our knowledge no study about the trade-offs involved in increasing the network exit bandwidth has been reported, as we attempt in this paper.

The rest of the paper is organized as follows: We start with giving a general overview of the platform used, to help the reader to relate the results to other work in the field. After this, we discuss how packets are generated and buffered in the system together with two hard limitations on what performance we could expect from "any" network at best. Then we present the contribution of the paper together with simulation results comparing *Dual Packet Exit* with simple packet exit. Finally, some discussions relate the approach to a general scenario in order to show where it is valid and useful.

2. System Overview

The topology that is chosen for the network is a n×n mesh which employs deflective routing with no explicit buffering (i.e. no queues) in the switches. Every switch is connected to a resource in a pair-wise fashion and a switch/resource pair is called a node. The total number of nodes in the system is $N = n \cdot n$. Packets are produced (generated) by the resource's **Packet Source** process, sent over the network, and later consumed by the **Packet Sink** process at the destination resource.The switches are individually connected to its four neighbouring switches in the direction of the compass. In addition we accept *no* packet loss. In our simulator a packet will receive a multitude of time tags for post simulation data analysis during its lifetime. The **Packet Source** generates λ packets, on average, every clock cycle. The packet is assigned a sequence number, tagged with a birth time, t_B, and thereafter pushed onto the resource's **Downstream Packet Queue** waiting for permission to enter the network. Once admitted to the network, the packet gets a send timing tag, t_S, and tries to reach its destination with a minimal number of hops according to the routing scheme described below. At the destination node the packet is ejected from the network and is pushed onto **Upstream Packet Queue** of the destination resource - this achievement renders the packet a reception time tag, t_R. The **Packet Sink** process polls the queue and if a packet is found it is popped from the queue and tagged with a finish time tag, t_F.

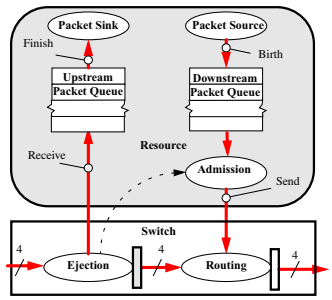


Fig. 2.1 The Switch and the Resource

Inside the switches there, conceptually, exist two separate stages - *Ejection* and *Routing*.

Ejection The ejection stage examines the incoming packets to detect if one (or more) have reached their destination and is to be delivered to the resource. In case of competition the packet with the highest priority is delivered. Also it informs the Resource's **Admission process** whether there is room in the switch for a packet to enter the network the next clock cycle. Since no explicit queues are employed in the network, admission can only be granted if the switch is currently holding fewer packets than its output buffer capacity, i.e. four packets.

Routing The deflective routing scheme is carried out in three phases: *Priority Assignment*, *Favoured Outport Selection* & *Permutation Routing*

Priority Assignment In this phase the incoming packets are dynamically assigned a priority, in our simulation the *Hop Count* (*HC*) is used. *Hop Count* is the time that the packets have spent in the network - a high *HC* means a high priority.

Favoured Outport Selection The packets now use their assigned priorities to select a desired outport. The priorities are utilised as credits which enable the packets to give different weights to favour a certain routing decisions in the coming *Permutation Routing* objective function. Here we try out two different strategies, *Uniform* and *Proportional*. *Uniform* implies that the packets use their priorities "uniformly" to select a favoured outport. That is, if a packet has a destination in a direction Northwest it will put a half of its priority to a routing decision where it gets routed to the West and half of its priority to a routing decision where it gets routed to the North. In the *Proportional*

strategy the packets favour a decision where the priorities are assigned proportionally to the direction of destination. An example: a packet has a destination two switches to the North and one to the West. The packet now chooses to use two thirds if its priority in favour of a decision where it will be routed North and the remaining third in favour of the West direction. The consequence of this strategy is that a packet will try to move in a direction where the degrees of freedom in routing is kept as long as possible to work against misroute closer to the destination.

Permutation Routing The weighted priorities of all the competing packets are summed to form the basis for selecting the best routing permutation. The number of permutations to select from in a four outport switch is 4! (= 24).

In order to slightly vary the routing strategy to make the analysis and claims about the importance of the buffer use stronger simulations are carried out with both the described variants of *Favoured Outport Selection*.

3. Packet Generation & Bounds on network Performance

Packets are generated with an average rate of λ. A λ of 0.3 means that there is a 30% chance that a packet will be generated during a clock cycle. The generation is "unaware" of the whereabouts of the network in the sense that it will perpetually generate packets regardless of the number of packets already in the **Downstream Packet Queues** or in the network.

Since each resource generates packets with an average rate of λ packets per clock cycle the total number of packets generated in the system every clock cycle is $N \cdot \lambda$.

The destinations of the generated packets are spread uniformly random over the network. This means that there, on average, will be a balanced load in the network, from the perspective of the source and destination nodes. The implications and validity of this approach will be further discussed in *8 - Discussions and future work*.

3.1. The Bisection Cut Bandwidth

The bisection bandwidth of the network is equal to the number of links crossing any bisection of the network [6].

The reasoning is the following: If all nodes emit packets with a uniformly random destination distribution half of the packets will with 50% probability cross the bisection in one direction.

$$\lambda \cdot N/4 \leq n$$

$$\lambda \leq 4n/N = 4n/(n \cdot n) = 4/n$$

If $n=4 \Rightarrow \lambda \leq 1$ and $n=6 \Rightarrow \lambda \leq 0.67$ this is the first upper bound of our network and it gives us a limit on how many packets that can, under uniform load, be transferred over the network per node.

4. Buffering

Buffers exist at three places: in the **Downstream Packet Queue**, in the network, and in the **Upstream Packet Queue**.

When talking about buffering we will discuss utilisation as well as minimum required capacity. Buffer utilisation is the average number of buffers utilised during the simulation, the utilisation is coupled to the dynamic energy consumption since we assume that energy consumption, in the buffers, is mainly dependent upon whether the buffers currently hold a packet or not. Regarding the minimum capacity it is the number of buffers needed to fulfil the requirement of no packet drop. For the **Up-** and **Downstream Packet Queues**, (which are implemented as FIFOs) the assumption of linear dependency of power consumption and buffer utilisation is unrealistic since they could be implemented in a memory structure as circular buffers. The buffer capacities of the FIFOs are derived from a worst case observed in the simulations presented in Section *7*. The buffer capacity of the network, is in general, equal to the number of switches times the number of buffers they contain.

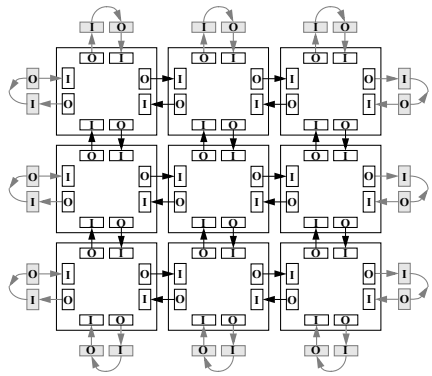


Fig. 4.1 The Buffers in the Network

Since the switch uses two stages: *Ejection* and *Routing* we have both in and output buffering (b=2). The number of buffers for the network is

$$BC = switchBuffers+edgeBuffers = 4 \cdot b \cdot n \cdot m + 2 \cdot b \cdot (n+m)$$

In the case $n=m$ we get

$$BC = 4 \cdot b \cdot n \cdot (n+1)$$

In our case with n=4 and n=6 we have 160 and 336 buffers available, respectively.

The geometrical distance between any two switches is denoted, $d_{geom.}$ The distance in clock cycles is the geometrical distance times the number of buffer stages, b, used in the switches. In addition to this, clock cycles for the buffer stages in the sender and receiver switches need to be taken into account to make a complete analysis, these are not, however, included in the formula below

$$d = b \cdot d_{geom}$$

If all nodes are emitting packets with a uniform random

destination, where the destinations are all other nodes excluding the sending node, the average geometrical distance, in a *n*×*n* mesh, is according to [6]:

$$d_{geom} = 2 \cdot n/3$$

The average distance in clock cycles is hence

$$d = b \cdot d_{geom} = b \cdot 2 \cdot n/3$$

On our case, with double buffering in the switches we get an average distance of $d \approx 5.3$ (n=4) and $d = 8$ (n=6). During the routing of a packet it will traverse a number of buffers along its way. The longer distance a packet has to travel the more buffers/network buffer capacity it will utilise. This was first described by Little [7] and in our example the average number of buffers utilised under "minimal" routing is:

$$B_{used} = P_{tot} \cdot d = \lambda \cdot N \cdot b \cdot 2 \cdot n/3 = 2 \cdot \lambda \cdot b \cdot n^3/3$$

4.1. The Network Buffer Capacity

In order to satisfy the demand of Little's formula the available number of buffers in the network has to be higher than the buffers required by the traffic. In the expression below the buffers at the edges are removed since they are not part of a minimal path.

$$B_{used} < n \cdot n \cdot b \cdot 4 - (n+n) \cdot 2 \cdot b$$
$$2 \cdot \lambda \cdot b \cdot n^3/3 < 4 \cdot b \cdot n \cdot (n-1)$$
$$\lambda < 6 \cdot (n-1)/n^2$$

This is the second upper bound of our network and with n= 4 we get $\lambda < 9/8$ and n=6 gives $\lambda < 5/6$

5. The Contribution - Dual Packet Exit

The contribution of this paper is the observation that the routing time for each packet inside the network has three different components: Minimum routing distance, Deflection *prior* to reaching destination for the first time, Deflection *after* reaching destination.

The first component - the minimum routing distance is not something that we can do much about, it is an lower bound that stems from the mapping and the traffic pattern.

The second component is the deflection prior to reaching destination. By this we mean the deflection that occurs before it reaches its destination for the first time. Once the packet has reached its destination for the first time and potentially is deflected due to an exit congestion it is considered to be in the third category.

The second component is however, a consequence of the competition for resources in the network. This in turn is dependent upon the overall load of the network and routing strategy. The load and routing strategy is tightly coupled in a looped fashion in the sense that a good routing strategy gives a lower load which in turn improves the possibilities for better routing. This component can easily be reduced by lowering the load of the network or by choosing a better routing strategy which is considerably more difficult. The difficulty lies within the problem of choosing a strategy that gives a good performance to *any* traffic pattern.

The third component is the deflection *after* reaching destination, which is the component that we here aim to lower.

5.1. Exit Congestion Limit

If two (or more) packets will reach the same destination node at the same time at least one of them will be deflected which contributes to the routing distance used in Buffer capacity bandwidth. The contribution will be four extra clock cycles for that particular packet since we are employing both in and output buffering of the switches.

The chance of two packets having the same destination is 1/N, if their individual destinations are randomly chosen in the range 1..N. If three random packets are chosen, {A, B, C} the cases of packet A and B, packet A and C and packet B and C having the same destination has to be taken into the formula. As well as the possibility of all three packets having the same destination has to be taken into account. The last scenario with three competing packets must be weighted with the penalty of *two* deflections. For an increasing number of competing packets the scenarios quickly becomes significantly more complex and a closed expression that captures the penalty for multiple packets having the same destination is hard to find. This problem has strong resemblance with what's referred to as the *Birthday Paradox*, which is the, not intuitive, high chance of two, or more, people having the same birthday in a group. If n is the number of people the chance of two or more people having the same birthday is given by the equation

$$P(n) = 1 - \frac{365!}{365^n (365 - n)!}$$

For the Birthday Paradox, a closed expression for the probability of coinciding birthdays apparently exist but does not, naturally, incorporate expressions for penalties varying with number of coinciding birthdays. In [12] this is well described and also it is shown that the uniform distribution gives rise to the smallest number of coincidences. For our problem this means that any non-uniform packet distribution worsens the problem of congestion at the exits of the network. However, this procedure of calculating the

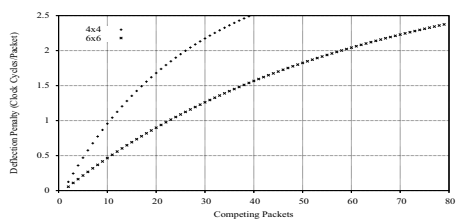


Fig. 5.1 Deflection Penalty

deflection penalties is carried out with an increasing number of packets competing and the number of deflections is depicted in Fig. 5.1. The horizontal axis shows the number of competing packets and the vertical shows the

average deflection penalty for each packet in a 4×4 and a 6×6 network. In short - this is the *average* penalty, in hops, that *every* packet gets its routing distance increased due to the exit node congestion!

If we compare these data with the circumstance that the average minimum routing distance in a 4×4 network is ≈5.33 the penalty of 1-2 clock cycles becomes significant due to the coupled load and average routing distance which will even further worsen the penalty in clock cycles due to the increased load, which will even further increase the load, and so on.

The obvious remedy for the exit packet congestion problem is to let more than a single packet exit the network per clock cycle and node. The cost connected to this is the dual wiring needed from the switch to the Resource together with the **Upstream Packet Queues** and corresponding logic. However this cost is relatively low compared to the gains that can be made in the network and **Downstream Packet Queues** as illustrated in Section *7*.

6. Simulation setup

The simulator used is fully written in SystemC [11] on a cycle accurate basis. It implements the network as well as the packet generators and the queues in the resources.

As mentioned before all resources generate packets with a rate of λ packets per cycle, with a random uniform destination pattern. All resources will generate 16000 packets each, in their respective, **Source Processes**. In total $P_{system} = N*P_{node}$ (N=16 ⇒ P=256000, N=36 ⇒ P=576000) packets will be generated during one simulation. The simulation is stopped when all packets are delivered. The reason for sending 16000 packets per node is that this is enough for making the effect of start-up and empty phases insignificant to the total result. The parameter that is changed from one simulation to the next, within one simulation run, is the injection rate, that is swept from 0.3 - 0.7 in steps of 0.002 for the 4×4 mesh. This means that we get 1+(0.7-0.3)/0.002 (201) measurement points from every simulation run. For the 6×6 network the range is 0.1 - 0.5 in steps of 0.004 which give 100 measurement points. In some graphs presented not the full range of measurement points are present in order to enhance the readability of the interesting portions of the graph.

We describe the measurements that we have chosen to present from two main perspectives: *End User* & *System Designer.* The *End User,* is interested in *Performance* like *Throughput & Latency* whereas the *System Architect* which may be more interested in implementation costs, like *Required Buffer Capacity*, and effectiveness measurements. Of course both of these are closely related and the *Average Latency* is an obvious shared concern since the *End User* sees packet latency but the *System Architect* may consider this as an increased cost in terms of energy due to buffering and switching.

6.1. Performance - Worst Case Latency

The *Worst Case Latency* is the biggest difference in t_B and t_F that we can find for any packet during the simulation. In order to enhance readability the worst cases in terms of latencies the graphs are made monotonously increasing, this is also done for the *Required Buffer Capacity.*

6.2. Average Latency

The average latency is the average of all packets' individual latencies. The *System Latency* and *Network Latency* of a packet is derived from $(t_F\text{-}t_B)$ and $(t_R\text{-}t_S)$, respectively.

6.3. Required Buffer Capacity

The Required Buffer Capacity has three components: The sum of all **Up-** and **Downstream Packet Queue** sizes and the buffer capacity of the network. The individual sizes of all the downstream packet queues in the resources are dimensioned from the observed worst case load of *any* downstream packet queue during the simulation. E.g. if one packet queue at any point in time held 10 packets all the packet queues in the network are given that size. The same is done for the up stream packet queues. In short:

$$N \cdot (DSLoad_{WorstCase} + USLoad_{WorstCase}) + BC$$

6.4. Operational Efficiency - Throughput per Buffers Used

The throughput (accept bandwidth) of the network is the average number of packets the system can deliver per clock cycle. When increasing the injection rate the throughput increases accordingly until the network is saturated. Given this fact it is easy to jump to the conclusion that the best performance is achieved by saturating the network! However, loading the network to this extent must come with a cost. The cost is the increase in average buffers needed to transfer a packet through the system, together with a, potentially, higher worst case latency. Ignoring the worst case latency, we propose the odd measure of *Operational Efficiency* to capture the *Throughput per Buffers Used* in the network. The idea is that, both, an underutilised as well as an overutilised network will give a bad ratio between the throughput delivered and number of buffers that is used. From a system perspective this is expressed as

$$OpEfficiency = \frac{PacketsTransmitted}{TotalTransmTime \cdot BuffersUsedPerPacket}$$

Since we do not have a realistic power model for the current implementation of the network it is hard to give any concrete numbers of how much energy a network will require in order to give a certain throughput. Anyway this measure will give a picture of the potential load of the network that gives acceptable performance.

7. Simulation Results - Comparisons and Discussion

7.1. Required Buffer Capacity

Both for the 4×4 and the 6×6 network it can be seen that the *Dual Packet Exit (DPE)* based approach has a higher buffer requirement at lower load, this since we get an early contribution from the **Up-stream buffers**. When the injection rate increases the 4×4 DPE network breaks down later than the *non-DPE*. This means that we, *either* can, drive the network harder *or* give performance guarantees with better margins! For the 6×6 network we get basically the same behaviour both with and without the DPE. The reason for not getting the improvements of the 4¥4 network is that since the network is bigger the packets spend a proportionally smaller time competing for ejection than on routing.

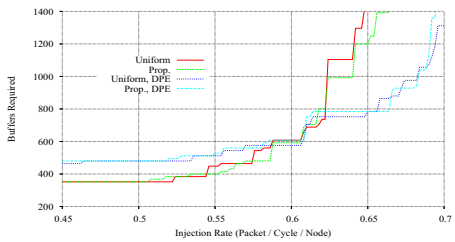


Fig. 7.1 Required Buffer Capacity with and without DPE - 4×4

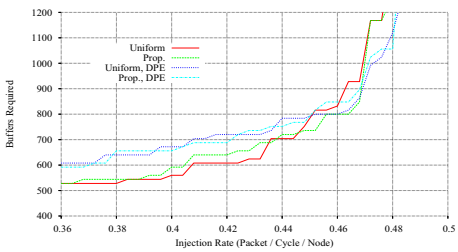


Fig. 7.2 Required Buffer Capacity with and without DPE - 6×6

7.2. Average Total Latency

If we look at the average total latency in the system we can see that all strategies have basically the same latency for low loads but for higher loads the DPE outperforms the other by far. If we set for an average latency of 10 cycles for the 4×4 mesh to the strategy with n*o DPE* and the *DPE* gives roughly 25% higher injection rate for the same latency. The main contributing factor is that packets, to higher extent, are given access to the network. The contribution of the *Exit Congestion Limit*, described in section *5 - The Contribution - Dual Packet Exit*, can be seen if we only look at the latency *within* the network, as depicted in Fig. 7.5. The *DPE* clearly reduces this effect and hence give a lowered average latency.

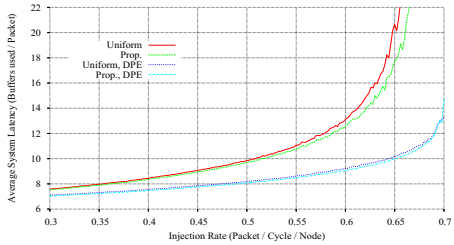


Fig. 7.3 Average System Latency - 4×4

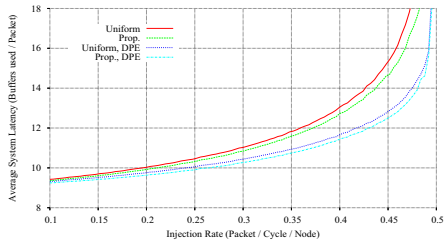


Fig. 7.4 Average System Latency - 6×6

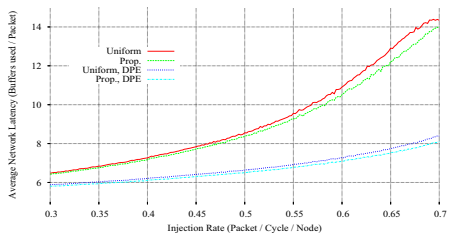


Fig. 7.5 Average Network Latency- 4×4

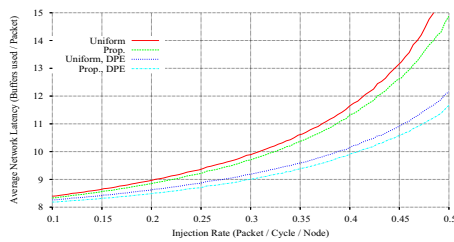


Fig. 7.6 Average Network Latency - 6×6

7.3. Performance - Worst Case Latency

As seen in Fig. 7.7 the worst case latency is reduced for the full spectra of injection rates for both sizes of the network. This result we consider to be the strongest benefit of the DPE approach since the worst case latencies now is within the same magnitude as the average case latencies. Also the worst case latency of the “break down” injection rate can be slightly shifted by the use of the *DPE* which

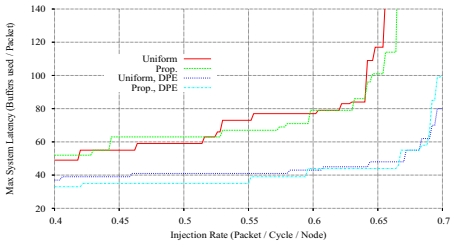


Fig. 7.7 Worst Case Latency - 4×4

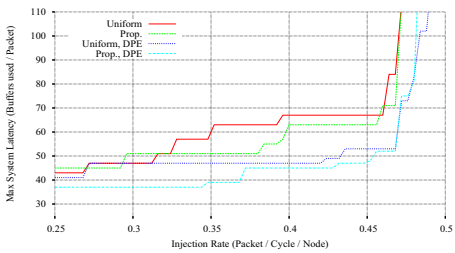


Fig. 7.8 Worst Case Latency - 6×6

can potentially give better margins before packet drop occur due to buffer overflow.

7.4. Operational Efficiency - Throughput per Buffers Used

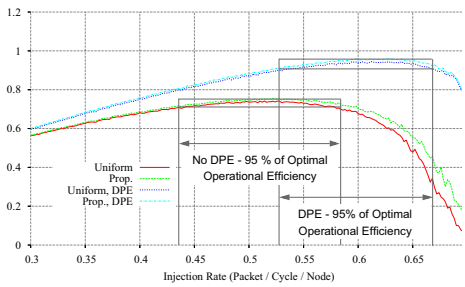


Fig. 7.9 Throughput Per Buffers Used - 4×4

Since the *Operational Efficiency* is derived from

$$\text{OperationalEfficency} = \frac{\text{SystemThroughput}}{\text{BuffersUsedPerPacket}}$$

the immediate observation that can be made in Fig. 7.9 is that with the *DPE* approach the throughput of the network can be increased while still using the same number of buffers. Also it can be seen the network utilising DPE is more effective. If we define a region where the network operates within 95% of its optimum that region is considerably moved in a higher throughput area ([0.44..0.59] ⇒ [0.53..0.67]) and the drop-off of the effectiveness also happens much closer to the saturation point of the DPE network. For the 6×6 network the situation is similar even

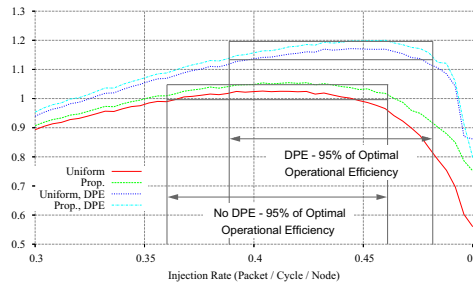


Fig. 7.10 Throughput Per Buffers Used - 6×6

though the *relative* effect is slightly reduced. Once again the relative reduction stems from the fact the packets spend relatively less time waiting for admittance in a larger network.

8. Discussions and future work

In this section we reason about the validity and generality of out approach.

8.1. Input/output balance

In our simulation we have assumed the network load to be statistically balanced. By this we mean that on average there exists a balance between the number of packets destined for a particular destination node in such way that if, on average, λ packets are generated by each of the N participating nodes at every clock cycle, λ packets will have a particular node as destination node. As shown in *5 - The Contribution - Dual Packet Exit* there exists a variance in the uniformity of the packet destination and due to this a potential congestion at the exits of the network. This is the reason for implementing the Dual Packet Exit. One might argue that if we have a "true" balance in the sense that we generated packets every λ clock cycle one "unique" packet for every destination, i.e., the packets generation process every λ is a permutation between the sets $\mathbf{N}_{16} \Rightarrow \mathbf{N}_{16}$ will the approach still hold? We claim that the answer is yes, and reason as follows:

The time the packets spend in the **Downstream Packet Queues** and in the network could for good reason be considered as random. For instance assume the latency for any packet varies by 10 clock cycles. The start permutation $\mathbf{N}_{16} \Rightarrow \mathbf{N}_{16}$ would become $\mathbf{N}_{160} \Rightarrow \mathbf{N}_{160}$, with 16 unique symbols, and the "permutation balance" would be more or less gone. This is not validated quantitatively and needs to be investigated further.

8.2. Applicability to other routing strategies and topologies

For any network that implements Best Effort and has a jitter in latencies that stems from random admission queuing times and/or there exist non-determinism in the routing

the problem of congestion at the output of the network will occur. And hence would benefit from an increased "exit-from-the-network" bandwidth.

We suspect that the effects for deterministic routing techniques such as wormhole routing are basically the same. Any worm denied exit from the network would be locking up resources in the network. These resources (buffers) would in turn, potentially, lock up other resources and so on. It will be interesting to investigate this presumption in detail.

8.3. Size of the Network

The results presented could potentially be claimed to be unrealistic due to the relatively small size of the network but we have a strong conviction that the network in question is quite realistic from the perspective of the uniform load in the sense that the 4×4 network could be seen as a subset of a bigger network where some nodes are employing an all-to-all communication pattern.

8.4. Uniform Load

The choice of uniform destination selection is most questionable from a real world application perspective. The obvious reason for choosing this pattern is simplicity and the possibility of making general statements from a relatively easily to describe approach. Up till now we are not aware of any "official", (by the NoC community approved) benchmarks for measuring performance. Hence any choice of traffic pattern that has some special characteristics has to be rigorously motivated before any claims can be made.

Except for the above mentioned difficulties we believe that the uniform traffic pattern actually is some kind of "best-case" pattern, because if we added burstiness to our approach the effect of limited exit bandwidth would actually be worsened since we know, if a small portion of packets get misrouted, will have a guaranteed exit congestion.

The same is valid for any stream between any two nodes since keeping the network under-utilised would be the only guarantee against the effect of a misroute disturbing the balance. Moreover, all experiments with specific application traffic patterns will not allow to draw general conclusions because any result may not hold for other traffic patterns. Even tiny differences of the traffic patterns may have a profound impact on cost and performance of the network. Thus, in summary the uniform distribution is a crude but robust assumption preferable to any other more "realistic" but arbitrary traffic.

8.5. Hardware Cost

The hardware cost that has been discussed in this paper is the relative cost of buffers. The cost due to extra wiring and logic between the switch and the resource is not included but our intention is to investigate this in future work.

9. Conclusions

We present the concept of *Dual Packet Exit* in order to increase the outgoing bandwidth between the network and resources since it is identified as a bottleneck. The effect of this increase in bandwidth gives better throughput and a lowered buffer requirement. This, *either* can be utilised for a higher packet injection rate offering better throughput with the same margins to network breakdown; *or* it can be utilised for a achieving a lowered load in the network for a fixed packet injection with better margins when offering QoS for the same throughput as result.

The buffers needed in order to buffer incoming traffic to the network are reduced but instead buffering of outgoing traffic is introduced. The net requirement of buffers is however reduced for a fixed injection rate. Using the DPE approach not only increases the throughput of the network while still using the same number of buffers, also it can be seen a system utilising DPE is more effective.

The generality of the approach is discussed Section *8* and we suppose that this bottleneck may exist in other networks than Nostrum as well. In addition to the future work suggested we will also investigate effects of further increase in the outgoing bandwidth.

10. References

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

[2] 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.

[3] L. S. Peh and W. J. Dally. A delay model for router micro-architectures. IEEE Micro, pages 26-34, Jan.-Feb. 2001.

[4] E. Rijpkema, et al. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In DATE, 2003.

[5] Z. Lu and A. Jantsch. Flit ejection in on-chip wormhole-switched networks with virtual channels. Proc. of the IEEE NorChip Conf., 2004.

[6] W. J. Dally and B. Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann, 2004

[7] J. D.C. Little. A proof of the queueing formulae L=λW. Operations Research, 9(3):383-387 May 1961

[8] E. Nilsson, M. Millberg, J. Öberg, A. Jantsch. Load distribution with the proximity congestion awareness in a network on chip. DATE 2003.

[9] M. Millberg, E. Nilsson, R Thid, A. Jantsch. Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip. In DATE, 2004

[10] U. Feige, P. Raghavan. Exact analysis of hot-potato routing. In Proc. of Foundations of Computer Science, p. 553 -562, 1992

[11] System C, www.systemc.org

[12] D. Bloom. A birthday problem, American Mathematical Monthly 80 (1973), pages 1141-1142.

Included Publications

Paper 6

Priority Based Forced Requeue to Reduce Worst-Case Latencies for Bursty Traffic

Mikael Millberg Axel Jantsch
ECS – ICT – KTH
Royal Institute of Technology, Sweden
{mick, axel}@kth.se

Pages: 1070-1075 Reference in Thesis – [Millberg2009]

Priority Based Forced Requeue to Reduce Worst-Case Latencies for Bursty Traffic

Mikael Millberg Axel Jantsch
ECS – ICT – KTH
Royal Institute of Technology, Sweden
{mick, axel}@kth.se

***Abstract* - In this paper we introduce *Priority Based Forced Requeue* to decrease worst-case latencies in NoCs offering best effort services. *Forced Requeue* is to prematurely lift out low priority packets from the network and requeue them outside using priority queues. The first benefit of this approach, applicable to any NoC offering best effort services, is that packets that have *not yet* entered the network now compete with packets *inside* the network and hence tighter bounds on admission times can be given. The second benefit – which is more specific to deflective routing as in the *Nostrum* NoC – is that packet "reshuffling" dramatically reduces the latency inside the network for bursty traffic due to a lowered risk of collisions at the exit of the network. This paper studies the *Forced Requeuing* on a mesh with varying burst sizes and traffic scenarios. The experimental results show a 50% reduction in worst-case latency from a system perspective thanks to a reshaped latency distribution whilst keeping the average latency the same.**

I. Introduction

When offering best effort services naturally no performance guarantees can be given. However, it is desirable to offer these services with the best possible performance in terms of throughput, latency, and worst-case behaviours. In this paper, focus is set on improving the worst-case latencies for multi-packet messages. Multi-packet messages manifest as traffic bursts in the network and dramatically worsen the performance. Most real world traffic exhibits burstiness to some degree. This does not constitute any problems if the traffic is orchestrated in such way that traffic bursts in the system do not collide on their way to destination. However, traffic that *is* utilising the best effort services often does so because the traffic behaviour *is* hard to predict in detail and hence, disqualified from utilising a guaranteed service.

During a packet's lifetime it will go through three separate phases: *admission* to the network, *transport* through the network, and *exit* from the network. Our previous work mainly focused on the two latter phases [7, 9] whereas this paper approaches the problem of *admission*. The approach that is chosen within the NoC *Nostrum* [8] for offering *best effort* services at a low cost uses deflective routing to keep the size of the switches small since no explicit buffers are used [4]. Most NoCs today employ variants of wormhole routing in favour of deflective routing mostly due to the packet reordering issue of deflection routing. However, studies carried out by Tota et al. shows the deflective NoC competitive, and possibly advantageous, to wormhole routing in terms of area and power [10]. The performance is neither better nor worse than its competitor on realistic multiprocessing benchmarks. In parallel to the deflective best effort services *Nostrum* also offers quality guaranteed services using a TDMA based scheme relying on the concept of Temporally Disjoint Networks [8]. During start-up of *Nostrum* different traffic streams are assigned to the appropriate services.

