TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Erno Salminen

On Design and Comparison of On-Chip Networks

Julkaisu 872 • Publication 872

Tampere 2010

Tampereen teknillinen yliopisto. Julkaisu 872
Tampere University of Technology. Publication 872

Erno Salminen

On Design and Comparison of On-Chip Networks

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 19th of February 2010, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2010

ISBN 978-952-15-2325-0 (printed)
ISBN 978-952-15-2348-9 (PDF)
ISSN 1459-2045

Erno Salminen

On Design and Comparison of On-Chip Networks

Contact Information:

Erno Salminen

mail: Tampere University of Technology
Department of Computer Systems
P.O.Box 553
FIN-33101 Tampere
Finland

tel: +358–3–3115 4540 (office)
+358–40–5832 182 (mobile)

fax: +358–3–3115 4561

e-mail: erno.salminen@tut.fi

ABSTRACT

This thesis focuses on the design of on-chip communication networks and methods for benchmarking them. Network-on-Chip (NoC) paradigm seeks to achieve greater design productivity and performance in large integrated circuits. Such systems include heterogeneous set of components that have different requirements for communication.

This thesis presents simulation-based evaluation methods for NoCs. In addition, several detailed guidelines are given in order to promote disciplined NoC benchmarking. Discussion starts with thorough surveys of 60 existing NoCs and over 40 evaluation studies. The presented benchmarking methodology relies on abstract workload models based on task graphs. They are executed with a Transaction Generator (TG) that sends and receives data to/from the benchmarked NoC and collects statistics. TG was used in several configurations and was essential part for completing this work. The error in time estimates was mostly below 10% whereas the speedup against cycle-accurate HW/SW co-simulation was over 200x.

Heterogeneous IP Block Interconnection (HIBI) was designed to obtain a topology-independent, scalable, and still high-performance network for integrating intellectual property blocks. Six other NoCs were implemented for reference and benchmarked with HIBI in various configurations and using multiple workloads. Over 30 published implementation results were gathered. Furthermore, several FPGA prototypes were implemented and they confirmed the utility of HIBI in multiprocessor environment. In general, HIBI and 2-D mesh performed better than others in the presented cases considering the trade-off between area and throughput.

The main goals of the work were met. The presented methodology along with TG has been adopted by an OCP-IP workgroup that is seeking to standardize NoC benchmarking methods.

PREFACE

The work presented in this thesis has been carried out in the Department of Computer Systems at Tampere University of Technology during the years 2001-2009.

I thank my supervisor Prof. *Timo D. Hämäläinen* for guiding and encouraging me towards doctoral degree. Grateful acknowledgements go also to Prof. *Kees Goossens* and Dr. *Juha Plosila* for the thorough reviews and the constructive comments on the manuscript.

It has been my pleasure to work in the Department of Computer Systems. Many thanks to all my colleagues for the discussions and pleasant working atmosphere. Especially Dr. *Kimmo Kuusilinna*, Dr. *Vesa Lahtinen*, Dr. *Tero Kangas*, *Jouni Riihimäki*, M.Sc., Dr. *Ari Kulmala*, *Petri Kukkala*, M.Sc., *Heikki Orsila*, M.Sc., and *Kalle Holma*, M.Sc., deserve a big hand for helping me in several matters. In addition, I would like to thank all the other co-authors and colleagues. Our lively discussions both on and off-topic have been pleasant and stimulating.

This thesis was financially supported by Graduate School in Telecommunication System-on-Chip Integration (TELESOC), Nokia Foundation, Ulla Tuominen Foundation, Foundation of Advancement of Technology, which are all gratefully acknowledged.

My warmest thanks go to my beloved wife *Hannele* and our lovely ever-energetic kids *Ilona* and *Otto* for their love, support, and understanding.

"Sitä saa, mitä tulee"

- A Finnish proverb and the guiding principle during this work

Tampere, December 2009

Erno Salminen

TABLE OF CONTENTS

LIST OF PUBLICATIONS

This thesis is a monograph which contains unpublished material but is mainly based on the following publications. In the text, these publications are referred to as [P1], [P2], ..., [P13].

[P1] Erno Salminen, Vesa Lahtinen, Kimmo Kuusilinna, and Timo D. Hämäläinen "Overview of Bus-based System-on-Chip Interconnections," in *IEEE International Symposium on Circuits and Systems*, vol. 2, Scottsdale, AZ, USA, May 2002, pp. 372-375.

[P2] Tero Kangas, Jouni Riihimäki, Erno Salminen, Kimmo Kuusilinna, and Timo D. Hämäläinen "Using a Communication Generator in SoC Architecture Exploration," in *International Symposium on System-on-Chip*, Tampere, Finland, Nov. 2003, pp. 105-108.

[P3] Erno Salminen, Kimmo Kuusilinna and Timo D. Hämäläinen, "Comparison of Hardware IP components for System-on-Chip," in *International Symposium on System-on-Chip*, Tampere, Finland, Nov. 2004, pp. 69-73.

[P4] Erno Salminen, Ari Kulmala and Timo D. Hämäläinen, "HIBI-Based Multiprocessor SoC on FPGA," in *IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 2005, pp. 3351-3354.

[P5] Erno Salminen, Tero Kangas, Timo D. Hämäläinen and Jouni Riihimäki, "Requirements for Network-on-Chip Benchmarking," in *Norchip*, Oulu, Finland, Nov. 2005, pp. 372-375.

[P6] Erno Salminen, Tero Kangas, Jouni Riihimäki, Vesa Lahtinen, Kimmo Kuusilinna, and Timo D. Hämäläinen "HIBI Communication Network for System-on-Chip," *Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology*, Jun. 2006, vol. 43, no. 2-3, pp. 185-205, June 2006.

[P7] Erno Salminen, Tero Kangas, and Timo D. Hämäläinen “The Impact of Communication on the Scalability of the Data-parallel Video Encoder on MP-SoC,” in *International Symposium on System-on-Chip*, Tampere, Finland, Nov. 2006, pp. 191-194.

[P8] Erno Salminen, Tero Kangas, Jouni Riihimäki, Vesa Lahtinen, Kimmo Kuusilinna, and Timo D. Hämäläinen “Benchmarking Mesh and Hierarchical Bus Networks in System-on-Chip Context,” *Journal of Systems Architecture*, Aug. 2007, Vol. 53, Issue 8, pp. 477-488.

[P9] Cristian Grecu, Andrè Ivanov, Partha Pratim Pande, Axel Jantsch, Erno Salminen, Umit Ogras, Radu Marculescu, “Towards Open Network-on-Chip Benchmarks”, in *International Symposium on Networks-on-Chip*, Princeton, NJ, USA, May 2007, pp. 205-205

[P10] Erno Salminen, Ari Kulmala, and Timo D. Hämäläinen “On Network-on-Chip Comparison,”in *Euromicro Conference on Digital System Design*, Lübeck, Germany, Aug. 2007, pp. 503-510.

[P11] Erno Salminen, Ari Kulmala, Timo D. Hämäläinen, "Survey of Network-on-Chip Proposals", white paper, OCP-IP, April 2008, 13 pages.

[P12] Erno Salminen, Cristian Grecu, Timo D. Hämäläinen, Andrè Ivanov, "Network-on-Chip Benchmarking Specifications Part I: Application Modeling and Hardware Description", Version 1.0, OCP-IP, April 4, 2008, 15 pages.

[P13] Erno Salminen, Ari Kulmala, Timo D. Hämäläinen, "On the Credibility of Load-latency Measurement of Network-on-Chips", in *International Symposium on System-on-Chip*, Tampere, Finland, Nov. 2008, pp. 91-97.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACK	Acknowledge
ACM	Association for Computing Machinery
ASIC	Application-Specific Integrated Circuit
BE	Best Effort
BIST	Built-In Self-Test
CAD	Computer-Aided Design
CIF	Common Intermediate Format
CMOS	Complementary Metal Oxide Semiconductor
CMP	Chip Multiprocessor
CPU	Central Processing Unit
DAA	Dynamically Adaptive Arbitration
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DRAM	Dynamic RAM
DSM	Deep-Submicron
DSP	Digital Signal Processing
eCos	Embedded Configurable Operating System
EDA	Electronic Design Automation

EIB	Element Interconnect Bus
FFT	Fast Fourier Transform
FIFO	First-In-First-Out
flit	Flow control digit
FPGA	Field Programmable Gate Array
GALS	Globally Asynchronous, Locally Synchronous
GT	Guaranteed Throughput
HW	Hardware
HIBI	Heterogeneous IP Block Interconnection
IIR	Infinite impulse response
IDCT	Inverse Discrete Cosine Transform
IEEE	the Institute of Electrical and Electronics Engineers
IP	Intellectual Property
ISS	Instruction-Set Simulator
JPEG	Joint Picture Experts Group
KPN	Kahn Process Network
LSB	Least Significant Bit
LUT	Look-Up Table
MCA	Multicriteria Analysis
MoC	Model of Computation
MPI	Message Passing Interface
MPEG	Motion Picture Experts Group
MPSoC	Multiprocessor System-on-Chip

MSB	Most Significant Bit
NACK	Negative Acknowledge
NI	Network Interface
NoC	Network-on-Chip
OCP	Open Core Protocol
OCP-IP	Open Core Protocol International Partnership
OFDM	Orthogonal Frequency Division Multiplexing
P2P	Point-to-point
PC	Personal Computer
PE	Processing Element
QCIF	Quarter CIF
QoS	Quality-of-Service
RC	Resistive-capacitive
R&D	Research and Development
ROM	Read-Only Memory
RAM	Random Access Memory
RM	Resource Manager
RTL	Register Transfer Level
RISC	Reduced Instruction-Set Computer
RTOS	Real-Time Operating System
SI	(French) le Système International d'unités, The International System of Units
SoC	System-on-Chip

SDRAM	Synchronous DRAM
SPMD	Single Program, Multiple Data
SRAM	Static RAM
SW	Software
TDMA	Time Division Multiplexed Access
TG	Transaction Generator
TLM	Transaction-Level Model
UML	Unified Modeling Language
VC	Virtual Channel
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integration
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
XSM	XML System Model

1. INTRODUCTION

The complexity of integrated chips has increased in an exponential rate for the past decades. This empirical observation is known as *Moore's law* [167]. The term complexity is commonly interpreted as a number of transistors but, historically, also the frequency has scaled in a similar manner. This is illustrated in Fig. 1. which plots the data collected from [49, 91]. The computation performance has also increased exponentially and that has been due to three facts: increased transistor budget, increased frequency, and architectural innovations (such as cache memories, pipelining, and instruction level parallelism). A system-on-chip (SoC) integrates a large number of components into a single chip. Usually, the components are not homogeneous but vary in both size and type, for example they can be processors, memories, hardwired

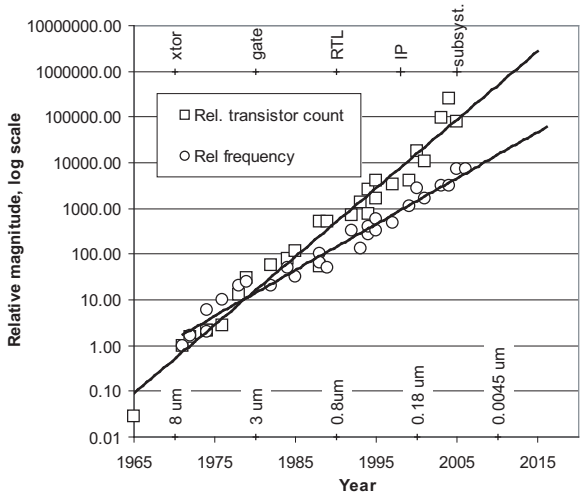


Fig. 1. *Historical scaling of transistor count and operating frequency in integrated circuits. Furthermore, feature size and design abstraction level are illustrated along the time axis.*

components, and interconnected with a network-on-chip (NoC). This Thesis concentrates on the design and benchmarking of NoCs.

The growth of transistor counts is estimated to continue for several years possibly at a slightly reduced rate, though [30, 92, 168]. Wire delays, clock skew together with finite rise and fall times of the clock will limit further frequency scaling [82,232]. Parallel processing is commonly utilized in scientific processing that requires high performance [49] and it is utilized to meet the strict computation requirements of modern user applications, such as real-time video encoding [98]. An integrated parallel processing device is referred to as a chip-level multiprocessing (CMP) platform [30] or multiprocessor system-on-chip (MPSoC) [97, 249]. Few recent general-purpose examples are Intel Dual and Quad Core processors [70,202], IBM Cell processor [100], and Intel TeraFLOPS [236]. The results of this thesis have been utilized in MPSoCs on FPGA for MPEG-4 encoding [128] and for wireless networking [124].

Component reuse offers a great improvement in system design and it is necessary to handle the complexity [37]. Sylvester and Keutzer have proposed block-based hierarchical design style where the size of the used hardware block is in the range of $50\,000 - 100\,000$ equivalent gates (or $50k - 100k\ gates$) [232]. Such blocks can be designed and verified without excessive attention on various deep-submicron (DSM) effects, such as wire delays. Direct consequence is that the number of components will increase as the chip complexity increases. Large component count emphasizes the impact of communication on the overall performance and cost of SoC. Note however that system integrators nowadays work with larger entities, subsystems that already include several intellectual property (IP) components in order to implement complex functions.

Automated design tools are necessary for handling the complexity of a large MPSoC. *Koski* design flow is one example which includes high-level design and requirements capture in UML, automated design space exploration, code generation and hardware synthesis, FPGA prototyping, and back-annotation of measured values. Despite author's contribution to Koski, detailed discussion is beyond the scope of this thesis and interested readers are referred to [103, 104].

Koski follows platform-based design methodology [112, 217]. The basic principle is a sequence of refinement steps that go from the initial specification towards the final implementation using platforms at various level of abstraction. One pillar of the

methodology is the separation (or orthogonalization) between:

- function (what the system is supposed to do) and architecture (how it does it);
- communication and computation.

Separation of concerns and clear, unambiguous interfaces allow the designer to concentrate only on one part of the system without modifying the other part. For example, the communication network can be designed, modified, and optimized while keeping the computation resources fixed. Separation between communication and computation, allows also interleaving their execution to maximize the performance.

Traditionally, SoCs utilize a single shared bus or a bus hierarchy for interconnecting the IP components. In the late 1990's and early 2000's, many researchers proposed replacing dedicated, design-specific wires with a general purpose, packet-switched network, hence marking the beginning of the network-on-chip (NoC) era. The major goal of NoC is to achieve greater design productivity by handling the increasing parallelism, manufacturing complexity, wiring problems, uncertainty, and by introducing communication-centric design methodologies.

NoCs have been widely reported in several special issues in journals, numerous special sessions on conferences and recently also in a dedicated NoC symposium [1]. As an example of increased interest, Fig. 2 shows the hit count for the search "network-on-chip" in the IEEE Xplore document archive [2].

The key research areas in network-on-chip design are summarized in [26, 180] as

- Communication infrastructure: topology and link optimization, buffer sizing, floorplanning, clock domains, power
- Communication paradigm: routing, switching, flow control, quality-of-service, network interfaces
- Benchmarking and traffic characterization for design- and runtime optimization
- Application mapping: task mapping/scheduling and IP component mapping.

[1] www.nocs.org

[2] http://ieeexplore.ieee.org

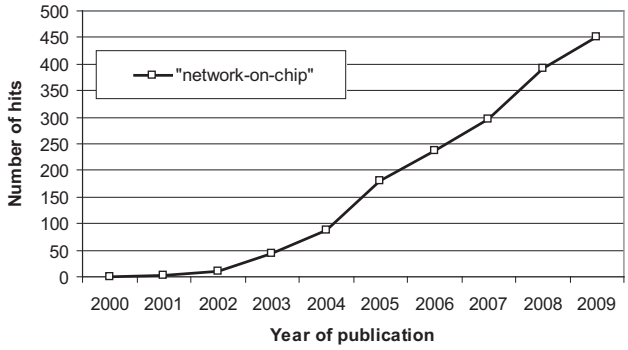


Fig. 2. *Search hits for "network-on-chip" in IEEE Xplore archive.*

The three critical challenges for NoC according to Owens *et al.* are power, latency, and CAD compatibility [184].

1.1 Objective and scope of research

The objective of this work was to design an efficient communication method for IP components in a System-on-chip.

This thesis presents an extensive survey of network-on-chips, a new proposal for benchmarking them, and a detailed comparison of various networks. In addition, a new NoC, HIBI, is presented and analyzed. A set of reference network implementations is implemented for comparison purposes. An analysis follows the conceived guidelines and considers HIBI and reference NoCs. Simulation and FPGA execution are emphasized. The results of the comparison are discussed relative to those from the literature.

This thesis concentrates on digital components only and abstraction is mostly at the register transfer level (RTL). The studied networks are targeted for interconnecting intellectual property components, such as microprocessor cores, memories, and hardwired accelerators.

To summarize, the main contributions of this thesis are the following:

- Surveys of existing NoCs and evalutation studies are presented

- Development of the HIBI NoC
- Systematic method for comparing NoCs
  - synthesizable reference implementation for various topologies
  - versatile traffic generation for networks
  - benchmarking guidelines
  - standardization work on benchmarking
- Comprehensive comparison of NoCs using the above method

1.2 Thesis outline

The outline of this thesis as follows. At first, Chapter 2 introduce the basic concepts of communication networks and Chapter 3 presents a literature study and analysis of existing network-on-chip proposals and their comparisons. In addition, a survey of IP components is given. Chapter 4 presents the basics of communication network evaluation and Chapter 5 presents a new benchmarking methodology for NoCs. Chapter 6 discusses the latency measurement of NoCs in detail. Chapter 7 presents the developed HIBI network and Chapter 8 the reference NoCs. The results of NoC comparison are given in Chapters 9 and 10. Chapter 11 concludes the Thesis.

2. COMMUNICATION NETWORKS

Due to some inconsistencies found in literature let us start by defining the terms that will be used in this thesis. After that the basic network structures and components are introduced, followed by a discussion about topology-independent issues in intra-SoC communication.

2.1 Basic properties and terminology

This terminology follows mostly the definitions by Dally and Towles [51]. Synonyms that are used in the literature are shown in parentheses. A *communication network* (*switching fabric*) is a system that transports data between its *terminals*, i.e. its input and output ports. A specific *network-on-chip*, NoC, paradigm will be discussed later in its own section. A communicating entity is here called an *agent* that may consist of one or multiple processing elements (PEs), memory banks, and a *network interface* (NI). The number of agents is here denoted as N_{ag}. Agents perform *transactions* by exchanging *messages*. A split transaction performs bidirectional communication with two separately transferred messages, such as a read request and a response. Each message can create one or more *packets* depending on the parameters of the communication network. Packets have a restricted maximum length unlike the messages. The data injected to the network are referred to as the *traffic load*. The load is characterized by its spatial (where) and temporal (when) distributions.

The network interface is responsible of splitting messages into packets at the source and combining, and possibly reordering, them at the destination. Agents that can initiate transfers are called *masters* and agents only responding are called *slaves*. Such distinction is not always made when some or all the agents operate in both modes.

A *non-blocking* network allows simultaneous transfers between all source-destination

pairs provided that the destination is free and is not the destination of any other transfers. *Blocking* networks have more restrictions on simultaneous transfers, for example, some link along the path may be blocked although the destination is free.

2.1.1 Structural properties

The *topology* defines the arrangement of the network's *routers* (*switches*, *network nodes*) and *links* (*channels*). The topologies are divided, somewhat superficially, into two categories: *direct* and *indirect*. The routers in a direct network are always connected to (at least one) agent; they all are network *terminals*. In an indirect network, there are routers that are connected only to other routers. Each router has two or more *ports* and their number is here referred to as router's *degree* (or *radix*). In this thesis, all links and ports are unidirectional and the degree defines the sum of input and output ports in a router. A bidirectional link is formed from two unidirectional links. The term router is not commonly used with bus networks. Instead, the connection between the resource and the network is done by a *bus wrapper* or *bridge*.

A link contains the data and control bits going *downstream* (direction from source to destination) and also the flow control bits going *upstream*. All these bits are considered to form a *wire bundle* to avoid confusion with the bus topology. A packet may require several *hops* to reach the destination; one hop denoting here the traversal of one link. Networks that do not have any routers, for example point-to-point, have just one hop.

2.1.2 Temporal properties

Networks are divided into two categories with respect to their connection type: *circuit-switched* and *packet-switched*. Circuit-switching forms a path from the source to the destination prior to the transfer by reserving the routers (switches) and links. All data follow that route and the path is torn down after the transfer has completed. This guarantees the latency and attained throughput. However, the time required for setting up the circuit may be long and the bounds for setup time hard to define. In the packet-switching scheme, the source creates and sends packets to the network and each packet is routed separately to the destination. This Thesis uses a terminological

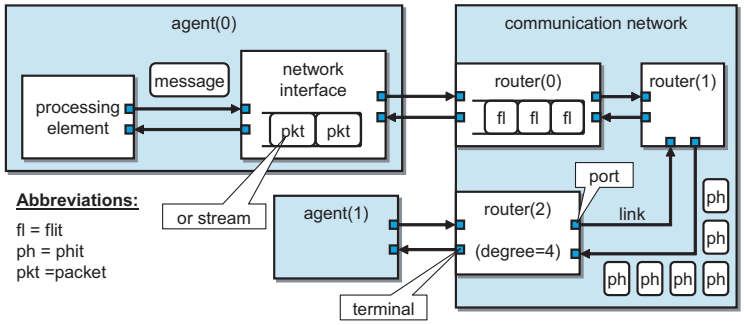


Fig. 3. *Simple network example and corresponding terminology.*

convention that circuit-switched networks include switches whereas packet-switched networks include routers.

Contention (*congestion*) for the shared resources of the network, such as router ports or links, introduces varying latencies for packets. *Arbitration* is the method to resolve the contention.

Routing is the mechanism that determines the *path* (*route*), the set of links, taken between the source and destination. With *deterministic* routing, all packets between the source-destination pair always follow the same route and hence remain in-order. *Adaptive* routing tries to avoid the congested regions, *hot-spots*, by varying the route. Hence, packets may occasionally arrive to the destination out-of-order. *Oblivious* routing, a subclass of adaptive algorithms, does not consider the state of the network when determining the route, for example, the route is selected pseudo-randomly for each packet. Some adaptive routing schemes allow *misrouting* which means that packets are routed away from the minimal route at some points.

Flow control manages the reservation of network resources, mainly router buffers and links. Each packet consists of one or multiple *flow control digits*, *flits*, that is the smallest unit of data to which resources can be allocated. A packet can be divided into *header*, *payload*, and *tail flits*. The header provides control data for the network and determines at least the destination, and optionally other control information, such as packet length or priority. It is followed by the payload, or *body*, that is the actual data. Tail denotes the end of the packet and may provide information for checking the integrity of the data. The length of these parts varies between networks and the payload and tail may be omitted in some cases. If they are present, they must follow

the same route as the header and remain in order. There are also cases where the header and the payload are transmitted in parallel within one flit [1] as well as cases where the length of the payload varies between packets. The data is transmitted on channels one *physical digit*, *phit*, at a time. A phit does not necessarily have the same size as a flit. For example, the buffers are allocated for 32-bit flits but serial links use 1-bit phits.

Fig. 3 depicts an example of the terms. The shown network is indirect since *router(1)* is not a terminal. The network has 4 unidirectional links between the routers and all routers have a degree of 4. The message sent by $PE(0)$ is split into two network packets and each packet contains three flits. Here, the phits are smaller than flits, and hence, each packet contains six phits. In the remainder of this thesis, however, the flit and phit sizes are always equal and that number of bits referred to as the *data width*. The hop count N_{hops} from *agent(0)* to *agent(1)* is 2 hops: *router(0)* $\rightarrow$ *router(1)* and *router(1)* $\rightarrow$ *router(2)*. The number of routers in that path is $N_{hops} + 1 = 3$.

2.1.3 Performance measures

In this thesis, the term *performance* refers to metrics that should be maximized, such as throughput and error tolerance. *Cost* metrics, such as application runtime, area, latency, and power consumption, should be minimized. The performance metrics are also discussed later, for example, in Sections 3.2.2 and 4.4.6.

Throughput, or *accepted traffic load*, is the maximum data rate that can be delivered to destination terminals with given load conditions, for example $Bytes/s$. It is the sum of data rates at destination ports [2]. In this thesis, the term *bandwidth* refers to theoretical maximum accepted load and throughput to realized accepted load. Some sources use these terms interchangeably, though. *Saturation* occurs when increasing offered load does not increase the accepted load; it remains constant and sometimes is may even drop. As the offered load increases beyond the saturation point, the average transfer latency experienced by the agents (end-to-end latency) approaches infinity.

Scalability in general means that something is capable of being easily expanded or upgraded on demand. Scalability is often hard to prove and the term is often used

[1] This scheme is traditionally used in bus topologies although they are not generally considered as packet-switched networks.

[2] Measuring at input terminals is misleading if network drops packets.

too carelessly. For parallel applications, there is some consensus that as the size of a scalable machine increases, a corresponding increase in performance is obtained. Scalable network architecture allows adding/removing components with modest effort, and the performance should increase/decrease correspondingly.

Bisection is a set of channels (*cut*) that partitions the network into two halves such that the number of routers of both halves differ at most by one. *Bisection bandwidth* is the bandwidth of the smallest such cut. Bisection is one scalability metric of the network. If bisection grows slower than N_{ag} or is constant, it will become a system bottleneck as network grows unless data is localized efficiently.

Latency refers to the time between the first bit entering the network at the source terminal and the last bit arriving at destination terminal. It is measured in seconds or in clock cycles but the latter option is not always possible, for example with asynchronous networks. Usually the maximum and average of all packet latencies are monitored. Another important aspect is the difference between latencies which is called *jitter*. Latency can be measured on multiple levels, for example for transactions, messages, packets, or packet headers.

Latency of a packet can be defined as

$$T_{pkt} = T_{hdr} + T_{ser} + T_{cont}. \tag{1}$$

where T_{hdr} denotes header latency, T_{ser} serialization latency, and T_{cont} contention overhead [51]. Header latency T_{hdr} is the time required for the head of the message to reach the destination:

$$T_{hdr} = (N_{hops} + 1) \cdot T_{router} + N_{hops} \cdot T_{link}. \tag{2}$$

T_{router} is the time needed per router without any contention and T_{link} is the propagation delay of a link. Note that there is one router more than there are links in a path. The message must be serialized on the links so that $T_{ser} = N_{msg\_bits}/b_{link}$, where N_{msg_bits} denotes message size in bits and b_{link} denotes the link bandwidth. The last term is caused by contention and is generally hardest to analyze without simulation. In most cases, the measurement setup must ensure that the traffic sources will not be stalled during measurement.

Fault tolerance describes the ability of the network to perform in the presence of one or more faults. It has been recognized for long a time in large networks such as

telephone and internet, but it is also emerging into the NoC domain. A fault in an integrated chip cannot always be repaired after fabrication. However, the faulty parts can be avoided once detected, for example by routing the data via a fully functional path. Deep-submicron technologies also increase the number of transient errors (soft errors) [92] and emphasize the importance of fault tolerance. There are five key elements to tolerate faults: avoidance, detection, containment, isolation, and recovery. Fault-tolerance can be achieved via error detection and correction, stochastic communication, adaptive routing, and both temporal and spatial redundancy [69].

Quality-of-service (QoS) refers to the levels of guarantees given for data transfers. Guarantees can be related to timing (min. throughput, max. latency, max. latency jitter), integrity (max. error rate, max. packet loss), and packet delivery (in-order or out-of-order). Most contemporary articles concentrate on timing aspects. Transfers are categorized as

a) *Best effort* (BE) scheme forwards packets as soon as possible but no guarantees are given for latency or throughput in general case. This is the most common approach nowadays. If packet injection to the network is restricted by the NI, a (loose) upper bound can be determined for the network latency [94] but not always for the waiting time at the NI.

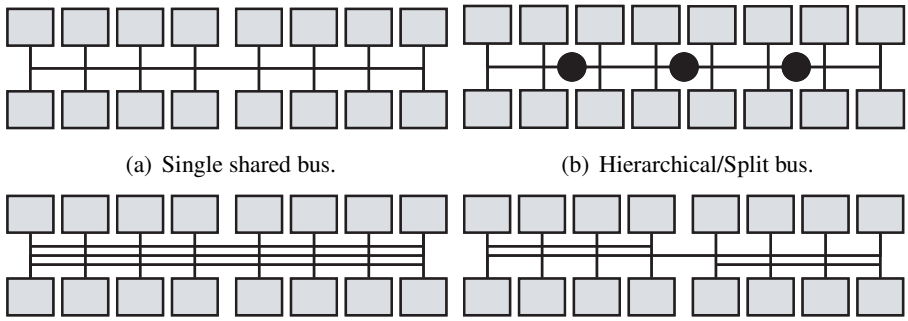
b) *Guaranteed throughput* (GT) scheme can offer a minimum level of transfer capability through the network.

2.2 Network topologies

This section reviews some basic topologies that have been proposed for NoCs. Their properties are analyzed later in more detail.

2.2.1 Single-hop topologies

The term *bus* refers to a set of signals connected to all communicating devices. A bus is called *multimaster* or *shared* when there are more than 2 agents that are able to initiate transfers. Fig. 4 shows few variations of the bus topology: single, hierarchical and multibus. Only a single agent (rectangle in figure) can transfer data on a shared



(a) Single shared bus. (b) Hierarchical/Split bus.

(c) Fully connected multi-bus. (d) Partially connected multi-bus.

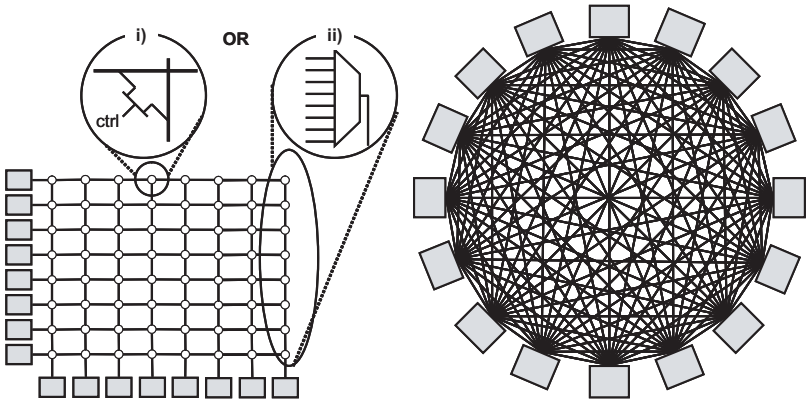
Fig. 4. *Various bus topologies for* $N_{ag} = 16$.

bus at any time. All other agents can receive data simultaneously and hence *broadcast* operation is very simple. The limitations are obvious: the bandwidth is shared among agents, the length of wires increases with system size, and some means for *arbitration* (resolving contention) are needed. Despite these limitations, bus topology has been very widely utilized, especially in small-scale systems, for example $N_{ag} < 16$. Moreover, parallel computers utilizing a shared bus with up to 70 processors have been built [49].

Simple modifications to the basic shared bus topology alleviate the problems. Dividing the long bus lines into shorter segments, Fig. 4(b), improves electrical properties and allows multiple parallel transfers. The segments are connected together with *bus bridges* (black circles). A topology that uses a simple three-state buffer or similar combinatorial logic is here called a *split bus* whereas a *hierarchical bus* uses more complex bridge elements that possibly buffer the data. However, bridges impose an area penalty and the transfers that cross the bridges experience a longer latency than local transfers within one segment. Therefore, hierarchical buses are best suited to traffic with locality. A hierarchical bus is built either as a simple chain of bus segments or as a tree like structure [3].

A ring-bus is formed by connecting together the ends of a split-bus [152]. It is considered a circuit-switched ring in the presented taxonomy. Latency in a hierarchical bus is analyzed, for example, in [43]. Energy-area and delay-area trade-offs in segmented buses are analyzed in [72].

[3] It is presented here due to similarity with other bus topologies although it is a multi-hop topology.



(a) 8x8 Crossbar (b) Fully connected 16-node point-to-point

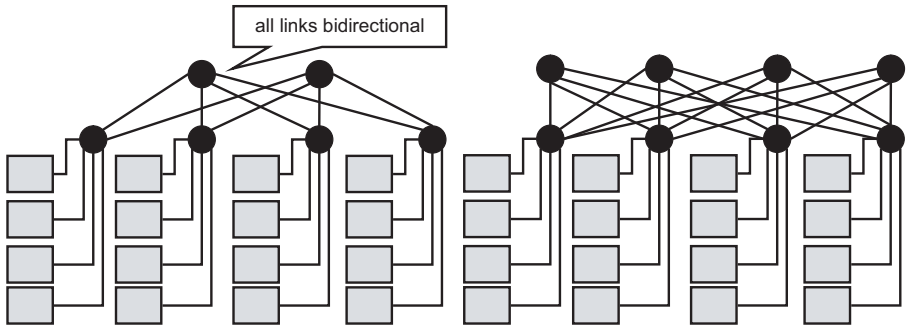
Fig. 5. *Crossbar and point-to-point topologies.*

The bandwidth can be increased by adding several parallel bus links. The agents can be connected to all links, as in Fig. 4(c), or only to a subset, as in Fig. 4(d). The former is also called *multi-layer bus* and suffers from long wires just like the single bus topology.

A multibus topology having an equal number of links and agents is called a *crossbar*, and shown in Fig. 5(a). A regular crossbar allows two transactions (sending and reception) per terminal. It is almost similar to a *fully connected point-to-point* (P2P) network, but a true P2P allows that one agent transfers data simultaneously to all targets or receives from all sources.

The crossbar in Fig. 5(a) shows inputs on the left and outputs on the bottom. The connections are formed with *i*) pass transistors or *ii*) with multiplexers. A crossbar topology is also called a *star* [44, 143] but the difference is merely related to the drawing style only and not to the functionality.

The implementation cost of both crossbar and P2P grows as N_{ag}^2 and they exhibit long wires which limit their utilization mostly to small scale systems. At the same time, these topologies offer by far the largest bandwidth. Partially connected P2P network can result in major saving in network area in cases where all agents do not communicate and the needed source-destination pairs are known, for example inside a hardwired accelerator component. Partially connected crossbar is actually a sort of an multilayer bus.



(a) Butterfly (b) Fat-tree, fan-out=4

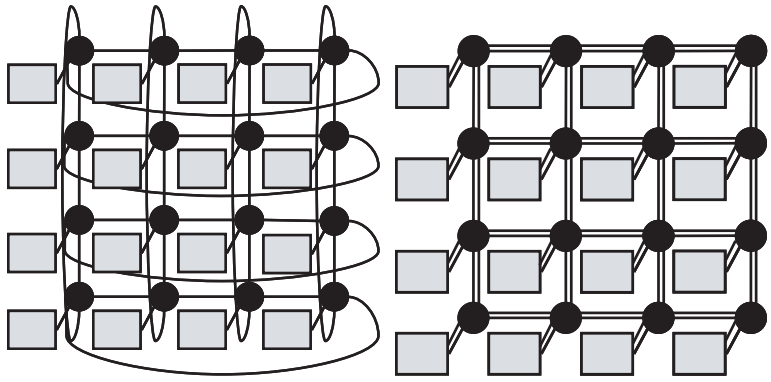
Fig. 6. *Two topologies based on butterfly. Examples shown for 16 nodes.*

2.2.2 Multi-hop topologies

The previous networks (except hierarchical bus) deliver data in one hop. Most other topologies deliver data with multiple hops, and they are derived from two families of regular networks : *butterflies* (k-ary n-flies) or *tori* (k-ary n-cubes, or hypercubes, or cubes for short). It is of course possible to have a *hybrid* topology or customize it according to application.

Fig. 6 shows two examples of butterflies. A fat-tree is a *folded butterfly* network and has many variations, for example [32,51,109,189]. It differs from a regular tree since the number of links (bandwidth) does not decrease towards the root. There are two major drawbacks with these networks: butterfly offers in its minimal form only a single path between terminals, and the longest links in a butterfly must traverse at least half the perimeter of the machine [51]. However, they allow simple routing and the maximum hop count increases logarithmically with the network size. Varying router degrees can be used to increase the scalability and performance in eXtended Generalized Fat-Tree (XGFT) [109]. Butterflies are also called *multistage interconnection networks* and there are several variations; for example, so called Banyan, Clos, and Benes variants of the butterfly network can replace a crossbar if some blocking is allowed. Splitting one large crossbar into several smaller crossbars results in notable area savings.

Fig. 7 and 8 shows four multihop network examples based on tori. Fig. 7(a) shows a 16-node *2-D torus* (*4-ary 2-cube*) where each PE is connected to a router which is connected to 4 neighbor routers. A 2-D mesh (*4-ary 2-mesh*), Fig. 7(b), is a de-



(a) Torus (b) Mesh

Fig. 7. *Networks based on two-dimensional tori: 16-node torus and mesh. For simplicity, bidirectional links are illustrated only for the mesh.*

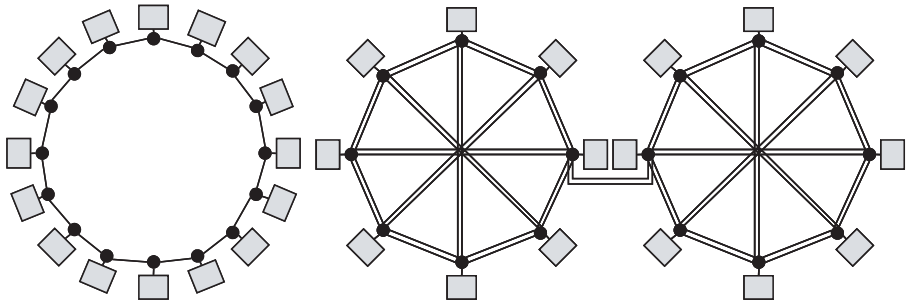
generate version of the torus since it does not have the wraparound links. All k-ary n-cubes have the number of nodes $N_{ag} = k^n$ packed in a regular n-dimensional grid and channels between nearest neighbors.

Typically, the links are bidirectional, as illustrated by two unidirectional links [4] for the mesh. Unidirectional links with a single driver are beneficial because they can be pipelined [22] A unidirectional mesh is also called a *Manhattan (street)* network.

A unidirectional ring (*16-ary 1-cube*) is shown in Fig. 8(a). A (hyper)cube topology can be extended to an *express (hyper)cube* by adding a number of long links between non-neighboring nodes. Adding long-range links to the mesh in systematic way has been studied by Ogras *et al.* [181] and it was shown to improve performance notably. For an 8-node ring this scheme is also called an *octagon* (octa = eight), Fig. 8(b). Assuming identical PEs, the system can be scaled without increasing the maximum wire length. The longest link inside one octagon ring is about 4 side lengths of a PE [107]. Larger networks can be built hierarchically from 8-node octagon rings or using a Spidergon topology that allows the scheme to be extended for other ring sizes than eight [61].

The performance increases with increased cube dimensions due to lower average hop count but saturates quickly when network costs are kept constant [51]. For example, the limited number of pins per router or the size of the crossbar inside each router

[4] A multibit, unidirectional link is actually a point-to-point *bus*.



(a) Ring (b) Hierarchical octagon

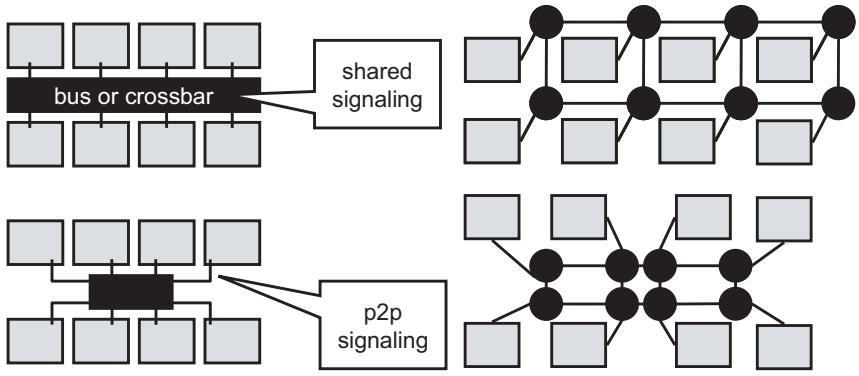
Fig. 8. *Networks based on one-dimensional tori: 16-node ring and 2 8-node octagons (hierarchical express ring).*

might force adoption of narrower links as the router's degree increases. However, Soteriou *et al.* found 3-D tori superior among the studied topologies in many cases [227]. They used a planar layout, whereas a "true" 3-D IC was considered in [62] and [192]. However, laying out 3-dimensional structures on a 2-dimensional chip may prove to be difficult, either resulting in long wires or routing congestion. Therefore, 3-D topologies are omitted here for brevity and because they are not widely utilized in NoCs (see Tables 2 and 3). The next section discusses the issues related to the floorplan of the chip.

2.3 Floorplan of network-on-chips

A floorplan refers to the placement of system components on a chip. The floorplan of the bus (single, hierarchical, multibus) and mesh resembles the conceptual view very closely. The long wraparound links in a torus can be removed by using so called *folded torus* floorplan. In a mesh, all links have a constant length (1 side length of PE) irrespective of N_{ag} whereas in a folded torus some links have twice this length.

The ring floorplan can also be realized with constant length links. Few links that are longer than one PE side length are needed depending on the number of PEs.



(a) Shared bus or crossbar (b) 2-D Mesh

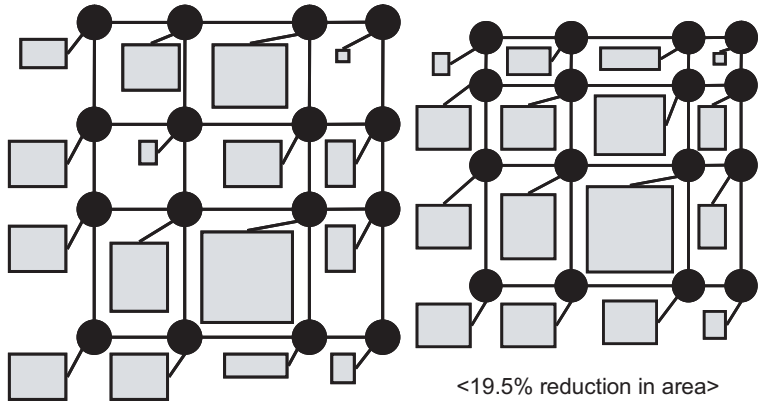
Fig. 9. *8-node examples of layout optimization.*

2.3.1 Basic floorplan optimizations

Fig. 9 shows simple floorplan optimizations for the bus and mesh. Similar techniques can be applied to other topologies as well. A conceptual, unoptimized layout, where the network logic, bus wrapper or router, is always adjacent to PE, is shown on top. A simple optimization to layout, shown on bottom, moves the network logic towards the center of the system. The longest link in the bus and the total link length in the mesh are consequently reduced although some of the PE-wrapper (PE-router) links become longer. Those links are unidirectional, point-to-point signals and are hence suitable for repeater insertion or pipelining. They also have lower utilization than those between the routers or wrappers. In the bus topology, the longest wire length is now defined as the size of the wrappers instead of the PE size. In the mesh, the longest wire remains the same but the total wire length is reduced. Routability of wires in different topologies has been studied in [213].

2.3.2 Irregular size of processing elements

Fig. 10(a) shows how a regular mesh can be applied when the size and shape of the agents varies. The minimum column width (or row height) is now determined by the largest agent on the column (row) which leads to unutilized space on the chip and hence excess area. The PE locations can be reordered in order to minimize area as shown in Fig. 10(b).



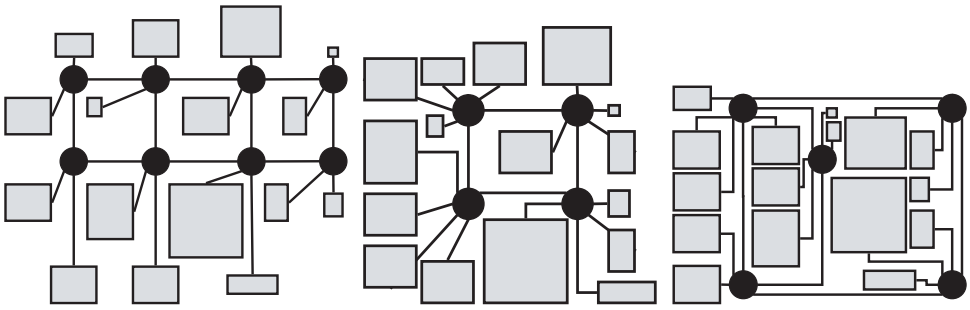
(a) Original placement (b) Floorplan-aware placement

Fig. 10. *Conceptual example of varying PE sizes in 16-node mesh.*

Mesh topology can be customized by connecting many PEs to the edge routers, as shown in Fig. 11(a) for top and bottom rows. The example has only 8 routers instead of 16 which implies notable savings in area at the expense of reduced aggregate bandwidth. The router degree remains the same ($4+4$ or $5+5$). Another option is to increase the router degree further and hence create so called *star-mesh* network. Fig. 11(b) shows a regular topology having 4-PE clusters. A cluster has star (crossbar) topology whereas the clusters are connected through mesh. The star-mesh is an example of a topology that combines two topologies: one at the local level (star, i.e. crossbar) and one at the global level (mesh). In principle, any topologies can be combined. Lee *et al.* [144] have shown that such hybrid topologies are beneficial in many cases.

An example of an irregular topology, where the routers are connected to a varying number of PEs, is shown in Fig. 11(c). This last approach may be supported by an automated synthesis tool for optimizing the topology and few examples are presented in [22]. One possibility (not shown) is to allow large PEs to span multiple "slots" in a regular mesh. Such an irregular scheme is called *mesh with regions*. To avoid deadlock, routing is somewhat more complex in irregular topologies than in regular ones [5]. These networks have been studied, for example, in [86].

[5] Note that a request-response deadlock between communicating ends can still happen regardless of the routing policy or topology.



(a) Sharing mesh routers on top and bottom row (b) Regular custom network (star-mesh) (c) Irregular custom network

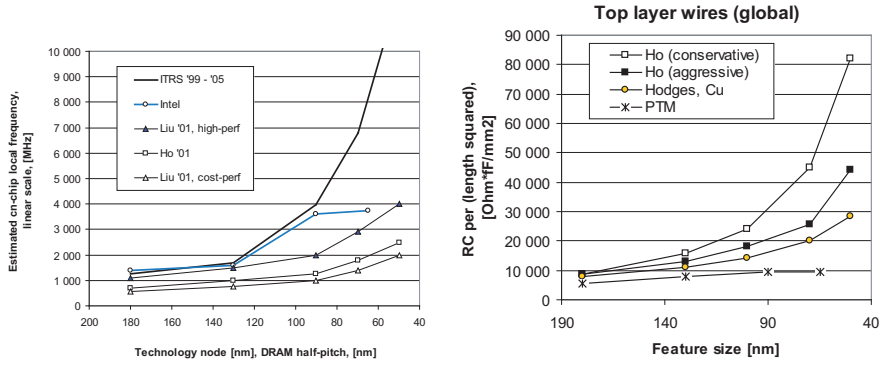
Fig. 11. *Customized 16-node networks with varying PE size.*

2.3.3 Wiring

In the past, the delay and power consumption in integrated circuits were mainly determined by the logic, i.e. transistors, and the impact of wiring was neglected. However, the dimensions of the metal wiring cannot be scaled down in the same pace as transistors. Mainly this is due to resistance of the wires; the narrower wires create larger resistance for electrical current. Similarly, the parasitic stray capacitance of the wires does not shrink as fast as transistors and inductive effects increase with frequencies. Therefore, the wiring must be taken into account already in the early phases of the design, but unfortunately, that requires complex and expensive design tools. Interested reader is referred to [82, 83, 232] for a detailed description of various wiring issues.

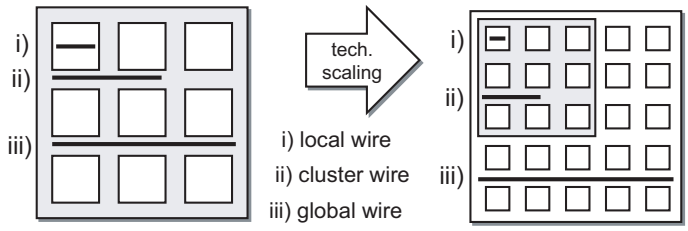
Estimates for the maximum frequency vary a lot as shown in Fig. 12(a). Values from ITRS reports 1999-2005 [92] are combined and they predict much larger frequencies for high-performance devices than other sources [82, 154]. However, all estimates agree on the fact that frequencies will continue to increase. Furthermore, frequencies for microprocessors by Intel Corporation [91] are also shown as their have historically been state-of-the-art. Interestingly, the difference between 90 nm and 65 nm technologies is small (3.60 GHz vs. 3.73 GHz). This stall is due to excess power consumption in high frequencies and therefore chip vendors are moving into multicore architectures [209].

Fig. 12(b) shows the parasitic resistance and capacitance per unit length as given



(a) Estimated frequency scaling with shrinking feature size.

(b) Various estimates for wiring delay



(c) Scaling of wires with technology

Fig. 12. *Scaling wires with technology*

in [82] [6] and [83, 154, 174] and their product. There are big differences in estimates especially in smaller geometries due to large number of parameters that must be approximated. However, the trend is clear in all presented estimates. In smaller geometries, the parasitic capacitance is slightly reduced but the resistance increases very rapidly. Consequently, their product, so called *RC* delay, increases. At the same time, the cycle period decreases and hence, a shorter distance can be reached in one clock cycle. Fig. 12(a) and Fig. 12(b) have paradoxically similar shape although the first shows frequency and the other shows the wire delay.

The delay of repeated wires depends linearly on the wire length whereas the relationship is quadratic in unrepeated wires. However, the repeaters increase the capacitive load, and hence dynamic power consumption.

Fig. 12(c) shows how wires are affected as the feature size scales down. The left

[6] Note that the [82] has mistyped units in the tables, instead of $\Omega/\mu m$ there should be Ω/mm. Units are correct in the figures.

part of the figure shows a system-on-chip with 9 processing elements. On the right, the newer silicon technology allows integration of 25 PEs with the same chip area. Wires are divided into three categories: *i*) local (inside a processing element), *ii*) cluster (between few neighboring processing elements) and *iii*) global (crossing the whole chip). The length of local and cluster wires scale with technology but global wires do not. The local wires are routed on lowest metal layers and cluster wires either on top or intermediate metal layers. Global wires use the top layers.

Globally asynchronous, locally synchronous (GALS) systems use multiple clocks within one chip and that brings many advantages, such as masking process variations and alleviating clock skew problems. In addition, the operating frequency of the whole system is not necessarily defined by the longest critical path among components. Hence, the frequency of certain parts may be increased to obtain higher performance. Alternatively, the frequencies can be minimized while still meeting the performance requirements. Removing global synchrony may also provide area and power saving and easier creation and routing of the clock circuitry. However, special synchronizing logic is needed when signals cross the clock domain boundaries to avoid meta-stability.

2.4 Routers

Fig. 13 depicts a generic router component similar to [17, 51, 207]. Input ports are shown on the left and output ports on the right. A crossbar switching fabric in the middle allows connecting any input to any output. The crossbar area is relative to $N_{inports} \cdot N_{outports}$. Due to quadratic growth, the radix of the router must be restricted. Pullini *et al.* conclude that 10×10 or at most 14×14 routers are feasible in 65 nm technology [199]. The routing logic decides which outputs are selected for the arriving packets and the arbitrator resolves the contention that arises when two or more packets are heading to the same output port. They can be implemented in centralized or parallel fashion. A centralized approach minimizes the area but may incur a larger latency for packets. Distributed schemes allocate routing and arbitration logic per port. It varies whether they are allocated per input or per output port.

Each input or output port may be accompanied with one or multiple data buffers, such as first-in-first-out (FIFO) buffers. An output-queued router has one buffer per input port at each output port. Hence, the buffering cost increases quadratically with the

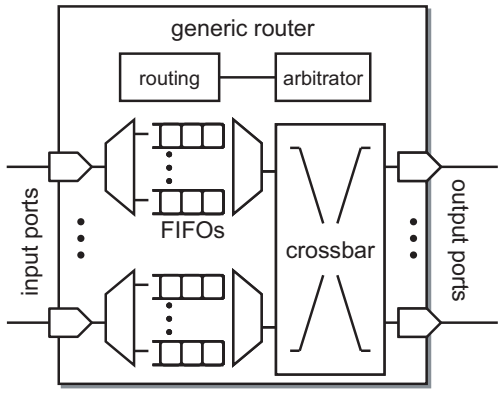


Fig. 13. *The basic structure of a network router.*

number of ports ($N_{inports} \cdot N_{outports}$). *Virtual output queuing* has the same number of buffers but they are at the inputs, and the router is a less wire dominated. A traditional input-queued router has a buffer only at each input port and, hence, a clearly smaller area (relative to $N_{inports}$) [207]. Both data width and buffer depth also affect the buffering area. Bartic *et al.* have studied size scaling of router areas in FGPA and came to similar conclusions [17].

There are four basic choices how packets are forwarded and stored at routers: *store-and-forward*, *cut-through*, *wormhole*, and *virtual channel* flow control. Store-and-forward method waits for the whole packet before making routing decisions whereas cut-through forwards the packet already when the header information is available. Both methods need buffering capacity for one full packet at minimum.

