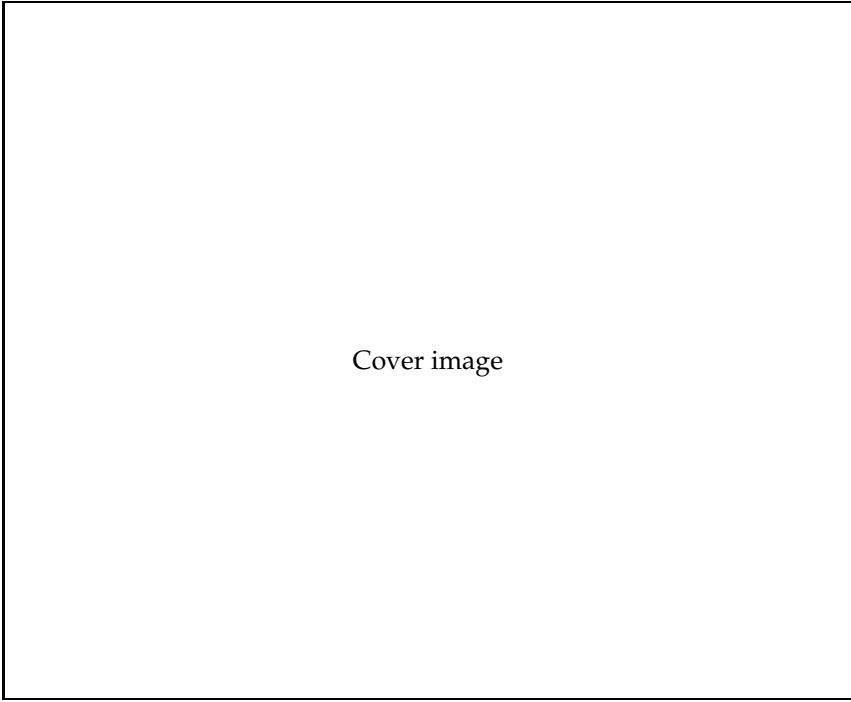


Linköping Studies in Science and Technology
Dissertation No. 932

Development and Performance Evaluation of Networks on Chip

Daniel Wiklund



Cover image

Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2005

Linköping Studies in Science and Technology
Dissertation No. 932

Development and Performance Evaluation of Networks on Chip

Daniel Wiklund



INSTITUTE OF TECHNOLOGY
LINKÖPINGS UNIVERSITET

Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2005

ISBN 91-85297-62-3

ISSN 0345-7524

**Development and Performance
Evaluation of Networks on Chip**

Daniel Wiklund

ISBN 91-85297-62-3

Copyright © Daniel Wiklund, 2005

Linköping Studies in Science and Technology

Dissertation No. 932

ISSN 0345-7524

Department of Electrical Engineering

Linköping University

SE-581 83 Linköping

Sweden

Phone: +46 13 28 10 00

Author e-mail: daniel.wiklund@ieee.org

Cover image

Visualization of the basestation application mapping in figure 10.4. The image has been generated using POV-Ray 3.5.

Printed by UniTryck, Linköping University

Linköping, Sweden, 2005

Abstract

Along with Moore's law there is a continuous development in architectures for electronic systems. Currently there is a trend towards integration of more and more processing elements, e.g. general-purpose processors and DSPs, onto a single chip. With the increasing complexity of such systems come difficulties in creating a proper communications infrastructure for the chip. When time-division buses and custom point to point communication are no longer sufficient, more elaborate networks are the obvious choice. By turning from the current path of buses and custom communication designs for the higher levels of interconnection on the chip, it is possible to reach high performance with lower design and verification costs.

This thesis presents a circuit-switched network for on-chip use that has been developed with signal processing tasks in mind. The network implementation is simple and thus area efficient while being able to operate at high speed. Circuit-switched technology has the advantage of allowing very simple network components while giving high performance for many applications in the telecom area where intra-system communication often can be scheduled tightly according to the performance requirements of the system.

Parts of the design flow CAD support have been implemented along with the network components. An extendable, event-driven simulator has been developed that allows for updates and elaboration of both network components and processing modules.

The simulator has been used as the basis to develop a general method

for benchmarking of networks on chip, where the end result should be comparable across different platforms and implementations. The performance of complex systems such as networks is not easily expressed analytically. Thus, the simulator is of paramount importance in assessing the performance of the network in an application.

The network implementation and simulation environment have been used for analysis of some applications. Applications that have been more thoroughly investigated are a single chip Internet core router and the baseband part of a 3G WCDMA/FDD radio basestation. The core router showed a performance in excess of 14 Gbit/s per port at 16 ports with realistic traffic. The 3G basestation application showed the applicability of the network for systems with lower requirements on communication bandwidth where significant savings in design effort can be made through the simplicity of the network system.

Preface

The contents of this thesis present the research that I have done during the last five years. Parts of the research have been presented on conferences. The network-on-chip part has been presented with the following conference papers:

- **Daniel Wiklund**, Sumant Sathe, and Dake Liu, "Benchmarking of on-chip interconnection networks", in *Proceedings of the International Conference on Microelectronics (ICM)*, Carthage, Tunisia, Dec 2004
- **Daniel Wiklund**, Sumant Sathe, and Dake Liu, "Network on chip simulations for benchmarking", in *Proceedings of the International workshop on SoC for real-time applications (IWSOC)*, Banff, Canada, July 2004
- Sumant Sathe, **Daniel Wiklund**, and Dake Liu, "Design of a switching node (router) for on-chip networks", in *Proceedings of Int'l conference on ASIC (ASICON)*, Beijing, China, Oct 2003
- **Daniel Wiklund** and Dake Liu, "SoCBUS: Switched network on chip for hard real time embedded systems", in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, Apr 2003
- **Daniel Wiklund** and Dake Liu, "Design of a system on chip switched network and its design support", in *Proceedings of the International conference on communications, circuits and systems (ICCCAS)*, Chengdu, China, July 2002, pp 1279-1283

- **Daniel Wiklund** and Dake Liu, "Switched interconnect for system-on-a-chip designs", in *Proceedings of the IP2000 Europe conference*, Edinburgh, Scotland, Oct 2000, pp 185-192

The applications part of the research has not yet been presented at conferences but there are two submitted articles:

- **Daniel Wiklund** and Dake Liu, "Design of an Internet core router using the SoCBUS network on chip", submitted manuscript
- **Daniel Wiklund** and Dake Liu, "Design, mapping, and simulations of a 3G WCDMA/FDD basestation", submitted manuscript

Apart from these publications, there is a number of related or unrelated publications that will not be touched upon in this thesis:

- **Daniel Wiklund**, "Processing and memory requirements for a 3G WCDMA basestation baseband solution", in *Proceedings of the Swedish System-on-Chip Conference (SSoCC)*, Båstad, Sweden, Apr 2004
- Tomas Henriksson, **Daniel Wiklund**, and Dake Liu, "VLSI implementation of a switch for on-chip networks", in *Proceedings of Int'l workshop on Design and diagnostics of electronic circuits and systems (DDECS)*, Poznan, Poland, Apr 2003
- **Daniel Wiklund**, "Mesochronous clocking and communication in on-chip networks", in *Proceedings of the Swedish System-on-Chip Conference (SSoCC)*, Eskilstuna, Apr 2003
- Dake Liu, **Daniel Wiklund**, Erik Svensson, Olle Seger, and Sumant Sathe, "SoCBUS: The solution of high communication bandwidth on chip and short TTM", in *Proceedings of the Real-Time and Embedded Computing Conference (RTECC)*, Gothenburg, Sweden, Sep 2002
- **Daniel Wiklund**, "Implementation of a behavioral simulator for on-chip switched networks", in *Proceedings of the Swedish System-on-Chip Conference (SSoCC)*, Falkenberg, Sweden, Mar 2002

- **Daniel Wiklund**, “Switched interconnect for embedded System-on-a-Chip signal processing designs”, in *Proceedings of the Swedish System-on-Chip Conference (SSoCC)*, Arild, Sweden, Mar 2001

Acknowledgments

**I know you half as well as I should have,
and like you half as much as you deserve**

Bilbo Baggins in "Lord of the Rings" (J.R.R. Tolkien)

Reaching the point in my life where I am able to present this booky thingy is not my work alone. There are many persons that deserve to be thanked for all they have done for me and my work.

- My supervisor Prof. Dake Liu. Thanks for letting me get the opportunity to do my Ph.D. together with you. It's been a great time!
- Prof. Christer Svensson for all interesting and stimulating discussions, both on research and more or less everything else.
- Dr. Anders Edman for many discussions and ideas on how to continue the research and attack the problems that have arisen.
- Dr. Kalle Folkesson for being such a whacko friend! No one can confuse people like you – you're the master! *"Vill du något? Du tittar så konstigt på mig."*
- The old "inner circle": Dr. Daniel "Bunnywhitewizard" Eckerbert, Dr. Henrik Eriksson, Dr. Tomas Henriksson, Dr. Ulf Nordqvist, and Lic. Eng. Mikael Olausson.
- The slightly newer friends from Computer Engineering: Eric Tell, 2 x Anders Nilsson (to both of you – hey, I saved some typing!), Andreas Ehliar, Johan Eilert, and Per Karlström.

- The neighboring friends at Electronic Devices: Lic. Eng. Stefan Andersson, Lic. Eng. Peter Caputa, Henrik Fredriksson, and Martin Hansson.
- To the team closely involved with me in teaching, Tomas Svensson, Dr. Olle Seger, and Camilla Eidem, I say "*Docendo discimus*".
- Ylva Jernling, Anna Folkesson, and Ingegärd Andersson for making all the tedious and complicated administrative stuff both bearable and simple!
- All other current and previous members of both Computer Engineering and Electronic Devices that I have had the opportunity to know. None mentioned, none forgotten – you know who you are anyway. I have had a great time in both groups!
- And last but definitely not least: My parents, Anna and Erling, for always supporting me – and because they still allow me to live in their house. *Tack för allt!*

Daniel Wiklund
Linköping, February 2005

Contents

Abstract	iii
Preface	v
Acknowledgments	ix
List of figures	xix
List of tables	xxiii
Abbreviations	xxv
I Background	1
1 Introduction	3
1.1 Background	3
1.2 Dissertation objective and scope	5
1.2.1 Network on chip design	6
1.2.2 Interconnection benchmarking	7
1.2.3 Applications on the SoCBUS platform	7
1.3 Background of the SoCBUS project	7
1.4 Contributions	8
1.5 Dissertation overview	8
Bibliography	9

2	Heterogeneous Multi-Processor Systems	11
2.1	Introduction	11
2.1.1	Parallelization of applications	12
2.2	Platform-based design	13
2.3	Evolution of processor-based platforms	14
2.4	Communication-centric platform design	15
2.4.1	Dependencies	16
2.5	High-level application/platform design	17
2.5.1	Profiling of applications	17
2.5.2	Selection of processing elements	17
2.6	Application implementation	18
2.6.1	Application mapping	19
2.6.2	Scheduling of real-time applications	19
	Bibliography	20
II	On-Chip Communication Infrastructures	23
3	On-Chip Communication Infrastructures	25
3.1	Communication environment	25
3.1.1	Internal synchronization	26
3.1.2	Area and power considerations	26
3.1.3	Comparison with parallel computer networks	27
3.2	Challenges and opportunities	27
3.2.1	Physical issues	28
3.2.2	Logical issues	30
3.2.3	Design efficiency issues	32
3.3	Traditional interconnects	34
3.4	Generalized communication structures	35
	Bibliography	36
4	Networks on Chip	39
4.1	The impact of context	39
4.2	The OSI model	40

4.3	Quality of service	42
4.4	Network topology	43
4.4.1	Theoretical performance of topologies	43
4.4.2	Properties of the 2D mesh and torus	44
4.4.3	Advantages with arbitrary topologies	44
4.5	Packet and circuit switching	45
4.5.1	Latency	46
4.5.2	Wormhole routing	47
4.5.3	Virtual circuits in packet networks	47
4.6	Routing in networks on chip	48
4.6.1	Nonminimal routing	48
4.6.2	Source and distributed routing	48
4.6.3	Local vs. global knowledge	49
4.7	Deadlock avoidance	50
4.7.1	Turn-model routing	50
4.7.2	Virtual channels	51
4.7.3	Circuit-switched networks	52
	Bibliography	53
 III SoCBUS		55
5	The SoCBUS Network	57
5.1	Introduction	57
5.2	Network components	57
5.3	SoCBUS architecture	59
5.4	Configuration and control layering	61
5.5	Network transaction handling	62
5.5.1	Route setup flow	62
5.5.2	PCC: Packet-connected circuit	63
5.6	Routing	64
5.6.1	Distributed routing	64
5.6.2	Source routing	64
5.7	Physical links	65

5.8	Link level protocol	66
5.8.1	Long packets	66
5.8.2	Speculative sending of data	68
5.8.3	Short packets	69
5.9	Router implementations	69
5.9.1	The first router	70
5.9.2	Short packet router	71
5.9.3	Speculative sending support	72
5.10	The fourth router implementation	73
5.10.1	Configurable and parameterized	73
5.10.2	Micro architecture	74
5.10.3	Request path	76
	Bibliography	77
6	Design and Simulation Environment	79
6.1	Design environment	79
6.2	Design flow for SoCBUS	80
6.2.1	Customer design flow	81
6.2.2	Tool coverage	82
6.3	Simulation flow	82
6.4	Traffic modeling	83
6.4.1	Stimuli files	85
6.5	Stimuli generator	85
6.6	Simulator architecture	86
6.6.1	Simulation event handling	88
6.6.2	Simulation models of network components	89
6.7	Network generator	90
6.8	Network generator architecture	90
	Bibliography	91
IV	Benchmarking of Interconnect Structures	93
7	Benchmarking in General	95

7.1	Motivation	95
7.2	Definitions	96
7.3	Performance metrics	96
7.4	Average vs. worst-case performance	97
7.5	Measurement techniques	98
7.6	Comparing results	98
7.7	Processor benchmarking	99
7.8	Network benchmarking	99
	Bibliography	99
8	Benchmarking of On-Chip Interconnects	101
8.1	Benchmarking of interconnects	101
8.1.1	Benchmarking method	102
8.1.2	Benchmark specification	103
8.1.3	Interpretation of results	103
8.2	Benchmarking examples	104
8.2.1	Example 1: Specification	104
8.2.2	Example 1: Results	105
8.2.3	Example 2: Specification	107
8.2.4	Example 2: Results	107
	Bibliography	109
V	Applications	111
9	Internet Core Router	113
9.1	Brief introduction to core routers	113
9.2	Core router processing flow	114
9.3	Function mapping	116
9.4	Simulation setup	117
9.5	Simulation results	118
9.5.1	Traffic patterns	118
9.5.2	Internet mix simulations	119
9.5.3	RFC2544 simulations	120

9.5.4	Minimum size packets simulations	120
9.6	Conclusions	121
	Bibliography	122
10	WCDMA/FDD Basestation	125
10.1	A brief introduction to WCDMA/FDD	125
10.2	Basestation processing flow	126
10.2.1	Downlink	126
10.2.2	Uplink	127
10.3	High-level design specification	128
10.4	Function mapping	128
10.5	Processing subsystems	130
10.5.1	Multipath search and Rake	130
10.5.2	Deinterleaving and rate matching	131
10.5.3	Radio-frame reassembly and deinterleaving	131
10.5.4	Viterbi and Turbo decoding	131
10.5.5	CRC checking	132
10.6	Processing and communication scheduling	132
10.6.1	Schedule analysis	133
10.6.2	Latency-induced storage	134
10.7	Simulation results	135
10.7.1	Minimum network frequency	135
10.7.2	Network usage	136
10.7.3	Control messages vs. transmission schedule	137
10.8	Conclusions	137
	Bibliography	137
VI	Conclusions and Future Work	139
11	Conclusions	141
11.1	Design of networks on chip	141
11.2	Performance evaluation	142
11.3	Application case studies	142

12 Future Work	143
12.1 Networks on chip	143
12.2 Extensions to the tool chain	143
12.3 System simulator	144
12.3.1 System simulator integration	144
12.3.2 Cycle and bit true simulations	145
12.3.3 Challenges	145
12.4 Application case studies	146
Bibliography	146
VII Appendix	147
A Tool Implementation Details	149
A.1 Details on the simulation flow	149
A.1.1 Stimuli generator implementation	149
A.1.2 Simulator implementation	150
A.2 Simulation models of network components	153
A.3 Tool usage	154
A.3.1 Stimuli generator	154
A.3.2 Simulator	154
A.3.3 Network generator	155
Bibliography	156
B XML Formats	157
B.1 Notation	157
B.2 General specifications	158
B.2.1 Model names	158
B.2.2 Frequencies, times, and lengths	158
B.3 Traffic model	158
B.3.1 XML structure	158
B.4 Network model	161
B.4.1 XML structure	162
Index	167

List of Figures

1.1	Hardware and software development cost vs. feature size	5
1.2	Relative market share for different electronics products	6
2.1	Speedup for parallel computers (Amdahl's law)	12
2.2	Classical bus-centric platform with a single CPU	14
2.3	New platform architecture with multiple processing elements (PEs) each containing one (or more) processors	15
2.4	Integration flow for software on top of a hardware platform	18
3.1	Transistors per chip vs. feature size	27
3.2	Bus based two-processor system (IC/DC = Instruction/-data cache)	35
3.3	Hierarchical bus/network architecture	36
4.1	2D mesh topology (4x4). Dashed line marks bisection.	44
4.2	2D torus variations (4x4). Dashed line marks bisection.	45
4.3	Example of an arbitrary topology	46
4.4	Turn model routing	51
4.5	Deadlock avoidance by virtual channels	52
5.1	Network-connected processing tile	58
5.2	A 3x3-node switched network with wrappers and subsystems	59
5.3	Interface between routers and wrappers within the network	61

5.4	Two successful circuit setups	62
5.5	Signals in a unidirectional link	65
5.6	Basic link protocol	68
5.7	Link protocol for speculative sending	68
5.8	Link protocol for short packets	69
5.9	Block diagram for the first two routers	70
5.10	Number of ports vs. router area for the first router	71
5.11	Block diagram for the third router (with speculative sending)	72
5.12	Block diagram for the fourth router	73
5.13	Micro architecture for the fourth router	75
6.1	Design flow for systems based on network on chip	80
6.2	SoCBUS simulation flow	82
6.3	Traffic modeling flow	84
6.4	Translation from traffic model to stimuli	86
6.5	Simulator top level flow	87
6.6	Simulator class hierarchy	88
6.7	Simulator event handling	88
6.8	State machine for the model of the first router	90
6.9	Network generator top flow	91
8.1	Throughput for random traffic on 2D mesh	106
8.2	Throughput for traffic with locality on 2D mesh	106
8.3	Throughput for 2D torus	107
8.4	Throughput for 3D mesh	108
8.5	Throughput for 3D torus	108
9.1	Schematic view of the Internet	114
9.2	Dataflow in the core router	115
9.3	Core router system architecture	116
9.4	Final network allocation and mapping	117
9.5	Internet mix: Average packet latency	119
9.6	Internet mix: SoCBUS router-port lock	119
9.7	RFC2544: Average packet latency	120

9.8	Min size: Average packet latency	121
9.9	Min size: Source port locking	122
10.1	Spreading function	126
10.2	Downlink transmission flow	127
10.3	Uplink reception flow	128
10.4	Architectural mapping of the basestation	130
10.5	Worst-case uplink processing schedule (not to scale)	133
10.6	First try successful routing	135
10.7	Maximum circuit-setup latency	136
A.1	SoCBUS simulation flow	150
A.2	Stimuli generator program flow chart	151
A.3	Simulator flow chart with partial message handling for a router model	152
B.1	Tag structure for a traffic model/test case	165
B.2	Tag structure for a network model	165

List of Tables

3.1	Some challenges in on-chip communication	28
4.1	The seven OSI layers and their coverage in the typical NoC	41
4.2	Theoretical performance of different network topologies as- suming N connected cores.	43
5.1	Configuration and control layering	61
5.2	Request format	66
5.3	Functions for the req0[3:0] signals	67
5.4	Link feedback signals	68
8.1	Comparison of DSP and NoC benchmarks	102
9.1	Packet-size distribution for Internet mix	118
10.1	Block types	129
10.2	Longest possible processing times in the reception flow . .	134
B.1	Notation of tags	157
B.2	XML tag description for the traffic model	160
B.3	XML tag description for the network model	163

Abbreviations

3G	Third generation mobile telecommunication
API	Application programming interface
ASIC	Application-specific integrated circuit
BIST	Built-in selftest
CAM	Content-addressable memory
CRC	Cyclic redundancy check
DSM	Deep sub-micron
DSP	Digital signal processor
FDD	Frequency-division multiplex
FIFO	First in, first out
Flit	Flow control digit, the basic transmission unit of a network
FPGA	Field-programmable gate array
GALS	Globally asynchronous, locally synchronous
GPP	General-purpose processor
IP	Intellectual property
MAC	Media access controller
MOC	Model of computation
MPI	Message-passing interface
NoC	Network on chip
NRE	Non-recurrent engineering
OSI	Open systems interconnect
OVSF	Orthogonal variable spreading factor

PCC	Packet-connected circuit
PLL	Phase-locked loop
PTP	Point-to-point
QoS	Quality of service
RTL	Register transfer level
SoC	System on chip
TCP/IP	Transmission control protocol / Internet protocol
TDM	Time-division multiplex
WCDMA	Wideband code-division multiple access
XML	Extensible markup language

Part I

Background

Chapter 1

Introduction

The future, according to some scientists, will be exactly like the past, only far more expensive.

John Sladek

1.1 Background

The electronics industry has come a long way since the first integrated circuit was built in the late 1950s. Possible circuit complexity continues to follow Moore's law with no apparent end in sight other than due to economic reasons.

With the evolution in circuit technology, the complexity of the designs follow, albeit slower. Thus there is an increasing gap between the possibilities of the technology and the extent to which it is possible to utilize this technology. This problem is known as the "design gap".

New hardware platforms and design methodologies must be used to help close the design gap. The hardware platforms are very much the same as they have always been since the dawn of the semiconductor computer in the early 1970s. Systems typically consist of a single (Harvard) processor, perhaps some acceleration hardware, a couple of local memories, and some peripherals. All components are connected through a single time-division multiplex (TDM) bus. This type of platform can only go so far and the industry is now on the verge of a new era in electronics design. The old-style platforms must be superseded by something new

that is more scalable.

The second generation of platforms, which are being deployed today resembles the first generation. The difference is that the single processor generally has been replaced by one microcontroller and one digital signal processor. These can coexist on a TDM bus by using extensions that allow multiple memory accesses. It is not possible to generalize this type of platform to multiprocessor because of the inherent limitation of the TDM bus.

The current trend is to go for heterogeneous multi-processor systems. As the number of processors go beyond one (or two), many new challenges appear. Challenges such as integration, communication, and software development is just a few of the issues that have to be researched further to allow for smooth development.

Complexity of software development is a huge obstacle towards success of multi-processor systems. This is obvious from the personal computer market where small multi-processor systems have been available for at least ten years without much success except for the server market. The reason being the difficulty in exploiting parallelism in software. Even so, these personal computers use the simpler programming paradigm of shared memory abstraction rather than the more involved message-passing abstraction.

When looking into single-chip solutions for complex systems, such as Internet gateways, mobile telecommunication basestations, the complexity of the problem dictates solutions with fairly many processing nodes. Many processing nodes in a system will inevitably result in message passing since shared memory is inherently bad when it comes to scaling [1].

Even for on-chip systems there is a significant software development cost. The cost for software development actually increases faster than hardware development cost as can be seen in figure 1.1. A report from IBS shows that already at 90 nm technology, the software has a higher cost than the hardware [2]. Since software is easier to update, it is quite obvious why system companies wish to define the behavior of their products more through software, thus prolonging the lifespan of the relatively

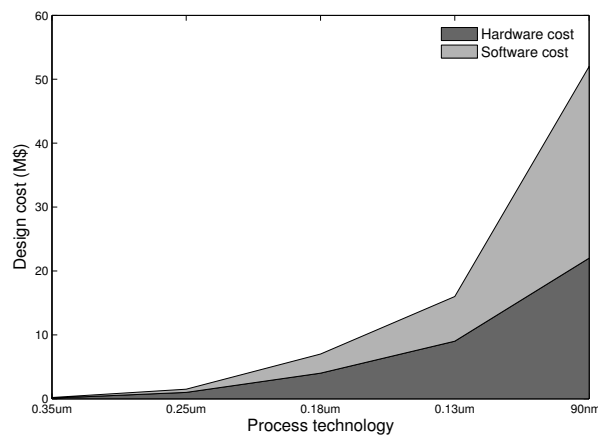


Figure 1.1: Hardware and software development cost vs. feature size

expensive and risky hardware platform. All evidence points to that this trend will continue for the foreseeable future and beyond.

There is already a clear trend towards a larger share of the total revenue in the electronics industry being generated by integrated (embedded) processors. Figure 1.2 shows a graph of the relative market share for different product categories. The total revenues also increases and it is projected that the embedded processor market will reach \$100B around year 2010.

What is not evident from the IBS report is the on-chip communication infrastructure and how this will be designed in the future. With an ever increasing number of processing cores integrated on a single chip this will become a severe bottleneck, both in terms of bandwidth and latency as well as software implementation issues.

1.2 Dissertation objective and scope

As outlined above, the number of upcoming issues with electronics design in deep submicron technology is daunting. Of all the challenges posed on research due to these, this thesis aims at two particular areas:

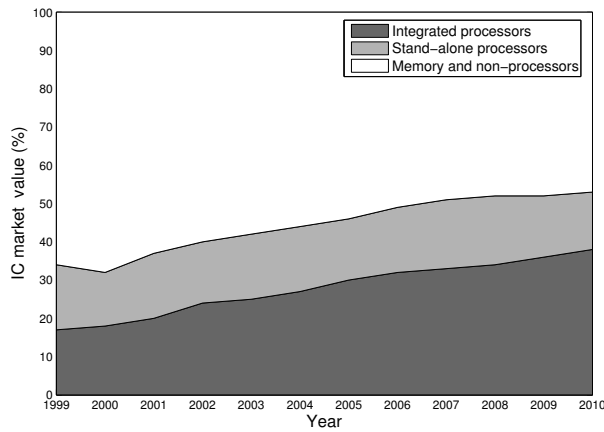


Figure 1.2: Relative market share for different electronics products

network on chip (NoC) design and interconnection benchmarking. The second area in turn evolved into a full-scale investigation of a couple of applications, integrated using the NoC.