The key problem that we address is that: *Regardless of routing policy the best effort services inherently have a problem giving statistical bounds on the admission time to the network, i.e. bounds on the down stream queuing time before a packet can enter the network.* The reason is that it is hard to predict the traffic in the switch connected to the resource where a packet is to be injected into the network. In our earlier work [7] a solution to the problem of a guaranteed throughput service utilising the concept of *looping containers* was presented. Here, a solution for the best effort case is proposed. The worst-case waiting time is kept down by prematurely lifting out low-priority packets from the network to be requeued. A successful concept similar to ours is the *Diverting Switch* of Lang et al. [6] where packets in a competitive situation are sent to an alternative destination in the network to be resent later.

The validity of the concept is demonstrated in simulations; one using a uniform random pattern and another focusing on traffic to centrally placed memories. Both scenarios explore varying degrees of bursty traffic. In order to explain why bursts are harmful to network traffic an estimate is derived on the extra cost of congestion at the exits of the network as a function of the emission probability and the burst size. To our knowledge, no work has been presented working on an estimate on the delay due to multi-packet admissions in deflection routing networks. However, a number of papers describing upper bounds on delivery times in a network exist, e.g. the work of Hajek [5] and Brassel [2].

The paper starts with an overview of the platform used to help the reader in relating results to other work in the field together with the technical contribution of the paper. Next, the implication on the network performance in the presence of bursts and an estimation on what performance that can be expected from "any" network at best is discussed. Then, a hard-

ware architecture with an "acceptable" cost is suggested and justified by simulations comparing a *Forced Requeue* system with a system without. Finally, some discussions relate the approach to a general scenario to show where it is valid and useful.

II. System Overview with Priority Based Forced Requeue

The topology of our network is an n×m mesh employing deflective routing with no explicit buffering, that is, no queues in the switches. Every switch is connected to a resource. A switch/resource pair is called a node. Packets are generated by the resources' **Packet Source** process, sent over the network, and later consumed by the **Packet Sink** process at the destination resource. The switches are individually connected to its four neighbouring switches in the direction of the compass. The **Packet Source** generates λ packets, on average, every system clock cycle. Generated packets are pushed onto the resource's **Downstream Packet Queue** waiting for permission to enter the network. To simplify the analysis in the current setup only one out of the four independent time-slots of the TDMA based network is utilised and analysed. Hence, the system delays are scaled according to the **Packet Source** process's clock. At the destination node the packet is ejected from the network and pushed onto the **Upstream Packet Queue** of the destination resource. The **Packet Sink** process polls the queue and if a packet is found it is taken from the queue and can be considered delivered.

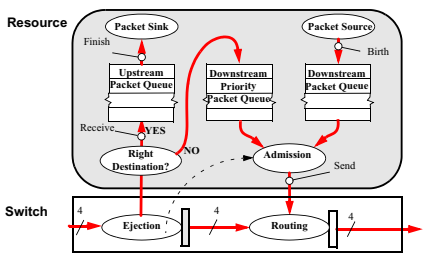


Fig. 1. The Switch and the Resource

Conceptually, inside the switches two separate stages exists – *Ejection* and *Routing*. In the *Ejection stage* incoming packets are examined to detect if they have reached their destination and are to be delivered to the resource. In case of competition the packet with the highest priority is delivered. Also, the Ejection stage informs the Resource's **Admission process** whether there is room in the switch for a packet to enter the network at the next clock cycle. Since no explicit queues are used in the network, admission can only be granted if the switch holds fewer packets than its capacity of four packets.

In the Routing stage the incoming packets are dynamically assigned a priority, in our simulation the *Hop Count* (*HC*) is used. The HC is the time a packet has spent in the network – a high HC means a high priority. The priorities of all competing packets are used to select the best routing permutation.

A. Priority Based Forced Requeue (PBFR).

The contribution of this paper is the idea that low priority packets/worms can be taken out from the network before they actually reach their final destination. The packets that have been forcefully taken out are requeued to be admitted to the network later. By forcefully taking out packets, the worst-case latencies that are caused by being in the downstream packet queues, potentially indefinitely long, are significantly reduced. The cost is mainly in hardware since a *priority queue* needs to be implemented in the Network Interfaces. The priority queue, however, *does not have to hold all the packets* of the **Downstream Packet Queue.** The packets originating from this very resource are known to be sorted already and can be kept in a separate queue. From a performance point of view, the penalty for taking packets out of the network is an increased delay for the *individual* packets that are forcefully requeued. From a burst point of view this is a non-issue rather the opposite thanks to the phenomena of Section III.

III. Understanding the Effects of Bursty Traffic

As can be seen in Section V., multi-packet bursts severely decrease the network performance; this despite that the traffic patterns, and the average packet injection rates, λ (in packets per cycle), are the same. Depicted in Fig. 2 are the increased individual packet latencies as a function of the burst sizes. As can be observed these latencies increase *at least* linearly with increasing burst sizes and hence make the packets of the bursts arrive at their respective destinations scattered in time. To develop an intuition why this degradation of the network performance occurs one must have knowledge about two related phenomena: How often packets in a network have the same destination and what is the cost incurred by this. We prioritise intuition over theoretical rigor and hence, this section should be considered a pointer in understanding the negative effect bursts could have on a network. Also, in our analysis only the effect of packet collisions at the destinations will be targeted, i.e. excluding the effects of bursts on the potential misroute of packets. Since the reasoning was developed with the deflective routing in mind the word *packet* is used – the reasoning in the

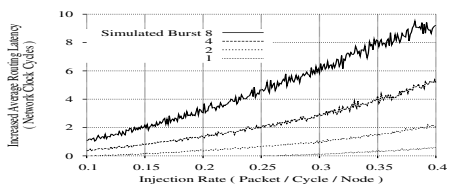


Fig. 2. Increased Routing Latency due to Bursts

upcoming Subsection III - *A. How Common are Common Destinations?* is, however, valid for a worm as well. The same holds for Subsection III - *B. What Does a Collision Cost?* where the burst size is changed into the length of the worm and a packet is a flit in the case of wormhole routing.

A. How Common are Common Destinations?

Assume a uniform distribution of the packet destinatios. How likely it is that packets will have the same destination? The answer is that it is, non-intuitively, very likely! To demonstrate, a very small network with only 4 nodes is chosen as a starting point. The nodes are all sending and receiving packets. If all permutations of destination patterns are transferred into a table that displays the relative frequencies of one or more packets sharing destination we get:

Competitors	Packets	Relative frequency
0	432	42 %
1	432	42 %
2	144	14 %
3	16	2 %

This means that only 42% of the packets will not experience *any* competition for destination whereas 58% will experience competition from *at least* one other packet! This problem is a variant of the *Birthday Paradox* [1] which is the, not intuitive, high chance of two (or more) people in a group having the same birthday. Intuitively one might object to that collision just *appears to be* this common due to two reasons. First, the small size of the network makes these numbers highly unrealistic. Second, uniform traffic does not coincide with any "real" scenario. To counter the first objection the competitions per packets for the 4x4 network are presented below.

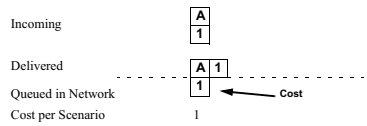
Competitors	Relative frequency
0	38 %
1	38 %
2	18 %

This means that, on average, 9 out of 16 packets will compete with other packets for any outcome. As it comes to the second objection it turns out that uniformly random traffic pattern actually gives the *least* number of coinciding destinations among the *random* traffic patterns. So, in general, packets/worms are most likely to share a destination under a random traffic pattern.

B. What Does a Collision Cost?

Given the conclusion about shared destinations – how likely is it that a packet will collide with other packets with the same destination and what does the collision cost? In order to develop an intuition some simplifications are made by only assuming the potential collision to take place at the destination nodes, i.e. ignoring effects of coinciding packet routes. Hence it is assumed that the burst will be delivered to the destination consecutively, i.e. it will not be split along its way. The validity of this assumption will be discussed in Subsection III - *D. The Moderating Effect on the Cost of Collisions.* Given these assumptions a *best case* scenario is derived as it comes to coinciding packet destinations.

To answer the question regarding the cost of collision burst size and the packet emission probability has to be known. In the case of not having bursts, i.e. the burst size is one; the potential cost will develop from one single scenario. Here, a packet denoted A competes with a packet denoted 1

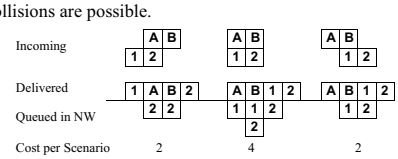


The cost per packet (cpp) in this single collision scenario is that one packet has to be queued per two incoming packets that collide. In the formula below the burst size is denoted b. To derive

$$cpp = \frac{1}{2 \cdot b} \cdot \sum cps = \frac{1}{2} \qquad cps = CostPerScenario$$

the average cost of having the same destination the emission probability, λ, is used to determine the interval length, i, in which a potential collision scenario will occur according to $i = b/\lambda$..

If the emission probability is 50% and the packets have the same destination, each packet will suffer an extra 0.25 clock cycle in delay due to competition for exit. If the burst size is increased, e.g. b = 2, the following three outcomes of packet collisions are possible.



In the general case, the total cost from all possible scenarios grows as b^3, which means that the average **c**ost of a pair-wise **c**ollision **p**er **p**acket (ccpp) can be reduced to:

$$ccpp = \frac{1}{2 \cdot b \cdot i} \cdot \sum cps = \frac{1}{2 \cdot b \cdot (b \div \lambda)} \cdot b^3 = \frac{b \cdot \lambda}{2}$$

In short – the cost for a collision increases linearly with the burst size! One important note to make here is that the average number of packets queued/in transit in the network per packet sent actually corresponds to an increased routing latency with the same amount of cycles.

C. The Combined Cost of Bursts

If the information about how often packets collide is combined with the relative cost of collisions a bound on the increase in latency due to the increased burst size is derived. Since it is only claimed to be a lower bound it is reasonable to

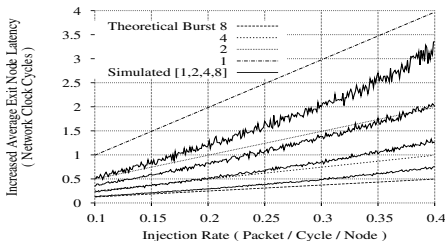


Fig. 3. "Theoretical Bound" on The Cost of Bursts

only consider all collisions to be pair-wise to simplify the analysis. For the 4x4 network the probability for a packet having a coinciding destination with another packet was 62%. For the *Nostrum* network, where switches are double buffered and deflection routing is employed, a deflection at destination means that the packet has to follow a minimal path of four hops before it can make a new attempt. This detour means that the penalty is quadrupled. If the penalty together with the relative number of packets competing is taken into the formula *cost of collision per packets* of Subsection III - B. it becomes:

$$ccpp_{16} = 4 \cdot 0.62 \cdot b \cdot \lambda \div 2 = 1.24 \cdot b \cdot \lambda$$

The graph of Figure 3 shows a family of plots over the increased latency that originates from pair-wise deflections at the destination vs. the emission probability. Depicted in Figure 3 are also the simulated corresponding "real" increased latencies. These simulated incremental latencies are *only* the latencies observed at the exits of the network due to bursty traffic. Additional latencies due to the bursty traffic in other parts of the network are *not* included; hence the situation is actually worse! On the other hand, what is striking when examining the graph closer is the rather distressing fact that the simulation result actually is *lower* than the theoretical lower bound! The main reason for this anomaly is that the effect of the increased burst size at the destination nodes are moderated by an increased degree of deflection for the packets on their way to the destination, i.e. the preconditions of the analysis do not hold – more on this in next section. In the case of wormhole routing the cost of collision would be different since flits are not deflected but simply buffered.

D. The Moderating Effect on the Cost of Collisions

With an increased burst size a moderating effect on the cost of collision will manifest itself. The cause of this moderation has two components. The first component is that the burst is split up due to misrouting along a packet's way from source to destination. The second is caused by the senders' inability to get packets that belong to the same bursts, out on the network consecutively. The downside of the analysis of Fig. 2 is that our assumption about having the *continuous* bursts only colliding at the destination nodes does not hold since the burst obviously have been *spread out over time*! The positive thing is that this moderating effect of spreading the burst to could be further be exploited as *a positive side effect* to the Forced Requeue approach!

IV. Hardware Implementation

The biggest objection to the use of the Priority Based Forced Requeue is the cost of additional hardware. Any solution that is going to operate at realistic speed will involve shift registers. In our search for an acceptable solution there are two lucky circumstances. The first one is the fact that the shift registers do not have to store complete packets but the packet's individual priorities together with a reference to a memory position where that actual packet is stored. The second is that only requeued packets need to be sorted. The packets originating from the Resources hosting the queue is already sorted and hence, only the heads of the individual queues need to be compared. Several proposals have been suggested for implementing priority queues – one appealing to our needs is the *Sequencer Chip* originally developed for the *ATM traffic shaper* of Chao and Uzun [3]. The basic idea is to keep sorting keys in registers and

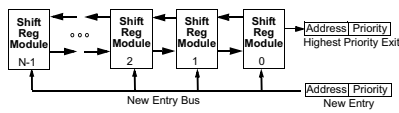


Fig. 4. Sequencer – Overview

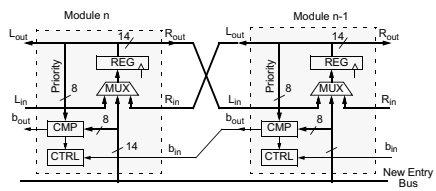


Fig. 5. Sequencer Module(s)

in parallel compare any incoming element to all the keys to determine which packets that needs to be shifted. From simulation data it could be observed that the priority queue of the individual network interfaces never exceeded 60 elements despite quite substantial traffic loads. Hence, a reasonable assumption is that 8 bits are needed for the sorting queues and 6 bits for memory references. This gives us the possibility to administer 64 packets with a maximum latency of 256 cycles. Each module will contain 14 bit registers and some combinatorial hardware. The combinatorial hardware needed is approximately 20k gates for the full sequencer. The memories required for the requeued packets will not add any extra cost since it can be observed from the simulations in Section V. that regardless of whether the priority queues of the *Forced Requeue* is used or not, roughly, the same number of packets will compete for

admission to the network. For a network using wormhole routing the cost is significantly less since only one time-tag per worm needs to be stored.

V. Simulation Results

Due to a limited space the result presented will focus on showing the decrease in observed worst-case latency. Regarding other aspects, such as average latency, throughput, etc. they are unaffected. In summary the network performance is claimed to be by no means worsened in any aspect due to the PBFR. The only effect observed is a reshaped latency curve for the delivered packets where the heavy tail is shortened and packets with low latency have a slightly increased latency but the *average* latency is kept constant.

A. Simulation Setup

Our cycle accurate simulator used is entirely written in SystemC. As mentioned before, all resources generate packets with a rate of λ packets per cycle. The experiments cover two different access patterns: (1) The Random Uniform Pattern (RUP) where all Resource nodes (R) are communicating with other nodes in the system with equal probability. (2) The Central Memory Pattern (CMP) implements an access pattern where

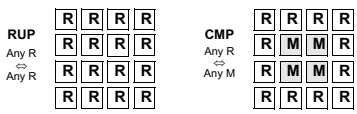


Fig. 6. Communication Patterns

the R is communicating with the centrally placed "memory" nodes (M) located in the centre of the chip. For both scenarios all nodes generate 2048 packets each, since it is enough to make the effect of start-up and empty phases insignificant. The parameter that is changed from one simulation to the next is the injection rate. For the RUP the λ increases from 0.100 - 0.400 in steps of 0.001. This gives 301 measurement points from a simulation run. For the CMP, the interval was 0.050 - 0.220 in steps of 0.001 giving 171 points. The lower range of CMP is due to the fact that the memory nodes are more heavily loaded. In some graphs not the full range of measurement points is presented to enhance the readability.

B. Required Downstream Buffer Capacity

The individual sizes of all Downstream Packet Queues in the resources are dimensioned from the observed worst-case load of *any* downstream packet queue during the simulation. E.g. if one packet queue at any time held 10 packets all packet queues of the network are given that size. In Fig. 7 it is seen that the total required buffer capacity is independent of whether PBFR is used or not. Also, the relative amount of traffic using the priority queues is depicted.

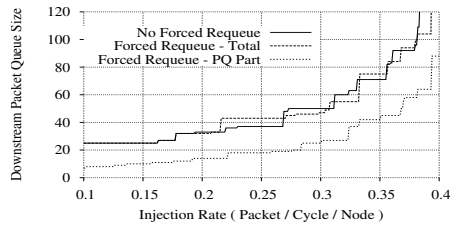


Fig. 7. Downstream Buffer Requirements

C. The Latency Distribution Shift

In order to understand what is happening with the latencies within one single simulation Fig. 8 is provided which is a histogram that depicts the latency. As can be seen the latency is shifted to the right when utilising Forced Requeue but with a shortened heavy tail but with average kept. To enhance the readability the graphs has been smoothed.

Fig. 8. Latency Distribution for λ = 0.400, burst = 8

D. Performance – Worst-Case Latency.

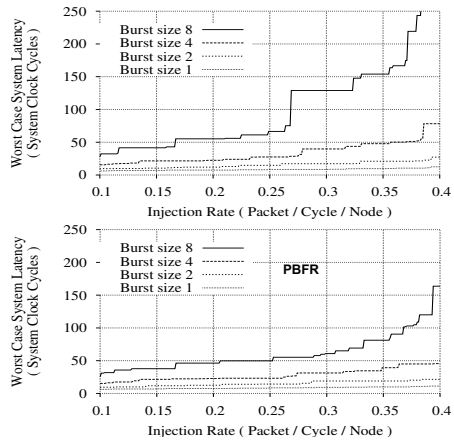


Fig. 9. RUP – Worst-Case Latency

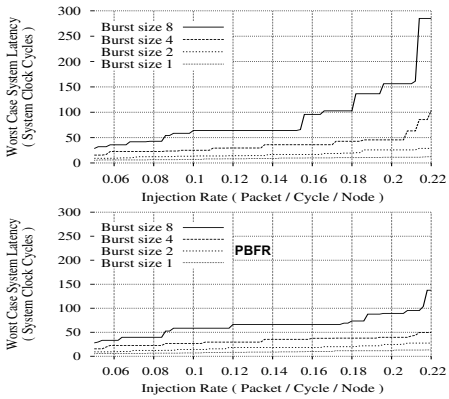


Fig. 10. CMP – Worst-Case Latency

In order to enhance readability the worst-cases in terms of latencies the graphs are made monotonously increasing, this is also done for the Fig. 7. As it can be seen in Fig. 9 (RUP) and in Fig. 10 (CMP), respectively the latency is roughly reduced by 50% if the PBFR approach is utilised.

VI. Discussion

For any NoC that implements *best effort* and is exposed to a random traffic pattern are bound to get packets competing for resources due to the phenomena described in Section III. The effects for routing techniques such as wormhole routing are basically the same. A worm denied exit from the network locks up resources in the network. These resources could in turn, potentially, lock up other resources and so forth. Hence, due to that the single biggest source of delay is the admission queuing time the Forced Requeue for routing techniques such as wormhole routing would be beneficial since tighter bounds on worst-case delay can be given. This has not yet been studied nor has the PBFR been implemented for wormhole routing.

A. *Nostrum - Full System Performances*

To keep the analysis clean from effects that comes from other improvements of *Nostrum* they are left out. If PBFR is combined with our Dual Packet Exit strategy [7] and a mild traffic regulation is used the worst-case latencies of Fig. 11 is presented and can be compared with Fig. 9.

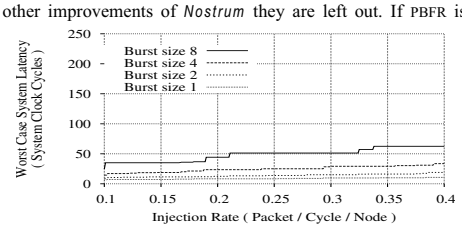


Fig. 11. Worst-Case latency with "All included"

VII. Conclusions

The concept of *Priority Based Forced Requeue* is presented due to that best effort services inherently have a problem giving statistical bounds on admission time to the network, i.e. bounds on the time a packet has to spend queuing before it can enter the network. The PBFR both reduces worst-case latencies as well as the harmful effects due to bursty traffic in the network. The contribution is the idea that low priority packets/worms can be taken out from the network before they actually reach their final destination to be resent later.

In order to give the reader an intuition about the harmful effect of bursts a model for giving an estimate on the effect of bursts is developed. The model shows that the cost in terms of extra delay is linearly dependent of the burst size.

From simulation data we claim that there are no performance degrading penalties related to the use of PBFR. However, there will be an extra cost in hardware and an implementation based on shift registers is proposed. As can be observed in simulation; using the PBFR approach will reduce the worst-case latencies by 50% while still using the same number of buffers! This is demonstrated both for a uniform traffic pattern as well as for a traffic pattern constructed to create hot-spots in the centre of the NoC. The generality of the approach is discussed and it is claimed that the performance improvements will exist in other networks than *Nostrum* as well.

References

[1] D. Bloom. A birthday problem, American Mathematical Monthly 80 (1973), pages 1141-1142.

[2] J. T. Brassel, "Deflection Routing in Certain Regular Networks", Ph.D. Dissertation, University of California (1991)

[3] H.J. Chao et al., A VLSI sequencer chip for ATM traffic shaper and queue manager, *IEEE J. Solid-State Circuits* 1992

[4] 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.

[5] H. B. Hajek, "Bounds on Evacuation Time for Deflection Routing," *Proc. CISS* (March 1989).

[6] T. Lang, L. Kurisaki: Nonuniform Traffic Spots (NUTS) in Multistage Interconnection Networks. *IEEE Journal of Parallel and Distributed Computing, 10(1): 55-67* (1990)

[7] M. Millberg, E. Nilsson, R Thid, A. Jantsch. "Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the *Nostrum* NoC." *DATE* (2004)

[8] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. "The *Nostrum* backbone – a communication protocol stack for networks on chip" *Proc. VLSI Design Conf., India* (2004)

[9] E. Nilsson et al. "Load distribution with proximity congestion awareness in NoC". *DATE* (2003)

[10] V. Tota, M. R. Casu, L. Macchiarulo, "Implementation Analysis of NoC: A MPSoC Trace-Driven Approach", pp. 204-209, in *Proc. of ACM Great Lakes Symposium on VLSI* (2006)

Included Publications

Paper 7

A Network on Chip Architecture and Design Methodology

Shashi Kumar[1], Axel Jantsch[1], Juha-Pekka Soininen[2], Martti Forsell[2],
Mikael Millberg[1], Johny Öberg[1], Kari Tiensyrjä[2] and Ahmed Hemani[3]
[1] *Laboratory of Electronics and Computer Systems, Department of Microelectronics and Information Technology, Royal Institute of Technology, 164 40 Kista, Stockholm, Sweden*
[2] *VTT Electronics, Box 1100, Oulu, FIN-90571, Finland*
[3] *Spirea AB, Kista Science Park, Electrum 209, S-164 40, Stockholm*

Pages: 105-112

Reference in Thesis – [Kumar2002]

A Network on Chip Architecture and Design Methodology

Shashi Kumar[1], Axel Jantsch[1], Juha-Pekka Soininen[2], Martti Forsell[2],
Mikael Millberg[1], Johny Öberg[1], Kari Tiensyrjä[2] and Ahmed Hemani[3]
[1] *Laboratory of Electronics and Computer Systems, Department of Microelectronics and Information Technology, Royal Institute of Technology, 164 40 Kista, Stockholm, Sweden*
[2] *VTT Electronics, Box 1100, Oulu, FIN-90571, Finland*
[3] *Spirea AB, Kista Science Park, Electrum 209, S-164 40, Stockholm*

Abstract

We propose a packet switched platform for single chip systems which scales well to an arbitrary number of processor like resources. The platform, which we call Network-on-Chip (NOC), includes both the architecture and the design methodology.

The NOC architecture is a $m \times n$ mesh of switches and resources are placed on the slots formed by the switches. We assume a direct layout of the 2-D mesh of switches and resources providing physical- architectural level design integration. Each switch is connected to one resource and four neighboring switches, and each resource is connected to one switch. A resource can be a processor core, memory, an FPGA, a custom hardware block or any other intellectual property (IP) block, which fits into the available slot and complies with the interface of the NOC. The NOC architecture essentially is the on-chip communication infrastructure comprising the physical layer, the data link layer and the network layer of the OSI protocol stack. We define the concept of a region, which occupies an area of any number of resources and switches. This concept allows the NOC to accommodate large resources such as large memory banks, FPGA areas, or special purpose computation resources such as high performance multi-processors.

The NOC design methodology consists of two phases. In the first phase a concrete architecture is derived from the general NOC template. The concrete architecture defines the number of switches and shape of the network, the kind and shape of regions and the number and kind of resources. The second phase maps the application onto the concrete architecture to form a concrete product.

1. Introduction

Current algorithm on chip and system on chip design methodologies cannot respond to the needs of the billion-transistor area. The design would take too much time and mapping of applications to dedicated architectures would be impossible. The possible solutions must be searched from platform based design and computer system design, which rely on the reuse of components, architectures, applications and implementations. The essential issue is the trade-off between generality and performance. Generality provides reusability of hardware, operating systems and development practices, while performance (delay, cost, power, etc.) is achieved by using application specific structures.

We propose a NOC platform, consisting of architecture and design methodology, which scales from a few dozens to several hundred or even thousands of resources. A resource may be a processor core, a DSP core, an FPGA block, a dedicated HW block, a mixed signal block, or a memory block of any kind such as RAM, ROM or CAM. We base this proposal on three assumptions:

1. Moore's law will continue to hold for another five to 15 years. In that case our platform should prove useful in the time period 2005-2015 [1].
2. Single processors will not be able to utilize the transistors of an entire chip. Single synchronous clock regions will span only a small fraction of the chip area [16, 2, 3].
3. Applications will be modeled as a large number of communicating tasks. The different tasks may have very different characteristics (e.g. control or data flow dominated) and origins (most of them are reused from earlier products or from external sources) [4]. This will make a heterogeneous implementation with different kind of resources for different tasks the most cost effective solution.

From this we conclude that a large number of different kinds of blocks, each of the size of a few hundred thousand gates, will constitute the computational resources. They have to be connected efficiently.

Increasing non-recurring cost of these chips require that design cost of chips must be shared across applications. Furthermore, the same or different variants of the same application have to be mapped onto different variants of

the product, each establishing a different solution of the cost/performance/functionality trade-off. If this can be done quickly and cost effectively, many product versions for various market niches can be supported. Physical level and architectural level design integration will be very useful for this. This implies that physical layout and implementation issues are kept in mind while taking architectural decisions, or the architectural design is carried out within constraints of physical size and a floor plan.

The proposed NOC platform would effectively separate the specification of inter-task communication from the implementation of that communication; separate the design, implementation and verification of individual tasks from the rest of the application (a precondition for task reuse); separate the development, optimization and verification of the individual resource from the network infrastructure. We argue that the consequent separation of different concerns is a way to develop high-performance, cost-effective products while boosting design productivity.

Here is not the place to speculate about the kind of products to be expected within five to ten years. However, we assume that the future devices will have the following requirements and features:

1. Processing of multiple ultra high data rate (> 100 MB/sec) streams of data including audio and video data. The devices will be required to store this data and process it in real time.
2. Devices will be multi-functional. The functionality could be a mix of entertainment (like games, music instruments), communication, remote control, surveillance etc.
3. Devices will have high-capacity wire line or more likely wireless interfaces to standard networks like telephone network, Internet, and will need to be able to handle multiple communication protocols simultaneously
4. Security and secrecy of data stored and flowing through these devices will become important.

Clearly, a NOC based design will not always be the preferred solution for all kinds of applications. We expect that NOC based designs will provide good solutions for flexible products that should be reconfigurable and programmable; for designs which are the basis for several product variants; for applications with a heterogeneous task mix; for applications with stringent time to market requirements; for products where reuse both at the block and the function and feature level is considered valuable.

The design costs can be justified by increasing the implementation volumes and it is likely that the billion-transistor chips are not designed for single product instances or single applications. The design methodology must therefore support product family management. Tolerance of incomplete specifications, management of configurations and modifications, support for multiple languages and methods, and capability to handle different abstraction levels simultaneously are desirable characteristics.

Verification and testing are ever increasing challenges in today's design routines. With every new technology generation they are becoming more pressing. We argue that the NOC platform effectively addresses these challenges by separating the computation resources from each other and from the communication network for all issues of design, verification and testing.

In section 2, we list some other research work related to complex system design on a chip. In section 3 we describe the basic ideas and concepts of our proposed NOC architecture. In section 4 we describe the principles of design methodology for NOC based systems. In section 5 we discuss issues of physical implementation and performance for NOC architecture.

2. Related work

It is being realized, by all research groups involved in system level design, that it is absolutely necessary to allow reuse of already designed components or blocks. Gajski et. al. [5] have proposed an IP-centric embedded system design methodology. The major challenges in the IP centric methodologies are the interface synthesis among various IP blocks and system verification. Recently, Platform Based Design methodology [6] has been proposed which not only allows reuse of components but also reuse of system architectures and topologies. The basic idea is that an architecture, which is suitable and efficient for one application will also be suitable and efficient for many similar applications. The idea of using the same architecture (platform) for development of application not only speeds up application design but also reduces its verification time. Keutzer et. al. [7] have extended the idea of platform based design by including a layer of software on top of the hardware platform to help application development. This layer is called Software Platform. The combination of hardware and software platforms is referred as System Platform. It has also been realized that the key to reuse and integration of IP components is the communication from the physical to the system and conceptual level, and consequently communication centric architectures, platforms and methodologies have been developed [8, 9, 10].

Many architectural templates have been proposed for hardware platforms for future SoCs. There is a general emphasis on providing efficient and standardized communication infrastructure for connecting multiple resources on the chip [11, 8, 9]. There is a trend to adapt layered approach of OSI reference model towards on Chip communication [12, 10, 13].

It is estimated that video and audio processing are going to be common tasks in many applications. These applications are going to require storage and processing of large amount of data. It is predicted that memories are going to take around two third of the chip area in future system on chips [14]. Many researchers have concentrated on analyzing hierarchical organizations of memories and

optimization of memory sizes and data storage strategies for data intensive applications [15]. Researchers have alos simulated theoretically elegant shared memory model on message passing parallel computers in order to develop data intensive applications on them [19].

The future system on a chip, incorporating many different types of processing and memory elements, has to operate using Globally Asynchronous Locally Synchronous (GALS) paradigm [16], at least at the hardware level. GALS paradigm not only avoids the problem of clock skew but also leads to lower power consumption.

3. Network on Chip Architecture

The NOC architecture provides the communication infrastructure for the resources. We have two main objectives. Firstly, it is possible to develop the hardware of resources independently as stand-alone blocks and create the NOC by connecting the blocks as elements in the network. Secondly, the scalable and configurable network is a flexible platform that can be adapted to the needs of different workloads, while maintaining the generality of application development methods and practices.

3.1. The NOC network

We chose a simple mesh interconnection topology as basic topology, because it is simplest from a layout perspective and the local interconnections between resources and switches are independent of the size of the network. Moreover, routing in a two-dimensional mesh is easy resulting in potentially small switches, high capacity, short clock cycle, and overall scalability.

A NOC consists (Figure 1) of resources and switches that are connected using channels as a mesh (Manhattan-like structure) so that they are able to communicate with each other by sending messages. A resource *R* is a computation or storage unit or their combination. A switch *S* (Figure 2) routes and buffers messages between resources. Each switch is connected to four other neighboring switches through input and output channels. A channel *C* consists of two one-directional point-to-point buses between two switches or a resource and a switch. Switches may have internal queues to handle congestion. We call this approach *Chip-Level Integration of Communicating Heterogeneous Elements* (CLICHÉ).