Wormhole switching splits the packets into several *flits* (flow control digits). Routing is done as soon as possible, similarly to cut-through, but the buffer space can be smaller (only one flit at smallest). Therefore, the packet may spread into many consecutive routers and links like a worm.

It is also possible to associates several virtual channels with a single physical channel which means having more than one FIFO and state holding logic at each (input) port. The buffer and channel bandwidth allocation are flit-based as in wormhole. However, blocking problems are less severe because virtual channels allow other packets to bypass the blocked one; hence creating a sort of an “overtaking lane”. This method can be used for enhancing performance and/or to avoid deadlocks that otherwise arise in certain routing schemes.

It is easy to note that the area and combinatorial delay increase rapidly with the number of ports per router [199] and/or number of virtual channels, see also [193]. Therefore, the data width and buffer depth must be decreased to keep the total area fixed in high-radix routers. Network latency as a function of network dimension was studied in [51] assuming a fixed number of pins per router. The minimum latency was achieved with 3-D and 4-D and higher dimension networks did not offer improvement. Moreover, high dimensions create difficulties in layout and routing (wire length and congestion) that were not considered. Given all this, this thesis concentrates on topologies with a low dimension (1-D like bus and ring, or 2-D like mesh).

The layout of router and wiring channels also affect the performance and cost of NoC. Interested readers are referred to the analysis presented in [186]. In this thesis, however, the impact of channel width to the wire length and area is neglected for simplicity.

2.5 Other aspects of communication performance

This section discusses few other aspects that affect the on-chip communication. The common thing is that most of them do not depend on the topology or its floorplan. These phenomena are easy to neglect in simple, benchmarking simulations, especially when traffic generators are used. However, many of these are present in real prototypes or products, and hence benchmarks that omit these may be misleading.

2.5.1 The latency components

The end-to-end latency of a transfer can be characterized with 5-tuple [4]:

$$< t_{tx,pe}, t_{tx,buf}, t_{hop}, t_{rx,buf}, t_{rx,pe} >, \tag{3}$$

whereas (1) considers only the latency inside the network.

Fig. 14 illustrates the 5 terms:

- Send Occupancy $t_{tx,pe}$ is the time a processing element and network interface are occupied to initiate a packet transmission, for example serialization of complex data structures.

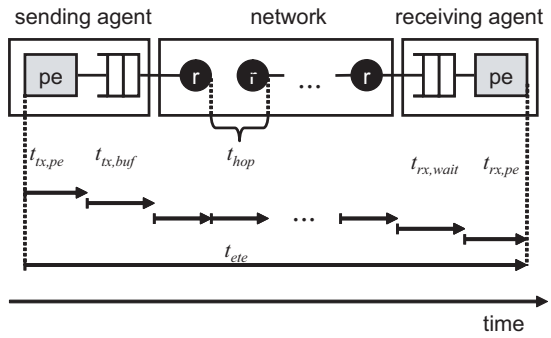


Fig. 14. *The 5 latency components of inter-PE data transfer.*

- Send Latency $t_{tx,buf}$ is the time data waits before injection into the network without occupying the processing element. This is mainly due to network contention.
- Network Hop Latency t_{hop} is the time needed to transport a message from one node (=router) to an adjacent node in the network.
- Receive Latency $t_{rx,buf}$ is the time the data on the receiving end waits before usage.
- Receiver Occupancy $t_{rx,pe}$ is the time the receiving processing element is occupied in making the remote data ready for use.

Division between $t_{tx,pe}$ and $t_{tx,buf}$ depends on how the communication functions are divided between the PE and NI. The same applies to the receiver side as well. For example in the Cell Broadband Engine's Element Interconnect Bus (EIB), the delays (in cycles) during sending phase are: $t_{tx,pe} = 5$, $t_{tx,buf} = 47 - 59$, $t_{hop} = 1$, $t_{rx,buf} \geq 0$, and $t_{rx,pe} \geq 0$ [4]. Considering all the phases, the latency varies from 80 cycles (best case) to 940 cycles (worst case).

Majority of NoC studies omit $t_{tx,pe}$ and $t_{rx,pe}$; see the references of Tables 4 and 5 for examples. However, the overhead of SW platform is substantial in many cases. For example, runtime overhead of 60 000 cycles due middleware functions is reported in [208]. A SW platform supporting distributed execution of applications is studied in [85,221]. The initialization time for inter-PE communication is about 8 000 cycles at transmitting side PE and 15 300 cycles at the receiver. The time needed for setting

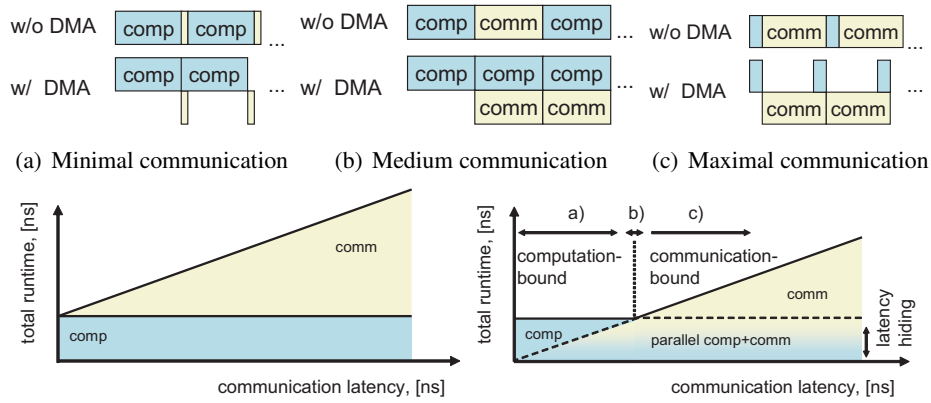
up end-to-end connection to obtain guaranteed services is analyzed in [76]. The reported values are in the range $4\,000 - 200\,000$ cycles depending on the system size and the number of connections. Hence, small differences observed between NoCs may easily become negligible at the system-level.

2.5.2 Network interface

In addition to the routers, some kind of network interface (NI) (also called network adapter) is usually needed to handle the packetization, and possibly re-ordering or retransmissions. It offers high-level services by abstracting the underlying network to a well-defined interface, such as OCP, and hence separates communication from computation. Details about the needed network interface are often omitted in many NoC studies although such interface may double the required silicon area, as in [207] and [201].

The interfacing functionality usually needs some hardware, for example to connect processors memory bus to the network, but some parts can be performed in software, for example retransmission or reordering. Hardwired accelerator functions (especially third-party components) and memories require that interfacing is done with special purpose hardware. Software approach is slower, requires more program memory than hardware counterpart [24] but easier to design and implement in the start of the development. Reusing the interfaces and other network components in several environments amortizes the development and verification cost. Hence, modularity and scalability are desired properties, as noted also in [201]. Network interface used in this thesis is presented in Chapter 8.1.

A generic packetizing process at the sender's side has the following phases: address translation to determine the target agent, preparation of header and possible tail information, data serialization and/or segmentation. Data serialization is needed if the transferred data is not stored in consecutive memory addresses, for example in case of linked list. The segmentation splits the large data chunk into limited-size packets supported by the network. At the receiver's side, interface inspects and checks the packet header and tail and extracts the payload data. The data are copied to local data memory of the PE first or consumed in first-in-first-out order. Either the corrupted data is dropped or a retransmission request is sent.



(a) Minimal communication (b) Medium communication (c) Maximal communication

(d) Total runtime without DMA (e) Total runtime with DMA

Fig. 15. *The three basic scenarios of overlapping computation with communication.*

2.5.3 Overlapping the computation with communication

A processing element can overlap the processing and transferring of data. A specific hardware component in network interface, called direct memory access (DMA), handles the communication instead of the processing element. At first, PE has to initilize DMA transfers by defining (at least) three parameters: network address, address in local its memory, and data amount. Then the DMA controller can copy data to/from local memory from/to the network while the PE does computation. The completion of a DMA transfer either causes an interrupt or the PE polls the status register of the DMA controller.

Fig. 15 shows the timing with and without a DMA. The impact on this overlap depends strongly on the relative durations of computation and communication. The biggest gain can be achieved when computation and communication times are roughly equal, Fig. 15(b), and PE always has tasks that are ready for execution after the previous is completed. Achieved parallelism is minimal when communication requirements are very low or extremely high, as shown in figures 15(a) and 15(c). Details about DMA engines used in this thesis are given in [125, 214].

The runtime is the sum of computation, communication, and synchronization. More precisely, only the non-overlapped portions of each in the critical path are considered [49]. For example, let us assume that PE_1 transfers data to busy PE_2 which cannot process the data before it has finished its current task. As long as data ar-

rives to destination prior to the end of its current task, the communication is fully overlapped with computation and has no impact on performance. In contrast, the network impacts runtime directly when a processor is stalled after miss until the memory operation is complete. Moreover, load imbalance and synchronization due to dependencies can cause idle periods in PEs even in the presence of an ideal network.

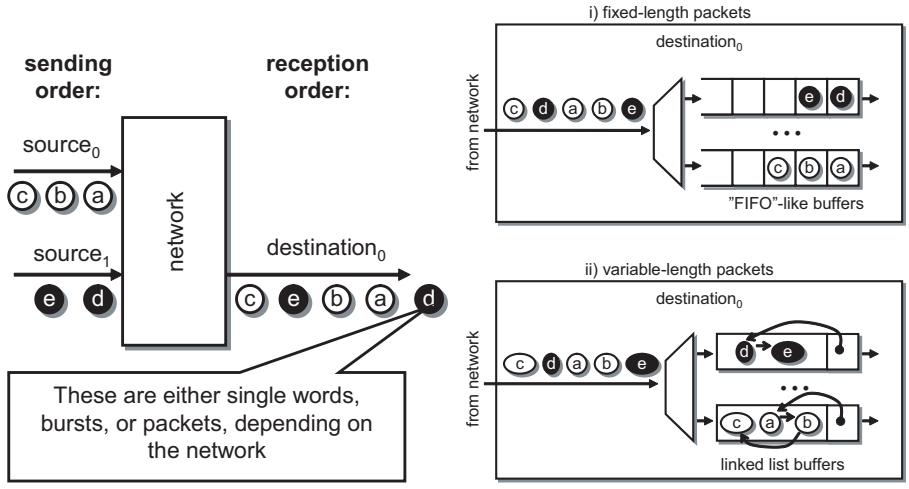
Overlapping communication with computation offers *latency tolerance* or *latency hiding*, in other words (part of the) communication latency is removed from the program's critical path. The data transfers that utilize *pre-communication* can pipeline the transfers with computation [49,128]. A prior knowledge of the applications needs, for example inside a loop with determinate behavior, allows some transfers to be initialized before-hand. This approach requires free buffer space at the receiver. One way is to use double buffering, where the data in one buffer is being processed while other data is being transferred to/from the second buffer.

Different memory architectures were compared in [150,160,198]. Scratch-pad memory scheme has obtained better performance than cache-coherent shared memory multiprocessors. Scratch-pad memories provide software with full flexibility on locality and communication management. Cache-coherent memory systems, on the other hand, use specialized hardware for automatically providing best-effort management. Scratch-pad scheme allows better optimization if program behavior is known, require less energy and network bandwidth, can better hide the network latency, but are also more complicated to program. However, the performance and energy consumption become similar in both methods with few cache optimizations [150].

2.5.4 Intertwined transfers and out-of-order data delivery

The transfers from multiple sources can be arbitrarily intertwined at the receiver depending on the state of the network. An example is shown in Fig. 16(a) where two sources (black and white) transfer data to the same target. The size of the data is irrelevant in this context. Assume that they are packets in the following discussion. The receiver must somehow identify the source to process correct data ($a-b-c$ and $d-e$, instead of $d-a-b-c-d$).

All networks have intertwined transfers but on different granularity. Packet-switching or non-preemptive *burst* transfers, guarantee the minimum number of words that are transferred consecutively to destination without interruption. On the other hand,



(a) Example of intertwined transfers. (b) Example of packet reordering.

Fig. 16. *Intertwined transfers and packet reordering.*

many buses, such as HIBI, guarantee only that the data are transmitted uncorrupted to the destination. Hence, the transfer may be interleaved on word-by-word basis.

In Fig. 16(a), all data from a single source remains in order. This is not always the case, for example, when adaptive routing algorithms are used. Assume also that packet *b* takes a shorter or otherwise faster route through the network than *a*. Then, *b* reaches destination before *a* although they entered the network in different order. This is illustrated in Fig. 16(b). Assume that the data must be processed in the same order as it was sent. Separate reordering storage must be allocated for all possible senders, data packets must be numbered, and source must be identified. The depth of the buffer is application-dependent. It must be long enough so that processing can begin when buffer becomes full. Separate buffers are needed to avoid head-of-line (or head-of-queue) blocking on the left side of the multiplexer. For example with single buffer, a deadlock may occur or data must be dropped. Once the data *e* is received, both *b* and *a* are dropped because *d* is needed before processing can start.

Fixed-length packets simplify reordering as shown on top of Fig. 16(b) (case *i*). The location where to store incoming packets is easy to determine once the size and number of the packet are known. Variable-length packets (case *ii*) cause problems because it is impossible to know the sizes of the packets that are not received. Therefore, a linked list must be used.

Possibility of data dropping necessitates sending a positive acknowledge (ACK) after the completion of transfers. Signaling only dropped packets is problematic since the negative ACK (NACK) packets could also be dropped. However, buffers are still required for each active source. The receiver must send acknowledges at known intervals, for example after 8 packets, to avoid infinite reordering buffers. The buffer size per source is a product of interval between acknowledges and the maximum packet size. In the worst case, all other agents are sending to one target which means that total buffer size grows with $N \cdot (N-i)$. With fewer buffers than senders, a separate handshake phase is needed (cf. Section 2.5.5).

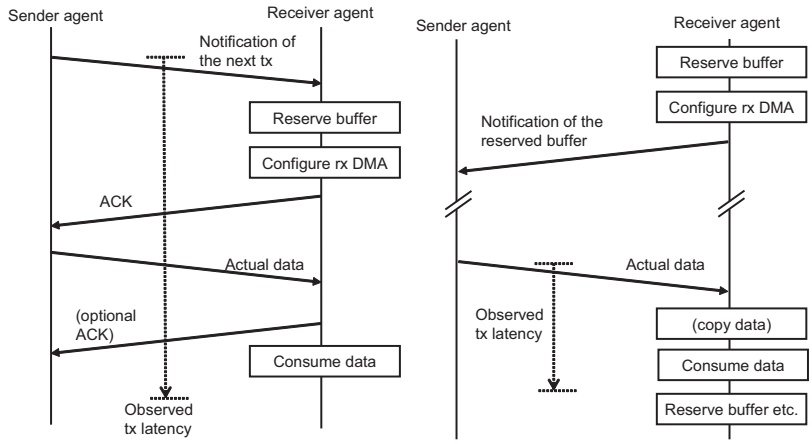
Open Core Protocol (OCP) [178] supports concurrency and out-of-order processing of transfers with notion of multiple *threads*. Transactions within different threads have no ordering requirements, and so they can be processed out of order. Within a single thread of data flow, all OCP transfers must remain ordered. While the notion of a thread is a local concept between a master and a slave communicating over an OCP, it is possible to pass thread information globally from initiator to target using connection identifiers. Connection information helps to identify the initiator and determine priorities or access permissions at the target.

For simplicity, all networks developed in this thesis use deterministic routing algorithms that are guaranteed to deliver data in-order.

2.5.5 Limited buffering at the receiver

In message-passing system, received data must be stored locally if the receiving PE cannot process the incoming data immediately, or if the data must be reordered. This creates few complications, either excess memory requirements or additional handshaking for buffer reservation prior to transfers. Without handshaking, all senders assume that they can start transfer without any preparation or negotiation. Therefore, each receiver must reserve a local buffer for each sender in the system. Naturally, the practical buffer size is limited. The memory requirement increases quadratically with number of agents (all agents can be both senders and receivers).

Having less receiving buffers than senders necessitates some sort of handshaking. In Fig. 17(a) a request is sent first denoting the length of the transfer and the initiator. Then, receiver allocates local buffer space, activates the receiving DMA is (this might



(a) Prior to transfer (b) Reservation beforehand

Fig. 17. *The impact of limited buffering to transfer initiation. There are two basic choices when the receiver reserves the buffers for incoming data.*

be omitted in some architectures), and sends an acknowledgement. The positive acknowledgement contains the address where to send the actual data (target address + port identifier for the DMA), so that initiators of concurrent transfers can be distinguished. Negative acknowledge can be sent if no buffer is available (non-blocking request). Another choice is to wait until a buffer is free (blocking request) and send only positive acknowledges. Note that the sizes of the transfers in Fig. 17(a) are arbitrary although each of them is drawn as a single arrow. For example, actual data is likely much larger than notifications or acknowledgements.

Fig. 17(b) shows another basic approach that is applicable when fixed-length transfers are used. The receiver reserves the buffers first and sends positive acknowledges to those initiators it considers appropriate. In general case, this means all initiators. The initiators that have received ACK can start a transfer at any time without handshaking which reduces the transfer latency. Difference to previous scheme is that the transfer length is now defined by the target and prior to transfer. Consequently, long messages require several of these transfers. Short messages must signal the actual data length somehow, for example as the first datum. Then, there are (at least) two choices. First, dummy data is used to fill the transfer and DMA interrupts the PE when expected number of words has been received. Second, receiver actively polls the DMA controller to detect the completion of the transfer.

The utilized network affects directly the latency of the handshakes and actual data transfers. However, the buffer management also presents some overhead, especially when executed in software. Allocating memory dynamically in software (tens to hundreds of cycles), CPU interrupt (tens to hundreds of cycles) and possible OS context switch latencies (hundreds of cycles) must be taken into account in $t_{rx,pe}$. The rough cycle count estimates are based on our prototypes that utilize Nios II CPU with eCos operating system [10]. These overheads may even take longer than the actual data transfer and, hence, the impact of the network diminishes.

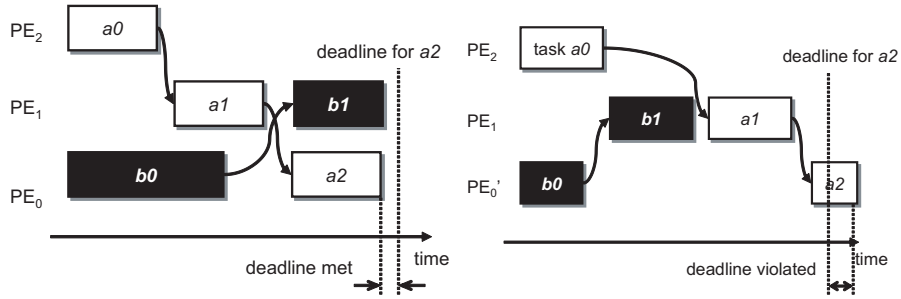
In the worst case, the DMA controller can access only a separate buffer memory that is dedicated to communication, and the data must be copied to (from) another memory before (after) processing. For maximum efficiency, the buffer management must be implemented with specialized hardware and a dual-port data memory used for both processing and communication.

As a conclusion, it is optimistic to expect that initiator can always start transfers without any handshaking overhead. Although that might happen most of the time (cf. Fig. 17(b)), in general, initiators have to stall and wait for acknowledge at some point.

2.5.6 Scheduling anomalies

Scheduling anomaly is a situation where an improvement in some part results is worse overall performance. For example, increasing the performance of the network or some PE, offers speedup for certain operations. However, the order or execution - the schedule - changes. This may lead to violated deadlines or lowered throughput. An example is illustrated in Fig. 18 for 3 processing elements and 5 tasks. The arrows represent the dependencies between tasks. The original schedule in Fig. 18(a) meets the deadline set to task $a2$. Fig. 18(b) shows what happens if the frequency of PE_0 is increased. The runtime of tasks $b0$ and $a2$ naturally decrease notably. However, task $b1$ is now executed before $a1$ in PE_1 and the deadline is violated. This is counterintuitive to the fact that PE_0 is clearly faster than previously.

Such situations are very hard to predict and avoid. Consequently, the impact of local optimization must be evaluated also in the system level, especially when deadlines are present.



(a) Original schedule. Deadline is met.

(b) Modified schedule due to faster PE_0. Deadline is now violated.)

Fig. 18. *An example of scheduling anomaly.*

2.5.7 Data synchronization

Data synchronization ensures that data is correctly sampled when crossing the clock domain boundary. Basic two-flip-flop synchronization is rather straightforward to implement and suited for RTL synthesis. The concept is illustrated in Fig. 19 It has a latency of 2 cycles from *request* to *data_valid* and 3 cycles from request to *acknowledge*. Note that also the acknowledge signal must be synchronized in similar manner by the sender. Hence, the bandwidth is only a fraction from the maximum.

Although this scheme is reliable, it clearly reduces performance even if both domains used the same clock frequency. Two clock signals can be derived from the common source so that their frequencies are integer-multiples of each other. This simplifies synchronization logic and achieves high performance but does not alleviate the problems in constructing global clock network. Several methods have been proposed to overcome these issues, see for example [53]. Unfortunately, some are very hazardous and difficult to implement [57]. For example, they require special Muller-C elements or rely on some prior knowledge about wiring or gate delays. Hence, they are poorly portable and sensitive to variations in the manufacturing process and runtime conditions. An example of synchronization method that is synthesizable with regular standard-cell or FPGA synthesis flow was presented in [127] along with similar conclusions as above.

The location of clock domain boundaries is also important in addition to the selected synchronization style. For example, the whole network (not depending on the topology) can be synchronous whereas the PEs operate on different clocks, or each net-

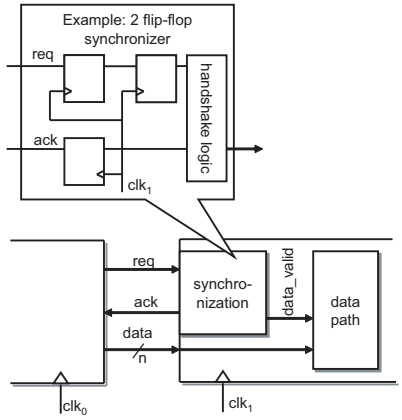


Fig. 19. *Example of synchronization between clock domains.*

work router can have different clock source. The first option is likely to incur less synchronization overhead, only at the network interfaces at source and destination ends of the path. The second, on the other, has the overhead on every router in the path.

Performance analyses in this thesis assume that the whole system is either fully synchronous or that network is synchronous and PEs utilize derived clocks (synchronous multi-clock). This simplification removes the synchronization overheads of our general-purpose synchronization [126]. Furthermore, the utilized FPGA prototypes did not encounter any major problems with the clock tree synthesis although they are evident in high-performance ASICs.

2.5.8 Serial links and data encoding

Using serial links instead of parallel has also been suggested [144, 170, 179]. One reason is that interwire capacitances in parallel links cause crosstalk which can be also alleviated with shielding wires and inserting repeaters. However, these methods increase the area and/or power consumption. On the other hand, parallel links allow lower frequency which reduces power consumption, especially if supply voltage is also reduced. Most contemporary NoCs (including this work) utilize parallel links [P10, P[11].

Transmitted data can be encoded in order to achieve low power [95, 144]. However,

the total power is not necessarily lowered by reducing only the toggle count because power due to encoding and decoding hardware must be accounted as well. It was found in [185] that dozens of hops must be traversed in many in order to compensate the power overhead of encoder/decoder. It has been argued that realistic buses or NoC links are too short to allow notable power saving [119] and that the encoding actually increases the overall power consumption [95]. However, encoding has also been shown beneficial, at least for serial links in [144]. Low-power encoding is better suited for off-chip than links since they have greater length and operating voltage.

Smaller process geometries and operating voltages combined with higher frequencies and larger variations are projected to lead larger number of bit upsets in integrated chips. Communication networks, especially their links, are estimated to prone to these soft errors [21]. The codes can detect or correct varying number of bit errors. Those data words that are detected as erroneous but cannot be repaired have to be retransmitted. The results regarding the memory traffic of a CPU in [21] suggest that error detection with retransmission is more energy efficient than error correction. This is due simpler encoding and decoding logic. However, the authors note that technology scaling may favor error detection in the future because it induces less communication. Furthermore, the power consumed in memory buffers must be accounted with NoCs. A special case of encoding that encrypts the on-chip traffic in high-security devices was envisioned in [66].

3. SURVEY OF NETWORK-ON-CHIPS

This section presents literature studies on NoC proposals and comparisons. Properties such as utilized topologies, switching methods, evaluation criteria are collected and commented. The findings will be summarized into a representative data sheet for a contemporary NoC accompanied with improvement suggestions. Finally, a set of IP components is reviewed briefly. The goal is to give examples of the modules that are suitable for NoC-based systems.

The early work and basic principles of the NoC paradigm were outlined in various seminal articles, for example [19, 52, 71, 78, 79, 132, 180, 184, 225, 246, 248] and text books [20, 96, 177]. However, the aforementioned sources do not present many implementation examples or conclusions about the current proposals. This Chapter complements the survey [26] by providing a categorized, in-depth literature study of NoC proposals and comparisons. The implementation results will be covered later in Section 9.1 together with results from this Thesis.

As a rather new research field, network-on-chip design suffers from contradictory and/or insufficient definitions of terminology. It is interesting that none of the NoC text books [20, 96, 177] gives a clear, short definition what is a NoC. This thesis assumes a simple and demystified definition that "*network-on-chip is a communication network that is implemented on chip*". It is a concept which presents a unification of on-chip communication solutions.

3.1 NoC proposals

Extensive summary of NoC proposals is given in Tables 2 and 3 [P11] whereas NoC comparisons are summarized in Tables 4 and 5 [P10]. Table 1 lists the used topology symbols.

Table 1. *The topology symbols used in other tables*

Symbol	Topology
b, hb	(single shared) bus, hierarchical bus
x	crossbar
m, t	2-D mesh, 2-D torus
em, sm	2-D extended mesh, 2-D star-mesh
3m, h	3-D mesh, 3-D hypercube (i.e. torus)
tr, ftr	tree, fat-tree
r, er	ring, extended ring
c	custom
(c)	custom, but only 1 topology is reported
p	point-to-point
db	de Bruijn graph

Tables 2 and 3 list the topology, routing, switching scheme as well as utilized evaluation methodology and criteria. The bottom rows show the number and percentage of the papers reporting/using the given property. Furthermore, the percentages from [P10] are also given at the bottom row for comparison.

Each property is explained in more detail in the following. Tables list support of various configurations if the papers explicitly report it. Any unclear or partially applying properties are marked in parentheses. Although very extensive, Tables 2 - 5 cannot be all-inclusive. However, it is argued that the selected publications offer a representative view of NoC design. The missing information may be addressed in other publications by the same authors, and hence, all the tables should be interpreted as *"at least these things are known"*.

3.1.1 Switching policy

The upper part of Table 2 lists the *circuit-switched* (*c* in the table) and lower part shows *packet-switched* (*p*) networks, both in alphabetical order. Switching policy is not clearly stated in all cases and sometimes both schemes are supported.

Packet-switching is more common and it is utilized in about 80% of the studied NoCs. Some sources assume packet-switching as key property of NoCs but this study will consider also the circuit switched networks.

Wormhole switching (*w*) is the clearly most popular and well suited on chip. On the other hand, cut-through (*c*) and store-and-forward (*s*) are rare. They require more

buffering capacity and it commonly outweighs the gains of reduced blocking on links during high traffic load.

Circuit-switching is best suited for predictable transfers that are long enough to amortize the setup latency, and which require performance guarantees. Circuit-switching scheme also reduces the buffering needs at the routers. Packet-switching necessitates buffering and introduces unpredictable latency variation (jitter) but is more flexible, especially for small transfers. It is an open research problem to analyze the break-even point (predictability and duration of transfers) between the two schemes.

3.1.2 Utilized topologies

Physical restrictions in IC layout favor the utilization of 2-dimensional topologies and all but 3 papers in Tables 2 and 3 have two-dimensional topology. The most common topologies are 2-D mesh and torus which constitute about 60% of cases. Custom, fat-tree, and crossbar have roughly even proportion followed by ring-based topologies. Otherwise similar distribution is observed in [P10] except that buses were used commonly as a reference instead of multi-hop topologies. A survey concentrating solely on on-chip buses can be found in [P1].

A crossbar (or star) is a non-blocking network with high performance. However, high implementation cost restricts the usage for local communication only instead of large systems. CDMA NoCs [113, 252] are here considered as crossbars. Hierarchically constructed networks, for example Slim-spider (hierarchical star) [144], star-mesh [44], and SNA [145], differentiate local and global topologies. This way the topology better matches the differences between local and global communication.

Unlike general-purpose macronetworks, NoCs can be tailored according to the requirements of the application domain. Application-specific networks can obtain superior performance while minimizing both area and energy [22, 88]. A topology can also be extended by systematically adding links between non-neighboring nodes [61, 107, 181]. Few approaches are customizable but the results are given for one topology only [9, 18, 42, 148, 223]. However, customization is contradictory to the early projections of simplified layout and wiring optimization that are possible with regular topologies [52, 78, 79]. Regular structure suites especially well general-purpose chip-multiprocessors (CMPs) with homogeneous resources [236]. Resources in con-

Table 2. *Extensive summary of network-on-chip proposals in literature [P11].*

#	Network	Author	Ref.	Switching		Guarantees	Topologies [2]				Rout	Eval. method						Evaluation metrics							
				circuit/packet	s/c/w/d [1]		bus, crossbar	mesh/torus	fat-tree/tree	(hier.) ring	custom	det. / adapt.	Analytical	Simulation	Synthesis	FPGA [3]	ASIC [3]	Applications	Comparison	area	frequency	latency	power/energy	throughput	runtime	utilization	other	# criteria
1	AET	Valtonen	[235]	c?	-	x		m				(d)	x		x			-	-	x	x					1	3
2	aSOC	Liang	[151]	c	-	x		m				(d)		x	x			5	x	x	x			x			3
3	Crossroad	Chang	[38]	c	-	x					c	(d)		x	x		5	3	x	x		x	x				4
4	dTDMA	Richardson	[204]	c	-	x	b					(d)	x	x				-	x		x	x						2
5	HIBI	Salm., Kulm.	[P6,129]	c,p	(w)	x	hb					(d)	x	x	x	58		3	x	x	x	x		x	x	x		6
6	Hu	Hu	[88]	c	-	x					p	(d)		x	x			1	x	(x)			x				1	3
7	Nexus	Lines	[153]	c	-	x	x					(d)					16	-	-	x	x	(x)	x					4
8	Pnoc	Hilton	[81]	c	-	x					c	(d)		x	x	8		1	x	x	x			x			3
9	ProtoNoc	Castells-Rufus	[36]	c	-	x		m				(d)	x		x			-	-	x			x					2
10	SDM NoC	Leroy	[148]	c	-	x		(m)			(c)	(d)		(x)	x			1	x	x	x		x					3
11	SocBus	Wikl., Henr.	[247,80]	c	-	x		m				(d)		x	x		2	2	x	x	x	x		x				4
12	Wolkotte	Wolkotte	[250]	c	-	x		m				(d)		x	x			-	x	x	x		x					3
13	Aethereal	Rijpkema	[207]	p,(c)	w	x		(m)			(c)	n/a			x			-	-	x	x							2
14	Ahmad	Ahmad	[2]	p,c	w	x		t				a		x	(x)			-	-	(x)		(x)		x				3
15	ANOC	Beigne	[18]	p	w	x		(m)			(c)	a			x			-	-	x	x							2
16	Arteris	Arteris	[11]	p	w,s	-					c	n/a	x					-	x	x	x	x	x					4
17	Bartic	Bartic	[17]	p	c	-	x	m,t,h	tr			d*			x			1	x	x								1
18	Blackbus	Anjo	[9]	p	s	?		(m)			(c)	any		x				7	-							1		1
19	Bouhraoua	Bouhraoua	[32]	p	w	(x)			f			a		x	x				-	x		x						2
20	Butterfly FT	Pande	[189]	p	w				ftr			a		x	x				-	x				x				2
21	Carloni	Carloni	[35]	p	n/a	-					p	n/a	x					1	-				(x)				1
22	CDMA NoC	Kim, D.	[113]	p	n/a	-	x	sm				n/a	x		x			-	-	x		x						1
23	CDMA NoC(2)	Xin Wang	[252]	p	n/a	x	x					n/a		x	x			-	x	x		x	x					3
24	Chi, H-C	Chi, H-C	[40]	p	s	x		t, (m)				d		x			1	-	x	x		x	x					3
25	Cliche	Kumar	[132]	p	n/a	-		m				d		x				-	-							1		1
26	Crossbow	Crossbow	[48]	p	n/a	x		m				n/a			x	x		-	-	x	x							2
27	DyAD	Hu	[89]	p	w	-		m				d+a		x	x			1	x	x		x						2
28	Eclipse	Forsell	[64]	p?	s?	-		sparse m				d?	x	x				5	x						x	x		2
29	Ext. mesh	Ogras	[181]	p	w	-		em				d*	x	x	x	16		2	x	x		x	x			x		4
30	Faust	Lattard	[141]	p	w	x		m				d					23	1	-	x	(x)	(x)	x	x				5
31	Feero	Feero	[62]	p	w	-		3m				n/a	x	x	x			-	x	x		x	x	x				4
32	Felicijan	Felicijan	[63]	p	w	x		m				d		x				1	-			x		x			1	3
33	Gezel	Ching	[42]	p	w	-		(t)			(c)	d, (a)		x	x			-	x	x	x	x	x					3
34	GDB	Hosseinabady	[87]	p	w	-					db	a		x	x			-	x	x	x	(x)	(x)					4
35	Hermes	Moraes	[169]	p	w	-		m				d,a		x	x	4		-	-	x	x	x		x				4

continues on the next page

Table 3. *Extensive summary of network-on-chip proposals in literature (continued) [P11].*

continued from the previous page

#	Network	Author	Ref.	Switching			Topologies [2]				Rout	Eval. method						Evaluation metrics								
				circuit/packet	s/c/w/d [1]	Guarantees	bus, crossbar	mesh/torus	fat-tree/tree	(hier.) ring	custom	det. / adapt.	Analytical	Simulation	Synthesis	FPGA [3]	ASIC [3]	Applications	Comparison	area	frequency	latency	power/energy	throughput	runtime	utilization	other	# criteria
36	Kavaldjiev	Kavaldjiev	[110]	p	w	x		m				d,a		x	x			1	-	x	x	x						3
37	Kim, J.	Kim, J.	[115]	p	w	-		m,t				a		x				-	x			x	x					2
38	Lee, H.G.	Lee, H.G.	[142]	p	n/a	-					c,p	d	x	x	x	8		1	x	x		x	x	x	x		5
39	Lochside	Mullins	[172]	p	w	-		m				d					16	-	-	(x)	x	x						3
40	Mango	Bjerregaard	[25]	p	w	x		m^2				d			x			-	-	x	x							2
41	Microspider	Evain	[61]	p	w	x				er		d			x				(x)	x	x							2
42	Mocres	Janarthanan	[93]	p	c	-		m				d		x	x			-	x	x	x	x						3
43	Mondinelli	Mondinelli	[166]	p	s	-			ftr			d				64		-	-		x			x				2
44	Nostrum	Millberg, Pen.	[163,194]	p	s	x		m				a		x				1	-			x	x					2
45	Octagon	Karim	[107]	p,c	n/a	x				er		d	x	x				-	x			x					2	3
46	Orion/Luna	Soteriou	[227]	p	w,e	-		em,t,h	ftr	r,er		d,a		x	x			30	x	x		x	x					3
47	Pavlidis	Pavlidis	[192]	p	w	-		3m				d	x	x				-	x			x	x					2
48	PMCNOC	Wang	[243]	p	*	-		m				d		x	x			-	x	x	x	x		x				4
49	Proteo	Sigue., Ahon.	[223,3]	p	c	-				(r)	(e)	d		x	x			1	-	x								1
50	Qnoc	Bolotin, Rost.	[29, 210]	p	w	x		m				d	x	x	x			-	x	x		x	x			x		4
51	Ring road	Samuelsson	[216]	p	s	-				r		n/a		x				-	-							x		1
52	Slim-Spider	Lee, K.	[143,144]	p,c	n/a	x	x					(d)	x	x		9		-	x	x	x	x						3
53	SNA	Lee, S.	[145]	p	n/a	-	x	sm				n/a		x				1	x						x			1
54	SocIn/RaSoc	Zeferino	[257]	p	w	-		m				d			x			-	-	x	x							2
55	SPIN	Guerr., Andr.	[71, 6]	p	w	-			ftr			a	(x)	x	x			-	x	x		x		x				3
56	Star-mesh	Cidon	[44]	p	w	-		sm				d	x	x				-	x					x				1
57	TeraFLOPS	Vangal	[236]	p	w	-		m				d,a				80	4	x	x	x	(x)	x				1	5	
58	XGFT	Kariniemi	[108]	p	w	-			ftr			a		x	x			-	x	x	x	x		x		1	5	
59	Xpipes	Bertozzi	[22]	p	w	-					c	d		x	x			4	x	x		x	x			1	4	
60	Xu	Xu	[253]	p	n/a	-	x					(d)	(x)	x	x			1	x	x	x		x		x	1	5	
	# reported	61	61	62	51	28	11	38	8	6	9	52	20	44	40	7	10	25	37	48	31	34	22	16	9	7	11	61
	% reported	100	100	102	84	46	18	62	13	10	15	85	33	72	66	11	16	41	61	79	51	56	36	26	15	11	18	100
	% in [P10]	44 papers	-	-	-	-	72	59	27	11	27	-	20	82	55	9	0	30	100	55	25	41	39	30	57	-	30	-

Notes: [1] w= wormhole, c=cut-through, s=store-and-forward
[2] see Table I for topology symbols
[3] number of NoC terminals in largest prototype

temporary SoCs, on the other hand, have often varying size and shape, see for example [22, 67, 88, 132], which should be accounted in topology/floorplan analyses.

3.1.3 Routing

Packet-switched networks mostly utilize deterministic (d in the table) routing (about 70 %) but adaptive schemes (a) or some means for reprogramming the routing policy is necessary for fault-tolerance. The studied circuit-switched NoCs do not commonly discuss how the circuits are setup. However, the transmitted data remains in-order since the circuit stays intact throughout the transfer, hence the notation (d). Hermes supports selection of the scheme at design time [169]. Updating the routing tables at runtime which allows adaptivity event though the routing is deterministic (marked $d*$) [17, 181]. DyAD [89] switches automatically between deterministic and adaptive at runtime and guarantees freedom from routing deadlock and livelock. Deflection routing forwards packets every cycle and uses adaptive misrouting (other than minimal path) in case that target direction is occupied [163]. An interesting option is to split the traffic across several paths to reduce congestion on certain area of the network [22].

Although deadlock is generally avoided, out-of-order packet delivery is problematic with adaptive routing. Many sources neglect this phenomenon totally and others, for example [71, 163], assume that network interface or software performs the reordering. The cost of reordering in terms of runtime and area overheads are unfortunately neglected which author finds as major deficiency.

3.1.4 Quality-of-Service

Over half of the articles promise guarantees for data transfers, mostly on timing aspects. Although many articles mention QoS, only few actually discuss the implementation in detail or evaluate its efficiency.

Three methods are employed to give timing guarantees: network dimensioning, circuit-switching, and prioritized packet scheduling [162].

The fundamental feature of GT is the necessity of *a priori* knowledge about the traffic load conditions. For example, the network size (link width, topology, buffer sizes)

is set to tolerate the *estimated worst case* conditions. Similarly, simultaneously activated circuits must be statically determined. Otherwise, the circuit setup may fail at runtime and lead to non-deterministic delay prior to transfer. Therefore, the complexity of traffic modeling and system simulation is high, and no guarantees can be given to new applications [162]. Prioritizing packets offers relative guarantees, for example lower latency for high-priority packets, but no exact guarantees either.

More research is needed to obtain accurate, representative traffic models that can be used in benchmarking and in NoC design. Conservative models are especially critical when designing real-time systems. At the same, guaranteed services should coexist with BE and provided with minimal overhead. Most of QoS issues are coupled with routing and flow control policies.

3.1.5 Testing and fault-tolerance

Testing aims to detect the errors that occurred during fabrication, such as stuck-at-faults or short circuits. Fault-tolerance means the ability to operate in the presence of faults, either hard or soft. Link testing must be addressed in addition to NoC routers [67, 187]. At the same time, test mechanism must meet the time, power, coverage, and temperature constraints. Using already tested routers to deliver test data and exploiting the NoC's parallelism for simultaneous testing of multiple nodes is presented in [69]. A special self-test mechanism for NoC was presented in [108].

Unfortunately, these issues are mostly neglected in the studied papers, except [153] and [108]. Decreasing feature size, higher frequencies, and increased process variation expose the modern SoC to various faults and countermeasures must be actively sought and studied.

3.2 NoC comparisons

Tables 4 and 5 list a summary of network comparisons found in literature.

The studies are sorted according to the number of networks, system sizes and then alphabetically. The properties are divided into three sections: *Topology and size*, *evaluation type*, and *criteria*.

Table 4. *Summary of comparative NoC studies [P10].*

#	Author	Ref.	Topology and size												Evaluation type					Evaluation criteria						
			single bus	hier.bus	crossbar	mesh/torus	tree/fat-tree	ring	other	# networks	# system sizes	size ratio	analytical	statistical tg	tx-dep. tg	application	#cases	Repeatable cases	runtime	area	power/energy	latency	throughput	frequency	other	# criteria
1	Salminen, T.	[215]				m				1	1	1		4			4	(x)				x			x	2
2	Moraes	[169]				m				1	2	6		3			3	x	x	x		x				3
3	Penolazzi	[194]				m				1	5	4		1			1	x			x					1
4	Hu	[88]	b						p	2	1	1			30	1	31	-			x				x	2
5	Saastam.	[212]					tr	r		2	1	1			1		1	x	x	x					x	3
6	Thid	[233]	b			m				2	1	1			1		1	x	x		x				x	3
7	Thid	[234]	b			m				2	1	1		2			2	x	x							1
8	Xu	[253]	b		x					2	1	1				1	1	-	x	x	x			x	x	5
9	Andriahant.	[6]	b				f			2	4	8		1	1		2	x	x			x				2
10	Lahtinen	[139]	b		x					2	4	4	x				1	x		x		x	x	(x)		4
11	Lee, H.G.	[142]			x				p	2	4	2		1			1	x	x	x	x			x	x	4
12	Ogras	[181]				m, em				2	5	6			3		3	(x)	x	x	x					3
13	Charlery	[39]	b				f			2	6	6		1			1	(x)	x				x			2
14	Lu, R.	[156]	b	sb						2	8	8		2			2	x				x	x			2
15	Arteris	[11]		hb					c	2	*f()*	*f()*	x				1	x		x	(x)	x	x	x		5
16	Zeferino	[258]	b			m				2	*f()*	*f()*	x				1	x	x	x				(x)		3
17	Zeferino	[256]	b				f			2	*f()*	*f()*	x				1	x	x	x				(x)		3
18	Chang, K.-C.	[38]		sb		m				3	1	1			3?		3	x	x	x	x	x				4
19	Dumitrascu	[58]	b					er	dms	3	1	1				1	1	-	x			x	x			3
20	Hilton	[81]	b							3	1	1				1	1	x	x	x				x		3
21	Kreutz	[120]				m,t	f			3	1	1	x			3	4	-		x	x					2
22	Lahiri	[134]	b	hb				r		3	1	1		3			3	x					x			1
23	Liljeberg	[152]	b	sb				r		3	1	1		3			3	(x)				x				1
24	Lu, Z.	[158]				m,t				3	1	1	x	1			2	x			x	x		x	x	3
25	Pimentel	[196]	b		x				omega	3	1	1			1		1	-	x					x		2

continues on the next page

Table 5. *Summary of comparative NoC studies (continued) [P10].*

continued from the previous page

#	Author	Ref.	Topology and size											Evaluation type					Evaluation criteria							
			single bus	hier.bus	crossbar	mesh/torus	tree/fat-tree	ring	other	# networks	# system sizes	size ratio	analytical	statistical tg	tx-dep. tg	application	#cases	Repeatable cases	runtime	area	power/energy	latency	throughput	frequency	other	# criteria
26	Shen	[222]	b	sb		m				3	1	1			2		2	-	x	x	x					3
27	Wolkotte	[250]				m				3	1	1		4			4	x		x	x			x		3
28	Cardoso	[34]				m				3	3	4				2	2	-		x		x				2
29	Liang	[151]	b	hb		m				3	5	5				5	5	-	x				x			2
30	Vassiliadis	[238]				m	f,tr			3	18	2^{17}				1	1	-	x	x				x		3
31	Salminen, E.	[P7]	b	(hb)					ideal	3	36	36	(x)		1		1	-	x					(x)		2
32	Zhang	[259]		mb		m			hier m.	3	49	49				3	3	-			x					1
33	Angiolini	[7]	b	mb		m				4	1	1			2		2	(x)	x	x	x	x	x	x		6
34	Kreutz	[121]	b			m	tr			4	1	1			3		3	-	x							1
35	Richardson	[204]	b			m				4	4	16		2			2	(x)				x				1
36	Ryu	[211]	b	hb					(c)	4	4	24				3	3	-	x	x			x			3
37	Bolotin	[28]	b	hb		m			p	4	*f()*	*f()*	x	3			4	x		x	x				x	3
38	Bartic	[17]			x	m,t	tr		h	5	1	1	x				1	x		x			x	x		3
39	Pande	[190]				m,t	f	er		5	1	1		3			3	x		x	x	x	x		x	5
40	Bertozzi	[22]				m,t	f		c	5	3	3		4		2	6	x		x	x	x			x	4
41	Loghi	[155]	b	mb	x					5	4	3				4	4	-	x			x			x	3
42	Salminen, E.	[P8]	b	hb		m				6	4	16		1	5		6	x	x	x		x			x	4
43	Bartic	[16]			x	m,t	tr			7	1	1				1	1	-	x	x						2
44	Lee, K.	[144]		mb	x	m			p	9	7	6		2			2	(x)		x	x					2
	Count	45	27	16	9	27	12	6	13	45	45	45	11	19	13	14	45	27	27	25	18	18	14	12	14	45
	%	-	60	36	20	60	27	13	29	-	-	-	24	42	29	31	-	60	60	56	40	40	31	27	31	-
	Avg	-	-	-	-	-	-	-	-	2.9	5.1	5.9	-	2.3	4.5	2.2	3.0	-	-	-	-	-	-	-	-	2.8

Symbols for topologies:

b, mb	(single shared) bus, multibus	x	crossbar	f, tr	fat-tree, tree
hb, sb	hierarchical bus, split bus	m, em	mesh, extended mesh	r, er	ring, extended ring
		t	torus	c	custom

p	point-to-point
h	hypercube
dms	distributed shared mem.

3.2.1 Compared topologies

The evaluated topologies and number of different system sizes are shown in the section *Topology and size*. Single bus is separated from bus topologies with multiple links (hierarchical, split, and multibus). Single bus and mesh/torus are the most studied topologies (over 50% of the cases) They are followed by hierarchical buses, fat-trees, and rings. One fourth of the studies consider also other topologies, such as point-to-point or custom. A special case is [P7] which compares a single shared bus is to "ideal" network.

Column *#networks* shows how many NoCs are compared. The value can be larger than the sum of the listed topologies because different versions of the same topology are counted separately. For example, wormhole and store-and-forward mesh are considered as two networks. On average, 3 networks are compared and the largest studies included 9 [144] and 12 networks [227].

Half of the cases consider only one system size (number of terminals) and about 5 sizes are included on average. The most sizes are covered in [238](18 system sizes), [P7] (36 sizes), and [259](49 sizes). Analytical models are shown with $f()$ and they offer naturally very large range of systems. Large networks have been studied in simulation also; up to 100 nodes in [34, 144, 181] and 256 nodes in [190].

The *size ratio* is the largest system size divided by the smallest and rounded to nearest integer. It gives coarse idea of how large design space is covered. It is about $6x$ on average and $11x$ when counting only the studies with multiple sizes. Varying the system size over $10x$ covers wide spectrum and hence gives good justification for presented claims and observations. An exceptional and clearly the largest system range, $2^3 - 2^{20}$ nodes, is covered in [238] and it is not included in the above average values.

It is worth noting no single fixed topology categorically outperforms all others. The properties of basic topologies are quite well-known but customization and IP mapping emphasize the importance of efficient design automation tools, application modeling, and further research.

3.2.2 Evaluation methods and metrics

Simulation and *synthesis* are clearly the most popular evaluation methods. One third of the studies use analytical methods and one fourth have presented prototype chips.

On average, the studies in Tables 2 - 5 use three different metrics for comparing NoCs which is barely adequate for good comparison. The most popular and important metrics are application *application runtime*, silicon *area*, *power* consumption, and *latency*. All these are to be minimized and an appropriate trade-off is sought.

Category *others* denotes metrics that are included only in few articles, for example, operating frequency, wire length, latency jitter, congestion, path diversity, or packet loss. These metrics are less suitable for direct comparison. Their major purpose is in understanding and optimizing the system, and their impact should be reflected in the four "major" comparison metrics. There are few papers where the marked metrics are measured only for a fraction of studied networks or system sizes. The obtained results are discussed in Section 10.1. The basic fault-tolerance metrics are discussed in [69] but they were not considered in the papers listed here.

Measurement setup and the definition of metrics vary between research groups and this complicates comparison. Third-party reference points are used in [81, 145, 151]. In other cases, comparison is practically always done between in-house developed approaches which are not well documented in the publication.

3.2.3 Test cases

Majority of publications use synthetic traffic, such as uniform random traffic, which is a valid approach on the first steps of design. However, author encourages systematically evaluating a large set of (representative) traffic parameters. Actual applications give the best accuracy but their traffic profiles are also suitable. For example, the profiles of [22] have been used by many researchers. Unfortunately, applications are rare used, in 40% and 30% of cases in Tables 2 & 3 and Tables 4 & 5, respectively.

Several cases are needed to make general conclusions. The listed test case counts for the synthetic cases are merely suggestive since it is not clear when the change in some parameter actually defines a new test case. Most studies use only limited number traffic scenarios, 3 on average. The largest synthetic test set, 30 random task

graphs, is in [88] whereas application sets with 4 [155] and 5 cases [151] have been reported.

Different scenarios are often modeled with some sort of traffic generator (*tg*), 43% of cases omitting the dependencies between tasks and 27% considering them. Sometimes the traffic generators are used to mimic the real applications but without performing actual computation.

Test case applications can be divided into computation kernels and full applications. Examples in the first category include image binarization, segmentation, and smoothing, IIR filter, FFT, DCT, dot product, vector sum, and matrix multiplication. Unfortunately, these applications are not realistic for a billion transistor multi-processor SoC (MPSoC). In addition, those applications represent too fine grained parallelism -short computation periods and frequent communication - which is out of the projected NoC scope.

The latter group of applications includes video encoders (H.263, MPEG-2, MPEG-4) and decoders (motion-JPEG, MPEG-2), smart camera, OFDM, radar signal analysis, and large database processing. These give a better view to real NoC performance, but are still insufficient alone. The largest study includes 30 application traces [227] but only 1.3 cases are used on average. Application results from a physical prototype are given only in [81, 88, 114, 129, 141, 142, 181, 236, 259] and the others rely on simulation.

None has yet reported a NoC running several large applications simultaneously. This is important step to bring evaluations closer to expected usage scenario of NoC-based systems. At the same, it necessitates including such aspects as (real-time) operating system, micronetwork stack [19] and other middleware that have a profound implications on runtime, scheduling, and memory usage. Similarly, the impact of more detailed resource models has not been widely explored yet.

The last column in this part denotes how well the test cases are documented. The test cases that can be repeated based on the publication are marked with *x*. Notation (x) is used when some information is missing but rough reproduction is possible. The analytical studies are reproducible by nature. The synthetic cases without dependencies can be defined with few parameters and are therefore easiest simulation cases to reproduce. Using dependencies and applications necessitates documenting also the mapping information. Applications are generally too complex to describe briefly.

3.2.4 Prototyping

The simulation evidently leads to some simplifications and inaccuracies. Such phenomena could be avoided with real prototypes although measurements are usually more complex than with a simulator and the costs are higher. Note that a prototype offers a valuable reference point for calibrating the simulation and analytical models.

Some reported prototypes omit the computation resources or results from real applications. Two ASIC examples include the network only; SocBus with 2 [80] and Nexus with 16 terminals [153]. Slim-Spider chip [144] includes the NoC and 9 resources and the same hierarchical star was used in object recognition system [114]. FAUST chip is targeted for baseband processing and it includes 23 IPs and an asynchronous 2-D mesh NoC [141].

Synthesizable traffic generators can be utilized in simulators but also if the complete system does not fit into single chip. For example, Nostrum NoC was prototyped with 16 traffic generators [176] and an ASIC prototype of 64-node fat-tree with traffic generators was presented in [166]. By far the largest NoC chip to date is the Intel's TeraFLOPS that has an 80-node 2-D mesh [236]. The chip is fabricated at 65 nm technology, designed to runs at 4 GHz, and contains 100 million transistors occupying 275 mm^2 silicon area.