1.2.1 Network on chip design

Design of networks on chip is a difficult process since the design space is so vast. There are a large number of design decisions that has to be made during the design process. Many of the design decisions have an impact on the resulting system that is very hard to predict at design time.

This thesis presents the design of a network on chip that is primarily intended to efficiently utilize silicon area, yielding small components with high performance. The issues presented is basic network theory with a preference towards on-chip networks, topology, routing, and the design and implementation of the SoCBUS project in more detail.

The objective of the on-chip communication architectures part of this thesis is to give an introduction to the area, show a possible solution, and evaluate this solution.

1.2.2 Interconnection benchmarking

With the transition to more complex systems on chip where many sub-systems should communicate, there is an increasing need to assess the performance required for the application and that given by the platform. The latter problem has been addressed in this thesis.

The objective is to give a general methodology as well as a couple of generic benchmarks that can be applied on any interconnection structure (within certain limits).

1.2.3 Applications on the SoCBUS platform

Besides the generic traffic cases mentioned in the previous section, two applications have been investigated further. These applications are an Internet core router for 10 Gbit/s networking and the baseband part of a 3G WCDMA/FDD radio basestation.

1.3 Background of the SoCBUS project

The SoCBUS project started as a research project in the summer of 1999. At that time, in principle no network on chip research had yet been published. The project started with an investigation into possible solutions and it was early decided to go for a simple network architecture that would allow small and efficient implementations of network components to be made.

The intention was that the network should be as generic as possible with no inherent limitations on issues such as traffic types, component synchronization, and process technology. Thus, no assumptions were made about these issues at the early stages.

More or less all aspects of networks on chip have been touched upon during the project although many topics have not been investigated deeply and are beyond the scope of this thesis.

1.4 Contributions

As described earlier, there are three main areas for contributions presented in this thesis: NoC design, interconnection benchmarking, and application mapping on the NoC platform.

The first area, NoC design, shows a complete network that has been developed with typical digital signal processing and network processing applications in mind. The network is based on a practical design that is usable for most systems with multiple processors. This research project aims for a solution that is intended to be practical and not an “on-chip Internet”. The network has support for arbitrary topologies and is completely relaxed with respect to clocking and handshaking. This will give a simple to use, low-constrained interconnection environment. Also, the total area cost for a network on chip has to be low, maximally around 5-10% of the chip size. This constraint will limit the component implementations and the total buffer memory in the network.

Since there were no publications on interconnect performance evaluation available, a benchmarking method was developed. This thesis also presents this methodology for interconnection system benchmarking from concept to example. The methodology is simple to use and will yield fair results for comparisons. This has been applied to a set of generic benchmarks for illustrative purposes.

The application mapping shows the feasibility and implementation platform for two specific applications. An Internet core router for 10 Gbit/s networks and a baseband part for a 3G WCDMA/FDD basestation are presented.

1.5 Dissertation overview

Part I is made up of the first two chapters. These deal with a general motivation to the work presented in the rest of the thesis and the basics of multi-processor systems.

Parts II and III go into details on on-chip infrastructure and networks

on chip. Chapters 3 and 4 discuss the concept of networks on chip and the basic challenges posed on such solutions. Chapter 5 introduces the solution, SoCBUS, that has been developed based on this background knowledge. Chapter 6 discusses the design environment and, more specifically, the simulation environment that has been developed for the SoCBUS network.

Part IV discusses benchmarking of interconnects. This includes a general introduction to benchmarking in chapter 7 and a newly developed methodology for interconnect benchmarking described in chapter 8.

Part V evaluates the SoCBUS network from a real-world application perspective. Two applications, an Internet core router (chapter 9) and a 3G basestation (chapter 10), have been investigated.

Finally, part VI concludes the thesis and gives an outlook on the future work that is possible with the previously presented work as a starting point.

Bibliography

- [1] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta, *Parallel computer architecture: A hardware/software approach*. Morgan Kaufmann Publishers, Inc., 1999. ISBN 1558603433.
- [2] Handel Jones, *Analysis of the relationship between EDA expenditures and competitive positioning of IC vendors for 2003*. IBS Inc. through EDA Consortium, <http://www.edac.org>, 2003.

Chapter 2

Heterogeneous Multi-Processor Systems

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke (Clarke's third law)

2.1 Introduction

People in the area of scientific computing have used multi-processor systems for many years. The advantage being the shorter run-time to solve a computationally intensive problem.

The problem with multi-processor systems is that in order to get a speedup of the application, significant parallelization is needed. Even very small amounts of nonparallelizable code will severely hamper the speedup as is shown by equation 2.1. This equation describes what is known as Amdahl's law. The program has a nonparallelizable part, s , and a parallelizable part, $p = 1 - s$. By increasing the number of processors, N , some speedup can be achieved.

$$S = \frac{s + p}{s + p/N} = \frac{N}{(1 - p)N + p} \quad (2.1)$$

The theoretical limit set by this law is shown in figure 2.1 for some values of p . It is clear from the figure that even when a small portion of the

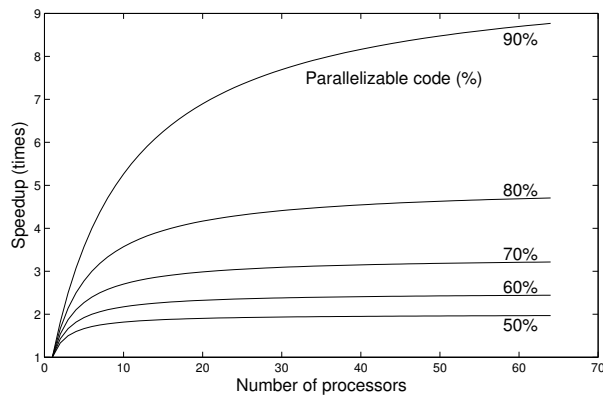


Figure 2.1: Speedup for parallel computers (Amdahl's law)

program can not be parallelized, this will rapidly reduce the additional speedup when adding processors.

It must be considered that Amdahl's law is only valid for applications that have a large amount of internal dependencies. Applications that are based on dataflows may well seem to break this law, although then the law has not been applied correctly. The dataflow applications may have large sequential parts when considering a single data block, whereas data blocks may be largely independent of each other. This will give the possibility to parallelize even such an application but instead over several data blocks. This thesis mainly focuses on dataflow based applications in the communications and media signal processing domains.

2.1.1 Parallelization of applications

There are three main methods of parallelizing an application. Which one is most appropriate depends on the type of application and the context in which it is run. For more general-purpose scientific applications that are run on a limited dataset, e.g. protein folding, the only method is to divide the problem into smaller chunks of the same type. These can then easily be handled and distributed on the processors. Doing this kind of division

is very difficult and can suffer heavily under Amdahl's law.

The second main method is appropriate for applications that are inherently sequential or dataflow based. Such applications are typically run over a large, possibly infinite, dataset. Dividing dataflow programs into smaller tasks is significantly easier than the previous case. Thus the application can easily be distributed onto a set of (different) processors. Applications in the areas of multimedia processing and radio baseband processing tend to be of the dataflow type. In this case, it is also possible to tailor the processors so that they are more appropriate for their part in the processing chain.

The final method can be used when several independent dataflows are handled, like a (general-purpose) network router. The dependencies in such a system comes to a large extent from the memory handling and partitioning.

By creating a dataflow platform with several heterogeneous and specialized DSP cores it is possible to reach the three ultimate goals for electronics design: high performance, low power consumption, and low cost.

2.2 Platform-based design

The concept of platform-based design in short means that a chip design should be reusable for a set of applications. This is preferred in order to amortize the non-recurrent engineering (NRE) costs for chip design and manufacturing over a larger number of chips. The idea is to create a generic enough platform for the set of applications by allowing software configuration of much functionality. The platforms are typically constructed around some programmable or configurable IP. This can be a mask-programmed ASIC (gate array), an FPGA, or a processor.

If such a platform is to be efficient for the new, more complex types of applications, a number of processors (or processing elements) are needed. Typically, these processing elements should be of different types, each type tailored to a specific aspect of the processing in the application area.

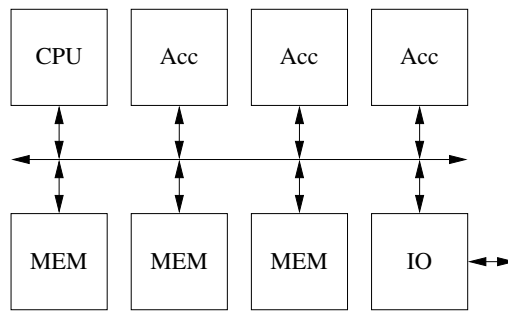


Figure 2.2: Classical bus-centric platform with a single CPU

2.3 Evolution of processor-based platforms

The classical processor-based platform is depicted in figure 2.2. The typical architecture consists of a single processor, either a general-purpose processor (GPP) or a digital signal processor (DSP). This is accompanied by a (relatively small) number of hardware units tailored to specific functionalities, known as accelerators. Further, there are a few memories for storing programs, coefficients, and data for the processing. Finally, some input/output circuitry is needed. These units are connected together using a bus or in rare cases a small-scale crossbar. This in total makes up a complete system.

The single drawback with such a solution is the scalability of the system. With an increase in the processing requirements, the platform has to follow. This is not trivial since there is no easy and scalable method to add further processing elements to the system.

The current trend is towards larger, more complex systems. The complete system will probably consist of many subsystems, each similar to the classical platform. These are in turn connected to each other using an interconnection network, see figure 2.3. A processing element should not be seen as a single processor, but rather as a small self-contained processor-based system.

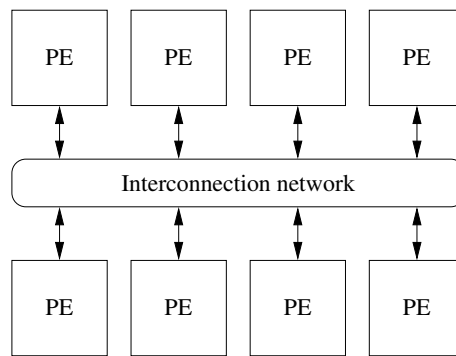


Figure 2.3: New platform architecture with multiple processing elements (PEs) each containing one (or more) processors

2.4 Communication-centric platform design

Considering the complex platforms outlined above it is clear that the interconnection network will become a hot issue when designing such systems. The basic performance of the processing elements are of no use unless the data can be fed to them at the appropriate rates. The design of a communication infrastructure for complex chips must be done in an orderly fashion to achieve good results at a reasonable design efficiency.

Rabaey identifies four key tenets for the design of such platforms [1]:

- Applying a discipline to on-chip communication design, giving a transition from ad-hoc SoCs to disciplined IC platforms.
- Base the design on formal models of computation (MOCs) and support a correct-by-construction synthesis design flow as well as a set of analysis tools for broad design exploration.
- Maximize reuse by defining a set of interfaces between layers.
- Provide the application programmer with a set of application programming interfaces (APIs) that hides the architectural details behind an abstraction layer.

From these bullet points one can identify several tools and libraries that have to be present for the efficient development of platforms and products built on these platforms. The tools include synthesis (e.g. network generator) as well as analysis (e.g. behavioral simulator).

Libraries have to be present for communication abstraction in the software. The advantages of using well-defined APIs have been appreciated for several years by the parallel-computer society through the message passing interface (MPI) [2].

Also, the use of well defined MOCs and APIs makes it possible to create the software components of the system concurrently with the hardware platform. By using architectural and behavioral simulators for the system components it is possible to run and debug software at a stage where no hardware is available. When the hardware becomes available it is (in theory) just to move the software components onto the hardware and the system should be up and running. Thus it is possible to shrink the time to market significantly.

2.4.1 Dependencies

There are three types of large scale dependencies in a complex system: synchronization, communication, and parallelization. The ultimate goal of communication centric design is to eliminate all three of these in order to fully utilize the power of the platform for the application. Dataflow applications based on data push between subsystem can easily eliminate the need for extra synchronization on the top level, assuming that the system is fast enough to fulfill the processing requirements.

By using appropriate hardware and strong tool support, it is possible to minimize the other two dependencies. Communication dependencies can be minimized through the use of a scalable and efficient interconnect structure with matching performance. When the communication is sorted out, parallelization problems are minimized through a multiplicity of heterogeneous subsystems with the appropriate task identification, subsystem mapping, and scheduling.

2.5 High-level application/platform design

The creation of a multiprocessor environment on a chip is a fairly straightforward task. Even though the complexity of the system is high and there are numerous problems at the physical level (e.g. DSM effects) it is still manageable to build quite complicated systems. Producing software that can exploit the platform efficiently is a completely different problem that is not necessarily simple.

Taking software development from the sequential uni-processor system onto a truly parallel tasking multiprocessor system that furthermore is heterogeneous can make software development complicated. The slow introduction of multiprocessor platforms in desktop PCs is a very good evidence for this fact.

2.5.1 Profiling of applications

The potential applications in the domain that is targeted for the platform have to be considered at an early stage in the platform design. The applications are typically examined for kernel operations and algorithms that are used for significant portions of time. These kernel parts are then considered when the hardware is designed. By using very efficient implementations of the kernel operations but still allowing any kind of program to run it is possible to achieve high performance within the application domain while not restricting the platform too much.

The profiling is done at a very early stage in system design and must be considered to be somewhat uncertain. Therefore it should not be relied on to the full extent but there should be headroom for unprofiled applications on the platform.

2.5.2 Selection of processing elements

The selection/design have to be performed with great care by experienced designers for a suitable end result. If the platform design is based on IP reuse to a large extent, which is desirable, the selection of which IP

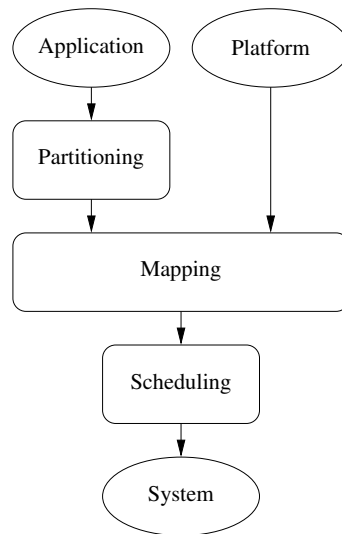


Figure 2.4: Integration flow for software on top of a hardware platform

blocks to use is a design decision with large impact on the performance of the platform. The design decisions when building new IP blocks have a similarly large impact on the result.

2.6 Application implementation

The last step in the platform-based design is the integration of an application on top of the platform. This work includes application partitioning and mapping onto processing elements.

A high-level implementation flow for the software on the platform is shown in figure 2.4. The application is first partitioned into small (atomic) units, commonly denoted “tasks”, that can easily be extracted from the programs. This could simply be subroutines or other (possibly explicit) entities in the program code. The tasks are then mapped onto the processing elements, considering the costs of communication between the tasks in the specific platform. The mapped tasks are then scheduled so that the

system meets the processing deadlines set by the designer.

2.6.1 Application mapping

How mapping of the application onto the hardware platform is done is important for the end result. The ideal application mapping should minimize memory usage, processing requirements, communication, and power consumption to a global optimum for the application. Mapping is a known NP-hard problem so there is little hope to find the optimum solution for other than the smallest problems.

The problem can be formulated as follows. Given a platform with processing elements E , a set of tasks T , and a communication graph C , find a mapping M that fulfills the constraints.

$$M = F_{map}(E, T, C) \quad (2.2)$$

The mapping function F_{map} must be somewhat tailored to the application area for near optimal results as stated above. The result of the mapping is basically a set of tuples $\langle E_i, T_j \rangle$ that describes which task(s) go on which processing element.

2.6.2 Scheduling of real-time applications

After the application has been mapped onto the hardware platform it is time to create a schedule. For a hard real time system it is important that the schedule meets the worst-case requirements for all situations. If there is a large ratio between the worst-case and the average runtime of the tasks, this may lead to a significantly overpowered system to be able to keep up with the worst-case timing. If the worst case is unlikely to occur, it may be worthwhile to consider a softer real-time approach. Thus it may be possible to lower the processing performance requirements with only a small cost in terms of system performance.

There are a vast number of approaches to scheduling, from the simple “as soon as possible” (ASAP) and “as late as possible” (ALAP) scheduling techniques to very elaborate ones. Achieving an optimal schedule is only

possible for reasonably small problems whereas large problem formulations generally use an ad-hoc approach to get nearly optimal results.

There are and have been a large number of different research projects around the world concerning scheduling. Some target specific application areas, e.g. matrix calculations [3], while others target specific goals, e.g. memory access minimization [4]. Traditionally, there have been much research on scheduling for single-processor systems [5] while the multi-processor systems have largely been neglected until recent years.

Discussions on high-level synthesis issues, such as mapping and scheduling, are far beyond the scope of this thesis. A good place to start for information in this area is the book by Eles, Kuchinski, and Peng [6].

Bibliography

- [1] Jari Nurmi, Hannu Tenhunen, Jouni Isoaho, and Axel Jantsch (Eds.), *Interconnect-centric design for advanced SoC and NoC: Communication-based design for network-on-chip (chapter 1)*. Kluwer Academic Publishers, 2004. ISBN 1-4020-7835-8.
- [2] "Message passing interface (2.0)," <http://www.mpi-forum.org/>, 1997.
- [3] G. N. S. Prasanna and B. R. Musicus, "Generalized multiprocessor scheduling and applications to matrix computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 650–664, June 1996.
- [4] Jenny Qingyan Wang, Edwin Hsing-Mean Sha, and Nelson Luiz Passos, "Minimization of memory access overhead for multidimensional DSP applications via multilevel partitioning and scheduling," *IEEE Transactions on Circuits And Systems II: Analog and Digital Signal Processing*, vol. 44, Sept. 1997.
- [5] Jiang Zhu, T. G. Lewis, W. Jackson, and R. L. Wilson, "Scheduling in hard real-time applications," *IEEE Software*, vol. 12, pp. 54–63, May 1995.

- [6] Petru Eles, Krzysztof Kuchinski, and Zebo Peng, *System synthesis with VHDL*. Kluwer Academic Publishers, 1998. ISBN 0792380827.

Part II

On-Chip Communication Infrastructures

Chapter 3

On-Chip Communication Infrastructures

Different computers need different cables.

HP DraftPro DXL/EXL User's guide

Merriam-Webster Online Dictionary defines infrastructure as “the underlying foundation or basic framework (as of a system or organization)”. Accordingly, the infrastructure is one of the most important aspects of a system.

In order to create a good infrastructure, it is very important to understand the context in which it should work. When a good understanding has been reached the potential for efficient, high-performance designs is very high. In the end, making a good design is one of the most important factors in product success.

In this chapter, problem areas related to on-chip communication infrastructure are identified and the possible solution spaces are explored.

3.1 Communication environment

It is important to see both the differences and likenesses between off- and on-chip communication. Several properties are very different between

the two domains, a topology that can e.g. change considerably during runtime for an off-chip network whereas the on-chip counterpart will be more or less the same. Other properties, such as synchronization, is more or less the same for the two domains.

3.1.1 Internal synchronization

Whereas a typical off-chip network (e.g. gigabit Ethernet) uses relatively high speed communication over long distances, the on-chip counterpart uses much shorter distances. Consider the gigabit Ethernet signal (@ 125 MHz) where each cycle corresponds to a couple of meters of wire. This will clearly give synchronization problems even for rather short wire lengths used in the network. On the chip where distances typically reach a maximum in the centimeter range, a few GHz is necessary to reach the same effects. This is by no means an impossible signaling rate on-chip [1].

Possible remedies for the synchronization problem may be the use of mesochronous or pleisochronous clocking schemes for the interconnect. Mesochronous means that both parties use the same clock frequency but with an unknown phase difference while pleisochronous means approximately the same clock frequency in both ends. A purely asynchronous approach will also work if that is desired.

3.1.2 Area and power considerations

Other very important aspect in on-chip communications are the area and power consumptions of the communications subsystem. Whereas the off-chip counterpart can occupy considerable space and consume considerable power, the on-chip version must be both small and power-efficient.

Even with an somewhat increasing gate count, the building blocks in the interconnect can become physically smaller as scaling continues. Another effect of process scaling is the significantly increased amount of transistors per chip, see figure 3.1. The increase comes almost entirely from the shrinking feature sizes while increased chip size has a smaller impact [2]. With scaling comes the possibility for more complex function-

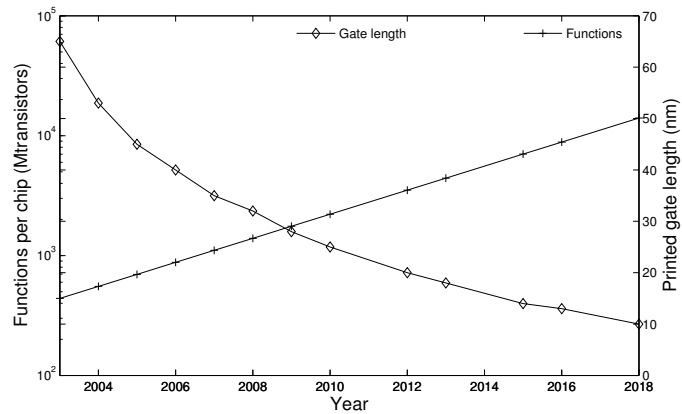


Figure 3.1: Transistors per chip vs. feature size

ality in the interconnect. This is not only a necessity but also a necessity since a number of problems will require the use of more advanced functionality in the interconnect as is shown in section 3.2.

3.1.3 Comparison with parallel computer networks

It is interesting to see the similarities between on-chip communication infrastructures for the real-time class applications and networks used in parallel computers. The demands for connectivity, performance, reliability, etc. are all in the same range as for the on-chip counterparts. Parallel computer networks also tend to be relatively fixed topology, but with a higher fault probability than on-chip. This all leads to the parallel computer network arena being a very good place for inspiration when it comes to advanced on-chip communication.

3.2 Challenges and opportunities

There are many challenges when designing the communication infrastructure. On the other hand, there are also many opportunities given in

Table 3.1: Some challenges in on-chip communication

Context	Challenge
Physical	Voltage regions
	Clock distribution
	GALS
	Area and power
Logical	Connectivity
	Bandwidth and latency
	Real-time behavior
	Dynamic vs. static behavior
	Robustness
Design efficiency	Productivity
	Verification
	Testing

the design process. Some of these challenges are shown in table 3.1. The remainder of this section will elaborate a bit on these challenges and outline the opportunities given.

3.2.1 Physical issues

There are several challenges for on-chip interconnects in the physical domain. Most of the current challenges derive from the urge to save on power consumption.

The dynamic power consumption of a CMOS system is governed by the well-known formula

$$P = \alpha \cdot C \cdot f \cdot V_{dd}^2 \quad (3.1)$$

As can be seen, the most important factor is the supply voltage (V_{dd}). The other factors are operating frequency (f), switched capacitance (C), and switching activity (α).

Operating frequency and switched capacitance are in principle hard

to reduce. The switched capacitance is determined primarily by physical layout considerations. The operating frequency is dictated by the requirements on bandwidth, latency, etc. It is in principle possible to reduce the frequency by adding more parallel wires for the interconnect. This method, while reducing the frequency, will increase the switched capacitance proportionally resulting in a similar power consumption. There is a possibility to save energy in this case, though this comes from the possibility to decrease the power supply for the lower frequency.

Switching activity can be reduced in several fashions. One of the simplest is using smart coding schemes for the interconnect [3]. This can be particularly useful in conjunction with error-correcting codes that will introduce redundancy, possibly giving more favorable autocorrelation properties in the dataflows.

Voltage regions

One increasingly important aspect in physical design is voltage regions. As power is rapidly becoming an important factor in design, adaptation of supply voltage to the computing power needed is increasing in popularity. This will lead to chips with many different voltage regions, with voltages that will vary during runtime. The communication subsystem must be able to allow the differences in supply voltages in different attached subsystems, possibly even running the interconnect at different voltages dependent on the communication demands.

Clock distribution and GALS

Clock distribution is another problem that is becoming increasingly difficult in complex systems. Keeping an entire chip synchronous is no longer possible for any useful frequencies. Instead the clocking can be divided into smaller regions that each internally run synchronously. Each such region will have its own clock running at the appropriate frequency for that module. The distribution of the clock can then be done using a low-frequency normal that is used in local phase-locked loops (PLL)

for clock generation on chip, thus circumventing the power-hungry high-frequency clock distribution. Clock distribution is not normally done through the communication infrastructure and will not be discussed further in this thesis.

With the increasing clock rates and multiple clock domains, the trend is towards globally asynchronous/locally synchronous (GALS) designs. These designs are based on asynchronous connections between the synchronous regions. The design of efficient asynchronous communication is error-prone with today's design tools and should preferably be done in as few places as possible. One advantageous possibility is to include the asynchronism in the communication infrastructure, isolating it to a single IP block instead of requiring asynchronous circuitry in many different IP blocks.

3.2.2 Logical issues

Connectivity

One important aspect of the interconnect is the connectivity, i.e. which processing elements can communicate with which other processing elements. As the trend towards application-domain-specific hardware platforms continues, there is an increasing need to supply appropriate connectivity to allow future adaptations to new, upcoming standards in the given area. It is obvious that more complex interconnect architectures can provide significantly higher usable connectivity than a bus or similar structure.

Bandwidth and latency

There are of course also challenges in the area of raw communication performance. As the applications and systems become more and more complex, the need for communication capacity will also increase and become more complex. The two primary targets when discussing capacity is bandwidth, i.e. how much data can be moved per time period, and latency, i.e. the end-to-end delay for data transfer.

An interesting sidenote on latency is that a higher latency will also imply more storage in the interconnect. This is obvious from the fact that more data is in the interconnect at the same time.

It is relatively simple to analyze and describe the bandwidth and latency for a simple infrastructural element, e.g. a bus. When going for more complex structures, the analysis becomes more and more difficult. There is currently no theory available that can cope with arbitrary networks.

Real-time behavior

Another important aspect in advanced designs, especially for media and communications, is the real-time behavior. Most of the applications have strict deadlines to meet if the system is to behave correctly. A typical example is a radio basestation that has to process one incoming radio frame before the arrival of the next frame.