The precise layout and geometry depends on the technology generation. We expect that the area of a resource is the maximal synchronous region in a given technology. It is expected to shrink with every new technology generation. Consequently the number of resources will grow, the switch-to-switch and the switch-to-resource bandwidth will grow, but the network wide communication protocols will be unaffected. Figure 1 illustrates the principles of the physical floor plan within the NOC. Consider a 60nm CMOS technology expected in 2008, a 22mm × 22mm chip size, and a resource size of 2mm × 2mm and a minimum wire pitch of 300nm. A NOC would accommodate 10 × 10 resources, each switch would occupy 30µm × 30µm and the channels would be 30µm wide. Assuming that we can use 3 metal layers for the switch-to-switch connection we have space for 300 wires. Since we need control, handshaking and signaling bits will yield an effective data bus width of 256 bits.

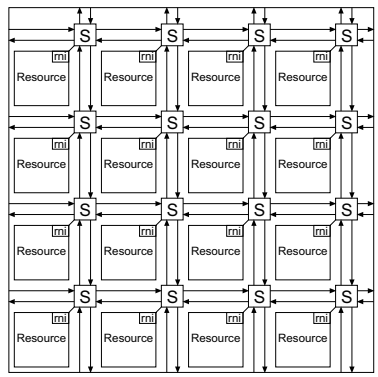


Figure 1. A NOC with 16 resources.

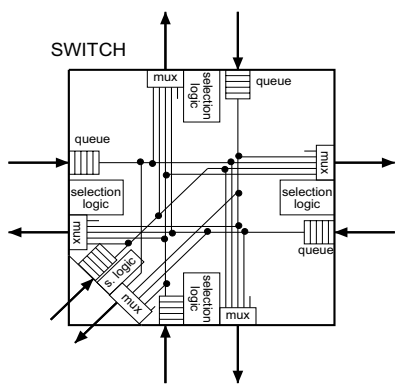


Figure 2. Block diagram of a switch.

3.2. NOC resources

The NOC would allow for arbitrary resources. Typical examples would be embedded processor and DSP cores provided with caches as well as local memories, dedicated hardware resources, and configurable hardware resources. Since the area of resource equals one synchronous clock

domain, the resource can be a combination of all previous types. The internal communication inside a resource is synchronous. In Figure 3 RNI=resource network interface, P=processor core, D=DSP core, c=cache, M=memory and re=reconfigurable block.

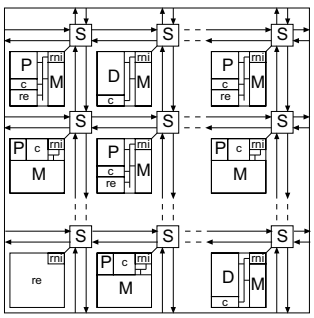


Figure 3. A typical NOC CLICHÉ featuring various types of resources.

The model of computation is a heterogeneous network of resources executing local computation. Communication between the resources is implemented by passing messages over the mesh network. Resources operate asynchronously with respect to each other. Synchronization is provided by synchronization primitives, which are implemented by passing messages around the network. Even a non-local memory is accessed through message passing.

In order to make the NOC interface with the outside world dedicated resources such as I/O elements are needed. The I/O could be of various kinds, they could glue many NOC chips together, interface with external memory or implement a TCP/IP interface. Interface modules also handle data buffering and packet reordering.

3.3. Communication

Every resource has a unique address and is connected to a network via a switch. It communicates with the switch through a RNI. Thus, any resource can be plugged into the network if its footprint fits into an available slot and if it is equipped with an RNI. The NOC defines four protocol layers:

1. The *physical layer* determines the number and length of wires connecting resources and switches.
2. The *data-link layer* defines the protocol to transmit a *cell* between a resource and a switch and between two switches. Both, the physical and the data link layer are dependent on the technology. Thus, for each new technology new technology generation these two layers are defined. Let w be the number of wires in the physical layer and c be the cell size of the data link layer. We expect that $c=n(w-w_c)$ with n=1,2,3 or 4. For n=2,3 or 4 the channel would be pipelined, accommodating n data link cells at any time instant. w_c is the number of control wires required by the physical layer, e.g. synchronization signals.
3. The *network layer* defines how a *packet* is transmitted over the network from an arbitrary sender to an arbitrary receiver directed by the receiver's network address. This layer is again technology dependent and each network layer packet, together with the destination address, is exactly 1 data link cell. Thus, taking up our previous example, we have w=300 and c may be 290. We need roughly 10 bits for the address and a few control bits (e.g. a hop count) for switching. Hence, the network packet would be 256 bit.
4. The *transport layer* is technology independent. The transport layer message size can be variable. The RNI interface has to pack transport layer messages into network layer packets.

The RNI implements all four layers towards the network. The switch-to-switch interfaces implement only the three lower protocol layers. The basic communication mechanism envisioned among computing resources is message passing. However, it is possible to add additional protocols on top of the transport layer to provide for instance a virtual shared memory abstraction, which will help the programmers in development of data and computation intensive application.

3.4. Regions and wrappers

A 2-D mesh topology provides access to all resources of the NOC, it is scalable and it has a simple structure. However, there are applications for which CLICHÉ structure is not suitable for performance reasons. Examples can found from parallel computation, digital signal processing and data flow processing areas.

A region G is an area inside the NOC, which is insulated from the network and which may have different internal topology and communication mechanisms. The concept of region allows for resources of larger size than the atomic slots in the mesh. In this way development, management, communication and instantiation concerns of various regions can be separated. Regions are connected to the NOC by special communication arrangements called wrappers W, which route packets so that regions are insulated from external traffic. Specific IO wrappers W_{io} allow communication between the region and its environment. It is also responsible for converting the messages into appropriate format. Thus, the region concept in NOC can be seen to address four aspects:

1. A region can be used to *dedicate* a set of resources and a part of the network to a specific task like processing of streaming-oriented data, processing of block-oriented data or parallel processing.
2. One can *arrange* communication inside a region differently than in the other regions. A NOC designer may e.g. want to define a region with high

communication capacity for efficient work-optimal implementation of shared memory abstraction [20].

3. A region can be used to *insulate* a set of resources from the traffic happening between the resources not belonging to the region.
4. A region can be used for *encapsulating* a specific technology into a NOC. For example an area dedicated to FPGA or embedded memory could be larger than the area of resource.

However, the shape of regions cannot be arbitrary but their boundaries must be convex. This definition of regions imply that resources requiring high-capacity intercommunication need to be placed into the same region, because wrappers between regions may cause some constraints to capacity and latency of communication. From the point of view of the network layer, regions do not form separate sub-networks, instead they can be considered as just lightweight mechanisms to organize communication in a more efficient and rational way.

4. Backbone-Platform-System Methodology

Our NOC concept is based on the idea to have a backbone based application specific platform where the final applications can be mapped as software or configurable hardware. Combination of design productivity and system quality requirements has led us to *the backbone-platform-system design methodology* (BPS). The idea with the BPS is to encapsulate the design work into reusable platforms. A NOC based system consists of a hierarchy of structural and behavioral objects, e.g. backbone, platform and system concepts. BPS has two main phases, platform development and application mapping, as depicted in Figure 4.

Even in a small 4x4 meshes of switches and resources there are 16 subsystems with a complexity of current state-of-the-art SOC design each. Management of such complexity must be based on extremely structured architecture and extensive reuse. In BPS methodology the generic, structured architecture and system development principles are described as a backbone concept. Development of several SOC complexity level subsystems, e.g. resources in CLICHÉ topology, must be based on the reuse of optimized virtual components or even computer systems. If we assume that current SOC design has a moderate complexity of 10 million gates, then even in small 4x4 mesh the hardware complexity would approach 200 million gates.

The computational capacity of NOC based system depends on the type of resources. If we assume that resources are general-purpose processor based computer systems with a capacity of 1000 MIPS each, the 4x4 mesh would have a total capacity of 16 GIPS. In real system, part of the capacity would be wasted due to communication and allocation problems, but it is obvious that reuse of applications, middleware and system architectures is required.

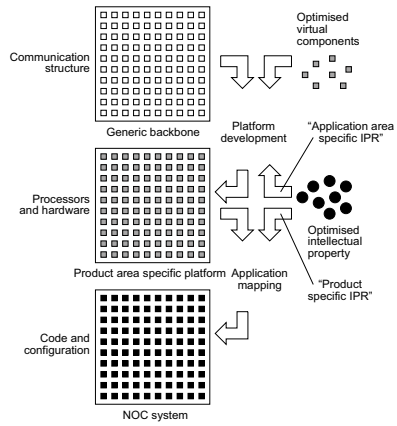


Figure 4. NOC based system design.

4.1. Backbone design

The NOC backbone encapsulates the topological and communication issues such as channels, switches, and network interfaces. The backbone is the development platform for all NOC based systems, so it is important that every system follows the basic operation principles defined in the backbone.

During the backbone design the focus is the network communication resources, e.g. switches and interfaces, and NOC system services and performance of different region topologies. From the definition of resource area follows that the connections between neighboring switches and the switch design are issues where physical design has an important role. The system-level communication challenges the technological limits. The amount of wires, wire lengths, synchronization, and buffering are all problems were physical layout and characteristics sets constraints. Customized region topology enables NOC based systems were the quality of the application mapping is optimized in the beginning. Definition of region requires that potential applications are analyzed and modeled. Mathematical and performance analyses and even performance simulations are the main tools to be used.

4.2. Platform design

The objective of platform development is to create a computation platform for an intended application area. Scaling of the network, definition or regions, design of the resource nodes, and definition of the system control are the main activities. It requires thorough understanding of the functionality of the target systems, but due to the platform

nature it is not possible to use exact applications as a starting point for architecture requirement definition. Use of optimized virtual components and knowledge of application-area requirements are essential in managing the complexity and performance requirements of the target system. During the development the characterization of application area domain and architecture and system quality estimations are essential tools. The application area specific platform encapsulates the hardware design problems and serves as a manufacturing integration platform for system developers.

For example, in 4x4 mesh CLICHÉ system, we have to define and design 16 resources, e.g. 16 communicating computer systems, if the NOC platforms would be used for the parallel implementation of heterogeneous applications. If we want to optimize the platform for some specific application area, we certainly need very efficient ways of making the right decisions and new figures of merit to describe the quality of NOC. Currently used metrics: performance, utilization, capacity must be adapted to handle temporal and spatial effects that are inevitable with target systems. For example with combined communication and computing systems, the required architectural features may vary from bit-based processing to parallel manipulation of huge data sets. The communication throughput and latency requirements are different in the same way.

4.3. System design

In the application mapping the functionality of application is mapped to the resources. The NOC concept should ultimately support both dynamic and static mapping of applications, but the main problems with both are the resource allocation, optimisation of network usage and verification of performance and correctness. Basically these issues are rather similar to what distributed and parallel system designers have to face.

The proposed NOC platform is very heterogeneous. The resources can vary from configurable hardware to multiprocessor computers of almost every type. Therefore, several modeling languages should be supported by NOC application development environment making it easy to integrate different tools into the design flow. As with platform design, the decision support and quality validation needs special attention and new approaches.

4.4. Methods and tools

Implementation of the BPS methodology or any other design flow for NOC systems will be a challenge for EDA industry. The traditional SOC, platform, and intellectual property based design flows must be extended to cover network-related issues, e.g. distribution and parallelism effects as described in Table 1.

Table 1. Design responsibilites during different phases of NOC development.

Instance	Responsibilities during design
Backbone development	Region types Communication channels and switches Network interfaces of resources Communication protocols (specification)
Platform development	Region scaling Resource design (units, interconnections) Dedicated hardware blocks System level control (implementation of communication, diagnostics, monitoring)
Application development	Resource level control (OS) Functionality of resources (SW, configurable HW) Control of the network Functionality of the network

Our NOC backbone defines the implementation of the network. The main task for designer at system level is to decide what to put into the NOC as resources, how to map functionality into those resources, and how to validate the decisions. The actual design relies on the reuse of virtual components and intellectual property, and enhanced methods and tools to support them are required. Especially at system level it is important to use abstract models and descriptions of both resources and applications. Otherwise the computational complexity of analyses, estimations and simulations will exceed the computational capacity of design tools. In traditional system design approaches the design space exploration has been done using with analytical approaches or with similar design methods and tools than the actual design. Most often, only the abstraction level of system models has been different.

In NOC design, we propose a clear distinction between decision making support, development and verification methods and tools. The *decision environment* should include methods for advanced complexity estimation, resource selection, and network analysis. Complexity estimation is needed for the scaling of NOC and for region type selection. The characteristic of computation is one issue that needs to be added to operational complexity. In the resource selection the mappability of algorithms and architectures is one alternative extension to currently used performance metrics that could provide more knowledge on the potential quality of the system. Similar analysis could be used during application mapping. Analysis of network behavior is a critical part of region definition and allocation of resources to functions. Modeling of network behavior, workload characterization and efficient simulation are the potential methods, if adapted to NOC concept. The *development and verification environments* should provide a virtual machine and development environment for software development, and tools for hardware design. Complexity is the biggest challenge in both. Abstraction, partitioning of problems and distribution of computation looks as viable alternatives.

5. Discussion

Design of a new product using NOC architecture is similar to the problem of designing a computer network with some computing and communication requirements. We have adapted ns-2 from Univ. of Berkeley at California, to study various design options in NOC architecture and their effect on performance[17].

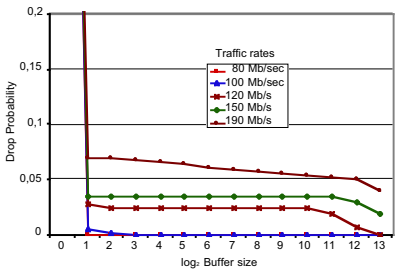


Figure 5. Drop probability vs. buffer size.

We have used a homogeneous 5 x 5 NOC architecture for our simulation experiments. In particular, we have studied the effect of buffer size in switches and network traffic (called network load) on delay and probability of message loss. These simulation experiments have been carried out using various types of network traffic cases like random traffic and local traffic and mix of these. The figure below shows relationship between the probability of a packet being dropped verses the size of buffer in the switch for each direction. We have assumed that a link between two switches supports a maximum traffic of 200Mbits/sec. Various lines in the graph show the drop probability verses buffer size for various actual traffic rates. We observe that for actual traffic of up to 100Mbits/sec, the drop probability is very close to zero if a buffer size of four packets is used. Traffic rate is controlled by controlling the rate at which a subset of resources generate packets and by controlling the destination address of the generated packets. The traffic generated for this study had a mix of local and random traffic. We have carried out many other similar experiments [17].

These simulation studies have resulted in many interesting conclusions: For moderate traffic, a buffer size of 8 messages for each direction leads to almost zero drop probability. Message delay increases with buffer size as well as network load. Message delay is more sensitive to network load than to buffer size. If the network load increases beyond 50% of network capacity, then it is impossible to avoid message drop even with large buffers.

This study helps us to decide size of buffer in switches. It also emphasizes the need for good mapping of applications to the NOC architecture so that the resulting traffic is local to a small area of the NOC. This will reduce network traffic.

5.1. Physical Aspects of NOC

We have investigated some physical issues in the design of the switches and the inter-switch connections for on-chip communication networks like NOC [18]. In particular, we have compared two distinct layouts for a switch, called “thin switch” and “square switch”. In thin switch, the switch functionality is distributed around a resource and wires are routed across the resources. A square switch is placed on the crossings in dedicated channels left between resources. The wires are routed in these channels.

We have considered wireability, delay and maximum signal bandwidth between switches, positioning of pads and positioning of repeaters in our study. The study has been conducted based on the 60nm CMOS technology expected in about 6 years. The main conclusion is that in five years 10 x 10 NOC architectures will be feasible. It will be possible to route 256 wires between a resource and a switch and between two neighboring switches in the mesh. The study also shows that the square switch option is superior with respect to performance and bandwidth while the thin switch requires relatively low area.

5.2. System development

The main objective for the NOC development environment will be to separate different concerns and activities and to shield some tools and design tasks from details in other tools and tasks.

The BPS methodology tries to benefit from reuse as much as possible and to give support for application development. The idea has also been to find an optimal balance between manufacturing and system level integration platforms. The role of the backbone is to provide a solid starting point for ASIC design with guidelines and flexibility.

The NOC system development environment will provide layered system services, which will shield an application developer from the details of the NOC lower level architecture. It will provide application level communication, synchronization, memory management, and resource management services.

Design tools, which map applications onto the NOC, must eventually implement all communications between resources by means of the three protocol layers provided by the network. This can be considered as a contract. If the applications comply with these protocols the network guarantees the communication services. Ideally we would like to extend this contract also to performance issues, for instance with a contract where applications guarantee a maximum number of messages per time unit and the network guarantees a maximum transport delay of all messages. It is part of our future work to define the conditions under which such a contract is feasible.

6. Conclusions

In this paper we have described an architectural template, called network on chip architecture, for developing large and complex systems on a single chip. The architecture supports physical level and architectural level design integration. Basic communication mechanism between resources is envisioned to be packet switched message passing through the switches. NOC architecture defines four layered inter-resource communication protocol (physical, data-link, network and transport layer), which are adapted from OSI standard. These protocols must be implemented in the resource to network interface (RNI) for every resource in NOC. We have also described a two-phase design methodology for developing systems for the proposed NOC architecture.

The NOC concept has been necessitated by three factors: First there is the increasing demand of on-chip interconnect bandwidth. The second equally crucial factor is to amortize the enormous engineering cost involved in designing such large chips over multiple applications. The third factor is demand for easy-to-use methods to exploit the parallel processing capacity provided by multiple computational resources. Programmable interconnectivity and efficient implementation of shared memory abstraction are keys to provide this generality.

Before NOC architectural template can be used to develop applications, one needs to work out the details of architecture, communication, design flow, and system services. Currently we are building many simulators for evaluating various architectural and communication options at different levels. We are also interested in analytical analysis of architectural options for NOC.

7. Acknowledgements

We gratefully acknowledge many valuable discussions we had with Dr. Li-Rong Zheng and Dinesh Pamunuwa. This work is a part of the joint Finnish-Swedish EXSITE (Explorative System Integrated Technologies) research program. This work was sponsored by TEKES (The National Technology Agency of Finland), VINNOVA (Swedish Agency for Innovation Systems), Nokia Oyj, Ericsson Radio Systems AB, and Spirea AB Kista.

References

[1] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, World Semiconductor Council, Edition 1999, 1999.

[2] D. Sylvester and K. Keutzer, "Getting to the Bottom of Deep Submicron", *Proc. of the Int. Conference on Computer-Aided Design*, 1998, 203-211.

[3] D. Sylvester and K. Keutzer, "Getting to the Bottom of Deep Submicron II: a global wiring paradigm", *Proc. of the 1999 Int. Symp. on Physical Design*, 1999, 193-200.

[4] C. Szyperski, *Component Software: Beyond Object Oriented Software*, Reading, MA, ACM/Addison Weseley, 1998.

[5] D. Gajski, R. Dömer and J. Zhu, "IP-Centric Methodology and Design with the SpecC Language" in *System Level Design*, Edited by Ahmed A. Jerraya and Jean Mermet, Nato Science Series 357, 1999.

[6] F. Vahid and T. Givargis, "Platform Tuning for Embedded Systems Design", *IEEE Computer* 34, 3.

[7] K. Keutzer, S. Malik, A. Newton, J. Rabaey and A. Sangiovanni-Vincentelli, "System Level Design: Orthogonolization of Concerns and Platform-Based Design", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 19, 12 (Dec. 2000).

[8] W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", *Proc. of the Design Automation Conference*, Jun. 2001.

[9] D. Wingard, "MicroNetwork-Based Integration of SOCs", *Proc. of the 38th Design Automation Conference*, Jun. 2001

[10] M. Sgroi et. al., "Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design", *Proc. of the 38th Design Automation Conference*, Jun. 2001.

[11] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist, "Network on Chip: An architecture for billion transistor era", *Proc. of the IEEE NorChip Conference*, Nov. 2000.

[12] A. Jantsch, J. Soininen, M. Forsell, L. Zheng, S. Kumar, M. Millberg, and J. Öberg, "Networks on Chip", *Workshop at the European Solid State Circuits Conference*, Sep. 2001.

[13] L. Benini and G. DeMicheli, "Powering Networks on Chip", *Proc. of the 14th Int. Symp. on System Synthesis*, 33-38, Oct. 2001.

[14] F. Catthoor, D. Verkest, and E. Brockmeyer, "Proposal for unified system design meta flow in task-level and instruction –level design technology research for multi-media applications", *Proc. 11th Int. Symp. on System Synthesis*, 1998, 89-95.

[15] P. Panda, N. D. Dutt, and A. Nicolau,, "Local Memory Exploration and Optimization in Embedded Systems", *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems* 18, 1 (1999), 3-13.

[16] A. Hemani et. al. , "Lowering power consumption in clock by using Globally Asynchronous Locally Synchronous Design style", *Proc. of Design Automation Conference*, 1999, USA.

[17] Yi-Ran Sun, "Simulation and Performance Evaluation for Network on Chip", *MSc thesis*, Dept. of Microelectronics and Information Technology, Royal Institute of Technology, Stockholm.

[18] Dinesh Pamunuwa et. al., " A study of Physical Issues in the design of an on-chip regular communication network", *submitted to DAC 2002*.

[19] V. Leppänen, Studies on the realization of PRAM, *Dissertation 3*, TUCS, University of Turku, 1996.

[20] M. Forsell and S. Kumar, Virtual Distributed Shared Memory for Network on Chip, *Proc. of the 19th IEEE NORCHIP Conference*, Nov. 12-13, 2001, Kista.

Included Publications

Paper 8

Evaluating NoC communication backbones with simulation

Rikard Thid, Mikael Millberg, and Axel Jantsch
Royal Institute of Technology (KTH), Electrum 229, SE-164 40 Kista, Sweden
{thid,micke,axel}@imit.kth.se*

Pages: 27-30

Reference in Thesis – [Thid2003]

Evaluating NoC communication backbones with simulation

Rikard Thid, Mikael Millberg, and Axel Jantsch
Royal Institute of Technology (KTH), Electrum 229, SE-164 40 Kista, Sweden
{thid,micke,axel}@imit.kth.se*

Abstract

This paper describes a Network on Chip simulator that was developed to evaluate our NoC architecture Nostrum. It is shown how SystemC's features for communication refinement is used to make a highly flexible simulator. The simulator is reconfigurable so that it is possible to try different NoC platforms and different mappings of workloads. In addition to the modeling of our Nostrum architecture, a bus-based architecture is modeled as well, and the performance for a simple workload model is compared.

1. Introduction

In the SIA silicon roadmap[1], it is predicted that the increase in chip capacity will continue for at least another 8-10 years and it will be possible to integrate systems with billions of transistors on a single chip. Current System-on-Chip (SoC) methodologies do not offer the required amount of reusability to enable system designers to meet the ever increasing time-to-market constraints. The desired future SoC methodology should enable, not only, reuse of traditional IP-cores but also communication infrastructure. Current bus-based platforms suffer from limited scalability and poor performance for large systems. In order to overcome these problems several approaches for networks on chip [2, 3, 4, 5, 6, 7] have been proposed. They allow reuse of the communication infrastructure among many products thus reducing the design and test effort and the time to market. In order to enhance reusability and to ease programmability, most NoC proposals recommend standardized and layered communication protocols for communication among cores.

The performance of busses versus NoC is mathematically analysed in [8]. A problem that arise when simulating NoC platforms is how to co-simulate the network with the rest of the chip. Our solution is based on the channel based communication model that is used in SystemC. This paper provides an simulation based comparison of busses and Nostrum to demonstrate the possibilities with the flexible NoC simulator.

The rest of this paper is organized as follows. Section 2 presents our NoC platform named Nostrum. In Section 3, the NoC simulator is presented. The modeling of workload models is described in Section 4. Section 5 explains and analyses the experiments that we perform. The last Section concludes the paper.

* This work is a part of the joint Finnish-Swedish EXSITE research program. This work was sponsored by TEKES, VINNOVA, Nokia Oyj, Ericsson Radio Systems AB, and Spirea AB Kista.

2. Nostrum

The overall purpose with a NoC platform is to act as host for a system that performs one or several tasks with hardware components. In the *Nostrum* architecture, the system is mapped to a set of *Resources*. A Resource is in this context a microprocessor, a memory, a FPGA, a digital signal processor, or an I/O - resource. An I/O-resource is a device that is connected to the chip's pins for the purpose of external communication. The Resources are physically organized in a two-dimensional mesh structure as depicted in Figure 1.

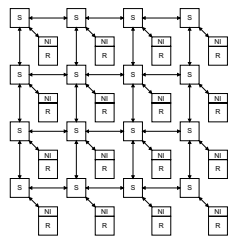


Figure 1. A Nostrum NoC with 16 Resources and switches.

All Resources are equipped with a *Network Interface*, which connects to the network in order to provide services for Resource-to-Resource communication.

The Switches route packets through the network using a hot-potato routing algorithm [9], which reduces the need for buffers within the switches. This is an attractive property for us since we want the area overhead to be as small as possible. Because of the ever increasing wire delays, it is desirable that information travels as short distances as possible. Therefore, Nostrum does not use a centralized router/arbiter such as many bus-based architectures. Instead all the routing decisions are made locally in the switches.

The Nostrum architecture is designed in a layered fashion: this allows us to partition functionality onto dif-

ferent layers, inspired by the OSI reference model. Notice that this does not necessarily mean that different layers are dealt with by different pieces of hardware or software. Instead we can merge functionality from different layers into the same piece of HW/SW. An *Entity* is the unit that implements the functionality of a layer. An example of an entity is a switch, which performs *Network layer* functionality. The way that the entities are interconnected is called *topology*.

A two dimensional mesh topology is used since it is mappable to two dimensions. This is due to the physical constraints of a chip, which does not allow more general topologies such as high-dimensional hypercubes.

3. Simulation environment

3.1. SystemC

In order to evaluate our Nostrum architecture, we have developed a SystemC based simulator. SystemC [10] is a superset of C++ targeted at simulating whole systems with both hardware and software components. In this paper, we will only deal with SystemC's properties as a simulation language.

Models in SystemC basically consist of *modules* whose behaviour is defined in C++. Each module has any number of *ports* that it uses to interact with other modules.

In order to cope with the increasing complexity of communication, SystemC (from version 2.0 and forward) has the ability to organize the communication into *channels*. Channels have interfaces that the modules use to communicate through. An example of ordinary channel is dedicated wires as depicted in Figure 2a. Hierarchical channels can connect to any number of modules and implement several other, non-hierarchical channels. A hierarchic channel could be for instance a bus (as in Figure 2b) or a NoC infrastructure.

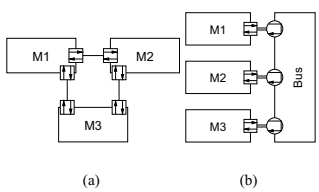


Figure 2. Three modules that are interconnected by (a) several primitive channels or (b) one hierarchical channel.

3.2. The simulator

The simulator is divided into an *Application Domain* and a *Communication Domain* as depicted in Figure 3. The system is distributed into Resources that are modeled in SystemC by *Resource models*. The purpose of them is to generate traffic so that the behaviour of the network for a given workload can be studied. These models interact by sending and receiving messages over the communication platform. The placement of the Resources is managed by the *Resource mapper.* A designer can easily change the mapping of Resources since all mapping is done in the Resource mapper, and no other part of the simulator is directly affected by the mapping.

The communication domain consists of models of entities that implement various layers. Four layers are represented in the simulation environment, namely the Transport(TL), Network(NL), Data link(LL), and the Physical layer(PL).

A topology generator is used to instantiate and connect entities with each other. The simulator has one topology generator that creates Nostrum models of arbitrary size. A small 2x2 example is depicted in Figure 4. There is also a bus topology generator that uses the same TL, LL, and PL entities as the Nostrum generator. However, while the Nostrum NL entity models a hot-potato routing algorithm, the bus NL entity uses a round-robin arbitration scheme.

In order to interface with SystemC models of Resources in a natural way, our simulator is a hierarchic channel. Any SystemC modules that will use NoC for communication does so using an *interface*. This interface features, the opening of channels, blocking, and non-blocking send and receive primitives. No knowledge of the platform is necessary when writing the Resource models, since all communication is handled through interfaces. This enables a user to change the network model but still use the same workload models.

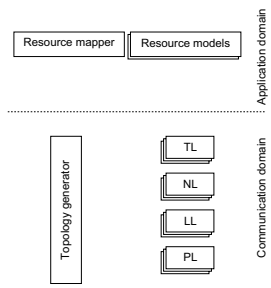


Figure 3. The main components of the simulation environment.

4. Workload models

In order to study our Nostrum architecture, we designed a simple workload model using SystemC. As the communication platform interface is very small, it is easy to write and integrate a simple workload model to the

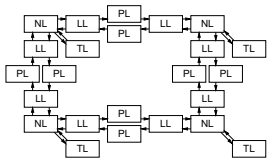


Figure 4. Entity interconnection in a 2x2 mesh topology.

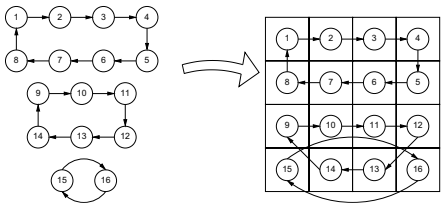


Figure 5. Workload model and its mapping on a 4x4 mesh.

simulator. Our model consists of 16 identical Resources that are logically interconnected in three rings as depicted in Figure 5a. The Resources were placed so that the number of hops needed to send packets is reasonably low (but not optimal). The placement is depicted in Figure 5b. The Resources repeatedly sends a packet, then wait a random number [1] of clock cycles before sending again. This kind of behaviour is what a pipelined signal processing application could look like. In average each Resource sends a message every ninth clock cycle. With a Resource clock frequency of 1 Ghz, approximately 111 million messages/second will be sent through the network interface. As the network interface need only one clock cycle per message, it may be possible to run the communication network on a lower clock frequency than the Resources, possibly saving power. Thanks to the flexible SystemC simulation engine it is easy to scale the frequency of the network.

5. Experiments

The purpose of the experiments is to determine how efficiently our Nostrum architecture can perform given the workload model previously discussed. It is also interesting to see how the Nostrum architecture performs in relation to a bus-based architecture. What are the differences in latency and what clock-frequency is the lowest that can be used without data loss due to low throughput?

[1] Normal distribution, mean value 9, standard deviation 2.

5.1. Clock frequency scaling

In order to minimize the power consumption on chips, the clock frequency is generally kept as low as possible. Most of the power in CMOS is consumed during the switching of logic gates. This is called the dynamic power consumption and it is expressed with the following formula:

$$P_{dyn} = C_L \cdot V_{DD}^2 \cdot f \cdot \alpha \quad (1)$$

The power consumption (P_{dyn}) depends on the capacitive load (C_L), the supply voltage (V_{DD}), clock frequency (f), and switching probability (α). Since an increase in clock frequency requires the supply voltage to be higher, we wish to run the clock as slow as possible. In a network, we should not drop the frequency to low since the performance will be lower and packets may be dropped. The lowest possible clock frequencies for the two discussed architectures for a specific workload are investigated .

5.2. Method

The simulator was configured to run the Nostrum and the bus-based architectures with the same workload models. The TL-entities, which are the highest entities in the protocol stack and therefore experience the most latency, measure the average and the maximum time between sending and receiving of data. The entities are configured to report any loss of data that occurs when more data is put into the system than it can handle. Multiple simulations were run with different clock frequencies for the respective communication infrastructures ranging from 125 MHz to 4 GHz. The Resource models were always running at 1 GHz and therefore the same amount of data was attempted to be sent. The average and the maximum latencies were recorded for each simulation. The lowest possible frequency for each platform was noted.

The PL-entities, which represent the links between wires measure how frequently data is transmitted over them. This information represents α in Formula 1, and together with the clock frequency the power can be calculated.