Modern FPGA devices are large enough for multiprocessor SoC implementation in the range of million gates and notably cheaper than ASICs in small series. They also allow special hardware structures such as monitors and traffic generators, which cannot be included in product ASICs. Two of the reported FPGA studies use 4 processing elements (PEs) but do not provide any application results [3, 169]. In [181], a 4x4 mesh is prototyped on FPGA and evaluated using autoindustry and telecom benchmarks from E3S set [55] as well as artificial traffic loads. Synthesizable MicroBlaze processor and 7 other PE are used for image binarization in [81].

An MPEG-2 encoder in FPGA with 8 hardwired PEs and either custom mesh or point-to-point network is presented in [142]. An FPGA-based MPEG-4 encoder with multiple Nios II processors, HW accelerators, and hierarchical bus has been presented in [P6] [129]. There can be 16 processors on a single FPGA and up to 35 processors with 23 other IPs when three FPGA boards are used. Furthermore, performance results of speechcoding kernels, such as the dot_product, IIR, vector sum with scalar multiply, are obtained from reconfigurable Maia chip [259]. The study considers

multibus, mesh, and hierarchical mesh topologies using multiple system sizes.

At the first stages, it is essential to perform verification with small systems, such as 4-16 terminals common nowadays. Nevertheless, to properly analyze NoCs, they must be utilized in their anticipated scale, which contains dozens of terminals. Note also that mere FPGA synthesis is not the same as running full applications on the FPGA (which naturally includes synthesis).

In summary, prototypes are too scarce and lack concrete results from real applications. In author's experience, FPGA prototypes are very valuable to ensure correct functionality of NoC, test the application, and include various overheads. Furthermore, the obtained (near) real-time execution of parallel applications allows longer runs and better credibility of the results. Hence, more effort towards prototyping and standardized evaluation methods in NoC community is strongly encouraged.

3.3 Representative NoC

The most remarkable observation is the lack of standardized NoC benchmarks, anecdotal nature, and limited number of results, and lack of comparison. However, more and more high-quality papers are published continuously as the research matures. Most studies cited here originate from academia but a growing interest is observed also in the industry.

Table 6 shows a representative data sheet for a contemporary average NoC and improvement suggestions. The upper part shows the data related to Tables 2 - 5 and the lower part to Table 18 To illustrate their importance, there are few properties that were not included in the previous tables, such as packet length, buffer type, and the parameters related to implementation results.

To put it simply, thorough benchmarking needs more applications, larger setups, more prototypes, larger range of evaluated parameters and configurations, better models for traffic cases and for networks (especially for NIs, network stack, and synchronization).

Table 6. *Datasheet for a stereotypical NoC and suggested research directions [P11].*

	Property	Average value	Necessary improvement
Basic properties	Switching	Packet-switched	QoS, predictability
	Topology	2-D mesh	Synthesis, IP mapping
	Links	Bidirectional	Pipelining, power reduction
	Flow control	Wormhole	QoS
	Data delivery	In-order	(Cannot be compromised)
	Virtual channels	No	Needed, e.g. 2-8 VC/port
	Routing	Deterministic	Fault-tolerance
	QoS	Mentioned	Details, overhead minimization
	Clocking	Synchronous (or GALS)	Account synchronization impact
	Network interface	Excluded	Considered
	Fault-tolerance	Neglected	Considered
	Testability	Neglected	Considered
	Evaluation	Simulation, synthesis	Analysis, prototype
	Prototype	No	Needed
	Used metrics	Area, latency	Appl. runtime, power
	Comparison	vs. in-house ref.	vs. third-party/std-reference
	Applications	Not used	Standardized, several simultaneous, modeling styles
Implementation	Flit width	32 bits	More detailed documentation
	Phit width	Same as flit	
	Packet length [flit]	Hdr:1, payload:n, tail:1	
	Buffering per router	5 x 3 x 6 flits	
	Buffer type	Based on flip-flops	
	Router area	0.14 mm^2 @ 130 nm, 15-18 kilogates	
	Router frequency	600 MHz @ 130 nm	
	Min. hdr.lat/router	5 cycles	
	Power	?	
	Results from	RTL	
	Silicon conditions	n/a	
	Wire+repeaters	n/a	

3.4 Survey of reusable hardware IP components

Intellectual property (IP) components, also called macros or cores, are reusable components offering verified and predictable functionality. A microprocessor core is a common example of reusable IP component. Designing components for reuse requires more effort than creating a component for single use only and that is sometimes used as an excuse of not designing them properly. Good design guidelines are presented in [111] and overview of the IP-based design in [240].

Selecting the most suitable IPs is a crucial step in system design. Good design tech-

Table 7. *Comparison of processor cores [P3].*

#	Processors	Data width	Area [mm^2]	Area [kilogates]	Max. freq. [MHz]	Energy [mW/MHz]	Performance [DMIPS/MHz]
1	*Cache SRAM, 4-way*	*32 KB*	*3.75*	*(361)*	*1040*	*0.51*	-
2	*Embedded DDR DRAM*	*2 MB*	*15.75*	*(1514)*	*714*	*1.0 - 2.3*	-
3	Arc 600	32b	-	27	200	0.13	-
4	Arc Turbo 186, 0.25 um	16b	-	30	100	-	0.25 MIPS/MHz
5	Arc v8 RISC, 0.25 um	8b	-	3	100	-	-
6	ARM7TDMI-S,	32b	0.62	(60)	80-100	0.39	0.9
7	ARM966E-S,	32b	2.00	(192)	180	0.7	1.1
8	COFFEE RISC	32b	1.42	(137)	200	4.77	-
9	Leon SPARC	32b	-	35	165	-	0.85
10	MC8051, 0.35 um	8b	-	10-13	100	-	-
11	MIPS32, 0-32KB cache	32b	0.8-2.5	(77)+(0-163)	160-240	1.1-2.8	1.29-1.94
12	MIPS64 5Kc, 0.13 um	64b	1.8-2.6	(333-481)	350	-	1.4
13	Open RISC 1200	32b	0.5	25	150	-	-
14	PowerPC 405	32b	1.4	(135)	266-390	1.87	1.4
15	Saturn DSP	16b	0.5	(48)	210	0.25	420 mega-MAC/s
16	TTA, small area / high-perf	32b	-	43 / 158	272 / 240	0.13 / 0.45	For 8x8 DCT: 6.6 / 13.8 Msample/s
17	XiRISC, max freq / min area / min power	32b	-	100 / 50 / 61	362 / 107 / 100	0.84 / 0.45 / 0.37	-
18	Xtensa,	32b	0.7	(67)	320	0.4	-
	AVERAGE (CPUs only)	29b	1.23 mm^2	75 kilogates	200 MHz	0.59 mW/MHz	1.05 DMIPS/MHz

(Area values in parenthesis) = converted value, assuming area of NAND2= 320F^2, where F=feature size

niques, such as documentation, commenting, code structuring and clarity, are basis for component reuse but they are not enough. Moreover, the IP should solve a general problem, be configurable for different applications, and designed to function with different technology libraries and synthesis/simulation tools [111]. Different models can be provided for synthesis and system-level design, however, their functional equivalence must be ensured. In addition, early estimates for performance and cost factors of an IP component (so called IP meta-data) are needed. The impact of verification cannot be exaggerated and it must be repeatable by the customer.

An overview of available IP components is shown in Tables 7 and 8. Details of the comparison and full citation list are presented in [P3]. Results are tabulated for 0.18μm processing technology with a few exceptions. The following parameters were gathered: data width, area, (maximum) operating frequency, energy, and performance. To ease the comparison, area results were converted to kilogates. In practice, equivalent gate count depends on the semiconductor vendor and therefore, the exact size is not always known. According to ITRS roadmaps [92], 2-input NAND (NAND2) occupies $320F^2$, where F is the minimum feature size. Equation results in

Table 8. *Comparison of accelerator IP cores [P3].*

#	Accelerator	Size / configuration	Area (memory not included)	Memory	Freq. [MHz]	Energy [mW/MHz]	Performance
1	ADPCM*	8 channels	20 kgates	624 B RAM	1	-	16-40 kb/s
2	ADPCM*	256 channels	20 kgates	8.8 KB DPRAM	2	-	16-40 kb/s
3	AES	128b key and data	173 kgates	-	125	0.45	1.6-2.3 Gb/s
4	AES	128b key module / data path	84 kgates / 310 kgates	-	400 / 266	-	34 Gb/s
5	DCT chip	8x8, 9b in, 12b out	2.16 mm^2 (208 kgates)	-	75	0.06	75 Msample/s
6	DCT	8x8, 8x9b in, 17b out	0.42 mm^2 (40 kgates)	-	40	0.22	320 Msample/s
7	DMA	32b, 5 channels	0.16 mm^2 (15 kgates)	-	200	-	-
8	DMA, 0.13 um	-	60 kgates		140	-	-
9	DWT	129 x 129, 16b	16 kagets	33.8 KB DPRAM	200	-	200 Msample/s
10	DWT	128 x 128	50 kgates	50 kB RAM	-	-	150 Msample/s
11	FFT	16b, full rate VDSL	114 kgates	37.5 KB RAM + 4.1 KB ROM	75	3.08	75 Msample/s
12	FFT	16b, 1024-point	1.35 mm^2 (130 kgates)	3.03 mm^2	113	2.3	-
13	MPEG-2 decoder	1920x1088 pixels	105 kgates	832 B RAM + 1.8 KB DPRAM	133	-	60 fps
14	MPEG-2 decoder	6 PAL/NTSC streams	110 kgates	518 B RAM + 1.7 KB DPRAM	135	-	6 x 25 fps
15	Reed-Solomon decoder	Time-division / freq.-division	39 kgates / 85 kgates	2 KB RAM / 1 KB RAM	20	2.9	160 Mb/s
16	Reed-Solomon decoder, 0.25 um	-	33 kgates	1 KB RAM	84	0.74 - 1.21	2.5 Gb/s
17	RSA, 0.25 um	1024 b	36 kgates	size unclear	66	-	25 ns for exponentiation
18	RSA, small / fast	32b, max key 8192b	20 kgates / 27 kgates	1 KB RAM	85 / 200	-	33ms / 14ms (1024b A mod C)
19	SHA-1 + MD5	FIPS-180-1, RFC 1321	27 kgates	-	166	-	SHA-1: 1.0 Gb/s, MD5: 1.3 Gb/s
20	SHA-1,	FIPS 180-1	15 kgates	-	140	-	874 Mb/s
21	Turbo codec	block size 32-432	373 kgates	4.5 KB DPRAM	171	< 23.41	81 Mb/s
22	Turbo encoder	W-CDMA, CDMA2000	20 kgates	.	200	-	13 Mb/s
23	Turbo decoder	W-CDMA, CDMA2000	60 kgates	-	150	-	10 MB/s
24	Viterbi decoder	3GPP	52 kgates	2 KB RAM	64	-	-
25	Viterbi decoder	SOVA_13	44 kgates, 0.5 mm^2	-	500	0.8	500 Mb/s
	AVERAGE	-	79 kgates, 0.92 mm^2	4.6 KB RAM + 3.8 DPRAM	144 MHz	3.29 mW/MHz	-

(Area values in parenthesis) = converted value, assuming area of NAND2= 320F^2, where F=feature size
* = ASIC technology but feature size not known

NAND2 sizes of 10.4μm^2 and 5.4μm^2 for 0.18μm and 0.13μm technologies, respectively. Converted gate count values are shown in parentheses. It is not always evident, whether published area values include routing and placement results and therefore the derived gate counts can be used only for rough comparison.

Power consumption was converted into energy per clock cycle. When power [$W =$

$J/s]$ is divided with frequency $[Hz = cycle/s]$, the result is energy per clock cycle $[J/cycle]$. Thus mW/MHz equals $nJ/cycle$. It is assumed that with $0.18\mu m$ technology, the power is dominated by dynamic power instead of leakage power. However, the effect of leakage power increases rapidly with smaller processing technologies. The average size of contemporary IPs is around $75-80$ kilogates and frequency around $140-200$ MHz for 0.18 μm technology. The average size is in accordance with the projections of [232]. The networks studied in this thesis are meant for interconnecting IP components similar to those in Tables 7 and 8.

4. BASICS OF COMMUNICATION NETWORK EVALUATION

This chapter presents the basics of benchmarking and communication network evaluation, namely implementation-independent comparison of topologies, data traffic modeling, and simulation-based methodologies in general. Moreover, general guidelines for network benchmarking are given.

4.1 Introduction to benchmarking

A benchmark in every-day language is a point of reference for measurements. A common benchmark set is required for fair and thorough comparison of different approaches. It increases scientific credibility as the new claims can be reproduced and checked by other researchers. Furthermore, benchmarks help to prune the design space. System designer can concentrate on approaches that are *likely* to perform well based on the existing benchmark scores. Benchmark that best resembles the targeted application (domain) is of special interest. Robert Colwell, Intel's chief IA32 architect through the Pentium II, III, and 4 microprocessors, captured quite well the essence of benchmarking in [46]:

> "It is dangerous to use benchmarks in designing new machines, but not as dangerous as *not* using them."

The point is that benchmarks are necessary for quantitative analysis but they must be selected with great care.

Traditionally, computer-related benchmarks have measured the performance of CPU and/or its compiler [59, 73, 74, 244, 245, 251]. Nowadays they are also used in many other areas of system design, for example, computer-aided design (CAD) tools [56, 75, 118, 161, 231], as well as macro-level networks and servers [60, 102, 149]. A

benchmark-based (also called a scenario-based) system design strategy that executes a mix of concurrent applications during optimization is presented in [191].

No NoC benchmarks have been published to date. Therefore, Chapter 5 of this thesis presents a methodology for NoC benchmarking that has been missing. Network-on-chip is often utilized in an embedded system with multiple resources and, hence, the traditional single-CPU benchmarks are not applicable. Unfortunately, also the multiprocessor benchmarks are unsuitable due to difference in scale. SPEC OMP2001 [230], for example, requires 8 GB memory and UNIX or Windows operating system.

4.1.1 Benchmark classification

There are four basic benchmark categories:

1. *synthetic* which abstracts out the functional details. For example, classical CPU benchmarks Dhrystone [245] and Whetstone [244], execute instructions according to certain statistical distribution. Distribution is either artificial or measured from a real life program. With NoCs, synthetic means mimicking spatial and temporal distribution of data transfers while abstracting the program functionality. A common example is the uniform random traffic used in load-latency measurements.

   Synthetic cases are intended to debugging and isolating certain functionality. Hence, they tend to be small in size, easily ported and adopted in various environments. However, their expressiveness is limited and they may be easily tricked to achieve good results, for example, with some special compiler options.

2. *algorithm-based (or derived) kernel* includes only the kernel of the real algorithm but not the full application [245].

3. *actual applications* are functionally accurate programs, such as C compiler and analog circuit simulator. They give the best accuracy but are harder to port to different systems, require standardized input data, and their simulation is usually slower than other types of benchmarks. The target application is naturally the best possible benchmark. Unfortunately actually trying out software on a prospective system generally isn't a practical option due difficulty in porting [45].

4. *combination* includes benchmarks from the three above categories. For example, a combination of small kernels and large applications is used in SPEC [244], MiBench [74], and SPLASH-2 [251]. Commercial EEMBC [59] suite includes benchmarks for many areas of embedded computing such as automotive and digital entertainment.

The benchmark categories 2-4, can be executed with varying level of optimization. In most cases, the benchmarks are complied in an automated way to the benchmarked computer but there are cases when all possible optimizations are allowed, including rewriting some parts of the codes.

A scalable benchmark can obtain a superset of the information given by any particular fixed size benchmark and, hence, remains valid for longer time [73]. As an example, Task Graphs for Free [56] is an open source approach for developing pseudo-random directed acyclic graphs for allocation and scheduling research. Note that even if graphs are (pseudo)random they must be reproducible by other researchers and easily shared to allow comparison. Stroobandt *et al.* have presented a synthetic circuit generation method for evaluating CAD tools [231]. They noted, however, that sometimes it is hard to prove that artificial cases represent properties of any known application. Synthetic cases are easier to scale than applications and their usage for benchmarking is recommended in [224, 234]. This thesis concentrates mostly on synthetic cases.

4.2 Analytical topology comparison

The main emphasis of this thesis is on simulation or execution-based evaluation. However, short discussion about analytical comparison is included for the sake of completeness.

The network properties have been studied with analytical methods in [28, 51, 136, 147, 258]. Lenoski and Weber [147] summarized the basic goals in designing an ideal scalable network

- low cost (routers and wires) that grows linearly with N_{ag}
- minimal latency independent of N_{ag}, however the best that can be achieved in practice is the growth of $log(N_{ag})$

- bisection bandwidth grows linearly with N_{ag}

Bandwidth can be scaled up by using wider links and pipelining with the expense of increased area. However, it is much harder to minimize latency, especially when accounting synchronization at clock domain boundaries. Few topologies are summarized in Table 9. Assuming uniformly distributed traffic, none of the real networks achieves all these goals but at least one of them is always violated. However, many networks are "neighbor-optimized" which means that they achieve much higher performance under appropriately localized traffic scenario. Traffic localization has no impact on frequency or wire cost but on bandwidth and hop count.

The number of links defines the maximum theoretical bandwidth but also the wiring cost when multiplied by average link length and wire bundle width. The number of router ports differentiates the single router cost if all networks are assumed to have identical buffers. The total area cost can be defined knowing the total number of routers and the cost of a single router. In general, area costs grow with N_{ag} except in crossbar and point-to-point networks.

Latency tends to increase with the system size either due to increased hop count or because the wire length affects the attainable operating frequency. These properties have non-direct impact on the system performance. For example, the latency hiding techniques, such as pre-fetching, can largely remove the impact of latency [49]. Similarly, the operating frequency is limited by the router logic instead of the wire delay in some cases. Increasing the number of segments in hierarchical buses allows high frequencies but causes larger hop counts. A fat-tree has few long links but they do not restrict the operating frequency of other links if GALS paradigm is applied inside the NoC.

The number of bisection links is related to scalability of the network. It also reflects the error-tolerance, i.e. the ability to operate in the presence of errors in wires or routers. In both cases, a large value is desired. However, error-tolerance generally requires a routing scheme that is either adaptive or that can be explicitly reconfigured (reprogrammed). Hence, a large bisection is a necessary but not adequate condition for error-tolerance. Half of the traffic goes across the bisection with spatially uniform traffic. Actually, the fraction is not exactly 0.5 but $\frac{N/2}{N-1}$ when none of the sources sends data to itself. Nevertheless, the value can be approximated with 0.5 in most cases.

Table 9. *A brief summary of network topology analysis.*

Topology	Which scalability goal violated under spatially uniform traffic			Neighbor-optimized topology	Violated goals under optimal traffic
	Bisection bandwidth	Latency (due to freq/hops)	Area cost		
Bus	b	f	-	-	b, f
Multibus	b	f	-	-	b, f
Split bus	b	f	-	x	b, f
Hier.bus (chain)	b	h or f	-	x	-
Hier.bus (tree)	b	h or f	-	x	-
Crossbar	-	f	a	-	f, a
Point-to-point	-	f	a	-	f, a
Bidir ring	b	h	-	x	-
Octagon	b	h	-	x	-
2D mesh	b	h	-	x	-
2D torus	b	h	-	x	-
Butterfly fat-tree	-	f [(1)]	a	x	f [(1)], a
Best case	-	-	-	x	-

Symbols: b =bandwidth
h= hops
f= frequency
a= area cost (routers + wires)

Notes: [(1)] Only in fully synchronous fat-tree

A bus is considered non-scalable because its bisection is constant which means that bandwidth per input terminal decreases with the number of terminals as $1/N$. A mesh is sometimes considered a "scalable" network, e.g. [132], because its bisection bandwidth increases with the number of routers. The bisection of the mesh is defined as $B_{mesh} = 2N/k$, where N equals the number of routers and k is the number of routers in one dimension [51].

For a 2-dimensional mesh, $N = k^2$ and $B_{mesh} = 2N/\sqrt{N} = 2\sqrt{N}$. Consequently, bandwidth per input terminal with uniform traffic becomes $(B_{mesh}/N)/0.5 = (2/\sqrt{N})/0.5 = 4/\sqrt{N}$, because half of the traffic crosses the bisection. Hence, it decreases with larger N, although not as rapidly as with a single bus. Bisection bandwidth grows less than linearly with network size in all networks except fully connected point-to-point and crossbar.

Table 10. *Comparison of bus and NoC. Adapted and extended from [26, 71].*

#	Single, shared bus	Pros & cons		Multi-hop network
1	Every attached unit adds parasitic capacitance, therefore electrical performance degrades.	-	+	Local, point-to-point one-way wires are not affected by the scaling system size.
2	Bus timing is difficult in a deep submicron process.	-	+	P2P wires can be pipelined because links are point-to-point.
3	Bus arbitration can become a bottleneck. The arbitration delay grows with the number of masters.	-	+	Routing decisions are distributed, if the network protocol is made non-central.
4	The bus arbiter is instance-specific [Guerrier, Bjerregaard] but can be synthesized [Salminen].	-	+	The same router may be reinstantiated for all network sizes.
5	Bus testability is problematic and slow.	-	+	Locally placed dedicated BIST is fast and offers good test coverage.
6	Bandwidth is limited and shared by all units attached.	-	+	Aggregated bandwidth scales with the network size.
7	Bus latency is wire-speed once arbiter has granted control.	+	-	Minimum delay is two routers and one link. Furthermore, internal network contention may increase latency.
8	Any bus is almost directly compatible with most available IPs, including software running on CPUs.	+	-	Bus-oriented IPs need packetization logic. SW needs clean synchronization in multiprocessor systems.
9	The concepts are simple and well understood.	+	-	System designers need reeducation for new concepts.
10	Network logic (wrappers+arbitration) rather small.	+	-	Larger network logic (routers, especially buffers)
11	Data arrives in-order.	+	-	Data may arrive out-of-order with adaptive routing.
			+	In-of-order delivery with deterministic routing guarantees in-order delivery.
12	Long links induce more bit-errors due to crosstalk.	-	+	Shorter links are less prone to crosstalk-induced errors.
13	Only one path between source and destination does not offer error tolerance.	-	+	Multiple paths increases error tolerance iff routing is adaptive or configurable.
			-	Adaptive routing requires reorder buffers at the receiver.
14	Whole link driven all the time (large capacitance) irrespective of source and destination.	-	+	Several short links are driven on given path, others are idle.
	Total +	5	10	
	Total -	9	6	
	Total	-4	4	➔ Multi-hop is better

Differences between single, shared bus and (multi-hop) network properties have been summarized in [26, 71]. Their rather strict conclusion was that it is better to use anything else than a single shared bus topology. This is perfectly valid for large systems but not a new one or surprising. The statements are shown in Table 10 with few additional comments.

Zeferino *et al.* have analyzed the switching point when a mesh-based NoC becomes preferable over a simple bus having no hierarchy [258]. The analyzed test case is total exchange which means that all N_{ag} agents send data to all other agents ($N_{ag} - 1$). They come up with an estimate that a mesh is preferable when the system sizes increases beyond 16-25 agents assuming the same frequency for all networks. If the maximum frequency is determined by signal propagation delays on wires, a mesh provides higher maximum frequency due to shorter point-to-point links.

Ideally, the speedup over the single bus equals the number of bus segments N_{seg} in hierarchical topology. However, bridges between the source and target induce extra latency but not all traffic crosses bridges. This suggests that the switching point between mesh and hierarchical bus is higher than $16 - 25$ agents. However, the hierarchical bus topology is sensitive to locality and the latency of the bridge component. It is actually slower than a single, shared bus if many accesses across the bridges are required. Hence, the mapping of tasks onto computation resources is a critical step to keep most of the communication within a small distance [43] and to ensure load balancing.

4.3 Modeling traffic load

The offered traffic is characterized by its *spatial* (where to send) and *temporal* properties (when to send). Temporal parameters include:

- data rate, for example $bits/s$
- burstiness - defines how much the size of the transfers varies
- dependencies - define whether data is injected

    a) continuously

b) only after certain initial conditions are met, for example, enough input data has been received first.

Uniform traffic pattern is widely utilized in network studies since it is easy to generate and bounds for latency and maximum bandwidth can be obtained analytically. However, traffic tends to be localized in practice [190] and the dependencies between application tasks cause throttling [51].

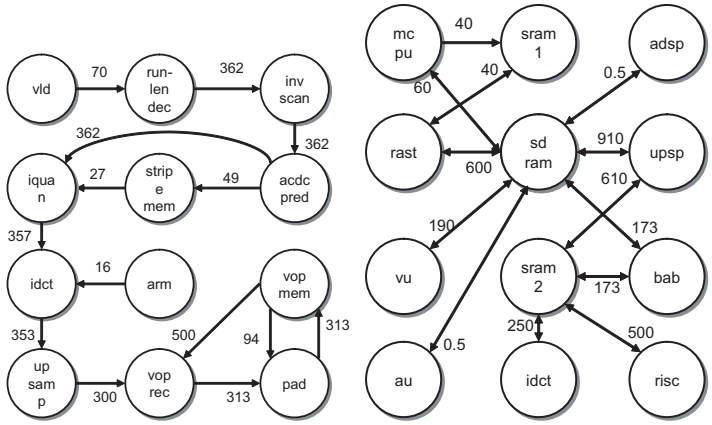
4.3.1 Data rate and spatial distribution

Fig. 20 shows four example SoC traffic models from [22]. The nodes in graphs denote PEs and memories whereas the edges denote communication between them. Sustained average data rate in MB/s is marked to each edge. These graphs and other reported state-of-the-art examples are summarized in Table 11. Graph-based traffic descriptions are emphasized here. Terms "task" and "process" are used interchangeably in this thesis.

Bidirectional edges are assumed to be split into two unidirectional edges, each with half of the data rate. Both the number of nodes and edges are small, 9 and 12 on average, respectively. The shown cases are among the best that are publicly available, but still unsuitable for benchmarking large systems with tens of resources.

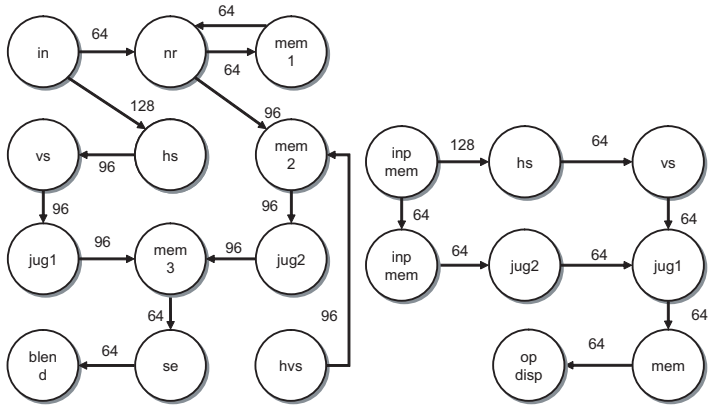
The column *Max #dst per node* shows that task nodes have only $1-3$ destinations, except SDRAM that communicates with 7 other nodes in MPEG-4 decoder. Hence, the spatial distribution of targets is far from uniform. Similar findings were reported also in [77]. Note also that the already one or few hot-modules in a wormhole-based NoC dramatically reduce network efficiency and cause an unfair allocation of system resources. For example, Walter *et al.* demonstrated that a single hot-module can destroy the performance of the entire SoC, even if network resources are over-provisioned but no access regulation is utilized [241]. Hence, this phenomenon must be accounted for in SoC architectures and applications to fully exploit the potential of NoCs.

Total emitted data rates range from 49 MB/s to over 3.5 GB/s, being about 1 GB/s on average. Both maximum and average data rates per node are also given. Ejected data rate denotes how much data is targeted to a single destination. There is large variance between data rates of the nodes. Ratio between largest and smallest sent



(a) Video Object Plane Decoder (VOPD).

(b) MPEG-4 decoder.



(c) Multi-Window Displayer (MWD).

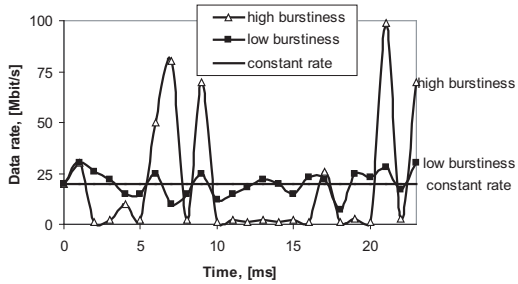
(d) Picture-in-picture (PIP).

Fig. 20. *Four examples of traffic profiles from [22]. Nodes represent SoC resources and arrows show the sustained traffic rate in Mbytes/s.*

data rate per node varies from $3x-4x$ (MWD, PIP, filter) to $750x-3000x$ (MPEG-4 dec). Uneven data rates were observed also in [228]. Maximum data rates are listed because the node that either emits or ejects the largest amount of data (a hot-spot) sets minimum bound for the frequency of network terminals.

Table 11. *Examples of reported traffic loads in multiprocessor applications.*

Author	Ref.	Application	#nodes	#edges	Max #dst per node	Data rate [MB/s] per node			Total emitted data rate [MB/s]	Notes
						eject max	emit max	emit avg		
Bertozzi	[22]	VOP-dec	12	14	2	813	594	290	3 478	-
Bertozzi	[22]	MPEG-4 dec	12	16	7	967	1 580	301	3 607	-
Bertozzi	[22]	MWD	12	14	2	192	192	93	1 120	-
Bertozzi	[22]	PIP	8	9	2	128	192	72	576	-
Chang	[38]	MPEG-4 dec	6	9	2	131	90	23	140	-
Holma	[85]	M-JPEG	10	12	3	42	42	25	254	16CIF@30fps
Leroy	[148]	MPEG-2	8	10	2	49	15	6	49	4CIF@30fps, traffic to/from ext. mem excluded
Murali	[173]	filter	7	9	3	200	225	116	810	-
Kogel	[117]	pixel	-	-	-	-	-	-	405	+ locally 260 MB/s
Kogel	[117]	IPv4 fwd	-	-	-	-	-	-	410	depends on topology
Kulmala	[131]	MPEG-4 enc	-	-	-	-	-	-	230	1020p@30fps, SW only
Kulmala	[131]								558	as above + HW accelerated
Xu	[254]	H.264 dec.	-	-	-	-	-	-	842	HDTV@150 fps
AVG	-	-	9.4	11.6	2.9	315	366	116	960	-
MAX	-	-	12	16	7	967	1 580	301	3 607	-



(a) Examples of varying burstiness.

Fig. 21. *Traffic examples with different level of burstiness. The average data rate is the same in three all cases.*

4.3.2 Burstiness

The above examples report the average throughput values. However, the level of *burstiness*, or bits per unit time, is not constant and varies from one cycle to the next [50, 77, 228, 234]. Bursty traffic injects occasionally large data amounts (bursts) instead of injecting data continuously with constant rate. This complicates the optimization of NoC as it should perform well with a mixture of very short and very long messages.

Fig. 21 illustrates the concept with three traffic scenarios which have the same average data rate their burstiness varies. The large spikes at data rate may momentarily saturate the network and hence increase the latency of certain transfers. Burstiness affects the runtime (up to 25% in [234]) also in cases where network offers adequate bandwidth.

4.3.3 Communication-to-computation ratio

Communication-to-computation ratio is sometimes used to describe and design parallel applications [50,237] [1] or computers [23,200]. Small communication/computation ratio values mean that communication demands are low which favors parallelization.

For example, ratio $0.5B/op$ means that 1 byte of data is sent for each 2 executed

[1] Some sources, for example [237], use inverse ratio which means that acronym *CCR* cannot be used.

operations. This format is recommended since it is easy to convert into data rate B/s when the duration of one operation is known. In contrast, a unitless ratio, derived from communication and computation times [237], is based on the algorithm, architecture, and their mapping. It cannot be directly utilized for modeling the same application with different mapping.

In traditional multiprocessor systems, communication is relatively slow compared to computation. For example, high-performance computer architectures offered an average ratio 0.06 $B/flop$ (including MPI overheads) [200] and $0.07-0.59$ $B/flop$ [23], where $flop$ denotes a floating point operation.

Therefore, the parallel algorithms try to minimize the communication. SPLASH-2 multiprocessor benchmark set has comm-to-comp ratios in the range of $0.01-0.26B/flop$ [49] and $0.5-3.1B/flop$ [251] but values are heavily influenced by the utilized cache configuration. Distributed memory parallel computer with explicit message-passing was studied in [50]. The average ratio was $0.3B/flop$ and maximum $1.9B/flop$.

Compared to large-scale parallel computers, MPSoC offers higher relative network capacity since the network can operate with the same frequency as the PEs. However, equally low ratios were reported for IEEE 802.11 MAC processor design [164]. The functions executed on SW had ratios $0.01-0.07B/cycle$, and those on HW had $0.06-0.38B/cycle$. A bit higher ratio of $0.1B/cycle$ for MJPEG application can be obtained from [85]. There, the ratio is lowered by the overhead of operating system and inter-process communication routines. In [160], the ratio during motion estimation becomes $0.03-0.06B/op$. Unfortunately, no other examples for SoCs were found at the time of writing. The common denominator here is that when the actual computation is performed, traffic demands are low. Hence, it is difficult to generate very stringent traffic scenarios with fully functional models compared to purely abstract models.

4.3.4 Traffic generation

Benchmarking a multiprocessor system using multiple instruction set simulators (ISSs) gives accurate results but is too slow even if the simulation is distributed to multiple computers [205]. For pure NoC benchmarking, only the external behavior of each PE needs to be modeled, i.e. the amount and timing of data transfers regardless of

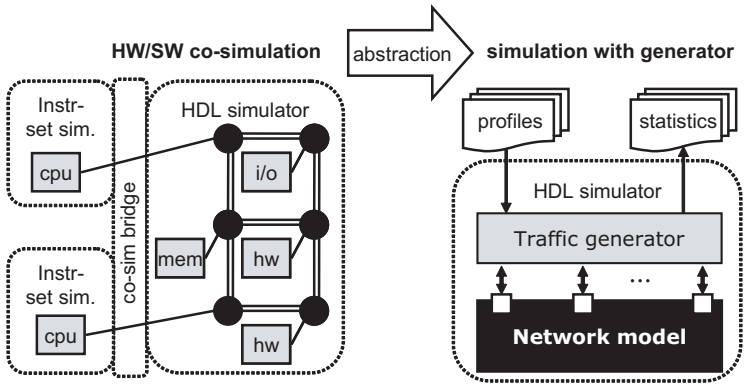


Fig. 22. *HW/SW co-simulation and simulation using a traffic generator.*

the actual values of the internal variables or the transferred data. Abstraction allows fine-tuning of traffic characteristics without the laborious optimizations on the fully-functional program models.

The basic principle is illustrated in Fig. 22. The left side shows HW/SW co-simulation that utilizes two types of simulators: an instruction set simulator for each CPU and an HDL (HW description language) simulator for other parts. The simulators communicate via a co-simulation bridge program. The right part shows an abstracted simulation setup. All resources are modeled with a single traffic generator, that injects and ejects data to/from the network according to given traffic profile, and generates statistics about performance. This method allows using a single simulation framework.

A higher abstraction level is utilized with traffic generators, which provide means to generate data transfers to the network according to a pre-defined communication profile. Hiding the internal functionality increases the simulation speed and allows researchers to more easily contribute data to the test suite. The contents of data can be freely chosen to simplify error checking, NoC debugging, and performance measurements. Actual bit patterns and toggle rates affect power consumption estimates, though. Traffic generators have been widely utilized, for example, for characterizing internet traffic in order to test and measure routers and servers [60, 102].

Transfer-independent, also called *stochastic*, traffic generator does not account dependencies of subsequent transfers but all PEs generate traffic according to a fixed probability and distribution, as in [134,173,215,218,227,254]. For example, each PE

transfers a random number of data words to all other PEs with uniform probability regardless of how it receives data from others [175].

More realistic traffic is generated by considering the dependencies between the transfers, in other words generating a *transfer-dependent*, also called *reactive* communication profile, for example [P2] [77, 121, 135, 151, 159, 247]. Small runtime estimation errors have been reported [135, 159]. Dependencies make the profile partially ordered, i.e. tasks are not executed before they have received their input data.

Functional accuracy combined with fast simulation can be obtained without use of ISSs. Applications can be run on a host simulation computer (native execution) and annotating wait statements to model the execution time. Fine-grained annotation on the basic block (assembly language) level has obtained both accurate and fast simulation [12, 31]. In this scheme, however, the application source codes must be distributed in the benchmark set which will limit the contributions from industry. Error detection and detailed performance measuring may also be laborious to implement with a real application. For example, checking real data values, say pixel values, is harder than spotting the discontinuities in a sequence of running integer numbers.

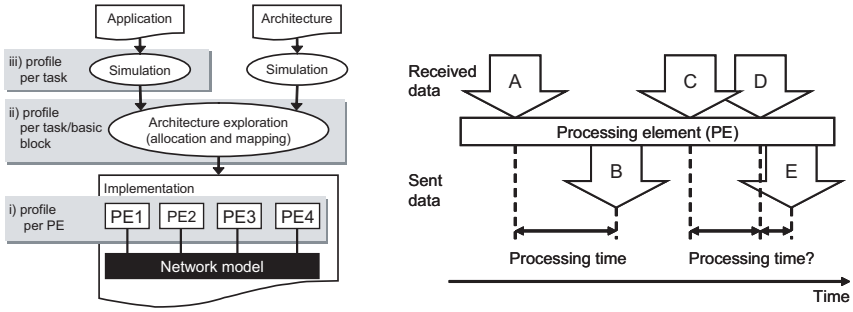
4.3.5 Traffic profile capture

Traffic characterization is required in order to utilize traffic generators. Static analysis prior to compilation is difficult and/or inaccurate since the program flow and execution time practically always depend on the input data set. For example, unbounded loops in SW are problematic. Static analysis, or "educated guess", is of course mandatory if the application is not available yet.

Application profiling is performed by running application in simulator or in real HW with multiple input data sets and collecting execution traces. However, the architecture or mapping are not yet fixed but designed with aid of profiled information. Fig. 23(a) illustrates three possible ways to profile the execution:

i) profiling is done after the application mapping by tracing the communication between PEs. The profile captures the traffic generated by all the tasks mapped on a given PE. Transfer-independent, stochastic, generator fall into this class.

ii) profile the application execution on each possible PE. The granularity level varies from basic block of assembly instructions [31] to complete tasks.



(a) Two possible traffic profiling points.

(b) Potential ambiguity in capturing a profile-per-PE (case *i*)).

Fig. 23. *Creating a traffic profile.*

iii) profile the application once at task-level on a reference PE. The performance of individual PEs is given relative to a reference PE. PE characterization is independent of the application at hand, and performed while constructing the PE library, prior to exploration.

Short example outcomes of three profiling styles:

i) PE1 sends with probability of 5% to PE2, with probability 7% to PE3, ...

ii) application task $t1$ takes $n1$ clock cycles on PE1, $n2$ cycles on PE2, ...

iii) application task $t1$ executes $o1$ operations on reference PE, $t2$ takes $o2$, ...

Method *i* is limited to exploring the NoC and the locations of PEs. Every time the application, mapping, scheduling, or PE allocation is changed, the profiling must be performed again. This is not recommended approach. By utilizing the method *ii* or *iii*, the same communication profile is applicable throughout the exploration for all application mappings, schedules, and PE allocations. Furthermore, the same communication profile remains valid for all NoC architectures. In *ii* this requires that the impact of the NoC (stalls and throttling) can be separated, for example by overclocking the NoC, as in [131].

Fig. 23(b) illustrates a further problem arising in method *i* that views the application as a black-box. In simple case, the PE receives message *A*, starts processing, and emits message *B*. The definition of processing time is more difficult with incoming

messages C and D. It is not possible to determine whether the processing starts after C, after D, or only when both of them have been received. In addition, all three profiling styles require knowledge, whether the processing starts when the first word or when the whole message arrives. Therefore, knowing the internal behavior of the tasks is necessary to solve such ambiguities.

In [P7], a data-parallel video encoder [105] has been manually profiled to derive a parameterizable transfer-dependent model for Transaction Generator. The obtained simulation results are presented also in Section 10.5.1.

Manual profiling of video encoder application was somewhat tedious and time-consuming. Therefore, MAC protocol for proprietary WLAN described in UML 2.0 [123] was profiled by linking special functions to the automatically generated C code. The application was run on FPGA and the collected execution traces were transformed automatically into a task graph model (18 tasks) with scripts in few minutes which is comparable time with results given in [159]. The automated profiling is an integral part of Koski design flow [104]. Other studied publications do not report time and effort needed for the profiling.

4.4 General guidelines for NoC benchmarking

This Section provides general guidelines for evaluation, metric selection, reporting in general. Table 12 provides short general guidelines to alleviate the observed shortcomings in NoC comparison. Few common issues were inspired by [5,13,15,65,224] although those articles deal with different problem domains; simulating and evaluating a mobile ad-hoc network (manet), job scheduling policies, computer architectures, and heuristic algorithms, respectively.

The guidelines given here might seem obvious and self-evident at the first look. However, in practice they are not; judging by [5,65] and the articles cited in Tables 2 - 5. Although some of them are usually considered, too many are not documented accurately. A reader cannot be sure about validity unless the setup and assumptions are explicitly stated, which is rare. First step to improve the situation is the brief checklist given in Table 12. Second, the research community should evaluate the guidelines, and then thrive for their wide adoption, as in [P9].

The proposed guidelines are divided into 4 parts: *workload, system modeling, mea-*

Table 12. *Simulation guidelines for NoC evaluation [P10].*

W.r.t.	#	Guidelines
Workload	1.1	Use several cases, ensure that they are representative
	1.2	Validate statistical workload models against real applications
	1.3	Ensure that cases are different enough and not biased towards some property
	1.4	Document properly if there are limiting assumptions about the usage scenario
	1.5	Include variable bit rate and bursty traffic
	1.6	Scale the load with great care to avoid distortion of important properties
	1.7	Use documented, freely available benchmark workloads when available
System model	2.1	Evaluate many system sizes and configurations
	2.2	Validate models against data from external source (other publications etc.)
	2.3	Validate the analytical and simulation models against real implementations
	2.4	Include various overheads (e.g. synchronization or interrupt latency)
Measurements	3.1	Document all settings and assumptions to enable repeatability
	3.2	Determine the number of required independent runs to gain statistical validity
	3.3	Address the sources of randomness to ensure simulation run independence
	3.4	Perform sensitivity analysis to identify the significance of a certain parameter
	3.5	Carefully select several metrics and document them
	3.6	Specify units and ratios explicitly. Use (de-facto) standard units when possible.
	3.7	Discard warm-up period and saturated workloads from the results
	3.8	Use long workloads and (simulation) runs
	3.9	Use "infinite" source queue in load-latency measurements to avoid self-throttling
Conclusions	4.1	Account for estimation/simulation errors in comparison
	4.2	Compare and contrast the results to the state-of-the-art
	4.3	Repeat the evaluations on a reference system
	4.4	Search for trends and correlation in the obtained results
	4.5	Do not use speedup from a small system directly to estimate the bigger ones

surement, and *conclusions*. Each part and NoC benchmarking metrics are discussed separately next.

4.4.1 Workload

Workload refers to applications or their models that are used during the evaluation. Several cases are needed to obtain an overall view on the behavior of a system and they must be representative for the target domain, for example multiprocessor SoC targeted for mobile devices. In addition to their number, the cases must present different characteristics to justify their inclusion in the test set. If the cases are biased, for example assuming very small transfer sizes, that must be explicitly documented and motivated. In the NoC domain, the traffic cases differ, for example, in their offered

load, locality, burstiness, and latency tolerance.

Compared to real applications, traffic generators offer speedup for simulation and easier modification. Hence, they are suitable for covering large application space with proper scaling. However, care must be taken when modifying the basic parameters not to distort the important properties which would give misleading results [65]. The models must be validated against real applications to identify the reasonable values (e.g. offered load and spatial distribution). Common benchmarks offer partial solution to the problem as they greatly simplify documenting the workload.

4.4.2 System model

It is practically impossible to include all the minor details to simulation and analysis which evidently leads to estimation errors. Therefore, the complete simulation (developed protocol, traffic, environment model, and usage scenario) has to be validated against a real-world implementation, analytical models, or protocol specifications. Less precise models are viable during early concept development but they can be further refined later [5]. This applies, for example, to models of the NoC (transaction-level, cycle-accurate models, area and wire estimates) as well as the environment (traffic generation, third-party IP). The utilized modeling techniques must be configurable to cover reasonable design space. Comparing just the analytical and in-house developed simulation models is dangerous as they both might have the same erroneous assumptions [65]. Therefore, validation must be performed against external, independent data, when possible. For example, the accuracy of Transaction Generator used in this thesis was compared against FPGA execution in addition to simulation.

If the models turn out to be inaccurate, they must be tuned and usually that means adding more details. In MPSoC, such details include, for example, interrupt latency at the receiver, data synchronization at the clock boundary, context switch overhead, DMA transfers, crosstalk, and chip layout. Some of these were introduced in Section 2.5. Estimates based on early models need revision during the course of research. For example, the area overhead of the NoC was first estimated to be $1-2\%$ [188] but after more research the estimates were updated to range $9-45\%$ [190]. The analytical studies in Tables 4 and 5 are well motivated and seem intuitively correct but no results were found on the validation of the models used. In [142], performance

was estimated analytically and with measurements on an FPGA prototype.

4.4.3 Measurement

Other researchers must be able to repeat the experiments and hence justify the findings. Very detailed description or the actual setting files can be published, for example, in research group's web pages so that they are freely accessible.

Several independent runs ensure statistical reliability when the environment models and synthesis tools apply some pseudo-random techniques. The number of runs can be determined empirically. For example, repeat executions until further runs change the average at most 2%. Independence can be achieved, for example, by explicitly varying the seed value for the random number generator. Sensitivity analysis identifies the most important parameters - those that produce the greatest variation in results. Proper selection of values for these parameters is a critical step at benchmarking and requires thorough motivation and background work.

Certain phenomena appear only with long enough simulation or execution run. For example, the saturation of a NoC is not visible yet when only a couple of packets have been sent. The warmup period at the beginning must be discarded from the results to measure steady-state behavior. For example, the first few packets experience zero-load conditions because NoC congestion arises only after several packets have been injected. Therefore, inclusion of first packets distorts the average values. The trace length can be determined similarly to the number or runs. The required trace length is studied in [77] and in Section 6.5. The traffic sources must not be stalled during latency measurements and hence a special source queue is required (see Chapter 6 for further details). However, such queue should be omitted when running application benchmarks.

The quantities related to implementation must be documented, see the lower part of Table 6. Otherwise, a dishonest evaluator gains exceptionally good results, for example, by using very wide flits while neglecting the area and power consumption.

The basic metrics and estimated PE utilization can be obtained automatically by the generic benchmark tool. To obtain greater insight, they are augmented with NoC-specific metrics with separate monitors, for example the fairness of flow control and the utilization of links and buffers. This way the designer can spot the bottlenecks

and design errors in the NoC.

4.4.4 Concluding the findings

Benchmarking must develop a method to gain insight from the numbers (results) and guarantee the quality of those numbers [45, 47]. Hence, the obtained results must be interpreted, compared to others, and generalized when appropriate. It is very beneficial to identify systematic trends from the results. Especially when the source of the phenomena is analyzed. At the same time, one must avoid generalizations without a large coverage of the parameter space. For example, deriving the speedup from a small system and extrapolating it to larger, is a common source of errors [65].

First of all, the comparison must take estimation errors of models into account. For example, any runtime difference less than estimation error, say 5%, is negligible due to approximations in traffic and NoC models. The fundamental problem is how to determine the estimation error beforehand. The only option seems to be using models that have been validated and verified earlier and assume that the accuracy remains static. In addition to min/avg/max values, the results can make use of confidence intervals [234] and percentiles. For example, the latency of 8-word packets is less than *30 cycles* in 95% and *40 cycles* in 99% of the cases.

An important, although difficult, issue is to compare the obtained results to the state-of-the-art in the field. Direct comparison, like *"average latency is 5 cycles lower than in [ref. X]"*, is more difficult but also more beneficial. Care must be taken to use exactly the same input values (e.g. common test cases) and key parameters. One option is to re-implement novel ideas from the literature such as routing or coding schemes. First, this either ensures the validity of the approach or reveals deficiencies that were not covered in the original article. Second, this allows just comparison, as many parameters, such as processing technology, can be held constant.

Third-party reference points are very valuable but currently, most NoCs and their test cases are proprietary and unavailable for public evaluation. For example, freely available NoC (simulator), such as Hermes [54], FPGA-NoC[2], Noxim[3], and Netmaker[4], as a common reference are beneficial. Furthermore, researchers must aim for a wide

[2] http://www.da.isy.liu.se/research/soc/fpganoc/

[3] http://noxim.sourceforge.net/

[4] http://www-dyn.cl.cam.ac.uk/ rdm34/wiki/

spectrum of results instead of common practice of reporting just a few, non-related values. For example, measuring area with sensitivity analysis over multiple router configurations.

4.4.5 Metrics

All the metrics and units must be defined explicitly (SI units are preferred). Application runtime is the most obvious cost metric at the system-level. The runtime is hard to measure without real applications or transfer-dependent traffic models. Latency can be usually measured more easily and it affects the runtime. However, the relation is not straightforward; for example, various latency hiding techniques can remove the impact of latency as long as it is below certain threshold. Therefore, smaller latency does not necessarily translate to smaller runtime; see for example [58]. However, it is very likely that shorter latency will not cause longer runtime either when the possible scheduling anomalies are neglected. Therefore, achieving the same latency with fewer resources (area, power) is a viable target for optimization. Similarly, the bandwidth of the NoC is only an indirect clue of performance [8].

Table 13 summarizes the most important metrics for NoC evaluation. Timing-related issues can be measured with four metrics and implementation cost with area and power/energy. They all can be measured at different levels (e.g. messages vs. packets, network only vs. the whole chip) and it is crucial to document which one is selected. To name a few for simplicity, the author argues that the three most important metrics of the system are *application runtime, silicon area, and power consumption*.

The above discussion deals with the network performance only which indirectly affects to whole system. For example, the area and power consumption of the PEs and memories may be such that the differences between compared NoCs become negligible. In [8], mesh NoC consumes over $5x$ power compared to multi-layer bus. However considering the PEs, the *energy* consumption of a mesh-based system is similar or less because its runtime is shorter.

4.4.6 Cost function

This thesis makes a clear distinction that the performance is to be maximized and the cost minimized. The *overall performance* or *merit* is a combination of several

Table 13. *Metrics for NoC evaluation. The values for all except throughput should be minimized.*

Related to	Metric	Symbol	Unit a)	Unit b)	Unit c)	Target value
Time	Runtime	t	Seconds, *s*	Clock cycles	-	min
	Latency	t	Seconds, *s*	Clock cycles	-	min
	Throughput rate	r	Case dependent, e.g.	Bytes/s	frames/s	max
	Deadline violations	v	Number of violations	Pass/no-pass	-	min
Silicon cost	Area	A	Millimeter squared, mm^2	Kilogates	FPGA resources	min
Energy	Power	P	Watt, *W*	-	-	min
	Energy	E	Joule, *J*	-	-	min
Error tolerance	Change in one of the above	Δ<x>	Cost increase as a function of error probability			min

factors.

A solution to a multi-objective optimization problem cannot be achieved by considering the design objectives separately. However, there is a set of acceptable trade-off optimal solutions. Each of these solutions is *Pareto-optimal* in the multidimensional space. It means that there is no solution that can improve at least one of the objectives without degradation any other objective. For example, in the case of timing and power, the solution is Pareto-optimal if no other configuration has a lower power for a given timing. Since the Pareto optimum gives usually more than one solution, the designer has to decide which one is selected for the final implementation. Another way is to define a combined objective function - a cost function - which collects the objectives (that are often opposing) to one function. [105]

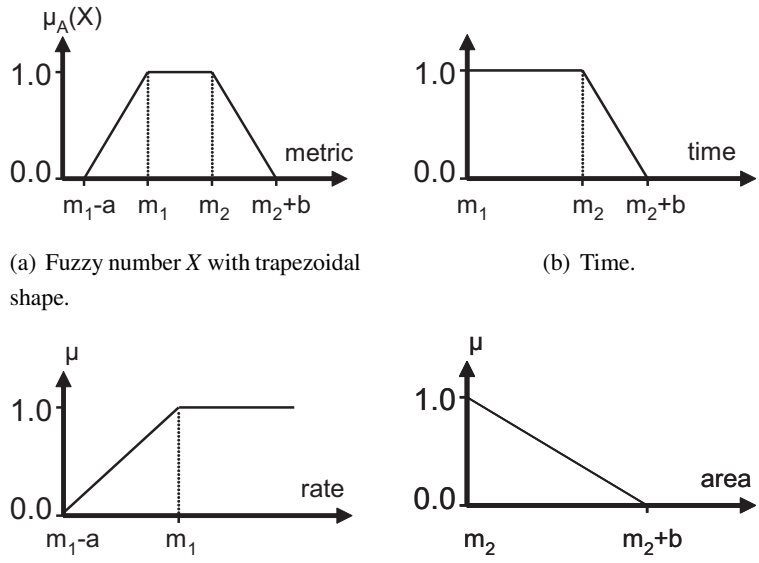
This thesis considers two ways to combine the individual cost components: weighted product, and fuzzy multicriteria analysis (MCA). Weighted product can be defined as

$$cost_{tot} = \prod cost_i^{w_i}, \tag{4}$$

where $cost_i$ denotes the value of criterion i (area, runtime etc.) and w_i its weigth. The cost value is either absolute (e.g. 500 kilogates) or relative (e.g. relative area of 0.75, unitless value). The benefit of relative values is that minimum and/or maximum bounds can be chosen freely but absolute values enable better comparison between papers. A good study of using weighted product in deep-submicron CMOS design and synthesis can be found in [220].