The requirements on the communication infrastructure becomes significantly tougher when considering real-time behavior. A typical remedy to the real-time behavior problem is quality of service (QoS).

Dynamic and static behavior

Whereas the static behavior of a system can be fairly simple to analyze, the dynamic behavior can be largely unpredictable. Intentional events, such as schedule changes and unscheduled or random traffic, as well as unintentional events, e.g. transient errors, can trigger complex event chains in the interconnect system. The analysis of such events can be more or less impossible, requiring the use of simulation to assess the impacts.

The issues in dynamic behavior is largely coupled to the real-time behavior. In order to guarantee the real-time behavior, all dynamic issues and their impact will have to be taken into account.

Robustness and reliability

Another aspect that is closely related to real-time behavior and QoS is robustness. A network must be reliable, i.e. provide correct functionality in all normal cases, in order to guarantee anything in the area of QoS. During exceptional circumstances, e.g. physical failure in the chip, reliability (and thus QoS) can normally not be provided at all.

There is a physical motivation for adding reliability circuitry to the infrastructure. With shrinking feature sizes the electrical noise margins will continue to shrink giving higher and higher probability that something will go wrong, e.g. that a signal may be inadvertently changed [4]. This can to a large extent be counteracted by error-correcting codes on the interconnect. If QoS is to be guaranteed, error detection is not enough since there will be no headroom for retransmissions. If the data is not correct on the first try the guaranteed bounds can not be kept.

3.2.3 Design efficiency issues

Design productivity is one of the absolutely worst problems for the industry today when it comes to exploiting the process technology available, as described in section 1.1. Three areas related to design productivity are clear candidates for help from the communication infrastructure.

Design productivity

Ease of integration in design projects primarily based on reuse of previous designs or external IP blocks is paramount for the shortening of the design cycle. Thus the communication infrastructure should provide easy to use, easily verified, standardized connections that can allow high productivity in integration.

Furthermore, it is vital that the performance requirements for the system can be met. This implies that the communication infrastructure must be easily understood and evaluated from a performance perspective to qualify the system for the intended application or application area.

System design verification

As mentioned in the previous paragraphs, ease of verification is very important for the design productivity. There are figures in the literature showing that the verification effort clearly outweighs the design effort in complex system on chip designs today [5]. Since the communication infrastructure is the glue that holds the system together, this is a key area to alleviating the verification pain in complex designs.

By allowing a reasonably decoupled verification of the system, where subsystems can be verified separately and then easily verified at the integration phase could slash the verification time considerably.

There are three typical methods to verify a multi-processor design. The first is to create a “super-model” where all functionality of the system is included and then verify the RTL designs and software implementations against this model. Another method is to use formal verification of all subsystem interaction before the customer modification, thus minimizing the inter-work verification as much as possible. The final variation is to generate a super-model for the interconnect functionality and the interconnect interfaces, leaving each subsystem to be verified using whichever method being most suitable.

The first method is generally too costly and complex for use in the most complex system designs, although the verification quality can be very high when such a model is used.

The second method will give good verification quality before customer modification, but in principle everything will have to be verified again after modifications. This second verification run may be significantly easier than it would have been without the previous verification efforts for the system.

The third and final method seems to be the most appropriate in the context. The use of a super-model for the interconnect allows for easy verification of the interconnect and interfaces in the system. Also, by building the model with a well-designed interface, this model can be used as a basis for an overall super-model of the system.

Chip testing

Chip testing is now a major cost in the manufacture of complex chips. With the unproportional scaling of logic resources compared to the number of pins on a package this will only worsen over time. The current trend is towards more built-in selftest (BIST) circuitry on the chip to test the various parts. Even with BIST circuitry there is a need to control the testing and distribute the test information over the chip. There is an ongoing effort to create a system on chip test structure today [6]. This effort does not specify the infrastructure for connecting different test subsystems so that is left for the designer to do.

Besides the inherent problem of testing the communication infrastructure itself, one can consider the advantages of using the infrastructure for communication purposes when testing other parts of the design. The infrastructure that is used for communication can thus be reused at chip testing for controlling the testing procedures from the chip tester. One obvious advantage is that a completely separate infrastructure for testing (that in turn has to be verified and tested) can be avoided.

Another advantage with this is that test ports will be connected to the network which in turn is connected to the system controller. The system controller can thus use the interconnect to run diagnostics and selftests on idle parts of the system without extra communication interfaces being needed for this purpose. This may also lead to simpler testing equipment being needed for the testing since much of the tester functionality is moved onto the chip.

3.3 Traditional interconnects

The historically proven most appropriate solution for a small system is the time-division bus. This is appropriate where a small number of units, typically less than ten, are attached to the bus. It is also possible to create multi-processor systems with a bus, see figure 3.2. In this case, the cache memories are crucial in offloading the traffic on the bus to get any

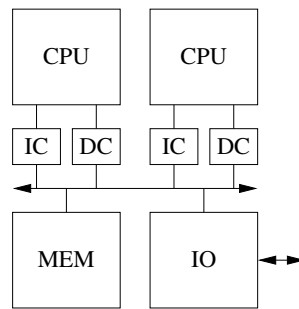


Figure 3.2: Bus based two-processor system (IC/DC = Instruction/data cache)

performance enhancement from the added processor(s).

This bus-based solution has very good properties for a limited system, but scales very poorly to larger systems. As the number of attached units increase, the bandwidth available to each one will decrease. Also, the increased number of units will result in higher capacitive bus loading (due to fan-out) or larger logic depth in the bus, further decreasing the performance.

Point-to-point (PTP) links are very nice from a bandwidth scaling perspective since each physical communication channel is completely separated from the other channels. The limiting factor for scalability here is complexity. For a fully connected network of PTP links with n units, a total of $n \cdot (n - 1)/2$ links will be needed. This number grows rapidly when the number of units is increased.

3.4 Generalized communication structures

Because of the bad scaling for both traditional buses and PTP links, other communication architectures must be used for larger systems. The first step in scaling is the use of a bridge between two buses. This idea can be generalized so that a bridging network of some kind is used between

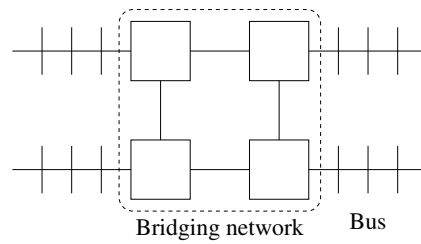


Figure 3.3: Hierarchical bus/network architecture

the buses. This bridging network could in principle be anything from a simple bus-to-bus bridge to the Internet. A network of some kind is the most suitable for a system where scalability is the primary issue. It is important to remember that the complexity has to be managed in a proper fashion, especially for on-chip communication as will be discussed in the next chapter.

For a more in-depth discussion on general interconnection networks than is given in this thesis, please refer to the book on interconnection networks by Dally and Towles [7].

Bibliography

- [1] Peter Caputa and Christer Svensson, "Well-behaved global on-chip interconnect," in *IEEE Transactions on Circuits and Systems I*, 2005.
- [2] *International Technology Roadmap for Semiconductors, 2003 Edition, Executive Summary*. <http://public.itrs.net/>, 2003.
- [3] Sumant Ramprasad, Naresh R. Shanbhag, and Ibrahim N. Hajj, "Signal coding for low power: Fundamental limits and practical realizations," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, July 1999.
- [4] G.R. Srinivasan, P.C. Murley, and H.K. Tang, "Accurate, predictive modeling of soft error rate due to cosmic rays and chip alpha radia-

- tion," in *Proceedings of the IEEE International Reliability Physics Symposium*, pp. 12–16, Apr. 2003.
- [5] Michael Keating and Pierre Bricaud, *Reuse methodology manual*, 3rd ed. Kluwer Academic Publishers, 2002. ISBN 1-4020-7141-8.
- [6] IEEE P1500 Working Group, *Standard for Embedded Core Test (SECT)*. <http://grouper.ieee.org/groups/1500/>, 2005.
- [7] William J. Dally and Brian Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann Publishers, Inc., 1998. ISBN 0-12-200751-4.

Chapter 4

Networks on Chip

Think like a wise man but communicate in the language of the people.

William Butler Yeats (1865 - 1939)

This chapter continues the discussion on communication infrastructure with a network on chip focus. Several general aspects related to networks on chip are discussed in more detail.

4.1 The impact of context

There are a number of differences between implementations of buses, off-chip (general) networks, and on-chip networks. These are important to have in mind when creating a network on chip in order to get a reasonable and efficient result.

The time division multiplex (TDM) bus is the traditional, single-media-based interconnect that may connect any two attached units (possibly with some restrictions) while blocking all other combinations at the same time. Thus there is no possibility to have more than one transfer running at the same time on a single TDM bus.

A general off-chip network, on the other hand, will allow multiple simultaneous transfers (in most cases). It may allow units to connect and disengage from the network without prior notification and that the topology

may change at any moment due to this. Also there are many sources of unpredictability in such a network. A new unit that is attached may create “bad” traffic which was not foreseen at the deployment of the network and may thus damage the performance of the network. This is possible since the entire system (i.e. network and units) was not verified together in the first place.

An on-chip network will reside in a controlled environment. After manufacturing the chip, all units must already be there and no new units can be attached. The topology is also fixed at design time. Because of this hard deadline on network parameters, many aspects of the network design are simplified.

4.2 The OSI model

The seven layer open systems interconnect (OSI) model [1] was developed with general purpose networks in mind. Despite its usefulness in understanding the workings of networks, it is important to realize that the OSI model is not necessarily applicable directly to all types of networks. This is especially true when it comes to implementation where an OSI-layered implementation could become cumbersome and complex without adding any significant value. One such network type where the layered approach of implementation is unsuitable is on-chip networks. The reason is that there is some fundamental differences between a general-purpose network (e.g. Internet) and an on-chip network. Whereas the general-purpose network is in principle unknown, i.e. nodes can be added and removed at any time, the topology is not fixed, and so on, the on-chip network is well-known since basically everything is fixed at the time of chip manufacturing.

With the additional knowledge of the network infrastructure in a controlled environment it is possible to group several layers of the OSI model, without losing any flexibility and abstraction, and thereby simplifying the hardware and software. The simplifications will at the same time cut latency in the network. A typical coverage of the OSI-layers in a network

Table 4.1: The seven OSI layers and their coverage in the typical NoC

#	Layer	Coverage
7	Application	Not covered
6	Presentation	Wrappers
5	Session	Wrappers
4	Transport	Components
3	Network	Components
2	Link	Components
1	Physical	Components

on chip is shown in table 4.1.

The OSI layers are naturally grouped in a network on chip because of the limited freedom. Typically, the link and physical layers are very much related as well as the network and transport layers. These four layers have to be covered by the network in order to allow easy separation of the hardware from the software. The session and presentation layers can be left to the software or handled by the wrappers depending on the complexity of the protocols on these layers.

There is no natural border (or even a reason for one) between the physical and link layers in a network on chip since there is basically just one physical media possible, i.e. is the silicon chip. If just physical signaling is considered at this level, e.g. low-swing techniques, then the physical layer is even below the level of the network on chip. Similar arguments are valid for other layer pairs. Thus there is no reason for these layers being separated and no need for the full OSI layering to be implemented. This will save on component complexity and thereby give a cheaper overall solution.

In short, the OSI model is very useful for understanding network protocols and network behavior but is not necessarily the ultimate guideline for protocol design. Creating a protocol stack that is adapted to the system requirements and limitations is far more important than OSI compliance.

4.3 Quality of service

General computer networks, like the Internet, offer only best-effort packet forwarding. When congestion starts to occur, the service providers and core network providers will simply increase the available bandwidth by buying new, more powerful equipment until the congestion situation disappears. This method of dealing with congestion, though efficient in its simplicity, is not always possible or even desirable from a technological perspective. One such situation is networks on chip where excessive bandwidth requirements will give too costly implementations, both in terms of area and power.

Instead, there are many possible levels where a real-time behavior can be guaranteed. The most straight forward is to give guarantees on the network bandwidth and latency per communication. Such guarantees are known as quality of service (QoS).

Support for QoS implies certain qualities in the network. All communications must be correct and complete, i.e. sent data must arrive unaltered at the destination. There must also be guaranteed bounds on minimum bandwidth and maximum latency. Thus it is necessary to identify and separate the different communication flows in the network. Separation can be done using time slots as in *Æthereal* (Philips Research [2]) or any other virtual circuit technique.

Quality of service is only necessary when the underlying network is unknown or unreliable and based on resources that are limited from the application perspective. It is useful in OSI-based systems where any level can be exchanged for another implementation, resulting in a network with different performance.

If there are adequate resources in the network, all requirements can be met in the design. This will allow for a network without the support of QoS and will thus simplify the components. Also, an embedded system is generally based on worst-case assumptions, which will take all situations into account, further reducing the need for QoS. Currently, the time to market is the most important aspect of system design. This will require

Table 4.2: Theoretical performance of different network topologies assuming N connected cores.

	Ring	2D Mesh	2D torus	Binary tree	Fat tree
No. of nodes	N	\sqrt{N}	\sqrt{N}	$2N - 1$	$2^{N/4-1}$
Bisection BW	2	\sqrt{N}	$2\sqrt{N}$	1	N
Links	Bidir	Bidir	Bidir	Bidir	Bidir
Complexity					
Wiring	Low	Low	Low	Low	High
Wire length	Short	Short	Medium	Medium	Long
Routing	Low	Medium	Medium	Low	High

the use of simple components and tools to achieve a complex system.

4.4 Network topology

The topology of a network is the geometrical configuration on logical level used to connect the different network components. Many different topologies exist, from the simplest crossbar to very complex hierarchical cubes and beyond.

The first aspect to take into account when selecting which topology to use for a network is the patterns of traffic that will go through the network. So, in order to determine the most appropriate topology for a system an investigation of the advantages and drawbacks of a number of common topologies with respect to the application at hand must be done during the early design stages.

For a more in-depth discussion on topologies than that given in this thesis, please refer to the book by Dally and Towles [3].

4.4.1 Theoretical performance of topologies

A short summary of the theoretical performance for some of the common network topologies can be found in table 4.2. The ring topology is very

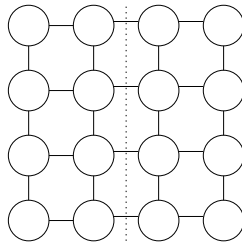


Figure 4.1: 2D mesh topology (4x4). Dashed line marks bisection.

easy to implement but is subject to the same fundamental limitations as a time-division bus since the only available resource for transmission at a node will be occupied whenever a transmission wants to pass that node. On the other hand, the more powerful topologies like fat trees are very complex when it comes to wiring.

4.4.2 Properties of the 2D mesh and torus

The two-dimensional mesh, see figure 4.1, and torus, see figure 4.2, are very suitable for on-chip networks. The main advantages of these are the good performance-to-resource ratio, the ease of routing, and that the topologies are very easily mapped onto a chip. The torus have somewhat better properties for random traffic but the added resources may be unnecessary for many systems. The better performance of the torus is due to the uniform connectivity of that topology. The uniform connectivity also removes any boundary effects from the torus, which will appear totally homogeneous. The average logical distance between routers will become shorter in the torus, but at the cost of possibly longer physical distances.

4.4.3 Advantages with arbitrary topologies

The clear advantage with allowing implementations with arbitrary topology is the possibility to adapt the network to the application domain at hand. This can give significant savings for parts of the network where

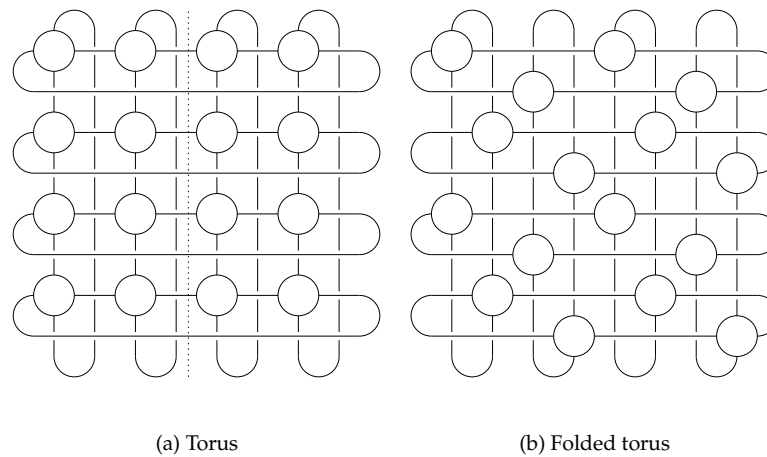


Figure 4.2: 2D torus variations (4x4). Dashed line marks bisection.

little traffic is handled while still being able to support significantly more traffic in other parts.

A small example of a nonstandardized network topology is shown in figure 4.3. Using two links between routers will effectively double the bandwidth there (pair 4-5), while removing links between adjacent routers will reduce network cost (pair 5-8). Also it is possible to skip routers to get shorter paths, leading to less resource usage and lower latency (pair 1-7).

4.5 Packet and circuit switching

A network using circuit switching has low complexity switching nodes because their main function is basically to connect an incoming link to an outgoing link. Deadlock avoidance is easily achieved since the circuit setup can either succeed or fail but may easily be prevented from stalling somewhere in the process. Packet switching inarguably leads to more complex switching nodes. Because the risk of deadlocks, one of three

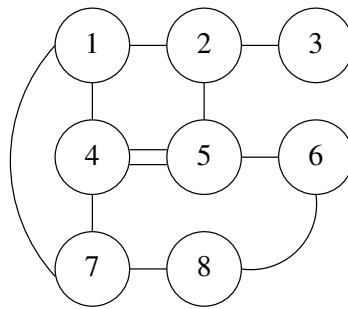


Figure 4.3: Example of an arbitrary topology

possible solutions has to be applied. Buffering of every packet in its entirety before routing it to the next node will prevent deadlock. This is very inefficient from memory usage and latency perspective in a network on chip. Another method is to use virtual channels for solving the loops in the dependency graph [4]. Finally, restrictions of possible paths can also be used to prevent loops in the dependency graph [5]. These alternate methods will be discussed further in section 4.7.

4.5.1 Latency

Packet switching may also suffer from latency problems where the packet delay through the network can be several hundred or even several thousands of cycles dependent on routing algorithms and router implementations [6]. One example is where a fully buffering implementation of a packet switched network transfers 128 bytes of information between two diagonally opposite corners of a 8×8 mesh, i.e. 15 hops. The time spent in the network would then be more than $128 \cdot 15 = 1920$ cycles. Even for a wormhole routing network there is a possibility that the packets will be stalled for long times due to other traffic. There is typically always a statistical distribution of packet delays in a packet-switched network. This will also create the possibility for packets to arrive out-of-order to the destination if distributed routing is used.

Circuit switching has a clear advantage over packet switching since the data latency is only dependent on the distance when the circuit has been setup. Also there is no dependency on other factors, e.g. other traffic in the network at that stage. All data is also guaranteed to arrive in the same order as it were sent. The only dependency on the traffic situation in a circuit switched network is when setting up a route.

4.5.2 Wormhole routing

The basic mode for packet transport is store-and-forward where a packet will be received at a router in its entirety before forwarding is done to the output. This is the typical mode of operation for general-purpose computer networks. The drawback is that store-and-forward is fairly inefficient for smaller, dedicated networks. Latency as well as the requirements on buffer memory size will be unnecessarily high.

For this discussion, one definition is necessary: A flow-control digit or *flit* is the basic transmission unit of the network. This means that a flit is the smallest piece of information that can be transported over the network.

By allowing the leading flit to continue through the network without awaiting the arrival of the following flits, it is possible to reduce the end-to-end latency of the network significantly. This method of “worming” through the network is generally very good for high-performance networks. The general drawback with wormhole routing is the increased resource occupancy that can increase the deadlock problems in the network, as is discussed in section 4.7.

4.5.3 Virtual circuits in packet networks

Both circuit- and packet-switched networks are inherently unreliable from a performance perspective. The interaction between different traffic flows is difficult to analyze and may give very unpredictable effects. One method to achieve quality of service in such an environment is the use of virtual

circuits (not to be confused with virtual channels), which in principle uses dedicated time slots on the packet-based network for specific traffic flows.

A typical use of virtual circuits is found in the *Æ*thereal network from Philips Research [2]. Distributed slot tables are used to enforce the time slots in the routers or network interfaces, thus providing QoS guarantees.

The drawback with virtual circuits is that the time slot assignments generally can not be done in a distributed fashion. Rather, in order to guarantee that a slot assignment solution is found if one exists, scheduling of transfers must be taken care of by a central resource with the full picture. This central resource (typically a processor) must then distribute the new schedule to the network and make sure that the changes take effect simultaneously throughout the network to avoid any glitches in the behavior. It may even be necessary to have a gradual transit from one schedule to the next to avoid problems with traffic currently in the network.

4.6 Routing in networks on chip

There are many different methods for choosing a route in a network. This section will touch upon some different methods and their main properties.

4.6.1 Nonminimal routing

Theory indicates that a nonminimal routing will severely worsen the congestion in a saturation situation. This has been validated by some simple experiments. Thus, for a graceful degradation at the saturation point, a simple minimum-path-length routing is better than the nonminimal counterpart.

4.6.2 Source and distributed routing

There are two distinct routing classes, based on where the routing decision takes place. Source routing relies on the packet source to supply the

routing information as the name implies. Thus the source decides the entire path for the packet. The necessary routing information can either be calculated in the source processor or in the network interface (wrapper).

In distributed routing, on the other hand, the source will only supply a destination identifier and the routers decide which path is the most appropriate.

There are advantages as well as drawbacks with both methods. Source routing will allow the use of very simple routers since they will, in principle, be nothing more than advanced multiplexers. The network topology knowledge must be known to the source and this may not be desirable. Source routing does not allow the network to use alternate routes in case of congestion, which is possible with distributed routing.

Source routing works well in networks where the traffic is well known, whereas the distributed routing tend to work better in an environment of random or unknown traffic. This is simply because the source routing can handle a well-behaved scheduled system whereas the distributed routing can handle the unforeseen situations that happens with random traffic.

4.6.3 Local vs. global knowledge

Both source and distributed routing can use information about the network state and current traffic flows in order to make the optimal routing decision. Globally optimal routing is only possible if the decisions are taken considering the full network state. This is not scalable since either one single block will have to have all information and take all routing decisions or all state information must be distributed (instantaneously) to all routers.

Instead, using local knowledge of just a single router or possibly the neighboring routers will yield suboptimal results but with high scalability. By using only local knowledge a packet may be routed around a local hotspot and into a more congested area in another area of the network that could have been avoided with global knowledge.

4.7 Deadlock avoidance

Deadlock becomes an issue as soon as the system may enter a state where resources may be locked indefinitely while waiting for other resources to become free. If a loop is formed in these dependencies, a deadlock will occur. This situation is especially prone to occur in wormhole-routed networks because of the extended resource occupation of the wormhole packets.

There are in principle only three methods to avoid deadlocks:

- Do not wait at all for resources.
- Use timeouts when waiting for resources.
- Prevent loops in the dependency graph from occurring.

The first and second methods would lead to dropped packets in a packet switched network when congestion occurs. If this is not acceptable, the third method must be employed. The common methods to do this is to prevent some turns in the route or by adding independent resources that will break the loop.

It is not trivial to prove freedom of deadlocks in a network with arbitrary topology. Some general theories for deadlock avoidance in certain types of networks have been presented during the last decade [7, 8]. The common ground for these is that the proof is based on complicated mathematics and that each work is only valid for that particular network type.

4.7.1 Turn-model routing

If the network has a regular topology, e.g. meshes, it is simple to prevent loops in the dependency graph by disallowing some turns in the network [5]. There are also extensions of these ideas allowing for fault-tolerance in the network [9].

Figure 4.4 shows two versions of the turn-model routing for 2D meshes. Figure 4.4(a) shows dimension-order routing. At most one turn is allowed in every route. Thus the route always follows the row where the source is

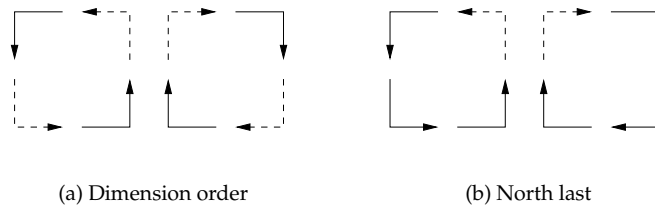


Figure 4.4: Turn model routing

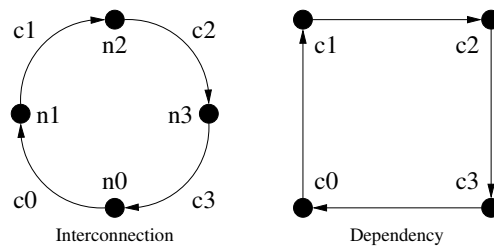
located until it reaches the column where the destination is located. This will prevent any dependency loops from forming in the network. The drawback with dimension-order routing is the very limited routing possibilities. Figure 4.4(b) shows a more flexible version where several paths may exist between a source and destination.

4.7.2 Virtual channels

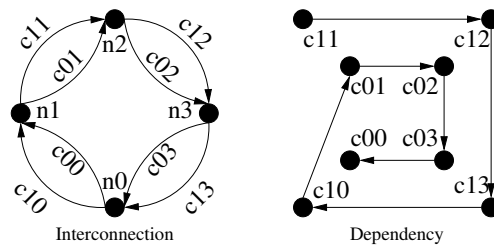
The previous section outlined a method to avoid deadlock by limiting the possible routes. This is undesirable from the performance perspective so other methods are used when possible. The most common is to use virtual channels where the output queues for the router is separated so that the forming of a dependency loop is prevented [10].

Consider the example given by Dally [10] for clarification. The situation in the example is four nodes interconnected by unidirectional links in a ring. The topology and dependency graphs will then be according to figure 4.5(a), with four channels numbered c_0 to c_3 . It is clear that a circular dependency exists and that the network is vulnerable to deadlock. By introducing virtual channels, i.e. logic links that may time-share the physical links, the possibility to break the dependency graph without cutting any link or preventing any routes is there.