5.3. Result

The results from the simulations are plotted in Figure 6 and 7. For the simulations with too low clock frequencies, the latencies are very high and they are not included in the plots.

For the bus architecture, a bus clock faster than 1.8 GHz is required, and the Nostrum only needs 200 MHz to handle all data. It is natural that the latency decreases with longer clock periods. However, as shown in the right figures in both cases the latency is not fixed in terms of clock cycles. In the bus-based architecture the decreased efficiency is explained with that packets may have to wait until they become routed since only one packet is routed each clock cycle. In the Nostrum architecture it is possible to route four packets in each switch simultaneously.

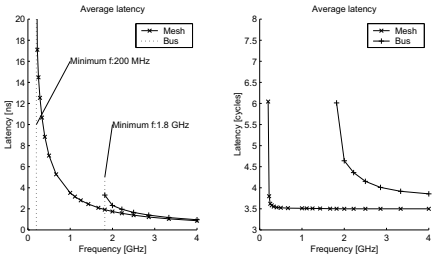


Figure 6. Average latency.

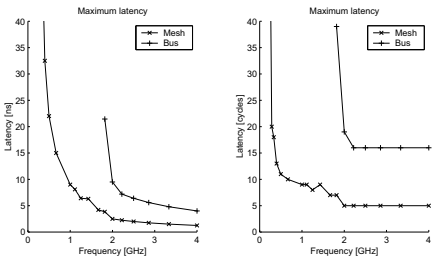


Figure 7. Maximum latency.

The decrease in efficiency is explained by the congestion that occurs when many wires are occupied. Figure 8 shows how occupied the links between switches are, and how this affects the link power consumption. The power figure is calculated with Formula 1, with V_{DD} and C_L constant.

6. Conclusions

It was demonstrated that NoC simulation can be done in a flexible manner thanks to the channel based communication model in SystemC. Two different communication platforms were modeled with the same workload model and it was shown that our *Nostrum* platform can operate at a much lower clock frequency than a shared bus platform. Topics that are interesting for future works include to study the impact of more workload models and comparison with other NoC platforms than Nostrum.

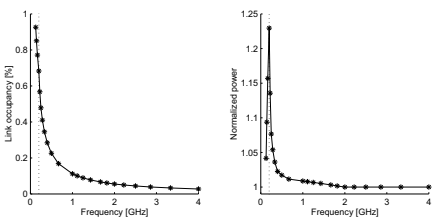


Figure 8. Link occupancy and power in Nostrum wires.

References

[1] "The international technology roadmap for semiconductors," International SEMATECH:Austin, TX., 2001.

[2] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *In Proceedings of Design, Automatation and test in Europe*, pp. 250–256, 2000.

[3] G. D. M. Luca Benini, "Powering networks on chips," in *Proceedings of the international symposium on Systems synthesis*, pp. 33–37, 2001.

[4] L. Benini and G. D. Micheli, "Networks on chips: A new SoC paradigm," *IEEE Computer*, pp. 71–78, January 2002.

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

[6] K. Goossens, J. van Meerbergen, and P. Peeters, A.and Wielage, "Networks on silicon: combining best-effort and guaranteed services," in *Design, Automation and Test in Europe Conference and Exhibition, Proceedings*, 2002.

[7] W. J. Dally and B. Towels, "Route packets, not wires: On-chip interconnection networks," in *38th Design and Automization Conference*, 2001.

[8] C. a. Zeferino, M. E. Creutz, L. Carro, and A. a: Susin, "A study on communication issues for systems-on-chip," in *Proceedings of Integrated Circuits and Systems Design (SBCCI)*, 2002.

[9] U. Feige and P. Raghavan, "Exact analysis of hot-potato routing," in *Foundations of Computer Science, Proceedings*, pp. 553 – 562, IEEE, 1992.

[10] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Kluwer Academic Publishers, 2002.

A Nostrum

The purpose of this chapter is to give an orientation and a coherent picture of the latest version of the Nostrum Network on Chip communication platform. Nostrum is referred to in most of the articles included in the thesis but since the platform is constantly under development it will change slightly from one paper to the next. In addition, the terminology used will also change over time. This, since new ideas are developed but also due to the Network on Chip community has matured and adapted to a more or less coherent terminology. The Nostrum and the chapter about Semla – our Network on Chip Simulator will complement each other and – to some extent – overlap. The Nostrum chapter starts with a brief overview of the platform that is followed by some historical notes on the motives and initial requirement that were set on the platform to be developed. Further I will discuss the Nostrum layering and end the chapter with some information about the "real" implementation of Nostrum that has been done.

A.1 Platform Overview

The Nostrum Network on Chip (NoC) is a mesh based NoC, that supports both Best Effort traffic (BE) as well as Guaranteed Throughput (GT). The cornerstones when designing Nostrum has been to create a network with small switches to save power that employs a no-packet-drop policy.

The goal of having small switches is achieved by employing deflective routing, with no additional buffering, for the Best Effort traffic. For the Guaranteed Throughput, a TDMA-based scheme is used where Virtual Circuits can be set up by explicit switch configurations.

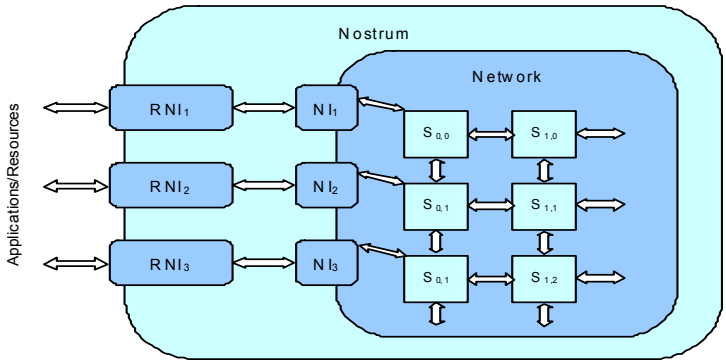


FIGURE A.1. Nostrum Schematic Overview

In Figure A.1 a schematic overview of Nostrum can be seen. The core network consists of switches connected in a grid fashion. To every switch, a Network Interface (NI) is attached. The purpose of the Network Interfaces is to provide the service of address translation and attachment of routing information. This, since the application does not know about the location of its communication peer – only its address, whereas the routing process, naturally, needs the actual location. Internally, the network interface lacks any explicit buffers for the packets to use – instead it is equipped with grant signals that are sent upstream. A packet admission grant means that any entity communicating with the NI is guaranteed to have its packet delivered to the network that very clock cycle.

On top of the Network Interface, there is the Resource Network Interfaces (RNI) which provides packetisation, traffic shaping, as well as buffering. The NI has a "standard" interface (standard to Nostrum that is) whereas the RNI could be equipped with an interface suitable for any resource that wishes to utilise the network. The communication between the RNI and the NI is packet based and the communication between the RNI and the applications are Message based (In the current implementation the Message sizes coincide with the packet sizes – i.e. no packet segmentation or desegmentation is implemented).

In order to conceptually understand the RNI a brief description of one interface that has been implemented can be given. The interface implemented in the RNI is a minimal form of message passing with the following functions

channelId = openChannel(channelType, source, destination)

status = read(channelId, message)

status = write(channelId, message)

When a channel is opened the channel type together with the source and the destination is given. The channel type can be either of a Best Effort kind or a Guaranteed Throughput (GT) kind. The GT can be implemented as a Virtual Circuit inside the network.

One of the features of Nostrum is the facility of implementing four truly disjoint networks. All channels will belong to one or more of these networks. The separation of the networks is based on a TDMA scheme that we have chosen to call Temporally Disjoint Networks (TDN) – see *Network Transport* on page 178. The benefit of having four logically separate networks is that different kinds of characteristics can be given to these individual networks. The assignment a channel to a particular network is defined within the channel type.

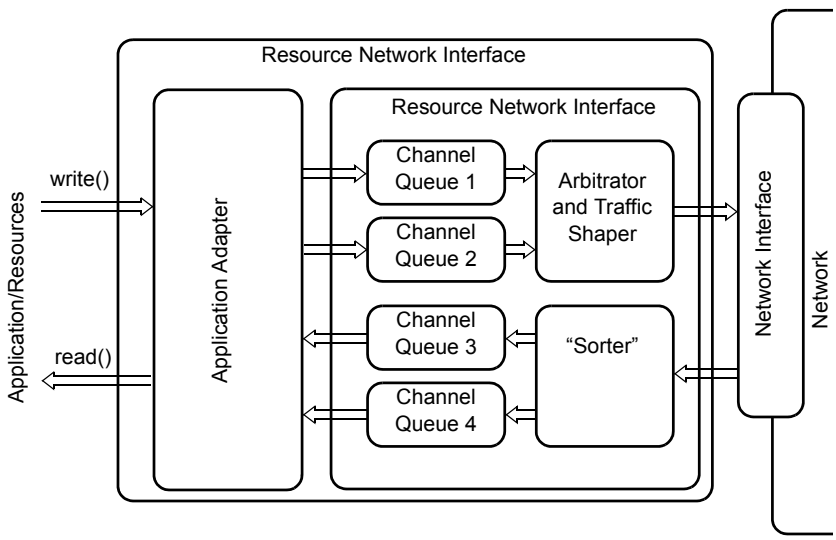


FIGURE A.2. Nostrum – Resource Network Interface

Conceptually, all buffering of packets is done within the RNI, and logically all channels are having their own separate packet queues. The arbitration in admitting packets to the network between the channels is based on channel type. A packet that belongs to a channel characterised as GT is automatically tagged as a high priority packet. Inside the network, the performance/detailed characteristic of a Virtual Circuit is defined by an explicit switch configuration – see Figure A.3. This configuration regulates, in detail, how packets are switched within a particular TDN. The process of setting up these Virtual Circuits, i.e. configuring the switches, is by no means automatic but is an explicit configuration of the network at start-up.

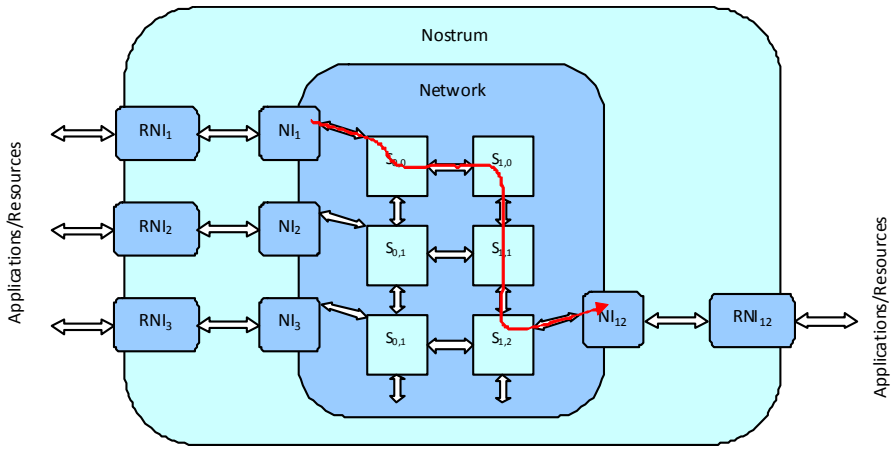


FIGURE A.3. Nostrum – Virtual Circuits

A.2 SETTING UP COMMUNICATION – PROCESSES, CHANNELS, PORTS AND THE MAGIC OS

From the Network's perspective, all communicating entities in the system are denoted *Processes* (Even the term *Atomic Communicators (ACs)* has been used – see *The Process of Mapping and Setting up Communication* on page 199). All Processes are identified by a Process Identifier – a *Pid*. When a network is instantiated/created all the Network interfaces of the network are given a unique *Network Interface Id* – a *Nid*. In a fully dynamic network implementation, all Processes have to register to their respective Network Interface during the set-up phase to make the network aware of their presence and location. If the system is likely not to change a more static network implementation can be chosen – the presence and location of the Processes are then hard-coded into the system at network instantiation time.

All communication over the network is done in the context of *Channels* – a channel is a communication link between two processes with a well defined service level. The Channels are identified by a *Channel Identifier* – a *Cid*. The Cid does not have to be system wide unique – only unique to the Network Interfaces hosting it. Once the network has become aware of the Processes in the system, a *channel opening requests* can be made. A channel opening request involves a sender process id, a receiver process id, channel information data from the upper layers, the service level, and – possibly – a port id. The channel information from the upper layers can be information about the message to be sent, sending policies and other meta information on the format of the data that is going to be transferred. The network will process as incoming requests and grant or decline them; in the case that the request is granted a channel identifier is returned. Depending on the knowledge about the system at design and instantiation time the process of granting open channel request can be more or less complex. An implementation of a system that have been thoroughly analysed and dimensioned during the system design time will require very little from the network to grant channels; the granting merely becomes a notification to the receiver that a communication peer process will have a certain channel id together with a registration/query to the networks "yellow pages". The yellow pages is part of a centralised minimal Network Operating System (NOS) that holds a database that keeps track of the processes in the system and their location. It also holds information about the channels and is responsible for any configuration of the switches in the system. For a system that is likely to change and/or have undergone a very coarse system analysis the channel opening procedure can be quite complex. The reason is that a system that is not static has to be able to cope with very complex decisions on how to administrate its resources to be able to give certain guarantees. In this case, the network has to be equipped with some sort of more intelligent NOS. This NOS has to be capable of analysing all incoming requests during the configuration phase of the network and act accordingly.

In the current implementation, the NOS has not been implemented fully. Hence, the internal whereabouts of the current NOS is carried out in a "magic" way to give the illusion of an implementation with full analysis capabilities. In concrete – any query that is sent to the NOS in the current implementation is resolved by all function call to the simulator that processes the query and responds accordingly. In a full implementation the NOS would have to reside in a resource of its own and any query (and response) would be sent to this entity as packets over the network.

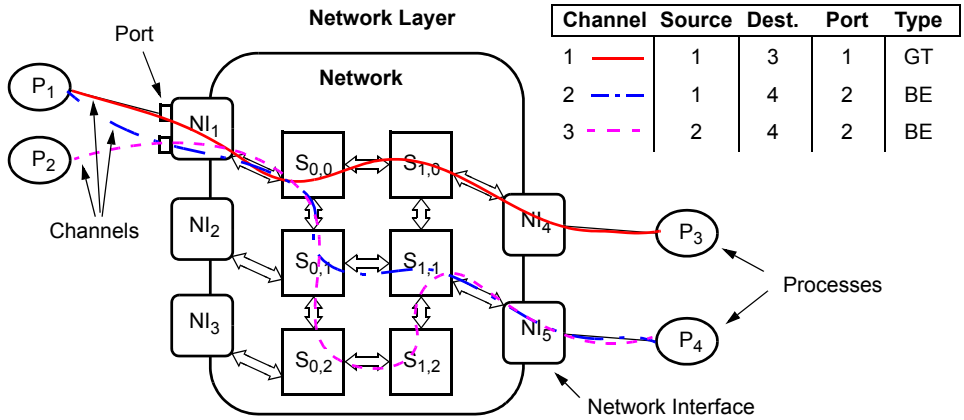


FIGURE A.4. Nostrum – Network Layer

In connection to the channel opening the concept of ports were mentioned. The port is a construct to aid in the process of orchestrating the granting of packet admission in the network interfaces that holds several channels. The Resource Network Interface (RNI) connected to a particular Network Interface will most likely host a set of queues for packets that awaits admission to the network. These queues will be associated with channels. Without the use of ports the Network Interface would have to explicitly notify the RNI which channels that are granted admission to the network the next clock cycle. Since several channels may belong to the same service class – hence would allow access simultaneously – the interface would become very complex. The solution is the concept of a port.

The port as a concept is an explicit association between a set of channels and a set of ports. If two (or more) channels are associated with the same/a similar service class they can be associated with the same port. The interface of a single port is simply a wire – one bit – that will notify the RNI that the channels associated to this port is granted access to the network the next clock cycle. In Figure A.4 this is illustrated with the channels 2 and 3 that are connected to the same port. Both channels belong to a best effort service class and can, in this particular example, share admission to the network. Channel 1 in this example will have a port of its own due to other requirements. This has the consequence that the channels belonging to the same port also could share a queue in the RNI. Despite how it is illustrated in Figure A.4, please note that the concept of ports only provides means of granting access to particular channels – *the individual packets of the different channels will have the same, single, physical entry-point to the NI!*

A.3 SOME HISTORICAL NOTES

Our work on a generic communication platform for System on Chip communication began around 2000 with the motivation and hope that a Network on Chip with clear interfaces would have advantageous properties

- Enhance and encourage reuse
- Enable a parallel design-flow
- Make the system easier to scale
- Give benefits to the next product generation

Particularly, when our Network on Chip platform Nostrum was created the following objectives were targeted – the Nostrum architecture should be

Layered

It should be natively layered to its nature with clear interfaces to enable all the advantages above. This is relatively "easy" accomplished – it is simply a matter of consistency and design style.

Reliable

It should be reliable – no packets should ever be dropped.

Minimal

The switch footprint should be minimal to minimise the area and power consumption. Deflective routing enables the use of a minimum number of buffers in the switches but requires reordering facilities for multi-packet messages in the network interfaces. Many research groups have later recognised the demand of a small switch design e.g. Jingcao Hu and Radu Marculescu in *Application Specific Buffer Space Allocation for Networks on Chip* ... [Hu2004]. In addition, it has been shown by Arnab Banarjee et al. that routers dissipate significant idle state power and that the dominating source is the data-path [Banerjee2007]. The leakage power is roughly proportional to the complexity – that is the gate count. The dynamic power grows approximately linearly with the number of synchronous element in the design. Hence, it is of importance to keep the gate count down and – in particular – the number of buffers used. The low dimension network is justified by assuming a strong locality in the traffic patterns [Govind2006].

Provide Best Effort and Guaranteed Bandwidth Service Support

The network should be capable of offering both best effort and guaranteed bandwidth services. This since SoCs have needs for both single-message passing between Resources without set-up times for e.g. synchronising messages as well as for stream oriented data transport with requirements on latency and/or throughput.

The demand of a reliable and minimal platform in combination with the Best effort and guaranteed bandwidth service support required some thinking. Since these requirements have different implications for the different service scenarios. In addition, segmentation is needed because the possible mismatch between message size and the packet payload size, i.e. the message might have to be sent as several packets due to a large size.

A.3.1 Best effort

The characteristics that were further decided desirable for the best effort traffic service class were the following

- No Set-up time
- Datagram based – to enable fast adoptions to changes in the Network
- Connectionless

Since packets can neither be dropped nor queued in the switches the only realistic alternative is to use a deflection routing policy that permits the forwarding of packets in a – potentially – wrong direction. The deflective routing policy also adheres to the additional requirements set up for the best-effort service class. To simply drop packets is not an option either – as shown by Tomas Lang and Lance Kurisaki [Lang1990]. They discovered that the discarded traffic was reissued and followed the same path that was congested. Except for not being effective it also requires that the source buffers the outgoing traffic until it is acknowledged and requires the protocol to signal that a particular packet was dropped, and retransmission is needed. Instead they implemented a diverting routing scheme where packets got diverted to intermediate destinations before being sent to the final destination. Furthermore, the policy of deflection routing may suffer from problems associated with the need for reordering of packets at the receiver side if the individual order of packets is of importance (See “Deflective Routing (or Hot potato)” on page 68).

A.3.2 Guaranteed Bandwidth/Throughput Services

For the guaranteed bandwidth services, a set-up time is necessary and hence acceptable. The data transport should be handled by some sorts of Virtual Channels (VC) that are set up between heavily communicating peers. Once the VCs are set up the communication can be relatively fast thanks to predictability and deliver a steady data flow. In Figure A.5 an example communication graph and the corresponding mapping of VC's are depicted.

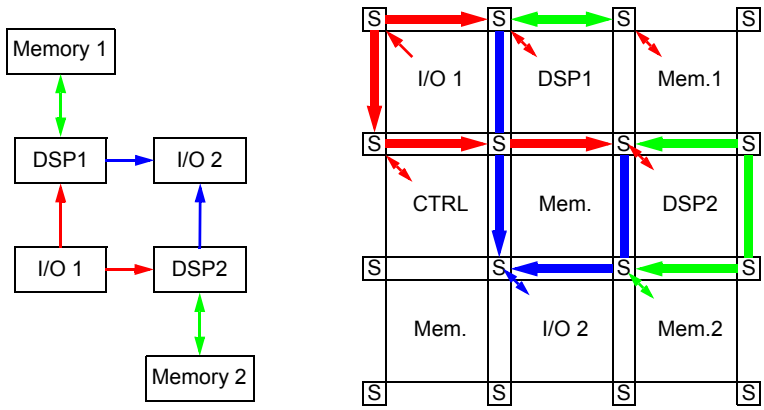


FIGURE A.5. Nostrum – Virtual Circuits – an Example

Since the best effort traffic is utilising a deflection routing scheme and is going to utilise the same network as the guaranteed services, they must be able to coexist. This in combination with the requirement of small footprint switches effectively rules out the use of e.g. worm-hole based routing strategies and suchlike. The solution that came up is based on the idea to use ordinary deflection routing packets as containers for the guaranteed service data. The container packets would have the highest possible priority and get routed along predefined paths/routes from source to destination. This is further described in *Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip* [Millberg2004a]

A.4 NOSTRUM LAYERING

In Nostrum the interfaces between the layers are called Service Access Points and act as contract brokers between the layers. An SAP provides a set of Services and, at the same time, requires the user of the service (the Service User Entity) to guarantee a set of services. The users of the services are called *entities*. Entities communicate with other entities on the same layer using Protocol Data Units (PDUs). The Protocol used is always separated from the service it implements to enable a variety of implementations of the same service. These implementations can have different characteristics/footprint sizes as well as being more or less refined to work as a vehicle for rapid parallel product development. The information in the PDU used by the layer needed to carry out its serve is called Header information and the data to transmit is called the Payload [Millberg2004b].

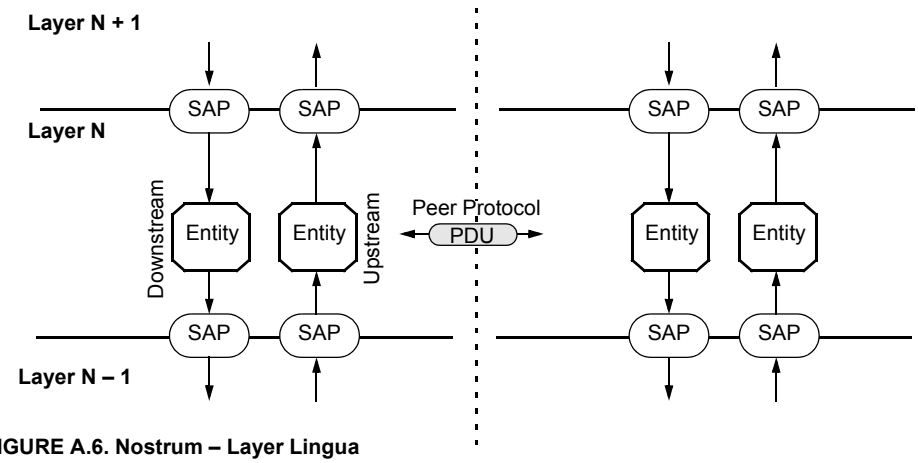


FIGURE A.6. Nostrum – Layer Lingua

The layers that are explicitly utilised in the Nostrum platform are Upper and lower transport Layer and the Network Layer. The Data link and the Physical layers are defined as well but merely work as layers used within simulation. The PDU of these two lowest layers is simply payload.

- Physical – Models transmission

  Only for error and delay insertion for simulation purposes

- Data Link – Ensures reliable communication over the physical channel

  Determines the structure of data, i.e. Frame size

- Network

  Interconnects data link communication paths on the network, i.e. it handles the routing from source to destination

  Provides flow-control

- Lower Transport Layer

  Segmentation, desegmentation and queuing

- Upper Transport Layer

  Bookkeeping and Administration of Channels

  Application Protocol Adaptation

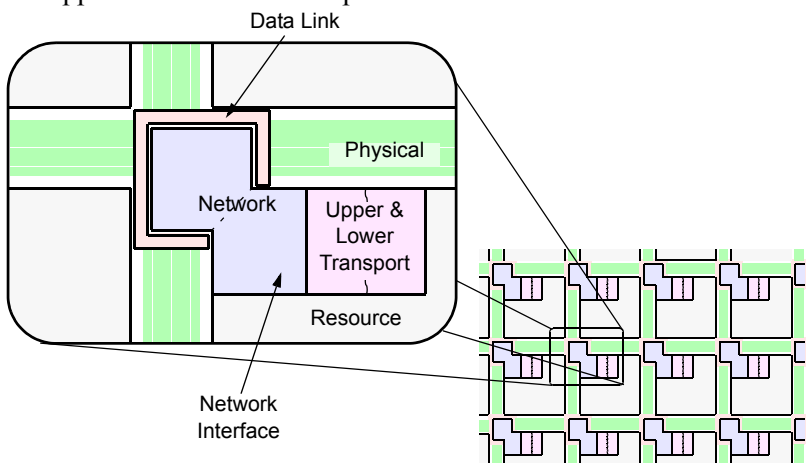


FIGURE A.7. Nostrum – Layer Layout

A.4.1 Network Layer

In Nostrum the network layer is implemented in the Switch and in the Network Interface (NI) as seen in Figure A.7. The service it provides to the upstream layers is the transmission of “Packets” from source to a destination address. The destination within the network layer is identified by a Network Address – basically an (x,y) pair coordinate in the mesh. The reason why packets are within quotation marks is that the actual write() call from the upstream SAP of the Network layer is done with two separate arguments – the Channel Identifier (CID) and data – and not a single packet argument as depicted in Figure A.8.

More on the channel identifier in Section "Setting up communication – Processes, Channels, Ports and the magic OS," on page 164. The call from the upstream SAP of the network layer is actually a conditional call – the caller is always notified on the clock cycle before the Network layer will accept a write from one or more specific channels. The contract of the SAP is defined in that way that the RNI connected to the NI is obliged to accept any incoming packets from the Network since the Network Interface does not have any buffering capacity.

The PDU of the network layer is called a packet that has a header that consists of a destination address and Routing Enhancing Information (REI) like e.g. a hop-counter. The actual REI is dependent on the current configuration of Nostrum and has varied over time as new strategies for routing have been tried out.

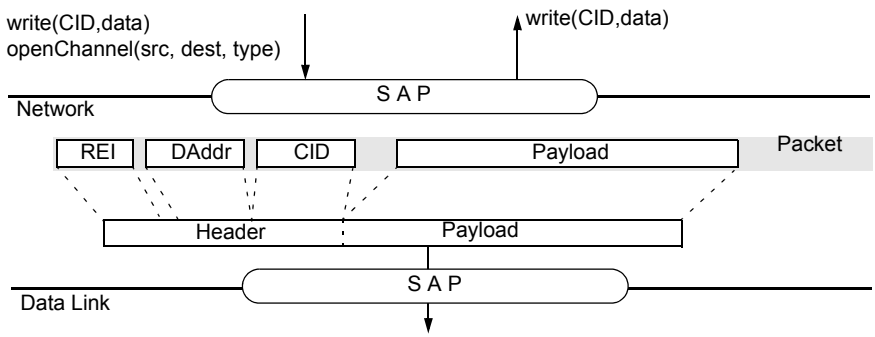


FIGURE A.8. Nostrum – Network Layer SAP, and PDU

A.4.2 Transport Layer

Traditionally, the Transport Layer provides transparent transfer of data between end users, providing reliable, error-free virtual point-to-point connection data transfer services to the upper layers. The transport layer controls the reliability of a given link through flow control, segmentation/desegmentation, and error control. In Nostrum, it also is responsible for the bookkeeping of the channels.

The implementation of the transport layer in the RNI of Nostrum is application specific. The transport layer is implemented in two parts – the Upper and the Lower Transport Layer – see Figure A.9. The lower transport layer consists of a configurable *Nostrum Add-on protocol stack* and the upper transport layer is an *Application Adaptor*.

The application adaptors work as proxies for the application to implement the desired application specific interface – e.g. AXI, MPI, etc. [AXI and MPI1993].

In Nostrum the RNI is the custom hardware/software used to connect the *Nostrum* backbone protocol stack with the communication protocols used by the Resource. The boundary between the applications and the RNI is, however, not regulated within the *Nostrum* architecture. The implementation of *Nostrum* protocol stack is configurable, i.e. if the application only requests a very basic functionality the stack can be more or less shallow.

Lower Transport Layer

The purpose of the Lower Transport Layer (LTL) is to transport *messages or streams* from source to destination. The payload of the messages passed to the SAP is segmented into appropriate packet payload sized chunks to be forward to the SAP of the Network layer. The upstream SAP interface will also provide a Channel identifier and a Message id as seen in Figure A.10.

The PDU of the transport layer is called a Lower Transport Layer PDU (LT-PDU) packet that has a header that consists of Channel Id (CID), Message Id (MID), Packet Sequence Number (PSN), and Payload. Since the implementation of the transport layer is configurable the actual PDU and SAP may be different from one implementation to the next.

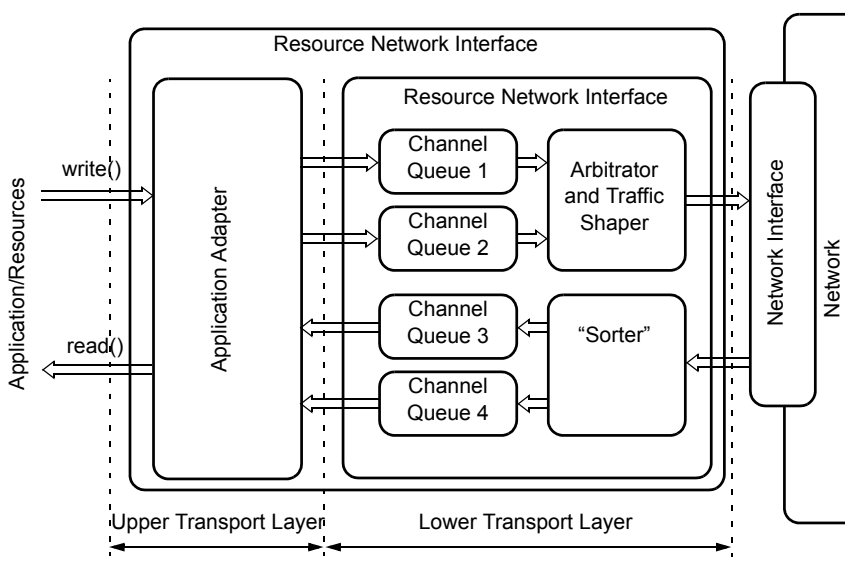


FIGURE A.9. Nostrum – Transport Layer

The format of the messages being sent over a channel is either, hard-coded at instantiation time of the network, or included as channel information data in the initial open channel request. The channel information data cannot only describe the actual format but also explicitly state buffer requirement at the receiver side, data coding formats and suchlike.

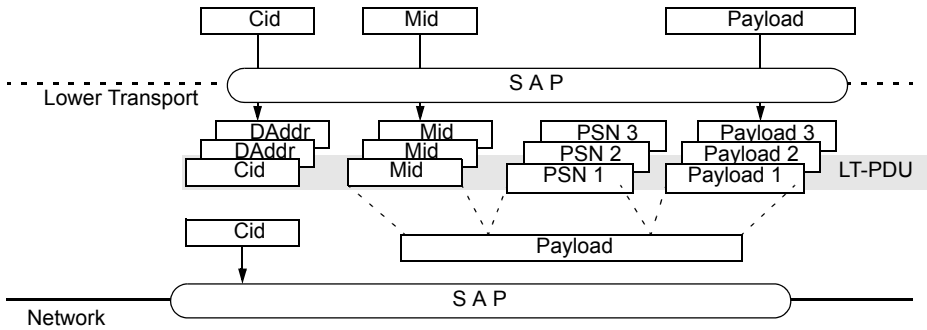


FIGURE A.10. Nostrum – Transport Layer SAP, and PDU

Upper Transport Layer

The purpose of the Lower transport is to work as an adapter between the Nostrum protocol stack and the Applications. It provides an error-free point-to-point connection between Processes and is responsible for the administration, maintenance, and book-keeping of channels. It also holds and maps information about the sender and receiver processes relevant to this particular application/resource. The Upstream Service Access Point will solely depend on the protocol that it implements. Within the Nostrum project, an MPI based implementation has been tried out. Moreover, an AXI/AMBA adapter has been implemented with success.