Unlike PCs, embedded systems are targeted for a certain performance level in a narrow application domain instead of maximum performance in general purpose computing. For example, consider two HW implementations for video encoding. The first achieves 35 $frames/s$ and the other 25 $frames/s$. However, the higher speed offers no performance gain if frame rate is limited by other parts of the system (e.g. camera).

Therefore, *performance with given constraint* is more natural objective. In other words, given a strict upper (lower) bound, find the solution that meets that while minimizing other factors. Hence, the solution is selected from the Pareto-optimal set. For example, try to find a NoC with smallest area while achieving certain throughput for test case *Foobar*. Constraints related to timing are especially important when designing real-time systems in which the violation of (hard) deadlines is hazardous.



(a) Fuzzy number X with trapezoidal shape.

(b) Time.

(c) Processing rate.

(d) Area.

Fig. 24. *Using fuzzy numbers for representing the design objectives. The membership function, shown on Y-axis, is interpreted as relative goodness. Value* 1.0 *means best solution(s) and value* 0.0 *unacceptable.*

Fuzzy numbers can be used for expressing such constraining conditions [99]. A membership function μ_A quantifies the grade of membership of the elements x to the fundamental set X. When $\mu_A(x) = 0.0$, the member x is not included in the given set X and value 1.0 describes a fully included member. Values strictly between 0 and 1 characterize the fuzzy members. In this case, the membership can be interpreted as relative performance (or merit, quality, applicability, goodness). Then, value 1 means the best solutions that meet the constraint whereas unacceptable solutions have value 0.

Fig. 24(a) shows a fuzzy set X that has a trapezoidal shape. Such a shape can described by four parameters $[m_1, m_2, a, b]$, where m_1 and m_2 describe the beginning and end points of full membership, respectively. Parameters a and b denote the slopes. Strict limits are defined by setting $a = 0$ or $b = 0$.

Fig. 24(b) and 24(c) show examples for runtime and processing rate, respectively. Both metrics have certain threshold (maximum runtime m_2, minimum rate m_1) and values meeting that have membership equal to 1. There are also cases that have lim-

ited applicability although they do not fully meet the criteria. Note that in Fig. 24(c) having rate larger than m_1 does not increase the applicability and m_2 is infinite.

Fig. 24(d) shows example of applying fuzzy set for silicon area. Zero area is desired, $m_1 = a = 0$, and any increase in area reduces the applicability ($m_2 = 0$). Architecture cannot be applied at all if it does not fit. The upper bound b depends, for example, on available packaging or selected FPGA device.

Once all the fuzzy sets are defined, fuzzy multicriteria analysis (MCA) combines several metrics:

$$\mu_{MCA} = \sum \mu_i^{w_i}, \tag{5}$$

where μ_i denotes the membership of criterion i and w_i its weight. The cost becomes the inverse of membership aggregation:

$$cost_{MCA} = \frac{1}{\mu_{MCA}}. \tag{6}$$

5. PROPOSED NOC BENCHMARKING METHODOLOGY

This Chapter presents a methodology for benchmarking NoCs, an XML (extensible markup language) file format for delivering the benchmarks, and discussion about the implementation of the methodology.

At first, the emphasis is on simulation-based evaluation of multimedia applications, such as processing of streaming video. It is characteristic to streaming applications that a long sequence of data items flows through a stable set of computation steps (tasks) with only occasional control messaging and branching. Each task waits for the data items, processes them, and outputs the results to the next task.

Fig. 25 depicts the utilized NoC system model. Some of the basic concepts of the system model and a very similar format were previously presented in the Koski design automation tool set [P2] [105]. The model and the corresponding XSM (XML system model) description are divided into four main sections:

1. *Application* defines the workload in terms of computation and communication
2. *Mapping* binds the application tasks to the resources
3. *Platform* defines the resources and the NoC interconnecting them
4. *Measurement* section defines how to perform the evaluation, for example metrics and simulation length.

Separation to distinct parts - function vs. architecture and computation vs. communication - is necessary to handle complex architectures and applications [217]. Orthogonality allows exercising or modifying one of the components, while keeping the rest at their previous (default) configuration. Thus, the mapping, for example, may be varied without touching the application or hardware models. Similarly, one can describe the particular NoC once but change the application when needed.

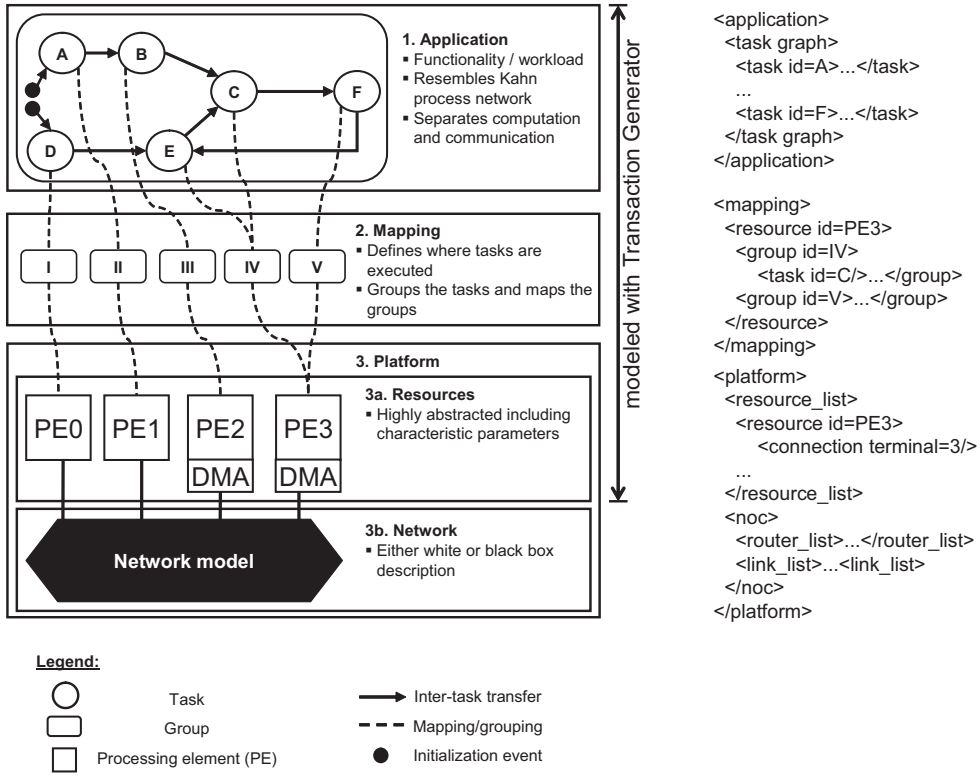


Fig. 25. *Conceptual view of the utilized system model and pseudo-XML.*

Fig. 25 depicts the proposed NoC system model. It shows a simple task graph consisting of six tasks ($A-F$) and two triggering events, which trigger tasks A and D. The tasks are grouped into five groups ($I-V$) that are mapped onto the platform, namely to four processing elements ($PE0-PE3$). Assuming that the events in this application are periodical and their time interval is set properly, all the PEs 0-3 can execute tasks simultaneously.

The benchmark set covers the three uppermost sections of Fig. 25 (application, mapping, computation resources) and measurement settings but leaves the network definition to the NoC designer/evaluator. Message-passing communication paradigm is assumed at this stage of research and the PEs are assumed to have local, private memories large enough to store all program code and processed data. However, no assumptions are made about the abstraction level of the NoC. The XML is given as input to a traffic generator, such as [P2] [68], which will be used during the simulation.

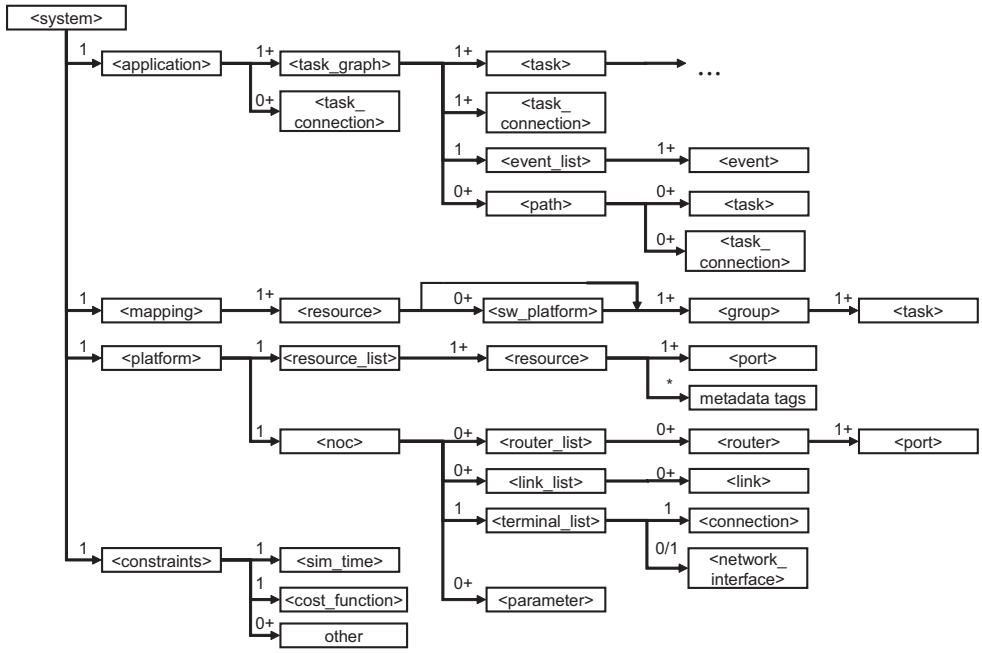


Fig. 26. *Major tags in XML system model. The numbers show the minimum number of occurrences of each tag. Each of the four major sections occurs exactly once. Task description is shown in Fig. 27.*

The traffic generator will inject/eject traffic to/from the NoC, calculate statistics (PE utilization, data latencies etc.), and detect transmission errors (corrupted, dropped, out-of-order data etc.). Design mistakes naturally cause errors but they could be also injected artificially to model soft errors. Error detection is especially useful when evaluating work-in-progress or third-party NoCs.

Fig. 26 shows the major tags and their relations in the proposed XML description. The top level tag is called *system* and it includes four tags under it. The first of those, *application*, includes two and so on. The numbers show the minimum number of occurrences of each tag. A number followed by the plus sign (+) denotes that multiple instances of the same tag are allowed. In those cases, the upper bound is case-dependent.

The major sections of the XML model are introduced next.

5.1 Application model

An application includes a task graph that models the computational load as well as the induced communication. In the basic case, there is only one task graph. The model includes dependencies, timing, destinations, and size of data transfers. In a task graph:

1. *task* nodes model the computation. The execution time is derived from the task's operation count and the properties of the PE executing that task. No actual computation is performed during the simulation, only the external behavior of application tasks is modeled. (Actual computation is possible in [P2], but not necessary at the first stage of NoC benchmarking.)

2. *connections* between nodes carry out the communication. Tasks nodes communicate via connection channels (directed edges) that carry the data tokens between tasks. Channels are attached to the tasks' ports. Bidirectional ports are not allowed.

3. *trigger events* generate stimuli to the tasks

4. *path definitions* are used for measurements, especially when real-time constraints are present.

Utilized Model of Computation (MoC) is such that vertices represent computation tasks and edges represent communication channels. The chosen model is reactive due to dependencies. This means that when a certain transfer is delayed, all the tasks (and transfers) depending on that transfer are also delayed. For example, a task that models memory does not send any data before it is requested to do so. Such dependencies capture *bursty* injection of traffic in addition to *throttling* where the injected data rate drops momentarily if there is a bottleneck in the system that cannot process and forward the data fast enough. An example of detecting such behavior can be found in [84]. Furthermore, this allows estimating the utilization of each PE and, hence, their power consumption.

The task set is static and no tasks are spawn during execution. Application model may include several task graphs (smaller applications) in order to model multitasking. Tasks in different graphs may be connected. Even if graphs are disjoint (no

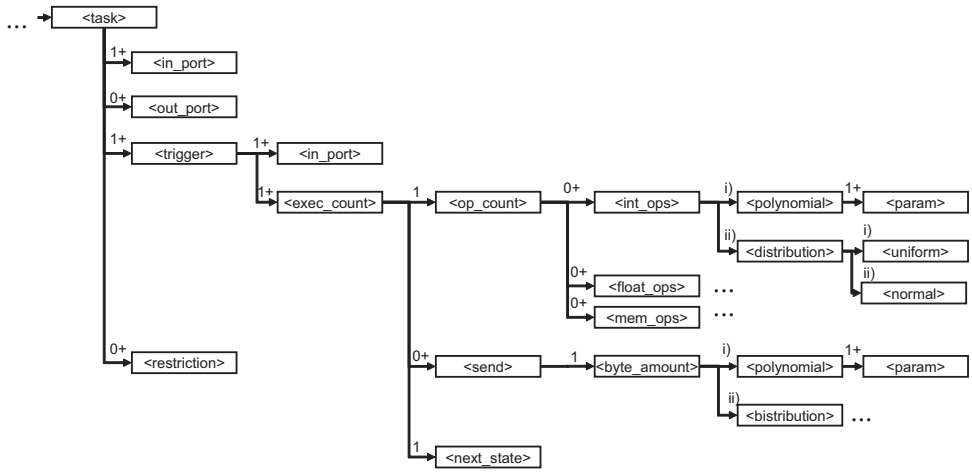


Fig. 27. *XML tags for describing the workload of application tasks.*

communication between graphs), they can still affect each other's execution times as they may compete for the shared resources, i.e. PEs and NoC. Some tasks may model memory accesses instead of computation.

5.1.1 Application tasks

Tasks are the primary means of expressing the communication and computation load. Fig. 27 shows the tags used for describing the tasks. Tasks communicate via unidirectional ports. A task is triggered for execution according to a condition that depends on the received data tokens and possibly on the internal state of the task. A task may include several behaviors but exactly one is executed at a time.

Triggering conditions have two variables: input ports and the execution history of the task. The number of ports in the task or trigger condition is not limited. Hence, all inputs of the task can be present, if needed. There are two types of dependencies on multiple input ports

1. *AND*: triggered when there is data in all inputs,
2. *OR*: triggered when any of the input receives a token.

The OR dependency distinguishes the model from the traditional Kahn Process Network (KPN). The history means here simply the count of how many times the task

has been executed. For example, a task may have a different behavior on the first execution but after that the same on all other executions. Another example could be that a task performs function *A* on every even execution count and function *B* on every odd one. In the extreme case, the task behavior is different on each execution. This is suitable for capturing a trace for highly varying tasks for which the average values are misleading. Hence, the same file format can store both the captured trace of events and the more abstract workload models.

Each execution behavior is characterized by three elements:

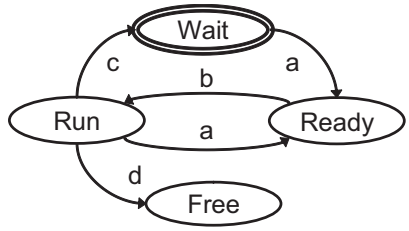
1. the operation count,
2. data amount to be sent (in bytes) and the output ports where the data will be sent, and
3. the next state of the task after execution.

The operation count and data amount are expressed with either a statistical distribution (uniform or normal), or as a polynomial function, which depends on the received data amount (a constant value is subset of the latter). Polynomial and statistical are mutually exclusive choices and hence denoted with (i) and (ii) in Fig. 27. However, the choice is done per task; certain tasks may have polynomial count/amount values and others statistical ones. The actual values are, for example, results from simulations of a virtual prototype when the application exists but educated guesses are needed in case of estimating the workload of future applications. By varying the values, the designer can search for corner cases, for example, finding the maximum allowed operation counts for certain tasks. These may serve as boundary conditions for the application developers.

There is one *send* tag for each destination. Each output is assigned a certain probability. The same output is always chosen if its probability is equal to 1.0, i.e. in 100% of cases. Smaller probability values allow a more compact model as fewer triggering conditions are needed.

Few possible example behaviors with different triggering conditions are

- Always perform the same action (same number of operations, same amount of output data, same destinations) irrespective of the execution count or input port.



a: The task has all the data for the next execution
b: Scheduler selected the task for execution
c: Fully executed but has not all data for the next execution
d: Fully executed and is not dependent on any data anymore

Fig. 28. *The possible states of an application task during simulation. It is assumed here that tasks cannot be pre-empted.*

- Operation count and data amount depend on the received data amount but not on the execution count; destinations are always the same.
- Select the output according to the input port similarly on every execution; however, the operation count and the data amount may be statistical values.
- All parameters depend on both the input port list and execution count.

Task state machine

During simulation, the internal state of a task is expressed with an execution counter and a status variable. Fig. 28 illustrates the scheduling state machine of a single task. Tasks start in state “wait”. Once a task receives enough data tokens and the triggering condition is fulfilled, it will become ready for execution. The scheduling policy of the PE defines which task from the ready list is selected for execution (edge b). Parameters related to scheduling are defined in the mapping part. After execution, the task’s next state is usually set again to "wait". It may also be "free" (as indicated in Fig. 28) which means that the task will not be executed anymore. The execution counter is incremented every time when the task leaves the state "run" (pre-emption would necessitate detecting the completion of execution - not just state change - and a new state “paused”).

Every task may have other restrictions and settings that affect the mapping and scheduling possibilities, for example, can a task be pre-empted, or to which PEs it may be

mapped. This section is included mainly for future extensions regarding design space exploration.

5.1.2 Connections

The connections simply define the structure of the task graph: which tasks communicate with each other. Their only parameters include the source and destination ports of the tasks. Each connection has exactly one source and one destination. The transmitted data token is an arbitrarily large data set. The NoC and NI settings define how many packets are needed to transmit each data token. Currently, the simulation engine assumes unbounded buffering capacity for the tokens at a connection. The designer can monitor the lengths of these token buffers and, in the future, also set limits to them. A separate source queue is not needed as in load-latency measurements.

5.1.3 Triggering events

Triggering events model the environment by providing stimulus to the task graph. In other words, they generate the input data stream that is "processed" by the application model.

Their behavior is similar to that of timers. An event fires when a given time period has passed and them it emits data token(s). Events are either *periodic* or *one-shot*. Their data emission may have a probability less than 100% if needed. This means that an event does not always send data at all although it is triggered. As mentioned, all tasks are initially in state "wait" and, therefore, at least one event is needed to trigger the execution. Currently, events do not have input ports so they cannot react to the application's outputs.

Events are special nodes because they are not mapped to any PE, and hence do not reserve resources. In other words, they do not consume "CPU-time" on processing resources or NoC capacity. However, they use regular connections to transmit their data to the tasks.

5.1.4 Real-time constraints and paths

A task graph can also include real-time constraints, for example, computation deadlines with tasks and communication deadlines with connections. In addition to single tasks (nodes) or connections (edges), *paths* that constitute several tasks may be defined.

A path is an ordered list of tasks and/or connections that is used for benchmarking. For example, path p includes tasks A, B, and C (in this order, cf. Fig. 25) and this path has a real-time constraint of 2 ms. It is measured from the triggering of task A to the completion of task C. The transaction generator will monitor if the deadlines are always met and report violations. In addition, the average runtime of the path may be also used in the cost function.

There are multiple options for choosing the start event in respect to which the real-time constraints are given. The time may be measured starting from the time instant when

- triggering condition X occurs,
- current task received all its input data,
- previous task is completed,
- current task last completed.

These conditions correspond, for example, to interrupt request from the user, operation-parallel (i.e. pipelined) execution, and jitter-aware application. The number of path definitions is not limited. Combining these constraints with other types of criteria, such as execution counts reflecting the throughput of the application, gives a good overall picture of the system's performance.

5.2 Mapping model

The task mapping model defines, on a per-PE basis, where the tasks are executed. Mapping and scheduling are steps of NoC design which deal with the implementation of applications onto the NoC architecture. Task mapping is the assignment of

application tasks to processing elements. Communication mapping is the allocation of architecture communication resources to the application transactions. Scheduling is the ordering for execution of application tasks and transactions on the assigned processing elements and NoC communication resources.

Task mapping is performed in two stages: grouping of tasks together, and mapping the groups to resources. A less expressive mapping could be done without the groups, but the chosen method allows more possibilities (including the simple mapping). If uncertain, users can have just one task per group. The main ideas in grouping are to model operating system threads and to restrict mapping exploration. Both tasks and groups may have parameters related to scheduling, such as priorities.

Group contents may be modified by automated design automation tools if allowed (*mutable="yes"*) or they can be moved elsewhere (*movable="yes"*). The former restriction can be also done for a PE and the latter for a task. Although the application is fixed in each benchmark, there are various options:

- assign fixed mapping for a NoC of a certain size, e.g. 16 terminals. Designers can easily vary NoC parameters without modifying other parts of the XML description.

- assign fixed grouping of tasks, and let the designer map them to his/her NoC freely. Note that the number of NoC terminals (i.e. PEs) can be smaller than the number of groups.

- as previous, but without groups. This option gives full freedom in mapping.

The tag *software platform* is included for future extensions for modeling the workload of the real-time operating system (RTOS) and impact of scheduling.

5.3 Platform model

The platform has two parts: resources and NoC. The resources will be defined by the benchmark set and the NoC by its designer/evaluator.

5.3.1 Resource model

Computation architecture is modeled at a very coarse level. The PEs are characterized with few parameters, such as operations-per-cycle, silicon area, power consumption. In [104], these values are stored in separate PE library but here they are written directly in the XML for simplicity. The benchmark defines for each resource

1. type: processing element (PE) or storage
2. NoC terminal where it is connected to
3. operating frequency, number of operations per cycle
4. inclusion of DMA controller
5. the associated communication overheads
6. area (mm^2 or kilogates, aspect ratio) and active/idle power consumption.

The first two belong to structural description and the next three are related to the execution time. The last one is necessary for system-level analysis, for example comparing power/energy of the whole chip instead of the NoCs alone. Similarly, PE sizes are needed to obtain estimates on layout, wire lengths, and total system area. These help to put the results from NoC into perspective since all differences between NoCs are not relevant at system-level. This is especially true for application-specific NoCs. For example, a NoC with much higher power consumption may achieve lower system energy due to shorter execution time, as in [8].

The PE model includes a task scheduler. The scheduling choices are FIFO (default choice), (non)pre-emptive static priority, and round-robin. Priorities are set at compile-time. When a task is mapped on a PE and scheduled for execution, the simulation engine calculates the associated runtime. The cycle count becomes:

$$N_{cycles,i,pe} = N_{ops,i}/IPC_{pe}, \tag{7}$$

where $N_{ops,i}$ is the operation count of the task i, and IPC_{pe} defines how many operations the PE can execute in one clock cycle. The runtime of task i on that PE is then:

$$t_{i,pe} = N_{cycles,i,pe}/f_{pe}. \tag{8}$$

When the task has finished its execution, it emits data token(s). A direct memory access (DMA) controller allows simultaneous computation and communication which means sending or receiving data while executing application tasks. The PE model checks the destination of the transfers in order to decide whether the data is transmitted to the NoC or is the communication internal to PE.

This defines which PE-specific communication costs are applied in addition to dynamic NoC delay [85]. The communication overhead has the form

$$t_{comm,pe} = a + bx, \tag{9}$$

where a is a constant cost (cycles), x is the data amount in bytes, and b is the cycles per byte. Values of a and b are defined separately for each PE type and its software platform. They are fixed (static) in the benchmark's XML document.

Time overhead is smallest when the source and destination tasks are in the same group (thread). A different cost is used when the destination is in another group but still on the same PE. The largest overhead occurs when the destination task is mapped on a different PE. In that case, the delay induced by the network will be added to $t_{comm,pe}$. However, note that the network delay is not static, unlike a and b, but it is determined at the simulation time.

For example in Fig. 25, communication between the tasks E and C is of type *intra-group*, between C and F is *intergroup*, and between C and B is *inter-PE*. The latter two can possibly cause a context switch at the receiver PE. Only the inter-PE communication is passed via network model. Intra-PE communication happens via local memory of a PE. The local communication style - pass-by-reference (as a pointer) or as pass-by-value (data copying) - affects the annotated communication overhead values.

Fig. 29 presents two examples of inter-task communication. In a), the communication occurs between two tasks in the same PE whereas b) presents inter-PE communication. In both cases, the execution times of the tasks are the same. The network delay is naturally not present in a), but otherwise the communication procedure is the same as in b). In addition, both send and receive costs are longer in b), because the data have to be prepared for the network transaction. These inter-PE transfers are naturally the most interesting in NoC evaluation. Note also that the benchmark measures latency of token transfer and not on per-packet basis. The latter is a NoC-specific

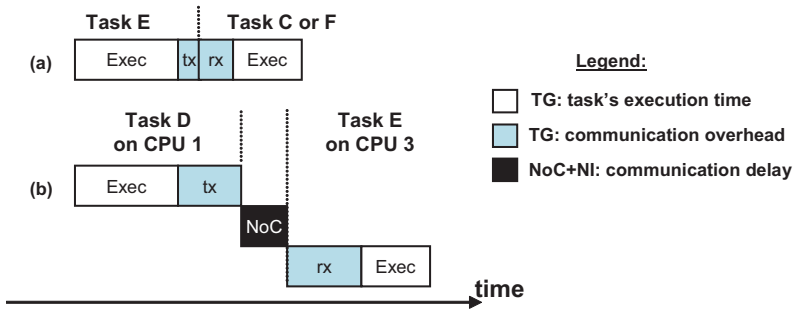


Fig. 29. *Different communication costs between two tasks when a) tasks are on the same PE or b) on different PEs. The task and PE symbols correspond to Fig. 25.*

low-level metric and inappropriate for comparison because the packet size may vary.

5.3.2 Network model

This section presents an XML description of a NoC, although the NoC is not usually defined in the benchmark. The purpose is to promote a consistent way of documenting the experiments and to enhance inter-operability of EDA tools. The network model may also be defined partially, hence setting some restrictions to the designer/evaluator.

The main emphasis is on documenting the topology and basic parameters. There is no point to create separate XML tags for all the possible configuration parameters of a NoC, especially those that are yet to be developed. However, the model still offers additional information to a black-box view of components. In other words, we only present a minimum set of properties that should be documented.

NoC is here defined by its:

1. Terminals which usually include network interfaces
2. Topology, i.e. routers and links
3. Parameters (optional part).

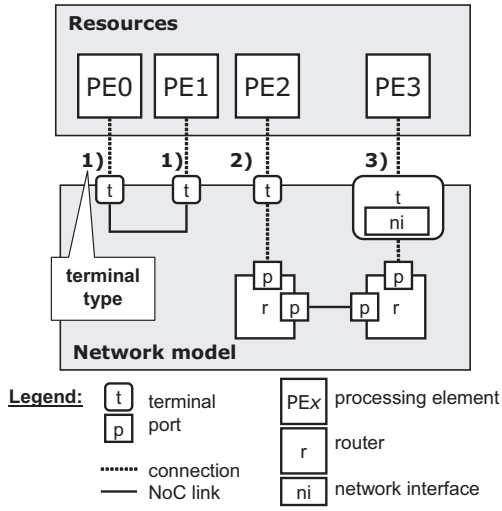


Fig. 30. *Three types of supported NoC terminals.*

Terminals and Network Interfaces

This section defines the locations of the resources with respect to NoC. PEs are bound to NoC terminals which are the "public interface" of a NoC. This provides a generic way to bind PEs without exposing the internals of the NoC. It depends on the specific NoC how the terminals are connected inside the network. On the contrary, connecting PEs directly to router ports would be network-specific and not reusable. The resource locations and the routing policy together define which communication routes are activated inside the NoC.

Each terminal may include a *network interface* but this is not always necessary. The NI, such as in [201], executes packetization (adding/removing header and tail flits), checks integrity, and possibly reordering of packets. In a multi-clock system, the synchronization between the PEs and NoC is left to the NoC designer. Hence, it occurs inside the NI or at the router's ports.

Currently, our Transaction Generator assumes that the NoC (its NI) has an OCP-IP compliant interface. In the future, we seek to adopt a more versatile concept for defining interfaces, for example similarly to [229].

Fig. 30 illustrates three different terminal types.

1. On the left, there is a direct point-to-point link between $PE0$ and $PE1$. The

corresponding terminals are simply the two ends of the link. Of course, the PEs must have directly compatible interfaces.

2. In the middle, $PE2$ communicates via a router. Each terminal is associated with a certain router port. PEs must use the same communication protocol (signals, their timing, and packet structure) as the router ports. Hence, the terminals may be conceptual, i.e. just aliases to certain router ports, which do not require any logic in HW implementation.

3. On the right, there is a network interface (NI) in addition to the router. This is the most generic and probably the most common option. The interface does conversion between communication protocols. For example, $PE3$ could have a standard interface but the router could use a different, proprietary protocol.

One might consider defining NI together with the resource. Here, they are within the NoC because that part is commonly left undefined in benchmark and completed by the NoC designer or evaluator. One could also define point-to-point links with network interfaces, although this option seems rare. It is left out from the figure but its behavior is easy to understand with the previous examples.

NoC topology

The NoC may be instantiated as a *white-box* model that defines all necessary topological details, or as a *black-box* that defines only the terminals and parameters in XML.

Fig. 30 illustrates also the internal structure of a NoC: there are links and routers. Links connect the ports of the routers which can be uni- or bidirectional. Bidirectional ports and links simplify XML description in many cases although in practice they are likely realized as a pair of unidirectional ports/links.

Each router includes the following basic parameters

1. varying number of ports and their directions

2. flit width and optionally also frequency

3. buffer capacity (number of flits per buffer) per port

4. number of virtual channels per port
5. minimum latency for forwarding the packet header

Each link defines

1. the end points, each defining a router and its port
2. number of data wires in one direction
3. pipeline depth (default is 0 which means no pipelining)
4. operating frequency (optional).

The connections between routers and PEs are included in terminal description. Occasionally, the link or router list may be empty.

NoC Parameters

Default values can be defined for any parameters, such as data width, to avoid repeating them for every single router or link instance. However, they can be overridden on a per-component basis.

The most common parameters, see the previous subsection, have specific tags but it is possible to include additional, arbitrary parameters as *name-value* -pairs. This supports using XML as input for proprietary configuration and synthesis tools. For example, specific arbitration, routing, multi-Vdd settings, queue management options may be given this way. The XML file may define arbitrary parameters for the NoC or its sub-lists.

Another use case is when the NoC is used as black-box and the actual structure is described separately in VHDL files (or similar) instead of XML. The compilation and synthesis parameters are extracted from the XML and passed to corresponding EDA tools. This allows instantiating third-party NoCs without knowledge of their internals while allowing parameterization.

The white-box NoC model can represent arbitrary network topologies but might be somewhat awkward for a large, say 100-node, NoC. The black-box approach is suitable in such case, especially for regular topologies. The designer/evaluator must

only define the name of the library component (e.g. 2-D mesh), network dimensions ($x = 10$, $y = 10$), terminal list ($0 - 99$), and possibly some other parameters. Effort is minimized as there is no need to describe individually all the routers and links. There is no need to describe individually all the 100 routers, all 360 links, connect links between west_out(x,y) to east_in(x+1,y) (except on the edges!), and so on. The detailed XML description, such as IP-XACT, can be generated automatically based on the above parameters, if desired.

5.4 Measurement constraints

This section includes restrictions related to the simulation and benchmarking. The simulation time may be defined as a constant value, for example 200 milliseconds. However, adequate absolute time is hard to come up with in general case and hence there are other, dynamic conditions for the simulation length:

- total of B bytes have been injected/ejected
- total count of all tasks' executions equals N
- task X has been executed N times
- communication edge X has been executed (used) N times
- path P has been traversed N times (paths are defined inside a task graph)
- same as previous but for several tasks/edges/paths are measured and average or maximum time will be used.

These conditions ensure that the evaluated system reaches steady state and that enough data has been transferred to obtain reliable performance results.

This section defines also the criteria that must be measured and reported and at least one cost function. Cost function combines various metrics into single value (the smaller the better) that can be used for direct comparison and ranking of approaches. Example metrics are execution count of a task, sum of all tasks' execution counts, deadline violations, token transfer latencies, simulation time, path time, silicon area, and power/energy consumption. The deadline related information is defined separately in the application description.

Basic arithmetic operations (sum, subtract, multiply, divide, power) and weights are applied to the metrics. The delivered benchmark set fixes a certain minimum set of criteria and cost functions. Researchers are, of course, encouraged to perform other complementary measurements and the same XML format can be used for reporting those studies also. For example, an OCP-IP workgroup [P9] is currently preparing separate rules and guidelines for conducting the NoC studies in order to avoid ambiguities and loopholes.

5.5 Implementation of the methodology

The proposed benchmarking methodology relies on traffic simulators, which are at this stage used in simulation environment. The generators model the traffic initiated by the resources. That means the application and the PEs executing it - three uppermost sections of Fig. 25.

There are two types of generators: one that models the dependencies between transactions and simpler one that is purely statistical without dependencies.

5.5.1 Utilized Transaction generator

Transaction Generator (TG) [P2] [85] was implemented in VHDL and in SystemC and both versions have been used in this work. The SystenC version has been published as open source. A limited but synthesizable version of TG in VHDL is currently being developed but discussion about that is omitted for brevity.

TG can be connected to different types of NoC models: VHDL at gate-level or RTL, and SystemC at RTL or transaction level. This section presents the SystemC version which is the newer and has more features. Fig.31 presents the SystemC classes for processing elements and tasks. The communication network model is not part of TG but is included here to present the required interface.

PE represents a hardware element such as a general-purpose processor or a hardware accelerator executing tasks that are mapped on it. Scheduler selects the executed task as depicted in Fig.28. PE class supports two types of inter-PE transfers. When direct memory access (DMA) is disabled (case *ii*)), I/O handler is added to the task list and executed sequentially with the other tasks.

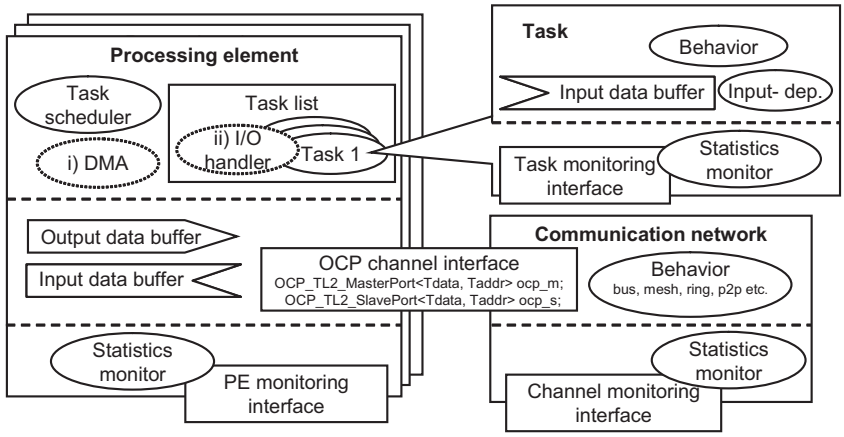


Fig. 31. *SystemC classes of Transaction Generator. Data transfers to/from processing element are implemented either i) with a DMA or ii) with own I/O handler thread.*

The implementation of the task class includes functions for resolving input dependencies and the actual behavior. The input dependence function determines the task triggering for execution according to the status of the received packet buffer, which contains pointers to packets in input data buffer of PE. In the behavior, the task functionality is executed and the output data stored in the output buffer of the related processing element. The processing element and task models get their operation, parameters, and relations as external models from the XSM.

In addition to generating the traffic, TG automatically checks all the transfers and collects statistics about system performance such as execution counts for tasks, execution and idle period lengths, and latency of data transfers. Statistics are collected for each PE and tasks. Furthermore, the transfers between the tasks also monitored for the analysis of the communication network performance.

Transaction Generator can be connected to Execution Monitor which is a versatile monitoring and visualization tool implemented in Java [84]. It supports any data source that provides execution information in a generic XML format, for example MPSoC on FPGA. Visualization helps extracting the most important issues in regarding system performance. For example, observing trends and anomalies in the behavior is much easier by looking at graphs drawn in real-time compared to textual log file interpretation afterwards. The data is visualized in three forms: graphs, tables, and flow chart.

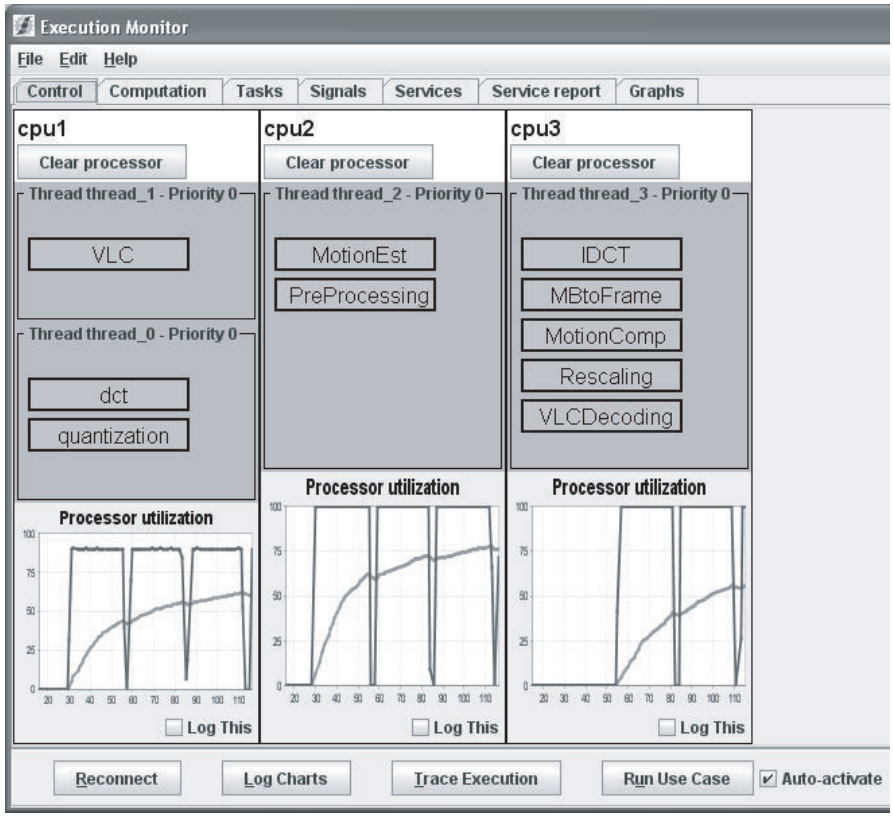


Fig. 32. *The main view of the Execution Monitor.*

Execution Monitor provides several views to the system, for example computation and communication statistics on a per-task and per-PE basis. Fig. 32 shows one view of the system with 3 CPUs and 10 software tasks. The view shows the mapping of the tasks as well CPU utilization. Table 14 summarizes what kind of data can be visualized with Execution Monitor. TG provides many values directly and some are derived from those. However, NoC-specific properties other than data volume and latency require additional special monitors in the simulation model or prototype.

5.5.2 Evaluation of accuracy and simulation speedup of TG

Table 15 from [85] summarizes reported system models, their target domains, obtained speedup, and error values from literature. They are introduced in detail in Section 5.5. Table data are collected from [27, 31, 77, 85, 101, 103, 116, 135, 159, 165,

Table 14. *Statistics visualized by Execution Monitor [84]. The information is collected either from TG simulation or FPGA prototype.*

	Category	Monitored values
Application	Application specific	E.g. frame rate, radio throughput
	Service	Service interaction graph, avg./tot. execution cycles, communication between services
	Task communication	Signals in/out, avg./tot. communication cycles, communication % of execution time, intra/inter-PE communication bytes and cycles, communication cycles/byte
	Task general	Execution count, avg./tot. execution cycles, execution % of thread/service total, signal queue, execution latency, response time
	Mapping	Task to thread/PE/service
Platform	SW platform	Thread priority, thread avg./tot. computation cycles, computation load, dynamically allocated memory
	PE	Utilization, allocated memory, power, inter-PE communication bytes, SW platform load, avg./tot. execution cycles
	Network specific	Utilization, efficiency, power, address cycles, data cycles

195, 197]. The two bottom rows correspond to TG used in this work. The closest match to the proposed methodology is presented by Pimentel *et al.* in [195, 197] and [117] (omitted from the table). Interestingly, the level of application granularity does not necessarily indicate much about the achieved speedup or accuracy. In addition, Fig. 33 shows the accuracy of TG when compared against cycle-accurate co-simulation [103] and against FPGA execution [85]. The latter study evaluates three different TG models, namely (in descending accuracy order) trace, modulo, and probabilistic. As a conclusion, TG provides an accurate and fast enough solution for benchmarking NoCs.

5.5.3 Utilized Stochastic generator

Although TG can be used also for stochastic traffic, that is not its initial purpose. The same behavior can be modeled with less effort by using a specific stochastic generator. The concept is very simple. All the traffic is stored into a text file, one file per terminal. Each row contains three values: target, data amount in bytes, and wait time in cycles. The files were opted to allow exact reproduction later despite randomness. Stochastic generator is currently implemented in VHDL.

A sender component reads the file and injects data to the network. At the beginning

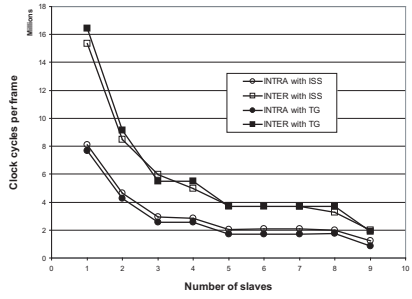
Table 15. *Summary of application models used in exploration [85]. This work utilizes the same TG as [85, 103].*

			Properties					Results		Evaluation			
#	Author	Ref.	Exploration domain	Automation	Application granularity	Functionally accurate	RTOS	Speed-up to low- level sim.	Runtime estimation error	Compared to	#Applications	#Mappings per application	#PEs in the system
1	Pimentel	[195]	Sys.	A	Medium	-	*n/a*	*n/a*	<2%	FPGA	1	2	(2)
2	Pimentel	[197]	Sys.	A	Variable	-	*n/a*	*n/a*	~12-34%,	FPGA	1	11	1-4
	-"-								<19-60%				
3	Mohanty	[165]	Sys.	A	Medium	-	*n/a*	*n/a*	<26%	Sim.	1	4	1
4	Bouchima	[31]	Sys.	M	Fine	x	x	~1000x	~5%	Sim.	1	2	7
5	Kakita	[101]	Sys.	M	Medium	-	x	n/a	<8%	FPGA	1	6	2-3
6	Lahiri	[135]	NoC	A	Coarse	-	*n/a*	~10x	<2%	Sim.	2	1	8
7	Heirman	[77]	NoC	M	Coarse	-	*n/a*	~9x	<(10+3)%	Sim.	1	2	16
8	Mahadevan	[159]	NoC	M	Fine	-	*n/a*	2-4x	<2%	Sim.	4	1-6	1-12
9	Bobrek	[27]	NoC	M	Variable	*n/a*	*n/a*	~100x	~18%	Sim.	2	6	2-32
10	Kim, S.	[116]	NoC	M	Medium	*n/a*	*n/a*	~450x	<10%	Sim.	1	1112	2-10
11	**Kangas**	**[103]**	**both**	**A**	**Medium**	**-**	**-**	**230x**	**~5-8%**	**Sim.**	**1**	**9**	**1-9**
12	**Holma**	**[85]**	**both**	**A**	**Medium**	**-**	**x**	**>230x**	**~4-13%,**	**FPGA**	**1**	**9**	**1-4**
	-"-								**<11-31%**				

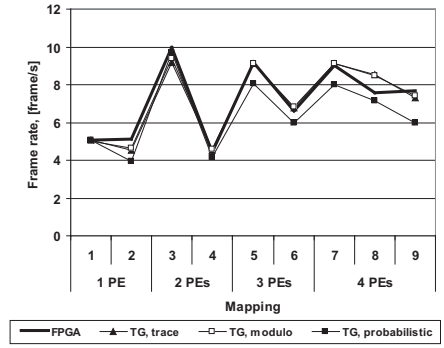
Sys. = System, NoC = Network-on-Chip, A = automated, M = manual
n/a = The information was not available, ~ = Average, < = Maximum, Sim. = Simulation

of the message, there is time stamp. Data values are simply consecutive integers with sender's ID concatenated. On the other side of the network, a reader component ejects the data, checks it, and calculates statistics. Sender and reader have a common notion of time, and hence latency per message is easy to calculate. Running numbers simplify error detection as the missing, duplicated, or corrupted date can be detected without difficulties.

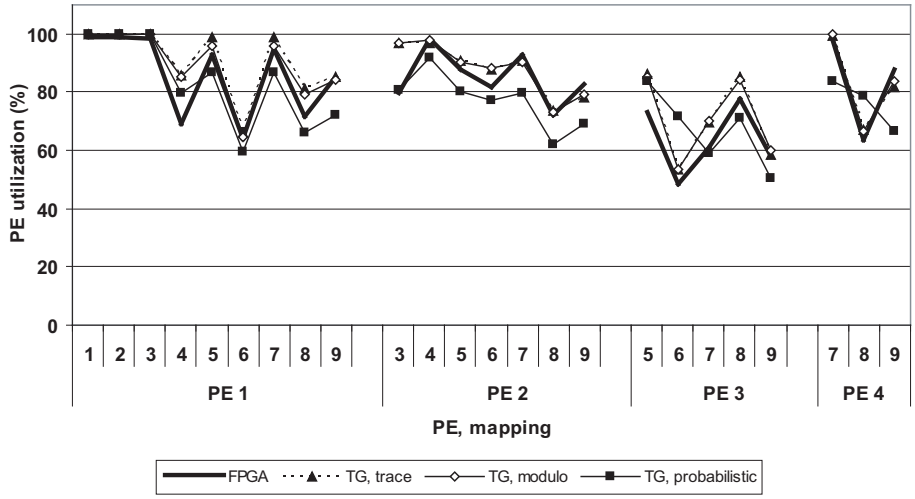
This thesis uses b parameter, "bias", to measure the level of burstiness similarly to [234, 242] whereas Hurst parameter is used in [228]. The b parameter lies in range $0.5 - 1.0$, larger values reflecting increased burstiness (only few large transfers). Many real life programs can be modeled with $b = 0.86$ [234]. Fig. 34 illustrates the procedure: X-axis shows the time of generated traffic trace, Y-axis shows the data rate, and each bar presents one time window. The procedure is as follows



(a) Cycle count in TG and in co-simulation.



(b) Frame rate in TG and in FPGA.



(c) Processing element utilization in TG and in FPGA.

Fig. 33. *The accuracy of Transaction Generator [85, 103].*

1. Designer sets the desired trace length, total data volume, and b. For example, 4096 clock cycles, 2000 Bytes, and $b = 0.6$.

2. The total time is divided in two equally long time windows which are given a fraction of the total data specified by b and $(1-b)$ respectively. The window getting larger fraction is selected randomly.

3. Each is window is divided again into two halves.

4. Splitting continues until the length of window equals user-defined limit, for

example 10 clock cycles.

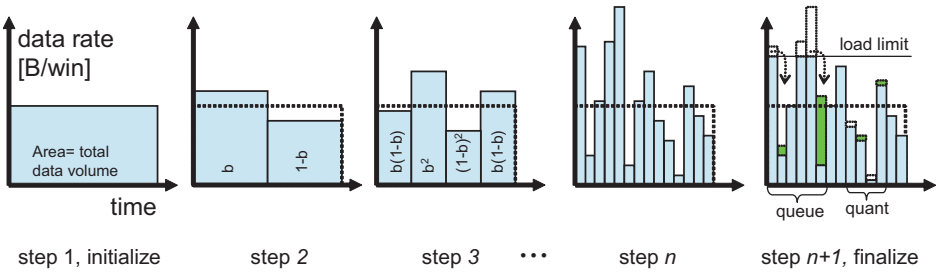
5. The final step quantizes the data rates to integer values smaller or equal to the maximum bandwidth of the terminal (marked as load limit in Fig. 34). Let us assume a 4 $B/cycle$ load limit due 32-bit interface to NoC. Then, the data rate of 5.4 $B/cycle$ is clipped to 4 $B/cycle$ and the overflowing 1.4 $B/cycle$ is summed to the next window. Consequently, very large bursts are spread to multiple consecutive time windows.

During simulation, data is emitted at the beginning of each window followed by wait period (if any).

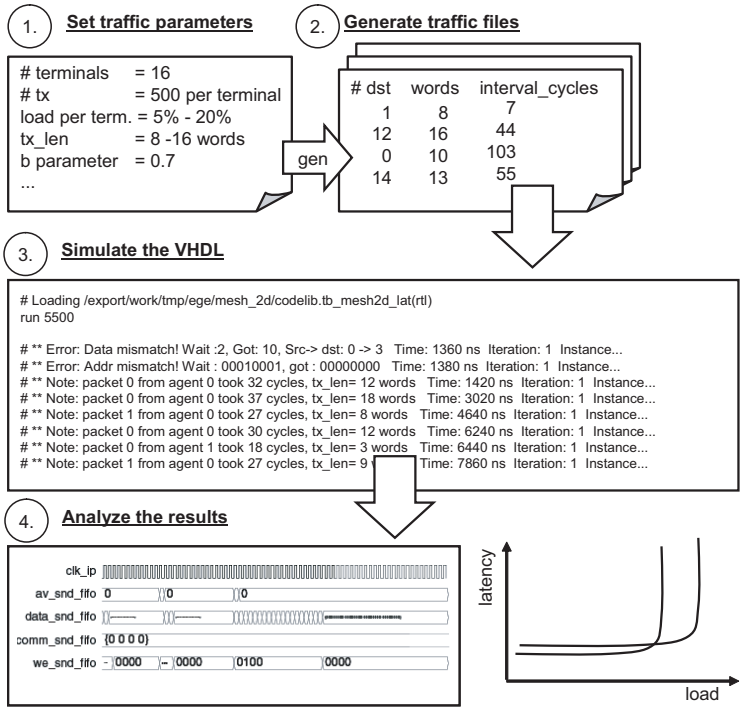
5.6 Discussion

Development of test applications was beyond the scope of this work. The final selection of applications and building models for them is left future work. However, brief projections about the practical test case types can be made. First of all, different application types, i.e. communication and computation intensive cases, must be present. Regular and simple functions like FFT, DCT, IIR, and matrix multiplication are sometimes used [121, 151, 159] but they can hardly be considered as typical applications for SoC, i.e. they present only kernels. In contrast, video coding [173] [105, 151, 254], 3GPP [247], WLAN [123] baseband processing, and data mining [184] are more demanding and, hence, interesting.

The presented benchmarking methodology relies on abstract workload models based on task graphs. The emphasis is currently on message-passing systems and simulation-based evaluation. The accuracy and simulation speed of the reference implementation were shown to be satisfactory. The future work will consider modeling of shared memories, cache memories, the workload imposed by SW platform, and better ways of creating the workload models.



(a) Generating bursty traffic with *b*-model.



(b) NoC evaluation using stochastic traffic generator.

Fig. 34. *Generating and using stochastic traffic.*

6. ON THE CREDIBILITY OF LOAD-LATENCY MEASUREMENTS

This Chapter was inspired by the observations in surveys [P10, P11]. Networks, especially NoCs, are typically compared and benchmarked via their *load vs. latency* behavior [51]. For example, this method was utilized in about 18% of the papers cited in Tables 2 - 5.

Unfortunately, the measurement setup is often unclear and sometimes even erroneous which leads to unfair comparisons or prevents them. For example, the utilized transfer sizes, metrics, and units are not always defined properly, the header and packet latencies may be confused, or latency values from the saturated state are sometimes given.

The following examples illustrate how the often undocumented features have a profound impact on the results [P13]. The differences between various measurements setups are analyzed next.

6.1 Introduction to load vs. latency measurements

Fig. 35. shows an example of the *offered traffic load vs. latency* curve for an imaginary network. It shows the transfer delay as a function of the transfer rate. The X-axis shows the average data rate that is offered, injected, to the network terminals. There is one traffic source per terminal and usually they all are active simultaneously. The Y-axis shows the transaction latency; in this case it is the average value. As the traffic load increases beyond the network-specific threshold, the injected data start stacking at the sender because the network cannot accept them fast enough. Consequently, their latency increases without bound as the situation continues, and the network is *saturated*.