Assuming two virtual channels on each physical channel gives the situation in figure 4.5(b). The new numbering scheme effectively makes a partial ordering of the channels if routes at a node with higher number



(a) Without virtual channels



(b) With virtual channels

Figure 4.5: Deadlock avoidance by virtual channels

than the destination node is routed along the low numbered channels, and vice versa. Channel c_{00} is not used. This will give a total ordering of the channels according to the subscripts: $C_{13} > C_{12} > C_{11} > C_{10} > C_{03} > C_{02} > C_{01}$. After this, the cycle in the dependency graph is gone and it is possible to route completely deadlock free.

The concept of virtual channels can be extended to networks with arbitrary topologies, although more virtual channels than two may be needed per link [11].

4.7.3 Circuit-switched networks

Circuit-switched networks are in general simpler than packet switched networks. A circuit, once it is connected, will lead to completion of the

transfer and will not contribute to deadlock problems. The dangerous task is the route setup where deadlock due to occupied resources can occur if not designed properly. The simple solution is to not wait for resources to become free, but rather fail the routing attempt and retry. The other possibility is to use some of the many methods developed for packet-based networks, e.g. virtual circuits, for the circuit setup.

Bibliography

- [1] International Standardization Organisation (ISO), *ISO/IEC 7498-1:1994: Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. <http://www.iso.org>, 1994.
- [2] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage, “Networks on silicon: Combining best-effort and guaranteed services,” in *Proceedings of the design automation and test conference*, Mar. 2002.
- [3] William J. Dally and Brian Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann Publishers, Inc., 1998. ISBN 0-12-200751-4.
- [4] Hussein G. Badr and Sunil Podar, “An optimal shortest-path routing policy for network computers with regular mesh-connected topologies,” *IEEE transactions on computers*, vol. 38, no. 10, 1989.
- [5] Christopher J. Glass and Lionel M. Ni, “Adaptive routing in mesh-connected networks,” in *Proceedings of the International Conference on Distributed Computing Systems*, 1992.
- [6] Pierre Guerrier and Alain Greiner, “A generic architecture for on-chip packet-switched interconnections,” in *Proceedings of the design and test in Europe (DATE) conference*, 2000.
- [7] Eric Fleury and Pierre Fraigniaud, “A general theory for deadlock avoidance in wormhole-routed networks,” *IEEE Transactions on parallel and distributed systems*, vol. 9, no. 7, 1998.

-
- [8] Loren Schwiebert and D. N. Jayasimha, "A universal proof technique for deadlock-free routing in interconnection networks," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, 1995.
 - [9] Christopher J. Glass and Lionel M. Ni, "Fault-tolerant wormhole routing in meshes without virtual channels," *IEEE transactions on parallel and distributed systems*, vol. 7, no. 6, 1996.
 - [10] William J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. 36, pp. 547–553, May 1987.
 - [11] Frederico Silla and Jose Duato, "On the use of virtual channels in networks of workstations with irregular topology," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, pp. 813–828, Aug. 2000.

Part III

SoCBUS

Chapter 5

The SoCBUS Network

The cure for boredom is curiosity. There is no cure for curiosity.

Dorothy Parker (1893 - 1967)

5.1 Introduction

The design space for networks on chip is vast. On every level of design there are numerous design decisions that have to be made. After the prestudy phase it was decided to go with circuit switching as the basic mode because of the resulting simplicity of the network components. A packet-based circuit setup scheme, described in section 5.5.1, was devised that allows shared use of the links for both circuit setup and payload transfer.

The network is made up of two separated component classes. The first is the routers and the second is the wrappers (a.k.a. network interfaces). These components are interconnected through bidirectional links (which in fact are two unidirectional links). This chapter will describe the design decisions, the components, and the implementations in detail.

5.2 Network components

If a 2-d mesh topology is used, the basic network can be seen as a matrix of tiles. Each tile consists of a router, a wrapper, and an IP block/subsystem

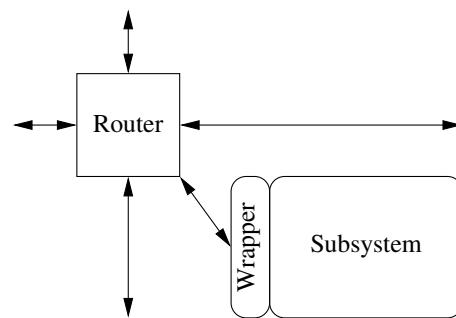


Figure 5.1: Network-connected processing tile

as shown in fig 5.1.

The purpose of the wrappers is to isolate the port of the subsystem from the internal link format of the SoCBUS network. One great advantage of using configurable wrappers on the border between subsystem and communication infrastructure is the possibility to ease the task of verification of the whole system on chip design. The configurable wrappers can be thoroughly verified for the different possible configurations together with the rest of the network system so that only the interfaces have to be verified at the system integration phase, thus simplifying the verification task.

The routers are responsible for setting up the circuits between source and destination as well as transporting the data streams between these. If distributed routing is used, setting up circuits involves decoding of the destination address and translating this into a set of output ports that will lead closer to the destination. One port is then chosen and the request is forwarded. Similar tasks have to be performed in case of source routing. At data transport the router will just blindly forward the data to the destination port.

The routers are designed so that one occupied port will not limit the ability to use the other ports. This is implemented as a central crossbar that handles the connections through the router. Since this is a full-scale crossbar, it will scale with the square of the number of ports.

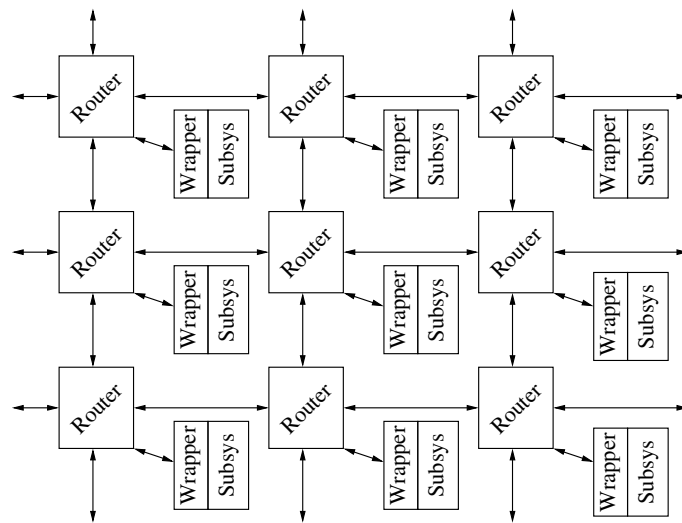


Figure 5.2: A 3x3-node switched network with wrappers and subsystems

5.3 SoCBUS architecture

The SoCBUS network was developed with the possibility to use arbitrary topologies in mind. Thus there are no inherent limits on the number of ports on a router, which route lookup algorithms that can be used, etc.

The investigation of different network topologies that were conducted during the prestudy indicated that a two-dimensional mesh should be the most suitable for most on-chip networks. This is also the common topology proposed by most other researchers [1, 2, 3]. The main reasons for selecting the two-dimensional mesh instead of other topologies such as hypercubes, butterflies, or trees are that a two-dimensional mesh have an acceptable wire cost, reasonably high bandwidth, and a nice mapping onto a chip.

Further, since it is easy to group subsystems that communicate much it is possible to map the subsystems onto the network so that they do not consume an unnecessarily high amount of resources in the network. Thus it is not necessary to have the “best” topology in order to get high

performance in the system.

The routers in a 2-d mesh network have five ports. One port is used for connection to the local subsystem while the other four ports are used to connect to the adjacent routers, see figure 5.2. The local port is connected through a wrapper to the local subsystem.

The interfaces internal to SoCBUS all use the same physical format, see figure 5.3. 20 wires in total are used in each direction. 16 wires carry data and request packets in the forward direction. Two wires are used for reverse control and two wires for forward control. These signals are described in more detail in section 5.7.

The simplicity of the SoCBUS components allow for implementations of routers and wrappers with very low logic depth, around 6-8 gate delays. With this low logic depth it is possible to reach an operating frequency of 1.2 GHz in a 0.18 μm process [4]. This is four times the maximum expected IP block clock frequency of about 300 MHz in the same process. This difference in clock rates will further mask the network latency since a four cycle latency in the network will look like a one-cycle latency to a core.

Considering the possibly high clock rates and the distributed nature of an on-chip network, the wire delays between components become a serious problem if using traditional synchronous design methodology. By using mesochronous clocking (i.e. same frequency but unknown phase) with distributed signal retiming it is possible to handle the wire delay and skew. Using mesochronous clocking still requires the wire delays within a link to have reasonably low skew but allows the design to use links that have very differing delays without any problem. To further simplify the connection of network components it is proposed to use optimized drivers and transmission-style wires [5, 6]. This will give many advantages, e.g. no repeaters are needed, the system consumes a minimum of power, and there is no need of laying out the network in an orderly fashion on chip.

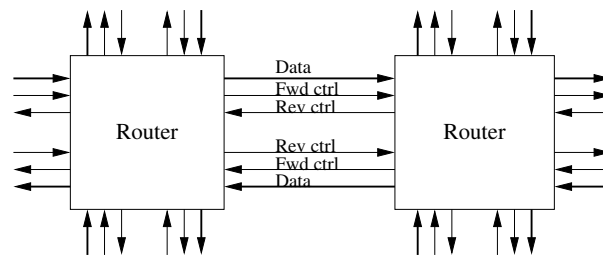


Figure 5.3: Interface between routers and wrappers within the network

Table 5.1: Configuration and control layering

Layer	Content
Data transport	Data package interpretation Application layer protocol
Link setup	Logical configuration Addressing and routing Congestion control Other (FEC, buffering, etc.)
Hardware design	Link layer protocol Physical protocol Network topology Wrapper configuration

5.4 Configuration and control layering

In order to make a usable system, the network components and the connections have to be configured according to the requirements during design time. This configuration makes up the hardware design which will be used in the platform. Further configuration and control tasks have to be carried out during runtime. All configuration/control tasks can be divided into different groups that reflect the level at which they are performed as shown in table 5.1. The layering can also be seen as a proto-

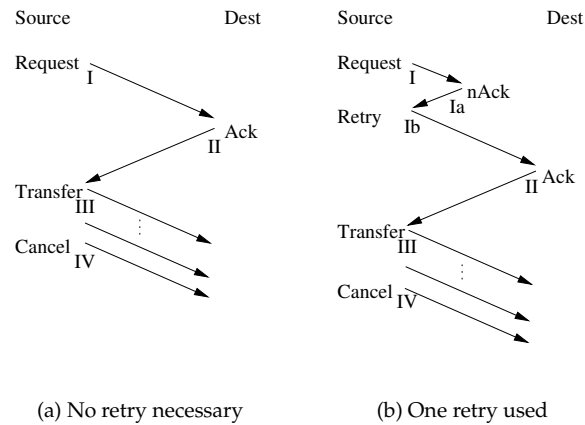


Figure 5.4: Two successful circuit setups

col hierarchy in the implemented system where lower levels are implemented as fixed hardware protocols and upper levels are implemented as software controlled protocols.

The data-transport layer includes the data link protocol for control of source-drain transfers and handles transmission specific control information such as packet sizes. The link-setup layer is at the wrapper-router or router-router level and handles link-by-link setup, e.g. FEC on the links. The last layer is the configuration done at hardware design.

5.5 Network transaction handling

5.5.1 Route setup flow

The network transactions consist of four to six phases dependent on whether the first try routing is successful or not. A successful transaction, see figure 5.4(a), has four phases. (I) First a request is sent from the source to the network. As this request finds its way through the network the route is locked and cannot be used by any other transactions. (II) The second

phase starts when the request reaches its destination and an acknowledgment is sent back along the route. (III) When the acknowledgment has returned to the source the third phase starts. This phase holds the actual transfer of payload data. (IV) Finally after the data has been transferred a cancel token is sent that releases all resources as it follows the route. This type of packet transport where the data is sent separated from the request is called "long packets".

If a route is blocked in a node the routing request is canceled by the blocking router (Ia) returning a negative acknowledge to the source, as shown in figure 5.4(b). (Ib) The source must then retry the route at a later stage which means that the additional two phases (nAck and retry) may need to be iterated.

5.5.2 PCC: Packet-connected circuit

The hybrid circuit switching with packet-based setup introduced in the previous subsection is referred to as "packet-connected circuit" or PCC for short. The PCC has very nice properties in several areas:

- PCC is deadlock free since no resources are locked while waiting (indefinitely) for other resources.
- The routers become very simple when no special cases, stalls, or virtual channels must be considered.
- A minimum of buffers capable of holding just a request package are needed in the routers.
- PCC gives the lowest possible latency of just one retiming flip-flop pipeline at each router.
- There is no inherent limit on route selection algorithms in the PCC scheme.

5.6 Routing

Routing in SoCBUS is the task of mapping a destination address or port number into a set of possible ports that can be used to get closer to the destination. Both basic methods for routing are implemented, namely distributed and source routing.

5.6.1 Distributed routing

In the case of distributed routing, the mapping can be seen a simple mathematical function where the address a is mapped onto the set of output ports P .

$$P = F(a) \quad (5.1)$$

The simplest and fastest method to implement this function is through a look-up table in the routers.

After studying publications on routing algorithms and performing some simple experiments it was decided to go for a simple minimum-path-length routing. Each router has the knowledge of the general direction to each destination, i.e. north, west, south, east, local, and combinations of these, e.g. north-west. Since the network do not change when the chip has been designed this knowledge is static if no faults are assumed to occur. Thus it can be decided at the time of network layout design. The routing decisions will then simply be based on the destination address which can be translated into the known direction(s). If there are more than one direction that leads to the destination one is selected according to a round-robin scheme. If the primary selection is occupied the second choice will be used instead and so on. If there are no free outputs that lead towards the destination the routing will fail and the router will return a negative acknowledgment to the source.

5.6.2 Source routing

Source routing is trivial compared to the distributed routing. In this case, routing information is transmitted along with the request and can directly

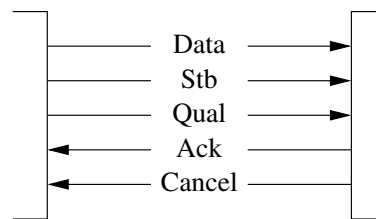


Figure 5.5: Signals in a unidirectional link

be interpreted as the output port number. If the intended output is occupied there will be no possibility to select a different port since that will violate the basic principles of source routing.

5.7 Physical links

The physical links in SoCBUS consists of five signals as shown in figure 5.5. The first signal is the data which is nominally 16 bits wide, although wider links are possible. The next two signals are the forward control consisting of strobe (*stb*) and data qualifier (*qual*). The last signals are the reverse control which consists of two acknowledgment signals, *ack* and *cancel*.

The main reason for defining the links like this is suitability and simplicity. The wires used are necessary for circuit setup and data transport. It would be possible to use a predefined link type, e.g. Wishbone or AMBA. Doing so would imply the link protocol which would not be suitable for a network on chip.

A routing request is signaled by a rising edge on the strobe signal. The first two cycles are used for the two request-packet words. A description of the data bits in the routing request is shown in table 5.2. The possible value combinations that decide which packet type to expect is listed in table 5.3. Out of the 16 combinations, only six are used. These combinations are sufficient to signal all possible packet types. The reason for not using only three bits for this is efficiency. The wires are already there so there

Table 5.2: Request format

Signal	Description
Req0 [15:8]	Destination address
Req0 [7:4]	<i>Reserved</i>
Req0 [3]	Speculative sending
Req0 [2]	End-to-end or local handshaking
Req0 [1]	Long or short packet
Req0 [0]	Distributed or source routing
Req1 [15:0]	Misc data or addressing

is no need for saving a bit and each functionality will get its own control bit, giving a smaller implementation in the router.

As can be seen from that table there are several possible packet types. The basic packet type is the long packet, as outlined in section 5.5.1. using distributed routing. The other possible packet types are long packets using source routing and short packets. Short packets are similar to long packets but the data and cancel phase have been replaced by a positive acknowledgment that rips up the route. Thus, a short packet will only be two words in size. Because the short packet can hold such a limited amount of information they have to use distributed routing.

Since there are two distinct positive acknowledgments and one negative ditto, there have to be two wires in the reverse direction. The functions encoded on these reverse control signals are shown in table 5.4.

5.8 Link level protocol

5.8.1 Long packets

The link-level protocol in SoCBUS is relatively simple. A typical time diagram of a long packet transfer is shown in figure 5.6. The start of a transmission is signified by the transition from low to high level on the strobe signal. The two first words that are accepted on the input are the rout-

Table 5.3: Functions for the req0[3:0] signals

[3]	[2]	[1]	[0]	Description
0	0	0	0	Long packet with distributed routing
0	0	0	1	Long packet with source routing
0	0	1	0	Short packet with distributed routing
0	-	1	1	<i>Illegal combinations</i>
0	1	0	-	<i>Illegal combinations</i>
0	1	1	0	Short packet with distributed routing using local handshaking
1	0	0	0	Long packet with speculative sending and distributed routing
1	0	0	1	Long packet with speculative sending and source routing
1	1	-	-	<i>Illegal combinations</i>
1	-	1	-	<i>Illegal combinations</i>

ing request packet. These are accepted immediately on the port without regard to the qualifier signal. After a while, the port replies with either an accept (*ack*) or a deny (*cancel*) signal which will tell whether the circuit setup was successful or if it failed.

If the routing was accepted as in the figure this means that the circuit has been completed and that the source is free to send data. The payload data is framed by the qualifier signal, making wait states possible in the transmission if such are desired. If a source has no reason to enter wait-states in the transmission, the qualifier may be kept asserted (i.e. high).

When the source is finished and wants to disconnect the circuit this is achieved by lowering the strobe signal. After a one-cycle waiting period the port is ready to accept a request again.

If the circuit could not be set up and the network replies with an asserted cancel signal, the source will have to lower the strobe signal for at least one cycle and then try again. Thus, the strobe will frame the transmission entirely and a low state means that the link is free to use.

Table 5.4: Link feedback signals

Ack	Cancel	Description	Usage
0	0	Idle	Anytime
0	1	Negative acknowledgment	Any packet
1	0	Pos ack, keep route	Long packet
1	1	Pos ack, cancel route	Short packet

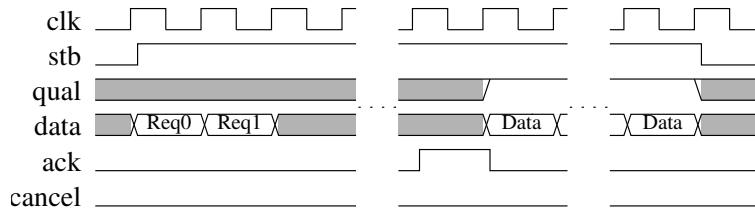


Figure 5.6: Basic link protocol

5.8.2 Speculative sending of data

A drawback with the PCC protocol is the relatively long latency in the setup phase when a packet has to wait for the circuit to be complete before the payload can be sent. A remedy to this is the speculative sending where the data is sent directly after the request has been sent as is shown in figure 5.7. This will require more buffering in the routers as is discussed in section 5.9.3.

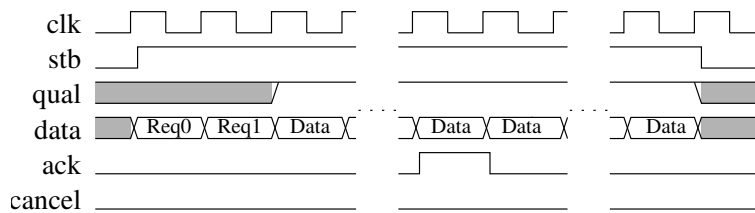


Figure 5.7: Link protocol for speculative sending

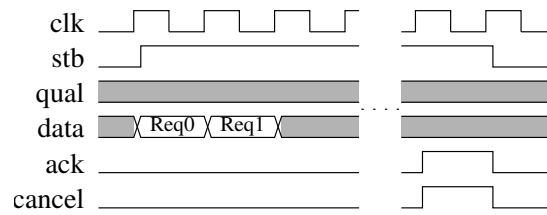


Figure 5.8: Link protocol for short packets

The speculative sending will allow most of the route setup latency to be masked when a successful circuit setup is performed. The data will have to be buffered in the source until the acknowledgment is received because all data will have to be resent at the next try if a circuit could not be set up.

5.8.3 Short packets

The short packets with end-to-end handshaking look exactly as the long packets until the acknowledgment is signaled. A successful transmission will be signaled by asserting both reverse control signals which will imply a disconnect of the circuit to the routers as is shown in figure 5.8.

The possibility to use local handshaking has been introduced as a means to lower the routing overhead for the short packets. When using local handshaking the packet will be moved from router to router while the handshaking will be done entirely locally, i.e. between pairs of adjacent routers. This will give the functionality of a packet switched network. Because there is no end-to-end handshaking there is no method to allow the source to know when the packet has arrived at the destination. The only guarantee is that the packet will eventually arrive there.

5.9 Router implementations

There are four versions of the router, all written in Verilog. The first three router implementations were done with speed as primary target and is

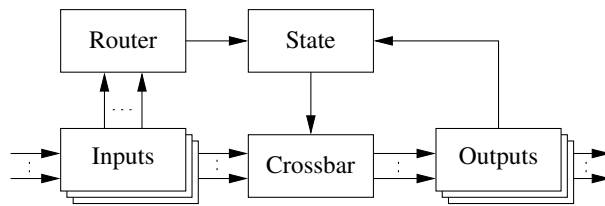


Figure 5.9: Block diagram for the first two routers

thus quite heavily pipelined [4]. These show the evolution of the protocol with extensions such as short packets and speculative sending.

The fourth version has been implemented with the focus of supporting all modes and being as configurable as possible. In this implementation it is possible (through Verilog parameters) to change the number of ports, the data width, and the routing table among others. This implementation has not been optimized for speed (as of this writing).

The three first routers are described in this section while the fourth router is described in detail in section 5.10. The fourth router is the only one to be described in much detail since it includes all functionality of the previous routers. All router implementations are similar so the description of the fourth router is very much applicable to the other implementations as well.

5.9.1 The first router

The first router was implemented so that it would be able to route only long packets. The router was pipelined for maximum performance in the clock domain, reaching an operating frequency of 1.2 GHz (synthesized). The latency for a routing decision is six cycles and the latency for data and ack transport is one cycle through the router.

The block diagram of the first two router implementations can be seen in figure 5.9. The input blocks each contain a small state machine that keeps track of incoming requests. The incoming requests are signaled to the router block that will make the routing decision. The results from

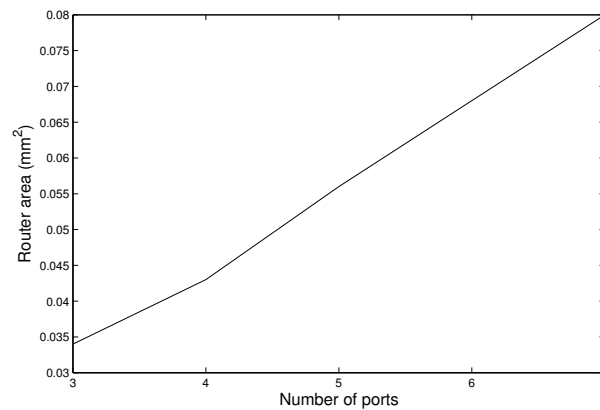


Figure 5.10: Number of ports vs. router area for the first router

the routing block in turn control the crossbar via the router state block. Finally, the output blocks decode the reverse control signals and handle the output state.

A five-port router useful for a mesh network will occupy slightly less than 0.06 mm^2 in a $0.18 \mu\text{m}$ process. The area scales roughly linearly with the number of ports as can be seen in figure 5.10. The reason for the linear scaling is that the only part that scale with the square of the number of ports (i.e. the crossbar) is small compared to the rest of the router.

5.9.2 Short packet router

The second router added support for short packets with end-to-end handshaking. The differences from the first version is basically that there is an improved interpretation of the two reverse control wires to allow a circuit to be ripped up even though a positive acknowledgment is returned. This has very little effect on the area of the router.

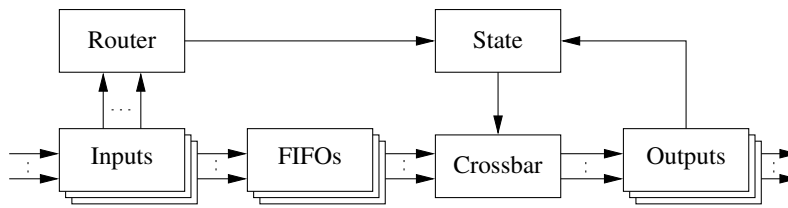


Figure 5.11: Block diagram for the third router (with speculative sending)

5.9.3 Speculative sending support

The routers will have to accept the data that can be sent during the waiting time without losses to allow speculative sending of data. This is necessary since the routers will stall the request for a few cycles and no cycle-by-cycle handshaking is done. The routers therefore require small FIFOs on every incoming port to act as a flexible buffer for the incoming data. The size of the FIFO must be equivalent to the longest waiting time in the router which is typically described by equation 5.2.

$$n_{cy,wait} = n_{cy,routing} + N - 1 \quad (5.2)$$

$n_{cy,routing}$ is the number of cycles required for handling a routing request and N is the number of ports on the router. Thus, $n_{cy,wait}$ is the number of words that the FIFO must be able to store. For this particular implementation, $n_{cy,routing}$ is four, giving a need of a FIFO with at least eight entries per port for a five-port router.

The implemented version with 16 entries per FIFO occupies roughly 0.3 mm^2 in a $0.18 \mu\text{m}$ process. Because of speed issues the FIFOs were implemented with flip-flops giving quite large area. The reason for selecting 16 entries is that the FIFO was not parameterized and therefore had to be large enough for all reasonable number of ports on the router. The FIFOs will give a quadratic scaling of the router area since an addition of one port will not only require the addition of a new FIFO but also an increase in size of all other FIFOs in that router.

The price for avoiding the initial latency incurred by the speculative

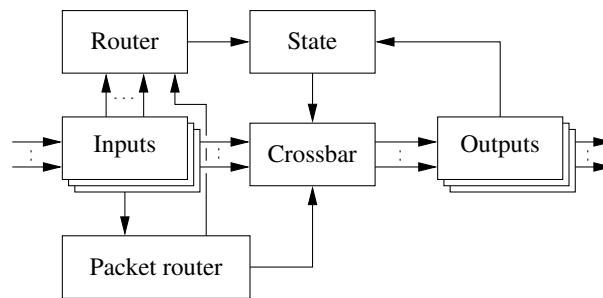


Figure 5.12: Block diagram for the fourth router

sending is quite high with a fivefold increase in area of a five-port router. In certain applications this may still be acceptable since a large portion of the latency in the network is masked by this scheme.