A.5 Network Admission, Routing and Exit

Any packet transported over a network will conceptually have to undergo three phases: *Admission, Routing* and *Exit*. All three phases are critical to good network performance. To gain a better understanding in the nature of these phases within the concept of Nostrum a few words need to be said about the Switch and Network Interface. Inside the network part of Nostrum there, conceptually, exist two separate stages – *Ejection* and *Routing*.

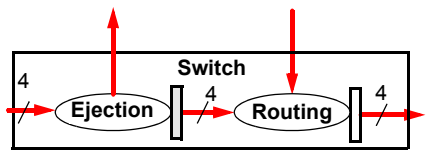


FIGURE A.11. Ejection and Routing Stages

Ejection

The ejection stage examines the incoming packets to detect if one (or more packets) has reached destination and is to be delivered to the Resource. In case of competition the packet with the highest priority is delivered. Furthermore, it informs the RNI, if there is possible for a packet to enter the network the next clock cycle. Since no explicit queues are employed in the network, admission can only be granted if the Switch is currently holding fewer packets than its output buffer capacity, i.e. four packets.

Routing

The deflective routing scheme is (conceptually) carried out in three phases:

Priority Assignment (1) In this phase the incoming packets are dynamically assigned a priority such as the *Hop Count* for example.

Favoured Outport Selection (2) The packets now use their assigned priorities to select a desired out-port. The priorities are utilised as credits, that enable the packets to give different weights to favour certain routing decisions in the *Permutation Routing* objective function.

Permutation Routing (3) The weighted priorities of all the competing packets are summed to form the basis for selecting the best routing permutation. For more information on this see *Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy* [Millberg2007b]

A.5.1 Network Admission

The process of network admission is a problem encountered in any network with limited buffer space and a no-packet-drop policy. The basic problem is that any packet that goes into the network also must come out; if it does not do so – at a sufficiently high speed – the network will get full and can no longer accept new packets. If the problem is seen from a more local perspective the limitation of one single switch can be illustrated as in Figure A.12. In Figure A.12(a) the packet that comes from the NI cannot enter the switch, due to the four incoming packets that going to occupy all outgoing links the next clock cycle. In Figure A.12(b) the packet from the NI can happily enter the network, since there will be one link available.

Within Nostrum the solution to the problem of guaranteeing network admission falls into two scenarios. The first is the Network Admission for Guaranteed Throughput, and the second is the Network Admission for the Best Effort scenario.

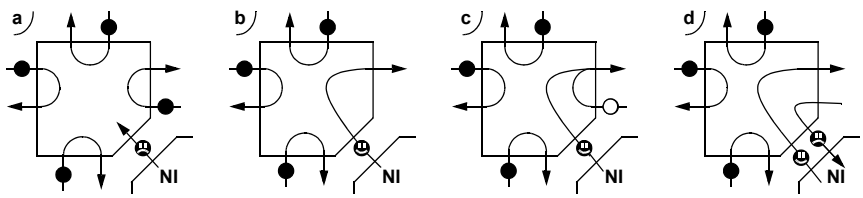


FIGURE A.12. Network Admission

Network Admission for Guaranteed Throughput

The Network Admission has in the Nostrum case been dealt with using the concept of 'looping containers' introduced in *Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip* [Millberg2004a]. In short – outgoing link capacity is available through a container packet. The container packet continuously goes from the source to the destination and back in a closed loop fashion. Once the container packet enters the switch at the sender side it gets loaded and once it gets to the receiver side it is unloaded – see Figure A.12(c). The route of the packet is hard-coded into the switches along the way of the route.

Network Admission for Best Effort

For the Best Effort traffic, the problem of network admission has been transformed into a problem of global fairness. The basic idea is that packets waiting for admission into the network should compete with packets already inside the network. In this implementation, the packet priority is based on age; older means higher priority. Once the packets arrive at a switch they get sorted in priority order. If no packet has reached its destination – the corresponding Network Interface that is – the youngest packet is forcefully taken out of the Network and delivered to the Network Interface – depicted in Figure A.12(d). If there is no outgoing packet from the RNI the packet that was forcefully taken out is reinserted into the switch. If *there is* a packet that is waiting for admission in the RNI that packet is delivered to the switch and the forcefully taken out packet is requeued in the RNI to be delivered to the network at a later time. The concept is called *Forced Requeue* and is described in more detail in *Priority Based Forced Requeue to Reduce Worst-case Latency for Bursty Traffic* [Millberg2009].

Utilisation of the Priority Based Forced Requeue changes the characteristics of the observed system latencies in the system. To illustrate a graph is provided which is a histogram that depicts the latency.

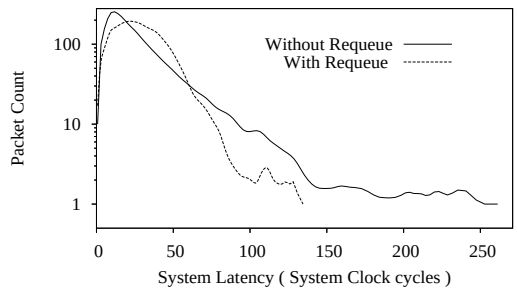


FIGURE A.13. Latency Distribution Shift

As seen in the graph and evident from simulation data the experimental results show a 50 percent reduction in worst-case latency from a system perspective thanks to the reshaped latency distribution. Noteworthy here is that the average latency is kept the same.

A.5.2 Network Transport

Network transport is the process of delivering a packet from its source to its destination *within* the network. Once again, the problem falls into two scenarios – Guaranteed Throughput and Best Effort Scenario. Common for both scenarios are the benefits that come with the concept of the Temporally Disjoint Networks (TDNs). The TDNs are a consequence of the deflection routing policy, the queue-less switches and the topology.

Temporally Disjoint Networks

The idea behind the *Temporally Disjoint Networks* is that a physical network, potentially, can be seen to contain a set of separate networks that a packet can enter dependent on *when* it enters the physical network. A necessary condition for the existence of these TDNs is that a position in the network can only be reached on a multiple of N hops where N must be greater than 1. As a consequence the number of TDNs that exist, N is equal to the number of hops it takes to leave a switch and then get back to the very same switch in such a network.

In our case, these conditions will be fulfilled in a Manhattan network with logically identical switches that performs no reordering of packets. If the network is a torus, the number and rows and columns must be even to render more than one TDN.

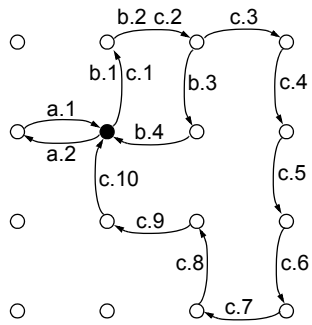


FIGURE A.14. Temporally Disjoint Networks

To illustrate the idea three different routes have been layered out in the Manhattan network in Figure A.14. Their respective path lengths are 2, 4, and 10. Hence, the number of TDNs that exist in this network will be 2. This means that two packets that are sent out consecutively on the network (i.e. in cycle n and cycle n+1) will never be able to collide!

These individual networks could be used to enhance overall network performance in many different ways.

- Different traffic loads in different networks give different Quality of Service behaviours in terms of network throughput, latency and jitter.
- Different spatial mapping of senders and receivers reduces the chances of traffic congestions, i.e. make adjacent receivers reside in different networks

The TDN scheme has smaller packet buffers compared to the alternatives such as rate based routing or deadline based packet switching [Rijpkema2003]. The details of the concept are described in *Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip* [Millberg2004b].

The increased relative hardware cost of implementing the TDN scheme using looped containers is less than 2 percent in term of additional gates. The effective relative payload for a packet with 128 bits is more than 98 percent due to the additional two extra bits to implement the concept. Simulations also show that background traffic in the network is very little affected by the looped containers. The average bandwidth of the traffic assigned to the looped containers is not changed – but now it is guaranteed!

Network Transport for Guaranteed Throughput

The Network transport for the *Guaranteed Throughput* is handled by the looped Containers as described in *Section "Network Admission for Guaranteed Throughput," on page 176*. Virtual Circuits (VCs) are set up between source and destination pairs by configuring routing tables in the switches along the path of the Virtual Circuit. This allows slots to be reserved consecutively in a series of routers. Any packet tagged as a looped container will get routed according to these tables. To enable overlapping VCs they are placed into different TDNs. Figure A.15 depicts two VCs; VC_A with black container packets and VC_B (path dashed) with grey ones. In switch [2,1] and [2,2] the containers of both VCs will share the same links (and switch). The numbers inscribed in the packets, denotes which TDN the respective packet belongs to; the numbers range from one to four since the number of TDNs, in Figure A.15, is four. The reason for having four TDNs stems from the bipartite topology in combination with two buffer stages in every switch. As seen in Figure A.15, VC_A have subscribed to TDN_2 and TDN_3, whereas VC_A only uses TDN_4.

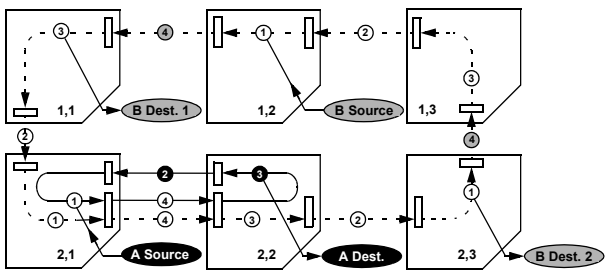


FIGURE A.15. Network Transport – Use of Looping Containers and TDNs

Network Transport for Best Effort

The main problem of the network transport phase is congestion. The problems of congestion are relatively easily solved in lossy networks, e.g. the internet, since packets can be dropped if congestion occurs, thus preventing the build up of saturation (or congestion) trees. For NoC this may not be an option for two reasons. (1) Packet drops will create, potentially, very long delays, which may not be acceptable. (2) The cost of implementation an end-to-end protocol for the detection of lost packets may be too costly. For parallel computer clusters this has led to the use of loss-less networks that are greatly over dimensioned to guarantee a low latency network. However, for NoC this is not acceptable due to the limited power budget.

Within the NoC community, the wormhole routing strategy is the most common solution for the lossless network transport phase. The switches of the wormhole routing have the advantage that they – potentially – can be configured to have very few buffers and hence give a small foot-print. This minimal switch does, however, come with the potential drawback of a bad network utilisation. This as a consequence of that a packet may be blocked. The solution is to add virtual channels in the input and out-put ports but this comes at the cost of increased buffer size [Tota2006]. Despite the flexibility that the use of Virtual Channels gives, this strategy still is unable to avoid congested areas in the network due to the Head of Line (HOL) blocking problem [Duato2005]. Solutions have, however, been proposed, e.g. the switch of José Duato et al. in *A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks* but these solutions give a significant foot-print. The area of the 64-entry Content Addressable Memory (CAM) and the area of the associated logic of their switch is approximately $2mm^2$ in 0.18µ technology [Duato2005]. This should be seen in relation to the 0.28 mm^2 area of the Nostrum switch described in Section A.6, "Nostrum Implementations," on page 184. Obviously there exist smaller implementations of wormhole switches but this seems to be the only implementation that is capable of avoiding the HOL blocking problem.

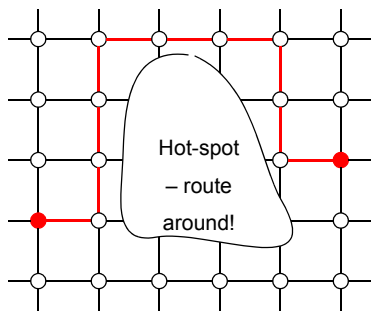


FIGURE A.16. Adaptive Best-Effort Routing using Proximity Congestion Awareness

In Nostrum the network transport phase for best effort traffic is handled by the deflection routing policy. The result is a relatively flexible solution with a small footprint due to the absence of buffers. To further enhance the adaptive characteristics of the deflective routing policy, we introduced the concept of P*roximity Congestion Awareness* (PCA) in *Load distribution with the Proximity Congestion Awareness* [Nilsson2003]. The basic idea is that all switches keep a running average as a measure of the current local network load – a stress value.

The stress value is the sum of the number of incoming packets averaged over the last four clock cycles. This stress value is continuously forwarded to all neighbouring switches; the neighbouring switches can now use this information to favour routing decisions that do not route packets into already congested areas. When utilising the PCA concept a substantial improvement with about 20-time load reductions has been observed. The related hardware cost is very moderate in relation to the benefit. When optimised on speed, an area of 21 029 and a gate depth of 48 was achieved.

A.5.3 Network Exit

The importance and impact of the Exit strategy have not been much emphasized in the collected NoC literature. The most likely reason for this is that the very process of getting information out from the network has been bundled together with the network transport phase. The cause might be the same – the contest for access to shared resource – but the cost and nature of the solution are different. In *Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy* we have shown that the exit phase could potentially be a severe bottleneck, and we also propose a solution to be used in the context of Best Effort [Millberg2007b].

Network Exit for Guaranteed Throughput

The Network Exit for the Guaranteed Throughput is handled "automatically" using the aforementioned *Looped Container* concept.

Network Exit for Best Effort

Isask'har Walter et al. formulated the problem of the network Exit as a bandwidth problem for so called *hot-modules* in *Access Regulation to Hot-Modules in Wormhole NoCs* [Walter2007]. Hot-modules are entities in a system that are bandwidth limited and in high demand by other units. Typical hot-modules could be e.g. DRAM controllers or floating point units. In their work, they claim wormhole-based systems to be very sensitive to hot-modules due to the hop-by-hop back-pressure, associated with wormhole routing. The back-pressure causes buffers at the router close to the hot-module to be filled up and become stalled, blocking new arrivals to this router. This will create a domino effect and saturation tree-to be formed.

The saturation tree-tree is – as the name suggests – a cluster of saturated, or blocked, routers in the constellation of a tree with the hot-module at its root. I claim that this forming of these saturation trees would form in other networks as well.

The solution that we propose in In *Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy* is simply to increase the exit band width of the network and handle the buffering of packets destined to these hot-modules in the RNI instead of in the network [Millberg2007b].

To prove the work of the Dual Packet Exit concept extensive simulations were carried out. For a 4×4 mesh the average system latency is reduced from 14 to 9 clock cycles and the observed worst case latency is reduced from 85 to 45 clock cycles at an injection rate of 0.63. In concrete this means a 50 percent reduction in terms of worst case latency and a 30 percent reduction in terms of average latency as well as an increased throughput both from a system and network perspective. If we set for an average latency of 10 cycles the network with DPE gives roughly 25 percent higher injection rate for the same latency compared to a system without. The validity of the chosen approach is not restricted to uniformly random traffic patterns on meshes but also applicable to "any" topology where the traffic pattern involves potential network exit congestions due to multiple sources having the same destination or where multiple routing paths are possible.

A.6 NOSTRUM IMPLEMENTATIONS

Even though the full Nostrum protocol stack only exists as a model it is suggested that (at least the) lower parts of the protocol stack are implemented in hardware. Below is a summary of different variants of the synthesised variants of Nostrum where the results and findings are touched in brief.

A.6.1 PANACEA

In 2005 a prototype of Nostrum was implemented in silicon – the PANACEA; the purpose was to prove the concept of the Nostrum NoC but also to investigate the area, power consumption, and the latency of the switch. The implementation was to a large extent based on the Master's thesis of Erland Nilsson – *Design and Implementation of a Hot-potato Switch in a Network on Chip* [Nilsson2002]. The project was carried out as a semester thesis at the Integrated Systems Laboratory (IIS) at the Swiss Federal Institute of Technology Zürich [Guindi2005 and Nilsson2006].

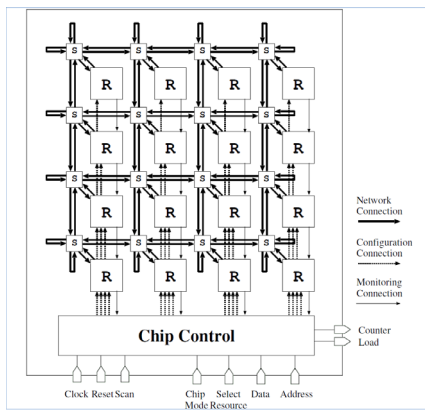


FIGURE A.17. PANACEA – A Nostrum Implementation

The PANACEA NoC is a 4×4 mesh network with 16 Switch/Resource pairs implemented in a 0.250 µm technology depicted in Figure A.17. The switch is employing deflective routing and is enhanced with Proximity Congestion Awareness [Nilsson2003]; further it is dual buffered with separate Ejection and Routing stages. In this rather small example, the total packet size was only 13 bits. The area of the switch is linearly dependent on the number of bits in the packet payload with 3750 μm^2 per bit for a network of 16 switches.

The switch area has a *bias-area* of 50000 μm^2. Hence, the resulting area of the 13 bit packet switch is 100000μm^2 – that is 0.1mm^2 – and it has a critical path length of 5.8 ns, which corresponds to a gate depth of 48 gates. In general, the area of the buffer-space of the switch grows linearly and the area of the decoder logic grows logarithmically with the address space. A very rough estimate on the area for the128 bit packet switch that was discussed in *The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone* [Millberg2002] would then be 0.53mm^2 given the 0.250 µm technology – the resulting number of equivalent gates is hence ~35.5k gates[1]. For a 0.18µ this would roughly translate into *an area of 0.3 mm^2 per switch;* this is to be compared with the switch of José Duato et al. in *A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks* with an area of approximately 2 mm^2 in a 0.18µ technology [Duato2005].

A.6.2 The Parameterisable Switch

In 2006 Vineeth Govind, Jaan Raik, and Raimund Ubar presented another synthesised implementation of the Nostrum Switch [Govind2006]. They used the Nostrum switch as a vehicle for their scalable test bench that was claimed to provide high-fault coverage test patterns for network implementations. For a single buffered 128 bit payload switch, they report a total area of 21k gates where the area of the non-combinatorial logic is ~12k gates. For a dual buffered switch, this would translate into a switch that is approximately 33k gates, which is in line with the results of previous subsection.

1. The approximate conversions between area and gates of the 0.18 and the 0.25 technologies are based on the UMC Free Library where the 0.18 and the 0.25 have a raw gate density of 110k and 67k gates/mm^2, respectively.

A.6.3 The Thin Switch vs. the Square Switch

In 2004 Dinesh Pamunauwa et al. did an investigation of how to physically implement the Switch of Nostrum [Pamunuwa2004]. The two alternatives that were elaborated were the *Thin switch* and the *Square switch*. The thin switch was first proposed by William J. Dally and Brian Towles in 2001 [Dally2001]. This proposed routing layout places the network wires *on top* of the resources in dedicated metal layers as depicted in Figure A.18a. The resources are the grey areas in the pictures and the combinatorial logic and the registers are the striped areas. The thin switch has no area overhead associated with the network wires, but routing the wires over the resource does impose a few restrictions on the design methodology of the resource; also the placing of repeaters may interfere with imported IP cores.

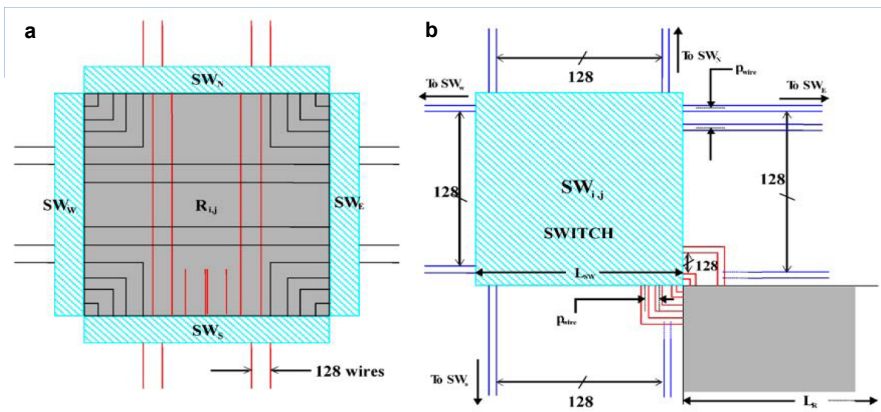


FIGURE A.18. The Thin Switch and the Square Switch

In the square switch, the wires run in dedicated channels as depicted in Figure A.18b. This strategy grants the signal integrity of the wires but is more costly in terms of the area dedicated for the network. In summary, the findings of this paper could be expressed with that the network will occupy a relatively large extent of the chip and the power consumption of the network will be substantial.

Some of the advantages and disadvantages of the two architectures are given in Table A.1 below.

TABLE A.1. The Thin Switch vs. the Square Switch

Square Switch Architecture	Thin Switch Architecture
• Area overhead between 10-20 percent for the network	• Area overhead of roughly 5 percent
• All metal layers can be freely utilized by the resource	• Number of available metal layers or available fractions of two metal layers reduced for resource
• No routing/pin congestion over resource due to network	• Routing/pin congestion introduced by network
• Dedicated channel allows repeater insertion, shielding and explicit signal return planes, guaranteeing signal integrity	• Repeater insertion and shielding more of a problem. More susceptible to noise coupling from above and below
• Max. link bandwidth of 1.5 Tbit/s in any direction	• Max. link bandwidth of 1.1 Tbit/s

B

The Semla Simulator

Given the complexity and cost of a System on Chip today simulation is for most cases the only realistic way to test, evaluate and analyse new architectural ideas. This statement is even truer in an academic environment since the researches are targeting novel ideas with scarce resources in terms of man-power as well as in time – e.g. the duration of a PhD. Simulators are flexible and easy to parametrise to gather statistical data for analysis. This chapter is written with the intention of being a support to understand the context in which the included papers of the thesis were developed.

To understand and appreciate any simulator the purpose of the simulator must be clear. Most simulators fall into either of the two categories (1) they are meant to investigate and test an existing system (2) they are meant to investigate architectural ideas of a system that do not yet exist – our Nostrum NoC Simulator Semla falls into the latter category.

For natural reasons this chapter will, to some extent, overlap the previous Nostrum chapter since it will be very hard to discuss the Semla simulator without putting it into a Nostrum specific context.

B.1 The Semla Simulator

Since the development of Semla has been an evolving process that has spanned almost a decade it is hard to deliver *the right snap shot in time* capturing the most relevant surroundings to the papers included in this thesis.

Today, Semla is *an object-oriented discrete event simulation tool implemented in C++* with the aid of SystemC. Semla has the purpose of being the vehicle in the development of the protocol stack and architectural features of our Network on Chip platform Nostrum. The Semla simulator is the core of a simulation tool-chain with the capability to, in detail, specify and customise the Nostrum platform as well as the applications. The tool-chain has three main components. (1) The *Simulation Generator* is a script based tool that parses a "high level" description of the experiment that the designer wishes to perform and generates an xml-description for the Semla tool to use as an input. The high-level description of the experiment carries information like

- System Information – Clock frequencies, conditions for aborting simulation, etc.
- Nostrum Specific – Which switching policy to use, what topology, what interface the RNIs should be equipped with, buffer sizes, etc.
- Application – How many and what kind of application(s) to run and with what parameters together with a description of what simulation parameter(s) to use, e.g. the packet injection intensity.
- Logging input – Which events to log (if any) and to what extent the individual packets should be traced.
- Simulation meta data – Contains meta-data about if the individual simulation runs should be remotely run on other servers and if and how many simulations runs that should be run in parallel.
- Output data treatment – Which results that should be presented, and in what format. The format could be graphs showing various results like latencies, congestions, queue sizes, with time-line plots, means, standard deviations, histograms, etc.

The Simulation generator then launches a set of *Nostrum Semla* core (2) instances together with the appropriate xml data input files. As said before Semla is the real workhorse of the tool-set and is the Nostrum platform in flesh – more on this in Section "The Semla-Nostrum Implementation," on page 194. The last part of the tool-chain is the *Simulation Output Data Analyser and Visualiser.*

Some History

The trip that led to the final simulator used for the development of Nostrum has been quite long and has gone in different directions since the immediate goals and requirements for the simulator have shifted over time. To motivate why the Simulator is written in the way it is, and why SystemC/C++ was chosen, I feel I must make some historical notes on the development of the simulators.

Our first attempts on a simulator were more directed towards *modelling* the behaviour of a potential switch in VHDL. The choice of language/modelling environment was an easy one, since we had a clear background as hardware designers and possessed an in-depth knowledge and expertise in VHDL and ModelSim [ModelSim]. Over the years several different languages and environments were tried out – often in parallel.

2001-2003 VHDL – highly accurate, synthesiseable and very good for implementing and verifying the behaviour of the switches and links. The problem was, however, that relatively small changes in the design took long time to implement and even longer time to verify. When the simulator/platform grew and a Network Interface with a protocol stack was needed the time for development, and turnarounds, became too large. Now, we were interested in a graphical front-end as well as support for statistical analysis and hence Matlab was chosen for a parallel version of the simulator [Matlab]. The VHDL version was actually *the model* and the Matlab implementation the first *simulator* since it only simulated the behaviour of hardware and not the hardware itself – even though it was cycle accurate.

2001-2003 The Matlab based simulator served its purpose well since it gave us a good graphical view of how the network behaved in different situations from one clock cycle to the next. Furthermore – the built-in support for statistical analysis enabled us to spot and correct the weaknesses of the various routing algorithms tried. The VHDL based simulator was developed/updated in parallel to incorporate stable improvements. The Matlab simulator did, however, not support any *rapid* development even though it was considerably faster than developing in VHDL. Time to get back to the drawing desk and try something different.

2003-2004 Since the goal now was rapid prototyping of a protocol stack the choice fell on Python [Python]. Python is a dynamic object-oriented programming language that can be learned in a few days. Python has the advantage of being extremely intuitive and comes with extensive standard libraries. This new language further decreased the time for development and investigation of novel ideas in comparison to Matlab; also it offered packages for graphical user environments, statistical packages, support for scripting, possibilities of creating advanced traffic generators et cetera. So what could possibly go wrong? Nothing – if you want to live in isolation... except for some low-level RTL projects in Python nobody did use it for hardware development at that point so the Python implementation of the simulator was abandoned and could, in retrospective, is considered a sidetrack. Nowadays, there exist a set of interesting Hardware Descriptive Languages (HDLs) in Python like e.g. MyHDL that both resemble – and can be converted into – Verilog [MyHDL, and Decaluwe2004].

2001-2009 In order to be compatible with the rest of the world, in 2003, I decided to join a colleague – Rikard Thid – in his effort in developing a simulator in SystemC [SystemC]. The name he had chosen for the simulator was Semla (Simulation EnvironMent for a Layered Architecture) [Thid2003, and Thid2006]. As a starting point Semla was, as the acronym suggests, much focused on a layered design; the layers here are referring to the layers of the OSI stack. The layered design style makes it easier to develop different parts of the network independently of already existing parts or parts to come. Clear interfaces and responsibilities alleviate a distributed design style where a natural refining process can take place. More on layered design in Section 4.6 on page 78

To sum up, this historical view I could say that the conceptual focus of the simulator has shifted in a subtle way over time – in the beginning, we *implemented* the Nostrum platform in VHDL – what we later ended up with was a *simulator simulating an implementation* of the Nostrum platform!

B.1.1 SystemC

Since the Semla simulator is written in SystemC it deserves a few words. SystemC is a C++ class library and has been developed to both be a functional Hardware Description Language (HDL) as well as being capable of modelling software to support system level design. In order to cope with the shortcomings of Verilog and VHDL, SystemC can, natively, simulate both hardware and software to allow even large systems to be modelled.

The class libraries of SystemC implement a scheduler that emulates concurrency between *processes*; processes in this context are small pieces of code that concurrently coexist in a system and could either be describing hardware or software. One of the key features is the ability to support models at different abstraction levels; for rapid prototyping very high level descriptions could be used. As the specification is verified to work, a more detailed description of the models could be employed to support the final design. Moreover, the capability of simulating at different levels of detail is essential for the fast verification of the system from a simulation perspective.

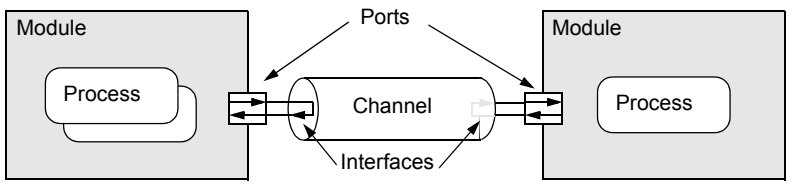


FIGURE B.1. SystemC – Modules, Processes, Ports, Interfaces, and the Primitive Channel

The models written, e.g. processes are encapsulated in *modules*. Modules are the building blocks of SystemC – the modules can consist of other modules in a hierarchical manner. Each module has any number of *ports* that it uses to interact with other modules. To enable complex communication between modules SystemC has the ability to organize the communication into *channels* as shown in Figure B.1 Channels are equipped with *interfaces* that the modules use to communicate through. The simplest channel is used to emulate the behaviour of plain wires as depicted in Figure B.1. More complex channels such as the *hierarchical channel* can connect any number of modules and implement several other, non-hierarchical channels. The hierarchical channel can contain processes, ports, and other modules. Examples of hierarchical channels are the model of a bus or a NoC infrastructure. The hierarchical channel is conceptually depicted in Figure B.2.

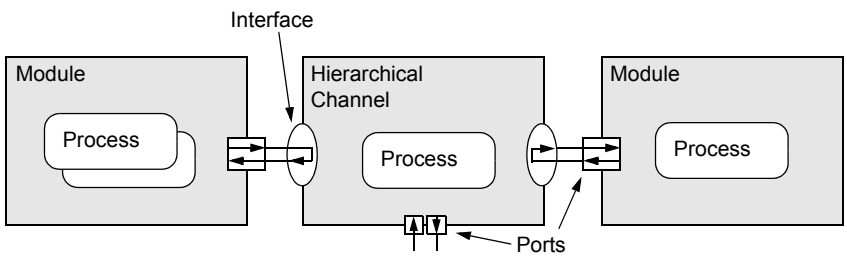


FIGURE B.2. SystemC – Modules, Processes, Ports, Interfaces, and the Hierarchical Channel

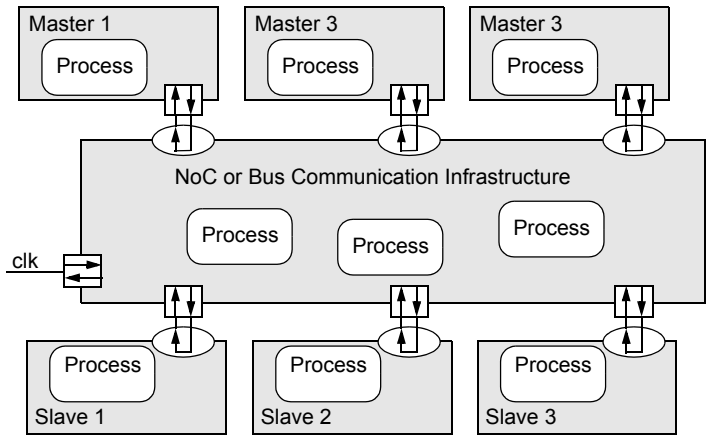


FIGURE B.3. SystemC – Hierarchical Channel used as a NoC or Bus Communication Infrastructure

If a bus or a NoC would be implemented by a hierarchical channel, in the simplest fashion, according to the Master-Slave model, it may look like Figure B.3.