The simplest traffic scenario sets equal probability for all targets (spatially uniform), injects data continuously, and keeps both the message size and injection rate fixed

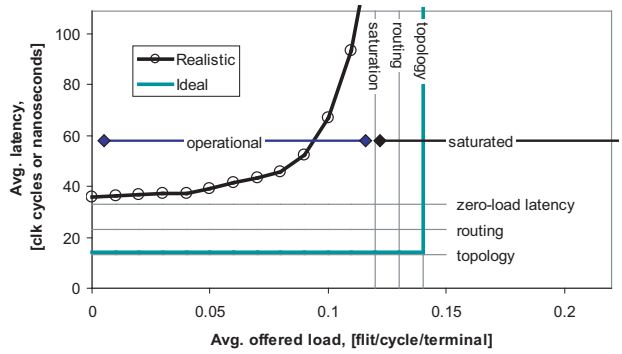


Fig. 35. *Example of load vs. latency curve. X-axis shows the transfer rate and Y-axis the delay. Six analytical bounds are shown in addition to measured values.*

(temporally uniform, periodic, i.e. constant bit rate). The traffic characteristics must be documented properly to enable repeatability and comparison. Section 4.3 analyzes the properties of traffic patterns found in real system-on-chips.

The exact curves are usually determined with simulation but certain bounds can be determined analytically [51]. The loosest bounds (bottom-right) are derived from the topology; namely the bisection bandwidth and the minimum number of hops (traversal of a link). Practical routing algorithms do not achieve perfect load balance and use more hops, hence defining tighter bounds (closer to the realized values). The actual load leading to *saturation* is obtained from simulation or measured from the implementation. *Zero-load latency* is measured so that only one sender is active which means that there is no contention. Due to congestion, the measured latency is usually higher than the zero-load latency, and increases slightly with the increased load before saturation.

6.2 Measurement setup

Unfortunately, both *offered load* and *latency*, are ambiguous terms, see [P10, P11] for examples. This does not ruin the validity of the results *within* one paper as long as all cases use the same definition. However, comparison *between* publications becomes impossible. This Section aims to alleviate these shortcomings.

The examples are given for a packet-switched 4*x*4 mesh with wormhole switching

[P8] and utilizing spatially uniform random traffic. In general, it is recommended using many types of spatial distributions but in this case, the simple uniform distribution suffices to bring out the issues author wishes to address. The average latency in clock cycles is here measured for transfers with 8 payload flits (flow control digits). When the transfer size varies, the average is still 8 payload flits. The traffic load is given for the payload only (the headers are excluded). Resources and NoC use the same clock. Each resource initiates 500 transfers; hence there are $16 \cdot 500 = 8\ 000$ transfers in total ($8\ 000 \cdot 8 = 64\ 000$ payload flits). The measurement settings are summarized in Table 16 in Section 6.7 with the corresponding deviations in results.

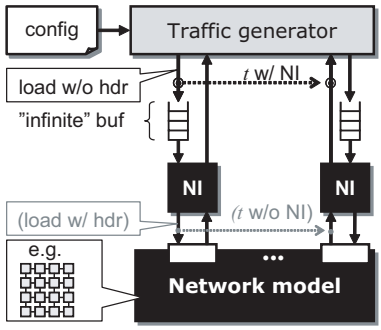
A canonical measurement setup is shown in Fig. 36(a). The traffic generator (TG) models the SoC resources, such as processors and memories, which initiate transfers. The configuration file defines the properties of the traffic:

- temporal - when and how much to send
- spatial - where to send.

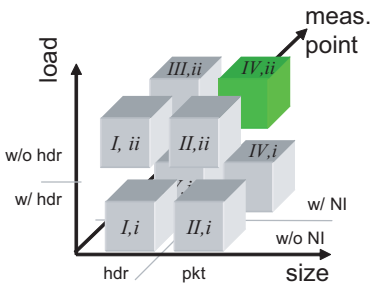
Network interfaces (NI) restructure the data stream from TG to packets accepted by the NoC, and vice versa. Large (infinite, in theory) buffers must be placed between the traffic source and network interfaces to avoid self-throttling during the measurement [51]. Without such buffers, the generator stops the data injection occasionally, and the real offered load does not match the expected load. When the network saturates, these source queues start growing infinitely. Longer simulations yield a larger latency for the saturated traffic but the maximum observed latency cannot exceed the length of the simulation. The interpretation of results must assume an infinite latency during saturation.

There are three basic choices in load-latency measurement each with two options. That means $2^3 = 8$ possible combinations to start with. Fig. 36(b) illustrates the resulting 3-dimensional measurement space. Each choice defines one axis in the design space:

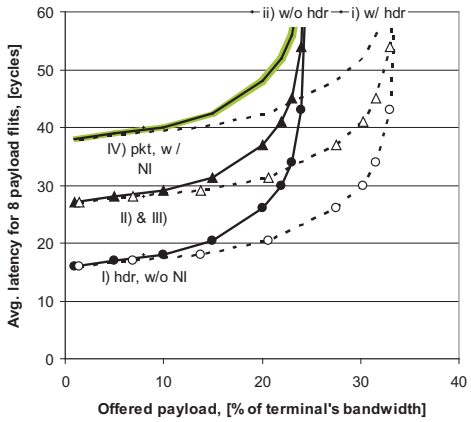
1. Size: Is latency t measured for the header or for the whole transfer?
2. Measurement point: Does latency t include the network interface (NI) and waiting at the source buffer?



(a) Measurement setup. The non-recommended ways are in parentheses. *NI* denotes the network interface, *t* is the measured transfer latency, and *hdr* refers to packet header.



(b) 3-dimensional measurement space with 8 setups. The recommended case *IV,ii* is highlighted.



(c) Measured latencies in various setups. Note the deceitful difference between cases *IV,ii* and *I,i* although the NoC and traffic are the same.

Fig. 36. *Basics of load-latency measurement: different setups and the resulting deviations in measured performance.*

3. Load: Does the load r (*i*) include or (*ii*) exclude the header?

Our recommendations are presented next.

Header latency gives the lower bound for communication delay in a system but headers do not transmit any useful data in general case. Hence, latency of full packets should be measured.

The latency measurement starts when the first bit of the transfer leaves the traffic generator and ends when the last bit exits the NI at the receiver. The NI must ensure that received data is in-order. Its latency and (buffer) area overheads may be substantial when reordering is required due to out-of-order delivery in the network.

The definition of offered load must be independent of the measured NoC. In other words, it is measured on a higher protocol layer than network. The header overhead depends on the evaluated NoC and NI, and therefore the header must be excluded from the load. Otherwise, a NoC with large headers will be favored in an unfair manner, as it seems to tolerate a larger load than it actually does. This applies even if the source buffer and NI are implemented as part of the traffic generator.

As a summary, the latency must be measured for the whole packet including source queue, NI, and NoC latencies, and the load must include the payload only. That way the values correspond closely to the latency experienced by the actual processing elements (PEs) during real operation. The recommended setup is denoted as case *IV,ii* in Fig. 36(b) and 36(c). The same curve is repeated in the following graphs for the sake of comparison.

Fig. 36(c) shows the impact of the various measurement options. Solid lines denote that headers are excluded from the load whereas they are included in the dashed lines. These load definitions do not affect the latency but they move the curves in the X-direction. On the other hand, the measured transfer size and the impact of NI move the curves in the Y-axis without affecting the saturation point. Differences are large: the latency appears to be reduced to $0.4x$ while obtaining $1.4x$ throughput when basic choices are done "creatively" (16 vs. 38 cycles and 33 vs. 24%, respectively). Case *II* is here coincidentally identical to case *III*.

6.3 Units of measurement

Another consideration regards the utilized measuring units and statistical values.

1) What values are reported? The most common and suitable choices for latency are average t_{avg} (10) and percentile $t(p)$ (11)

$$t_{avg} = \frac{1}{N}\sum_{i=0}^{N-1} t_i \tag{10}$$

$$fraction\ p\ of\ transfers:\ t_i \leq t(p) \tag{11}$$

$$t_i = t_{lastbit\_to\_rx\_TG} - t_{firstbit\_from\_tx\_TG} \tag{12}$$

where t_i denotes the latency of a single transfer and N is their total number. Latency percentiles or box-plots are especially important when dealing with real-time constraints as the average performance is inadequate metric in those cases. The maximum jitter t_Δ (variation between transfer latencies) is calculated as

$$t_\Delta = \frac{t_i}{min(all\ t_i)} - 1. \tag{13}$$

Naturally, the latency equation must be reported explicitly to avoid confusion. For example percentiles $t(90\%)$, $t(99\%)$, and $t(100\%)$ are reported for latency and jitter [157]. The last one equals the maximum value of the set. Metrics t_i and $t(p)$ have the same unit (see next subsection) whereas t_Δ is unitless.

2) What is the unit of latency? Latency measures how long it takes to perform a certain operation; a transfer in this case. In principle, its unit is $time/operation$ - the inverse of data rate (throughput). Note that inverse value of the data rate does not equal latency in pipelined systems although units are reciprocal.

Both seconds and cycles are plausible time units. The latter is SI unit but affected by the (assumed) NoC frequency. Frequency is an implementation-dependent variable but most design-choices (routing algorithm, buffering, virtual channels etc.) can be compared with their cycle counts. Care must be taken to define "clock cycle" appropriately in a multi-clock system, e.g. using specifiers $\#cycles_{PE}$ or $\#cycles_{NoC}$.

The monitored *operation* is case-dependent and needs rigor definition. For example, latency per packet needs information about inclusion/exclusion of header, payload length and is it varying dynamically between traffic sources.

3) What is the unit of load? It is preferred to measure traffic load per terminal so that its maximum value does not depend on the network size or topology. If latency is measured at clock cycles, the recommended load unit is

$$[offered\ load] = flit/cycle/source \in [0.0, 1.0]. \tag{14}$$

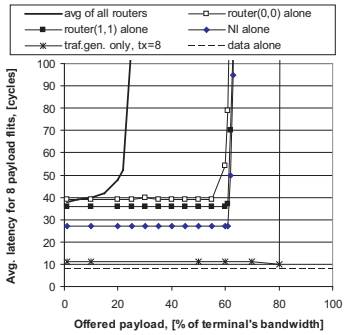
The number of traffic sources usually equals the number of terminals but not always, for example when measuring zero-load latency. In most cases, $flit/cycle/terminal$ may be opted to emphasize the fact that data is injected to all inputs. Either way, the chosen load metric has intuitive bounds that do not depend on the network size and only the payload flits are accounted. The data width and frequency of NoC terminals are implementation-dependent choices. Replacing them to (14) yields data rate in $bits/s/source$ or $Bytes/s/source$.

When the load varies between terminals or dynamically, characteristics are calculated with equations (10)-(13) by replacing latency t with data rate r.

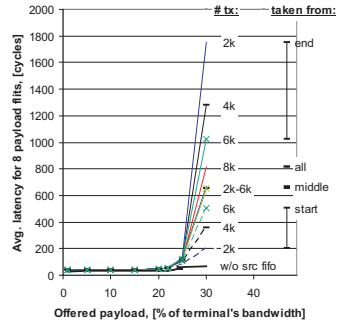
Expressing the load as a fraction (%) of terminal's bandwidth also seems to be a viable option. Unfortunately, the load is sometimes given as a fraction of network's *ideal capacity* on uniform traffic [51]. Capacity Θ means the input bandwidth that saturates the network, and the aim is to evaluate how close to ideal performance the network implementation gets. Percentage is always relative to something so it is obviously important to document clearly how it is measured, or avoid its usage. For example in Fig. 35, saturation occurs at 12% or 85% depending whether it is relative to terminal bandwidth or to network's ideal capacity. Note that it is no use to compare latency curves from different NoC topologies in a single graph if load is expressed as fraction of capacity. Furthermore, capacity naturally depends on the traffic pattern, and it may be hard to derive it for other cases than uniform random traffic. Hence, capacity Θ as an offered load measure should be avoided in most cases.

6.4 Latency breakdown

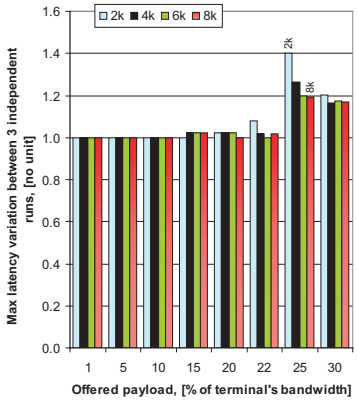
A part of the latency is due to utilized network interfaces but also to inefficiencies of the traffic generator. Moreover, the evaluator must ensure the saturation is really caused by the NoC itself and not by these secondary components. Fig. 37(a) shows the results of such sanity check where the components are measured in separation.



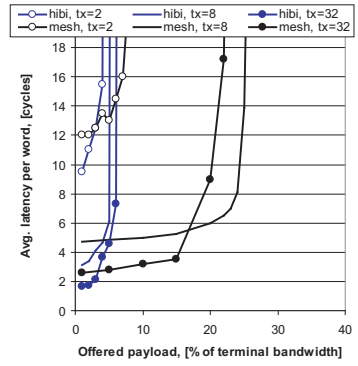
(a) Measurements done separately for each component.



(b) Varying the number of monitored transfers from 2000 to 8000. The impact of warmup period at the beginning is also visible.



(c) Maximum variation between 3 independent runs as load and the number of monitored transfers vary.



(d) Impact of transfer length when it varies from 2 to 32 words (flits). Note the exceptional scale on Y-axis.

Fig. 37. *Comparing load-latency per component and between different measurement runs.*

Let us define the total latency t and the saturating data rate r as

$$t = \sum t_{TG} + t_{NI} + t_{NoC} \tag{15}$$

$$r = min(r_{TG} + r_{NI} + r_{NoC}) \tag{16}$$

The lower bound for latency on the bottom illustrates the time needed for the mere data copying: 8 cycles in this case. Above that is the 3-cycle additional latency due to traffic generator. This is rather small overhead and could perhaps be eliminated in

future versions of TG. However, the generator used in this study spends 2 extra cycles between for each transfer, namely one cycle for address and one related to internal state machine. This is compensated with small traffic loads but affects the maximum load. Hence, it can inject at most

$$\frac{8\ flits}{(8\ flits \cdot 1\ cycle/flit) + 2\ cycles} = 0.8\ flit/cycle.$$

Network saturation occurring beyond that would be masked away but these empty cycles have no effect in lightly loaded case. Large transfers move the NI's saturation upwards, for example, 32-word transfers saturate at 94% of terminal bandwidth (not shown).

The utilized NI has latency overhead that grows with packet length. Time needed in NI for 8 payload words is 8 cycles for data, 3 for the header, and 2 empty cycles related to the current implementation. In other words, maximum offered load becomes $\frac{8\ flits}{13\ cycles} = 0.62\ flit/cycle$ per terminal due to this particular NI. It is by chance identical to the saturation point of a single router because the control logic inside a router also wastes two cycles between the packets.

The location of the router affects its average latency but not the saturation load. For example, the corner router (0,0) has larger latency than middle router (1,1). Note also that data copying, traffic generator and NI together cause 27-cycle latency whereas the (header) latency through the NoC attributes only additional 9 cycles. Let us assume that an optimized NoC might reduce the latency by, say, one cycle. This would mean a notable 11% reduction or mere 3% depending on whether one considers the network only or the full end-to-end latency. The differences between NoC latencies diminish even further, when the runtime overhead of SW platform running on a processor are also accounted.

The above simulation validates that the following holds

$$r = r_{NoC} < r_{bisection} < r_{router} \leq r_{NI} < r_{TG} \tag{17}$$

In this case, the example mesh saturates at 24% load. The bisection B_{mesh} has $2k$ unidirectional links, where k denotes the number of nodes in one dimension [51]. Hence, the theoretical load bound for uniform load in 16-node mesh is $B_{mesh} \cdot \frac{2}{N} = \frac{4k}{N} = \frac{4}{\sqrt{N}} = 1.0\ flit/cycle/terminal$. This reveals room for improvement regarding routing policy, header overhead, and the control logic of the routers. Furthermore,

we observe that the saturation load decreases with larger network sizes even if mesh is often dubbed “scalable”.

6.5 Impact of the transfer count and length

The number of monitored data transfers, their length, and pseudo-random behavior affect the results. These phenomena are studied next. Fig. 37(b) shows the results when the number of transfers varies, averaged over 3 independent runs. In all cases, 8000 transfers were generated and $2000-8000$ were included for calculating the average. The sampled transfers were taken from beginning, middle, or from the end of the simulation. Furthermore, an example of erroneous measurement setup is shown with a case where the infinite FIFO was omitted from the source. A similar pitfall is to use a too small source FIFO.

The differences become visible only near the saturation. During saturation, the last packets will experience longer latency than the first. The knee-point of curve gets sharper with larger transfer counts and when the warmup period (i.e. the first packets) is discarded. However, all transfer counts had the same non-saturated latency and were able to recognize the saturation point. Note that the saturation point can be identified more accurately from the load-throughput curve.

The measurements were repeated 3 times independently in order to study the impact of randomness. In this case, the generation of traffic files includes randomness whereas the simulation produces always the same results with given traffic file. The average latencies, eq. (10), of each run were compared, and the ratio between largest and smallest values is shown in Fig. 37(c). The differences are negligible at small loads and visible near saturation. Differences between runs decrease when more transfers are monitored. However, the transfer counts are here high enough that one simulation run suffices. In general, designers are encouraged to use largest transfer count that is practical regarding the simulation time. In general, the number of needed runs and transfers may be sought empirically case-by-case, for example as

$$for\ all\ loads:\ for\ all\ runs: \frac{max(t_{avg})}{min(t_{avg})} - 1 \leq \varepsilon \tag{18}$$

and let’s say $\varepsilon = 2\%$ for non-saturated loads.

Fig. 37(d) illustrates the variation due to different transfer lengths. Three cases are shown: transfer length being $2, 8$ or 32 payload words (=flits in this case). Similar measurements were done with single HIBI (Heterogeneous IP Block Interconnection) segment. Note that the latency is here given *per word* whereas other graphs measure the latency *per transfer*. The latency per word decreases with longer transfers because the empty cycles in traffic generator and NI occur only once per transfer. This is clearly shown with HIBI that has 1-word header (address) and varying burst length needs only single header per arbitrarily long transfer.

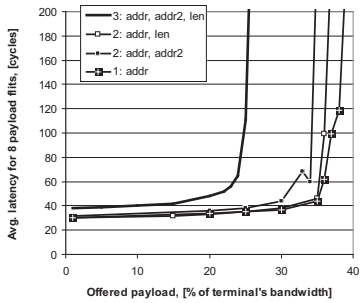
In addition, longer transfers often move the saturation point to higher loads. Both networks have inferior results with 2-word transfers, especially the mesh that used fixed length packets of 8 payload flits and 3 header flits. Comparing results from simulations that utilize different transfer length is not meaningful. An obvious recommendation is to study multiple transfer lengths for each compared NoC and document the settings.

6.6 Network-specific settings

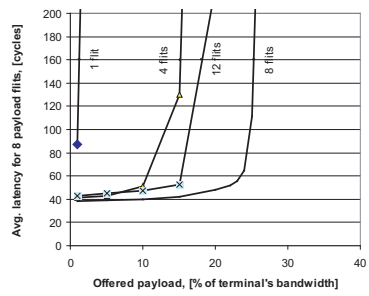
Final remarks concern the settings related to the measured networks. The *overall performance* or *merit* is a combination of several factors, not just latency. For example, a network with the highest performance may be prohibitively costly to implement and hence out of the question. Therefore, an appropriate multiobjective trade-off is sought. Automated design space exploration tools often seek to optimize a *cost function* that considers properties such as silicon area, power consumption, runtime, and latency. Network-specific settings - such as data width, frequency, buffer depth - affect all these metrics which motivates their careful selection.

Packet structure is clearly network-specific parameter whereas transfers are defined in the benchmark set. Preferably, several packet sizes must be evaluated. There are three basic choices for selecting network settings, for example packet size, for the compared NoCs:

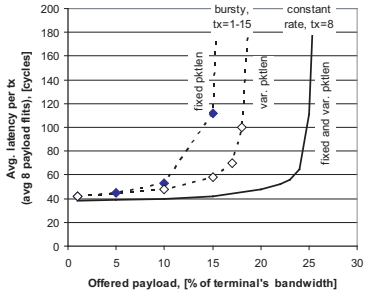
1. Use the default packet size for each NoC, so called out-of-the-box execution. The resulting differences in performance, area, and power are accounted in cost function. A variant of this scheme uses the same packet size for each NoC.



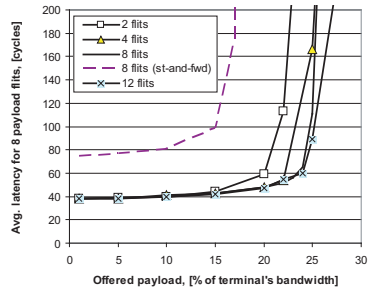
(a) Header length of $1-3$ flits. Payload is 8 flits.



(b) Fixed payload length of $1-12$ flits.



(c) Varying versus fixed payload length.



(d) Varying the buffer depth at the routers from 2 to 12 flits.

Fig. 38. *Impact of network specific settings: lengths of packet header and payload, and the depth of buffers at the routers. Header is* 3 *flits unless otherwise noted.*

2. Select packet size that minimizes cost function, so called full-fury optimization.

3. Select sizes that produce (nearly) equal area, and/or power, i.e. compare NoCs that conform to given constraint. The obtained performance values are now directly comparable.

Similar selection procedure can be easily generalized to other parameters, such as buffering, pipelining, frequency, and data width.

Examples are shown in Fig. 38. Utilized mesh supports up to three header flits: mandatory NoC address, optional packet length, and optional second level address. The examples so far utilized all 3 fields. Performance with shorter headers is shown in Fig. 38(a) for four header types. The differences in latency are quite small but

shorter headers provide a notable $1.45x$ increase in tolerated load. This observation clearly supports our recommendation to exclude headers from the offered load. Of course, shortest possible headers are desired and minimum length is dictated by the data width and the complexity of the network protocol.

Mesh in Fig. 38(b) uses fixed-size packets with 3-word headers and payload of $1, 4, 8$, or 12 words. The header overhead is intolerable for 1-word payload and larger payloads naturally increase performance. For example, increasing payload length from 4 flits to 8 increases the maximum data rate by a factor 1.6. However, dummy data that is used to fill packets deteriorates the performance with $12-$word payload. Dynamically varying packet overcomes this drawback.

Fig. 38(c) varies both transfer and packet size. The traffic is now *bursty* because the bit rate varies which resembles real-life applications better than constant load. The average transfer size is 8 payload words in all cases and varies uniformly within shown range. This situation is bad for fixed-size packets that need dummy data for nearly all packets. A mesh that is able to vary packet size at runtime obtains categorically better results. However, the constant bit rate traffic saturates at 24% and bursty at 18%, i.e. $1.4x$ difference.

Fig. 38(d) illustrates the variation due to buffer depth in the routers as it varies from 2 to 12 flits. Furthermore, a mesh with store-and-forward switching scheme was measured and found less capable than the wormhole switched. Larger buffers clearly increase the performance but also the area cost. In this case, increasing buffering by factor of $6x$, results in $5.5x$ area cost and $1.14x$ saturation point. Hence, performance increase is rather modest compared to cost.

6.7 Discussion

Very large differences are observed due to innocent-looking basic choices in measurement setup and network settings.

The guidelines are summarized as:

- An “infinite” source queue is placed between traffic generator and NI.
- The measurement points are between traffic generator and source queue.

Table 16. *Measurement settings and the maximum observed differences on peformance.*

	Property	Value and Unit (/note)	Max. impact in case study	Recommendation
Network	size	16 terminals	-	evaluate many
	topology	2-D mesh, bus (2 choices)	3.5x throughput	evaluate many
	header length	1 - 3 flits	1.6x throughput	shortest possible
	a) payload length	1 - 12 flits	25x throughput	evaluate many
	b) payload length	fixed or varying (2 choices)	1.2x throughput	(dynamically varying)
	buffer depth	2 - 12 flits	1.14 throughput	smallest adequate
	switching	wormhole, st-and-fwd (2 choices)	1.6x throughput, 2x lat.	(wormhole)
Traffic	spatial	uniform dst distrib. (but not to itself)	-	evaluate many
	offered load	0.0 - 1.0 flits/cycle/terminal	-	evaluate many
	a) tx length	2 - 32 payload flits	1.6x throughput	evaluate many
	b) tx length	fixed or varying (2 choices)	1.4x throughput	varying
	# tx	2000 - 8000 tx /terminal	9x latency	largest practical
Measurement	latency for	header or whole tx (2 choices)	1.6x throughput, 1.3x lat.	whole tx
	load definition	incl./excl. header - " -	1.3x latency	excl. header
	meas. point	incl./excl. NI - " -	1.3x latency	incl. NI
	infinite src queue	included/excluded - " -	exclusion is illegal	included
	latency	per transfer, per flit - " -	-	per transfer
	# src terminals	corner, middle, or all (3 choices)	3x throughput, 1.1x lat.	all
	# independent runs	3 runs	1.4x latency	largest practical
	warmup length	0 - 6000 transfers	9x latency	discard warmup

- The latency is measured for the whole packet including source queue, NI, and NoC latencies.
- Average, percentiles, and jitter are the most common values for characterizing latency. The choice must be stated explicitly to avoid confusion.
- The recommended unit for offered load is $flit/cycle/terminal$ and it includes the payload only.
- Large number of samples (monitored transfers) increases the validity.
- The min/avg/max values of packet size must be given when the size varies at runtime or across the nodes.
- Study multiple transfer lengths for each compared NoC and document the settings.
- Ensure that the bandwidth is not limited by the traffic generator or NI to effectively measure the impact of NoC parameters.
- The parameters of the studied NoC must be selected with care and considering the overall cost.

Table 16 summarizes the measurements, maximum impact of parameters, and recommendations.

7. HETEROGENEOUS IP BLOCK INTERCONNECTION (HIBI) V.2

This chapter presents the second version of Heterogeneous IP Block Interconnection (HIBI). Topology, arbitration and data transfers are presented first. After that, data buffering and the structure of wrapper component are discussed. Finally, the developed runtime configuration is presented followed by comparison to the previous version of HIBI.

The development of HIBI [133, 136, 138] started in 1997 in Tampere University of Technology with a goal of defining a reusable communication network for system-on-chips. HIBI is intended for integrating coarse-grain components such as intellectual property blocks that have size of thousands of gates, see Tables 7 and 8 for examples.

A survey [P1] and its extended versions [136, 177] indicated how fixed constraints dominate the contemporary buses. For example, the number of agents (i.e. devices connected to the network) and the available bus widths are strictly limited, leading to constrained modularity and scalability. Furthermore, the hierarchy was often restricted to two bus segments connected with a bridge.

Hence, the design objectives behind HIBI v.2 were to design a topology-independent, scalable, and still high-performance network based on experiences from HIBI v.1 Special attention was paid on configurability, efficiency of the protocol, and quality of service. New features include better support for hierarchical topologies with multiple clock domains, more efficient and versatile runtime configuration, and better modularity enabling silicon area minimization. Furthermore, more emphasis was put into FPGA prototyping.

A parameterizable HW component, called HIBI wrapper, is used to construct modular, hierarchical bus structures with distributed arbitration and multiple clock domains as shown in Fig 39 (explained later in detail). This simplifies design and allows reuse since the same wrapper can always be utilized. Configuration takes place both at synthesis time (e.g. data width and buffer sizes) and on runtime (arbitration parameters).

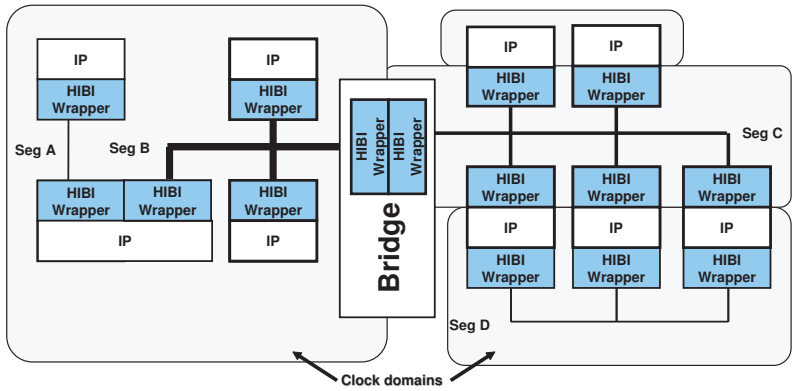


Fig. 39. *Example of a hierarchical HIBI network with multiple clock domains and bus segments*

A method for automating the parameter selection at design time is presented in [206].

7.1 HIBI topology

Bus performance can be scaled up by using bridges as shown in Section II. The key is to cluster frequently communicating agents in the same segment to provide small latency and adequate throughput. Clustering offers more freedom in the optimization since segments can operate with various data widths and clock frequencies. The implementation area and power consumption are lowered by making the segments as slow and narrow as possible but still meeting the application requirements. Segments having only simple peripheral devices can have a slow and narrow bus while the main processing parts have higher capacity buses.

Fig 39 depicts an irregular HIBI network. The example has a point-to-point link (*SegA*), hierarchical bus (*SegB* and *SegC*), and multibus topology (*SegC* and *SegD*). Furthermore, *SegB* is wider than other segments and thus offers greater bandwidth. In the multibus configuration, each IP must decide which bus to use while sending. Note that *SegA* could be implemented without wrappers since there is no need for arbitration.

The example shows four clock domains. Agents in *SegA* and *SegB* are inside one domain and HIBI wrappers on *SegC* are in one domain. However, two IPs in the top right corner use different clock than the wrappers of *SegC*. The IPs in the bottom right

corner and all wrappers in *SegD* are in one domain. The number of clock domains is not otherwise restricted but all wrappers in one bus segment must use the same clock. Handshaking between the clock domains is done in the IP-wrapper interface or inside the bridge [126, 127]. This allows the construction of GALS systems. The example shows only one bridge but HIBI does not restrict either the number of bridges or hierarchy levels in contrast to many bus architectures.

All wrappers in the system are instantiated from the same parameterizable HDL (HW description language) entity and bridges are constructed by connecting two wrappers together. If the connected segments use different data widths, the bridges are responsible for the data width adaptation.

Transfers inside a bus segment are circuit-switched and use a common clock due to (current) implementation of the distributed arbitration. However, HIBI bridges utilize switching principle that resembles packet-switching so that bus segments are not circuit-switched together. Instead, the data is stored inside the bridge until it gets an access to the other segment. The data is forwarded to next segment as soon as possible like in wormhole routing. However, no guarantees are given for the minimum length of continuous transfer. If the bridge cannot buffer all the data, the transfer is interrupted and the source segment is free for other transfers. The interrupted wrapper will continue the transfer on its next turn. It is also possible that a bridge buffers parts from multiple transfers.

7.2 Arbitration

A distinct feature in HIBI is that arbitration is distributed to wrappers, meaning that they can decide the correct time to access the bus by themselves. Therefore, no central arbiter is required. For example, Intel Itanium 2 system bus uses symmetric distributed arbitration (with fixed round-robin arbitration) [90]. In addition, at least Silicon Backplane [226] and Hewlett-Packard's Runaway bus use distributed arbitration [33].

A two-level arbitration scheme, a combination of time division multiple access (TDMA) and competition, is used in HIBI. In TDMA, time is divided into repeating time frames. Inside frames, agents are provided time slots when they are guaranteed an access to the communication channel. This way the throughput of each wrapper can

be guaranteed. For example, Aethereal network [207] uses TDMA also. The worst-case response time for a bus access through TDMA is the interval of the adjacent time slots. TDMA in HIBI supports two flavors for handling the slots when there is no data send: keeping them or releasing the bus for competition.

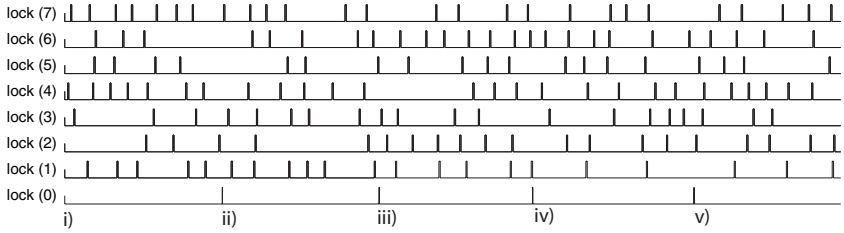
Competition is based either on round-robin or non-pre-emptive priority arbitration. The second level mechanism is used to arbitrate the unassigned or unused time slots. If the agent does not have anything to send in the beginning of its time slot, the time slot can be given away to allow maximal bus utilization. Priority arbitration as a second level method attempts to guarantee a small latency for high priority agents whereas round-robin provides a fair arbitration scheme. When the bus is freed and priority scheme is utilized, the agent with the highest priority can reserve the bus on the first cycle. If the bus has been idle for two cycles, the agent with the second highest priority may reserve it and so on. The maximum transfer length is restricted with runtime configurable parameter *max_send*. For round-robin, the maximum wait time for accessing the bus is obtained by summing all *max_send* values. For priority-based arbitration, the maximum wait time can be defined only for the two highest priorities. This means that the low-priority agents may suffer starvation and system may end up in deadlock. Therefore, using only priority arbitration is not recommended. Moreover, a new scheme called Dynamically Adaptive Arbitration (DAA) was presented in [130].

TDMA arbitration may prove beneficial also for energy saving because transfer times are known in advance and, therefore, timing of the shutdown and predictive power-up operations are simple [137]. The application can actively set the power save modes through a configuration memory.

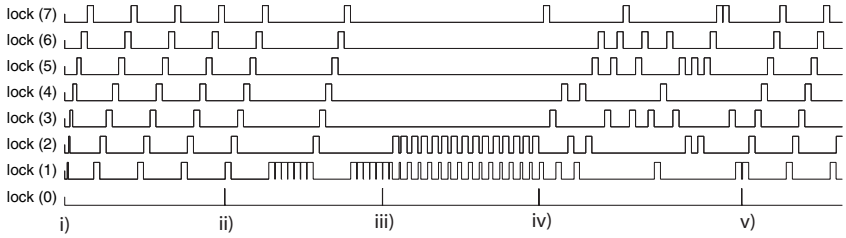
Fig. 40 shows the differences in various arbitration policies and two traffic loads (low and high contention). HIBI is configured as single bus with 8 agents. Agent 0 performs dynamic reconfiguration (time instants $i - v$) and other agents generate uniformly distributed random traffic. The reconfiguration changes the arbitration policy at runtime. The exact configuration procedure is explained in more detail later The utilized arbitration policies are

i) round-robin

ii) combination of priority and round-robin



(a) Low contention (send probability 4% per agent).



(b) High contention (send probability 30% per agent).

Fig. 40. *Various arbitration schemes for 8-agent single bus and uniform random traffic. The differences become evident on highly utilized bus.*

iii) priority

iv) random

v) round-robin (again).

Round-robin offers fair arbitration (each agent has its share) whereas priority favors the highest priority agents and leads to starvation of others. Their combination switches between them at user-defined intervals. Arbitration policy does not play a major role when bus is lightly loaded, as illustrated in Fig. 40(a). The differences are clear with higher load, Fig. 40(b).

Various arbitration methods of HIBI were compared in [130]. The test case was MPEG-4 encoding on MPSoC. HIBI has 6 arbitrated components: 4 CPUs, SDRAM, and performance monitor; all operating at $50MHz$ frequency. The maximum transfer length was varied from 5 words (denoted as $tx = 5$) to non-limited. Transfer length has major impact but all lengths of 50 words or over (tx>49) resulted in equal performance. The bus frequency was set to $1, 2, 5$, or 50 MHz in order to achieve varying bus utilization ($75\%, 56\%, 26\%$, and 3%, respectively) with single application. The best and worse algorithms vary case by case but DAA performed well in general.

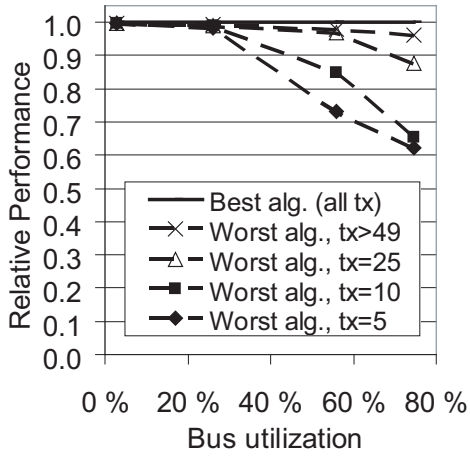


Fig. 41. *Relative performance of arbitration algorithms in MPEG-4 encoding [130]*

Fig. 41 plots the relative encoding performance between the worst and best algorithms. The curves denote different transfer lengths, and 1.0 is the best algorithm for each case. Tx lengths over 49 are joined for clarity because they yield practically the same results. With short transfers, the worst algorithm at 1 MHz HIBI (75% utilization) offers only $0.62x$ the performance of the best, at 2 MHz $0.73x$, at 5 MHz $0.98x$, and at 50 MHz there are no differences.

7.3 Data transfer operations

Since there is exactly one path between each source and destination, all data is guaranteed to arrive in-order and hence no reordering buffers are needed at the receiver. In HIBI v.2, all operations can be targeted either to a single agent or to a group of agents with multicast. Data can be sent with different relative priorities. High priority data, such as control messages, bypass the normal data transfers inside the wrappers and bridges resulting in smaller latency. This does not change the timing of bus reservations, but it selects what is transferred first.

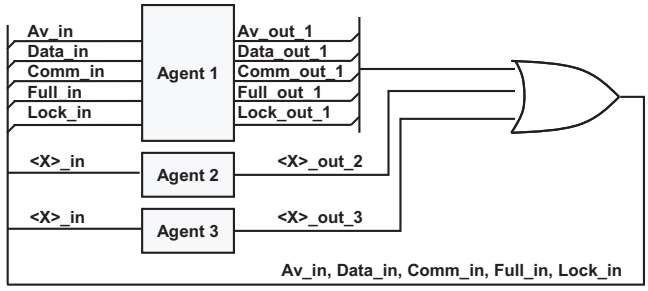
In traditional circuit-switched protocols, the bus is reserved during the whole read operation which is problematic when accessing slow agents, such as off-chip memories. To overcome this, HIBI uses a *split transaction* for read operation. The reading

agent first sends a read *request*, after which it frees the bus for other transfers. The requested data (*response*) is returned with a normal write operation. The reader does not get data any faster but the advantage is that the shared medium is available for other agents in the middle of the transmission process and consequently the achieved total throughput increases. In packet-switched networks the split-transactions are commonly used and also in modern bus protocols, such as AMBA [1].

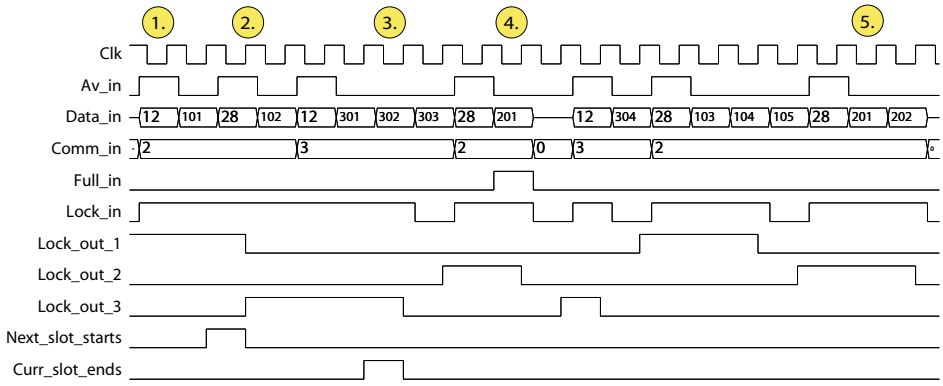
In fact, arbitration for response may increase the read latency but that is usually compensated by the increased throughput, especially when memory latency is large. For example, Chow and Sohi evaluated several shared-bus multiprocessor configurations and found that split-transaction protocol offered 1.2x - 2x increase in system performance over a traditional circuit-switched protocol [41]. The benefit of split-transactions increased with system size in fat-tree network and over 5x speedup was reported for 32-core system [1].

The bus signals are *Clk*, *Av*, *Data*, *Comm*, *Lock*, and *Full*. Address and data are multiplexed into a single set of signal lines (*Data*) to save logic and routing resources. Signal *Av* indicates when an address is transferred on the data lines, i.e. address is valid. The number of data bits can be freely chosen. This is beneficial, for example, when error correcting or detecting codes are added to data and the resulting total data width is not equal to any power of two. Three bits are needed for *Comm* to present eight different network commands (*idle*, *write_data*, *write_hi_prior_data*, *multicast_data*, *multicast_hi_prior_data*, *read_request*, *write_config*, *read_config*). Active master asserts *Lock* signal when it reserves the bus. Handshaking is done with the *Full* signal. When *Full* is asserted, the data word on the bus must be retransmitted. To improve modularity, all signals are shared by all wrappers within a segment and no point-to-point signaling is required. Consequently, the interface of a wrapper does not depend on the number of agents and the wrapper can be reused more easily. An OR network was selected for bus signal resolution.

In HIBI v.2, all transfers are bursts, i.e. address is transmitted only in the beginning of the transfer and it is followed by one or more data words. The maximum burst length is wrapper-specific. HIBI uses mainly two-level addressing scheme: the upper bits of the address identify the target terminal (e.g. $destination_0$) whereas the lower bits define the additional identifier. This identifier can be used either as an address to local memory, to select the correct reception channel on DMA, to identify the source of the data, or to select requested service. Certain packet-switched networks (at least



(a) Bus with three wrappers. The signal lines and OR port compose the bus



(b) Bus timing for three agents

Fig. 42. *Example of bus structure and timing of bus transfers*

those implemented in this work) allow only one address per terminal. In that case, the second level address must increase the header length.

The HIBI implementation pays special attention on minimizing the transfer latency by removing empty cycles from the arbitration process by pipelining as shown in Fig. 42. Empty cycles are here defined as cycles when at least one wrapper has data to send but the bus segment is not reserved. An optimized protocol allows lower frequency, and hence lower power, for certain performance level than inefficient protocol. Fig 42(a) shows an example of bus having three agents and Fig 42(b) shows the related timing. Output signals from wrappers are denoted with post-fix *_out* and inputs with *_in*. Wrapper outputs are connected to OR gate and its output is connected to inputs of every wrapper. All wrappers generate internally the two time slot signals (*next_slot_starts* and *curr_slot_ends*, shown on bottom) and they are not transmitted on the bus. The example shows the bus reservation through round-robin arbitration

and TDMA. Agents 1 and 2 gain the ownership through round-robin and agent 3 through both round-robin and TDMA. The example shows only small transfers for the sake of simplicity. In practice, it is more efficient to send tens or hundreds of data words in each turn (cf. Fig. 41). The signals shown in figure are explained in the following. Data values are here selected so that the most significant digit denotes the sender and the two others are the running number of the data word.

1. Agent 1 reserves bus by asserting *Lock*. At the same time, the first address (12) and the address valid (*Av*) signals are asserted. Address if followed by the data word 101. After that, the same agent transfers next address and data without empty cycles. The data and addresses are sent with the normal priority ($Comm = 2$).

2. The time slot of agent 3 starts and therefore agent 1 must release the bus by deasserting the *Lock* signal. Agent 3 transfers one address and 3 data words with high priority ($Comm = 3$) on its turn. Note that *Lock_out_*3 is asserted one cycle before the address. Hence, the empty cycles are avoided even if the time slot was not used.

3. After 4 cycles, the time slot ends and agent 3 releases the bus. *Lock* signal is always deasserted one cycle in advance to avoid empty cycles as shown in figure. According to the round-robin, the next agent in turn is agent 2 which reserves the bus.

4. Agent with the address 28 cannot temporarily accept more data and asserts signal $Full$. Hence, agent 2 must interrupt its transfer. When a transfer is interrupted, there is one empty cycle and after that agents continue transfer in a normal manner.

5. When agent 2 has the ownership next time, it retransfers the data word 201 before sending other data.

Empty cycles appear also when bus utilization is low as distributed round-robin arbitration takes one cycle per agent. If only one agent is transmitting, it has to wait a whole round-robin cycle between transfers. In such cases, the priority-based arbitration is useful.

7.4 Buffering and signaling

The model of computation used in HIBI design approach assumes bounded first-in-first-out (FIFO) buffers between processes. A simple FIFO interface can be adapted to other interfaces such as the OCP (Open Core Protocol) [178]. Consequently, IP components use only OCP protocol and are isolated from the actual network implementation. Ideally, network can be chosen freely without affecting the IPs. However, not all features of HIBI, such as relative data priorities or dynamic reconfiguration, can be used with OCP directly but only the basic transfers.

To avoid excess buffering or retransfers, the received data must be read from the FIFO as soon as possible, for example by using a direct memory access controller. As a result, the receiver buffer space is not dictated by the *amount* of transferred data, but the *latency* of reading data from the wrapper. This scheme resembles wormhole routing, but the links are not reserved if the receiver is stalled.

7.5 Wrapper structure

The structure of the HIBI v.2 wrapper is depicted in Fig 43. The modular wrapper structure can be tuned to better meet the application requirements by using different versions of the internal units or leaving out properties that are not needed in a particular application. On IP side, there can be separate interfaces for every data priority or they can be multiplexed into one interface. Furthermore, the power control signals can be routed out of the wrapper if the IP block can utilize them. The main parts are buffers for transferring and receiving data and the corresponding controllers. The transfer controller takes care of distributed arbitration. The configuration memory stores the arbitration parameters. Relative data priority is implemented by adding extra FIFOs. A (de)multiplexer is placed between the FIFOs and the corresponding controller so that the controller operates only on a single FIFO interface. The separate (de)multiplexer allows adding FIFOs to support priorities in excess of two without changing the control. Currently, transmit multiplexer uses pre-emptive scheduling.

HIBI v.2 has multiplexed address and data lines whereas HIBI v.1 uses separate address and data lines. Multiplexing decreases implementation area because signal lines are removed and less buffering capacity is needed for the addresses. This causes overhead in control logic but that is less than the saving in buffering. Having fewer wires

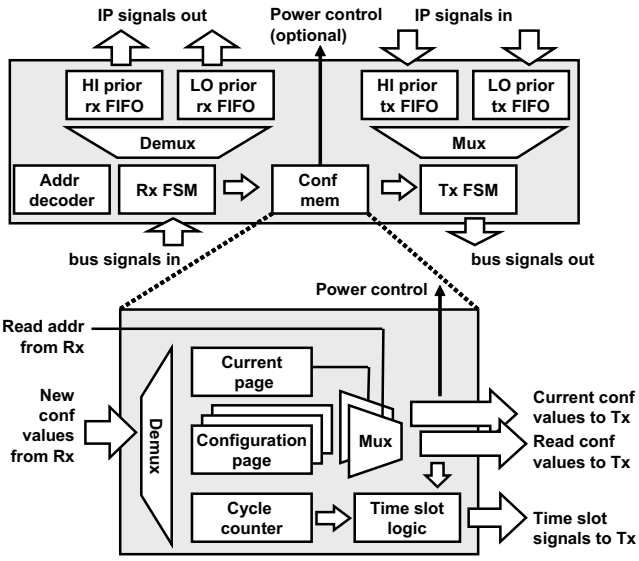


Fig. 43. *Structure of HIBI v.2 wrapper and configuration memory*

allows wider spacing between wires and hence lower coupling capacitance. On the other hand, the saved wiring area can be used for wider data transfers to increase the available bandwidth. The HIBI protocol does not require any specific control signals, but message-passing is utilized when needed. HIBI v.1 assumes strictly non-blocking transfers and omits handshake signals to minimize transfer latency but one handshake signal *Full* was added to HIBI v.2 to avoid FIFO overflow at the receiver. As a result, blocking models of computation can be used in system design and, in addition, the depths of FIFOs can be considerably smaller than in HIBI v.1.

The structure of the configuration memory is illustrated at the bottom of Fig 43. It includes multiple configuration pages for storing the parameter values, a register storing the number of currently active page, clock cycle counter, and logic that checks the start and end of times of the time slots. The receive controller takes care of writing new configuration values whereas the configuration values and time slot signals are fed to the transfer controller. Configuration values can be written to non-active pages before they are used to minimize the risk of conflict when the configuration is performed.

7.6 Runtime reconfiguration

HIBI allows the runtime configuration of all arbitration parameters to maximize performance. This is achieved so that one of the agents (e.g. system controller CPU) writes the new configuration values to all wrappers. The configuration values are sent through the regular data lines. During the normal operation, i.e. when the configuration is not changed, the controller CPU can perform its computation tasks. In the best case, other PEs can continue their transfers even if HIBI is being configured. However, some operations, such as swapping priorities of two wrappers, necessitate disabling other transfers momentarily.

For very regular traffic, the TDMA slots can be set to minimize the latency, i.e. slot starts shortly after the availability of data. For TDMA, each wrapper has an internal cycle counter to decide correct times to access the bus. For this reason, wrappers in one bus segment must be synchronized. When data is produced with varying time intervals or quantities, the time slots cannot be optimally located. By runtime reconfiguration, the cycle counters can be reset to an arbitrary clock cycle value within the time frame to keep time slots in the correct place with respect to data availability. Also the length and owner of the slots can be changed. The resynchronization can be triggered explicitly from software or automatically by a specific monitor unit, which monitors how effectively time slots are used and starts the reconfiguration if needed [106]. Roughly 10 % improvement in HIBI v.1 throughput in video encoding due to dynamic reconfiguration was reported in [138]. Larger gains are expected when several applications are executed on a single platform. Reconfiguration was used in [130] to speed-up the exploration on FPGA. It allowed notably less synthesis runs, each of which took several hours.

As a new feature in HIBI v.2, the second-level arbitration method can be changed at runtime between priority and round-robin or both of them can be disabled. When the second-level arbitration is disabled, only the basic TDMA is used and the slot owner reserves the bus always for the whole allocated time slot. Similarly, only the second-level arbitration is utilized when no time slots are allocated.

In HIBI v.2, three methods are used to improve the configuration procedure. First, by making use of the bus nature, each common parameter can be broadcast to all wrappers. Second, enabling the reading of configuration values simplifies the procedure as the whole configuration does not have to be stored in the configuring agent. In con-

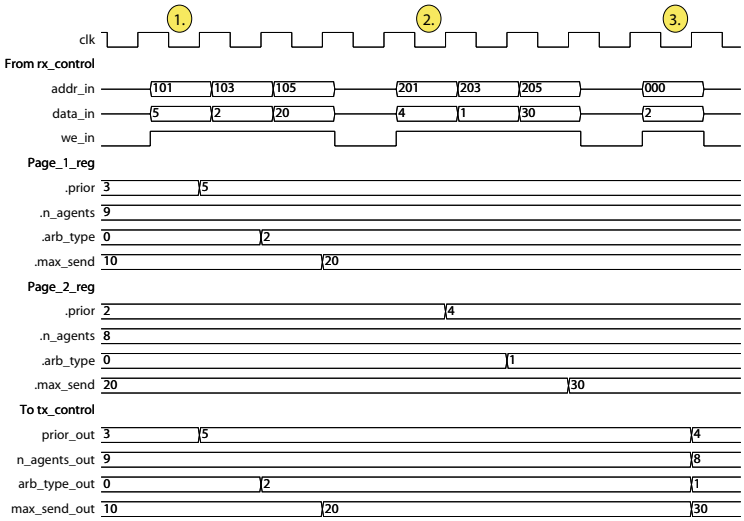


Fig. 44. *Example of runtime configuration*

trast, the configuring agent can read the old parameter values to help determining the new ones. Third, additional storage capacity for multiple parameter pages has been added to enable rapid change of all parameters. When a configuration page changes, all the parameters are updated immediately with one bus operation. It is possible to store a specific configuration for every application (phase) in its own configuration page to enable fast configuration switching.

Runtime reconfiguration is illustrated in Fig 44 for 2-page configuration memory. Signals coming from receive controller to configuration memory (*addr_in, data_in, we_in*) are shown on top. In the middle are the registers *.prior, .n_agents, .arb_type, .max_send* for both configuration pages (all parameter registers are not shown for clarity). On the bottom, are the signals from memory to transfer controller (*prior_out, n_agents_out, arb_type_out, max_send_out*). In the example, the first digit of the address defines the page and two last digits define the parameter number.

1. The parameter registers for priority (*.prior*), arbitration type (*.arb_type*), and maximum send amount (*.max_send*) on current page (page 1) are configured to values 5, 2, and 20, respectively.

2. Parameters on the inactive page are updated: priority is set to 4, arbitration type is changed from round-robin (0) to priority (1), and max_send is increased to

30.

3. Page 2 is activated by writing value 2 to address 0x000. When the page is changed, all outputs to transfer controller change immediately. Since the number of agents (*n_agents*) changes to value 8, the wrapper with priority 9 cannot access the bus anymore. This way arbitration latency can be decreased if some agent is known to be idle.

7.7 Summary and comparison to previous HIBI version

HIBI network allows multiple topologies and utilizes distributed arbitration. The network is constructed by instantiating multiple wrapper components and and connecting them together. The wrapper is modular allowing good parameterization at design time and possibility to reconfigure certain parameters of the network runtime. The most important properties of HIBI are summarized in Table. 17. Several changes were made to the previous HIBI implementation (HIBI v.1) [136] as shown.