5.10 The fourth router implementation

5.10.1 Configurable and parameterized

The fourth router is a merge of all previous ideas into a single router implementation. This router is designed to be fully parameterized in order to reach the full potential of the network in experimental setups. The focus for this implementation is not extreme speed on an ASIC target but rather a small and reasonably fast implementation on FPGA based systems. The main intention is to use this implementation for experiments and tests that are run on a large FPGA.

The overall architecture is similar to the three earlier versions. The block diagram for the fourth router is shown in figure 5.12. The main change from the previous routers is the addition of the packet router block, which is responsible for the local router-to-router handshaking when using packet switched transport of short packets. Also, the FIFOs for speculative transport has been incorporated in the input blocks and is thus not shown explicitly.

The router is fully parameterized with the possibility to select which of the previously described protocol features that should be supported. An example configuration could involve support for long, short, and local packets but no support for speculative sending. Parameters are also used to select the width of data (at least 16 bits) and which routing table to use.

The function of the packet router is to handle the local handshaking for short packets. An incoming local short packet will be directed to this block that will acknowledge the reception, freeing the resources from the previous hop. The packet router will then find a suitable output port and send the packet to the next router which in turn will accept the packet. Thus it is possible to send the short packets with little resource occupation overhead through the network.

A tool for network generation is easily achieved once a configurable router implementation has been made. Such a tool has been created based on this router implementation and is described further in section 6.7.

5.10.2 Micro architecture

The micro architecture of the router is designed from the blocks shown in the overall architecture in figure 5.12. Each block is intentionally kept simple to simplify verification and keep the overall complexity of the router at a minimum.

Inputs

The input blocks handle the input buffering in the router. The block diagram for this is shown in figure 5.13(a). The incoming requests are buffered in the request buffer that also handles extraction of routing information and stripping of source routing entries.

The optional data buffer is used for speculative sending. The size of this buffer is equivalent to the routing decision delay plus the number of ports on the router minus the request buffer size (i.e. two words). This size corresponds to the longest time to get a routing decision from the router and state blocks.

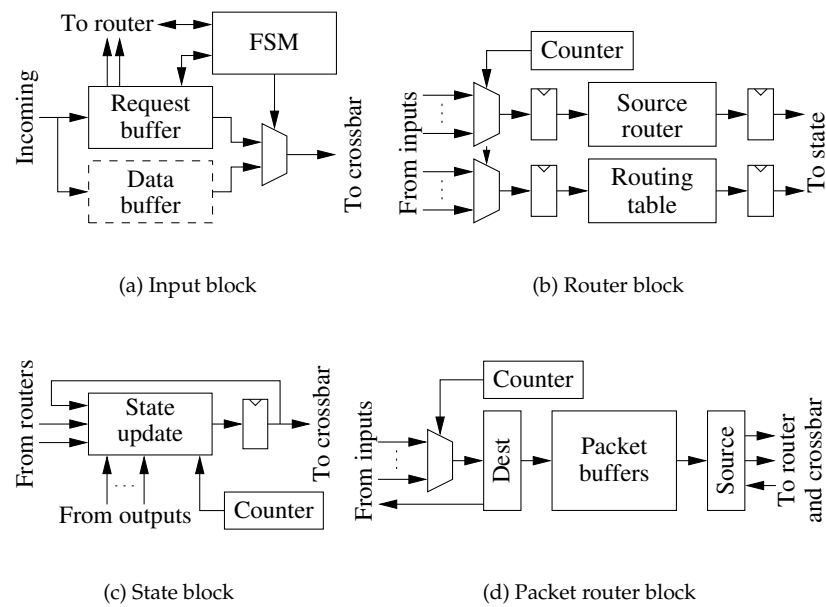


Figure 5.13: Micro architecture for the fourth router

Router

The router block, see figure 5.13(b), is responsible for translation of the incoming routing information into a set of possible outputs. The upper path corresponds to the source routing handling where a request is simply decoded. The lower path is the distributed routing centered around the routing table. The routing table is designed as a ROM table in the current implementation but could be changed into a RAM with the appropriate supporting functionality.

The counter is used for round-robin selection of routing requests if more than one request comes simultaneously. This counter is very short since there is no point in being able to count further than the number of input ports.

State

The router locking state is kept in a single register as shown in figure 5.13(c). The state updating is done atomically in accordance to the incoming routing decisions and acknowledgments to the block. The state is simply a description of which inputs are connected to which outputs in the router. If more than one route is possible, the output port is selected based on the round-robin counter.

Packet router

The packet router is shown in figure 5.13(d). This will act as a destination for incoming packets. Incoming packets are acknowledged and stored in the packet buffer where they are fetched by the packet source. The source will generate a routing request to the rest of the router as if it were an input block. The router will then try to find a route according to this request and the packet can be sent. The packet buffer supports virtual channels to avoid deadlock problems in the packet based network.

Crossbar

The crossbar is a simple N inputs, M outputs crossbar that is controlled externally. No additional functionality is included in this block.

Outputs

The output block is responsible for decoding the reverse control signals and signaling these to the state and input blocks. This block is very simple since the necessary functionality is very limited.

5.10.3 Request path

An incoming request is received at the input where it is buffered. The appropriate routing information is extracted and sent to the router block where it goes through the selector and a pipeline stage. After the pipeline stage comes the routing decision and another pipeline stage. The result

is sent to the state block that will update the state accordingly at the next clock edge. After the state is updated the path will be connected through the router and it is possible to forward the request through the output pipeline stage.

The request path is therefore at least five cycles long from the request being available on the input until it can be pushed to the output. If there are multiple simultaneous routing requests, this will delay the routing decision for some of the inputs accordingly. The latency through the router is one pipeline stage after a path has been established.

The same pipeline stages are present in the three previous routers, although they have been supplemented with an extra stage in the route decision block to achieve maximum clock frequency.

Bibliography

- [1] Ilkka Saastamoinen, David Sigüenza-Tortosa, and Jari Nurmi, "Interconnect IP node for future system-on-chip designs," in *IEEE International workshop on Electronic design, Test, and Applications*, 2002.

- [2] Marco Sgroi, Michael Sheets, Andrew Mihal, Kurt Keutzer, Sharad Malik, Jan Rabaey, and Alberto Sangiovanni-Vincentelli, "Addressing the system-on-chip interconnect woes through communication-based design," in *Proceedings of the Design Automation Conference (DAC)*, 2001.

- [3] William J. Dally and Brian Towles, "Route packets, not wires: On-chip interconnection networks," in *Proceedings of the Design Automation Conference (DAC)*, 2001.

- [4] Sumant Sathe, Daniel Wiklund, and Dake Liu, "Design of a switching node (router) for on-chip networks," in *Proceedings of the ASICON 2003 conference*, Oct. 2003.

- [5] C. Svensson, "Optimum voltage swing on on-chip and off-chip interconnect," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 7, pp. 1108–1112.
- [6] Peter Caputa and Christer Svensson, "Low power, low latency global interconnect," in *Proceedings of the 15th International ASIC/SoC conference*, 2002.

Chapter 6

Design and Simulation Environment

Don't leave port without a multimeter.

"Boatowner's mechanical and electrical manual" (Nigel Calder)

6.1 Design environment

The main difference in design of a network on chip instead of a traditional interconnect is the complexity. Both design and performance evaluation become much more difficult as the complexity increases.

The design environment must be able to support overall performance assessment through analysis or simulation since there is no general theories that will work for a generalized interconnect. Simulation has been chosen as the primary method in the SoCBUS project because of the difficulties involved in accurate analysis of such a complex system.

A relatively complete design environment consists of simulation tools, synthesis tools, and design libraries. The libraries contain simulation models and implementations for the components. The simulation models are used in conjunction with a simulator and the implementations can be used together with the synthesis tools to generate complete network implementations. The design databases used for the toolchain should preferably be as unified as possible, allowing for easy migration from design

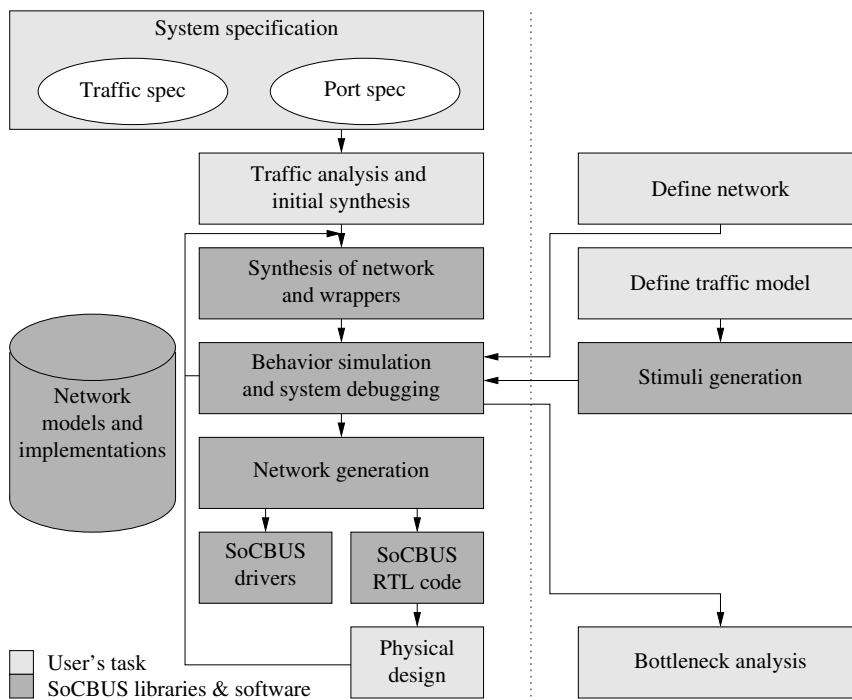


Figure 6.1: Design flow for systems based on network on chip

to simulation and on to synthesis.

A simple method to describe the complete network is essential for the understanding of the design process and the expected results. The choice fell on intuitive XML descriptions for both the network and traffic models.

6.2 Design flow for SoCBUS

The typical design flow for a network on chip starts with an application or an application domain. From this it is possible to extract the communication patterns and requirements that should be put on the interconnect structure. This could be done in a similar fashion as it would be done for a classical TDM based interconnect structure. Generally, this step must

be done with great care to make sure that the traffic model matches the actual traffic behavior in the application. This traffic specification can be used as input for simulations as well as synthesis and optimizations of the network.

The network design flow, see figure 6.1, is divided into two parts as indicated by the dotted line. The part to the left is the customer design flow for implementing a system and is described in this section. The part to the right is for evaluation of performance for general network implementations which can be considered to be parts of a benchmarking flow.

6.2.1 Customer design flow

The customer design flow starts from the specification with an analysis of the traffic between subsystems. The ultimate goal here would be a traffic qualifier tool that takes the (executable) specification/model as input and gives the traffic model for this specification. Just as with other complex co-design tools this one must (for now) be considered utopia and the model have to be extracted using partially manual methods.

The analysis of the traffic will result in an initial network that can be used as a starting point for synthesis. The synthesis generates a network that typically should be reasonably appropriate for the application. The synthesis is followed by the analysis step, based on behavioral simulations, that will verify the appropriateness of the synthesized solution. If the simulation step shows too low or too high capacity in the network the two last steps can be iterated to generate a more optimal network for the application.

The initial synthesis step is still to be implemented as a tool and must currently be performed manually. The simple approach is to start with a mesh network of reasonable size and use this as a basis for optimizations. A tool for supporting the manual synthesis is available in the form of a network model generator that gives an XML description based on a mesh topology. This network model can then be refined into the final network and used as an input to the network generator that will implement this

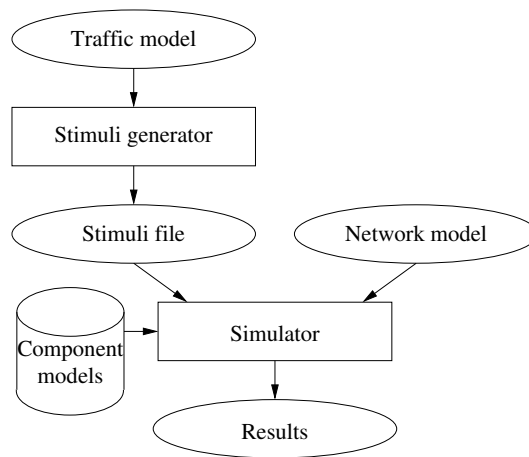


Figure 6.2: SoCBUS simulation flow

XML description on RTL level in Verilog.

6.2.2 Tool coverage

Tools are available for all steps in the design flow, including network model generator, stimuli generator, simulator, and network generator (to RTL). There are also RTL implementations of the router and some basic wrappers. The missing parts include an application-driven initial synthesis, more elaborate wrappers, and driver software for the network.

6.3 Simulation flow

The simulations for the SoCBUS project are done using an in-house simulator that is tailored to the task of simulating on-chip networks. The simulator is used in conjunction with a stimuli generator tool according to the simulation flow in figure 6.2. The inputs to the flow are the traffic and network models, described using XML files.

The network model describes which components to use and the topology of the network. This model can be annotated with additional infor-

mation such as link delays to achieve better simulation accuracy. The network model can also directly be used for network generation to RTL. The mapping of subsystems, i.e. traffic sources and destinations, to network ports is done using literal names in the model files. With this method, all connections between the network interfaces of the subsystems and the routers are done entirely in the network model.

The traffic model consists of a set of communication tasks that are described using any combination of deterministic and stochastic parameters. There is also the possibility to describe dependencies between tasks in order to allow for more realistic traffic cases. The traffic model is used as input to the stimuli generation tool to create fully deterministic stimuli to the simulator.

By separating the traffic and stimuli from the network model, these can be completely decoupled. This will allow traffic modeling that is independent of the network implementation and different networks can be developed without any particular traffic in mind. Thus it is possible to use the same traffic for evaluation of many networks and vice versa. This is a clear advantage in the case of benchmarking.

The stimuli tool will take the traffic model and execute the probabilistic models to create a specific instance of that model. This instance is the stimuli that will be used for the simulation. All stochastic variables are eliminated in the stimuli file, which is fully deterministic so that two runs of the simulator with the same stimuli and the same network should yield the same results. Having the opportunity to reuse stimuli is important for several reasons. The primary reason is for evaluation of several networks, which becomes more fair if the stimuli is exactly the same for every network. Another reason is that debugging of tools and models can be significantly simplified by running the same stimuli over and over.

6.4 Traffic modeling

Modeling of traffic is by no means a simple task. The modeling language chosen for the SoCBUS project is XML with specialized tags for the traffic

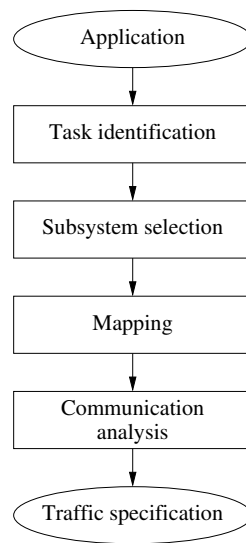


Figure 6.3: Traffic modeling flow

description. The models of computation that is supported by the XML models is basically communicating tasks, similar but not restricted to Kahn graphs [1]. These tasks can be described along the entire scale from deterministic, periodic tasks to tasks with completely random behavior. Most signal-processing-style applications can be described by this type of model of computation. The power of Kahn graphs (and similar models of computation) is widely recognized throughout the industry and academia today [2, 3].

The traffic modeling for a regular application can be quite simple with a straight forward approach. Figure 6.3 shows a typical flow from application to traffic model. The application is divided into communicating tasks using some model of computation. These tasks can then be used as a basis for selecting the appropriate subsystems in the platform if the platform is not already given. The tasks can then be mapped onto the subsystems and the communication patterns between the subsystems can be analyzed.

As can be expected, the complexity of the application has a significant influence of the complexity of the traffic model. A simple dataflow application can have a very simple model even though there is much traffic. A more complex system on the other hand may give a quite complicated model that is difficult to understand and debug.

6.4.1 Stimuli files

The stimuli file that is generated is a simple text file. For traffic without dependencies the stimuli will use absolute time stamps (i.e. fully scheduled traffic) so the information in the stimuli file will just be source identifier, destination identifier, start time, and packet size for each transfer.

If the traffic is based on dependencies there will be the same information showing the earliest time of initiation, together with dependency clauses which describe the dependency conditions for starting a transfer.

Since the stimuli is represented by text files, it is possible to write the stimuli files directly, bypassing the stimuli generation tool. This is only practical for very small models and is suitable for debugging only.

6.5 Stimuli generator

The stimuli generator is responsible for taking the traffic model in XML as input and generate the deterministic stimuli as output. Since the input model may describe traffic using statistical mathematical models these have to be calculated into deterministic values.

The traffic model specifies each traffic flow in relative detail using a set of XML tags. The stimuli generator must use each of these specifications to create the stimuli for that particular traffic. Each value in the traffic model can either be explicitly given or specified through the use of a stochastic model. In turn, each of the parameters of the stochastic model can be specified explicitly or as another stochastic model. With this recursive definition there is no limit on the probability-based parameterization of the model.

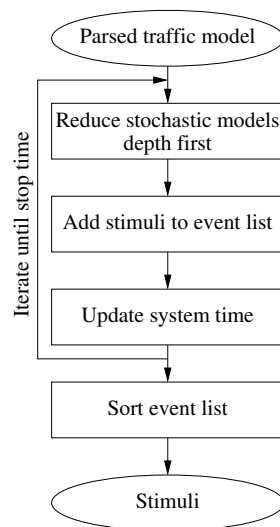


Figure 6.4: Translation from traffic model to stimuli

The program flow of the translation process in the stimuli generator is shown in figure 6.4. Because of the hierarchical stochastic models in the model, the stimuli generator must do a depth-first reduction of the stochastic models in each iteration.

The stimuli generator is implemented entirely in ANSI C++ using the GNU Scientific Library (GSL) [4] for implementations of the statistical models and handling of probability-based models. The XML parser has been developed and reused for all three XML-based tools, i.e. the stimuli generator, the simulator, and the network generator. This parser has been implemented using the GNU tools Flex [5] and Bison [6]. More details on the stimuli generator implementation can be found in appendix A.

6.6 Simulator architecture

The simulator is event based, similar to a typical VHDL/Verilog simulator. The big advantage of event-driven simulation is the fact that idle

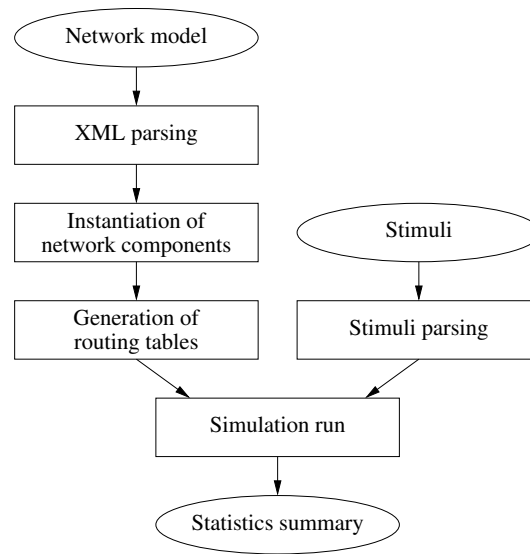


Figure 6.5: Simulator top level flow

parts of the system will not generate events giving faster simulation runs. This, on the other hand, also implies that a simulation with much traffic will have a significantly longer runtime compared to a low-density simulation.

The main program flow of the simulator is shown in figure 6.5. The network model XML file is read and parsed into an internal representation. This internal representation is used to instantiate the equivalent network from the component models. The routing tables that have not been explicitly given in the network model are generated using Dijkstra's algorithm [7]. The stimuli file is read and parsed and used to excite the network model in the core simulator. Finally, the results and statistics acquired during simulation is written to a log file.

All network components (routers, links, sources, and destinations) are modeled using cycle-accurate behavioral models. These models are implemented using the same programming language as the simulator, i.e. C/C++. These models are currently linked to the simulator at compile

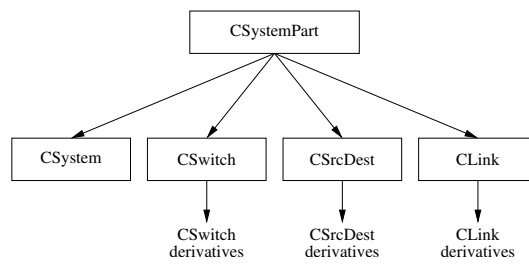


Figure 6.6: Simulator class hierarchy



Figure 6.7: Simulator event handling

time. The use of shared libraries would also be possible making it possible to compile new models without recompiling the rest of the simulator if that is desirable.

Figure 6.6 shows the inheritance hierarchy among the models in the simulator. All models are based on the CSystemPart class which gives a common ancestor class, making it possible to treat all models as one type of entity in the simulator.

Using C++ also allows extensions to be made to the tools in this commonly known and efficient programming language. Further details on the simulator implementation than is given here can be found in appendix A.

6.6.1 Simulation event handling

A simulation message is created every time a network component generates an event. These event messages are distributed to the proper destination(s) through a message server, see figure 6.7. This central message server will handle all time ordering of event messages through a single

message queue. This guarantees that the events will happen in the correct order independent of the order in which they have been created. This is true as long as only causal events (i.e. forward going in time) are used.

The models are connected logically to each other through the message server. If a model wants to send a message to another model, e.g. a router to a link, it will create the message and tag it with the appropriate address for the destination model and then send it to the message server. The message server will queue the message and dispatch it at the correct time to the destination which is then able to react to the message, possibly creating a new message in the process.

The clock generator module is used to create clock event messages that is distributed to the appropriate models. This feature is used to get cycle-true simulations.

6.6.2 Simulation models of network components

All network related models that are used in the simulations are derived from the RTL implementations. By using cycle-accurate models together with cycle-true simulation, it is possible to get accurate results faster than through RTL simulations.

Models can be easily changed or added through the use of a simple API in C/C++. This is very useful if new implementations of components are made or the need to model certain IP blocks/subsystems in a more thorough fashion arises. Multiple models for a single component type (e.g. routers) are supported. Which model to use is selected by supplying a model type name in the network description.

The model for the first router implementation is based on a set of state machines, one for every input port of the router. A simplified state diagram of these machines can be found in figure 6.8. All updating of the state machine is done upon reception of an appropriate clock message. Even though the router consumes five cycles for a route lookup, the model will generate the response message immediately. This message is timestamped five cycles in the future to give the correct delay. This type of

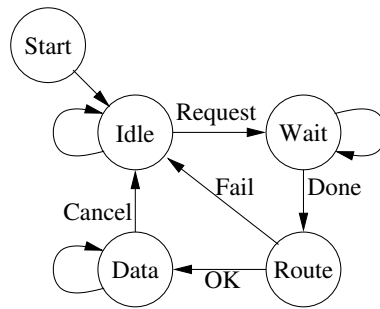


Figure 6.8: State machine for the model of the first router

model will guarantee the cycle accurate behavior without the model having to keep track of a number of events that happened earlier on in the simulation.

6.7 Network generator

A network generation tool has been developed that takes a network description in XML as input and generates the Verilog code for that particular network. The network generator relies on the router implementation being parameterized so that the number of ports etc. can be set for each router.

The top level of the generated Verilog will have a set of ports, one for each IP block (subsystem) that is described in the XML file. By instantiating this top level, all network functionality including routing tables are included in the design. The Verilog code have all relevant parameters set within the generated code with the possibility to change the appropriate parameters (e.g. data width) at instantiation.

6.8 Network generator architecture

Figure 6.9 shows the top level flow for the network generator. This is intentionally very similar to the simulator flow, allowing for reuse of large

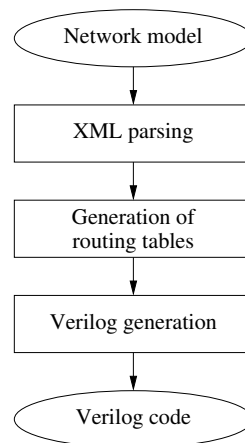


Figure 6.9: Network generator top flow

portions of the nonsimulation code. A XML file is parsed and the internal representation is used to derive the topology for creating the implicit routing tables, again using Dijkstra's algorithm. The complete internal representation with topology, routing tables, and so on is then used to generate the Verilog code for the network. Currently, only the fourth router implementation is supported in the network generator, simply because this is the only router with a high enough level of parameterization.

Bibliography

- [1] Gilles Kahn, "The semantics of a simple language for parallel programming," in *Proc. of the IFIP congress*, pp. 471–474, Aug. 1974.
- [2] Marino T. J. Strik, Adwin H. Timmer, Jef L. van Meerbergen, and Gert-Jan van Rootselaar, "Heterogeneous multiprocessor for the management of real-time video and graphics streams," *IEEE Journal of Solid-State Circuits*, vol. 35, Nov. 2000.

- [3] Thomas M. Parks and David Roberts, "Distributed process networks in Java," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.
- [4] Free Software Foundation, *GNU Scientific Library – Reference manual*. <http://www.gnu.org/software/gsl/manual/>, 2005.
- [5] Free Software Foundation, *GNU Flex 2.5 – Reference manual*. <http://www.gnu.org/software/flex/manual/>, 2005.
- [6] Free Software Foundation, *GNU Bison 2.0 – Reference manual*. <http://www.gnu.org/software/bison/manual/>, 2005.
- [7] Harry R. Lewis and Larry Denenberg, *Data structures and their algorithms*. Harper Collins Publishers, 1991. ISBN 0-673-39736-X.

Part IV

Benchmarking of Interconnect Structures

Chapter 7

Benchmarking in General

...being honest and impartial and serving with fidelity...

The American Society of Civil Engineers' Code of Ethics

7.1 Motivation

The motivation for benchmarking of hardware is quite simple. The number of hardware platforms and IP blocks available continually increase. Many of these are targeted for the same or similar applications. With this increasing range of implementation come the problems of selecting the most appropriate solution for a particular problem. The relative performance for a set of solutions is the most influential fact (apart from the actual price) used in the selection process.

The performance evaluation is commonly known as *benchmarking*. The need for benchmarking is omnipresent in the electronics industry, even if the target metric differs widely depending on the intended application and market. In a microcontroller used for a laundry machine the price, availability, and simplicity in mounting may well outweigh the performance, while a military radio will have a significantly different set of conditions.