B.1.2 The Semla-Nostrum Implementation

The Semla-Nostrum simulator is conceptually divided into an *Application domain* and a *Communication domain* as depicted in Figure B.4. The application domain contains the *Resources* that send and receive traffic utilising the communication infrastructure. The communication domain contains the communication infrastructure *Nostrum*. The purpose of the Resources of the Application domain is to generate traffic so that the behaviour of the network for a given workload can be studied. The implementation of the Resources can, potentially, span

- Very detailed models of "real" IPs
- Reactive traffic models
- Traffic generators emitting recorded "real" traffic traces
- Very simple models generating random traffic

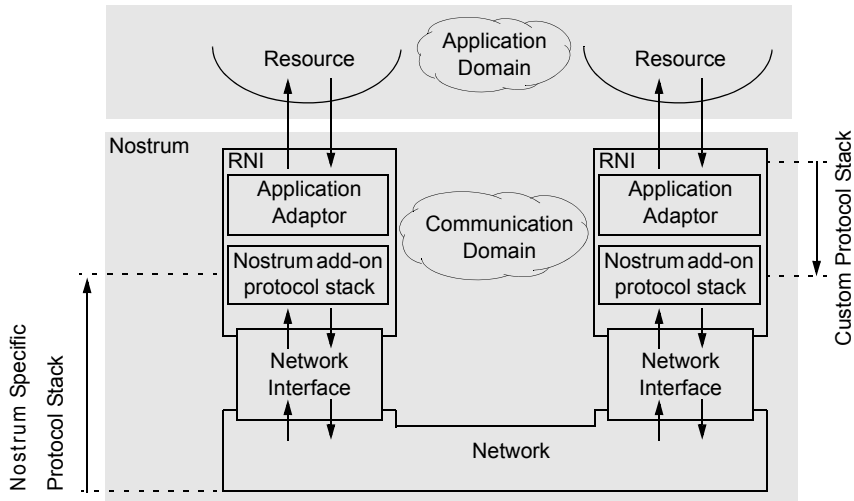


FIGURE B.4. The Application, the RNI, the NI, and the Network

The resources interact by sending and receiving messages over the communication platform in the communication domain – Nostrum.

As it can be seen in Figure B.4, the Nostrum platform is conceptually divided into three parts

- Network
- Network Interface – NI
- Resource Network Interface – RNI

The protocol of the network is independent of the application using it – even though the topology may vary. The network communicates with the outside world via the *Network Interfaces (NIs)*. The API of the Network Interfaces is Nostrum-specific and will be hidden from the applications by the *Resource Network Interfaces (RNI)*. The implementation of the RNI is application specific and consists of a configurable *Nostrum Add-on protocol stack* in combination with an *Application Adaptor*. The application adaptors work as proxies for the application to implement the desired application specific interface – e.g. AXI, MPI, etc. [AXI and MPI1993].

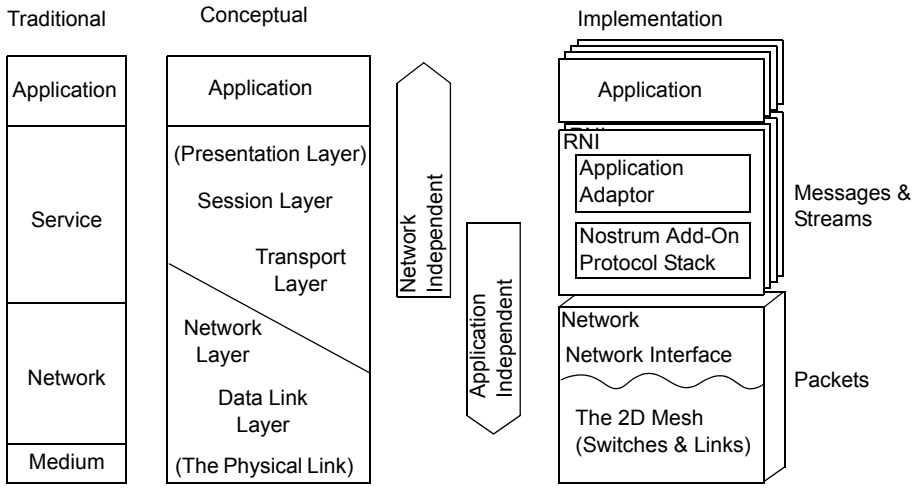


FIGURE B.5. The Nostrum Implementation in Relation to the OSI Protocol Stack

The configurable Nostrum Add-on protocol stack is a *custom subset* of predefined components needed to fulfil the requirement of this very application adaptor. By only selecting a subset of these components the RNI can be customised at a minimal cost. The individual components are providing different behaviours like stream oriented communication, message oriented communicating, FIFO queue-sizes, acknowledgments, traffic shaping functionality, etc.

The Network and the Network Interface

From the outside, the Network is a *black box* equipped with Network Interfaces to connect with the outside world. The purpose of hiding the internals of the network is twofold. It explicitly enforces an independence from the actual implementation detail of the network such as topology, protocol, etc. Also this independence works the other way around – it is easy to relocate applications running in a resource to another location without having to change the network[1]. To facilitate this flexibility the traffic uses Connection (or channel) Identifiers (CIDs) to address the desired receiver application. Each network interface, consequently, has to have a look-up table where the CID is translated into a *Network Address*.

1. This relocation is done off-line for the purpose of analysing how different mappings of application will affect the traffic behaviour of the network.

The look-up table only has to keep a record of the *Channels* associated with its respective application. A channel constitutes a contract between the communicating applications and the communication infrastructure with certain guarantees and characteristics attached to it.

Internally – in the current implementation – a two-dimensional mesh network structure is realised. Figure B.6 shows an implementation view of a corner portion of this network. The ports connecting the Network Interfaces (NI) to the switches (Sw) is a pure modelling construct; in a "real" design the switches and NI would be tightly integrated. The purpose of the wire element is to enable modelling of 'physical' implications such as the introduction of wire delay, bit errors, dead links, etc. The existence of the *Edge* elements is optional, and if they are present they will serve as an extra set of buffers in the network. Simulations have shown that the presence of these buffers will enhance the overall performance of the network.

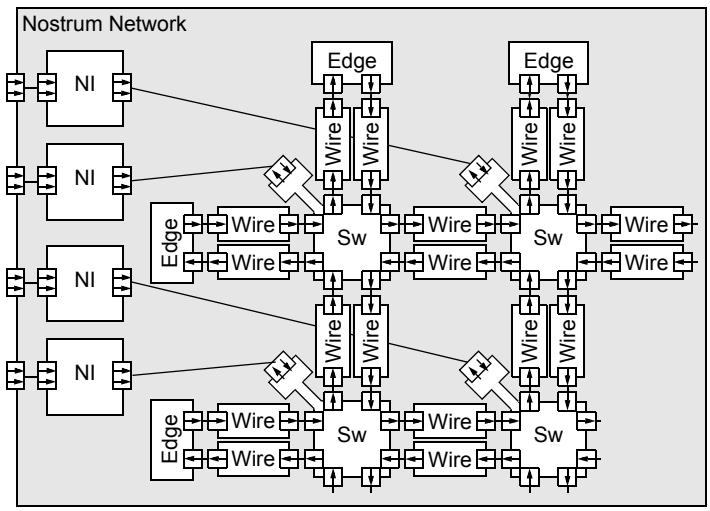


FIGURE B.6. The Network Part of the Nostrum Network

The choice of topology is done in the start-up phase of the simulation. Different models, mappings, and topologies could easily be selected thanks to a modular structure combined with highly customisable, xml based, configuration files. However, in the current implementation of Nostrum the 2D mesh was the most thoroughly investigated even though an implementation of the tori topologies have been tried out. The ease of customising the network has been extensively used throughout the project to analyse different routing and buffering strategies.

The Resource Network Interface – The RNI

The RNI mainly implements the functionality of the transport layer and functions as an adapter between the network and its Nostrum specific protocol and the Applications.

Traditionally, the Transport Layer provides transparent transfer of data between end users, providing reliable, error-free virtual point-to-point connection data transfer services to the upper layers. The transport layer controls the reliability of a given link through flow control, segmentation/desegmentation, and error control. In Nostrum, it also is responsible for the bookkeeping of the channels.

In Nostrum the RNI is the custom hardware/software used to connect the *Nostrum* backbone protocol stack with the communication protocols used by the Resource. The RNI handles all the application specific communication issues. Some examples: In the case of the application being a memory the RNI could act as an arbiter or DMA. If the application is a processor, e.g. an ARM core [Furber2000] the RNI could act as a bridge between e.g. AMBA bus and the NI. In addition, the RNI can act as a bridge between already proposed standards for communication and the *Nostrum*, e.g. the VSI Alliance [VSI2001]. The boundary between the applications and the RNI is, however, not regulated within the *Nostrum* architecture. The *Nostrum* protocol stack can be implemented with an 'arbitrary' depth, i.e. if the application only requests a very basic functionality the stack can be more or less shallow.

Other groups that have implemented an adapter between the applications and the network are Tobias Bjerregaard et al. [Bjerregaard2005b]. Their Network Adaptor (NA) functions as an OCP compliant interface between their NoC architecture MANGO and any system complying with the OCP standard of communication. Their NA decouples communication from computation and provides an interface between their asynchronous message-passing based network to the Memory mapped synchronous interface of OCP. Even they have chosen a layered approach to the design of their network with the argument that it enhances system level composability; this despite the overhead of packetisation/depacketisation since it enhances design productivity.

B.2 The Process of Mapping and Setting up Communication

Below is a very coarse description of the process of getting from a simple description of the system to be simulated to the working process when the simulation starts and all the channels are set up.

In order to explain the mechanism used when a channel is set up a basic set of "definitions" are needed. Below some definition together with a general flow in mapping of Atomic Communicators and their respective communication is described.

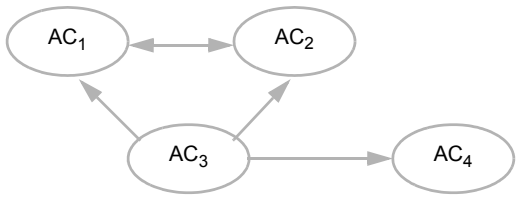


FIGURE B.7. The Raw Task Graph and the Atomic Communicators

Raw Task Graph

Everything starts with the 'Back of a napkin' description of the system where the different communication entities are present. The **Raw Task Graph** only identifies the *existence* of communication (Figure B.7).

Atomic Communicator (AC)

Communicating entities that are the end users of the system that *from a system perspective* are atomic when it comes to communication. The ACs are identified from the Raw Task Graph. An application can be seen as can be seen as one (or several) Atomic Communicators. All ACs have a unique Atomic Communicator ID (**ACID**) – Figure B.8.

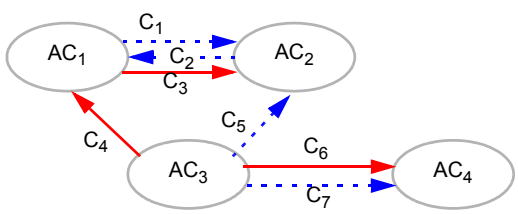


FIGURE B.8. Channel Mapped Task Graph

Channel Mapped Task Graph & Channels

Once the existence of communication between **AC** has been identified the actual needs for communication is analysed and the requirements are expressed as *Channels (C)*. With a channel comes direction and desired QoS properties (e.g. bandwidth, latency, time of communication, etc.)

Resources (R)

The Resources of the system are the entities where the applications will be run, and the different ACs will be implemented. A Resource can hold one or more ACs, with or without internal communication.

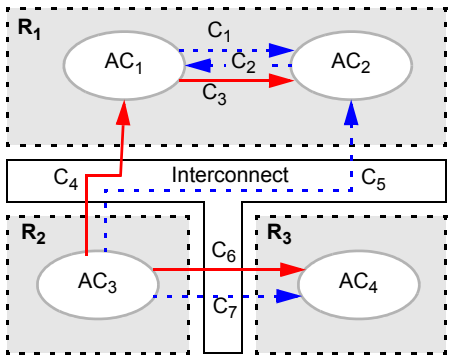


FIGURE B.9. Resource Mapped Task Graph

Resource Mapped Task Graph

Now it is possible to make an attempt towards a mapping where some of the ACs are placed in a single **Resource (R),** and other ACs can share one resource. The inter-Resource communication is handled by the interconnect, e.g. Nostrum whereas the intra-Resource communication is custom. This mapping is called the **Resource Mapped Task Graph** (Figure B.9). From this mapping, an analysis can be carried out in order to determine the feasibility of the mapping and a performance estimation can be made. This is entirely done offline and currently not part of the Semla-Nostrum Simulator tool-chain.

From the perspective of the interconnect it needs to know the location of the AC, i.e. the AC → Resource mapping as well as the QoS requirement of the different channels. This information is handled to the interconnect at the start-up of the simulation as we will see.

The Channel Mapping Process

This far nothing is said about how the interconnect "knows" how to locate the receiver of a message. Hence, this very brief section will touch upon the subject.

In Nostrum every RNI is equipped with an interface to the resources – the *application adaptors*. This interface can be of different kinds dependent on the particular needs of the applications inside the Resources. The application adaptor that most resembles the native interface of the adaptive Nostrum Protocol stack is the message passing interface. The development of this basic interface has been inspired by the Message Passing Interface (MPI) [MPI1993] and hence will be referred to as *mp_if*.

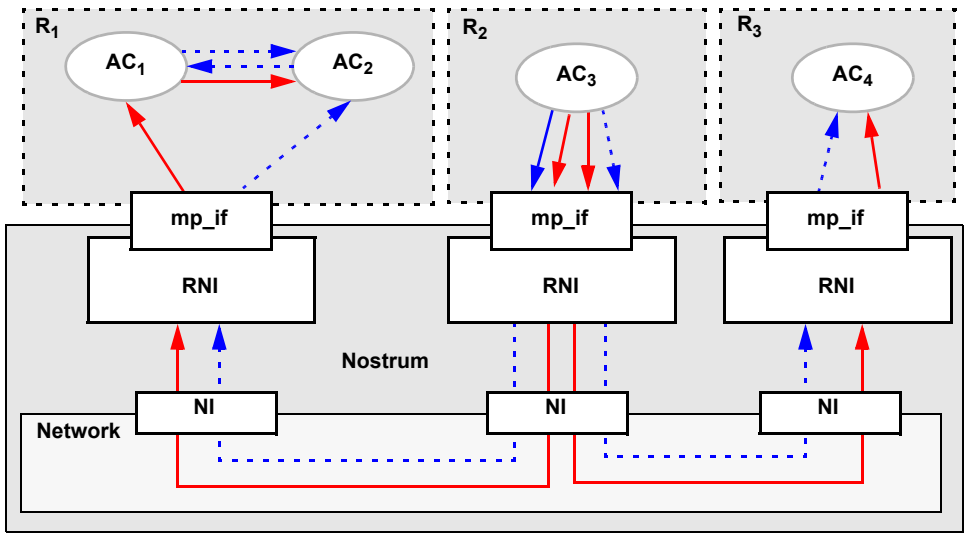


FIGURE B.10. Resource Mapped Task Graph

The following assumptions are made:

- The discussion will be carried out without going into detail of the requested Quality of Service (QoS) of the channels and the process of granting/guaranteeing QoS done by the interconnect. Currently, all QoS calculations are done offline and hence no application will ever ask for a QoS that cannot be fulfilled. So whatever QoS level that is requested they are granted!
- The Atomic Communicators are all aware of their communicating peers and identify them with their respective ACID.

To set up the interconnect to function the following is needed. First all Atomic Communicators initially perform a *requestAndRegisterAcid(Acid myAcid)* call – this will notify Nostrum in which Resource the different ACs resides. After that the application will make an *openChannel(Acid source, Acid destination, ChannelType)* request. The ChannelType is a predefined Quality of Service descriptor that can be understood by the interconnect. Once the request is processed by the interconnect and a Channel Id (CID) is returned to be used as a handle through which communication (read or write) can be handled – e.g. *write(CID, Data)*. The administration and bookkeeping of the channels, atomic communicators, packet routes and suchlike are handled by an "invisible" operating system that is beyond the scope of this thesis.

B.3 SEMLA SIMULATION ORDER

A simulation run with the Semla simulator involves execution of a few logical phases as shown in Figure B.11

- First all the necessary parameters need to be extracted to be able to build the simulated system. The system here is Nostrum with its RNIs, NIs and network and the connected resources with its applications. The parameters are collected by a container instance – simParams – using the xmlHandler to parse all the xml input data read from the file that was previously created by the *Simulation Generator*. The simulator and the components that will be generated will heavily rely on the information that now resides within the container instance simParams since this information is the core of how the simulator will look and behave.

- The first thing that is being configured based on the content of simParams is the LogHandler that has the responsibility to log and trace execution as well as individual packets. The LogHandler will keep a record of all this information in a database for later processing. The reason for implementation this very dedicated logHandler is that an implementation of a network will be composed of many identical components like switches, wires, etc. This creates a challenging task when it comes to logging and debugging code since the very same source code will be used to create a multitude of instances where the only thing that is distinguishing one instance from another is the instance name. Hence the logHandler is capable of creating different logging information based on the instance names.

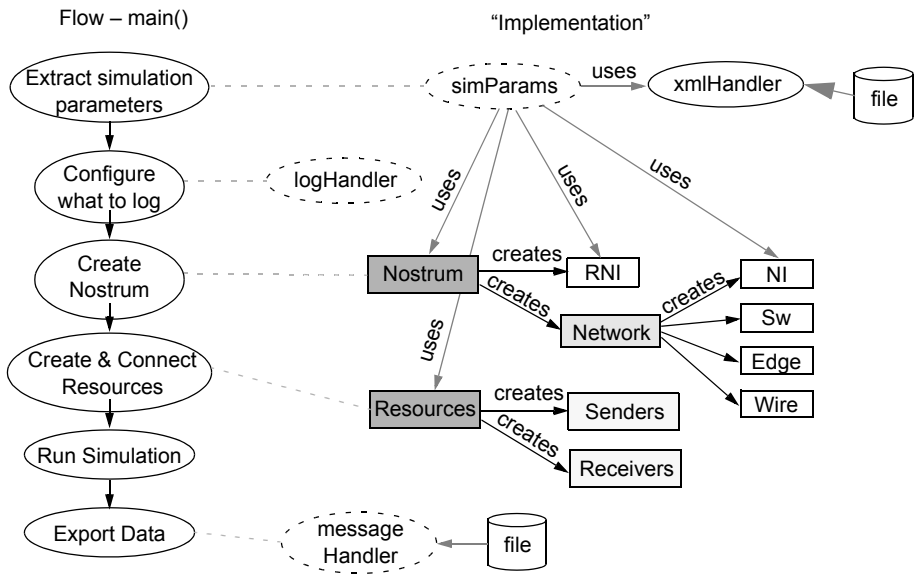


FIGURE B.11. Semla Simulation Order

- Once the logger is configured Nostrum is created. The Nostrum instance in turn creates, and connects, the RNI and the Network. The Network – once created – will consequently, create and connect the necessary Network Interfaces, Switches, Wires and Edge elements. The size, configuration in terms of switching policy, buffer-sizes, etc. is completely dictated by the information in the simParams instance.
- The same procedure is then repeated with the Resources that create and connect the sender and receiver parts of the applications. Finally, all Resources are connected with Nostrum.
- The simulation now can start and will stop as soon as all sent packets have been received or if the simulation times out. The reason can be that the number of packets/data to be sent is not limited, or the simulation has run havoc due to an "unexpected event" (read – *bug*) in the network implementation, or a dead-lock situation in the resources.

When the simulation has finished the messageHandler collects the requested raw and statistically derived data concerning packet and messages and writes it to a file. The logHandler will also output its raw data or a summary report of interesting events to a file.

Packet Logging

During a typical simulation run traffic will be generated in an application, get transferred over the network to finally be consumed by another application. To make correct and meaningful measurements it is important to know what and where to measure. To achieve this; a set of tags are attached to each individual packet. The tags are a pure simulation construct and will not be included in the packet payload.

At creation all packets get a unique packet id. This identifier is used throughout the simulation and later in the post data processing to keep track of the whereabouts of individual packets.

Let's first assume, for simplicity, that all communication has the granularity of packets. By that I mean that whatever quanta of information the application wishes to send is it is in the form of packets. During a packets lifetime, it will be tagged with various time stamps as depicted in Figure B.12.

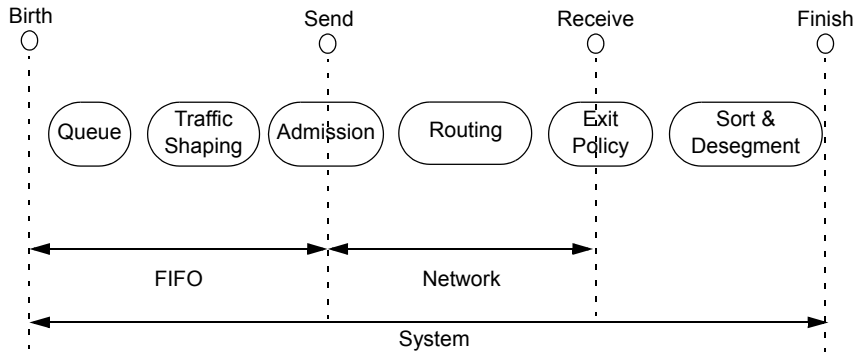


FIGURE B.12. Packet Time Tags

- t_{birth} – Once the packet is created it will get tagged with its time of birth. The packet will now, potentially, undergo traffic shaping and will be put in a down-stream queue waiting for admission onto the network.
- t_{send} – As soon the packet is granted admission to the network the t_{send} tag will receive its value.
- $t_{receive}$ – Set when the packet is leaving the network. The packet is now put into the upstream reception buffers.
- t_{finish} – Set as soon the receiving application is accepting the packet, and the packet leaves Nostrum.

With the aid of these time tags, it is now easy to calculate individual and accumulated latencies of the System (t_{finish}- t_{birth}), the Network ($t_{receive}$- t_{send}), Downstream Queuing times (t_{send}- t_{birth}) etc.

In the case that the application wants to transmit data that will not fit into one single packet a *messag*e is created. All packets that belong to this particular message will get a Message Id (MID). The message identifier will later be used in the data post processing stages to associate packets belonging to the same message. From the collected knowledge about the individual packets belonging to the same message, delivery times, latencies, queue size requirements, etc. can be deduced.

B.4 NNSE

For the sake of being complete the graphical front-end NNSE of Semla deserves a few words [Lu2005]. NNSE (Nostrum Network-on-Chip Simulation Environment) was built to aid in the process in selecting a network architecture that suits a particular application. In Figure B.13 the work flow of NNSE is illustrated.

Furthermore, it gives easy access to Semla's configuration parameters and presents the output of a simulation in a comprehensive way. NNSE was written in Python as a wrapper to the Nostrum Semla core; this means that it does not utilise the aforementioned *Simulation Generator* or the *Simulation Output Data Analyser and Visualiser*. From the Graphical User Interface (GUI) of NNSE it is easy to configure the network with respect to topology, flow control, routing algorithm, etc. In addition to customise the network parameters the network should be evaluated extensively with various regular and application-specific traffic patterns, hence the traffic patterns are configurable from the GUI that gives access to a set of pre-defined patterns.

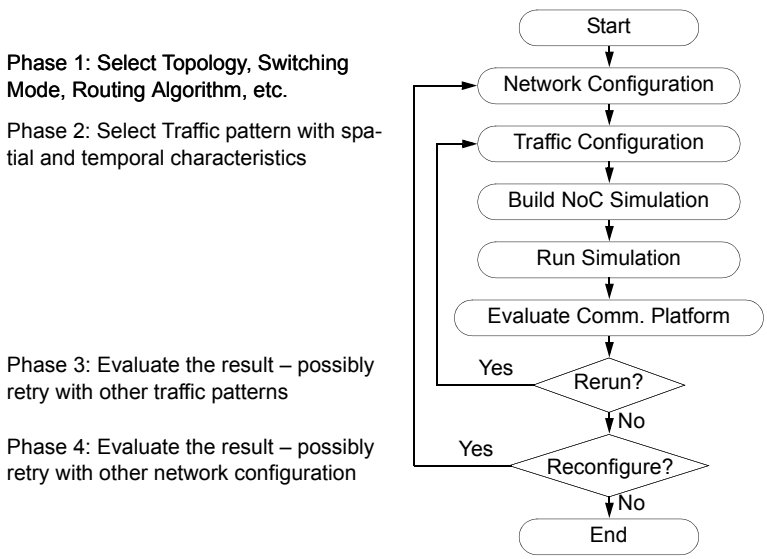


FIGURE B.13. NNSE Work Flow

Network Configuration

During configuration phase one the user has to select topology, switching mode and routing algorithm. In our current implementation, the selectable topologies are limited to two dimensional meshes and tori, and the switching modes implemented are deflective and wormhole routing. The wormhole routing option is further customisable when it comes to the number of Virtual Channel (VCs) and the depth of these channels.

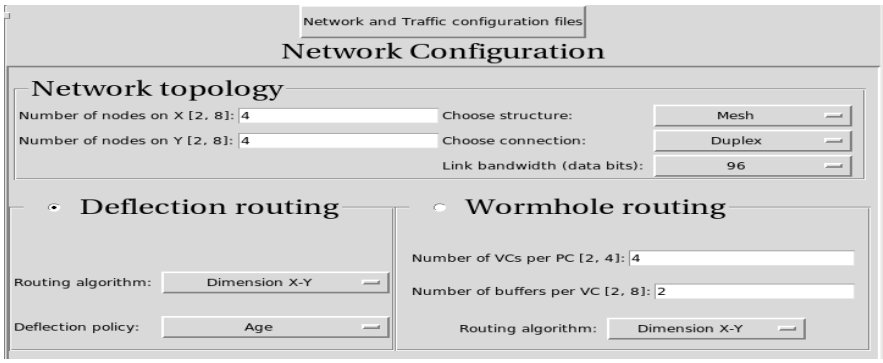


FIGURE B.14. NNSE Network Configuration

Traffic Configuration

The traffic patterns in NNSE can be either regular or application specific. The application-specific traffic is set up on a per-channel basis and can be used to configure user defined irregular traffic. The communication characteristics of each channel can hence be explicitly defined and tuned.

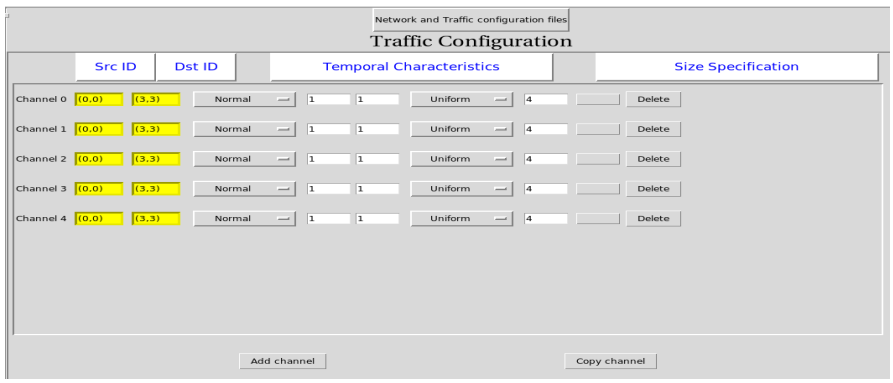


FIGURE B.15. NNSE Traffic Configuration – Based on Channels

The regular traffic patterns are divided into *uniform* and *locality traffic*. The uniform traffic is distributed over the network nodes uniformly. Whereas the locality traffic is based on a locality index that controls the communication probability between nodes at different distances. Further it is possible to specify the spatial characteristics, temporal characteristics, and the sizes of the messages to be sent over the network.

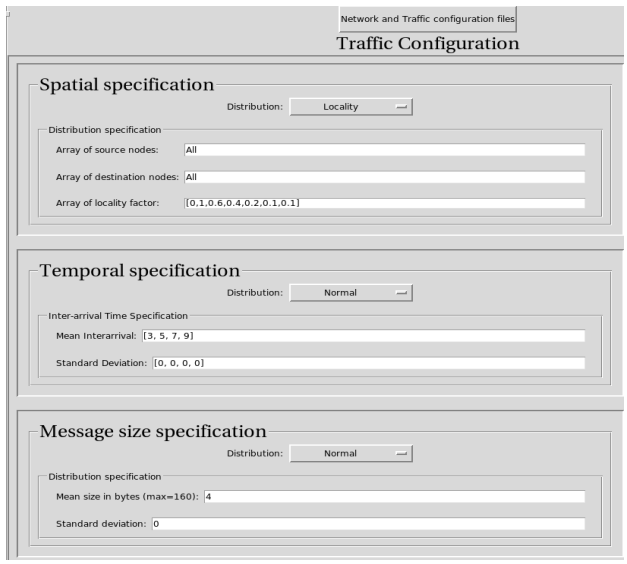


FIGURE B.16. NNSE Traffic Configuration – Regular Patterns

Performance Evaluation

Once the network is configured, and the traffic pattern defined the simulator is built and run. The evaluation is based on the kernel simulation results and presented as graphs. The main performance measures are latency and throughput. Typical figures include average latency vs. offered traffic, throughput vs. offered traffic, etc.

The NNSE also offers the possibility to store network and traffic configurations to be reused. To facilitate data exchange, they are stored as XML files.

B.5 Simulation Speed, Validation and Traffic Patterns

Simulation Speed and The Validation Issue

Semla's detail comes at a cost of slow simulation speed – a limitation shared by most detailed/low-level simulators. As stated by Christopher J. Hughes et al. in their paper about their simulator *RSim "... the execution times predicted from simulation deviate significantly from real hardware results and the tuning of these simulators to match the hardware require considerable effort."* [Hughes2002]. In their paper, they acknowledged the idea a hardware implementation should be built to validate the final ideas developed from simulation to be good but potentially controversial. The reasons why there might be a problem in selecting a reference architecture are essentially the three listed below with a twist to adopt it to our application:

- Technology and ideas evolve very quickly. Once the effort is put in building a hardware prototype, new ideas and a new desired hardware implementation has made it obsolete.
- There is no consensus on the *ideal architecture* – in our case this means that we don't know what to optimise the hardware architecture for; this since we do not have any reference applications.
- Most machines/network on chips have architectural features that in hindsight were considered incorrect or non ideal for the current traffic pattern. If "real" performance numbers should be given these imperfections should/must be removed.

There have been a few reports on Network on chip emulators in the literature. For example, Nicolas Genko et al. built a support system on a FPGA board that delivers a complete environment for HW-SW NoC Emulation. The synthesiseable switches are generated by using the ×pipes compiler [Jalabert2004]. On board on their emulator platform are controller, traffic generators and receptors as well. During emulation, they report speed-ups up to four orders of magnitude in comparison to traditional HDL simulations [Genko2005]. In their paper, they report rapid turn-around times when it comes to changing parameters in their design like packet sizes, number of switches, etc. There is however not much said about the cost of changing or developing protocols.

Another way to increase the simulation speed is to replace the IP's simulated with corresponding Traffic Generators. By capturing the type and timestamp of communication events at the boundaries of an IP core in a reference environment the Traffic Generator can later emulate the IP's communication pattern. Shankar Mahadevan et al. reports of simulation time speed-ups above a factor of two with close to 100 percent accurate to the original IP [Mahadevan2005].

A second reason for using traffic generators is that a full system might not fit onto a single prototyping chip. Erland Nilsson and Johnny Öberg used 16 traffic generators in the Nostrum prototype chip called PANACEA [Nilsson2006]. For more information see Section 5.4 on page 92.

Regardless if a hardware version of the architecture is to be built the simulation should always be accompanied by a theoretical model or at least a reasoning of the expected outcomes of the simulations. In retrospective I've come to realise that most of the development time of the simulator has actually been spent on validating and verifying that the outcomes of a simulation are sane and "true". That is – is the simulator really behaving as expected and is this behaviour correct? Surprisingly often the results have *appeared* to be correct and in line with the assumed results in statistical terms but later discarded due to an erroneous implementation. With erroneous I mean "buggy" from an implementation language point of view or not in line with the hardware the simulator it is supposed to simulate.