Table 17. *Properties of HIBI v.1 and v.2.*

	Property	HIBI v.1	HIBI v.2 (this work)	Change/Impact
Basics	Description language	VHDL, (Matlab)	VHDL, SystemC	SystemC models for exploration
	Topology	(Hier.) bus	Hier.bus	Better support for arbitrary hierarchy levels
	Interface	HIBI-specific -	HIBI-specific OCP (TLM1, TLM2)	Four versions in v.2
	Clocking	Synchronous wrapper Synchronous network	Synchronous wrapper GALS network	- Support for GALS
	Switching type (within segment)	Circuit-switching	Circuit-switching	-
	Switching type (between segments)	Wormhole (packet)switching	Wormhole (packet)switching	-
Configuration	Configuration	Design-time, (runtime)	Design-time, runtime	Runtime reconfiguration improved
	Design-time configurable parameters	Data width, addr width, initial configuration, addresses FIFO sizes -	Data width, addr width, initial configuration, addresses, FIFO sizes Number of config pages and their type (RAM/ROM), included properties	- Wrapper-specific in v.2 Better configurability
	Run-time configurable parameters	TDMA cycle and slots, max send, own priority Own id Own address - - -	TDMA cycle and slots, max send, own priority - - Current TDMA clk cycle Utilized arbitration algorithm Change configuration page	More TDMA slots allowed in v.2 Fixed at synthesis in v.2 Fixed at synthesis in v.2 Synchronization added to v.2 Run-time selection added to v.2 Fast update of multiple parameters
Transfers	Burst transfers	Separate command Data on addr lines also	All transfers are bursts -	More choices on burst length Multiplexed addr+data in v.2
	Multicast	Limited support	Versatile	Addressing was redesigned
	Data priority	None	2-level: regular and high-prior	Differentiated services included
	Handshaking signals	None	One	Included the signal: *target_full*
	Commands	6 Idle, write, read req, write cfg Start burst, continue burst	8 Idle, write, read req, write cfg Multicast, read cfg, write hi prior, multicast hi prior	More - Replaced two with four
Signals	Bus signals	Data [n-1:0] Addr [m-1:0] Command [2:0] Lock -	Data [n-1:0] Addr valid Command [2:0] Lock Full	Multiplexed with address Multiplexing and 1-bit valid signal - - Added to v.2
	Address lines	Separate from data	Multiplexed with data	Less wires and buffers
	Signal type	Bidirectional, all shared	Unidirectional, all shared	Unidirectional better suited on chip
	Signal resolution	Three-state logic	OR-based	OR better suited on chip
Flow control	Arbitration algorithms	TDMA, round-robin, combination -	TDMA, round-robin, priority, combination Random, DAA	Run-times selection in v.2, two versions of TDMA in v.2 New algorithms and combination
	Arbitration implementation	Distributed Pipelined	Distributed Pipelined	- -
	Handshaking	Application-level	Appl + signal level	Allows smaller FIFOs in v.2
	Qos	TDMA, round-robin/prior with limited tx length - - -	TDMA, round-robin/prior with limited tx length Multiple priorities for data Fast runtime configuration TDMA synchronization	- Added Added Added
Usage	Verification	HW sim, HW/SW sim - HW emulation	HW sim, HW/SW sim FPGA prototypes -	More thorough testbenches Much effort into this Not applicable anymore
	Conference publications	11	11	
	Journal articles	1	5	
	Test applications (simulation)	10-CPU H.263 video Synthetic test cases WLAN baseband	10-CPU H.263 video encoder Synthetic test cases WLAN baseband	- More cases Also distributed simulation with v.2
	Test applications (FPGA)	- - -	H.263 H.263 + WLAN (FPGA) MPEG-4, up to 35 CPUs + accelerators	Added Added Largest architecture requires 3 FPGAs

8. BENCHMARKED REFERENCE NETWORK-ON-CHIPS

This Chapter presents reference networks and a network interface implemented during this work. To allow fair comparison, the networks are compared by using synthesizable building blocks. Studied networks utilize the same interface to processing element (PE) which allows using exactly the same traffic generator (modeling the PEs) in all cases.

The router structure is quite similar in reference mesh, ring, and octagon whereas the wrapper of packet-switched bus resembles HIBI. At first, all topologies were implemented using packet-switching with store-and-forward buffering. It is easy to implement but may be sub-optimal regarding the latency and the required buffering area. Hence, other versions of each topology were developed by implementing cut-through and wormhole flow control policies also. All networks utilize deterministic routing and hence, the data always arrive in-order. Furthermore, NoCs use identical packet structure.

8.1 Network interface

The transferred data must be structured into packets before injection to a packet-switched network. Fig. 45 shows the approach used in this work. The data stream is coming from the resource (PE, memory, external interface) and the necessary packet headers are added by the component dubbed as *pkt_enc*. The address of the data stream is translated into the NoC address with an address look-up-table (addr LUT). For example, a 32-bit address, say $0x5500F000$, may be directly utilized with shared bus topology. However, with mesh it translates to address $(2,1)$ and with octagon to $(1,1,2)$.

The structure of the packet is configurable. The NoC address is always the first flit of the packet. The length of the packet is either sent as part of address flit or as 2nd

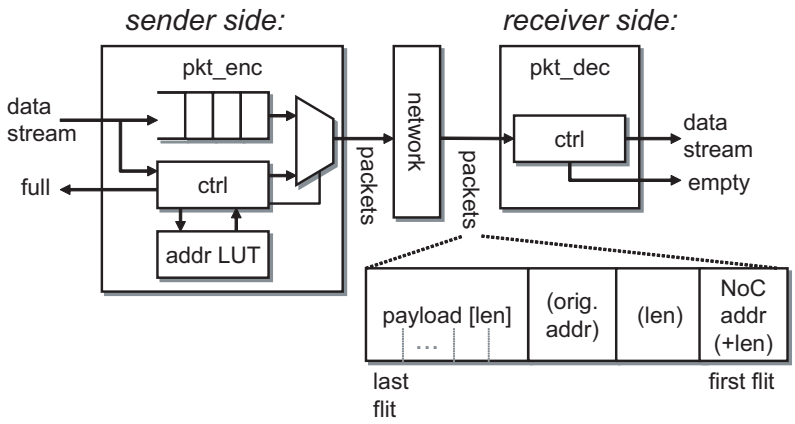
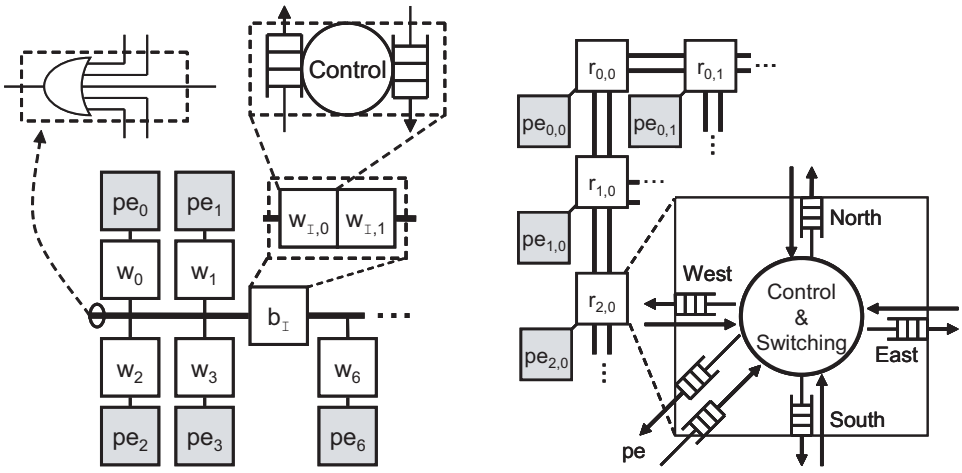


Fig. 45. *Network interface logic needed for constructing packets.*

flit of the packet. The original address ($0x5500F000$) is the last field of the header, if included. The source identifier is encoded into the original address if necessary. The header is 1 to 3 flits long and followed by the data payload. Dummy stuffing data can be optionally inserted to create fixed-length packets if that is required by the network. Currently, no tail flits or packet numbering are used. All the studied packet-networks inspect only the NoC address and packet length, and handle all the rest as data payload.

At the sender side, the data stream is first collected into a FIFO to determine the length of the packet. The length of the FIFO is the same as the maximum amount of payload per packet. The packet is sent if the data stream stops or its address changes. When the data stream stops, the controller first waits for a pre-defined number of clock cycles before sending. Without waiting, any short interruption in the data stream would initiate packet transfer. That leads to shorter average payload length and hence emphasizes the overhead caused by the headers. Moreover, fixed-length packets include also the overhead from the dummy data. The reception of the next stream to the FIFO can happen while writing the previous packet to the network. The logic at the receiving end is much simpler. It only removes the packet headers and the possible dummy data. Only the address and payload are forwarded to the resource.

The presented scheme can operate in general case where the length of the data stream is unknown. The main reason was the fact that utilized resources or traffic generators



(a) Hierarchical bus (b) 2-dimensional 4-way mesh

Fig. 46. *Network implementations.*

do not readily provide the transfer length as an output. However, knowledge about the length prior to transfer improves the procedure. The sender tells the length using the input port *len*. The area is saved by removing the FIFO buffer from the *pkt_enc*. The packet latency is also reduced by forwarding the packet already when receiving the data stream. However, the measurements in this thesis were performed assuming a general case.

8.2 Packet-switched bus

Processing elements (pe_i) are connected to bus segments via wrappers (w_i) and together they form an agent. The traditional single bus topology can be extended to a hierarchical bus scheme as shown in Fig 46(a). Extension is done by using bridges (marked with b_I) to connect several bus segments together. In this case, each segment operates with the same frequency and has the same number of signal lines. The number of segments is scaled with the number of agents.

The implemented bus wrapper is a fairly simple device with two FIFO buffers and a control unit for arbitration. Arbitration is based on a distributed round-robin scheme. In the packet-switched bus, the ownership is passed to the next wrapper after each transmitted packet. As in HIBI, the bridge components are implemented by con-

necting two wrappers together and the bus signal resolution is implemented with an OR-based structure as shown in Fig 46(a). In the hierarchical bus, the problems inherent in long signal wires are solved by grouping only a limited number of agents, in this case four, in each bus segment. Therefore, single bus and hierarchical bus topologies are, in this case, the same for a system with 4 agents.

8.3 Packet-switched mesh

Fig 46(b) depicts a packet-switched 2-dimensional 4-way mesh used in this thesis. The processing elements are connected to an array where router elements ($r_{y,x}$) handle the storing and forwarding of the data. A router comprises of a control unit taking care of switching and routing, and a FIFO buffer for each direction (North, East, South, West, and two for PE). Note that West FIFO can actually be removed from the highlighted router that resides on the mesh boundary. Removing such unnecessary FIFOs clearly reduces the mesh area: from 9% (64 routers) up to 49% (4 routers). Routers implement a simple dimension-order routing scheme where the transfers are first directed to the correct row and then to the requested column. This scheme uses the minimal path but does not allow certain turns and, hence, avoids deadlocks [51].

8.4 HERMES mesh

HERMES is a mesh-based NoC presented in [169] and its source codes are publicly available [54]. There are only minor differences with reference mesh, for example, a couple of adaptive routing algorithms have been implemented and crossing a link takes two clock cycles due to handshaking. However, the utilized test environment requires that all data arrives in-order and, therefore, only the deterministic XY-routing is utilized here. The interface is different to other utilized networks as the HERMES router pulls the data at the sender side. The compatibility was achieved by adding extra FIFO to routers inputs. Its area is not considered in the results. Furthermore, the VHDL codes were somewhat modified in order to allow synthesis and few bugs were also corrected.

8.5 Packet-switched ring and octagon

The router design is very similar in ring, mesh, and octagon topologies, only the number of ports and routing algorithm changes. The octagon routers and ring bridges have 4 bidirectional ports whereas octagon bridges have 5 ports. This reference octagon was inspired by [107] but may differ in some details. For example, only packet-switching is supported.

8.6 Circuit and packet-switched crossbar

Crossbar network was implemented in several configurations. Three switching schemes are supported: circuit, store-and-forward, and wormhole. Furthermore, two arbitration styles are supported. The on-going transfers are handled similarly in both schemes and the difference is related only the initiation of new transfers. Sequential arbitration checks one input per cycle for new transfers. This increases latency but minimizes area.

Parallel arbitration, on the other hand, checks all the inputs every cycle. The latency decreases but area increases. In all configurations, the switch matrix is implemented with multiplexers to allow easy synthesis with various technologies.

9. COMPARISON OF NOC IMPLEMENTATION RESULTS

This chapter presents implementation results of NoCs. The results from literature are presented first and then from reference NoCs. The studied properties include silicon area, operating frequency, routing latency, and power consumption.

9.1 Implementation results from literature

Table 18 lists examples of reported implementation results for NoCs [P10]. FPGA results are omitted for brevity. The results are given per router or wrapper, but their number in the full network depends on the topology. The results are sorted according to processing technology [1], topology, and then alphabetically.

Table 18 includes also results for networks that are used as reference in the studies (marked with prefix *Ref.*).

9.1.1 Router parameters

For a given topology, the area is mainly defined by the flit width and the buffer size. The flit width refers here to the data payload alone and the control signals are excluded. The most common width is 32 bits which is significantly smaller than 256 bits assumed in [52]. Few approaches use wide flits, for example 96 bits [207], 128 bits [163], and 256 bits [40]. Note that the flits are used for flow control (reserving buffers and links) whereas the phit (physical digit) denotes the number of parallel wires between the routers. However, they have the same width in most cases.

The reported buffer depths range from 1 flit to 30 flits but, unfortunately, all sources do not document the buffer sizes. However, several sources report buffers taking 50-90% of the router area and major part of the power consumption, see for example

[1] Note that some feature sizes are grouped together, for example 90 *nm* and 100 *nm*.

Table 18. *Implementation result examples of NoC routers [P10].*

#	Network	Author	Ref.	Top.	Flit width [bit]	Buffering ports	Buffering VC	Buffering flits	Tech. [nm]	Min. lat. [cycle]	Min. lat. [ns]	Router area [mm^2]	Router area [kgates]	Freq. [MHz]	Detail level[(1)]
1	Orion/Luna	Soteriou	[228]	m	64	5	4	16	50	7	0.7	0.11	-	10000	pred.
2	TeraFLOPS	Vangal	[236]	m	32	4	2	16	65	5	1.2	0.34	53	4270	C
3	Xpipes	Bertozzi	[22]	c	32	4	1	?	100	7	-	0.1	-	-	pred.
4	Example NoC	Dally	[52]	t	256	5	8	4	100	3?	-	0.59	-	200-2000	pred.
5	GDB	Hosseinaba	[87]	db	22?		n/a		90	-	-	0.1-0.15	-	510-590	S
6	HIBI v.2, rom	Salminen	[P6]	b,(hb)	32	2	1	2,8	130	4	9.2	0.03, 0.05	4, 8	435	R
7	Ref. octagon	Salminen	[P11]	er	32	4	1	2,8	130	4	9.2	0.04, 0.09	6, 15	435	R
8	Mondinelli	Mondinelli	[166]	ftr	16	8	1	8+1	130	-	-	<0.29	<21	200	C
9	SPIN	Andriah.	[6]	ftr	32	8	1	4?	130	-	-	0.24	-	200	L
10	XGFT	Kariniemi	[109]	ftr	32	3,6	1	7	130	-	-	0.16, 0.25	16, 21	400	R
11	Aethereal router	Rijpkema	[207]	m	3 x 32	5	1	8	120	-	-	0.26	-	500	L
12	Aethereal NI	Radulescu	[201]	m(NI)	3 x 32	4-8	1	8/3	130	4	8.0	0.17-0.25	-	500	L
13	Anoc	Beigne	[18]	m	32		n/a		130	-	2-2.5	0.25	20	250	L
14	Faust	Lattard	[141]	m	32	5	2	?	130	-	0.6	~0.6	19	async.	C
15	Kavaldjiev	Kavaldjiev	[110]	m	16?	5	4	4?	130	-	-	0.18	-	500	S
16	Mango	Bjerreg.	[25]	m	32	5	8	1	130	-	-	0.19	-	795	R
17	Ref. 2-D mesh	Salminen	[P11]	m	32	6	1	2,8	130	4	9.2	0.08, 0.14	13, 24	435	R
18	Ref. Hermes	de Mello	[54, P11]	m	32	5	1	2,8	130	10	23.0	0.05, 0.11	9, 18	435	R
19	SDM NoC	Leroy	[148]	m	32	5	1-32	1	130	-	-	0.03-0.14	-	2270	L
20	Wolk., circ, pkt	Wolkotte	[250]	m	16	5,5	1,4	1,4	130	-	-	0.05, 0.18	-	1075,507	R
21	Ref. seg. bus	Chang	[38]	b	69		n/a		180	-	3.2	0.18	-	-	R
22	SocBus	Sathe	[219]	c	16	3	1	1	180	6	setup	0.04-0.08	-	1200	R
23	Asoc	Liang	[151]	m	32	4	1	2	180	-	-	0.07	-	400	L
24	Crossroad	Chang	[38]	m	69	5	1	1	180	-	3.5	0.06	-	-	R
25	DyAD mesh	Hu	[89]	m	32	5	1	8	160	-	-	-	26	333	C
26	Lochside	Mullins	[172]	m	64	5	4	4	180	1	4.0	0.5	-	250	C
27	Ref. pkt mesh	Chang	[38]	m	69	5	1	8	180	-	29.7	0.65	-	-	R
28	Proteo	Siguenz.	[223]	r	8	3	1	7	180	-	-	0.20	-	-	R
29	Slim-spider	Lee, K.	[144]	x, c	8		n/a		180	-	-	-	-	1600	C
30	Chi	Chi	[40]	m,t	256	5	4	4	250	-	-	2.89	-	150	C
31	Chain	Rostislav	[210]	b	-		n/a		350	-	-	-	4.75	async.	R
32	AET	Valtonen	[235]	m	16		n/a		350	3	26.3	0.8	15	114	L
33	Hermes	Moraes	[169]	m	32	5	1	5-30	350	10	-	-	10-49	(25)	R
34	Qnoc, async	Rostislav	[210]	m	-		n/a		350	-	4-9	0.09-0.47	2-12	async.	R
35	Qnoc, sync	Rostislav	[210]	m	-		n/a		350	1	3.7	0.2-0.96	4-18	270-303	R
	# reported	35	35		32		27		35	14	17.0	31	15	30	35
	% reported	100	100		91		77		100	40	49	89	43	86	100
	AVG (all)				50.1	4.8	3.1	6.4	169.9	5.2	9.2	-	17.9	-	R
	AVG (130 nm)				57.4		-		-	5.5	12.3	0.14	14.9	596	-
	AVG (180 nm)				41.9		-		-	3.5	10.1	0.22	26.0	328[(2)]	-

[(1)] pred.=predicted, C=chip, L=layout, R=RTL

[(2)] 756.6 MHz including [219][144]

[110, 144, 223]. Buffers are usually constructed from flip-flops but using SRAM is also viable, especially on FPGAs. Special FIFO design can offer clear area savings [207].

When specified, the buffer size is listed as a product of number of ports, virtual channels per port and depth of a buffer in flits. In other words $ports \cdot VC \cdot flits$, average being 5 x 3 x 6 flits per router. The number of ports is defined by the topology, mesh being the most popular (5 ports: PE, North, East, south, and West) [2]. Circuit-switched proposals are often called "bufferless" but 1-flit buffers at each output are assumed for them.

Multiple virtual channels (VCs) may be associated with each physical port. Each VC has a separate buffer which reduces blocking and hence improves performance. They are needed by certain routing algorithms to ensure correct operation without deadlock. Virtual channels are omitted in most current implementations. However, Mango supports 8 VCs [25] and circuit-switched SDM up to 32 VCs per port [148]. Virtual channels are necessary in adaptive routing schemes to avoid deadlocks. Higher VC count also increases performance up to 4-8 VCs/port [190].

9.1.2 Minimum latency

The minimum header latency induced by a router is given in column *Min. lat.*, in clock cycles and/or nanoseconds. The shown values are for the optimum case without contention. Once the header has been forwarded, the payload is transmitted (usually) at the rate of one phit per clock cycle.

Latency depends on the level of pipelining being about 5 clock cycles on average but details are not usually available. A basic 5-cycle router pipeline has the following stages: input buffering, virtual channel handling, routing, output arbitration, and switching. Detailed analysis of router pipelining is presented in [193].

A novel design of router allows single-cycle latency [172] and a look-ahead mechanism that performs routing decision for the next router is presented in [40]. SoCBus [219] reports a 6-cycle latency for circuit-setup whereas the latency for data transfer is most likely one cycle. A combined latency of 10 cycles for the two network

[2] Reference mesh on the bottom has two buffers for PE, one for tx and the other for rx.

interfaces (sending and receiving ends) was reported in [22]. The latency of Aethereal NI is $4-10$ cycles whereas SW implementation of packetization takes nearly 50 cycles [201]. In [141], both NI and GALS interface introduce a double latency compared to a single router (12 *ns* vs. 6 *ns*). According to [4], initiating the DMA send operation in the Cell Broadband Engine takes about $53-65$ processor's cycles in the best case and over 90 cycles in the worst case. The reported HW latencies are low enough in author's opinion, especially when the overheads regarding the middleware and network stack are included. Minimizing the latency and memory overhead of micronetwork stack that still enables simple reuse is an open research problem.

9.1.3 Area

The router area is given in mm^2 or kilogates. These values are meant only for coarse-level comparison since the router parameters or inclusion/exclusion of wiring area in results are not always stated. This is rather counterintuitive to note because the area is the most often reported metric (cf. Tables 2 - 5).

The reported gate count per router is around 18 kilogates on average, which is reasonably low for large MPSoCs. Reported average area at 130 *nm* technology is $0.14mm^2$. Circuit-switching allows even smaller routers due to absence of buffers, see for example [148, 151, 250]. Sylvester and Keutzer [232] have estimated PEs with complexity of about $50-100$ kilogates will allow dependable design and verification also in the future. Hence, an average router ($15-18$ kilogates) would impose a notable overhead in chip area, roughly $15-36\%$ even without NIs. Area overheads in the range $9-45\%$ were reported in [190]. The overhead gets larger if separate wiring channels are reserved for the signals between the routers. Hence, NoC area optimization is encouraged when NoCs are used with modest-sized IPs.

Network interface (NI) handles the packetization, and possibly re-ordering or retransmissions. Details about the needed network interface are too often omitted. However, interface may double the required silicon area, as in [201] and [207]. In [141], the sum of router and the associated network and GALS interfaces, is $19+10+12=41$ kilogates per NoC terminal.

Some sources, for example [52, 132, 172, 236], assume quite large blocks to be connected via NoC, for example $1-9mm^2$, which would clearly diminish the relative

area overhead. For example, TeraFLOPS has 3 mm^2 tiles and 53-kilogate routers account about 17% of the transistors. Large tiles consist of few hundred kilogates, which is large enough to necessitate a local network, for example bus (for area-optimization) or crossbar/point-to-point (performance optimization). This is a natural spot for the hierarchical NoCs.

9.1.4 Operating frequency

The next column shows the reported operating frequencies. Similar to the area, wiring has notable impact on the frequency in deep sub-micron technologies. The results are for the router only assuming that the link delays do not affect. Few NoCs are asynchronous [18, 25, 63, 141, 153, 210, 252]. Most NoCs use synchronous components that may be asynchronous with respect to each other (globally asynchronous, locally synchronous).

The average frequency is about 600 MHz for 130 *nm* technology which seems adequate for cost-optimized, medium-performance devices. However, few very fast NoCs [148, 250] increase to the average value notably.

There are great differences between the fastest and the slowest results; over $11x$ with 130 *nm* and $6x$ with 180 *nm* technology. An order of magnitude difference is so large that one cannot assume anything about the NoC's performance unless it is explicitly given.

The last column shows the detail level. Most implementation results are obtained from synthesis tool but good results are obtained also in real silicon implementations: 1.6 GHz for Slim-Spider at 180 *nm* [144], 1.35 GHz for Nexus at 130 *nm* [153], and astonishing $4.27 - 5.67$ GHz for TeraFLOPS at 65 *nm*.

9.1.5 Router power

Power consumption is omitted from the Table 18 since it is not a "static" metric in the same sense as the area or maximum frequency. Hence, large variations in reported power results are apparent. Leakage power is proportional to area but dynamic power is related to the sustainable traffic rate of a NoC and hence application-dependent[3].

[3] Leakage is also application-dependent if power supply gating is applied.

Furthermore, the power depends on the used wire model and required wire length which both require knowledge about the layout.

Further analysis of power consumption in NoCs can be found, for example, in [14, 22, 140, 142, 142, 144, 171, 194, 199, 236, 239, 253, 255]. Table 19 summarizes few studies regarding the power consumption of NoCs. Table shows the proportion of routers and links from the NoCs power, their system-level impact, and proportion between dynamic and static power consumption. Wires are sometimes predicted to become very power-hungry, e.g. [52], but the quantitative analyses in the table do not fully support that claim. At least in multi-hop topologies, the wires account for roughly $20-30\%$ of the network power. Ye *et al.* [255], note that storing a packet into a buffer consumes far more energy than transmitting it on interconnect wires on 180 nm technology. Futhermore, buffer power increases with network throughput due to contention.

The impact of static power consumption rises with shrinking feature size, from 2% to 18% when moving from 180 *nm* to 90 *nm* and below. However, Pullini *et al.* have analyzed power consumption of NoC assuming 65 nm processing technology and nominal operating conditions [199]. In their experiment leakage power contributed only negligible $0.46\% - 2.42\%$ of the switch power

In [8], mesh has over $5x$ power (377 mW vs. 72 $m\,W$) compared to multi-layer AMBA bus but at the same time runtime is $10-15\%$ shorter. Consequently, mesh consumes more energy in this case. However, the total energy of the system depends also on the PEs: with high-power PEs, the time saving may reduce the total energy as well. Hence, differences between networks appear larger when considered in isolation than as part of the whole system. Either way, to minimize dynamic power, the lowest frequency that meets the application demands is sought. It may be set at runtime as the computational/communication load varies. This is very efficient for power reduction when combined with supply voltage scaling. Furthermore, estimating and lowering frequency prior to synthesis avoids over-design, for example large signal drivers. In addition, low-leakage logic gates can be opted with less stringent delay constraints.

Table 19. *Relative power consumption in NoCs*

#	Author	Ref.	Topology	Technology [nm]	Breakdown of NoC power		NoC power from total [%]	Power	
					Routers [%]	Wires [%]		Dyn. [%]	Static [%]
1	Pullini	[199]	mesh	65	-	-	-	98-100	0-2
2	Vangal	[236]	mesh	65	83	17	28	84-94	6-16
3	Banerjee	[14]	mesh	90	62-77	23-38	-	44-68	32-56
4	Mullins	[171]	mesh	90	72-88	11-28	-	78-92	8-22
5	Lambrecths	[140]	custom	130	89	11	2	-	-
6	Xu	[253]	xbar	130	31-47	53-69	-	-	-
7	Ye	[255]	custom	180	98	2	-	-	-
8	Penolazzi	[194]	mesh	180	46-49	51-54	-	100	0.003
9	Vellanki	[239]	mesh	180	96	4	-	96-98	2-4
10	Lee, H.G.	[142]	hier.xbar	XC2V4000[(1)]	59-76[(2)]	3-8	21-31	-	-
	min (65-90 nm)			62	62	11	28	44	0
	avg (65-90 nm)			107	76	23	28	82	18
	max (65-90 nm)			180	88	38	28	100	56
	min (130+ nm)			31	31	2	2	96	0
	avg (130+ nm)			144	65	35	18	98	2
	max (130+ nm)			180	98	69	31	100	4

[(1)] Xilinx Virtex II FPGA

[(2)] network interface consumes 20-35% of NoC's power

9.2 Implementation results of studied networks

9.2.1 Reference networks

Table 20 shows the synthesis results of the reference networks routers[4]. In addition, there are results for a network interface, DMA controller, and crossbar with 4 and 16 terminals. The network components are described in Section 8 and HIBI's result will be discussed in more detail in the following Section.

Reference buses support hierarchical structures and results are for one wrapper. The reference octagon is based on [107] but uses packet switching only. HIBI and bidirectional ring are the smallest since they have least ports (= buffers). The reference mesh and Hermes are the largest. Note that all others than HIBI and circuit-switched crossbar require an NI, the area of which equals one average router.

The DMA controller can be used with all of these networks but it is not mandatory. In this thesis, DMA is commonly included to all CPU-subsystems but not used with HW accelerators. It does not have much buffering but the area depends heavily on the number of reception channels (marked as *VC* in table). Each channel requires storage for incoming address, address to local memory, expected data amount, and a counter for the received data amount.

The relative areas of 16-terminal reference NoCs are shown in Fig. 47. Three values are shown, depending on whether the network interface and DMA controller are included. Values in each data set are scaled so that the smallest has an area of 1.0. Hence, one should compare NoC within one data set and not across the sets. Note that ratios between NoCs change clearly between the sets. This happens especially when the NI can be omitted: HIBI and circuit-switched crossbar are among the smallest in all cases. Hence, it is important to clearly state how additional logic (e.g. NI or DMA) belongs to the compared "network".

All components were synthesized targeting 400 MHz frequency and all meet that limit except the 16-node crossbar. Furthermore, the area of the crossbars grows fast: increasing the number of terminals by $4x$ (from 4 to 16), the logic area increases by $6.4 - 8.3x$, depending on the depths of the FIFO buffers. The reason is that the area of the switch matrix grows quadratically and the FIFO area linearly. Hence, a

[4] Some results are copied from Table 18 to simplify comparison.

Table 20. *Implementation results of reference NoCs.*

#	Network/ component	Author	Ref.	Top.	Flit width [bit]	Buffering ports	Buffering VC	Buffering flits	Tech. [nm]	Min. headerlat. [cycle]	Min. headerlat. [ns]	Area [mm^2]	Area [kgates]	Freq. [MHz]	Detail level[1]
1	Network interface	Salminen	this	-	32	2	1	11	130	12	29.7	0.06	10	404	R
2	DMA ctrl	Kulmala	[125]	-	32	2	8	1	130	1	2.5	0.09	15	400	R
3	Ref pkt. bus	Salminen	[P8]	b, (hb)	32	2	1	8	130	10	24.9	0.04	7	401	R
4	HIBI v.2, rom	Salminen	[P6]	b,(hb)	32	2	1	2 , 8	130	4	9.9	0.03 , 0.05	4 , 8	403	R
5	Ref. octagon	Salminen	[P11]	er	32	4	1	2 , 8	130	4	9.9	0.04 , 0.09	6 , 15	403	R
6	Ref. 2-D mesh	Salminen	[P11]	m	32	6	1	2 , 8	130	4	9.9	0.08 , 0.14	13 , 24	403	R
7	Ref. Hermes	de Mello	[54, P11]	m	32	5	1	2 , 8	130	10	24.8	0.05 , 0.11	9 , 18	403	R
8	Ref. bidir ring	Salminen	this	r	32	3	1	2 , 8	130	4	9.5	0.03 , 0.07	4 , 11	423	R
9	Ref. crossbar	Salminen	this	x	32	4	1	2 , 8	130	4	10.0	0.05 , 0.14	8 , 23	400	R
10	Ref. crossbar [2]	Salminen	this	x	32	16	1	2 , 8	130	4	10.4	0.40 , 0.89	67 , 147	384	R
AVG					32	3.3	1.8	5.3	130	5.7	14.6	0.071	11.7	404	R

[1] pred.=predicted, C=chip, L=layout, R=RTL

[2] excluded from average calculation

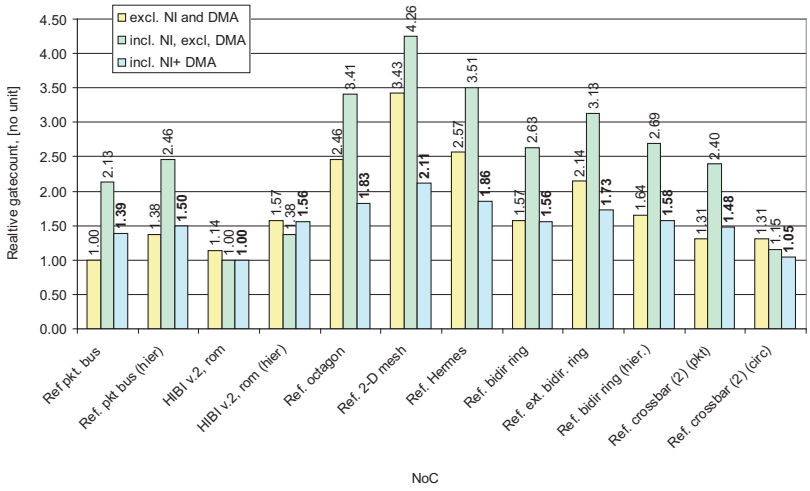


Fig. 47. *The relative areas of reference networks. Calculated for 16-terminal networks. Values in each data set are scaled so that the smallest has an area of* 1.0.

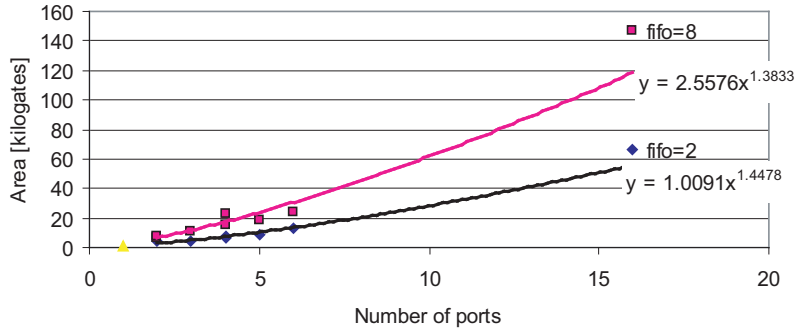


Fig. 48. *The area of network (router) as a function of degree.*

crossbar can be used in small configurations and this limits also the degree of network routers since they include a crossbar. This phenomenon is illustrated in Fig. 48 using automatic trend line fitting. Similarly, it was found in [199] that routers should have at most 10 or 14 ports.

The different versions of the crossbar (serial vs. parallel arbitration, packet-switched vs. circuit-switched) resulted in practically identical synthesis results. However, the latency varies: serial arbitration takes one cycle more than parallel and store-and-forward has overhead relative to the packet length. Circuit-switched and packet-switched crossbars have equal latency but the latter needs a network interface which

has its own latency overhead.

The reference NoCs imply a 4-cycle latency overhead in most cases and Hermes takes 10 cycles. The packet-switched bus and NI are implemented only in a store-and-forward manner which means that the latency depends on the length of the packet. DMA is the fastest as it causes only one cycle overhead.

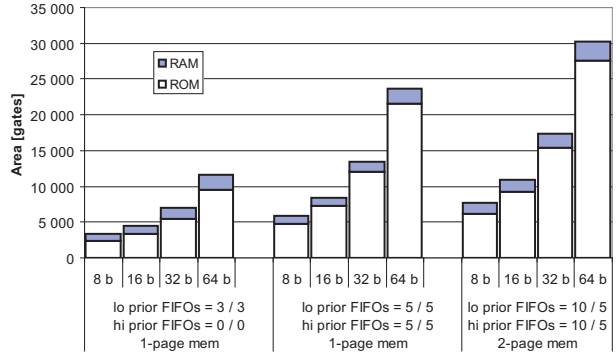
9.2.2 HIBI version 2

HIBI v.2 was implemented with synthesizable RTL VHDL and SystemC. In addition, the transaction-level SystemC model has an embedded performance monitor which is used to analyze and tune the architecture to better fit the application. It also offers a speedup of 20x-75x compared to RTL VHDL simulation and 9x-30x speedup compared to RTL SystemC.

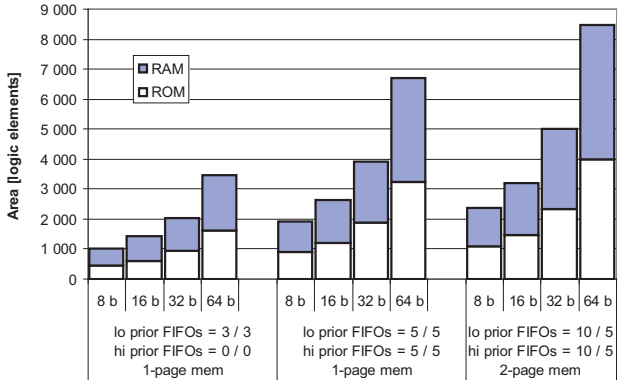
The HIBI v.2 wrapper was synthesized using a 0.18 μm CMOS ASIC technology and Altera FPGA. No scan chains were synthesized. The ASIC results do not include the wiring or place-and-route information. Nominal silicon conditions were used. Areas for different wrapper configurations with four different data widths are shown in Fig 49(a) and 49(b). White bars (denoted with *ROM*) show cases where all configuration parameters are fixed at synthesis time. Darker color bars (*RAM*) show the overhead of enabling runtime reconfiguration.

The first wrapper has 3-word buffers for transmitting and receiving but no separate buffers for high priority data. The configuration memory has one page. The second wrapper has also high priority buffers and buffers are also bigger than in the first one. The third wrapper has the biggest buffers of the three and includes 2-page configuration memory. On FPGA, wrapper with ROM-type memory results in relatively smaller area than in ASIC. This is most likely due to different synthesis tool that can better optimize the logic with constant input values. The maximum frequency on ASIC is in the range of 245 to 420 MHz depending on the selected configuration. On Stratix II FPGA, frequency varies from 81 MHz to 120 MHz. For comparison, soft Nios processor developed by Altera achieves about 120 MHz on the same FPGA.

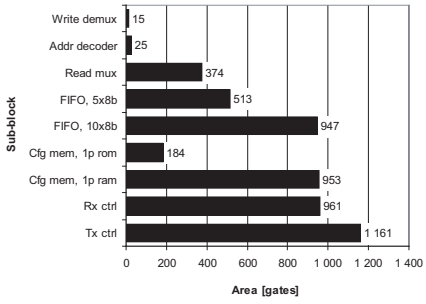
In these syntheses, the configuration memories and FIFO buffers are implemented with flip-flops to support soft IP core methodologies, which results in rather large area. This is illustrated in Fig 49(c) for an eight-bit wrapper. In many cases, the



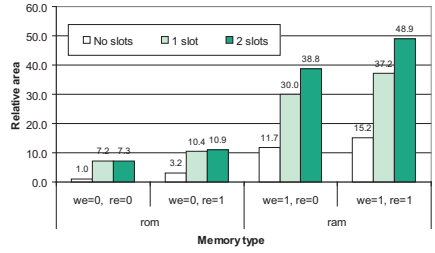
(a) Area of the HIBI v.2 wrapper on 0.18 μm ASIC



(b) Area of the HIBI v.2 wrapper on Altera Stratix II FPGA



(c) Areas of sub-blocks in 8-bit wrapper on ASIC



(d) Area of different flip-flop-based configuration memories on ASIC. Abbreviation *we* denotes write enable and *re* read enable

Fig. 49. *Area of HIBI v.2 wrapper and its sub-blocks*

FIFO buffers occupy more than half of the wrapper area. Still, the area of FIFO buffers is smaller than in the first version of HIBI due to multiplexed $address/data$ lines.

The number of configuration pages, time slots, and the width of parameters can be parameterized according to the application to minimize the implementation area. Naturally, having multiple pages results in a large implementation area. A way to reduce the area is to restrict the configuration options by hardwiring certain values. Still, it is possible to leave some parameters to be configured at runtime. For example, allowing clock counter synchronization and the use of multiple hardwired configuration pages might be a good compromise in many cases.

Fig 49(d) shows the effect of configuration memory parameters on the area of a single page memory. The reading of configuration values is not always needed and this feature can be disabled ($re = 0$) which saves area by roughly 15%. Even bigger savings are obtained by using hardwired memory ($we = 0$) that does not allow runtime reconfiguration. When time slots are not used, the clock cycle counter and time slot logic become obsolete and area is reduced, see Fig 43. Having more than one time slot has only minor effect when the slot parameters are hardwired.

9.3 Reference NoCs relative to literature

The NoCs implemented for this thesis appear to be “representative” NoCs when their properties and implementation results are compared to those from literature. Hence, using them for performance comparison is justified.

10. COMPARISON OF NOC PERFORMANCE RESULTS

This Chapter compares the performance of NoCs. First, published results from literature are reviewed. Second, implemented reference NoCs are compared using synthetic traffic in simulation. The impact of HIBI parameter selection is studied in more detail. Third, simulation-based evaluation of a video encoder application is presented. Finally, findings obtained from HIBI in FPGA prototypes are summarized briefly.

10.1 Performance results from literature

Comparison results from literature are summarized in Table 21 with emphasis on runtime comparisons. The compared topologies are listed first, followed by citation information, and then the runtime ratio when available. When the ratio is < 1.0, NoC_1 is faster and vice versa. For example, a runtime ratio $2x$ means that NoC_2 offers *two-fold speedup* because NoC_1 takes twice the runtime compared to NoC_2. The last columns show the ratio of other studied metrics and reference to the source.

Sometimes, the results from the literature may sound contradicting at first but it must be noted that the performance depends on the application. The various studies have simply utilized test cases with different characteristics and requirements. There are also large differences between various configurations of the same topology. For example, circuit-switched mesh outperforms packet-switched in [151]($1.1-2x$ speedup) and in [38] ($13x$ energy reduction, $6x$ speedup). On contrary, packet-switching offers $2.5-3.5x$ speedup in [121]. For buses, different versions can have $1.4-3x$ and $9.5x$ difference in runtime [P8] [81].

Mesh and hierarchical bus (hier.bus) achieve shorter runtimes than single bus in most cases. However, both single and hierarchical buses require smaller areas than mesh. Crossbar is also superior to bus but suffers from unacceptable area in large systems.

Table 21. *Reported differences between NoCs [P11]*

#	Compared NoCs		Author	Ref.	Runtime ratio	Other ratio
	#1	#2			#1 /#2	#1 /#2
1	bus	mesh	Bolotin	[28]	1x	$N^{2.5}$area, $N^{0.5}$ power
2			Kreutz	[121]	0.1-0.5x	-
3			Liang	[151]	1.1 - 3x	-
4			Shen	[222]	1.2x	1.4x energy
5			Hilton	[81]	2x	0.8x area
6			Thid	[234]	0.3-0.9x, 3-6x	-
7			Salminen	[P8]	1-50x (11x avg)	0.3x area
8	bus	hier.bus	Bolotin	[28]	1x	Nx area, 1x power
9			Liang	[151]	0.9x	-
10			Ryu	[211]	1.4-2.4x	0.3-1.3x area
11			Salminen	[P8]	1-30x (10x avg)	0.7x area
12			Lahiri	[134]	-	0.3-0.6x TP
13	bus	crossbar	Pimentel	[196]	1.5x	-
14			Xu	[253]	1.6x	1x area
15			Loghi	[155]	1.5-2.5x	-
16			Lahtinen	[139]	1.1-4x	0.5x area
17	bus	ring	Lahiri	[134]	-	1.5-3x TP
18			Liljeberg	[152]	-	1.3-1.9x TP
19	bus	fat-tree	Adriahantenaina	[1]	-	0.2x sat.point
20		multibus	Angiolini	[7]	4-7x	-
21		split bus	Lu, R.	[156]	-	0.2x BW, 5.5x lat.
22		octagon	Dumitrascu	[58]	1x	3x lat.
23		p2p	Hu	[88]	-	4x energy
24		tree	Kreutz	[121]	0.3x	-
25	mesh	ext.mesh	Ogras	[181]	1.5x	0.9x area, 1.5x energy
26		hier.bus	Liang	[151]	0.2-0.9x	-
27		hier.mesh	Zhang	[259]	-	1.3-1.9x energy
28		p2p	Lee, H.G.	[142]	1x	0.6x power
29		tree	Kreutz	[121]	0.9x	-
30	mesh	crossbar	Bartic	[17]	-	0.8x area/BW
31			Lee, K.	[144]	-	1x area, 2x energy
32	mesh	multilayer-bus	Angiolini	[7]	0.9x	1.9x area (NoC)
33			Angiolini	[7]	0.9x	1.3x area (chip)
34			Zhang	[259]	-	0.1-0.4x energy
35			Lee, K.	[144]	-	1.5-3x area
36	mesh	fat-tree	Vassiliadis	[238]	4-16x	-
37			Kreutz	[120]	-	0.9x lat., 1.2x energy
38			Bartic	[17]	-	1x area/BW
39	hier.bus	ring	Lahiri	[134]	-	1x TP
40			Liljeberg	[152]	-	1x TP
41	crossbar	fat-tree	Bartic	[17]	-	1.3x area/BW
42	tree	fat-tree	Vassiliadis	[238]	0.5-1.8x	1x area

N =number of NoC terminals, used for asymptotic cost functions
BW =bandwidth
TP =throughput

Hierarchical bus has quite similar performance with ring in [134, 152] and mesh in [P8]. On the other hand, 2-D mesh is outperformed by hierarchical mesh [259], mesh with express links [181], custom [22], and fat-tree [238]. In few cases, a single bus is actually faster than hierarchical bus, $0.9x$ runtime [151], or mesh $0.1-0.5x$ [121] and $0.3-0.9x$ runtime [234].

The simulated speedups are clearly smaller than expected by analytical studies or by those concentrating purely on load-latency curves. However, the speedup depends on how runtime is measured; in clock cycles or in seconds. Using cycle counts favors buses and crossbars that have limited operating frequency. Fat-tree also exhibits few links near the root whose length and delay depend on the system size.

Multi-hop topologies offer substantially larger maximum bandwidth than shared bus, but also higher average latency with small load [1, 190]. Note that a network with increased parallelism does not translate directly to shorter runtime. Octagon network outperforms single bus clearly in terms of latency [107] when synthetic traffic is utilized. For video encoding [58] bus and octagon obtain identical runtime despite the fact that bus exhibits larger and more varying latency. Difference in encoding time only 0.12% although bus has up to $3x$ latency compared to octagon.

As another example, a $5x5$ crossbar enables 5 parallel transfers and gives 50% increase in application performance compared to single bus [196]. This study emphasizes an important point: *the increased parallelism in the network allows improvements via allocating more resources* (memory banks in this case). Parallel memories are unlikely to produce any speedup with the bus. Similarly, the crossbar is unlikely to improve the performance when utilized with one centralized memory. Hence, the resource allocation and optimal network are tightly coupled to each other.

10.2 Performance evaluation of reference NoCs

The reference NoCs are compared briefly using random traffic. The spatial distribution is either uniform or localized. The other parameters are the same in both tests. The network size is fixed to 16 terminals, flits are 4 bytes wide, average transfer length is 6 words, burstiness parameter $b = 0.6$, NoC and transaction generator use the same 100 MHz clock signal. Theoretical maximum offered load is hence $16\ terminals \cdot 4\ Bytes/terminal \cdot 100\ MHz = 6400\ MB/s$.

The buffer size is 2 flits in all NoCs except the packet-switched bus. It supports only store-and-forwards routing and hence the buffer size is 9 flits (3-flit header and 6-flit payload).

10.2.1 Spatially uniform random traffic

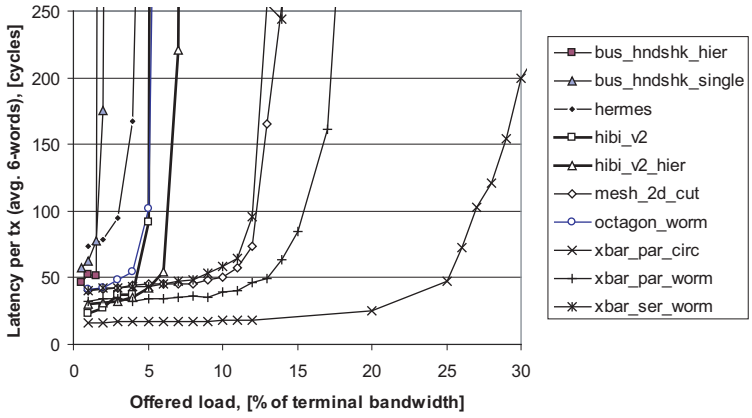
In the first test, the target is chosen randomly and all targets have a uniform probability $1/(N-1)$. The results are shown in Fig. 50 for 6-word transfers. The crossbars have clearly the highest performance. Mesh is quite close to packet-switched crossbar that has sequential arbiter. Parallel arbitration in crossbar naturally improves performance. Single HIBI, hierarchical HIBI, octagon, and Hermes obtain similar throughputs although Hermes has higher latency than the others do. Packet-switched buses are clearly the worst.

Hierarchical bus offers only minor benefit with spatially uniform traffic. Bridges actually degrade the performance with packet-switched bus - from 146 MB/s to 129 MB/s. The reason is that bridge does not get enough bandwidth from the target segment causing the initiator to stall. Initiator keeps retrying the transfer until it succeeds and every trial takes few bus cycles, hence wasting the bandwidth. The original packet-switched bus uses store-and-forward switching and sends only 1 packet in one turn. That would give only $84MB/s$ throughput. The Fig. 50 has results for packet-swithced bus with cut-through and sending multiple packets in a row is allowed.

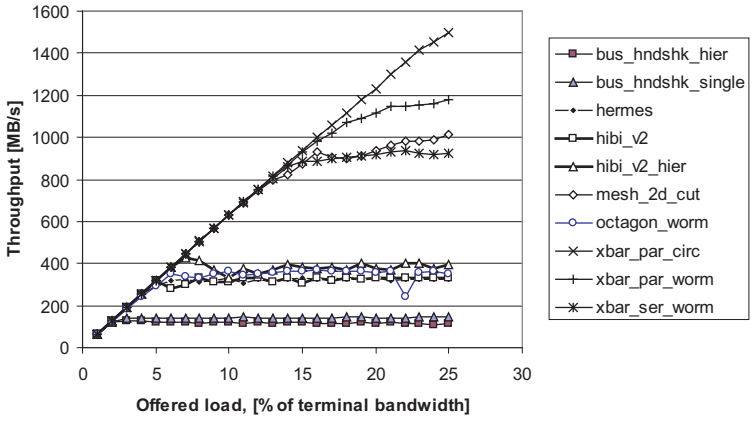
The phenomenon occurs also in HIBI but it is less severe. The single and hierarchical HIBI obtain similar bandwidth when round-robin arbitration is used. Fig. 50 shows the results with DAA and hierarchy brings clear benefit to HIBI. In addition, there is a peculiar peak in the throughput of hierarchical buses. The throughput drops a little from the maximum value, but remains above single bus, when the load increased beyond optimum. Such networks are called *unstable* [51].

10.2.2 Spatially localized random traffic

Localized traffic is modeled by defining clusters and probabilities so that a certain fraction p_{loc} of traffic goes to a certain cluster. Two ways were examined for defining the cluster cl of each source. The localization is network agnostic; it is calculated



(a) Latency



(b) Throughput.

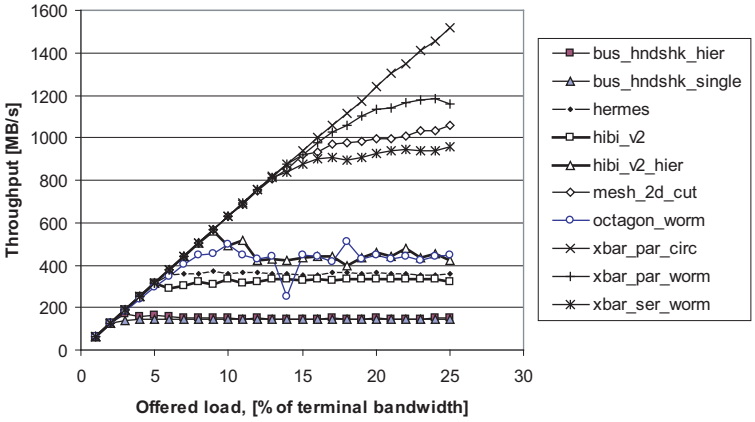
Fig. 50. *Latency and throughput as function of offered load. Average transfer length is 6 payload words. Spatially uniform traffic between 16 agents.*

from the terminal id $i \in [0, N_{ag}]$. Hence, routers in a cluster are necessarily neighbors in these topologies.

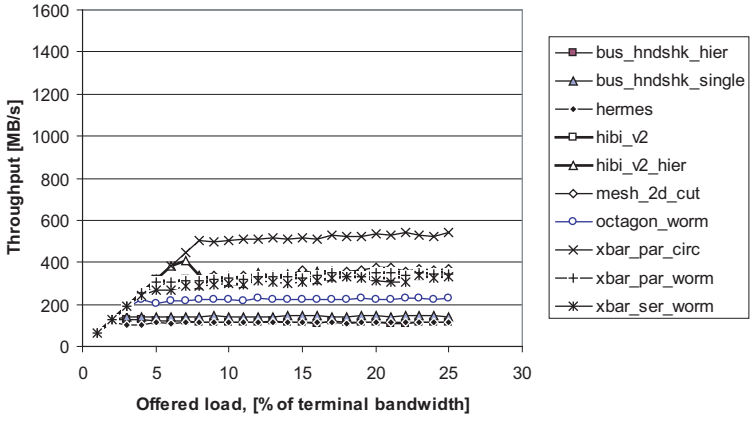
First, the cluster of sender i is defined as

$$cl = \lfloor \frac{i}{N_{agents\_per\_cluster}} \rfloor. \tag{19}$$

Another choice is to define as many clusters as their are agents. This creates overlap-



(a) Localized traffic. Half of the sent data localized to the nearest 3 targets.