7.2 Definitions

There are some definitions that are very useful for the continuing discussion on the topic of benchmarking. These are mostly concerned with how to interpret the word *benchmark* in different contexts.

Definition 1 *A benchmark is the combination of the specification(s) that have been used and the result(s) that have been achieved in the process of benchmarking.*

Definition 2 *A bottleneck is a performance-limiting factor in a system.*

Definition 3 *A benchmarking method is a specified method used to create (i.e. specify) and run benchmarks in order to find bottlenecks.*

Definition 4 *A benchmarking process is the process of using a benchmarking method and a benchmark specification in order to get benchmark results and finding bottlenecks.*

7.3 Performance metrics

The performance metrics used in benchmarking may also vary significantly. There are many possible metrics but only a few that really indicate the true performance. Consider a processor benchmark where there is a small program, written in C, that is compiled to the native assembly of the processor. Two distinct metrics are easily derived from the single measurement of running time. First is the rate, i.e. how many runs per second that the processor is capable of. The second is the number of instructions per second that the processor executes for the benchmark. The former metric is a clear indication on the true performance of the processor and can freely be compared to other processors. The latter metric does not really say anything about the performance since a processor with few, specialized instructions may complete the processing in the same time as a general-purpose RISC machine running at much higher clock frequency

and that uses many more instructions to solve the same task. If this sole metric of instructions per second is used, it seems like the RISC machine is superior to the specialized machine even though the real situation is the opposite.

Metrics that may be useful for network performance benchmarking is bandwidth, latency, power dissipation, etc. These are by no means the only metrics since other factors will also affect the overall performance. For example, a network implementation that will give the appropriate bandwidth and latency may be far too costly in terms of silicon area. Such a network would not be suitable at all even though the metrics indicate that it is. This example clearly shows the importance of realizing the significance of using the correct metrics for the evaluation process.

7.4 Average vs. worst-case performance

Not only the measurements but also the method of sampling is important for the results of the benchmark. The two most commonly used methods are average-case and worst-case sampling. Average-case focuses on the average value of the measurements to get the metric. This could for example be an average over a set of runtimes for a particular application. Worst-case sampling on the other hand is only interested in the worst possible value, e.g. the longest runtime.

Average sampling is appropriate in many cases where there is only a need to estimate the overall performance during prolonged time periods, as is typically the case for personal computers. This average can be supplemented by statistical variances and deviation measures to form a measurement interval.

Using statistical measurements based on averages will only give indications on the expected value. If real-time behavior is to be guaranteed this is not adequate because the worst-case behavior is not given even implicitly. In this case the worst-case performance is the crucial sampling point, although average behavior may well be interesting for making optimizations to the system.

7.5 Measurement techniques

The accuracy of the benchmark is not only affected by which measurements to take but also how these measurements are taken. There are three main techniques used for measuring the system behavior, namely event driven, sampling, and indirect.

Event-driven measurements are based on events that happen in the system, i.e. a counter may be incremented when a packet has been received. Event-driven measuring is generally the most accurate method. A special case of event driven measurements is called *tracing*. Tracing involves saving information about the system state at the events, making it possible to replay the causes or effects of the events.

Sampling uses periodic checks of the state to deduce the events between the states. In this case, measuring the number of received packets could be done by checking if a new packet is coming in at the sampling instant. Sampling may miss events if they happen too often (analogous to the Nyquist sampling theorem).

Indirect methods have to be used if there is no direct method of reacting to events or sample some quantity. With indirect measurements, a quantity that can be measured directly is used to deduce the probable value of the indirect quantity.

The accuracy of the measurements is important to consider. Quantization effects may give very imprecise results for measurement values that are small compared to the resolution of the measurement.

One clear advantage of benchmarking using behavioral simulation is the possibility to take the measurements nonintrusively. The accuracy of the measurements can be significantly increased when the system is not directly affected by the fact that measurements are being taken.

7.6 Comparing results

Taking measures and formulating metrics can be done relatively easy in a objective fashion. Making results comparable across platforms and per-

forming the comparisons can be more of a problem. In order to have fair benchmarking, all measurements must be taken in a strictly controlled environment. This is necessary in order to prevent some systems from having more or less favourable conditions during benchmarking compared to the other which will lead to a biased benchmark result. This problem is well known in any type of performance comparison and has been known to confuse customers for decades.

7.7 Processor benchmarking

There are several benchmarking suites for processors, like the SPECint [1] and SPECfp [2] for general-purpose processors. These benchmarking suites are very common among both commercial and academic users. Digital signal processors are benchmarked commercially by Berkeley Design Technology, Inc. [3].

A general discussion on processor benchmarking is beyond the scope of this thesis. See the book by Lilja for more information on this topic [4].

7.8 Network benchmarking

Benchmarking of networks is usually constrained to the individual components, e.g. routers, or particular applications, e.g. web servers. An example is the Internet core router test published by Light Reading [5].

Bibliography

- [1] Standard Performance Evaluation Corporation, *CPU2000 benchmark suite, Integer benchmarks (SPECint)*. <http://www.spec.com>, 2000.
- [2] Standard Performance Evaluation Corporation, *CPU2000 benchmark suite, Floating point benchmarks (SPECfp)*. <http://www.spec.com>, 2000.
- [3] Berkeley Design Technology, Inc. (BDTI), *Evaluating DSP Processor Performance*. <http://www.bdti.com>.

- [4] David J. Lilja, *Measuring computer performance*. Cambridge University Press, 2000. ISBN 0-521-64105-5.
- [5] David Newman, *Internet core router test*. Light Reading online, <http://www.lightreading.com>, 2001.

Chapter 8

Benchmarking of On-Chip Interconnects

If you can find something everyone agrees on, it's wrong.

Mo Udall

8.1 Benchmarking of interconnects

There are a number of differences between benchmarking of computational resources, e.g. DSP processors and interconnection networks. The most important differences are shown in Table 8.1. The two upper sections relate to the benchmark specification and the two lower sections relate to the results. The idea behind each category is similar, e.g. constraints make both the DSPs and NoCs comparable in the sense of disallowing “cheating”.

The constraints set the limitations of the interconnect to be benchmarked such that it has to be implementable (and possibly reliable). The specification in turn tells what kind of traffic should be simulated over the interconnect.

The results are a combination of the target information and simulation results. Target information are architectural choices such as word length (i.e. link width), if quality of service is supported in hardware, etc. Some results that may be acquired in the benchmarking are measures on

Table 8.1: Comparison of DSP and NoC benchmarks

	DSP	NoC
Constraints	<ul style="list-style-type: none"> · Native precision · Include round/sat · Only core DSP (i.e. no accelerators) 	<ul style="list-style-type: none"> · Transaction level · Implementable NoC · Only complete transactions (no loss)
Specification	BDTI examples: <ul style="list-style-type: none"> · Real block FIR · Complex block FIR · Vector dot product 	<ul style="list-style-type: none"> · Traffic pattern · Packet size(s) · Number of ports
Target info	<ul style="list-style-type: none"> · Word length · Clock frequency · Hardware cost 	<ul style="list-style-type: none"> · Word length · QoS in hardware · Hardware cost
Results	<ul style="list-style-type: none"> · Cycle cost · Program memory use · Data memory use 	<ul style="list-style-type: none"> · Throughput · Latency · Buffer usage

throughput, latency, buffer usage, etc.

8.1.1 Benchmarking method

The principle of the benchmarking method is to create a traffic model for the interconnect simulation that reflects the benchmark specification. This traffic model is then used to excite the network model in the appropriate simulator. The simulations will give (approximations of) the performance for the given network(s) under the traffic specified.

The benchmarking flow is as follows:

1. If possible, create the necessary instrumentation in the simulator to allow nonintrusive benchmarking if this is not already supported.
2. Translate the benchmark specification into a traffic model for the simulator. It is important to note that this translation might become

suboptimal. Depending on the impact of the design flow the benchmark may also take the tool chain and methodology into account.

3. Execute the simulations with the traffic model and one or more network models. The traffic model is hereby used to create stimuli that is used to excite the hardware model as given in the network specification.
4. Collect the results from the simulations and evaluate these.

8.1.2 Benchmark specification

The specification of the benchmark is basically the traffic model specification. As is discussed in section 6.4, such a specification can be presented using many different methods. Examples are Kahn graphs [1] and communicating synchronous dataflow graphs [2]. Another possibility is to simply use multiple state machines and flow charts to describe the traffic flows. The important point is that the model used must be able to describe the system accurately without the model becoming too complex and error-prone.

Apart from the traffic model the benchmark specification may also put constraints on the network model, e.g. the maximum silicon area allowed for the network or the maximum clock frequency in the network. The expected level of real-time behavior in the system is also important to specify since this is important for the interpretation of the benchmark results.

8.1.3 Interpretation of results

The most important part of the benchmarking process is to interpret the simulation results correctly. Even direct figures like a throughput measurement can be misleading if taken out of context. The only appropriate method to read a simulation result is to couple it with the traffic situation and interconnect architecture used for the specific simulation. Because

of this and the general complexity of interconnect systems, each comparison must then be based on an individual interpretation to find the specific differences in each case.

This complicated interpretation of the results is one of the major obstacles for the use of generic benchmarks for interconnects. For a processor it is very simple to state measurements, e.g. the run times for a set of kernel algorithms, which will characterize the processor quite thoroughly. This kind of information can easily be published while revealing very few architectural details.

8.2 Benchmarking examples

This section presents two benchmarking examples where the impact of burst size (i.e. packet size) and the total data rate is investigated. By using a specification with open variables the benchmark will cover a certain design space, giving not only a single number as a result but rather a result hypercurve over this design space. Generally, with n independent variables in the benchmark specification, the results will be a hypercurve in n dimensions. In this case, the variables are limited to two giving results over two dimensions which can easily be visualized as 3D graphs.

The main target result for the examples given here is data throughput through the network. Several other result functions may be considered, e.g. latency, but this is limited to a single result function for simplicity.

These traffic models are not intended to model a specific application but have rather been selected to show the applicability of the methods described in this thesis.

8.2.1 Example 1: Specification

Network

The network considered should have 64 ports for connection to subsystems. The network is limited to a mesh implementation of the SoCBUS network. This implementation is selected because of the simplicity in the

design and the fact that this would typically be the original topology chosen for the initial network before optimizations. The reason for just looking at SoCBUS is that no other network on chip models were available to us at the time.

Traffic

All traffic originates and terminates in the 64 combined sources/sinks connected to the network. The outgoing data rate for each source is varied over a range from 5% to 80% of the cycles (assuming synchronous model). Also the packet size is varied from 20 to 3000 words per packet for each of the data rates. These communicate with each other in a random fashion with uniformly distributed starting times.

Two variations on this benchmark has been simulated. The two versions differ in how the source/sink pair selection have been specified. The first case is totally random selection of source and sink over the entire set. The second case assumes locality when the sources and sinks are distributed evenly over a square (i.e. a 2D mesh). The sources are then selected randomly while the sinks are selected randomly within a radius of two sinks away from the source.

8.2.2 Example 1: Results

The theoretical maximum throughput at the inputs is 64 words per cycle. The simulation results for the first sink selection case can be found in figure 8.1. The graph shows a low saturation level about five words per cycle for the smallest packet size (20 words). The saturation level increases to about 21 words per cycle for larger packet sizes. The reason for the relatively low saturation limit with small packets is that the overhead will dominate the transfer. The low overall saturation level is due to the congestion in the network center when traffic can go over the entire network.

For the second case (with a sane network port allocation) the throughput will reach the levels shown in figure 8.2. Small packet still give a

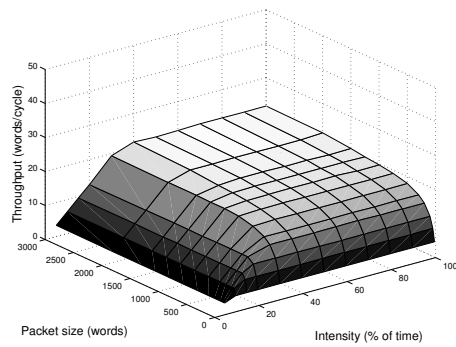


Figure 8.1: Throughput for random traffic on 2D mesh

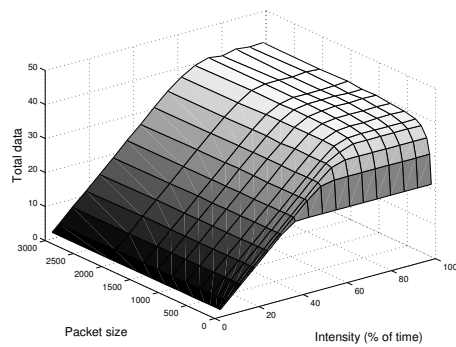


Figure 8.2: Throughput for traffic with locality on 2D mesh

low saturation limit of about 22 words per cycle whereas the use of larger packets will raise the throughput to roughly 40 words per cycle. The reasons for the higher saturation level is the significantly reduced congestion in the network. These two benchmark variations clearly show the impact of locality for this specific network.

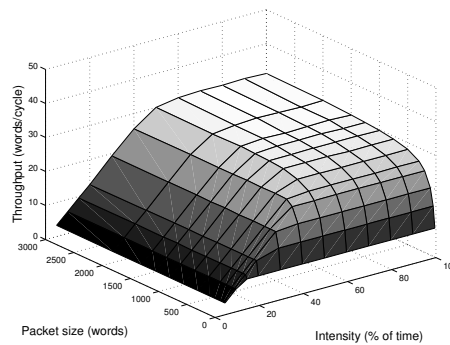


Figure 8.3: Throughput for 2D torus

8.2.3 Example 2: Specification

Network

The network specification is the same as in the first example, with the exception of the topology. In this case, the topology is selected as an additional variable. Four different network topologies have been simulated, namely the 2D mesh, 2D torus, 3D mesh, and 3D torus. Both 2D topologies are of size 8×8 while the two 3D topologies are built from $4 \times 4 \times 4$ routers.

Traffic

The traffic specification is exactly the same as in the first case of the first benchmark example. All traffic is completely random without any restrictions on selection on source/sink pairs.

8.2.4 Example 2: Results

The results for the 2D mesh is exactly the same as for the first case in the previous example, see figure 8.1. If the simple change from a mesh to a torus is made, this will have a quite large impact on the result. With this topology it is possible to reach beyond 31 words per cycle at the saturation

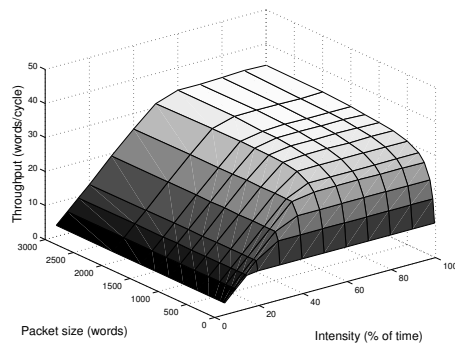


Figure 8.4: Throughput for 3D mesh

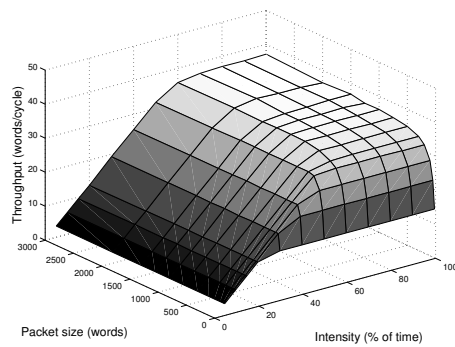


Figure 8.5: Throughput for 3D torus

point, which is almost 50% more than for the mesh. The reason is the decreased congestion in the network coming from the significant increase in the number of possible routes between sources and sinks.

The 3D mesh, see figure 8.4, shows a small improvement over the 2D torus. Here it is possible to reach above 32 words per cycle. The last case, the 3D torus is the best in this benchmark. Figure 8.5 shows the throughput for this network, reaching more than 37 words per cycle. The increase from the 2D torus to the 3D torus is not very impressive, considering the fact that there are 50% more links in the 3D torus. The 2D torus is definitely interesting for systems with random or near-random traffic

patterns, e.g. general computing.

Bibliography

- [1] Gilles Kahn, "The semantics of a simple language for parallel programming," in *Proc. of the IFIP congress*, pp. 471–474, Aug. 1974.
- [2] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE transactions on computers*, vol. C-36, Jan. 1987.

Part V

Applications

Chapter 9

Internet Core Router

Core: a tiny doughnut-shaped piece of magnetic material (as ferrite) used in computer memories

Merriam-Webster Online Dictionary (2005)

9.1 Brief introduction to core routers

The evolution in electronics together with the rapidly increasing interest in the Internet has led to a bandwidth explosion. Today, the traffic on the Internet doubles about every six months. All service providers and backbone suppliers must therefore keep up with the pace in deploying new and faster network components.

Figure 9.1 shows a schematic view of the Internet. The end users have their terminals connected to the service provider's network. The service providers in turn connect their network through their edge routers to the core network. The core network consists of a set of high-performance routers, known as core routers.

There are several design differences between core routers and edge routers. The perhaps most obvious is the bandwidth requirement that is higher in the core network. Also, the sizes of routing tables are significantly larger in a core router because of the large number of links to other routers. On the other hand, the edge routers may consider other aspects, e.g. quality of service requirements and complex packet filtering.

The application case study described in this chapter is a core router for

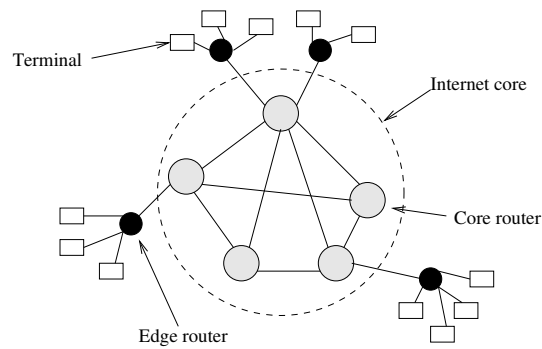


Figure 9.1: Schematic view of the Internet

the Internet, where high packet throughput is the main design target. The design goal is a single-chip core router that is capable of accepting 16 ports with 10 Gbit/s data rate transporting TCP/IP traffic. The standard that has been targeted is 10 Gbit/s Ethernet [1], although Sonet/SDH OC-192 [2] would also be appropriate as a physical media. This solution will provide a cost-effective means to expand the core network with more bandwidth.

Several years have passed since the first OC-192 router was introduced on the market. The main difference to the one considered in this case study is the physical size where the currently deployed core routers tend to be on the scale of racks rather than chips.

9.2 Core router processing flow

The main task of the router is to filter out bad packets and forward the good packets to the appropriate output port. In order to do this, four main tasks has to be done:

1. Check incoming packets for errors, e.g. through a CRC check.
2. Classify the packets according to type, destination address, etc.
3. Make a routing decision.

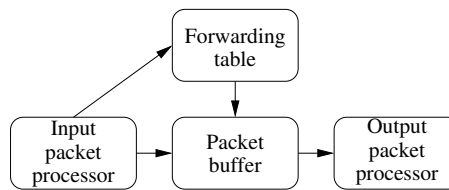


Figure 9.2: Dataflow in the core router

4. Update headers and calculate new checksum and CRC.

A side effect of this processing flow is that the payload data has to be stored during the route lookup. This is necessary because the route lookup may consume a varying time dependent on how the appropriate address matching rules looks like. Based on the routing decision, the payload will be repacketized with updated headers at the output of the core router.

The processing and dataflow is shown in figure 9.2. The packet arrives at the input packet processor (IPP) where initial classification, error checking, etc. is done. The header information is forwarded to the forwarding table (FT) and the payload is sent to the packet buffer (PB) for storage. The packet buffers also handle any off-chip memory that is necessary to cope with bursts of heavy traffic. The forwarding table will make a routing decision and send this to the packet buffer for further forwarding to the appropriate output packet processor (OPP). The OPP must then repacketize the data, update the headers, and calculate the new checksum.

Because of processing requirements with line speed packet handling, the IPPs and OPPs all take care of one Ethernet port each. Because of the central functionality of the FT and PB, these should preferably only be instantiated once. Because of bandwidth and processing limitations this is impossible as will be shown later.

The processor (CPU) is used for the regular maintenance and routing table update tasks. The multicast unit (MU) handles multicast packets. These two blocks are not considered further in this case study.

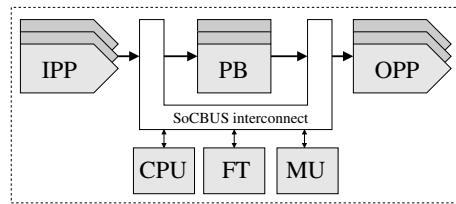


Figure 9.3: Core router system architecture

9.3 Function mapping

The function mapping of the core router is quite straight forward. The four dataflow steps in figure 9.2 are mapped directly to hardware blocks. These are interconnected using a SoCBUS network as depicted in figure 9.3. The input to the IPPs and the output from the OPPs are connected to the (off-chip) Ethernet PHYs. All other connections are made through the SoCBUS network.

Iterative design where the current solutions were evaluated using simulations resulted in the final network mapping and functional unit allocation found in figure 9.4. The forwarding table unit has been replicated four times because of the required bandwidth and processing requirements will be far from reachable in an implementation. If a single port to the FT was used, this would give too high requirements on the SoCBUS network connection of that unit. The maximum lookup rate for minimum size packets will be around 260 million lookups per second. This rate is possible to achieve without using specialized CAM memories [3].

Further, the packet buffer unit has been replicated eight times because of the bandwidth requirements. Every packet buffer will use its own buffer memory, which may lead to inefficient memory usage in case of bursty traffic on some ports while other ports are idle. The resulting advantage in bandwidth is necessary in order to meet requirements.

Because of the processing requirements for handling packets at line speeds of 10 Gbit/s all units must be implemented as dedicated (possibly configurable) hardware blocks. Using a general processor is not viable for

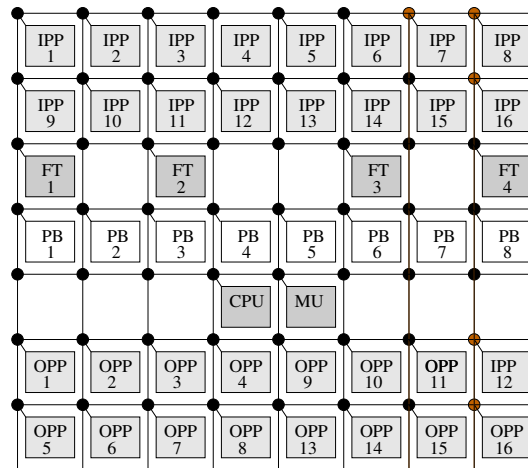


Figure 9.4: Final network allocation and mapping

this kind of system because of the limited processing power due to the unsuitable architecture. The individual components can be implemented as dataflow processors in the case where programmability is needed.

The selected architectures for the units will ensure ordering of the individual flows. This means that all packets coming in on one port will be delivered in the same order to the output ports. In-order delivery of packets is highly desirable for TCP flows.

9.4 Simulation setup

The simulations were done using an in-house simulator specifically developed for the SoCBUS project. Models for the IPP, FT, PB, and OPP were developed and compiled into the simulator. The capability to cosimulate the network and IP blocks considering the dependencies between internal communications is essential for the accuracy of the simulation results.

Separate models for the internal traffic and SoCBUS network connections are written in XML and used as input to the simulator. By separating the models it is possible to change the traffic or network model without

Table 9.1: Packet-size distribution for Internet mix

Probability	Packet size
56%	40 bytes
23%	1500 bytes
17%	576 bytes
5%	52 bytes

affecting the other. This is particularly useful when a set of traffic patterns should be run over a set of networks.

9.5 Simulation results

9.5.1 Traffic patterns

Three distinct traffic patterns have been used for the Internet traffic. The first pattern is a statistical distribution of packet sizes that is derived from observations on live traffic on the Internet, known as the Internet mix [4]. The Internet mix is a combination of four fixed packet sizes, see Table 9.1. Using this traffic will mimic the behavior of a real network router.

The second pattern is evenly distributed packet sizes according to RFC2544 [5]. The packet sizes according to this distribution are evenly spread across 64, 128, 256, 512, 1024, 1280, and 1518 bytes.

The last traffic pattern used is constant streams of minimum-size packets. This is used to check the route lookup performance of the core router. All three patterns use a flat distribution for selection of destination ports, making the bandwidth per output port the same as for the input ports on average.

By varying the intensity of the traffic, i.e. the arrival rate of packets, it is possible to create incoming traffic with variable bandwidth. This can be used to find the point where congestion will occur in the router.

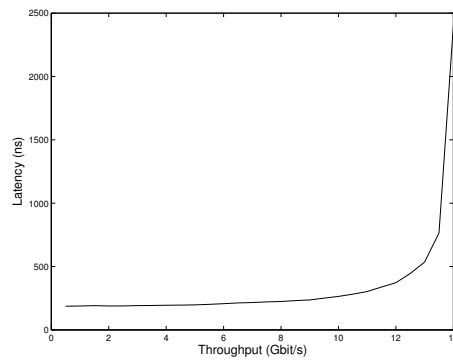


Figure 9.5: Internet mix: Average packet latency

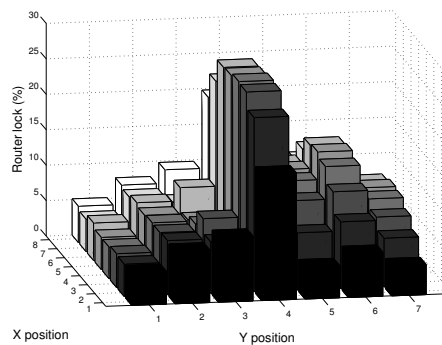


Figure 9.6: Internet mix: SoCBUS router-ports lock

9.5.2 Internet mix simulations

The first simulation runs were done using the Internet mix traffic. The average routing latency as a function of offered load (i.e. expected throughput) per network port is shown in figure 9.5. For the Internet mix type traffic, it is possible to reach beyond 13 Gbit/s sustained throughput per port before congestion occurs and the router will begin to drop packets.