Moreover, the choice of traffic patterns/behaviours is a big issue when it comes to measuring the performance of a simulator. When the papers in the thesis were written there were no benchmark suits available, which meant that it has been very hard to give any reasonable numbers on the speed of the simulator under realistic loads. With available benchmark suits, I mean that no benchmark suits had been presented that had reached a common acceptance within the NoC community. If the problem with realistic traffic patterns is ignored and a synthetic, uniform random traffic pattern is accepted, we could present raw performance numbers in terms of packet traversals per second. Packet traversals per second is the throughput in the number of packets the simulator is capable of accepting multiplied with the average routing distance/hops. Semla as of today gives a raw performance of 2800 packet traversals (hops) per second if it is run on a 3200 Pentium with 2 GB of memory (no swap). The number of packet traversals per second is almost independent of the network size and traffic pattern.

The Choice of Traffic Patterns

The traffic pattern consists of a communication graph and the traffic characteristics on the respective edges of the graph as described in e.g. Section "The Channel Mapping Process," on page 201– see Figure B.17.

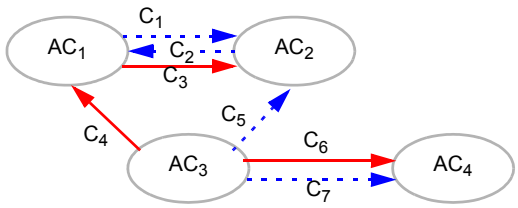


FIGURE B.17. Channel Mapped Task Graph

Network traffic modelling is an approach suggested by many [Genko2005 and Mahadevan2005]. The reasons are that (i) full system modelling/simulation usually gives too slow emulation speed; (ii) the system to be simulated may not exist yet. Network modelling is a key component in understanding the power /performance characteristics of the network.

Network traffic generators range from the simplest synthetic traffic pattern to a very sophisticated full system emulator. The simple models could well be used for evaluation of the network during the early developments of the network concept. Models such as uniform random, weighted random, Poison and permutation and suchlike provide the means to stress and measure profound network characteristics in the early design phases. These models are regular, predictable and hence aid the NoC designer in acquiring insights of the network in questions strengths and weaknesses. However, as Vassos Soteriou et al. so well point out – they do not represent real-life traffic and cannot be used to drive realistic network design space exploration! [Soteriou2006]

Soteriou et al. instead proposes a model that is based on a 3-tuple to capture the traffic's, both, spatial and temporal behaviour. The triplet captures the statistical parameters of hop-count, burstiness, and packet injection distribution. The values of the 3-tuple are statistically derived from a set of full-system simulation. The values can later be used in two ways. First they can give valuable insights on the characteristics of the NoC traffic pattern in question; second they can be used for creating synthetic traffic for NoC evaluation. They report a 95 percent accuracy of their simulation traces from synthetically generated traffic with respect to the "real" full system simulation.

In addition, analysis of the system execution traces could be used as a basis for identifying critical communication events [Lahiri2000]. The traces can be analysed to examine the impact of individual (or groups of) communication events on the system's performance. Communication events which are on the system "critical paths", and whose delays significantly impact the specified performance metrics can be classified as critical events. By assigning appropriate priority levels to these communication events the system's performance can be enhanced.

In the work of Sergio Tota et al. they decided to adopt an emulation approach for the following reason: the communication of an MPSoC is truly bidirectional – regardless if shared memory model or message passing paradigm is used. The circular interdependencies hence cannot be captured by a random traffic generator – no matter how sophisticated! The traces they used were derived from real-life applications running on a multiprocessor system [Tota2006].

Abbreviations & Acronyms

A

AC	Atomic Communicator	164, 199
ACID	Atomic Communicator ID	199
AMBA	Advanced Microcontroller Bus Architecture	29, 35, 198
API	Application Programming Interface	29, 88, 195
ARM	Advanced RISC Machine	17, 198
ASB	Advanced System Bus	17
ASIC	Application Specific Integrated Circuit	27
AXI	Advanced eXtensible Interface	29, 173, 195

B

BE	Best Effort	72, 80, 161
BIST	Built In Self Test	24
BLESS	set of Buffer-LESS routing algorithms	69

C

C	Channel	200
CAM	Content Addressable Memory	181
CID	Channel Identifier	171, 173, 196, 202
CMP	Chip MultiProcessor or Multi-core processor	49
CPU	Central Processing Unit	16, 28

D

DATE	Design, Automation and Test in Europe	xvii, 225
DIP	Dual In-line Package	10
DMA	Direct Memory Access	11, 76, 198
DPE	Dual Packet Exit	7, 75, 100
DRAM	Dynamic Random-Access Memory	182
DSD	Euromicro Conference on Digital System Design	xvi
DSP	Digital Signal Processor	5, 18, 84, 99
DSPIN	Distributed, Scalable, Programmable, Integrated Network	42
DyAD	Dynamic switching between Adaptive and Deterministic modes	61

E

ECS	Electronic, Computer, and Software Systems	xvii
ENIAC	Electronic Numerical Integrator And Computer	11

F

FIFO	First In First Out	40, 95, 196
flits	flow control digits	32, 66
FPGA	Field Programmable Gate Array	27, 209
FPU	Floating-point Unit	28

G

GALS	Globally Asynchronous Locally Synchronous	12, 21, 24, 42, 87
GB	Giga Byte	210
GHz	Giga Hertz	19
GT	Guaranteed Throughput	72, 80, 83, 161, 163
GUI	Graphical User Interface	206

H

HDL	Hardware Description Language	20, 192
HOL	Head of Line	181
HW	HardWare	209

I

K

L

M

N

O

OCP	Open Core Protocol	29, 198
OCP-IP	Open Core Protocol International Partnership	228
OS	Operating System	165
OSCI	Open SystemC Initiative	230
OSI	Open Systems Interconnect	5, 32, 84, 99, 192

P

PANACEA	Synthesised Prototype of Nostrum	184
PC	Personal computer	15
PCA	Proximity Congestion Awareness	95
PCI	Peripheral Component Interconnect	15
PDU	Protocol Data Unit	170
PE	Processing Elements	30
PIP	Picture-In-Picture	52
PSN	Packet Sequence Number	173

Q

QNoC	Quality of service Network on Chip	49, 76
QoS	Quality of Service	70, 201

R

R	Resource	200
RAM	Random Access Memory	11
RD	Read	76
REI	Routing Enhancing Information	172
RISC	Reduced Instruction Set Computer	17
RNI	Resource Network Interface	87, 162, 166, 195
ROM	Read-Only Memory	11
RTL	Register-Transfer Level	192

S

SAP	Service Access Point	170, 173
SDRAM	Synchronous Dynamic Random Access Memory	44
SE	Sweden	xvii
Semla	Simulation EnvironMent for a Layered Architecture	192
SIA	United States Semiconductor Industry Association	25
SL	Service Level	76
SoC	System on Chip	17, 30
SPIN	Scalable Programmable Integrated Network-on-chip	35, 42
SW	SoftWare	209
Sw	Switch	197

T

TDMA	Time Division Multiple Access	3, 49, 55, 80, 82, 161
TDN	Temporally Disjoint Network	5, 96, 163, 178
TTL	Transistor-Transistor Logic	10

U

ULSI	Ultra Large Scale Integration	11
UMC	United Microelectronics Corporation	185

V

VC	Virtual Channel	62, 169, 207
VC	Virtual Circuit	180
VHDL	Very High Speed Integrated Circuit (VHSIC) Hardware Description Language	20
VLSI	Very Large Scale Integration	11, 42, 228
VOPD	Video Object Plane Decoder	46, 52
VSI	Virtual Socket Interface	198

W

WR	Write	76

X

XML	eXtensible Markup Language	208

References

[Arifin2004] Farhadur Arifin, *Implementation and Evaluation of Segmented-Bus Architecture*, MSc. Thesis, Institute of Microelectronics and Information Technology, Royal Institute of Technology (KTH), 2004

[AXI] ARM, *AMBA AXI Specification*, http://www.arm.com/ Website maintained by ARM Information Center [Online] (Retrieved 6 Oct.2009)

[Atienza2008] David Atienza, Federico Angiolini, Srinivasan Murali, Antonio Pullini, Luca Benini, and Giovanni De Micheli, *Network-on-Chip Design and Synthesis Outlook*, Integration VLSI J., vol. 41, no. 3, pp. 340–359, May 2008.

[Aurrecoechea1998] Cristina Aurrecoechea, Andrew T Campbell, and Linda Hauw, *A Survey of QoS Architectures*, Multimedia Syst. (3), pp. 138-151, Springer-Verlag New York, Inc.Secaucus, NJ, USA, 1998

[Banerjee2007] Arnab Banerjee, Robert Mullins, and Simon Moore, *A Power and Energy Exploration of Network-on-Chip Architectures*, Networks-on-Chip, 2007. NOCS 2007. First International Symposium on, pp. 163-172, 2007

[Benini2001] Luca Benini and Giovanni De Micheli, *Powering Networks on Chips: Energy-Efficient and Reliable Interconnect Design for SoCs*, ISSS '01: Proceedings of the 14th international symposium on Systems synthesis, pp. 33-38, 2001

[Benini2002] Luca Benini and Giovanni De Micheli, *Networks on Chips: A New SoC Paradigm*, Computer (1), pp. 70-78, IEEE Computer Society, Los Alamitos, CA, USA, 2002

[Bertozzi2005] Davide Bertozzi, Antoine Jalabert, Srinivasan Murali, Rutuparna Tamhankar, Stergios Stergiou, Luca Benini, and Giovanni De Micheli, *NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip*, Parallel and Distributed Systems, IEEE Transactions on (2), pp. 113-129, 2005

[Bjerregaard2005a] Tobias Bjerregaard and Jens Sparsö, *A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip*, Design, Automation and Test in Europe, 2005. Proceedings, p. 1226-1231 Vol. 2, 2005

[Bjerregaard2005b] Tobias Bjerregaard, Shankar Mahadevan, Rasmus Grondahl Olsen, and Jens Sparsö, *An OCP Compliant Network Adapter for GALS-based SoC Design Using the MANGO Network-on-Chip*, System-on-Chip, 2005. Proceedings. 2005 International Symposium on, pp. 171-174, 2005

[Bjerregaard2006] Tobias Bjerregaard and Shankar Mahadevan, *A Survey of Research and Practices of Network-on-Chip*, ACM Computing Surveys (1), p. 1, ACM, New York, USA, 2006

[Bolotin2004a] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny, *Cost Considerations in Network on Chip*, Integr. VLSI J. (1), pp. 19-42, Elsevier Science Publishers B. V. Amsterdam, The Netherlands, The Netherlands, 2004

[Bolotin2004b] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny, *QNoC: QoS Architecture and Design Process for Network on Chip*, Journal of Systems Architecture (2-3), pp. 105-128, Elsevier Science Publishers B. V.Amsterdam, The Netherlands, The Netherlands, 2004

[Bolotin2005] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny, *Efficient Routing in Irregular Topology NoCs*, Research report, Technion Department of Electrical Engineering, CCIT Report #554, 2005

[Bolotin2007] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny, *Routing Table Minimization for Irregular Mesh NoCs*, DATE '07: Proceedings of the conference on Design, automation and test in Europe, pp. 942-947, 2007

[Cidon2005] Israel Cidon and Idit Keidar, *Zooming in on Network-on-Chip Architectures*, Research report, Technion Department of Electrical Engineering, CCIT Report #565, 2005

[Chang2006] Martijn Coenen, Srinivasan Murali, Andrei Rădulescu, Kees Goossens, and Giovanni De Micheli, *A Buffer-Sizing Algorithm for Networks on Chip using TDMA and Credit-Based End-to-End Flow Control*, CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis, pp. 130-135, 2006

[Cisco2003] Cisco Systems Inc. Internetworking Technologies Handbook, Fourth Edition, Cisco Press, 2003.

[Coenen2006] Martijn Coenen, Srinivasan Murali, Andrei Rădulescu, Kees Goossens, and Giovanni De Micheli. *A Buffer-Sizing Algorithm for Networks on Chip using TDMA and Credit-Based End-to-End Flow Control*. CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis, pp. 130-135, 2006.

[Coppola2008] Marcello Coppola, Miltos D. Grammatikakis, Giuseppe Maruccia, Riccardo Locatelli, and Lorenzo Pieralisi, *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*. Boca Raton, FL, USA: CRC Press, Inc., 2008.

[CoreConnect] Intel Corporation, *Coreconnect Bus Architecture* (white paper), http://www-03.ibm.com/chips/products/coreconnect/Website maintained by IBM Microelectronics [Online] (Retrieved 6 Oct 2009)

[Culler1999] David E Culler, Jaswinder Pal Singh, and Anoop Gupta, *Parallel Computer Architecture: a Hardware/Software Approach*, Morgan Kaufmann, San Francisco, ISBN 1-55860-343-3, 1999

[Dally1986] William J Dally and Charles L Seitz, *The Torus Routing Chip*, Journal of Parallel and Distributed Computing (4), pp. 187-196, 1986

[Dally2001] William J Dally and Brian Towles, *Route Packets, not Wires: On-Chip Interconnection Networks*, Design Automation Conference, 2001. Proceedings, pp. 684-689, 2001

[Dally2003] William J Dally and Brian Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 0122007514, 2003

[Dandamudi2002] Sivarama P Dandamudi, Fundamentals of Computer Organization and Design, Springer-Verlag, New York, 2002.

[Decaluwe2004] Jan Decaluwe, MyHDL: *A Python-based Hardware Description Language*, j-LINUX-J (127), 2004

[Duato1997] Jose Duato, Sudhakar Yalamanchili and Lionel M Ni, *Interconnection Networks: An Engineering Approach*, IEEE Computer Society Press, Los Alamitos, CA, USA, ISBN 0818678003, 1997

[Duato2005] Jose Duato, Ian Johnson, Jose Flich, Finbar Naven, Pedro J Garcia, and Teresa Nachiondo, *A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks,* HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture, pp. 108-119, 2005

[Feige1992] Uriel Feige and Prabhakar Raghavan, *Exact Analysis of Hot Potato Routing*, Foundations of Computer Science, 1992. Proceedings., 33rd Annual Symposium on, pp. 553-562, 1992

[Felicijan2004] Tomaz Felicijan and Steve B Furber, *An Asynchronous On-Chip Network Router with Quality-of-Service (QoS) Support*, Proceedings IEEE International SOC Conference, pp. 274-277, 2004

[Furber2000] Patrick T Gaughan and Sudhakar Yalamanchili, *Adaptive Routing Protocols for Hypercube Interconnection Networks*, Computer (5), pp. 12-23, IEEE Computer Society PressLos Alamitos, CA, USA, 1993

[Gaughan1993] Patrick T Gaughan and Sudhakar Yalamanchili, *Adaptive Routing Protocols for Hypercube Interconnection Networks*, Computer (5), pp. 12-23, IEEE Computer Society PressLos Alamitos, CA, USA, 1993

[Genko2005] Nicolas Genko, David Atienza, Giovanni De Micheli, Luca Benini, José Mendias, Roman Hermida, and Francky Catthoor, *A Novel Approach for Network on Chip Emulation*, International Symposium on Circuits and Systems (ISCAS 2005), pp. 2365-2368, 2005

[Glass1992] Christopher J Glass and Lionel M Ni, *The Turn Model for Adaptive Routing*, Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on, pp. 278-287, 1992

[Goldstine1972] Herman H Goldstine. *The Computer from Pascal to Von Neumann*, Princeton University Press, Princeton, NJ, USA, 1972.

[Goossens2002] Kees Goossens, Jef L van Meerbergen, Ad Peeters, and Paul Wielage, *Networks on Silicon: Combining Best-Effort and Guaranteed Services*, Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings, pp. 423-425, 2002

[Goossens2005] Kees Goossens, John Dielissen, and Andrei Rădulescu, *The Aethereal Network on Chip: Concepts, Architectures, and Implementations*, IEEE Design and Test of Computers, pp. 414-421, 2005

[Goossens2008] Kees Goossens, Martijn Bennebroek, Jae Young Hur. and Muhammad Aqeel Wahlah, *Hardwired Networks on Chip in FPGAs to Unify Functional and Configuration Interconnects*, Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on, pp. 45-54, 2008

[Govind2006] Vineeth Govind, Jaan Raik, and Raimund Ubar, *A Generic Synthesizable NoC Switch with a Scalable Testbench*, Baltic Electronics Conference, 2006 International, pp.1-4, 2-4 Oct. 2006

[Grama2003] Ananth Grama, I*ntroduction to Parallel Computing*, Addison-Wesley, Harlow, ISBN 0-201-64865-2, 2003

[Grecu2005] Cristian Grecu, Partha Pratim Pande, André Ivanov, and Res Saleh, *Timing Analysis of Network on Chip Architectures for MP-SoC Platforms*, Microelectronics Journal, pp. 833-845, 2005

[Greensmith] Greensmiths, Inc., *Ants*, http://www.greensmiths.com/ants.htm Website maintained by [Online] (Retrieved 7 Oct 2009)

[Guerrier2000] Pierre Guerrier and Alain Greiner, *A Generic Architecture for On-Chip Packet-Switched Interconnections,* DATE '00: Proceedings of the conference on Design, automation and test in Europe, pp. 250-256, 2000

[Guindi2005] Nadim El Guindi and Pascal Elsener, *Network on chip: PANACEA – A NOSTRUM integration*. Technical report, Integrated Systems Laboratory / Swiss Federal Institute of Technology Zürich, 2005.

[Gupta1997] Rajesh K Gupta and Yervant Zorian, *Introducing Core-Based System Design*, Design & Test of Computers, IEEE (4), pp. 15-25, 1997

[Guz2006] Zvika Guz, Isask'har Walter, Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny, *Efficient Link Capacity and QoS Design for Network-on-Chip*, DATE '06: Proceedings of the conference on Design, automation and test in Europe, pp. 9-14, 2006

[Hansson2005] Andreas Hansson, Kees Goossens, and Andrei Rădulescu, *A Unified Approach to Constrained Mapping and Routing on Network-on-Chip Architectures*, Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on, pp. 75-80, 2005

[Hansson2007a] Andreas Hansson, Martijn Coenen, and Kees Goossens, *Undisrupted Quality-of-Service during Reconfiguration of Multiple Applications in Networks on Chip*, Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07, p. 1226-1231 Vol. 2, 2005

[Hansson2007b] Andreas Hansson, Kees Goossens, and Andrei Rădulescu, *Avoiding Message-Dependent Deadlock in Network-Based Systems on Chip*, VLSI Design, Article ID 95859, 10 pages, 2007

[Hansson2007c] Andreas Hansson, Kees Goossens, and Andrei Rădulescu, *A Unified Approach to Mapping and Routing on a Network on Chip for both Best-Effort and Guaranteed Service Traffic*, VLSI Design, Article ID 68432, 16 pages, Hindawi Publishing Corporation, 2007

[Hansson2008] Andreas Hansson, Maarten Wiggers, Arno Moonen, Kees Goossens, and Marco Bekooij, *Applying Dataflow Analysis to Dimension Buffers for Guaranteed Performance in Networks on Chip*, Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on, pp. 211-212, 2008

[Hemani1999] Ahmed Hemani, Thomas Meincke, Shashi Kumar, Adam Postula, Thomas Olsson, Peter Nilsson, Johnny Öberg, Peeter Ellervee, and Dan Lundqvist, *Lowering Power Consumption in Clock by Using Globally Asynchronous Locally Synchronous Design Style*, DAC '99: Proceedings of the 36th annual ACM/IEEE Design Automation Conference, pp. 873-878, 1999

[Hemani2000] Ahmed Hemani, Axel Jantsch, Shashi Kumar, Adam Postula, Johnny Öberg, Mikael Millberg, and Dan Lindqvist. *Network on chip: An Architecture for Billion transistor era*. Proceeding of the IEEE NorChip Conference, pp. 166-173, 2000

[Ho2001] Ron Ho, Kenneth W. Mai and Mark A Horowitz, *The Future of Wires*, Proceedings of the IEEE, p. 490–504, 2001

[Hu2003] Jingcao Hu and Radu Marculescu, *Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures*, Design, Automation and Test in Europe Conference and Exhibition, 2003, pp. 688-693, 2003

[Hu2004] Jingcao Hu and Radu Marculescu, *Application-Specific Buffer Space Allocation for Networks-on-Chip Router Design*, ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design, pp. 354-361, 2004

[Hu2005] Jingcao Hu, *Design Methodologies For Application Specific Networks-on-Chip*, PhD Thesis, Carnegie Mellon University, 2005

[Hughes2002] Christopher J Hughes, Vijay S Pai, Parthasarathy Ranganathan, and Sarita V Adve, *Rsim: Simulating Shared-Memory Multiprocessors with ILP Processors*, Computer (2), pp. 40-49, 2002

[Intel4004] Intel Corporation, *Intel's First Microprocessor — The Intel 4004 Fun Facts,* http://www.intel.com/museum/archives/4004facts.htm Website maintained by Intel Corporation [Online] (Retrieved 2 Oct 2009)

[ISO1994] International Organization for Standards (ISO), *Information technology – Open Systems Interconnection – Basic Reference Model*, 1994

[Jalabert2004] Antoine Jalabert, Srinivasan Murali, Luca Benini, and Giovanni De Micheli, *×pipesCompiler: a Tool for Instantiating Application Specific Networks on Chip*, Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, p. 884-889 Vol.2, 2004

[Jantsch2009] Axel Jantsch and Zhonghai Lu, *Resource Allocation for Quality of Service in On-Chip Communication*. In Fayez Gebali and Haytham Elmiligi, editors, Networks on Chip: Theory and Practice. Taylor & Francis Group LLC – CRC Press, 2009

[Jennings1993] Raymond D Jennings, Robert F Linfield, and Michael D Meister, *Network Management: A Review of Emerging Concepts, Standards, and Products, Research report*, National Telecommunications and Information Administration, U.S. Department of Commerce, TR-93-295, 1993

[Kavaldjiev2003] Nikolay Kavaldjiev and Gerald JM Smit, *A Survey of Efficient On-Chip Communications for SoC,* Research report, EWI-CAES: Computer Architecture for Embedded Systems, EWI-DIES: Distributed and Embedded Security, TR-CTIT-03-41, 2003

[Kermani1979] Parviz Kermani and Leonard Kleinrock, *Virtual Cut-through: A New Computer Communication Switching Technique*, Computer Networks (1976) (4), pp. 267-286, 1979

[Keutzer2000] Kurt Keutzer, A Richard Newton, Jan M Rabaey, and Alberto L Sangiovanni-Vencentelli, *System-level Design: Orthogonalization of Concerns and Platform-Based Design*, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on (12), pp. 1523-1543, 2000

[Konstantinidou1994] Smaragda Konstantinidou and Lawrence Snyder, *The Chaos router*, Computers, IEEE Transactions on (12), pp. 1386-1397, 1994

[Kumar2002] Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johnny Öberg, Kari Tiensyrja, and Ahmed Hemani. *A network on Chip Architecture and Design Methodology.* VLSI, Proceedings. IEEE Computer Society Annual Symposium on, pp. 105-112, 2002.

[Lahiri2000] Kanishka Lahiri, Anand Raghunathan, Ganesh Lakshminarayana, and Sujit Dey, *Communication Architecture Tuners: A Methodology for the Design of High-Performance Communication Architectures for Systems-on-Chips,* DAC '00: Proceedings of the 37th conference on Design automation, pp. 513-518, 2000

[Lang1990] Tomas Lang and Lance Kurisaki, *Nonuniform Traffic Spots (NUTS) in Multistage Interconnection Networks*, J. Parallel Distrib. Comput. (1), pp. 55-67, Academic Press, Inc.Orlando, FL, USA, 1990

[Lee2005a] Se-Joong Lee, Kangmin Lee, Seong-Jun Song, and Hoi-Jun Yoo, *Packet-Switched On-Chip Interconnection Network for System-on-Chip Applications*, Circuits and Systems II: Express Briefs, IEEE Transactions on [see also Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on] (6), pp. 308-312, 2005

[Lee2005b] Se-Joong Lee, Kangmin Lee, and Hoi-Jun Yoo, Analysis and Implementation of Practical, *Cost-Effective Networks on Chips*, IEEE Design and Test of Computers (5), pp. 422-433, IEEE Computer SocietyLos Alamitos, CA, USA, 2005

[Leiserson1985] Charles E Leiserson, *Fat-trees: Universal Networks for Hardware-Efficient Supercomputing*, IEEE Trans. Comput. (10), pp. 892-901, IEEE Computer Society Washington, DC, USA, 1985

[Leung2006] Lap-Fai Leung and Chi-Ying Tsui, *Optimal Link Scheduling on Improving Best-Effort and Guaranteed Services Performance in Network-on-Chip Systems*, Design Automation Conference, 2006 43rd ACM/IEEE, pp. 833-838, 2006

[Liang2000] Jian Liang, Sriram Swaminathan, and Russell Tessier, *aSOC: A Scalable, Single-Chip Communications Architecture*, PACT '00: Proceedings of the 2000 International Conference on Parallel Architectures and Compilation Techniques, p. 37, 2000

[Lis2009] Mieszko Lis, Keun Sup Shim, Myong Hyon Cho, and Srinivas Devadas, *Guaranteed in-order packet delivery using Exclusive Dynamic Virtual Channel Allocation*, Research report, Computer Science and Artificial Intelligence Laboratory, MIT-CSAIL-TR-2009-036, 2009

[Lu2005] Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson, and Axel Jantsch. *NNSE: Nostrum network-on-chip simulation environment*. In Proceedings of the Swedish System-on-Chip Conference, 2005

[Lu2009] Zhonghai Lu, Mikael Millberg, Axel Jantsch, Alistair Bruce, Pieter van der Wolf, and Tomas Henriksson. *Flow Regulation for On-Chip Communication.* In Proceedings of the Design Automation and Test Europe Conference (DATE), 2009.

[Mahadevan2005] Shankar Mahadevan, Federico Angiolini, Michael Storgaard, Rasmus Grondahl Olsen, Jens Sparsö, and Jan Madsen, *A Network Traffic Generator Model for Fast Network-on-Chip Simulation*, Design, Automation and Test in Europe, 2005. Proceedings, pp. 780-785, 2005

[Marculescu2008] Radu Marculescu, Umit Ogras, Li-Shiuan Peh, Natalie Enright-Jerger, and Yatin Hoskote, *Outstanding Research Problems in NoC Design: Circuit-, Microarchitecture- and System-Level Perspective*, IEEE Transactions on Computer-Aided Design 1, pp. 3-21, 2009

[Marescaux2002] Théodore Marescaux, Andrei Bartic, Diederik Verkest, Serge Vernalde, and Rudy Lauwerein, *Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs*, Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications, pp. 795-805, 2002

[Marescaux2007] Théodore Marescaux and Henk Corporaal, *Introducing the SuperGT Network-on-Chip; SuperGT QoS: more than just GT*, Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE, pp. 116-121, 2007

[Matlab] The MathWorks, Inc., Matlab and simulink for technical computing, http://www.mathworks.com/ Website maintained by The MathWorks, Inc. [Online] (Retrieved 2 Oct 2009)

[Meincke1999] Thomas Meincke, Ahmed Hemani, Shashi Kumar, Peeter Ellervee, Johnny Öberg, Thomas Olsson, Peter Nilsson, Dan Lindqvist, and Hannu Tenhunen, *Globally Asynchronous Locally Synchronous Architecture for Large High-Performance ASICs*, Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on, p. 512-515 vol.2, 1999

[Mello2004] Aline V de Mello, Luciano C Ost, Ney L V Calazans, and Fernando G Moraes, *Evaluation of Routing Algorithms in Mesh Based NoCs*, Research report, Faculdade de Informatica PUCRS-Brazil, Brazil, TR 040, 2004

[Mello2005] Aline V de Mello, L Tedesco, Ney L V Calazans, and Fernando G Moraes, *Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC*, Integrated Circuits and Systems Design, 18th Symposium on, pp. 178-183, 2005

[Micheli2006] Giovanni De Micheli and Luca Benini, *Networks on Chips: Technology and Tools (Systems on Silicon)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 0123705215, 2006

[Millberg2002] Mikael Millberg, *The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone*, Research report, LECS, ECS, Royal Institute of Technology, Sweden, TRITA-IMIT-LECSR02:01, ISSN1651-4661, ISRN KTH/IMIT/LECS/R-02/01 SE, 2002

[Millberg2004a] Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel Jantsch, *Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip*, Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, p. 890-895 Vol.2, 2004

[Millberg2004b] Mikael Millberg, Erland Nilsson, Rikard Thid, Shashi Kumar, and Axel Jantsch, *The Nostrum Backbone – A Communication Protocol Stack for Networks on Chip*, VLSI Design, 2004. Proceedings. 17th International Conference on, pp. 693-696, 2004

[Millberg2007a] Mikael Millberg and Axel Jantsch, *A Study of NoC Exit Strategies, Networks-on-Chip*, 2007. NOCS 2007. First International Symposium on, pp. 217-217, 2007

[Millberg2007b] Mikael Millberg and Axel Jantsch, *Increasing NoC Performance and Utilisation using a Dual Packet Exit Strategy*, Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on, pp. 511-518, 2007

[Millberg2009] Mikael Millberg and Axel Jantsch, *Priority Based Forced Requeue to Reduce Worst-case Latency for Bursty Traffic*, Design, Automation and Test in Europe Conference and Exhibition, 2009. Proceedings, 2009

[Millman1987] Jacob Millman and Arvin Grabel, *Microelectronics*, 2nd ed., McGraw-Hill, Inc., New York, NY, USA, 1987

[MyHDL] Jan Decaluwe, *Myhdl – From Python to Silicon*, http:// www.myhdl.org/ Website maintained by Jan Decaluwe (Retrieved 23 Sep. 2009)

[ModelSim] Model Technology, *Modelsim – Advanced Simulation and Debugging*, http://model.com/ Website maintained by Mentor Graphics [Online] (Retrieved 2 Oct 2009)

[Moore1995] Simon W Moore and Brian T Grahamy, *Tagged Up/Down Sorter– A Hardware Priority Queue*, Oxford University Press, 1995

[Moraes2004] Fernando G Moraes, Ney L V Calazans, Aline V de Mello, Leandro Möller, and Luciano C Ost, *HERMES: An Infrastructure for Low Area Overhead Packet-Switching Networks on Chip*, Integration, the VLSI Journal (1), pp. 69-93, 2004

[Moscibroda2009] Thomas Moscibroda and Onur Mutlu. *A case for bufferless routing in on-chip networks.* ACM SIGARCH Computer Architecture News, Vol. 37, pp. 196-207, 2009

[MPI1993] Message Passing Interface Forum, *MPI: A message Passing Interface, Supercomputing* '93. Proceedings, pp. 878-883, 1993

[Murali2004] Srinivasan Murali and Giovanni De Micheli, *Bandwidth-Constrained Mapping of Cores onto NoC Architectures*, DATE '04: Proceedings of the conference on Design, automation and test in Europe, p. 20896, 2004

[Murali2006] Srinivasan Murali, Paolo Meloni, Federico Angiolini, David Atienza, Salvatore Carta, Luca Benini, Giovanni De Micheli, and Luigi Raffo, *Designing Application-Specific Networks on Chips with Floorplan Information*, ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design, pp. 355-362, 2006

[Muttersbach1999] Jens Muttersbach, Thomas Villiger, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner, *Globally-Asynchronous Locally-Synchronous Architectures to Simplify the Design of On-Chip Systems*, ASIC/SOC Conference, 1999. Proceedings. Twelfth Annual IEEE International, pp. 317-321, 1999

[Nilsson2002] Erland Nilsson, Design and Implementation of a Hot-potato Switch in a Network on Chip, masters thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, IMIT/ LECS 2002-11, Stockholm, Sweden, June 2002.