(b) Hot spot traffic. Half of the all traffic goes to terminal 0.

Fig. 51. *Offered load vs. throughput with spatially localized traffic. There are 16 agents and transfer length is 6 words on average.*

ping "sliding" clusters so that

$$cl = i \tag{20}$$

and i is in the center of the cluster. To retain appropriate cluster size, the cluster bounds are clipped in cases where they would point outside the list of terminals.

Setting $N_{agents\_per\_cluster} = 4$ means that in the first case agents $0-3$ belong to cluster 0 agents $4-7$ to cluster 1 and so on. In the second case, all first three clusters have agents $0-3$, fourth one has $1-4$, fifth $2-5$ and so on. The results for locality

probability $p_{loc} = 50\%$ and sliding cluster scheme are shown in Fig. 51. The other cluster selection method exhibits quite similar behavior. However, that localization scheme causes less traffic across the bridge in hierarchical bus and hence about 10% higher throughput. In mesh, the benefit is about 3%.

All versions of single bus and crossbar have the same performance regardless of locality but the other topologies benefit from the increased locality.

10.2.3 Hot spot random traffic

An extreme case of localised traffic creates a hot spot; much traffic flows into (out of) single network node. This experiment sets agent 0 as hot spot target and the others send half of their traffic to it. Agent 0 sends uniformly to all others.

Fig. 51(b) shows the results. Four distinct categories can be identified. First, Hermes and packet-switched bus (both single and hierarchical) saturate at $115 - 150$ MB/s. Second, hierarchical octagon saturates at 220 MB/s. Third and fourth, at $300 - 350$ MB/s all the rest except circuit-switched crossbar which reaches over 500 MB/s.

Table 22 summarizes the throughputs from the three experiments. Localized traffic near the source always improves the throughput or has no effect (excluding the noise in measurements). A single hot spot node, on the other hand, severely limits the network performance. For example reference mesh, crossbars, and Hermes obtain much lower throughput; only about one third compared to uniform traffic. However, single bus topology is not affected and hierarchical bus only a little. Consequently, the differences between studied NoC become smaller. Similar observation has been made by Walter *et al.* [241]. Unfortunately, many contemporary MPSoCs have only one external memory interface [249] which is susceptible to create a hot spot.

10.2.4 Relative cost of reference NoCs

Throughput is only one criterion in NoC selection and it must be combined with others. As an example, Table 23 shows the relative throughputs, areas, and their combination. It also emphasizes the fact that relative cost depends on the traffic scenario.

Table 22. *The impact of spatial distribution of traffic to the NoC's throughput. Minimum and maximum values in each column are shown in* ***bold****.*

NoC	Max. throughput [MB/s]			Ratio		Diff. from uniform	
	uni	loc	hot	loc/uni	hot/uni	loc	hot
bus_hndshk_hier	**129**	165	129	1.28	**1.00**	++	0
bus_hndshk_single	146	**145**	146	**0.99**	**1.00**	0	0
hermes	329	368	**117**	1.12	0.36	+	---
hibi_v2	333	334	334	1.00	**1.00**	0	0
hibi_v2_hier	437	537	411	1.23	0.94	++	-
mesh_2d_cut	1 018	1 060	377	1.04	0.37	+	---
octagon_worm	364	511	231	**1.40**	0.63	+++	--
xbar_par_circ	**1 507**	**1 516**	**544**	1.01	0.36	0	---
xbar_par_worm	1 169	1 184	351	1.01	**0.30**	0	---
xbar_ser_worm	934	956	339	1.02	0.36	0	---
min	129	145	117	0.99	0.30	n/a	n/a
avg	636	678	298	1.11	0.63	n/a	n/a
max	1 507	1 516	544	1.40	1.00	n/a	n/a

Table 23. *Summary of throughputs and relative costs with three traffic classes.*

NoC	Relative max. throughput [no unit]			Relative	Relative throughput/area [no unit]		
	uni	loc	hot	area	uni	loc	hot
bus_hndshk_hier	**1.00**	1.14	1.10	1.50	**1.00**	1.05	1.37
bus_hndshk_single	1.13	**1.00**	1.25	1.39	1.22	**1.00**	1.67
hermes	2.55	2.53	**1.00**	1.86	2.06	1.89	**1.00**
hibi_v2	2.58	2.30	2.85	**1.00**	3.87	3.19	5.30
hibi_v2_hier	3.39	3.70	3.51	1.56	3.26	3.29	4.19
mesh_2d_cut	7.89	7.31	3.22	**2.11**	5.59	4.80	2.83
octagon_worm	2.82	3.52	1.97	1.83	2.32	2.68	2.01
xbar_par_circ	**11.68**	**10.46**	**4.65**	1.05	**16.66**	**13.80**	**8.22**
xbar_par_worm	9.06	8.17	3.00	1.48	9.18	7.65	3.76
xbar_ser_worm	7.24	6.59	2.90	1.48	7.33	6.18	3.63
min	1.00	1.00	1.00	1.00	1.00	1.00	1.00
avg	4.93	4.67	2.55	1.53	5.25	4.55	3.40
max	11.68	10.46	4.65	2.11	16.66	13.80	8.22

Crossbars offers clearly the highest throughput and (RTL synthesis) areas remain quite modest in 16-node configuration. However, the long wires and arbitration will limit the maximum frequency in larger configuration, which will also be very area consuming. Single bus saturates quickly but offers a small latency for very small traffic load (1-3% per agent). Hierarchical bus practically doubles the accepted traffic with uniform traffic but the real gain is achieved when transfers are localized. Reference octagon is quite close to hierarchical bus. Reference mesh exhibits the larger latency (partly due to network interface) than buses for small load but tolerates the largest traffic. However, Hermes - which is also a mesh - has rather poor performance.

10.3 Comparison of hierarchical bus and 2-D mesh

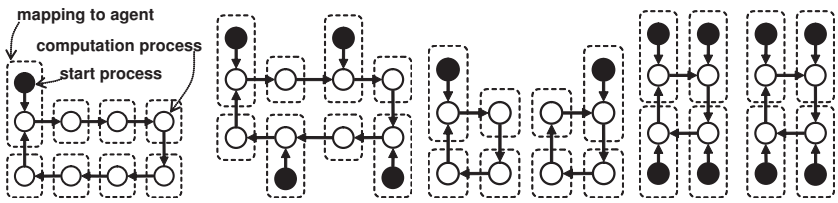
Another study was performed with considering the dependencies between transfers [P8], i.e. with Transaction Generator. The results were extended for this thesis with a smaller set of experiments with reference ring, octagon, and crossbar once they were implemented.

10.3.1 Test Setup

The comparison is done with synthetic benchmarks that are generated to represent characteristic application properties, such as sequential/parallel behavior, communication/computation intensiveness, and spatial traffic distribution. The test cases are executed with a VHDL version of Transaction Generator [P2]. Each task can be either waiting for data, reading data, processing, writing data, or finished. TG notably accelerates (up to 40x) the simulation compared to HW/SW co-simulation with multiple instruction set simulators. At the same time, the timing error is less than 10% with respect to real application.

Since the actual sizes of PEs and wire length are unknown, it is impossible to determine the maximum operating frequencies for networks accurately. For simplicity, all networks have the same operating frequency in the simulations, meaning that the runtimes are measured in clock cycles. Considering also the operating frequency affects the single bus the most, as its maximum wire length increases with the system size.

The communication networks were synthesized using a 180 *nm* CMOS technology. The packet payload size was set to eight 32-bit words. Packets have a three-word header and eight-word payload data and hence the required buffer size is $(3+8)\cdot 32$ bits in all the internal network buffers. The header includes target's NoC address, data amount, and address fields. In contrast, HIBI requires a one-word header for address and does not utilize packets. In packet-switched bus, the ownership is passed to the next wrapper after each transmitted packet. In HIBI, the ownership changes when a pre-defined number of words have been transferred (40 words in this case) or when the sender runs out of data.



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

Fig. 52. *Test case types used in comparison, $N = 8$.*

Table 24. *The number of trigger events in test cases as they are scaled with the number of agents (N).*

Case	Number of start processes (S)				
	4	16	36	64	N
1	1	1	1	1	1
2	2	8	18	32	N/2
3	1	4	9	16	N/4
4	4	16	36	64	N
5	8	29	64	113	(7N/4)+1

10.3.2 Test Cases

Fig 52 shows the used task graphs for 8 agents. The white nodes depict computation tasks that can have an arbitrary processing time of P clock cycles. The edges represent data transmissions with length of D words. Computation at a node cannot start until at least one of the arriving transfers has completed (transfer dependence). The trigger events, marked with black nodes, have $P = 1$ and $D = 1$ and are executed only once to trigger computation tasks. By changing P and D, the application model in TG can be made more computation or communication intensive. Both P and D can be varied randomly within a user-defined range. All these benchmarks are scaled with N so that there is one computation process and variable number of trigger events per agent. The dashed lines describe a simple 1-to-1 mapping of the task graph onto eight agents. However, the task graphs are totally independent of the hardware architecture and other mappings can be easily explored.

The first benchmark, case 1 (Fig 52(a)), resembles a sequential data flow application having only one trigger event. Case 2 (Fig 52(b)) is partly sequential and partly parallel in nature having $N/2$ trigger events. Case 3 (Fig 52(c)) presents an application where the transfers are sequential as in case 1 but within groups of four tasks so the

communication is localized. Case 4 (Fig 52(d)) has tasks in a group of four transmitting data in parallel to each other. Cases 2 and 4 can also be thought as pipelined versions of cases 1 and 3, respectively. These cases do not include divergence (1 task sends to multiple tasks) or convergence (vice versa). Hence topologies that handle locality well will have an advantage.

In addition, case 5 combines the cases 1-4 into one simulation to represent heterogeneous behavior. In case 5, cases 1-4 are run together so that each agent executes one computation task from each case. For example, all cases have start and computation tasks grouped together in the top left corner (cf. Fig 52(a)-52(d)) and they all are mapped to first agent in case 5. The next computation task to the right is mapped to agent 2 and so on. The mutual order of the subcases is not specified. The number of trigger events is shown in Table 24.

The mapping of PEs to NoC affects performance clearly in multi-hop topologies and practically negligibly in a single shared bus. The PEs are numbered in the same order as data flows in Fig. 52(a). In the hierarchical bus, a group of 4 consecutive PEs always forms one segment hence exploiting the locality of the task graphs. In the mesh, the PEs are placed according to data flow row-by-row always starting from the top-left corner. The traffic is well localized but not optimally. The last connection in cases 1 and 2 (one that closes the loop) travel all the way though the hierarchical bus and mesh from corner to the opposite one.

10.3.3 Static Runtime Analysis

The total runtime of an application is a sum of computation time and communication time. It can be estimated in a heuristic fashion for these graphs as

$$t_{\text{tot}} = \frac{\sum P_i}{min(N,S)} + \frac{\sum (D_i \cdot k)}{min(N,L,S)} \quad , \qquad \text{[clock cycles]} \tag{21}$$

where

$$k = \frac{payload + header + arbitration}{payload} \tag{22}$$

is an implementation specific factor that is explained later. The dividers in (21) define the achievable parallelism. The parallelism of an application is defined here as the maximum number of parallel transfers and active computation tasks, and it equals the

number of trigger events (S) in these cases. If there are less agents (N) or communication links (L) than trigger events (S), the maximum parallelism of the application cannot be achieved. Similarly, the maximum number of initiated transfers per clock cycle cannot exceed the number of agents in any network. For example, case 1 is a sequential application having only one trigger event and utilizes only one processor or communication link at a time. Adding more communication links does not directly speed up the operation at all. However, the arbitration may get simpler (less requestors per arbiter, smaller k) and can thereby reduce the communication time.

The factor k, caused by arbitration and the overhead from packet headers, is calculated with equation (22). It is defined as the number of clock cycles needed to transfer one packet divided by the amount of transferred payload data words. Ideally, k would be one. Packet and header sizes are expressed as multiples of the word size, because one word can be transferred in one clock cycle. With distributed arbitration, each agent in a bus segment has to wait whole round-robin iteration between consecutive packets. However, there can be S agents active in each round, which reduces the overall arbitration delay. In this study, the routing algorithm of a mesh simply checks one input each clock cycle for new transfers, thus, on average $5/2$ clock cycles are needed for routing. The term $arbitration$ is assumed to be $(N-1)/S$ for bus, 6 for hierarchical bus (4 agents and 2 bridges per segment), and 2.5 for mesh. Estimate for case 5 is a sum of individual estimates for cases 1-4. Still, (21) does not take transfer dependencies into account and assumes a uniform, localized mapping.

10.3.4 Synthesis Results

The resulting network logic areas in kilogates are depicted in Table 25. The wiring area is not taken into account. The difference is mainly due to buffers. HIBI has 5-word buffers but more complex control logic, and, hence its area is only about 13% smaller than the area of simple, packet-switched bus (abbreviated as *pkt*). In mesh, the control logic for cut-through switching is slightly more complex than in store-and-forward. Both types of mesh omit the unnecessary buffers at network boundaries, hence saving $9\% - 49\%$ of area.

The relative sizes are shown for networks only and for the whole system assuming a 50-kilogate area for each processing element (PE). The overhead from network considering the total system size naturally decreases as the size of processing elements

Table 25. *Absolute logic area of the networks (in kilogates) and their relative areas. Area of PE is assumed to be 50 kilogates.*

Network	Absolute logic area [kilogates]				Relative area (on average)	
	4	16	36	64	network	network+PEs
Single bus (pkt)	28.7	114.8	258.4	459.1	1.14	1.02
Single bus (HIBI)	25.2	100.4	225.8	401.4	1.00	1.00
Hier. bus (pkt)	28.7	158.1	373.5	675.2	1.51	1.06
Hier. bus (HIBI)	25.2	138.7	327.7	592.2	1.33	1.04
Mesh (st&fwd)	59.3	295.4	708.6	1 299.0	2.92	1.21
Mesh (cut)	59.8	297.7	714.1	1 309.1	2.94	1.22

increases. The area results suggest that it is possible to use wider bus, e.g. 64-bit data instead of 32-bit, and still have smaller area than with mesh. The impact of this will be described later.

10.3.5 Simulation Results

In the following simulations, the processing time P varies from 16 to 1024 cycles and the transfer length D varies from 16 to 1024 words. The extreme case $P = 16, D = 1024$ sets tight requirements for the communication network. This kind of communication intensive transfer patterns can be found, for example, in packet processing inside an Internet router. In addition, such case appears when the processing elements operate with higher frequency than the network (processing time P refers to clock cycles of the network in that case). The measured average runtimes for case $P = 16, D = 1024$ are listed in Table 26. Times are given in thousands of cycles (kilocycles). The shortest runtimes are shown in bold. Test cases were executed several times, 30 times on average, for statistical reliability. All networks have the same data width. The run-time of case 5 is defined by the slowest individual application, case 1, when applications 1-4 are run together.

In case 1, the poor result of the packet-switched single bus is due to inefficiency of the utilized distributed round-robin arbitration during low levels of contention. HIBI, on the other hand, can transfer more data in one turn and, hence, arbitration and address transfers have less impact. Therefore, HIBI achieves the shortest execution time.

In test case 2, the differences between the networks are more apparent. The transfer times of the single bus grows very fast with system (and application) size. On the

Table 26. *The runtimes in kilocycles for test cases with $P = 16$ cycles, $D = 1024$ words.*

	4 agents					16 agents				
Network \ case	1	2	3	4	5	1	2	3	4	5
Single bus (pkt)	12.9	7.7	12.9	7.2	43.0	60.0	30.7	34.8	28.7	585.2
Single bus (HIBI)	**6.5**	**4.4**	**6.5**	5.0	24.9	23.6	18.5	18.9	19.1	257.7
Hier. bus (pkt)	same as single bus(pkt)					55.0	11.3	14.5	7.3	110.5
Hier. bus (HIBI)	same as single bus(HIBI)					**20.4**	8.9	**4.9**	5.3	97.1
Mesh (st&fwd)	15.6	7.8	15.6	3.9	43.0	62.4	7.8	15.6	3.9	115.9
Mesh (cut)	10.4	5.2	10.4	**2.7**	**22.9**	41.7	**5.2**	10.4	**2.7**	**74.4**

	36 agents					64 agents				
Network \ case	1	2	3	4	5	1	2	3	4	5
Single bus (pkt)	227.1	69.1	78.3	64.5	2 935.7	630.8	122.9	139.3	114.7	11 661.4
Single bus (HIBI)	71.9	31.9	42.7	38.6	1 242.2	176.2	67.1	74.9	91.4	3 607.7
Hier. bus (pkt)	120.0	11.5	14.5	7.3	266.4	211.1	11.5	14.5	7.3	494.3
Hier. bus (HIBI)	**46.6**	10.5	**4.9**	5.3	308.8	**82.6**	12.9	**4.9**	5.3	332.4
Mesh (st&fwd)	140.7	7.8	15.6	4.7	308.6	249.9	7.8	15.6	3.9	267.4
Mesh (cut)	93.8	**5.2**	10.4	**4.3**	**209.8**	167.0	**5.2**	10.4	**2.7**	**196.1**

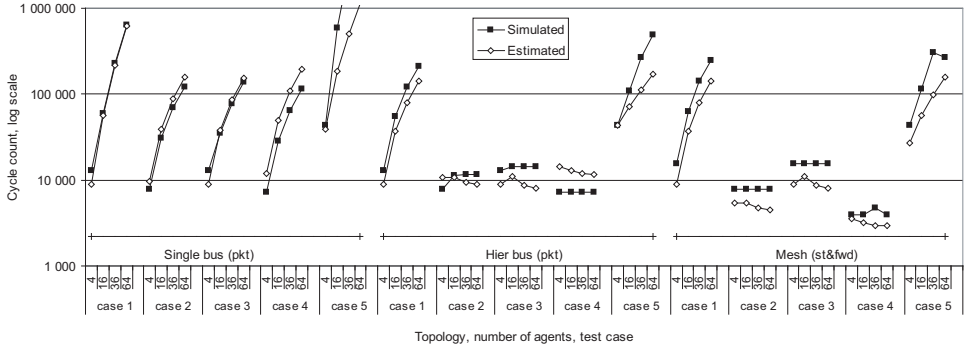


Fig. 53. *Estimated and simulated cycle counts for three networks.*

other hand, the results for hierarchical bus and mesh are quite close to each other and follow the results predicted by (21), which means that both networks are able to systematically exploit the inherent parallelism of the application. The same happens with test cases 3, 4, and 5. For these test cases, a hierarchical, packet-switched bus offers a comparable performance as 2-dimensional mesh with store-and-forward switching. However, HIBI outperforms the packet-switched bus and, similarly, cut-through mesh outperforms store-and-forward switching. Thus, in addition to topology, the switching policy has also a notable impact on the performance.

Fig 53 shows both the simulated cycle counts as well as those estimated using equation (21). Three networks out of six are shown for clarity. The shapes of the esti-

mated and the measured performance curves match rather well, which implies that equation (21) predicts the ratio between runtimes in many cases. However, in some cases estimates are clearly inaccurate, e.g. case 2 with mesh. This is mainly due to the implementation of TG, network contention, and inefficiencies in implemented network protocols and packetization logic that were not included in equations. Other than 1-to-1 mappings cause more contention and larger average hop counts and hence bigger estimation errors are probable. Therefore, fast simulation-based approaches for benchmarking are needed.

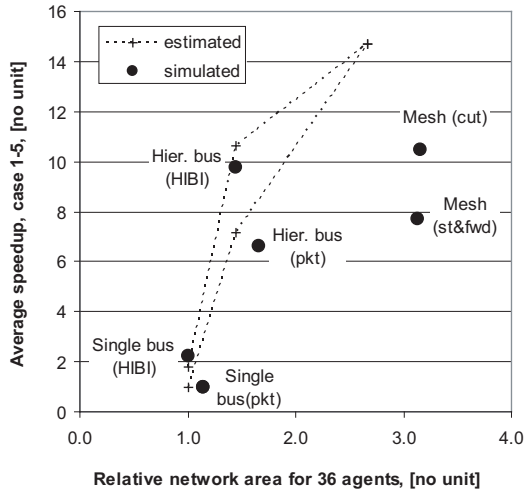
10.3.6 Relative Cost of the Networks

Fig 54(a) shows the area overhead versus average speedup curves for 36-agent networks. The speedup is estimated according to (21) and area according to the number of buffers. Results are scaled so that both the smallest area and speedup are equal to 1.0. The best approaches are in the top left corner. Hierarchical bus offers good speedup over single bus with moderate area overhead, especially HIBI that scales better than the packet-bus. Mesh offers biggest average speedup with largest area. The choice between the mesh and the hierarchical bus is, therefore, a trade-off between area cost and performance. For comparison, let us define the architectural performance as the inverse of the costs:

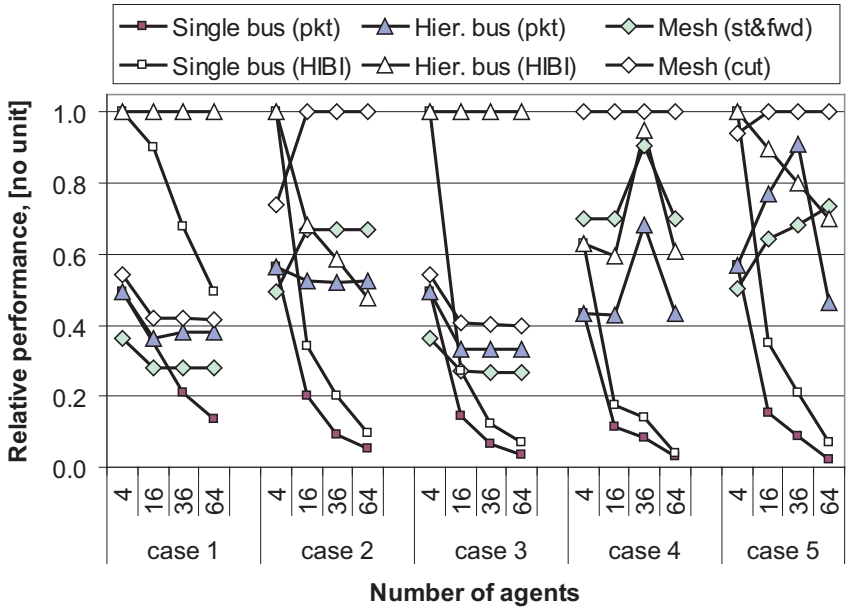
$$Performance = cost^{-1} = (t_{\mathrm{tot}}{}^{w_t} \cdot A_{tot})^{-1} \quad . \tag{23}$$

The cost is defined as a product of runtime t_{tot} and total system area A_{tot}. System area includes both the network and all the PEs (50 kilogates each, in this case). The weighting exponent w_t can be utilized to make the runtime less ($w_t < 1$) or more important ($w_t > 1$) than the area. In these cases, smaller weights favor the hierarchical bus and large ones favor the mesh. Fig 54(b) shows the measured performance of all networks so that the best case is scaled to 1.0 and execution and area have equal weights ($w_t = 1$). Hierarchical HIBI and cut-through mesh offer the best trade-off. A single bus is applicable only in sequential test case 1.

Fig. 55 shows cases where $P = 64$ cycles and D varies between 32 and 128 words. Only the fastest networks, i.e. HIBI and cut-through mesh, are shown for clarity. The differences between networks become more apparent as the data amount D increases. Likewise, the network performance depends strongly on the application and system



(a) Average speedup versus area for 36 agents.



(b) Relative overall performance (bigger the better)

Fig. 54. *Speedup and relative performance,* $P = 16, D = 1024$.

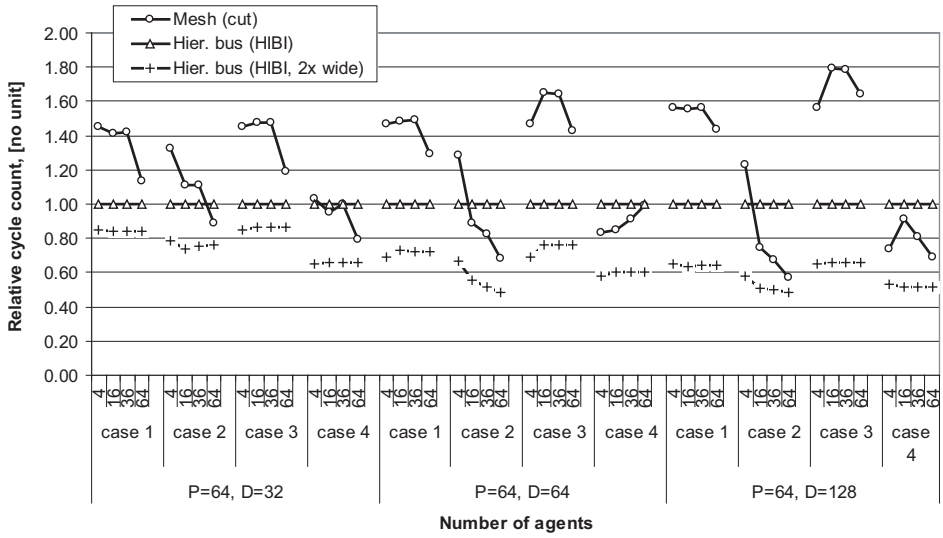


Fig. 55. *The relative runtimes with 4 test cases. Processing time P=64 cycles and data amount D=32, 64 or 128 words*

size. For example, HIBI is superior in cases 1 and 3 but mesh is better with cases 2 and 4, especially, with large number of agents and large data amount. In addition, Fig. 55 shows the runtime of hierarchical HIBI when its data width is doubled. It is measured by halving the amount of transferred data D in the simulation. As the data width doubles, the area of HIBI increases 86% but it is still about 15% smaller than mesh. At the same time, the runtime decreases in all test cases. The reduction varies from 20% up to 50%, being smallest in sequential cases with small D. In addition, hierarchical HIBI with wide links achieves the lowest cycle count in these cases. The average (relative) cycle counts are 1.0 (hier. HIBI), 1.18 (mesh), and 0.67 (double wide hier. HIBI).

10.3.7 Results with other networks

Similar test were run also for other networks. The results are shown in Fig. 56 and Fig. 57 assuming processing time $P = 16$ cycles and data amount $D = 16$ words. The results include also mesh as well as single and hierarchical HIBI v.2 and pkt. bus as in previous studies. Due to vast number of combinations, other values of D and P are omitted here. Moreover, only few choices of network parameters are selected, for example arbitration style (serial or parallel) and switching (store-and-forward, cut-

through, or wormhole). Other parameters, such as buffers sizes, packet lengths, and frequencies are fixed.

A couple of straightforward conclusions can be drawn from the figures. Packet-switching is unnatural choice for single-hop topologies, such as bus or crossbar. It increases the runtime although it might provide benefits regarding error detection (which is not covered in this Thesis).

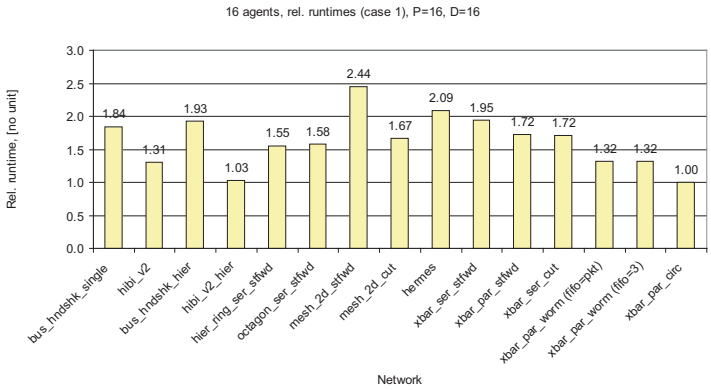
The differences between multi-hop topologies are rather small with these test cases. Crossbar is naturally the fastest network when cycle counts are measured but large area restricts it usage, as noted earlier. Reference mesh clearly outperforms Hermes mesh. This justifies the previous comparison as the reference mesh can be considered representative implementation for that topology.

However, differences due to network settings other than topology are notable, for example up to $2x$ with crossbar. With packet-switched network cut-through (or wormhole) should be chosen instead of store-and-forward switching. Following that, hierarchical ring and octagon should be re-implemented with cut-through and parallel output port arbitration as well.

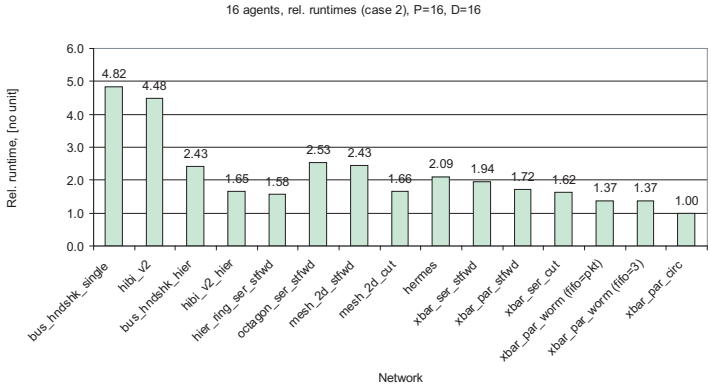
10.3.8 Impact of task mapping

So far the mapping has been very straightforward: neighboring tasks are mapped to neighboring PEs. This results in very localized traffic patterns, which benefits especially the hierarchical bus. However, when tasks are mapped further apart, traffic is less localized and performance will be different.

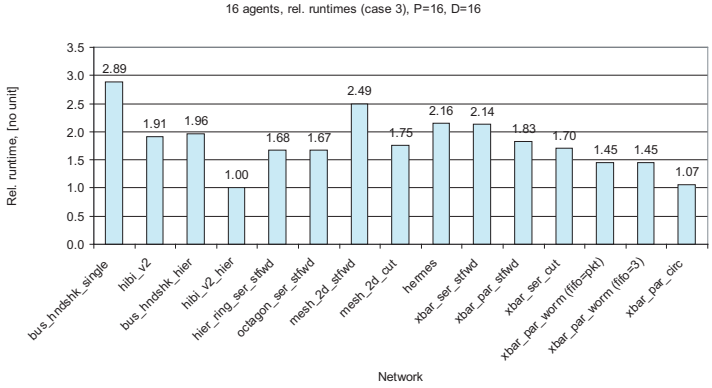
Fig. 58 shows the results for three cases. The mapping is varied with so called stride parameter so that consecutive tasks i and $i+1$ are mapped to PEs p and $p + stride$. The original mapping has $stride = 1$. We notice that single bus and mesh are not affected much when stride varies. On the other hand, the runtime in hierarchical bus is very sensitive to mapping. For example, the differences between minimum and maximum in case 3 are $+16.8\%$ for single HIBI, $+95.3\%$ for hierarchical bus, and $+8.6\%$ for mesh.



(a) Case 1

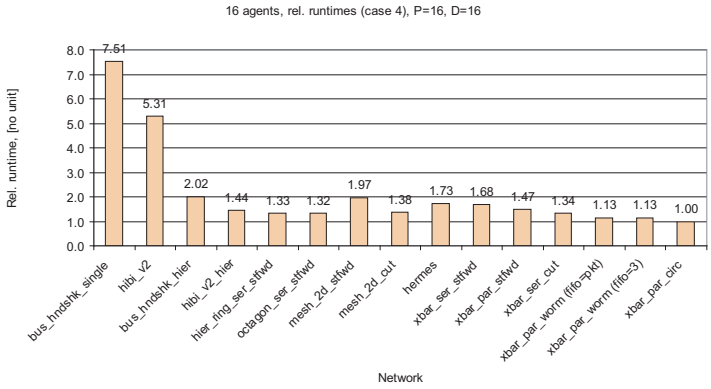


(b) Case 2

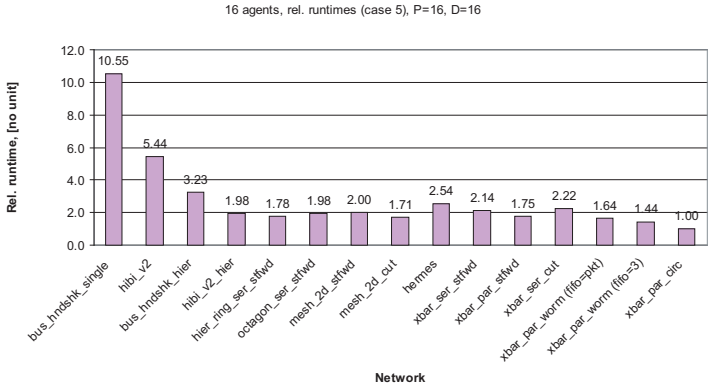


(c) Case 3

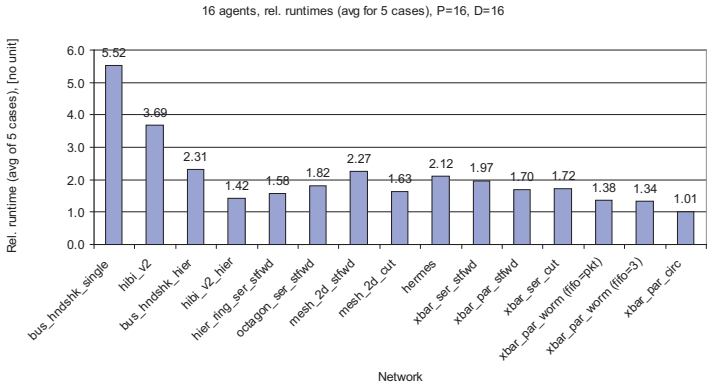
Fig. 56. *The relative runtimes with test cases 1-3. Processing time P=16 cycles and data amount D=16 words*



(a) Case 4

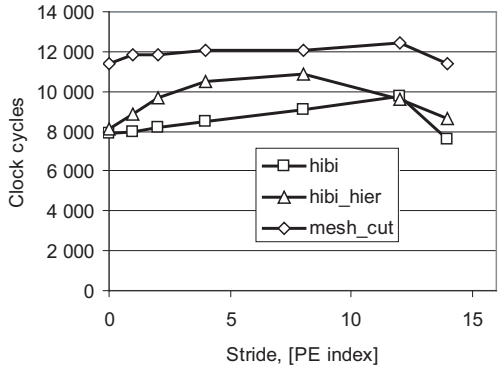


(b) Case 5

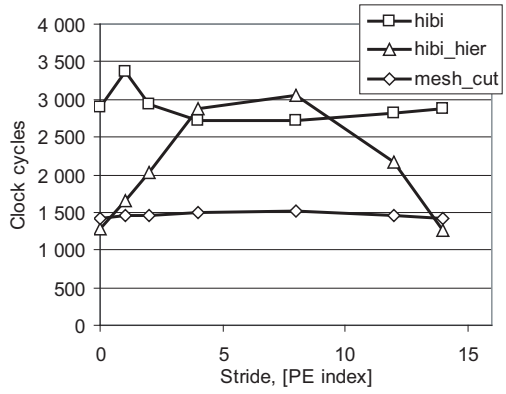


(c) Average of cases 1-5

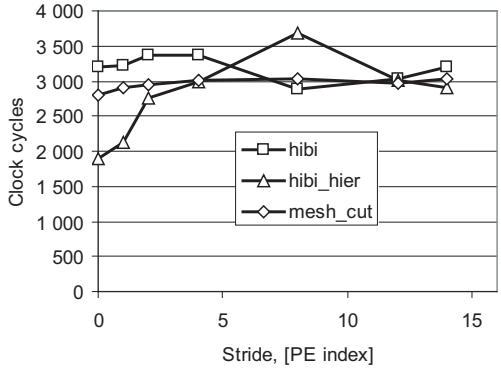
Fig. 57. *The relative runtimes with test cases 4 and 5 and average of all five. Processing time P=16 cycles and data amount D=16 words*



(a) Case 1



(b) Case 2



(c) Cases 3

Fig. 58. *The runtimes for test cases 1, 2 and 3 with varying stride value in mapping of tasks. Processing time P=16 cycles and data amount D=16 words*

10.3.9 Discussion

Application dependence is a factor that cannot be disregarded in communication-based system design. Results with these test cases show that theoretical performance derived from the aggregate bandwidth of the network does not reflect the application runtime linearly. The presented formal equation provides reasonably accurate estimate in some cases but not in general. However, more advanced estimates considering router latency and congestion would require rather complex equations. Therefore, fast, cycle-accurate simulation is preferred.

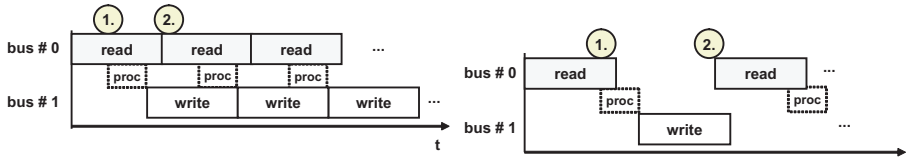
Many contemporary analyzes depict bus as poorly fitting to large systems because only the single bus is used as a reference. The presented hierarchical bus scales quite easily to large systems and provides a good area-performance trade-off while retaining many of the advantageous features of simpler bus arrangements. The hierarchical bus exhibits good runtime results with relatively small implementation area. The analyzed 2-dimensional mesh network provides higher performance but needs larger area. The architectural performance, defined as a product of area and runtime, favors the use of hierarchical bus topology.

10.4 Performance evaluation of HIBI with synthetic traffic

For comparison, HIBI v.2 protocol was tested with a test case similar to the one described by Saastamoinen *et al.* [96] (pp. 193-213). Furthermore, several synthetic test applications were developed to test and measure different NoCs and also Koski design flow as well. The test cases are parameterizable allowing the designer to easily create cases with varying characteristics (e.g. communication interval, transfer size and burstiness, and load distribution).

10.4.1 Simple image processing

The application reads 10x10 byte pictures from source memory, processes the data, and writes 10x10 byte result pictures to target memory. Original test case carries out handshaking with messages for every transfer. It is not needed in HIBI because no data is dropped, however, it was included to allow fair comparison. The test case was modeled with Transaction Generator implemented in VHDL. The test case model



(a) Optimal pipeline (b) Inefficient pipeline

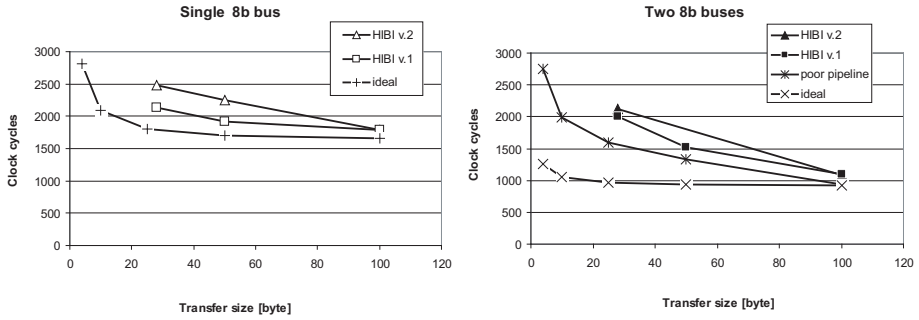
Fig. 59. *Examples of the test case pipelining*

assumes minimum latency of 10 cycles for processing and 5 cycles for memory reads. The actual function of image processing algorithm is irrelevant in this context; only the execution times and communication behavior are needed.

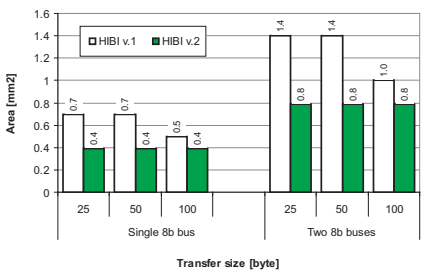
The architecture includes eight processing units, single-ported source and target memories, and either single- or dual-bus 8-bit HIBI v.1 or v.2. Round-robin was used as the arbitration method. Single- and dual-bus networks are similar to *SegC* and $SegC + D$ in Fig 39, respectively.

Doubling the bus width or adding another bus naturally improves the performance but the improvement is less than two-fold. The reason is that requests and acknowledgements cannot utilize the increased bandwidth. Furthermore, the task pipelining may be inefficient. Two cases of pipelining are illustrated in Fig 59. Processing (dashed box) is shown for the first agent even if it does not actually use either of the buses. In an ideal case (a), first agent gets the whole input data when requesting it (shown with circle 1) and it can start processing immediately. Likewise, it can write all the results without interruption. The first agent is ready to start reading another picture at the time instant denoted with circle 2. Simple application model does not perform any handshaking between processing units and, therefore, they all start requesting source data at the same time. The memory sends slices of the picture to all requesting processors and the processing cannot start until the whole picture has been received, and hence the completion of the tasks is delayed. The same situation happens when results are written. This is shown in Fig 59(b) still assuming ideal bus. Consequently, one of the buses is idle for a long time and execution time does not improve much from the single bus topology. This pipeline impact is negligible in the single bus topology as the processing latency is so small.

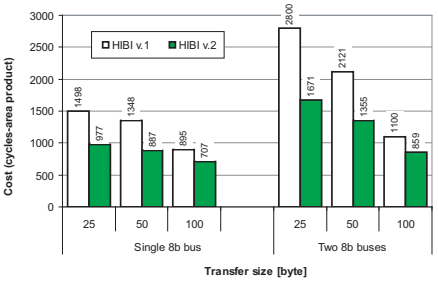
Fig 60(a) shows the simulated run time for processing one picture in each processor. Large transfer size is clearly beneficial and HIBI performance approaches the theo-



(a) Execution time



(b) Network logic area



(c) Cycle-area product

Fig. 60. *Results of the case study: execution time in cycles, logic areas, and cycle-area product*

retical minimum. Both dual-bus HIBI networks have 15-35% shorter execution times for subsequent pictures when the time for filling the pipeline has only a minor effect. Fig 60(b) illustrates the estimated area for different networks. The overall cost, defined here as the product of execution time and area, is shown in 60(c). This cost factor suggests that a single HIBI v.2 bus with $max\_send = 100$ offers the best trade-off between execution time and area among the studied networks. This simple example shows the importance of careful application design and pipelining in addition to network optimization. For this reason, complex networks seldom achieve performance that is close to their ideal case. Therefore, theoretical maximum bandwidth, which is sometimes reported for NoCs, cannot be used for accurate comparison of networks.

HIBI v.2 has 21%- 43% smaller area than HIBI v.1 depending on transfer size. This is due to multiplexed address/data lines and handshaking which both reduce buffering. As noted earlier, optimal buffer size in HIBI v.2 is determined by the latency of

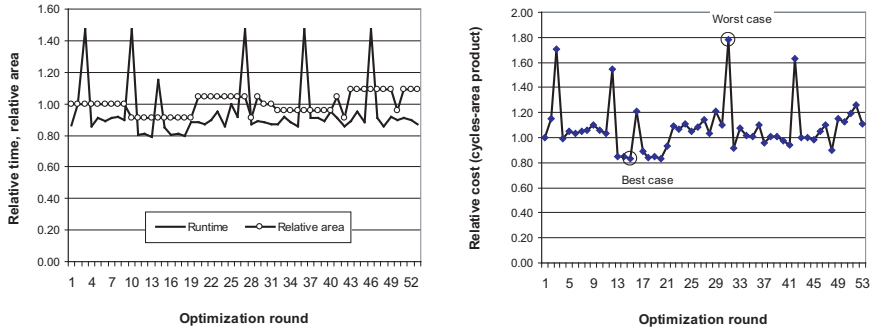
a functional unit instead of the packet size. The biggest impact is seen on destination memory wrapper: HIBI v.1 requires 65-word receiver buffer to avoid overflow whereas HIBI v.2 used 7-word buffer. Even smaller buffers could be used in HIBI v.2 with a small penalty of increased number of retransfers. In contrast, storing all packets inside the network logic, like the store-and-forward method, would result in area consumption that is linearly dependent on the packet size. As a result, the area of HIBI would be approximately 1.7x - 17x larger. Comparison with results in [96] (about 2500 cycles, 4.4 mm^2 with 28B/tx) shows that even a single HIBI bus provides better performance than a Proteo ring network with considerably smaller area cost for this test case. Adding a second bus to HIBI network improves performance nearly 40%.

10.4.2 The effect of network parameters

HIBI is highly parameterizable interconnection and this Section discusses briefly the impact of various settings. As an example, evolution of optimization with 53 iteration rounds is shown in Fig 61 for test case *hier_seq*2. Each iteration round modifies HIBI parameters, for example, FIFO sizes, priorities, and transfer sizes. In this case, the best case offers 15% saving in clock cycles and 9% saving in area which give 17% reduction in relative cost metric compared to initial configuration. The worst case resulted over 2 times bigger cost than the best case. Exploring also the mapping of tasks onto processing elements has even larger impact on system performance [182, 183] and that is performed prior to network optimization. Automated tool provides these results within minutes or hours depending on the size of the application and architectural space. A system having several applications can be optimized for different scenarios depending on which applications are currently active. Optimal HIBI parameters can then be reconfigured at runtime.

In addition, monitoring capabilities of HIBI allow the designer to identify the bottlenecks by examining the network statistics. For example, the achieved network throughput and the fullness of all FIFOs and number of retransfer cycles are tabulated for every wrapper automatically by the HIBI models. In addition, the Koski tool can report the latencies of transfers, number of calculation and communication cycles for each agent.

The effects of different HIBI parameters were evaluated with the aid of Koski. As



(a) Relative time and area (b) Relative cost

Fig. 61. *Results of the optimization process*

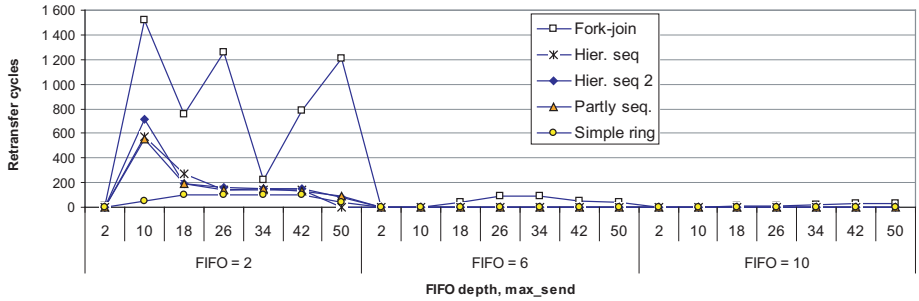


Fig. 62. *Number of retransfers as a function of FIFO size and max_send parameter*

noted earlier, retransfer occurs when the receiver cannot accept the data immediately. Fig 62 shows the total number of required retransfers in a system as a function of FIFO size and *max_send* parameter for several test cases. It shows that other sizes than minimum for FIFOs, i.e. 3 words or more, eliminate the need for retransmission almost completely. In these cases, retransmissions account for 0.3% of the traffic on average and 3.5% at maximum. With minimal FIFO size, a large number of retransfers happens in test case $fork\_join$ when all agents send data to a single agent. This is a good example of how execution statistics are used to identify the system bottleneck. If the optimizing the latency at the target IP proves to be difficult, its receive buffers may be simply increased. Consequently, system performance is improved notably as retransfers are removed. Note that the number of retransfers varies irregularly with transfer size. Such behavior further emphasizes the importance of

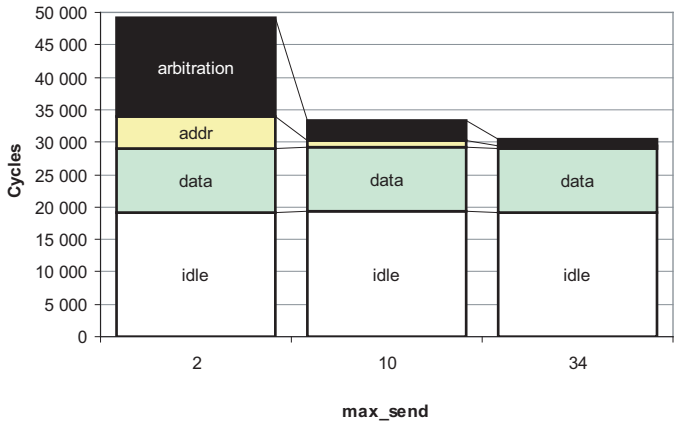


Fig. 63. *Increasing transfer length reduces arbitration overhead*

automated exploration and parameter optimization.

With small values of *max_send*, the distributed round-robin performs poorly when the bus utilization is low. As an example, Fig 63 shows the breakdown of clock cycles for test case *hier_seq*2 when all FIFOs are 6 words deep. Many cycles are needed for arbitration and transmitting addresses when transfers are split into several small transfers. However, already a small increase in *max_send* decreases these overheads without affecting the area of the network. The possible drawback of large transfer size is the increased latency of waiting for the bus access. The experiment shows how the multiplexed address and data lines cause only minor performance degradation (address cycles) with respect to separate lines. The bars labeled as *idle* depend heavily on application as they denote the cycles when the agents are only processing (or waiting) data without any data transmissions.

The latency of HIBI was measured in two ways, both without TDMA. First, four agents send a total of 1000 39-word messages to random targets as in [169]. The latency is defined as the difference of times when the data is written to Tx FIFO and when it is read from Rx FIFO. No time slots were used and all data was sent with the same priority. Simulated latency histogram of one source-target pair is shown in Fig 64. Average latency was 158 cycles which means 4 cycles per data word. Latency varies between 1.3-10 cycles per data word depending on the contention on the bus and on the target.

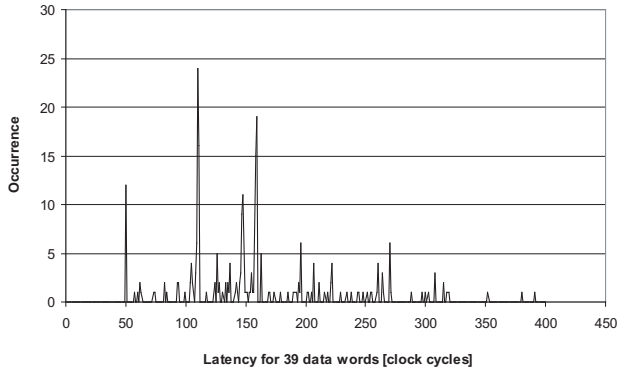
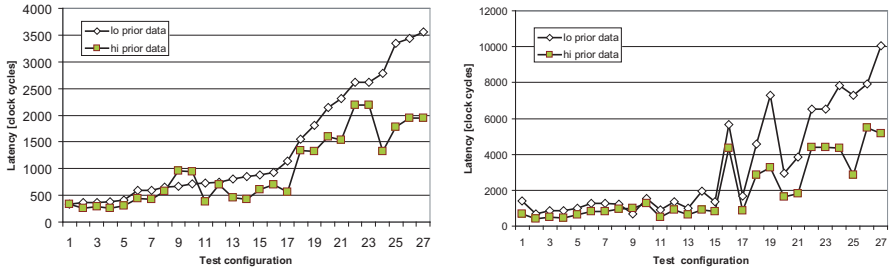


Fig. 64. *Latency variation for system in which four agents send a total of* 1000 39*-word messages with uniform distribution. Both bus width and data word width are* 32 *bits*



(a) Average latency (b) Maximum latency

Fig. 65. *Effect of relative priority on latency in a test case in which two agents write to the same target*

Second, the benefit of the relative priority is highlighted with a test case in which two agents write to the same target. The first agent uses both high and low priority data whereas the other uses only low priority. The average and maximum latencies for different priorities are shown in Fig 65. Total of 27 different test configurations were tried out by varying transfer size (in words), transfer interval (in clock cycles), and *max_send* parameters. All of these parameters were varied from 2 to 100. Results are sorted in ascending order of average latency of low priority transfers to emphasize the differences. Low contention on the bus and on the target (test configurations 1-12) result in low latency and the relative priority does not offer much benefit. Therefore, the latency is not affected notably. However, for the configurations resulting in higher latencies (configurations 12-27), the high priority offers great benefit. Regarding all

configurations, high relative priority obtains 31% lower average latency and 41% lower maximum latency than the low priority.

10.5 H.263 Video encoder simulation

This Section presents design space exploration of a data-parallel video encoder running on Multiprocessor System-on-Chip (MPSoC) [P7]. The impact of communication on the scalability is analyzed. The exploration is carried out with the abstract models for application and architecture by using SystemC Transaction Generator. The application model is based on profiled execution times and data sizes from real application code.

Many studies consider how changes in application affect the speedup assuming fixed computation platform. This study analyzes how the changes in communication mechanisms affect the speedup assuming fixed application. Efficient communication between processing elements and load balancing are crucial to obtain good scalability. The communication requirements are proportional to the number of processing elements and, in general, inversely proportional to their complexity.

In this Chapter, a design space exploration for data-parallel video encoder MPSoC [105] is presented. The processing elements (PEs) are interconnected with a single shared bus to analyze its feasibility. Various architectures are explored by changing the number of PEs, their frequency, and bus frequency. Furthermore, the upper bound for performance resulting from ideal network and the impact of direct memory access (DMA) are determined. Data-parallel video encoder application is modeled with Transaction Generator [P2].