Figure 9.6 shows the usage of the router ports through the SoCBUS network. The X- and Y-axes in the figure corresponds to the horizontal and vertical axes in figure 9.4. It is clear that the most intense traffic is

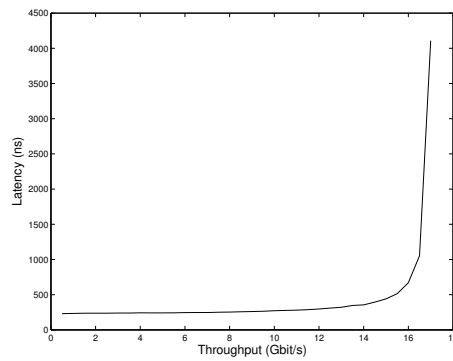


Figure 9.7: RFC2544: Average packet latency

around the packet buffers where almost all traffic originates or terminates. The payload is sent from the IPPs to the PBs and then on to the OPPs. This will give a network usage around the packet buffers equivalent to more than double the bandwidth of incoming packets to the router. There are also routing results sent from the FTs to the PBs, further increasing the congestion around the packet buffers.

9.5.3 RFC2544 simulations

The simulations based on RFC2544 patterns show a similar behavior as the Internet mix but with the maximum throughput increased to around 16 Gbit/s per port, as shown in figure 9.7. Because of the similarities with the Internet mix, this case will not be discussed further.

9.5.4 Minimum size packets simulations

The minimum size packet simulations show a significantly lower throughput than the previous two cases, as shown in figure 9.8. The congestion point is now at around 2.6 Gbit/s per port.

The reason is that the SoCBUS ports to and from the forwarding tables and the packet buffers can no longer cope with the immense number of

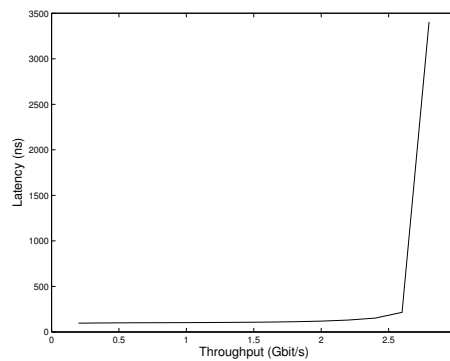


Figure 9.8: Min size: Average packet latency

packets that arrive. Figure 9.9 shows the port usage for outgoing transmissions from the functional blocks. The row corresponding to the packet buffers show a peak usage of about 80% which is very high, considering that the traffic into the packet buffers actually exceeds the output traffic because both the IPPs and FTs communicate to the packet buffers.

One reason for the high port usage is that the PCC circuit setup scheme will put a high communication overhead penalty on small packets. Packets that are of a single word, such as the routing information sent from the IPPs to the FTs, will have a transmission overhead of several times the payload size on the SoCBUS network. Either grouping several requests together using a single SoCBUS packet or lowering the overhead will increase the throughput significantly.

9.6 Conclusions

A viable design for a 16-port, single-chip Internet core router has been presented. This design shows a performance limit of about 14 Gbit/s per port with typical Internet traffic and 16 Gbit/s per port for RFC2544-style traffic.

Pure minimum size packet traffic will limit the performance to about

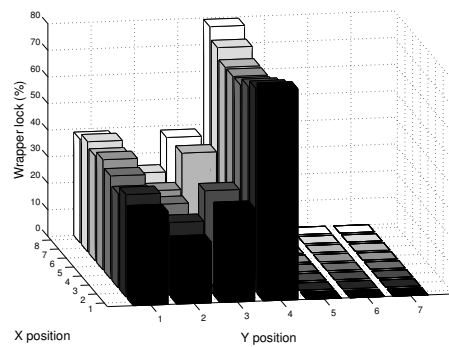


Figure 9.9: Min size: Source port locking

2.6 Gbit/s per port because of limitations in the internal communication structure and the forwarding tables. The limiting factor in the minimum-size packet case is the SoCBUS connections to the forwarding table units and the circuit switching scheme used for communications that puts a large overhead on small transfers.

Bibliography

- [1] "IEEE 802.3an (10GBASE-T) task force," <http://www.ieee802.org/3/an/>, 2005.
- [2] "Synchronous optical network (SONET) basic description including multiplex structures, rates, and formats," *ANSI T1.105*, 2001.
- [3] N. Soni, N. Richardson, L. Huang, S. Rajgopal, and G. Vlantis, "NPSE: A high-performance network packet search engine," in *Proceedings of the design and test in Europe (DATE) conference*, 2003.
- [4] David Newman, *Internet core router test*. Light Reading online, <http://www.lightreading.com>, 2001.

-
- [5] S. Bradner and J. McQuaid, *Benchmarking Methodology for Network Interconnect Devices*. RFC2544, <http://www.ietf.org/rfc/rfc2544.txt>, 1999.

Chapter 10

WCDMA/FDD Basestation

10.1 A brief introduction to WCDMA/FDD

Currently, the third generation of mobile telephone systems is deployed throughout the world. The standard used in Europe and some other parts of the world is the WCDMA/FDD mode of the 3GPP specification [1]. WCDMA (wideband code-division multiple access) means that different users share a relatively wide spectral band using coding instead of time slots. Each user is allocated a spreading sequence used to smear its narrowband data signal over the broader spectral band. Each user is then differentiated from other users by the chosen spreading sequence which preferably should be orthogonal to the other spreading sequences in use. The narrowband signal can then be recovered at the receiver by the same mechanism.

The spreading code must be generated at a higher rate than the data to be transmitted. This spreading code rate is normally denoted *chip rate* and is 3.84 Mchip/s for WCDMA/FDD. The chip rate is typically 16-256 times higher than the final data rate.

Figure 10.1 shows the schematic function of the spreading for the downlink channels [2]. The data sequence enters the spreading block and is parallelized to two bits, which are mapped into a QPSK (or 16QAM) signal. The outputs from the mapper are multiplied with the channelization

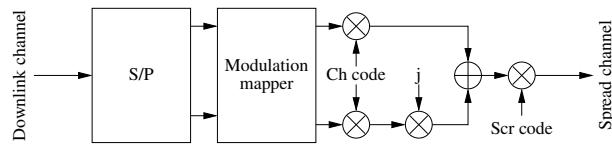


Figure 10.1: Spreading function

code, which is an orthogonal variable spreading factor (OVSF) code.

The sequences are then multiplied by the scrambling code. The scrambling code is also generated at the chip rate. The output can then be summed with other downlink channels and transmitted over the radio.

The channelization codes are used to separate data streams from the same source, which allows a number of downlink channels to be combined before transmission. The scrambling code is unique for the basestation and is used to differentiate this from other basestations in the vicinity.

10.2 Basestation processing flow

10.2.1 Downlink

The downlink processing is relatively straight forward. The basestation will transmit a set of downlink physical channels, typically one for each terminal, plus a set of common channels. The common channels are three synchronization channels, a broadcast channel, a paging channel, and so on. All these common channels are transmitted to all users within the reach of the basestation. Of special interest among these is the synchronization channels that are used for initial cell search where a mobile terminal can find and connect to a basestation [3]. The primary synchronization channel uses a so called generalized hierarchical Golay sequence with excellent autocorrelation properties for easy detection and synchronization.

Figure 10.2 shows the dataflow for user data. The incoming data from the media access control (MAC) layer is made up of two different streams. One is the dedicated transport channel (DTCH) for data and the other

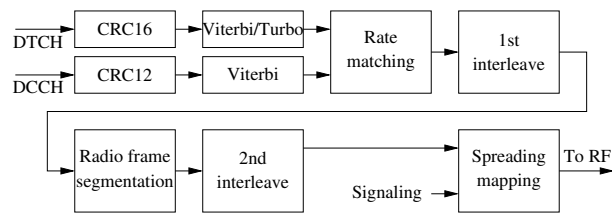


Figure 10.2: Downlink transmission flow

is the dedicated control channel (DCCH). The first step is to add error correction and detection coding to these streams. The streams are then sent to a rate matcher that will make sure that the data rate of the stream is correct for the physical layer. The stream is then interleaved, segmented into slots, and interleaved again. Finally, the stream will be mapped and spread to the chip rate and output to the radio frontend.

10.2.2 Uplink

The uplink processing flow is similar to the downlink flow but involves much more and a different order of computation. Figure 10.3 shows the dataflow for the received data. The first step in the flow is multipath combination. This step is based on a multipath search filter and a Rake receiver in cooperation. Since each terminal will experience different multipath propagation conditions, this will require the use of one multipath combiner per terminal in the basestation. The Rake receiver will do energy combination and despreading of the incoming signal.

The despread stream is then sent through deinterleaving followed by reverse rate matching. The stream is then sent through radio frame re-assembly and a second deinterleaver before the forward error correction coding is used to restore the received data, which can then be sent to the MAC layer.

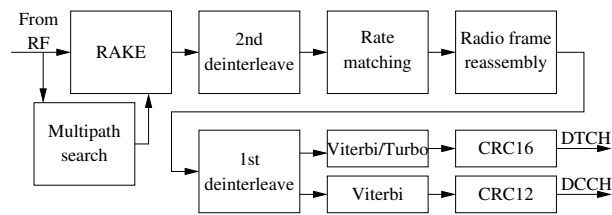


Figure 10.3: Uplink reception flow

10.3 High-level design specification

The case study presented here concerns the baseband part of a 128-channel radio basestation for WCDMA/FDD. The case study is limited to a specific traffic case in the basestation. The basic assumption is that 128 data channels each transporting 384 kbit/s is used in both uplink and downlink. No regard is given to changes in this situation that would occur if a new user connects or a channel wants to change the transfer parameters. The input at the radio is assumed to be samples, ready for multipath search. This assumes that the radio interface is responsible for gain control and such, although this may well be based on information from later stages of the flow.

10.4 Function mapping

The basestation dataflow implementation is based on a set of functional subsystems that each perform a part of the flow. These subsystems are listed in table 10.1. Some functionality has been grouped, e.g. the rate matcher and one interleaver. This is possible since these functions are each very simple and that a net gain in cost can be found when the functions share the memory. The subsystem selection is based on an analysis of the processing and memory requirements of the basestation and is beyond the scope of this thesis.

The architectural mapping is found in figure 10.4. The architecture

Table 10.1: Block types

Abbr	Subsystem function
RI	Radio interface
SM	Spreading and modulation
Rake	Rake receiver and despreader
IRM	Interleaver and rate matcher
RFRI	Radio frame reassembly and interleaver
VTC	Viterbi/Turbo encoder
VTD	Viterbi/Turbo decoder
CRC	CRC calculator and checker
Mac	Mac layer interface

is based on a mesh network of size 8x8 routers. The network has been optimized somewhat to get a lower overall cost with only 52 routers. Further optimizations are possible since the top and bottom rows (with two routers each) are superfluous for the functionality since the blocks can be attached to the nearest full row. Also, some of the links in other parts of the network are unused and can be removed from the design. An optimal network is not the main design target and this is therefore ignored.

The network considered uses only the first type of router (section 5.9.1) because the extra functionality of the latter routers are not necessary. Some routers have less than five ports, which will give a smaller contribution to the total area than 0.06 mm^2 . Thus, with the selected network topology, the routers will occupy a chip area of less than $52 \cdot 0.06 = 3.12 \text{ mm}^2$ plus wiring.

The functional allocation follows the dataflow with the radio interfaces on the top edge of the network to the MAC interfaces at the bottom edge. Uplink data goes from top to bottom, and vice versa for the downlink data. The allocation of subsystems to network ports is done to minimize the network distance between communicating subsystems.

A real basestation would need a system controller to be present. This is not included in the described allocation but can be connected more or

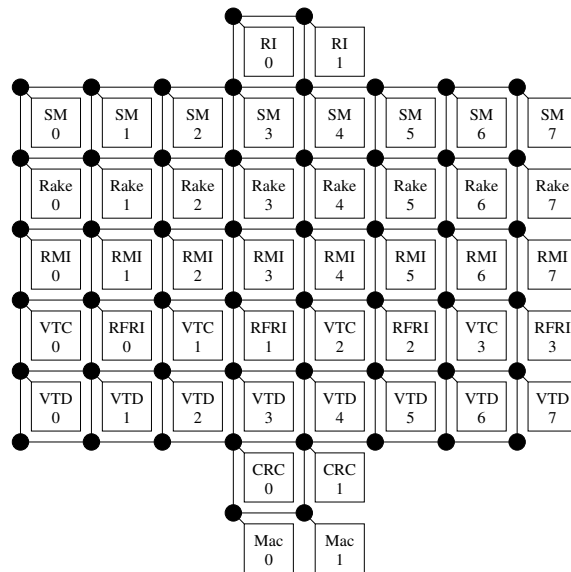


Figure 10.4: Architectural mapping of the basestation

less anywhere in the network.

10.5 Processing subsystems

10.5.1 Multipath search and Rake

The multipath search is by far the most processing intensive task in the flow. The longest path delay (from the ITU pedestrian B model) is $3.7 \mu\text{s}$, or about 60 samples. Each channel must then be correlated over 60 samples to get the channel estimate. Each block of samples will then consume about 46000 complex multiply and accumulate (CMAC) operations where the coefficients are $\pm 1 \pm j$ per sample block and channel. Not all blocks have to be used for multipath search once the initial multipath search has been done. This will lower the requirements to one tenth (i.e. using only one uplink pilot bit) of the otherwise preventively high processing requirements.

After the multipath search is complete, Rake combining can then be done using simple techniques, e.g. the FlexRake architecture [4].

The multipath search will require the use of multiple CMAC units in order to meet the processing deadlines. The total processing requirement will be on the order of 1500 CMAC/ μ s or the equivalent of eight units running at 200 MHz. With somewhat more resources at hand it will be possible to acquire initial synchronization for the users one at a time.

10.5.2 Deinterleaving and rate matching

Deinterleaving of data is a trivial operation in the 3G standard. Data is simply written in order to a memory and then read from the memory using a permutation of row and column order.

The rate matching is somewhat more involved since it uses bit repetition and puncturing. Thus, the correct bits have to be removed from the data stream so that the remaining bits can be sent to the radio frame reassembly.

10.5.3 Radio-frame reassembly and deinterleaving

The radio-frame reassembly step is more or less just concatenation of data, which is a trivial operation. The interleaving is done in the same fashion as the first interleaving described in section 10.5.2.

10.5.4 Viterbi and Turbo decoding

The Viterbi and Turbo decoding is the second most processing-intensive task in the receiver. The control flow goes through a convolutional coder with rate 1/3 (or 1/2) and constraint length 9 in the terminal. The total data rate for the control channels is low which gives a low processing complexity for the Viterbi decoder.

Data is encoded with rate 1/3 using two 8-state constituent encoders. The total incoming data rate to each Turbo decoder subsystem is about 6.2 Mbit/s. This target can be reached using ASIC-style hardware [5].

10.5.5 CRC checking

CRC checking in the receiver path is divided into two different paths where the control uses 12-bit CRC and the data uses 16-bit CRC. With the compound data rate of about 25 MBit/s per direction (i.e. 64 channels times 384 kbit/s) the CRC checking must be considered a simple task. No special ASIC-style hardware is necessary since this can be performed with a suitable processor.

10.6 Processing and communication scheduling

The maximum latency for processing a 3G slot is on the order of a few slot times. A maximum latency of two slot times (1333 μ s) is assumed for the schedule in this case study. Therefore, the communication and processing schedule must be relatively tight to meet the deadline.

The downlink processing is practically trivial compared to the uplink processing and will not be discussed further here although it has been considered in the scheduling of the system. The uplink processing is more interesting since all heavy processing are done in that part. A nice property of the 3G system is that the received uplink channels must be (almost) in synchronization with each other, simplifying the task of handling several simultaneous channels.

There are four distinct subschedules for the receiver depending on which specific instances of the subsystems that are involved in the processing. Figure 10.5 shows the worst path schedule for the basestation. The transmissions are shown as grey boxes and processing as white boxes. The annotations are the transmission size in words (i.e. 16 bits) and maximum processing time in microseconds. The gaps in the schedule is due to synchronization delays between the communicating subsystems and the different schedules for these including the downlink processing.

The schedule shows the processing of the data for a single time slot. Combining the four individual schedules and folding these over a slot time will give the full picture of the schedule.

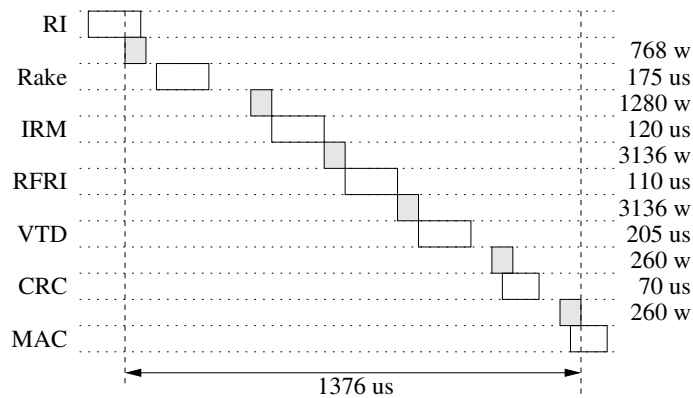


Figure 10.5: Worst-case uplink processing schedule (not to scale)

The critical chain of communication in total is 8840 words plus overhead, which gives less than 10000 network clock cycles for the communication. With a network running at 100 MHz this will give a total delay on the order of 100 μs . Simulation results later showed that the minimum acceptable network frequency of 75 MHz and maximum overhead of 41 cycles per communication yielded about 122 μs total communication delay, which confirmed this assumption. With maximum two time slot delays in the basestation, this will give about 1200 μs for processing assuming no flow synchronization overhead.

10.6.1 Schedule analysis

The first step in the schedule is to send a set of samples from the radio interfaces to the Rake/despreader blocks every 50 μs . With an oversampling factor of four, this will be 768 samples per period. The reason for selecting a period of 50 μs is that the radio interface will require significantly less memory and that the Rake/despreader blocks can work continuously during reception of a slot without getting excessively large sets of samples at long intervals. Also the radio interfaces need to send the samples to four Rake blocks which adds about 400 μs to the latency for

Table 10.2: Longest possible processing times in the reception flow

Abbr	Subsystem	Max time
RI	Radio interface	n/a
Rake	Rake receiver and despreader	175 μ s
IRM	Interleaver and rate matcher	120 μ s
RFRI	Radio frame reassembly and interleaver	110 μ s
VTD	Viterbi/Turbo decoder	205 μ s
CRC	CRC calculator and checker	70 μ s
Mac	Mac layer interface	n/a

the last receiving Rake block.

The schedule continues with the rest of the processing subsystems and their associated communication. The maximum allowable processing latency for each block in the schedule is given in table 10.2. The shortest schedule with low synchronization overhead is approx 907 μ s long while the longest schedule is 1376 μ s.

10.6.2 Latency-induced storage

Extra memory requirements are added due to the additional latency incurred from the schedule. This latency is not part of the network latency but rather related to the processing schedule for the subsystems. Whenever a subsystem is occupied by computation, it is also considered unable to receive incoming data.

The maximum extra buffer memory requirements consist of a single buffer for each communication, in total about 100 kilobytes for all reception flows. This requirement can be relaxed somewhat by tweaking the schedule to get tighter communication. With the simulated schedule this has been lowered to about 25 kilobytes in total. The extra buffer memory can either be placed within the subsystems or in the wrappers.

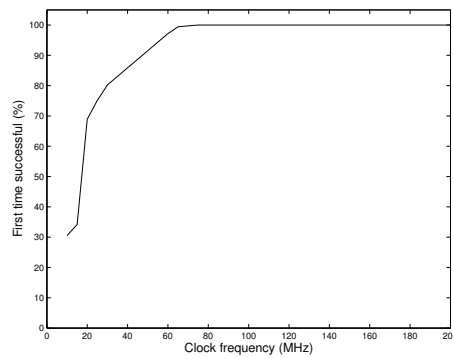


Figure 10.6: First try successful routing

10.7 Simulation results

10.7.1 Minimum network frequency

The basestation model has been implemented using the fixed scheduling for the communication as described above. Simulation results show perfect adherence to the schedule with the SoCBUS running at 75 MHz. The number of first-try successful routings are shown in figure 10.6. The reason for the dropoff in first-try successful routing is that the bandwidth to the radio interface ports are too low for the lower frequencies.

Figure 10.7 shows the maximum circuit setup latency as a function of clock frequency. The average setup latency is about 65% of this value. The full latency of a transmission is the setup latency plus data transmission time which is one cycle per word plus one cycle per hop. The longest path is five hops in this case. The maximum time for the longest transmission in the system is thus expressed by equation 10.1.

$$t = t_{setup} + \frac{n_{words} + n_{hops}}{f_{clk}} = 0.5 + \frac{3136 + 5}{75} \approx 42.4 \mu s \quad (10.1)$$

This time is an overestimate since the longest setup time is not associated with the longest transmission for this application.

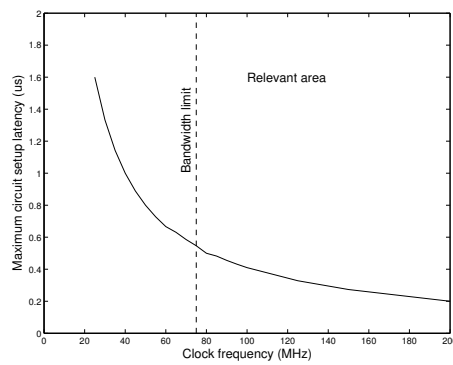


Figure 10.7: Maximum circuit-setup latency

10.7.2 Network usage

The network usage on average is relatively low. The by far most used part is the radio interfaces where all samples have to be multicast to the Rake blocks giving a total transfer rate of 61.44 million words per second for these source ports.

The other parts of the network are significantly underused with a maximum transfer rate of slightly less than 12 million words per second. Changing the radio interfaces so that the requirements in that part are lowered below this will yield a significantly lower minimum network clock frequency. The estimated lower limit on clock frequency given from other parts of the network is about 15 MHz.

It must be considered that the data rates in the basestation are relatively low with only about 50 Mbit/s effective payload rate. The maximum between stages is (of course) significantly higher, considering convolutional coding and soft decision bits. With rate 1/3 coding and four soft bits per data bit, the maximum rate between two stages after de-spreading is about 600 Mbit/s. Simulations have shown that the total transferred data is slightly less than 27 million words during 80 ms, corresponding to about 5.4 Gbits/s.

10.7.3 Control messages vs. transmission schedule

The overhead from the network is limited to the circuit setup time. Thus, most parts of the network have plenty of free time where control packets can be transferred to the different subsystems from a central controller. Even the radio interface is available for control messages since the path into the radio interface has a low utilization.

10.8 Conclusions

This chapter has introduced a case study based on a single-chip WCDMA/FDD basestation which is feasible to implement. The case study has shown the appropriateness of the SoCBUS network for this kind of scheduled hard real-time application. The SoCBUS network is capable of delivering the necessary performance at 75 MHz, which is significantly lower than the maximum achievable speed for the network. Increasing the clock frequency somewhat will directly render gaps in the communication schedule. These gaps can be used for system control related transmissions.

The limiting factors for further channels in the basestation is rather the complexity of processing in the multipath search and Turbo decoding stages. The complicated control flow necessary for changes in the basestation dataflow is also a complicating matter that may well challenge the system design.

Bibliography

- [1] 3rd Generation Partnership Project, *3GPP web site*. <http://www.3gpp.org>, 2005.
- [2] 3rd Generation Partnership Project (3GPP), *TS25.213 – Spreading and modulation (FDD)*. <http://www.3gpp.org>, 2003.

-
- [3] 3rd Generation Partnership Project (3GPP), *TS25.214 – Physical layer procedures (FDD)*. <http://www.3gpp.org>, 2003.
- [4] Lasse Harju, Mika Kuulusa, and Jari Nurmi, “Flexible implementation of a WCDMA Rake receiver,” in *Proceedings of the IEEE Workshop on Signal Processing Systems (SIPS)*, Oct. 2002.
- [5] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, Gongyu Zhou, L. M. Davis, G. Woodward, C. Nicol, and Ran-Hong Yan, “A unified turbo/viterbi channel decoder for 3gpp mobile wireless in 0.18- μ m cmos,” *IEEE Journal of Solid-State Circuits*, pp. 1555–1564, Nov. 2002.

Part VI

Conclusions and Future Work

Chapter 11

Conclusions

Acti labores jucundi

Cicero

The requirements on communication within a system on chip follow the ever-increasing requirements on processing power closely. The communication infrastructure can no longer be taken for granted as the systems become more complex. Deep submicron effects make the interconnections slow compared to the logic and the design must therefore be limited to smaller blocks that communicate in a controlled fashion. This communication is the focus of the thesis.

11.1 Design of networks on chip

Starting from an analysis of the infrastructural context of contemporary chip designs, a network for on-chip use has been developed. This network is designed with simplicity in mind, giving small, fast, and area-efficient implementations of the network components. A speed-optimized implementation of a 5-port router capable of up to 1.2 GHz operation uses only 0.06 mm^2 in a $0.18 \mu\text{m}$ process.

The link protocol has evolved during design from a purely circuit-switched protocol with packet-based circuit setup into a hybrid solution that allows small packets to be sent without circuit overhead. This hy-

brid solution can get advantages from circuit-based networks, e.g. low latency, and combine these with advantages from the packet-based networks, such as low resource usage.

The necessary design tools have also been developed, consisting (in part) of a stimuli generator, a simulator, and a network generator.

11.2 Performance evaluation

After identifying the need to assess the performance of interconnect structures, a methodology for benchmarking has been developed. The design tools for SocBUS have been used with this methodology to develop some example benchmarks and try these on the SoCBUS network.

11.3 Application case studies

The final part of the work has been on application case studies. Two case studies, an Internet core router and a WCDMA/FDD basestation, show the applicability of SoCBUS in real world applications.

Chapter 12

Future Work

And here I'll go crazy

Prof. Dake Liu, describing a plan of his research in the distant future

12.1 Networks on chip

The main extensions to the network on chip implementation would be the implementation of a set of wrappers for standard interfaces, e.g. Wishbone, OCP, and AMBA. Creation of message-passing-style wrappers would be reasonably straight forward and should be possible to implement. A wrapper for Wishbone will be developed within a continuation of the core router project.

Other potential extensions to the core network could result from further studies of applications and traffic types. There are provisions for further extensions already in the protocol, which should make this possible even with backward compatibility to the current protocol set.