[Nilsson2003] Erland Nilsson, Mikael Millberg, Johnny Öberg, and Axel Jantsch, *Load distribution With the Proximity Congestion Awareness in a Network on Chip*, Design, Automation and Test in Europe Conference and Exhibition, 2003, pp. 1126-1127, 2003

[Nilsson2006] Erland Nilsson and Johnny Öberg, PANACEA – *A Case Study on the PANACEA NoC – a Nostrum Network on Chip Prototype*, Research report, School of Information and Communication Technology, TRITA-ICT/ECS R 06:01, 2006

[OCP] OCP-IP, *Open Core Protocol International Partnership*, Open core protocol, http://www.ocpip.org Website maintained by Open Core Protocol International Partnership (OCP-IP) [Online] (Retrieved 6 Oct 2009)

[OSI-Wiki] Wikipedia, The Free Encyclopedia, *OSI model*, http://en.wikipedia.org/wiki/OSI_model, Website maintained by Wikipedia contributors (Retrieved 29 Sep 2009)

[Pamunuwa2004] Dinesh Pamunuwa, Johnny Öberg, Li-Rong Zheng, Mikael Millberg, Axel Jantsch, and Hannu Tenhunen. *A study on the implementation of 2-D mesh-based networks-on-chip in the nanometre regime.* Integration, the VLSI Journal, Vol. 38, pp. 3-17, 2004

[Panades2006] Ivan Miro Panades, Alain Grenier, and Abbas Sheibanyrad, *A Low Cost Network-on-Chip with Guaranteed Service Well Suited to the GALS Approach*, Nano-Networks and Workshops, 2006. NanoNet '06. 1st International Conference on, pp. 1-5, 2006

[Pande2003] Partha Pratim Pande, Cristian Grecu, André Ivanov, and Res Saleh, *Design of a Switch for Network on Chip Applications*, Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on, p. V-217-V-220 vol.5, 2003

[Pande2005] Partha Pratim Pande, Cristian Grecu, Michael Jones, André Ivanov, and Res Saleh, *Effect of Traffic Localization on Energy Dissipation in NoC-based Interconnect*, International Symposium on Circuits and Systems (ISCAS 2005), pp. 1774-1777, 2005

[Patterson2008] David A. Patterson and John L. Hennessy, *Computer Organization & Design, The Hardware/Software Interface*, Fourth Edition, Morgan Kaufmann, San Francisco, CA., ISBN: 978-0-12-374493-7, 2008

[Pavlidis2007] Vasilis F Pavlidis and Eby G Friedman, *3-D Topologies for Networks-on-Chip*, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on (10), pp. 1081-1090, 2007

[Pfister1985] Gregory Pfister and Alan Norton, *`"Hot spot" Contention and Combining in Multistage Interconnection Networks*, pp. 276-281, IEEE Computer Society PressLos Alamitos, CA, USA, 1994

[Python] Python Community, *Python programming language – official website*, http://www.python.org/ Website maintained by Python community (Retrieved 23 Sep 2009)

[Radulescu2004] Andrei Rădulescu and Kees Goossens, *Communication Services for Networks on Silicon Domain-Specific Processor*, In Book: Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation, pp. 193–213., Marcel Dekker, 2004

[Radulescu2005] Andrei Rădulescu, John Dielissen, Santiago González Pestana, Om Prakash Gangwal, Edwin Rijpkema, Paul Wielage, and Kees Goossens, *An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Programming*, IEEE Transactions on CAD of Integrated Circuits and Systems (1), pp. 4-17, 2005

[Rexford1994] Jennifer Rexford and Kang G Shin, *Support for Multiple Classes of Traffic in Multicomputer Routers*, PCRCW, pp. 116-130, 1994

[Rijpkema2001] Kees Goossens Edwin Rijpkema, and Paul Wielage. *A Router Architecture for Networks on Silicon.* Proceedings of Progress 2001, 2nd Workshop on Embedded Systems, 2001

[Rijpkema2003] Edwin Rijpkema, Kees Goossens, Andrei Rădulescu, John Dielissen, Jef L van Meerbergen, Paul Wielage, and E Waterlander, *Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip*, Design, Automation and Test in Europe Conference and Exhibition, p. 10350, 2003

[SIA-Design2007] International Roadmap Committee and I TWGs, *International Roadmap for Semiconductors* – Update (ITRS), Research report, SIA, http://public.itrs.net/Links/2007ITRS/2007_Chapters/2007_Design.pdf, Website International Roadmap Committe (Retrieved 23 Sep 2009) 2007

[SIA-Interconnect2005] International Roadmap Committee and I TWGs, *International Roadmap for Semiconductors* – Update (ITRS), Research report, SIA, http://www.itrs.net/Links/2005ITRS/Interconnect2005.pdf, Website International Roadmap Committe (Retrieved 23 Sep 2009), 2005

[Salminen2007] Erno Salminen, Ari Kulmala, and Timo D Hämäläinen, *On Network-on-Chip Comparison, Digital System Design Architectures*, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on, pp. 503-510, 2007

[Salminen2008] Erno Salminen, Ari Kulmala, and Timo D Hämäläinen, *Survey of Network-on-chip Proposals*, Research report, OCP-IP, http://www.ocpip.org/uploads/documents/OCP-IP_Survey_of_NoC_Proposals_White_Paper_April_2008.pdf, 2008

[Santi2005] Stefano Santi, Bill Lin, Ljupco Kocarev, Gian Mario Maggio, Riccardo Rovatti, and Gianluca Setti, *On the Impact of Traffic Statistics on Quality of Service for Networks on Chip*, International Symposium on Circuits and Systems (ISCAS 2005), pp. 2349-2352, 2005

[Sgroi2001] Marco Sgroi, Michael A Sheets, Andrew C Mihal, Kurt Keutzer, Sharad Malik, Jan M Rabaey, and Alberto L Sangiovanni-Vencentelli, *Addressing the System-on-a-Chip Interconnect Woes through Communication-Based Design*, DAC '01: Proceedings of the 38th conference on Design automation, pp. 667-672, 2001

[SoC-Wiki] Wikipedia, The Free Encyclopedia, System-on-a-chip, http://en.wikipedia.org/wiki/System-on-a-chip, Website maintained by Wikipedia contributors (Retrieved 27 Sep 2011)

[Soteriou2006] Vassos Soteriou, Hangsheng Wang, and Li-Shiuan Peh, *A Statistical Traffic Model for On-Chip Interconnection Networks*, MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation, pp. 104-116, 2006

[Stefan2009] Radu Stefan and Kees Goossens, *Multi-Path Routing in Time-Division-Multiplexed Networks on Chip*, Proc. IFIP Int'l Conference on Very Large Scale Integration (VLSI-SoC), 2009

[Sylvester2001] Dennis Sylvester and Kurt Keutzer, *Impact of Small Process Geometries on Microarchitectures in Systems on a Chip*, Proceedings of the IEEE (4), pp. 467-489, 2001

[SystemC] OSCI, *SystemC*, http://www.systemc.org/ Website maintained by Open SystemC Initiative (OSCI) (Retrieved 24 Sep 2009)

[Tanenbaum2003] Andrew S Tanenbaum, *Computer Networks*, Prentice-Hall PTR, Upper Saddle River, NJ, USA, 2003

[Thid2003] Rikard Thid, Mikael Millberg, and Axel Jantsch. *Evaluating NoC Communication Backbones with Simulation*. In Proceedings of the IEEE NorChip Conference, pp. 27-30, 2003

[Thid2006] Rikard Thid, Ingo Sander, and Axel Jantsch, *Flexible Bus and NoC Performance Analysis with Configurable Synthetic Workloads*, Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on, pp. 681-688, 2006

[Tota2006] Mario R Casu, Sergio V Tota, and Luca Macchiarulo, Implementation *Analysis of NoC: a MPSoC Trace-Driven Approach*, GLSVLSI '06: Proceedings of the 16th ACM Great Lakes symposium on VLSI, pp. 204-209, 2006

[Tota2007] Sergio V Tota, Mario R Casu, Paolo Motto, Massimo Ruo Roch, and Maurizio Zamboni, *A methodology and a Case-study for Network-on-Chip based MP-SoC Architectures*, Nano-Net '07: Proceedings of the 2nd international conference on Nano-Networks, pp. 1-5, 2007

[Towles2002] Brian Towles and William J Dally, *Worst-case Traffic for Oblivious Routing Functions*, SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures, pp. 1-8, 2002

[Vellanki2004] Praveen Vellanki, Nilanjan Banerjee, and Karam S Chatha, *Quality-of-Service and Error Control Techniques for Network-on-Chip Architectures*, GLSVLSI '04: Proceedings of the 14th ACM Great Lakes symposium on VLSI, pp. 45-50, 2004

[VSI2001] Isask'har Walter, Israel Cidon, Ran Ginosar, and Avinoam Kolodny, *Access Regulation to Hot-Modules in Wormhole NoCs*, NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip, pp. 137-148, 2007

[Walter2007] Isask'har Walter, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. *Access Regulation to Hot-Modules in Wormhole NoCs*. NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip, pp. 137-148, 2007

[Wang2005] Hangsheng Wang, Li-Shiuan Peh and Sharad Malik, *A Technology-Aware and Energy-Oriented Topology Exploration for On-Chip Networks*, DATE 05: Proceedings of the conference on Design, Automation and Test in Europe, pp. 1238-1243, 2005

[Wielage2002] Paul Wielage and Kees Goossens, *Networks on Silicon: Blessing or Nightmare?*, DSD '02: Proceedings of the Euromicro Symposium on Digital Systems Design, p. 196, 2002

[Wiklund2003] Daniel Wiklund and Dake Liu. *SoCBUS: switched network on chip for hard real time embedded systems.* Proceedings of the 17th International Symposium on Parallel and Distributed Processing, pp. 78-71, 2003

[Wingard2005] Drew Wingard, *Socket-Based Design Using Decoupled Interconnects*, In Book Interconnect-Centric Design for Advanced SoC and NoC, pp. 367-396, Springer US, Dordrecht, The Netherlands, 2005

[Wolkotte2005] Pascal T Wolkotte, Gerard JM Smit, Gerard K Rauwerda, and Lodewijk T Smit, *An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip*, Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International, p. 155a-155a, 2005

[Wulf1995] William A. Wulf and Sally A. McKee, *Hitting the Memory Wall: Implications of the Obvious*, Computer Architecture News, p. 20-24 vol.23 (no.1), March 1995

[Yang2008] Huimin Du, Xiaoqiang Yang, and Jungang Han, *A Node Coding and the Improved Routing Algorithm in Torus Topology*, Computer Science and Information Technology, 2008. ICCSIT '08. International Conference on, pp. 443-447, 2008

[Ye2003] Terry Tao Ye, Luca Benini, and Giovanni De Micheli, *Packetized On-Chip Interconnect Communication Analysis for MPSoC*, DATE '03: Proceedings of the conference on Design, Automation and Test in Europe, p. 10344, 2003

[Ye2004] Terry Tao Ye, Luca Benini, and Giovanni De Micheli, *Packetization and Routing Analysis of On-Chip Multiprocessor Networks*, Journal of Systems Architecture (2-3), pp. 81-104, Elsevier North-Holland, Inc., 2004

[Zimmermann1980] Hubert Zimmermann, *OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection*, Communications, IEEE Transactions on (4), pp. 425-432, 1980

Appendix

Load distribution with the Proximity Congestion Awareness in a Network on Chip

Erland Nilsson, Mikael Millberg, Johnny Öberg, and Axel Jantsch
Laboratory of Electronics and Computer Systems / Royal Institute of Technology (KTH)
Email: {erlandn, micke, johnny, axel}@imit.kth.se

Pages: 1126-1127 Reference in Thesis – [Nilsson2003]

Note: To give more detail on the concept and provide the full information two versions of this paper have been included; (i) the submitted full length version of the paper that appears in this appendix (ii) the shorter version that appeared in the proceedings and was accepted to the poster session. The latter version is the one included in the thesis.

Submitted to DATE 2003

Load distribution with the Proximity Congestion Awareness in a Network on Chip

Erland Nilsson, Mikael Millberg, Johnny Öberg, and Axel Jantsch
Laboratory of Electronics and Computer Systems,
Royal Institute of Technology (KTH),
LECS/IMIT/KTH-Electrum, Electrum 229, SE-164 40 Kista, Sweden
Email: {erlandn, micke, johnny, axel}@imit.kth.se

Abstract

In networks on chip, NoC, very low cost and high performance switches will be of critical importance. For a regular two-dimensional NoC we propose a very simple, memoryless switch. Packets are never lost. In case of congestion packets are emitted in a non-ideal direction, which is called deflective routing. To increase the maximum tolerable load of the network, we propose a Proximity Congestion Awareness (PCA) technique, where switches use load information of neighbouring switches, called stress values, for their own switching decisions, thus avoiding congested areas. We present simulation results with random traffic which show that the PCA technique can increase the maximum traffic load by a factor of over 20.

1 Introduction

On-chip communication becomes a challenge as the number of transistor functions increases on a single silicon die. Clock and data distribution over large distances is impossible to accomplish in a simple manner. Several research groups propose packet switched Network on Chip [1] [4] [5] to address the problem with communication between Intellectual Properties, IP [10], where each IP-block is a Resource or a part of a Resource.

We propose the use of Nostrum [8], a two-dimensional Network-on-Chip with the hot-potato routing algorithm applied as switching policy for datagram distribution [7]. The hot-potato results in a no-queue system where packets may be deflected in directions contradicting the optimal path in case of a tie between concurrent packets arriving at the same switch cycle [2]. The mesh consists of Switch-Resource pairs where each Resource is able to transmit and receive packets. However, preliminary experiments show that our proposed PCA based switching policy also has a positive effect on the maximum load even for more realistic traffic scenarios based on more localised communication.

There are many ways to organise this mesh. Examples are the flattened torus model [3], where boundary connections are wrapped around the edges to the opposite side. Another model is a plain two-dimensional mesh [4], where boundary connections are reconnected into the same Switch of origin, this is also the way we have choosen to implement our NoC. In the plain two-dimensional mesh, the number of packets passing through the centre of the mesh is significantly higher compared to packets travelling along the edges. As every Resource may transmit to any other Resource in the mesh with equal probability, most packets will pass through the centre, which becomes a hot-spot. Such a hot-spot is not desirable.

However, hot-spots can be avoided if often intercommunicating Resources are placed within proximity to each other. Another way is to make each Switch aware of the load of all neighbouring Switches, by letting a cellular automata select the "best" direction to route each packet. We call this concept Proximity Congestion Awareness, PCA. PCA uses control information called stress values that are sent between nearby Switches. The control information is a measure of the workload of a particular Switch. This information is evaluated on each Switch and transmitted to the four most adjacent Switches every packet cycle. We assume that the ideal Switch should be able to switch all incoming packets in one clock cycle, i.e. one switch cycle is equal to one packet cycle. The simulations made in this paper are based on random traffic where each Resource has the same probability of communication to any other Resource in either way.

The rest of the paper is organised in the following way. In section 2, we describe the routing methodology used in our Switch. In section 3, we describe three different implementations of our Switch, two which are using stress values and one reference Switch which does not. In section 4, we

present results of simulations of the load distribution in the network using the three Switch implementations. Finally in section 5, we draw some conclusions and discuss future work.

2 Switch load distribution

A main objective is to keep the final Switch very simple. The Switch is responsible for communication up to the network layer, in the terminology of the OSI reference model [6]. Every little intelligence added in the Switch results in a larger design, which is expensive in area, speed and not at least power. In later stages, improvements may be added if the benefits exceed the cost. The number of gates between the input and the output of the Switch is the switch gate depth. A higher number of gates through the Switch, larger gate depth, the lower clock frequency is the Switch possible to run at. If a packet is supposed to be delivered in one clock cycle, no buffers used for pipelining can be used. The control logic needs a certain time to make the appropriate decisions and set the multiplexers controlling the outputs. The frequency of packets must not be higher than the corresponding time period. To achieve as short set up times as possible, many tasks must be executed in parallel. On the other hand, it may still be preferable, that improved intelligence that gains the network utilisation may decrease the highest possible packet frequency.

The load in the centre of the mesh is much higher compared to the total average. The maximum number of packets transmitted for a Resource is limited by its load. If the number of packets on transit in a Switch is decreased, the Resource connected to that particular Switch are able to transmit more packets. With lower load, less packets will be deflected and the maximum number of packets that simultaneously can reside in the network can be significantly increased, hence the network throughput is increased.

The load be spread over a larger area by using different routing rules. Examples of routing rules can be:

- Round Robin
- Try to avoid congestion
- Force avoidance

The first rule, Round Robin [11], can be useful when there is some built-in locality of the Resources, i.e. the distance between normally communicating Resources is short. However, in a situation where the source and destination of every packet is more distributed, there will be congestions that limit the network performance.

By trying to avoid congested areas, the network throughput can be improved. For larger networks, there are almost always two directions that are preferable. The cost of taking the other two directions is two switch cycles, one when going in the wrong direction and another one to return back on track again. Depending on the load in the two directions to which a packet is heading, the direction where there is least congestion is the preferred one.

The concept of Proximity Congestion Awareness, can be used to make the load distribution more uniform. Information to help the Switches in their routing decision is sent between the Switches. The information is sent from one Switch to its neighbours in all directions. That is the result of a calculation that relates to the load level in that Switch. The surrounding Switches get information from the Switches in all four directions; this helps the Switch to get a picture of the surroundings, see figure 1. The informative value PCA is using is called stress value.

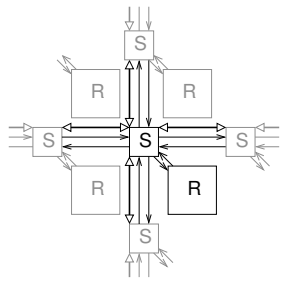


Figure 1. Stress value distribution. The unfilled arrows represents the stress values that are sent between the Switches without interaction of the Resources.

The load information from each Switch progress over the network in the manner of a cellular automata. Cellular automata is basically a concept for making status information about one cell progress to its surroundings. One cell in a mesh can tell its most adjacent neighbours about its status, such as its load. Other cells will not know about one specific cell, instead the status information from several cells are combined and transmitted further. This helps each Switch to decide in what direction each packet should be sent to avoid the hot-spots; where the load is high for a longer time. The status information will not only be about one specific cell, but in addition about how loaded a larger area is in that particular direction.

The evaluation of the stress value can be made in many different ways, for example:

- Number of packets switched
- Number of packets switched, averaged over a few switch cycles

- Any of the above and combined with incoming information from adjacent Switches

The complexity of the Switch is increased in the order in which the methods are presented. The least cost implementation in terms of physical size and execution time is in descending order from the top.

The simplest implementation is to count the number of packets switched and transmit the result to all the neighbouring Switches. The increase in performance is noticeable. Assume that one Switch, Switch A, is heavy loaded at a given time. Switch A will send a high stress value to the adjacent Switches. The surrounding Switches will then hesitate to transmit to Switch A during the next switch cycle. It is not impossible that Switch A does not get any incoming packet in the next switch cycle, the stress value will then decrease to zero. This can proceed over a long period of time. The result is oscillations in the load over the network. To achieve the highest possible network throughput, the number of packets switched in each Switch should remain constant over time and equal over the network.

The goal is to achieve a stable and equally distributed load over the whole network. The utilisation of the network capacity is rapidly increased if there are few Switches running empty. To overcome the problem with oscillations, the number of packets in a Switch could be counted and averaged over a few switch cycles.

To improve the stress value calculation even more, the stress values coming from the surrounding Switches can be accounted for in the new stress value together with the number of outputs used over a few switch cycles. These should be weighted and added together. For example, stress values coming from the surrounding Switches are multiplied by a constant and added together with the number of busy outputs in the Switch itself. In the end, the final value will have to be scaled so that the maximum value is not exceeding the maximum stress value available.

3 Switch design

The Switch must be able to make a connection from any input to any output. This means that there must be five multiplexers with five inputs each, for the two-dimensional network; where each Switch has a connection to a Resource. One input is allowed to be connected to one output only, an incoming connection can only be used once. Duplication of packets must not occur. To accomplish this, a controlling unit makes decisions of what combination the setup of the multiplexers should have. The remaining units in the design are standard units such as buffers and counters.

A packet header includes a destination address, hop-counter and an empty flag telling if the packet is an actual packet or not. The empty flag is used to make a simple packet indicator and to save power. Instead of removing a whole packet from an input buffer when no new packets are waiting, only the flag needs to be changed for notification. The control unit examines every incoming packet for its destination. From the destination address, the destination direction is derived. The destination address can be divided into two parts, the row address and the column address. When using relative addressing, the first bit in the row address is the sign for north or south and the remaining bits are simply the binary number of how many rows in that direction the destination row is. The same relation applies for the column address. If both the row and column address are zero, the packet has reached its final destination.

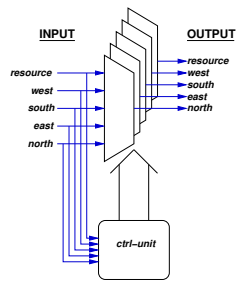


Figure 2. Basic Switch operation.

In the current implementation [9], the priority is a function of the distance, i.e. the age, which is in fact the number of hops since the packet was launched on the network.

The incoming packets are sorted in priority order, the packet with the highest priority gets to make the first choice of output and the following packets in descending order. If two packets have the same priority, the order of which input that makes the first choice is fixed. The remaining inputs with the same priority are given their turn after hand as Round Robin. The packets which preferred output is occupied by a higher priority packet must be forced in a direction that contradicts to the desired. After these operations, if there are any unused outputs, packets can be taken from the Resource.

Methods of avoiding fixed output assignment in case of concurrent packets is either to make a random choice or to have a cyclic choice of which packet that will be favoured.

3.1 With no stress value

It has been revealed that for random traffic, the highest load is in the centre of the network mesh. The amount of packets waiting in the network interfaces of the Resources in the centre Switches are limiting the availability for pack-

ets to be launched there. If many packets are travelling through the centre, packets leaving the centre Resources will have to wait for a long time for an empty output where the new packet can be put. The average number of packets waiting for transmission must not increase over a longer period of time. An empty Switch output is needed to release a packet from the Resource on the network. This occurs when one incoming packet is forwarded to the Resource, or when the number of incoming packets is less than four. If the possibility of a packet creation is greater than the possibility of an empty output in that particular Switch, the number of packets waiting will increase.

The most basic operation is to ignore the hot-spot in the centre. Instead, an easy and natural solution is to avoid packets to travel through the centre by placing Resources with frequent communication close to each other.

3.2 With stress value

The number of packets going through the very centre of the mesh can be spread over a larger area by sending stress values to the surrounding Switches. That information aids surrounding Switches to select paths through Switches with lower load.

When using stress values from the surrounding Switches, the order of outputs is updated for every switch cycle. Four stress values are imported on wires separate from the packets from the most adjacent Switches. The four values are inserted into a sorting unit that renders an output priority order. The output priority order is updated every switch cycle and contain four numbers, with the output directing to the least loaded area is first.

One could imagine, that by adding extra logic to handle the output priority order with the sorting unit and the logic for generating the stress value will increase the gate depth of the whole Switch. However, the stress value sorting unit is applied in parallel with the priority sorting unit while, in the same way, the stress value evaluation works in parallel to the output buffers. This results in a Switch which has a larger area and higher power consumption compared to the simpler one, but with gate depth unchanged.

3.3 Averaged stress value

The stress values are sent from every Switch to its four most adjacent Switches. Hence, every Switch will therefore also receive four stress values, one from each direction. In the previously described functionality, these values are used once directly in the following switch cycle. The reaction to a high stress value is that the surrounding Switches will avoid sending in the direction of the Switch with the high stress value. The high load may the next switch cycle rapidly decrease to a considerable lower value, which in its turn makes the surrounding Switches to prefer that output. The value increases again and will keep on oscillating in this way and so will the packets during its journey. A too low stress value in an area with high load, which could become the case after the Switch reaction of a high stress value, will indicate to all the surrounding Switches that the particular Switch is a good choice for routing.

The solution is to create an average of the stress value over a few switch cycles to prevent the surrounding Switches of making these oscillating routing decisions. It can be done on the stress value input on every Switch but it is better to make the averaged stress value on the Switch it concerns before it is transmitted. The gain is that only one stress value averager per Switch is needed. Instead of adding the stress value during for example four switch cycles, divide by four to get the average and round up to an integer. It could be found better to transmit the result after the adder without dividing. To better distinguish between two nearby values during the stress value sorting phase, a larger range of possible numbers may be used. This will need more wires between the Switches but averaging logic is saved at the cost of larger stress values. The implementation using averaged stress values are averaging over four cycles used and a value between zero and fifteen is sent. The stress value averager in the implemented model is fairly small, about 200[1] gates.

3.4 Gate depth

The maximum clock period has an upper limit which is related to the distance between the input and the output. When a clock pulse occurs, every signal in the design must stabilise before the next clock pulse. This restricts the number of following gates between input and output. If the gate depth exceeds the maximum allowed depth, pipelining must be used. The negative effect of using pipelining, is that a packet can not be forwarded the next clock cycle.

4 Simulation results

4.1 FIFO-buffer study

Packets that are sent from the Resource that are intended to be launched on the network are first put in a FIFO-buffer in order to wait for an empty slot on the Switch. For a low loaded Switch, there are no packets waiting in the FIFO, since packets from the Resource passes through the buffer and to an empty output on the Switch the following packet cycle. The study of all FIFOs in the network mesh is a method to see how the load is distributed around a hot-spot. A visualisation of the average number of packets in every

[1] 218 gates using the lsi10k-library.

FIFO is made. A load of 0.01 tells us that the FIFO is occupied by one packet every 100 switch cycle, which is fairly low.

The intention is now to show the real difference between the three cases of using stress value.

- with no stress value
- with stress value
- with averaged stress value

In the three cases, the same input data has been used. The mesh size is quite large, 16×16, to be able to spread the load over a larger area to avoid the influence of boundary effects. The packet probability is for the 'no stress value' case close to the maximum possible, shown by simulations [9], to create as much congestions as possible, to be able to make a clear separation where the use of stress value really brings matters to its head. Observe the scaling of the x-axis since every bar in the whole figure is scaled from the Switch with the maximum average load.

4.1.1 With no stress value

In figure 3, the hot-spot is pushed to the north-west corner, if north is up in the figure, which is the logical reference direction. The reason for the non-centered load is a result of the routing decisions in the Switch. If two packets arrive to an intended destination at the same time, one packet is captured by the Resource while the other is deflected. In this case, it is a fixed choice of deflection output order. First north, then west, south, and last east.

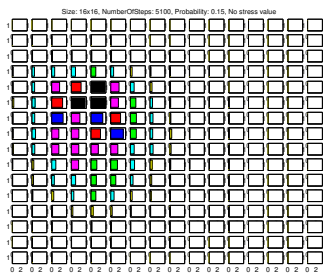


Figure 3. Average load in FIFOs without using stress value. The largest average number is 3.2 packets waiting in the FIFO.

4.1.2 With stress value

The stress value notifies the surrounding Switches about how many packets the Switch handles during that cycle. The stress value is updated every switch cycle and is not dependent on the previous values. Using stress values in this manner increases the performance of the Switch since the outputs are ordered in the most preferable order. Compare

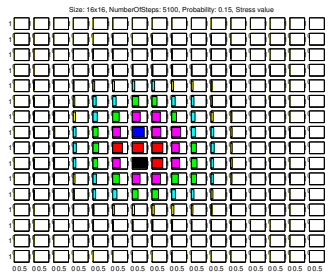


Figure 4. Average load in FIFOs using stress value. The largest average number is 0.9 packets waiting in the FIFO.

the maximum value on the x-axis in figure 4, which has a maximum value of 0.9 to the value in figure 3, which is 3.2. The number of packets waiting in the buffers has decreased with approximately a factor of 3.5 by using stress values.

4.1.3 Averaged stress value

The averaged stress value is the sum of the four last stress values, see section 3.3. The number of Switches that presents a high load is greater than before. The load is therefore more distributed over the mesh when the averaged stress value is used. In comparison to the previous figures, the load is not only close to maximum in a few Switches, instead, a larger area of the network shares the work of transporting packets through the centre zone. From the figure, the maximum average load can be found by estimating the value of the axis at the right end of the x-axis. With experience of the visualised data, the maximum average load is estimated to be 0.15. Compared to the implementation using stress value with no averaging, the average load now achieved is six times less compared to the non averaged stress value. In relation to the most basic implementation where no stress value was used, it is enhanced with a factor of more than 20.

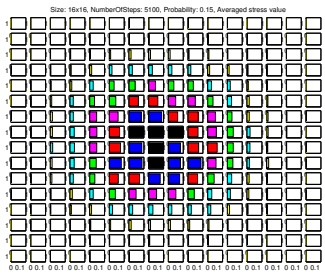


Figure 5. Average load in FIFOs using averaged stress value. The largest average number is 0.1 packets waiting in the FIFO.

4.2 Synthesis

The synthesis of the Switch was made using the lsi10k-library. Todays technology is improved many times compared to the technology of the library used in the synthesised model. Still, the number of gates are likely to be in the same order but the delays extracted from the synthesis are much less with today's technology.

The final Switch has been synthesised with Synopsys using two different constraints, first the default which is optimised on area and second optimised on speed. When speed is considered, the area may be increased rapidly since extra logic is used in parallel for making the gate depth shorter.

In table 1, it is shown in what order the total combined logic and the critical path gate depth are needed to accomplish the Switch depending on the optimisation constraints.

constraint	*total combined logic*	*critical path gate depth*
area	13 964	79
speed	21 029	48

Table 1. Number of gates and gate depth with averaged stress value using the lsi10k-library.

5 Conclusion

It can clearly be seen from the previous discussion that the implementation of a more balanced load using stress values increases the network throughput and decreases the packet delivery time. The network load is decreased with a factor of 20 for a heavily loaded network. When the network becomes more loaded, all outputs are occupied and the output priority makes no effect since there are always packets in all buffers. All simulations in this paper has been made using random traffic. However, it is shown in further experiments in preliminary results that the positive effect of using PCA is also valid for a traffic model where the probability for communication between nearby Resources is higher compared to Resources on far distance. In fact, we expect that the effect of "smearing out" high load over a large network increases the highest tolerable network load for almost any kind of traffic. In a less loaded network, the stress values together with the output priority order are not necessary since most packets get routed the shortest path anyway and almost always there are free outputs in every Switch for every switch cycle. If there are no packets waiting in the FIFO buffers, the gain of using stress values is not as evident as before.

References

[1] L. Benini and G. D. Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, pages 71–78, January 2002.

[2] I. Caragiannis, C. Kaklamanis, and I. Vergados. Greedy dynamic hot-potato routing on arrays. In *Parallel Architectures, Algorithms, and Networks*, December 2000.

[3] W. Dally and C. Seitz. The torus routing chip. In *Parallel Architectures, Algorithms, and Networks*, California Institute of Technology, 1986.

[4] W. J. Dally and B. Towels. Route packets, not wires: On-chip interconnection networks. In *Proceedings of Design and Automization Conference (DAC)*, 2001.

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

[6] J. Irvine and D. Harle. *Data Communications and Networks: An Engineering Approach*. John Wiley & Sons, Ltd, 2002.

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

[8] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The nostrum backbone - a communication protocol stack for networks on chip. In *submission to Design, Automation, and Test in Europe (DATE)*, March 2003.

[9] E. Nilsson. Design and implementation of a hot-potato switch in a network on chip. Master's thesis, KTH, June 2002. IMIT/LECS 2002-11.

[10] I. Saastamoinen, D. Siguenza-Tortosa, and J. Nurmi. Interconnect ip node for future system-on-chip designs. In *Proceedings of The First IEEE International Workshop on Electronic Design, Test and Applicatins*, 2002.

[11] W. Stallings. *Data and Computer Communications*. Prentice Hall, fourth edition, 1994.