12.2 Extensions to the tool chain

There are always extensions that can be made in the tool chain so this is an obvious area for further work. Possible extensions to the stimuli generator includes support for more statistical models, e.g. Markov chains. A possibility to model more complex dependencies would also be nice.

A tool for communication and processing scheduling would be another very useful addition to the toolchain. This tool should be aware of both task scheduling of the subsystems and transfer scheduling on the network. A possible input could be as simple as an annotated system description using communicating tasks together with a system mapping to an executable system specification.

12.3 System simulator

Extending and changing the simulator to include support for external simulations of the subsystems would turn the simulator into a “super simulator” than can be used to evaluate the entire system. There are two methods to do this, either to use the current simulator kernel as basis for the system simulator or to change the current simulator so that it can be plugged into the system simulator. The latter option seems like the most promising since this will allow the system simulator design to be as open as possible.

12.3.1 System simulator integration

The purpose of the system simulator is act as an umbrella under which the other simulators can coexist within a single system simulation. With such a simulator it is possible to get the full picture of the system design at an early stage. A multitude of different simulators should be supported so that high-level behavioral models on algorithm level can be used in the beginning and then gradually refined until the implementation level is reached.

The single most important part of the system simulator is the interface to plug-in simulators. This API must be capable of connecting anything from MATLAB to RTL simulations. The system simulator should be event driven to allow easy integration of any kind of simulator into the system simulator. This will give the all-important infrastructure necessary to implement simulations using heterogeneous models at different layers of

abstraction.

An attempt to create such a simulator is the Simulation Bridge proposed by Nagendra et al. [1]. The bridge functionality is to supply the individual simulators with clock triggers so that the synchronicity of the simulators can be kept. The reference does not give a clear view on how the integration is done. Fummi et al. [2] have made an effort to integrate instruction-set simulators within the GNU Debugger (GDB) with SystemC models based on the GDB remote debugging interface. This type of integration is nice since it is based on a universally agreed on interface that is supported by all platforms with GDB support.

12.3.2 Cycle and bit true simulations

The system simulator must support both cycle and bit true simulations for the integration to get the maximum accuracy in the results. Bit true communication between subsimulators can easily be achieved through bit-handling communication primitives. Getting the system simulator cycle true is a more tricky problem because of synchronization issues between the subsimulators.

The first step towards accurate synchronization is to give the system simulator a global notion of absolute time that can be translated into clock cycles for the different subsystems. The second step is to keep the simulators synchronized at all times to allow events to be passed between the simulators at the correct time instances. This will allow all events generated in one simulator to be passed to another simulator at the correct time in the simulation.

12.3.3 Challenges

There are a number of challenging areas left to conquer apart from the integration issue. The previously mentioned detailed synchronization of different simulators, the inter-simulator communication formats, and getting the necessary simulation speed are some challenges.

Another challenge is to increase the performance of the instruction-set simulators that generally are too slow today. A new implementation style for these to achieve simulation speeds beyond one million instructions per second would further enhance the simulation infrastructure, making system simulations even more useful for early stage evaluation of system performance.

12.4 Application case studies

There is always room for more case studies or more thorough modeling of the previous case studies to evaluate the network, toolchain, etc. Possible applications include but are not limited to media processing, radio baseband, and network processing. Example applications are transcoding gateways, software-defined radio basestations, and network equipment.

Bibliography

- [1] G. D. Nagendra, V. G. P. Kumar, and B. S. Sheshadri, "Simulation bridge: a framework for multi-processor simulation," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES)*, pp. 49–54, May 2002.
- [2] Franco Fummi, Stefano Martini, Giovanni Perbellini, and Massimo Poncino, "Native iss-systemc integration for the co-simulation of multi-processor soc," in *Proceedings of the design and test in Europe (DATE) conference*, pp. 564–569, Feb. 2004.

Part VII

Appendix

Appendix A

Tool Implementation

Details

A.1 Details on the simulation flow

The simulator in the SoCBUS project is tailored to the task of simulating on-chip networks. The simulator is used in conjunction with an in-house stimuli generator tool according to the simulation flow in figure A.1. The inputs to the flow are the traffic and network models, represented in XML. The XML formats are described in appendix B.

A.1.1 Stimuli generator implementation

The stimuli generator will take a traffic model (in XML), that can use both statistical distributions and deterministic data, as input and generates a deterministic set of stimuli for the simulator. An overview of the program flow of the stimuli generator can be found in figure A.2. The stimuli are generated for each traffic class described in the XML model file. The stimuli are generated iteratively within a class if the class is periodic or described as a statistical process.

The generated stimuli is always based on absolute time stamps (i.e. it is scheduled) so the basic information in the stimuli file is just the source,

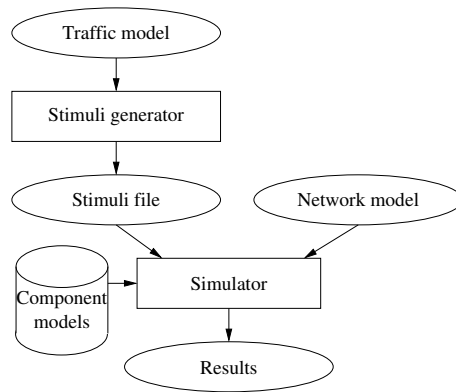


Figure A.1: SoCBUS simulation flow

destination, start time, and size for each transfer. If the stimuli is specified using dependencies this will also be included. The generated stimuli is stored using a simple text format which can easily be read by the simulator. The stimuli generator has been implemented using C++.

A.1.2 Simulator implementation

The simulator will take a network description (in XML) together with a stimuli file to run the simulation. By disconnecting the stimuli from the network description it is easy to run simulations with exactly the same stimuli on a different network setup to see the relative performance of the different networks. It is also easy to use exact copies of the network for different traffic situations to see how the traffic style affects the network performance. Both of these variations are necessary to get a broad view of the network performance and will be discussed later in the paper.

The tool is implemented in C++ to achieve high simulation speeds. Using C++ for the tools also allows extensions to be made to them in this commonly known and efficient programming language. All network component models (routers, links, sources, destinations, etc.) are implemented as compiled-in behavioral models written in the same program-

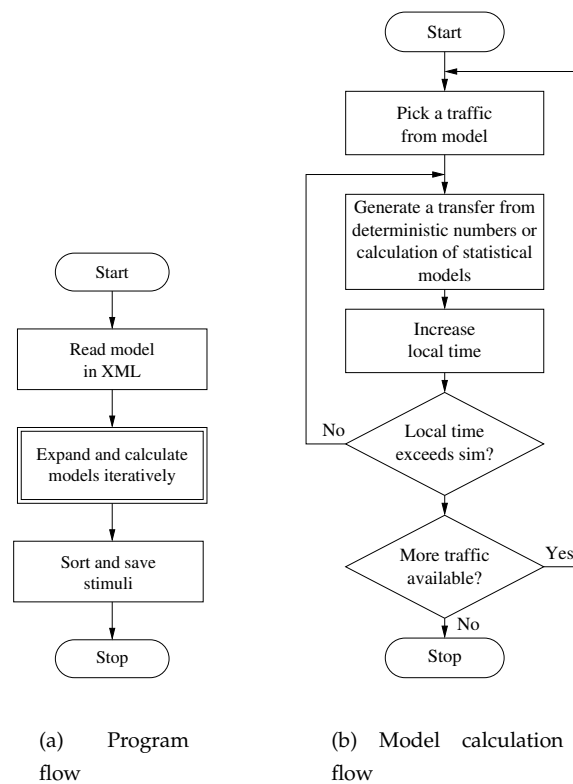


Figure A.2: Stimuli generator program flow chart

ming language as the simulator.

The simulator is event based in a fashion similar to a typical VHDL simulator. Every time a network component generates an event a message is created. These event messages are distributed to the proper destinations through a message server. The central message server handles all time ordering of events/messages through a message queue. This guarantees that the events will happen in the correct order independent of the order in which they have been created.

An overview of the internal program flow in the simulator can be found in figure A.3(a). The simulator starts by reading a network de-

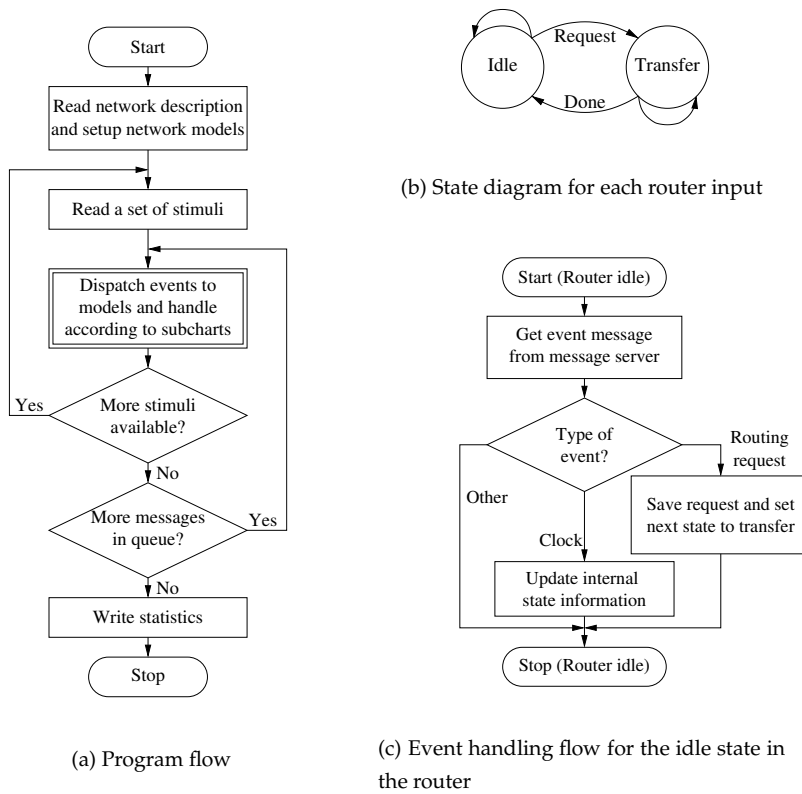


Figure A.3: Simulator flow chart with partial message handling for a router model

scription in XML. This description is then translated into a network setup using the appropriate network component models within the simulator. The network description contains four distinct parts, sources/destinations, routers, links, and optional routing tables. Each of these parts will cooperate to give the full network picture. If a routing table is not declared for a router (using the possibility given in the network model) a minimal distance based routing table produced by Dijkstra's algorithm will be used [1].

After the network description has been parsed, a chunk of the stimuli

is read¹. This reading of stimuli is basically only a transfer from a file to a central stimuli list in memory. When the stimuli have been read, the event message server is triggered.

The handling of the messages in the models differ dependent on which model is the target of the message. An example of the event handling is shown in figure A.3(c). This is the flowchart for a router input that is in the idle state (see figure A.3(b)). As can be seen in the figure the router only accepts a routing request in the idle state. The state will then change to transfer meaning that the port has been locked. If a routing request can not be granted the negative acknowledgment will be generated in the transfer state. The state will then be changed to idle.

When more stimuli are needed another chunk of stimuli is read and the event message serving continues. This continues until no more stimuli are available and the message queue runs out of messages. At this stage the simulation is finished. The final step is to write the measurements taken during simulation to a statistics log file.

A.2 Simulation models of network components

All network related models that are used in the simulations are based on the implementations that have been made at the register transfer level in Verilog. The first router implementation of the router model uses six cycles for a routing decision and one cycle for transferring data and acknowledgments and is the router that have been most used during simulations.

The links are modeled as a simple discrete time delay of zero or more cycles. The basic IP subsystem is currently also modeled in a rather simple fashion. The IP block and the wrapper are seen as a single entity that will generate and consume transfers directly according to the stimuli file.

Models are easily changed or added if new implementations of components are made or the need to model certain IP blocks in a more thor-

¹The simulator reads the stimuli file in smaller fragments because of memory requirement issues.

ough fashion arises. Multiple models for a single component type are supported and are selected through model names in the network description file. A simple API is used to implement the models in C/C++.

A.3 Tool usage

A.3.1 Stimuli generator

Command line options

```
stimuli [-i input.xml] [-o output.sti] [-y] [-dddddd...]
```

The command line flags for the stimuli generator are:

-i input	Specification of test case XML input
-o output	Specification of output stimuli file
-d	Debug mode (verbose)
-y	Yacc debug mode (very verbose)

A.3.2 Simulator

Command line options

```
sim -i input.xml [-dddddd...]
```

or

```
sim -n input.xml -s input.sti  
-t output.tra -l log.txt [-dddddd...]
```

The command line flags for the simulator are:

-i input	Setup file input (XML) (Not to be used with -n, -s, -t, -l)
-n input	Network model input file (XML) (Not to be used with -i)
-s input	Stimuli input file (Not to be used with -i)
-t output	Trace output file (Not to be used with -i)

-l output Log output file (Not to be used with -i)
-d Debug mode (verbose)
-d also enables trace mode in the simulations

Simulator setup input file

The setup input file is a convenient method to specify all input and output files at once. The setup file is a XML file with the following structure:

```
<SIMCONFIG>
  <NETWORK>networkfile</NETWORK>
  <STIMULI>stimulifile</STIMULI>
  <TRACE>tracefile</TRACE>
  <LOG>logfile</LOG>
</SIMCONFIG>
```

A.3.3 Network generator

Command line options

```
netgen -n input.xml [-s stem] -o outdir [-dddddd...] [-v]
```

The command line flags for the simulator are:

-n input Network model input file (XML)
-s stem Stem for generated file names
-o outdir Output directory
-v Show version
-d Debug mode (verbose)

Function description

The network generator will generate a set of files saved into the directory given by the -o flag. All filenames will be prefixed with the stem given by the -s flag.

Bibliography

- [1] Harry R. Lewis and Larry Denenberg, *Data structures and their algorithms*. Harper Collins Publishers, 1991. ISBN 0-673-39736-X.

Appendix B

XML Formats

B.1 Notation

This chapter uses XML notations extensively. The definitions used for the XML structure uses the basic notation of regular expressions shown in table B.1.

The contents of a XML file is hierarchically built with one top (or outer) tag. All contents within the XML file is contained in this outer tag. All other tags are defined within the context of a mother tag. A tag that is defined within the context of a mother tag is of course dependent on the instantiation of the mother tag.

All XML tags adhere to a certain structure. All tags are containers,

Table B.1: Notation of tags

Form	Description
tag	One occurrence of "tag"
tag?	Zero or one occurrence of "tag"
tag+	One or more occurrences of "tag"
tag*	Any number of occurrences of "tag"
tag1/tag2	Either an occurrence of "tag1" or an occurrence of "tag2"

i.e. they have a start and a stop. A container will have contents (but the contents can be empty). A generic XML tag will look like this:

```
<TAG argument1="value1" argument2="value2">Contents</TAG>
```

A container with empty contents can be written as a single entity that is closed by a trailing slash:

```
<TAG argument1="value1" argument2="value2"/>
```

B.2 General specifications

B.2.1 Model names

All the names for blocks, routers, and links must be unique within the network since these are used for identification purposes in the network. It is not possible to give the same name to, e.g. both a router and a block.

B.2.2 Frequencies, times, and lengths

The XML parser allows the use of suffixes for different quantities. It is possible to use *kHz*, *MHz*, and *GHz* for frequencies. Times can be described using *ps*, *ns*, *us*, *ms*, and *s*. Whitespace is not allowed between the number and suffix. The default suffix for frequencies is kHz. An empty suffix on a time value implies that the value is in clock cycles.

The lengths of packets can be described using *b* for bits, *B* for bytes (8 bits), *W* for words (16 bits), and *L* for longwords (32 bits). Each of these can be prefixed with *k* (10^3) and *M* (10^6). No suffix implies the native word length given by the `<BUSWIDTH/>` tag.

B.3 Traffic model

B.3.1 XML structure

The basic traffic specification describes each traffic by source, destination, starting time, and transfer size. These parameters are enough to model a system where

the communication is (at least partially) scheduled. More complicated or unpredictable traffic flows will require the use of dependencies in the traffic specification.

The traffic model is completely decoupled from the network topology. The mapping of traffic sources and destinations are done using literal names in the model files. Each named source/destination must be present as a named source/destination in the network model. If a source named "Src" has been specified in the traffic model there must also be a wrapper/IP block connection named "Src" in the network model. This allows the allocation of connections between wrappers and routers to be done entirely in the network model.

The XML document describing the traffic model must follow the structure shown in table B.2 and figure B.1. An example of such a XML document is shown in the following listing:

```
<TMODEL>
  <TITLE>Title of model</TITLE>
  <BLOCKS>
    <BLOCK>Block 0</BLOCK>
    <BLOCK>Block 1</BLOCK>
  </BLOCKS>
  <CLOCKRATE>100MHz</CLOCKRATE>
  <SIMULTIME>1ms</SIMULTIME>
  <BUSWIDTH>16</BUSWIDTH>
  <TRAFFIC>
  <TRAFFIC>
    <SOURCE>Block 0</SOURCE>
    <SOURCE>Block 1</SOURCE>
    <DESTINATION>Block 0</DESTINATION>
    <DESTINATION>Block 1</DESTINATION>
    <PERIOD>10000</PERIOD>
  <TASK>
    <WORKING>
      <EVENT_QUANTITY>
        <VALUE>10</VALUE>
      </EVENT_QUANTITY>
      <EVENT_LENGTH>
        <MATH_MODEL>
```

```

    <MODEL_NAME>UNIFORM</MODEL_NAME>
    <PARAM>
      <PARAM_NAME>A</PARAM_NAME>
      <PARAM_VALUE>32</PARAM_VALUE>
    </PARAM>
    <PARAM>
      <PARAM_NAME>B</PARAM_NAME>
      <PARAM_VALUE>1200</PARAM_VALUE>
    </PARAM>
  </MATH_MODEL>
</EVENT_LENGTH>
<EVENT_POSITION>
  <VALUE>1</VALUE>
</EVENT_POSITION>
</WORKING>
</TASK>
</TRAFFIC>
</TMODEL>

```

Table B.2: XML tag description for the traffic model

Tag	Contents	Args
TMODEL	TITLE? BLOCKS CLOCKRATE SIMULTIME BUSWIDTH TRAFFIC+	None
TITLE	The title text	None
BLOCKS	BLOCK+	None
BLOCK	The block name	None
CLOCKRATE	The clock rate	None
SIMULTIME	The simulation time	None
BUSWIDTH	The bus width (in bits)	None
TRAFFIC	SOURCE+ DESTINATION+	None

Tag	Contents	Args
	PERIOD OFFSET? TASK+	
SOURCE	Source name	None
DESTINATION	Destination name	None
PERIOD	Period time	None
OFFSET	Offset time	None
TASK	DEPENDENCY? DURATION IDLE / WORKING	None
DEPENDENCY	Dependency specifier	None
DURATION	Task duration time	None
IDLE	Empty	None
WORKING	EVENT_QUANTITY EVENT_LENGTH EVENT_POSITION	None
EVENT_QUANTITY	VALUE / MATH_MODEL	None
EVENT_LENGTH	VALUE / MATH_MODEL	None
EVENT_POSITION	VALUE / MATH_MODEL	None
VALUE	The value	None
MATH_MODEL	MODEL_NAME PARAM+	None
MODEL_NAME	The model name	None
PARAM	PARAM_NAME PARAM_VALUE / MATH_MODEL	None
PARAM_NAME	Parameter name	None
PARAM_VALUE	Parameter value	None

B.4 Network model

The network model contains the complete description of a network with sources, destinations, routers, and links. The network model also contains a description of the connectivity between these components within the network.

A network model document consists of four parts. The first part defines all the sources/destinations connected to the network. The second part defines all the routers while the third part defines the links and how they are connected between other components. The last part is optional and may contain custom routing tables for the routers.

A network model is described as an XML document according to the specification in this chapter.

B.4.1 XML structure

The XML document describing the network model must follow the structure shown in table B.3 and figure B.2. An example of such a description can be found in the following listing:

```
<NETWORKMODEL>
  <BLOCKS>
    <BLOCK>blockname0</BLOCK>
    <BLOCK>blockname1</BLOCK>
  </BLOCKS>
  <ROUTERS>
    <ROUTER>routename0</ROUTER>
    <ROUTER>routename1</ROUTER>
  <ROUTERS>
  <LINKS>
    <LINK>linkname0
      <PORT>blockname0.0</PORT>
      <PORT>routename0.0</PORT>
    </LINK>
    <LINK>linkname1
      <PORT>blockname1.0</PORT>
      <PORT>routename1.0</PORT>
    </LINK>
    <LINK>linkname2
      <PORT>routename0.1</PORT>
      <PORT>routename1.1</PORT>
    </LINK>
  </LINKS>
```

```

<ROUTINGTABLES>
  <TABLE>routername0
    <ENTRY>blockname0.0</ENTRY>
    <ENTRY>blockname1.1</ENTRY>
  </TABLE>
</ROUTINGTABLES>
</NETWORKMODEL>

```

Table B.3: XML tag description for the network model

Tag	Contents	Args
NETWORKMODEL	BLOCKS ROUTERS LINKS ROUTINGTABLES?	None
BLOCKS	BLOCK+	None
BLOCK	The block name	MODEL="value"
ROUTERS	ROUTER+	None
ROUTER	The router name	MODEL="value"
LINKS	LINK+	None
LINK	PORT PORT	MODEL="value" DELAY="value"
PORT	modelname.portnumber	None
ROUTINGTABLES	TABLE+	None
TABLE	ENTRY+	None
ENTRY	The table entry	None

This example shows a complete network system with two blocks connected to two routers. The routers are also connected to each other and the routing table is specified for one of the routers. All sections in the example are mandatory with the exception of the <ROUTINGTABLES> container.

The routingtable <ENTRY> tag contains one entry in the routing table. The syntax is `destination_name.output_port_number`. If the routing table is not explicitly given it will be filled for minimum path length routing using Dijkstra's algorithm. The routing table example above is equivalent to the table that

would be generated automatically.

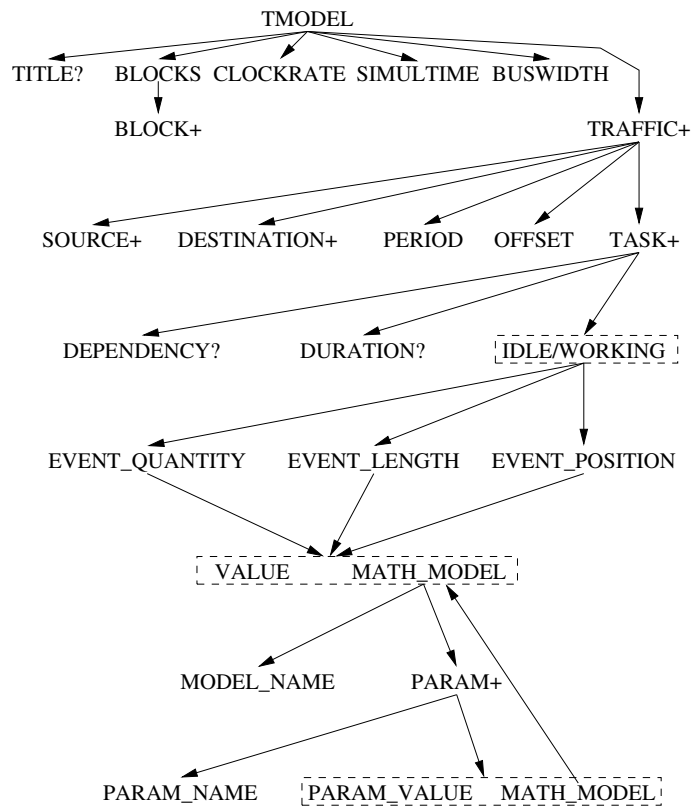


Figure B.1: Tag structure for a traffic model/test case

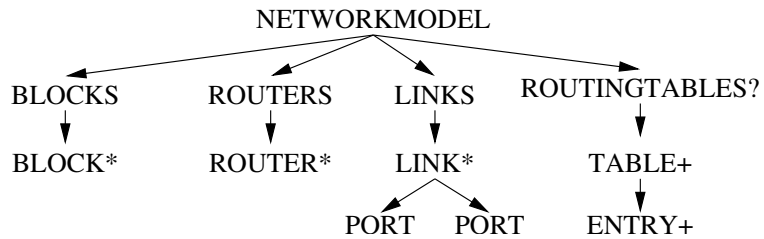


Figure B.2: Tag structure for a network model

Index

- abbreviations, xxiv
- acknowledgments, ix
- Amdahl's law, 11
- applications, 13

- background, 3
- bandwidth, 30
- benchmark, 96
- benchmarking, 81, 101
 - examples, 104
 - method, 102
 - processors, 99, 101
- bottleneck, 96

- chip testing, 34
- circuit, 52
 - virtual, 42, 47
- configuration, 61
- contributions, 8
- core router, 113

- dataflow applications, 13
- deadlock, 50
- design environment, 79
- design flow, 80
- development cost, 4

- Ethernet, 114

- implementation
 - router, 60, 69
 - tools, 149
- Internet mix, 118

- latency, 30, 46, 119, 135

- mapping, 19, 116, 128
- measurement technique, 98
- model
 - network, 161
 - traffic, 158
- Moore's law, 3
- MPI, 16

- network generator, 82, 90, 155
- network model format, 161
- networks, 35, 39

- objective, 5
- OSI model, 41
- overview, 8

- packets
 - long, 66

- short, 69
- parallelization, 12
- performance, 97, 121
- physical format, 60
- platform based design, 13
 - Rabaey's key tenets, 15
- processing flow, 126
- profiling, 17
- protocol
 - link level, 66
 - PCC, 63
 - transaction level, 62
- quality of service, 42
- reliability, 32
- router, 58
- routing, 48
 - distributed, 48, 64
 - nonminimal, 48
 - source, 48, 64
 - turn model, 50
 - wormhole, 47
- scheduling, 19, 132
- scope, 5
- simulation, 81, 117, 135
 - flow, 82
- simulator, 16, 154
 - architecture, 86
 - system, 144
- speculative sending, 68
- stimuli, 85
 - generator, 85
- stimuli generator, 154
- switching, 45
- synchronization, 26
- topology, 43
 - arbitrary, 44
 - mesh, 44, 59
 - torus, 44
- traffic model, 81, 83
- traffic model format, 158
- verification, 33
- virtual channels, 51
- WCDMA, 125
- wormhole routing, 47
- wrapper, 58
- XML, 157