10.5.1 Data-parallel video encoder test case

A detailed presentation of the video encoder can be found in [105]. The video encoding processing is done in master-slave configuration, in which all processing elements have local data and instruction memories. All the slaves execute the same encoding functions but operate on different parts of the image. Each PE processes one slice consisting of one or more horizontal macroblock rows as shown in Fig. 66(a). At first, the master PE reads one frame of raw video from I/O unit. Second, it divides

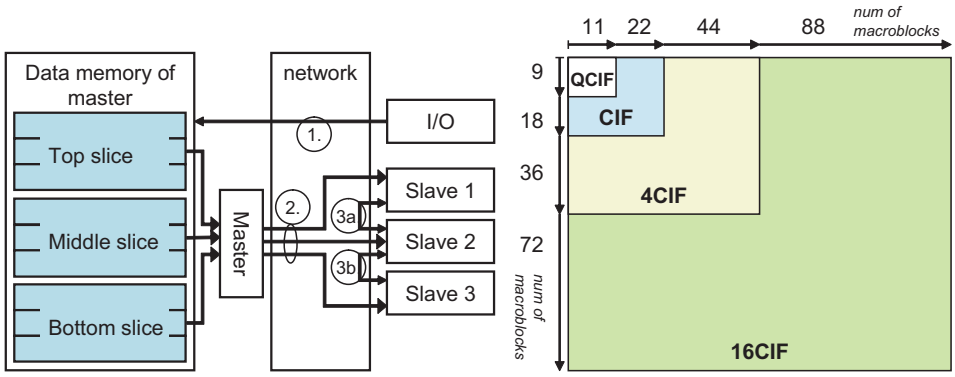
the frame into slices and sends them to the slaves. Hence, a raw frame is transferred twice across the communication network at the beginning of each frame encoding. Third, the slave PEs must exchange rows with their neighbors (if any) to carry out motion estimation. Finally, the master merges their results (opposite to transfer 2) into a final bit-stream and sends that to I/O (opposite to transfer 1). Data amount depends on the number of PEs in phase 3 only.

Only one macroblock row per neighbor is transferred when motion vector length is restricted to 16 pixels or less. The slaves processing the top-most or the lower-most slice have only one neighbor whereas others have two. Hence, the amount of exchanged data equals $2 * (n_{slaves} - 1)$ macroblock rows, n_{slaves} denoting the number of slaves. A macroblock (*mb*) corresponds to 16x16 pixel area. With 8 bits per pixel, a macroblock size is 384 bytes (256 bytes of luminance and 128 bytes of chrominance data). The different video frames sizes are shown in Fig. 66(b).

Such coarse-grained parallelization approach results in unequal load balance when slices are not equally sized. For example, when 9 rows of QCIF (Quarter Common Intermediate Format) frame are processed with 4 slaves, one slave must process 3 rows whereas others process only 2. Larger frame sizes allow a better load balance for small number of PEs. Nevertheless, the total processing time is determined by the slowest slave.

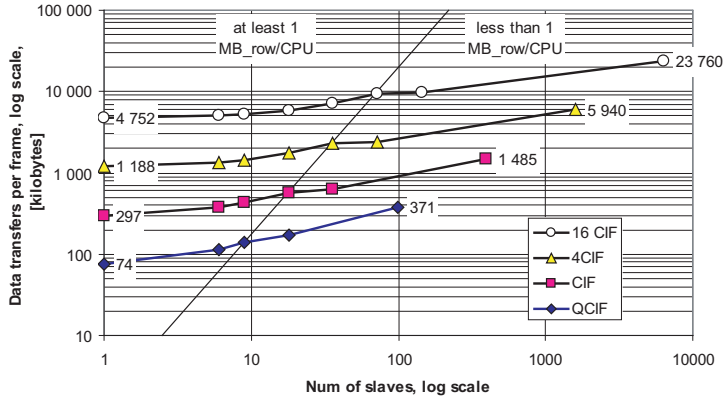
The data transfer requirements per frame are illustrated in Fig. 66(c). Four different frame sizes are included from QCIF (176 x 144 pixels) up to 16CIF (1408 x 1152). Note that both axes are logarithmic. Here, only the transfers of video data are considered, since they account approximately 99 per cent of all traffic. The graph is divided into two parts: left side depicting coarse, row-wise parallelization and right side depicting fine-grained parallelization. The division happens when the number of slaves equals the number of macroblock rows (*MB_row*). The largest possible system has equal number of slaves and macroblocks.

With one slave, the transferred data amount per frame equals 2 raw image frames and increases linearly to roughly 4 frames when the slice size is one row. At the most parallel configuration, i.e. one macroblock per PE, the total data amount is roughly 10 frames. At the same time, the required amount of local data memory per slave decreases. For 25 frames per second, the worst case total data rates, i.e. with maximum number of slaves, are approximately 10 MB/s (QCIF), 37 MB/s (CIF),



(a) Horizontal data-parallelization of QCIF video to 3 slaves.

(b) Frame sizes.



(c) Data transfer requirements per frame.

Fig. 66. *Data parallel video encoding.*

146 MB/s (4CIF), and 581 MB/s (16CIF). The data rates to the network from both master and IO are equal and vary between 1 MB/s (QCIF) and 60 MB/s (16CIF). The slave data rates are at maximum 0.2 MB/s (QCIF) and 1.6 MB/s (16CIF). As a conclusion, the data rates increase only modestly with the number of PEs. The above values account only the data transfers. Artifactual communication, for example cache misses, increase the traffic but is not considered here. Note that the number of PEs affects the problem size only in master's tasks which constitute only very small fraction of all computation.

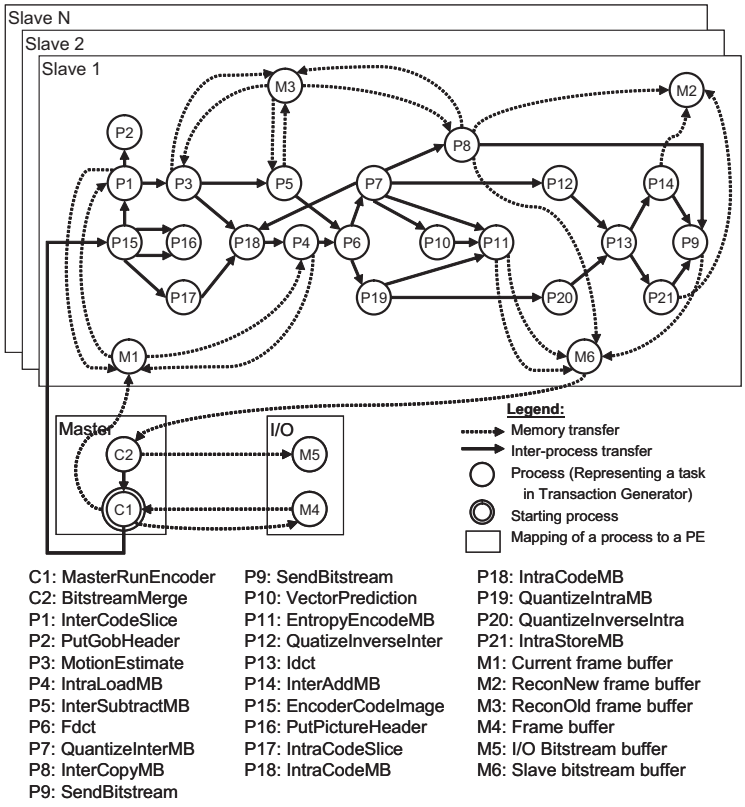


Fig. 67. *Task graph and the example mapping of the video encoder application. Each slave has identical set of encoding tasks that are initiated by master's task C*1.

10.5.2 Modeling with Transaction Generator

Transaction Generator [P2] [85] is utilized for simulating a transfer-dependent application model. It injects and reads data to/from a network according to external behavior of the application tasks. The data transfers between tasks that are mapped to separate PEs are forwarded to the network. The task runtime and transfer sizes are profiled from a real application in ARM7 instruction set simulator. The impact of hardware accelerator can be easily analyzed by profiling them and updating the corresponding task runtimes in the TG model. Only the data transfers are profiled since all PEs are assumed to have local instruction memories.

The application model is illustrated in Fig. 67. Both master and I/O unit execute two tasks. All slaves have the same program code and, hence, similar task graph

consisting of 21 computation tasks and 4 memory tasks. A shared memory system (omitted here for brevity) can be modeled by mapping the memory tasks into a shared memory instead of a slave PE. The structure of the task graph of one slave does not depend on the frame size whereas the transfer sizes and computation times do. Adding more slaves, increases the total task count from 29 with one slave to 1520 tasks with 72 slaves.

The accuracy of the model has been evaluated against cycle-accurate HW/SW co-simulation for QCIF frames and 1 to 9 PEs. The estimation error for runtime is about 10% on average. The co-simulation for 9 slaves took nearly 16 hours and less than 10 minutes in TG. On a basic parallelization scheme [105], there is at least one row per PE, but more parallel cases can be easily estimated with TG. However, the minimum slice size is set to 9 macroblocks for CIF and 18 macroblocks for 4CIF.

10.5.3 Exploration results

All the encoding times are measured for INTER frames. Unless otherwise specified, the PEs are operating at 100 MHz. All cases are run with DMA switched on and off. In this study, it is assumed that PE without DMA can transfer one word per cycle when network buffer allows that, i.e. transmit buffer is not full or receive buffer has valid data. Since the transfer requirements are rather modest, this study concentrates on single shared bus that is modeled in transaction level. HIBI network is used and configured as single bus. The transaction level model allows estimating ideal network by transmitting the whole data amount in one clock cycle. Such a transfer may contain the whole macroblock or even the whole frame. This way, it is possible to determine the minimum runtime for certain application by considering only the computation times and idle times due to transfer dependencies. In other cases, the width of the bus is 32 bits, i.e. 4 bytes, and bus frequency is either 1, 10, or 100 MHz.

The simulated speedups for CIF-sized video encoding are shown in Fig. 68. The straight line depicts ideal speedup which equals the number of slaves. Ideal network nearly achieves that estimate when the load is balanced and a 100 MHz bus with DMA is very close. The actual speedup is not linear due to load imbalance causing plateau areas in the curves. At the plateau, increasing the number of slaves does not decrease the largest slice size but only increases the traffic. A 100 MHz bus handles the increased traffic well but the frame rate drops slightly with smaller bus

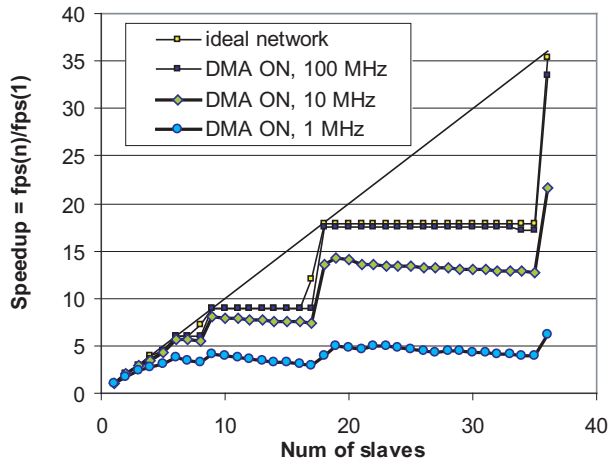


Fig. 68. *Speedup for CIF-sized frames with different bus frequencies. DMA is ON in all cases.*

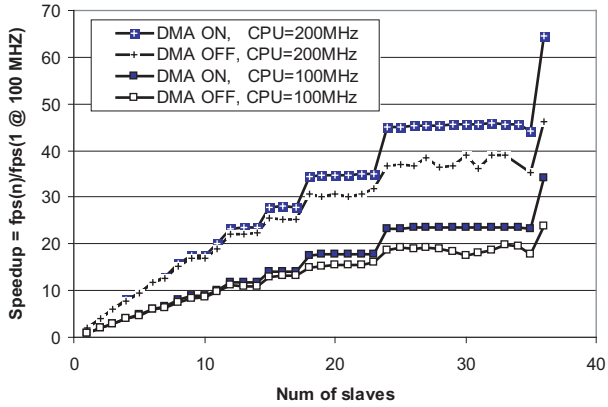


Fig. 69. *Speedup for 4CIF with PE frequency is 100 or 200 MHz, and bus frequency is 100 MHz. Results are given with and without DMA.*

frequencies. The improving steps occur when the number of macroblock rows is an integer multiple of the number of slaves.

As shown, the basic video encoder does not stress the communication network heavily due to modest PE frequency. Furthermore, the C codes could be optimized or some parts executed with hardware accelerators. Fig. 69 shows the speedup when the frequency of the PEs is doubled and bus frequency is 100 MHz. Speedup is given rel-

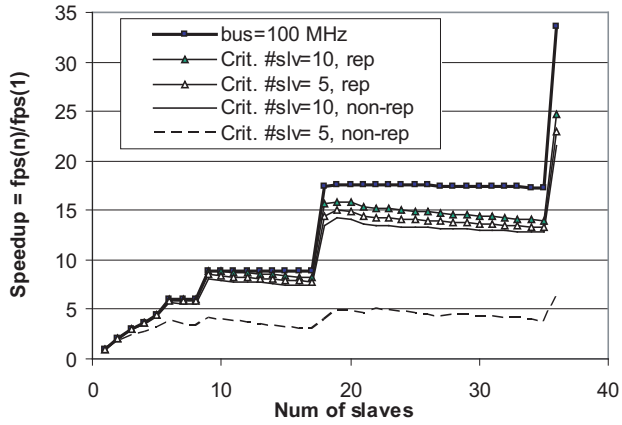


Fig. 70. *Speedup for CIF when bus frequency depends on the number of slaves. Results are shown for repeated and non-repeated wires. DMA is ON in all cases.*

ative to system with 1 CPU at 100 MHz and 100 MHz bus as in Fig. 68. This clearly demonstrates how the video encoding is dominated by computation times instead of communication.

Compared to Fig. 68 the shape of the curve is smoother due to larger frame size and, hence, load is more balanced. Furthermore, the impact of DMA, due to overlapping of computation as communication, is illustrated. Interestingly, a 10 MHZ bus with DMA gives similar performance as 100 MHz bus without DMA. DMA behavior is similar in all evaluated frames sizes. The additional speedup due to DMA depends linearly on the system size irrespective of the bus frequency. These speedups were at maximum 1.9x (CIF) and 1.55x (4CIF). Note that the speedup is larger if increased data rate of DMA compared to processor-controlled transfers was included.

So far, the bus frequency was assumed constant irrespective of the system size. In reality, the delay of a wire depends on its length. Repeater insertion changes the delay dependence from quadratic to linear but increases the power dissipation and area. The examples have shown that already a bus frequency of 100 MHz (same as PE frequency) provides enough capacity for real-time video processing and allows good scalability. More realistic frame rates can be interpolated from previous results, when the maximum bus frequency is determined. These are shown in Fig. 70 along with constant 100 MHz case.

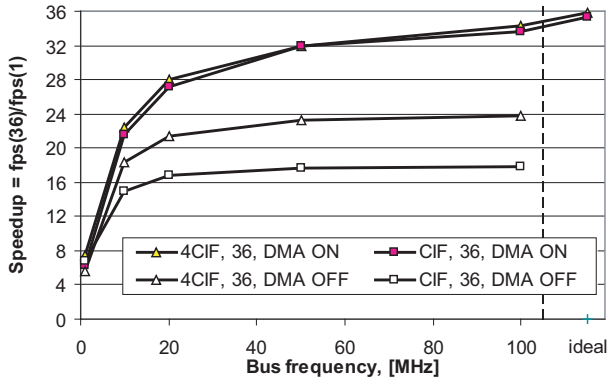


Fig. 71. *Speedup for CIF and 4CIF as a function of bus frequency. System size is 36 slave PEs. Results are shown with and without DMA.*

Since the frequency depends on the utilized silicon technology and final layout, simple frequency estimation is utilized here. Cases with different critical system size, i.e. the largest possible system that can operate at PE frequency, are shown. For example, when the system size is two times the critical size, the bus length has doubled. Consequently, the frequency of non-repeated bus has dropped to 25% of the original in case (and to 50% with repeaters). Cases with critical sizes 5 and 10 PEs are shown. For repeated global wires, the reachable distance per clock cycle is, for example, 28 mm (180 nm process, 700 MHz) and 10 mm (70 nm process, 2.5 GHz) [82]. Hierarchical networks have shorter links and, hence, their operating frequency is less dependent (in optimal case independent) of the system size. They need to be adopted in large parallel systems - not necessarily due to shared bandwidth of the bus, but due to prohibitively long delays in long wires. Considering the area costs of the bus are often lower than those of NoCs, a bus-based network is still a good candidate for many systems. Simulations were run also with 2-level hierarchical bus. The runtimes are (practically) identical whereas the maximum wire lengths are halved.

Fig. 71 shows the correlation between bus frequency and obtained speedup in a system with 36 PEs. The speedup achieved with ideal network is also shown (DMA has no effect then). The speedup is the same for both frame sizes when DMA used. However, larger frame size, 4CIF, achieves greater speedup than CIF when DMA is not used.

10.6 HIBI-based multiprocessor SoCs

As mentioned, artificial traffic is necessary during NoC design and evaluation. At the same time, such studies must be complemented with results from real applications. HIBI has been utilized in multiprocessor video encoder [105, 214] and in wireless local area network (WLAN) terminal [122, 205]. These MPSoCs were evaluated with HW/SW co-simulation.

Network evaluation with simulation has limited applicability due low execution speed and poor availability of compatible models. Hence, multiple FPGA prototypes were developed using HIBI v.2. At first, a basic platform was implemented using synthesizable Nios processors by Altera [P4]. An 8-processor multiprocessor system utilizing HIBI has also been prototyped and it takes about 36 400 logic elements, which is 88% of logic resources on Altera Stratix 1S40 FPGA. The prototypes and the lessons learned are discussed only briefly here due to space limitations. Interested readers are referred to cited works for further details.

In all simulated and prototyped cases, HIBI has met all the performance requirements with clear margin and enabled simple integration of system components. For example, in WLAN terminal [122], HIBI was responsible of mere 0.3% of packet's transmission latency. The FPGA experiments showed well the applicability of HIBI on a reconfigurable devices where the wiring resources are scarce and affect the operating frequency notably.

10.6.1 Wireless video terminal on FPGA

HIBI has been used to implement a configurable multiprocessor platform on FPGA [10]. It supports distributed execution of UML 2.0 designed applications with aid of eCos real-time operating system and in-house developed SW platform. Multiprocessor platform has been used for implementing, for example, a wireless video terminal on FPGA [124] that is shown in Fig. 72. Commercial WLAN radio and camera modules were connected to the expansion headers of the development board. The user can, for example, record and playback video, or stream it to/from another terminal. The main objective of this work was to study the feasibility of the used Koski design methodology and tools to implement a multimedia terminal comprising various subsystems, each comprising of several functional components. This objective was

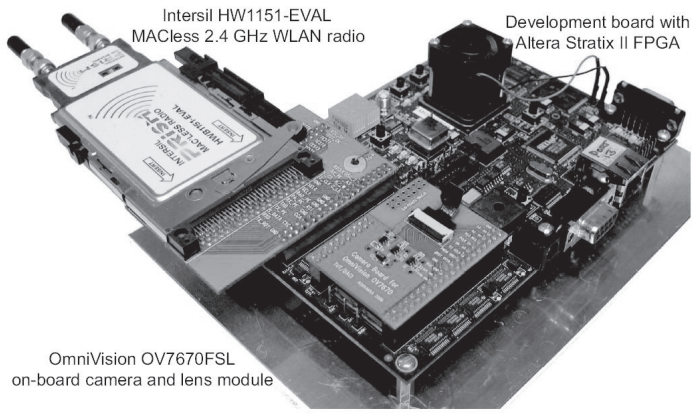


Fig. 72. *Wireless video terminal on FPGA development board. [124]*

fulfilled with very pleasant results as the design flow tools enable extensive design automation in implementation from high-abstraction level models to a complete multiprocessor SoC on FPGA. It was rather straightforward to add new CPU subsystems and accelerators to the HIBI-based system.

10.6.2 MPEG-4 Video encoder of FPGA

An MPEG-4 video encoder has been implemented on HIBI-based 4-processor system on a single FPGA [146]. The system has been expanded and enhanced in many ways after that [128, 131]. The basic structure of the configurable platform is shown in Fig. 73. It is a combination of in-house developed IP and commercial components delivered by FPGA vendor, Altera in this case. The bridges are optional. The example shows 5 CPU sub-systems, each equipped with local memories, DMA controller, and basic peripherals, such as timers. Note that slave CPUs can share an on-chip instruction memory. This is especially beneficial when single-program multiple data (SPMD) programming paradigm is used.

In addition to programmable processors, there are hardwired accelerators and special units. For example, a resource manager (RM) unit keeps track of accelerator utilization and reservation, and provides mutual exclusion when CPUs wish to access them [203].

For example, it is straightforward to increase the number of motion estimators whereas

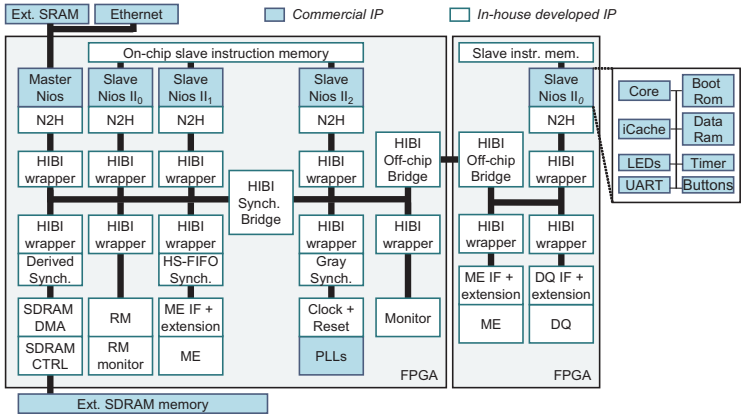
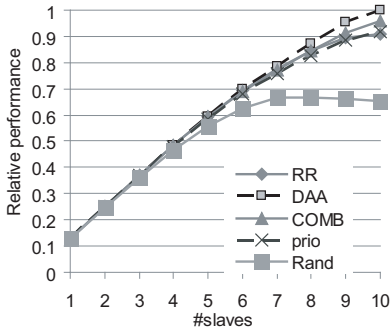
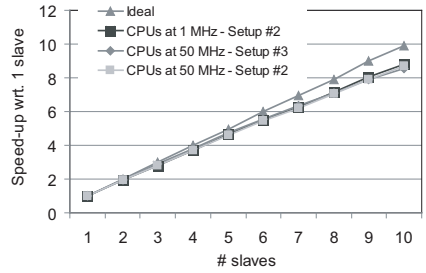


Fig. 73. *The block diagram of HIBI-based MPSoC on (two) FPGA(s).*



(a) Impact of arbitration algorithm for HW-accelerated video encoder. [130]

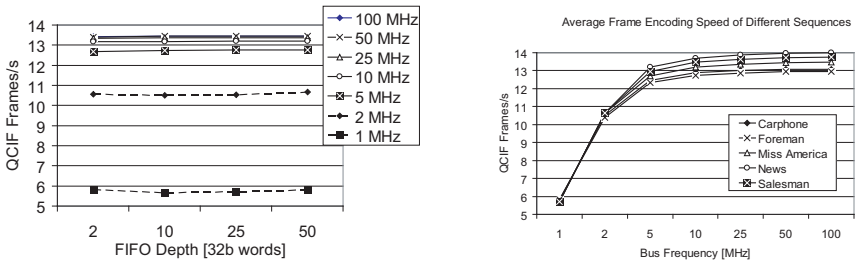


(b) Impact of 50 MHz HIBI on speedup. Setup #2 is SW only and setup #3 is HW-accelerated. [131]

Fig. 74. *The speedup achieved by increasing the number of encoding slave processors with respect to one slave. Test case is MPEG-4 encoder with CIF frames.*

the other architecture and application software stay intact. The shown MPSoC supports GALS paradigm as there are various clock synchronizers. For example, designer may use one clock for the processors, one for HIBI, and one for all the rest. Depending on the utilized synchronizers, the clocks may be totally independent or derived from each other.

Fig. 74 shows the frame rate as a function of the number of encoding slave CPUs [130, 131]. Fig. 74(a) shows the impact of arbitration algorithm. Random algorithm



(a) Frame rate as a function of FIFO depth. (b) Frame rate as a function of bus frequency.

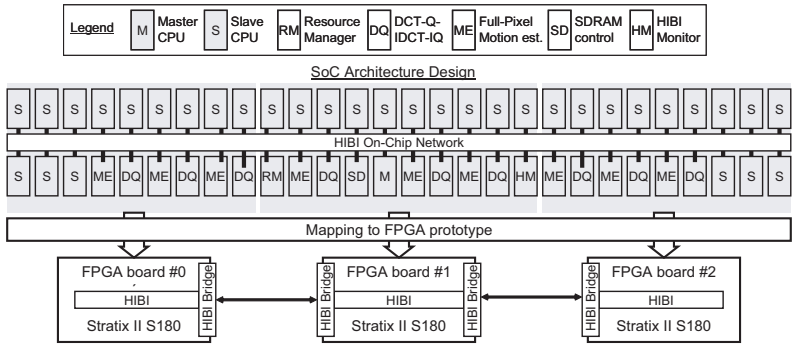
Fig. 75. *The impact of HIBI to MPEG-4 QCIF frame rate [131].*

is clearly the worst whereas the others are within 10% of each other. DAA performed best. The scalability of performance is almost linear when the system size increases from 1 to 10 encoding slave CPUs.

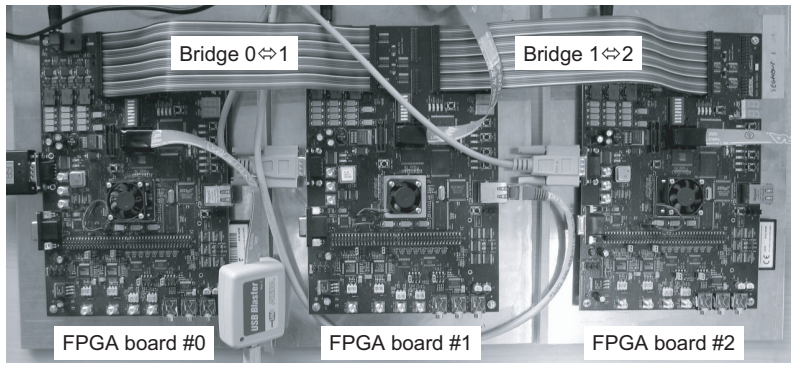
Fig. 74(b) illustrates the impact of HIBI on the performance of the video encoder. In practice, the case with 1 MHz CPUs and other components at 100 MHz depicts how the software scales. The amount of sequential code causes the gap between that curve and the ideal. On the other hand, the hardware scalability is very high since the difference between the "CPUs at 1 MHz"-curve and the others is negligible. HW-acceleration increases the data traffic by factor of 2.4x compared to SW only implementation. Despite this, scalability is not affected and hence one can conclude that HIBI offers adequate bandwidth.

A novel approach of frequency scaling is used to isolate the impact of various architecture components. Scaling the bus frequency with respect to CPUs reveals the minimum requirements for the bus as well as system's maximum performance. The impact of shared bus on the encoding performance was studied in [131]. The depth of the FIFO buffers affects the number of retransfers (cf. Fig. 62) but did not affect the encoding speed, as illustrated in Fig. 75(a). Bus was shown to offer adequate performance for real-time encoding Fig. 75(b). It turned out that 10 MHz was adequate for HIBI and higher frequencies improved performance only little.

A special case for HIBI was presented in [129] where a large MPSoC is spread into several FPGA chips. The largest evaluated system had 58 IP components (35 soft NIOS II CPUs and 23 other IP) on 3 FPGA boards each them having one HIBI segment, as shown in Fig. 76. The boards were connected together between the two halves of a HIBI bridge.



(a) The block diagram of the architecture.



(b) A photo of the platform used with necessary cabling.

Fig. 76. *A hierarchical HIBI used for interconnecting* 58 *IP components on a 3-FPGA prototype [129].*

10.7 Lessons learned

Very large differences were observed between NoCs, both in literature and in this work. Furthermore, the performance is very sensitive to the studied application, its mapping, and the parameters of the NoC. All this emphasizes the need for disciplined benchmarking methodology to obtain good estimates on the performance in different scenarios. The presented Transaction Generator provided a good framework for running benchmarks.

Fig. 77 shows a rough summary of the studies. The suitability of the network is evaluated by their area vs. runtime trade-off. The X axis the shows the system size, namely the number of terminals whereas the Y axis shows the difficulty of the traffic load. The difficulty is a product of four factors: high throughput requirement, require-

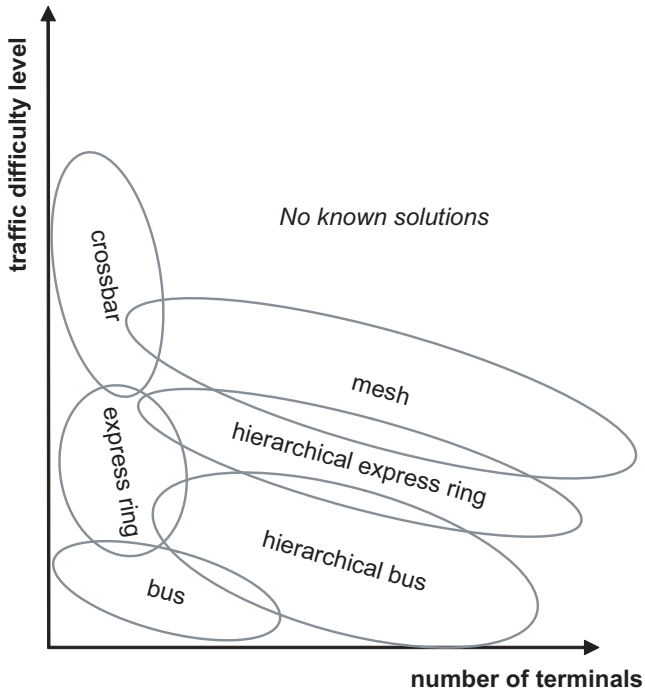


Fig. 77. *The suitability of reference networks considering the number of terminals and difficulty of the traffic load. The cost function is area divided by runtime.*

ment for low latency, very bursty nature, and wide target distribtution. The circles approximate the most appropriate usage scenario for 6 NoC topologies. Single-hop topologies - bus and crossbar- are limited to small or medium sized systems, e.g. 2-16 terminals. Multi-hop topologies support larger systems. Note that the top right corner is considered too difficult for all known approaches.

In general, HIBI and 2-D mesh performed well in the presented cases. However, it was noted during the development of MPEG-4 encoder that debugging was much easier when (hierarchical) bus was used instead of mesh. Both networks showed that easy instantiation directly from VHDL code was very useful feature for design space exploration. This was enabled by algorithmic parameterization of the components (wrappers, NIs, routers etc.) and there was no need for designer interaction via graphical user interface. The implemented scratch-pad memory system with DMA controllers offered enough performance and its program control was not too difficult in such a regular application.

Furthermore, several FPGA prototypes were implemented and they confirmed the

utility of HIBI in MP-SoC environment. For example, the clock frequency or area of HIBI did not restrict the FPGA system development at any point. In many cases, the memories (both internal and external) and the sheer complexity of the application were the most severe bottlenecks. Physical prototypes with real applications help to account such real-world phenomena in evaluation. Hence, they are an excellent way to remove an excess optimism that may arise when a complex systems is considered only on a very high abstraction level.

11. CONCLUSIONS

The main goals of the work were met. The initial objective in this work was to develop HIBI NoC further. The work started with a survey on on-chip buses and the first versions of traffic generator was developed at the same time. Then, a new version of HIBI was developed and evaluated. At that point, the importance of comparison, contrasting between approaches, and wide range of scenarios became evident. That led to surveys regarding NoCs, NoC comparisons, and larger emphasis on traffic generation and benchmarking. At the same time, the development of HIBI continued and reference NoCs were also implemented.

During this work, it was found that NoC parameters affect the performance unpredictably and are thus hard to optimize. This fact emphasizes the importance of automated exploration and parameter optimization.

An OCP-IP workgroup seeks to define a common NoC benchmarks [P9, P12] [157]. It is formalizing a set of relevant metrics, associated measurement methodologies, and a set of parameterized reference inputs for the NoC benchmarks. These ensure meaningful comparison between various sources and the overall view can be determined in incremental steps. The presented benchmarking method is very suitable for evaluating network-on-chips and was adopted by the workgroup. The author has contributed, for example, to the basic requirements, application and traffic modeling, metric selection, and XML description format for benchmarks. The author is currently the chairman of the workgroup. The workgroup aims to provide the academic and industrial research and development (R&D) communities with a set of characteristic benchmark designs and guidelines that will serve as a common repository of relevant information with the following objectives:

- Enable the sharing and comparison of NoC-related R&D efforts and findings
- Enable and accelerate the NoC paradigm development

- Increase the reproducibility of R&D claims and results
- Bridge the gap between academic and industrial state of the art.

As noted earlier, the term NoC has a multitude of contradictory and vague definitions. Hence, just having a *NoC* does not tell much. After studying the seminal work in the field and according to own experiments, the author projects that *a good NoC*

- separates communication from computation
- does not utilize long, global wires spanning the whole chip
- does not need a global, centralized controller for communication
- has a topology that allows the addition of links as the system size grows (offers scalability)
- allows customization (link width, buffer sizes, even topology)
- allows an arbitrary number of terminals
- allows multiple voltage and frequency domains
- delivers data in-order either naturally or via a layered protocol
- offers services with varying guarantees
- offers support for system testing
- is reported in adequate detail to justify the claims and to allow meaningful comparison.

The first property is necessary to design and verify PEs independently from the network and simplifies reuse and design space exploration. Long signal wires are problematic due to their poor scaling with technology, large delay, noise, and power consumption. In a parallel system, it is necessary to remove the obvious bottleneck, whether it is a single shared link, memory, or arbiter. Allowing an arbitrary number of terminals (not just power of two, for example) and in-order delivery of data ensure suitability to a wide range of systems. Services, for example low-latency, high-throughput, or limited jitter, provide support for the heterogeneous data traffic

inherent in SoC. Built-in-self-test supports timely delivery of products and field testing. The two last are mainly related to reporting style to obtain maximum benefit for the research community. Naturally, certain properties, such as energy, area, and bit error rate, have to be minimized but the opposite goal is very unlikely anyway.

Based on the work, it can be concluded that no single network categorically outperforms all others. Moreover, it must be stressed that performance is very case dependent and affected by so numerous parameters that generalizations are very complex to achieve. Hence, the impact of certain settings cannot always be captured analytically or only with rough simplifications. To make things worse, comparisons in literature are often carried out optimistically promoting the author's own approach, one may add. Therefore in most papers, *"this work"* seems to be the best approach by default. Independent evaluation using the presented methods should alleviate that problem.

BIBLIOGRAPHY

[1] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, "SPIN: a scalable, packet switched, on-chip micro-network," in *DATE*, Mar. 2003, pp. 70–73.

[2] B. Ahmad, A. Erdogan, and S. Khawam, "Architecture of a dynamically reconfigurable NoC for adaptive reconfigurable MPSoC," in *AHS*, Jun. 2006, pp. 405–411.

[3] T. Ahonen and J. Nurmi, "Integration of a NOC-based multimedia processing platform," in *FPL*, Aug. 2005, pp. 606–611.

[4] T. Ainsworth and T. Pinkston, "On characterizing performance of the Cell Broadband Engine Element Interconnect Bus," in *NOCS*, May 2007, pp. 18–29.

[5] T. Andel and A. Yasinsac, "On the credibility of manet simulations," *IEEE Computer*, vol. 39, no. 7, pp. 48–54, Jul. 2006.

[6] A. Andriahantenaina and A. Greiner, "Micro-network for SoC: Implementation of a 32-port SPIN network," in *DATE*, Mar. 2003, pp. 1128–1129.

[7] F. Angiolini, P. Meloni, S. Carta, L. Benini, and L. Raffo, "Contrasting a NoC and a traditional interconnect fabric with layout awareness," in *DATE*, Mar. 2006, pp. 1–6.

[8] F. Angiolini, P. Meloni, S. Carta, L. Raffo, and L. Benini, "A layout-aware analysis of networks-on-chip and traditional interconnects for MPSoCs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 3, pp. 421–434, Mar. 2007.

[9] K. Anjo, Y. Yamada, M. Koibuchi, A. Jouraku, and H. Amano, "Black-bus: a new data-transfer technique using local address on networks-on-chips," in *IPDPS*, Apr. 2004, pp. 10–17.

[10] T. Arpinen, P. Kukkala, E. Salminen, M. Hännikäinen, and T. Hämäläinen, "Configurable multiprocessor platform with RTOS for distributed execution of UML 2.0 designed applications," in *DATE*, Mar. 2006, pp. 1324–1329.

[11] Arteris S.A., "A comparison of network-on-chip and busses," white paper, 2005.

[12] A. Baghdadi, W. Cesario, A. Jerraya, and N.-E. Zergainoh, "Combining a performance estimation methodology with a hardware/software codesign flow supporting multiprocessor systems," *IEEE Trans. Software Engineering*, vol. 28, no. 9, pp. 822–831, Sep. 2002.

[13] D. H. Bailey, "Twelve ways to fool the masses when giving performance results on parallel computers," *Supercomputing Review*, vol. 4, no. 8, pp. 54–57, Aug. 1991.

[14] A. Banerjee, R. Mullins, and S. Moore, "A power and energy exploration of network-on-chip architectures," in *NOCS*, May 2007, pp. 163–172.

[15] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende, and W. R. Stewart Jr., "Designing and reporting on computational experiments with heuristic methods," *Journal of Heuristics*, vol. 1, no. 1, pp. 9–32, Sep. 1995.

[16] T. Bartic, D. Desmet, J.-Y. Mignolet, T. Marescaux, D. Verkest, S. Vernalde, R. Lauwereins, J. Miller, and F. Robert, "Network-on-chip for reconfigurable systems: From high-level design down to implementation," in *FPL*, 2004, pp. 637–647.

[17] T. Bartic, J.-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins, "Topology adaptive network-on-chip design and implementation," *IEE Proc. Comput. Digit. Tech.*, vol. 152, no. 4, pp. 467–472, Jul. 2005.

[18] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An asynchronous NOC architecture providing low latency service and its multi-level design framework," in *ASYNC*, Mar. 2005, pp. 54–63.

[19] L. Benini and G. de Micheli, "Networks on chips: A new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[20] ——, *Networks on chips: technology and tools*. Morgan Kaufmann, 2006.

[21] D. Bertozzi, L. Benini, and G. de Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 818–831, Jun. 2005.

[22] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. de Micheli, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 113–129, Feb. 2005.

[23] O. Bessonov, D. Fougere, and B. Roux, "Using a parallel CFD code for evaluation of clusters and MPPs," in *IPDPS*, Apr. 2003.

[24] P. Bhojwani and R. Mahapatra, "Interfacing cores with on-chip packet-switched networks," in *VLSI design*, Jan. 2003, pp. 382–387.

[25] T. Bjerregaard and J. Sparsoe, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *DATE*, vol. 2, Mar. 2005, pp. 1226–1231.

[26] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, p. article No. 1, 2006.

[27] A. Bobrek *et al.*, "Modeling shared resource contention using a hybrid simulation/analytical approach," in *DATE*, vol. 2, Feb. 2004, pp. 1144–1149.

[28] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Cost considerations in network on chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 19–42, Oct. 2004.

[29] E. Bolotin, I. Gidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of Systems Architecture*, vol. 50, no. 2–3, pp. 105–128, Feb. 2004.

[30] S. Y. Borkar, P. Dubey, K. C. Kahn, D. J. Kuck, H. Mulder, S. S. Pawlowski, and J. R. Rattner, "Platform 2015: Intel processor and platform evolution for the next decade," Intel Corporation, white paper, 2005.

[31] A. Bouchhima, I. Bacivarow, W. Youssef, M. Bonaciu, and A. Jerraya, "Using abstract CPU subsystem simulation model for high level HW/SW architecture exploration," in *ASP-DAC*, Jan. 2005, pp. 969–972.

[32] A. Bouhraoua and M. E. Elrabaa, "A high-throughput network-on-chip architecture for systems-on-chip interconnect," in *Intl. Symposium on Soc*, Nov. 2006, pp. 127–130.

[33] W. R. Bryg, K. K. Chan, and N. S. Fiduccia, "A high-performance, low-cost multiprocessor bus for workstations and midrange servers," *Hewlett-Packard Journal*, vol. 47, no. 1, Feb. 1996.

[34] R. Cardoso, M. Kreutz, L. Carro, and A. Susin, "Design space exploration on heterogeneous network-on-chip," in *ISCAS*, May 2005, pp. 428–431.

[35] L. Carloni and A. Sangiovanni-Vincentelli, "Coping with latency in SOC design," *IEEE Micro*, vol. 22, no. 5, pp. 24–35, Sep.-Oct. 2002.

[36] D. Castells-Rufas, J. Joven, and J. Carrabina, "A validation and performance evaluation tool for ProtoNoc," in *Intl. Symposium on Soc*, Nov. 2006, pp. 159–162.

[37] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, *Surviving the SoC Revolution*. Kluwer Academic Publishers, 1999.

[38] K.-C. Chang, J.-S. Shen, and T.-F. Chen, "Evaluation and design trade-offs between circuit-switched and packet-switched NOCs for application-specific SOCs," in *DAC*, Jul. 2006, pp. 143–148.

[39] H. Charlery and A. Greiner, "A SystemC test environment for SPIN network," in *MIXDES*, Jun. 2006, pp. 449–453.

[40] H.-C. Chi and J.-H. Chen, "Design and implementation of a routing switch for on-chip interconnection networks," in *AP-ASIC*, Aug. 2004, pp. 392–395.

[41] M.-C. Chiang and G. S. Sohi, "Evaluating design choices for shared bus multiprocessors in a throughput-oriented environment," *IEEE Trans. Computers*, vol. 41, no. 3, pp. 297–317, Mar. 1992.

[42] D. Ching, P. Schaumont, and I. Verbauwhede, "Integrated modeling and generation of a reconfigurable network-on-chip," in *IPDPS*, Apr. 2004, pp. 139–145.

[43] Y.-S. Cho, E.-J. Choi, and K.-R. Cho, "Modeling and analysis of the system bus latency on the SoC platform," in *SLIP*, Mar. 2006, pp. 67–74.

[44] I. Cidon and I. Keidar, "Zooming in on network-on-chip architectures," Technion Department of Electrical Engineering, Tech. Rep. CCIT 565, Dec. 2005.

[45] B. Colwell, "Benchmarketing competition," *Computer*, pp. 9–11, Dec. 2003.

[46] R. P. Colwell, *Pentium Chronicles - The people, passion, and politics behind Intel's landmark chips*. Wiley Interscience, 2006.

[47] T. Conte, "Insight, not (random) numbers, keynote presentation,," keynote presentation at ISPASS, [online], http://ispass.org/ispass2005/keynote-conte.pdf, 2005.

[48] *2D-Fabric 402c Parallel Processing Interface*, Crossbow Technologies Inc., Sugar Land, TX, USA, Jul. 2002.

[49] D. Culler, J. Singh, and A. Gupta, *Parallel Computer Architecture - A Hardware/Software Approach*. Morgan Kaufmann, 1998.

[50] R. Cypher, A. Ho, S. Konstantinidou, and P. Messian, "Architectural requirements of parallel scientific applications with explicit communication," in *ISCA*, May 1993, pp. 2–13.

[51] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann Publishers, 2004.

[52] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *DAC*, Jun. 2001, pp. 684–689.

[53] S. Dasgupta and A. Yakovlev, "Modeling and performance analysis of GALS architectures," in *Intl. Symposium on Soc*, Nov. 2006, pp. 187–190.

[54] A. V. de Mello and L. H. Möller, "Hermes project web page," [online], http://toledo.inf.pucrs.br/ gaph/Projects/Hermes/Hermes.html, 2004.

[55] R. Dick, "Embedded system synthesis benchmarks suites (E3S), version 0.9," [online],http://www.ece.northwestern.edu/ dickrp/e3s/, 2002.

[56] R. Dick, D. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *CODES/CASHE*, Mar. 1998, pp. 97–101.

[57] R. Dobkin, R. Ginosar, and C. Sotiriou, "High rate data synchronization in GALS SoCs," *IEEE Trans. VLSI Syst.*, vol. 14, no. 10, pp. 1063–1074, 2006.

[58] F. Dumitrascu, I. Bacivarov, L. Pieralisi, M. Bonaciu, and A. Jerraya, "Flexible MPSoC platform with fast interconnect exploration for optimal system performance for a specific application," in *DATE*, Mar. 2006, pp. 1–6.

[59] EEMBC, "Certified performance analysis for embedded systems designers," [online], http://www.eembc.org/.

[60] D. Emma, A. Pescapè, and G. Ventre, "Analysis and experimentation of an open distributed platform for synthetic traffic generation," in *FTDCS*, Suzhou, China, May 2004, pp. 277–283.

[61] S. Evain, J. P. Diguet, and D. Houzet, "μspider: a CAD tool for efficient NoC design," in *Norchip*, Nov. 2004, pp. 218–221.

[62] B. Feero and P. Pande, "Performance evaluation for three-dimensional networks-on-chip," in *ISVLSI*, May 2007, pp. 305–310.

[63] F. Felicijan and S. Furber, "An asynchronous on-chip network router with quality-of-service (QoS) support," in *SOCC*, Sep. 2004, pp. 274–277.

[64] M. Forsell, "A scalable high-performance computing solution for networks on chips," *IEEE Micro*, vol. 22, no. 5, pp. 46–55, Sep.-Oct. 2002.

[65] E. Frachtenberg and D. G. Feitelson, "Pitfalls in parallel job scheduling evaluation," in *LNCS 3834: Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 2005, pp. 257–282.

[66] C. Gebotys and R. Gebotys, "A framework for security on NoC technologies," in *VLSI*, Feb. 2003, pp. 113–117.

[67] C. Grecu, A. Ivanov, R. Saleh, and P. Pande, "Testing network-on-chip communication fabrics," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 10, pp. 2201–2014, Dec. 2007.

[68] C. Grecu, A. Ivanov, R. Saleh, C. Rusu, L. Anghel, P. Pande, and V. Nuca, "A flexible network-on-chip simulator for early design space exploration," in *MNRC*, Oct. 2008, pp. 33–36.

[69] C. Grecu, L. Anghel, P. P. Pande, A. Ivanov, and R. Saleh, "Essential fault-tolerance metrics for NoC infrastructures," in *IOLTS*, Jul. 2007, pp. 37–42.

[70] E. Grochowski and M. Annavaram, "Energy per instruction trends in Intel microprocessors," *Technology at Intel Magazine*, pp. 1–8, Mar. 2006.

[71] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *DATE*, Mar. 2000, pp. 250–256.

[72] J. Guo, A. Papanikolaou, P. Marchal, and F. Catthoor, "Energy/area/delay trade-offs in the physical design of on-chip segmented bus architecture," in *SLIP*, Mar. 2006, pp. 75–81.

[73] J. Gustafson and R. Todi, "Conventional benchmarks as a sample of the performance spectrum," in *HICSS*, vol. 7, Jan. 1998, pp. 514–523.

[74] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *WWC-4*, Dec 2001, pp. 3–14.

[75] M. Hansen, H. Yalcin, and J. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *IEEE Des. Test Comput*, vol. 16, no. 3, pp. 72–80, 1999.

[76] A. Hansson and K. Goossens, "Trade-offs in the configuration of a network on chip for multiple use-cases," in *NOCS*, May 2007, pp. 233–242.

[77] W. Heirman, J. Dambre, and J. V. Campenhout, "Synthetic traffic generation as a tool for dynamic interconnect evaluation," in *SLIP*, Apr. 2007, pp. 65–72.

[78] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era," in *Norchip*, Nov. 2000.

[79] J. Henkel, W. Wolf, and S. Chakradhar, "On-chip networks: a scalable, communication-centric embedded system design paradigm," in *VLSI*, Jan. 2004, pp. 845–851.

[80] T. Henriksson, D. Wiklund, and D. Liu, "VLSI implementation of a switch for on-chip networks," in *DDECS*, Apr. 2003.

[81] C. Hilton and B. Nelson, "A flexible circuit switched NOC for FPGA based systems," in *FPL*, Aug. 2005, pp. 24–26.

[82] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.

[83] D. A. Hodges, H. G. Jackson, and R. A. Saleh, *Analysis and Design of Digital Integrated Circuits: in deep submicron technology*, 3rd ed. McGraw-Hill, 2004.

[84] K. Holma, T. Arpinen, E. Salminen, M. Hännikäinen, and T. Hämäläinen, "Real-time execution monitoring on multi-processor system-on-chip," in *Intl. Symposium on Soc*, Nov. 2008, pp. 23–28.

[85] K. Holma, M. Setälä, E. Salminen, and T. Hämäläinen, "Evaluating the model accuracy in automated design space exploration," *Microprocessors and Microsystems*, vol. 32, no. 5-6, pp. 321–329, Aug. 2008.

[86] R. Holsmark and S. Kumar, "Design issues and performance evaluation of mesh NoC with regions," in *Norchip*, Nov. 2005, pp. 40–43.

[87] M. Hosseinabady, M. Kakoee, J. Mathew, and D. Pradhan, "Reliable network-on-chip based on generalized de Bruijn graph," in *HLVDT*, Nov. 2007, pp. 3–10.

[88] J. Hu, Y. Deng, and R. Marculescu, "System-level point-to-point communication synthesis using floorplanning information," in *ASP-DAC/VLSI*, Jan. 2002, pp. 573–579.

[89] J. Hu and R. Marculescu, "DyAD - smart routing for networks-on-chip," in *DAC*, June 2004, pp. 260–263.

[90] *Intel Itanium 2 Processor - Hardware Developer's Manual*, Intel Corporation, Jul. 2002.

[91] Intel Corporation, "Quick processor reference guide," [online], http://www.intel.com/pressroom/kits/quickrefffram.htm, May 2006.

[92] ITRS, "International technology roadmap for semiconductors, editions 1999-2005," [online], www.http://www.itrs.net, May 2006.

[93] A. Janarthanan and K. A. Tomko, "MoCSYS: A multi-clock hybrid two-layer router architecture and integrated topology synthesis framework for system-level design of FPGA based on-chip networks," in *VLSID*, Jan. 2008, pp. 397–402.

[94] A. Jantsch, "Models of computation for networks on chip," in *ACSD*, Jun. 165–178, p. 2006.

[95] A. Jantsch, R. Lauter, and A. Vitkowski, "Power analysis of link level and end-to-end data protection in networks on chip," in *ISCAS*, vol. 2, May 2005, pp. 1770–1773.

[96] A. Jantsch and H. Tenhunen, Eds., *Networks on Chip*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2003.

[97] A. Jerraya and W. Wolf, *Multiprocessor Systems-on-Chips*. Morgan Kaufmann, Sep. 2004.

[98] A. A. Jerraya, "Long term trends for embedded system design," in *Euromicro DSD*, Sep. 2004, pp. 20–26.

[99] H. Jeschke, "Efficiency measures for multimedia SOCs," in *SAMOS VII*, Jul. 2007, pp. 190–199.

[100] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the Cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, no. 4/5, pp. 598–604, Jul.-Sep. 2005.

[101] S. Kakita *et al.*, "Functional model exploration for multimedia application via algebraic operators," in *ACSD*, June 2006, pp. 229–238.

[102] S. Kalidindi, N. Huynh, B. Eklow, and J. Goldstein, ""Real life" system testing of networking equipment," in *ITC*, Oct. 2004, pp. 1072–1077.

[103] T. Kangas, "Methods and implementations for automated system on chip architecture exploration," Ph.D. dissertation, Tampere University of Technology, 2006.

[104] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, and K. Kuusilinna, "UML-based multi-processor SoC design framework," *ACM Trans. Embedded Computing Systems*, vol. 5, no. 2, pp. 281–320, May 2006.

[105] T. Kangas, K. Kuusilinna, and T. D. Hämäläinen, "Scalable architecture for SoC video encoders," *Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology*, vol. 44, no. 1-2, pp. 79–95, Aug. 2006.

[106] T. Kangas, V. Lahtinen, K. Kuusilinna, and T. Hämäläinen, "System-on-chip communication optimization with bus monitoring," in *DDECS*, Apr. 2002, pp. 304–309.

[107] F. Karim, A. Nguyen, and S. Dey, "An interconnect architecture for networking systems on chips," *IEEE Micro*, vol. 22, no. 5, pp. 36–45, Sep.-Oct. 2002.

[108] H. Kariniemi, "On-line reconfigurable extended generalized fat tree network-on-chip for multiprocessor system-on-chip circuits," Ph.D. dissertation, Tampere University of Technology, 2006.

[109] H. Kariniemi and J. Nurmi, "Reusable XGFT interconnect IP for network-on-chip implementations," in *Intl. Symposium on Soc*, Nov. 2004, pp. 94–102.

[110] N. Kavaldjiev, G. Smit, P. Jansen, and P. Wolkotte, "A virtual channel network-on-chip for GT and BE traffic," in *ISVLSI*, vol. 00, Mar. 2006.

[111] M. Keating and P. Bricaud, *Reuse Methodology Manual*, 3rd ed. Kluwer Academic Publishers, 2002.

[112] K. Keutzer, S. Malik, R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, Dec. 2000.

[113] D. Kim, M. Kim, and G. E. Sobelman, "Design of a high-performance scalable CDMA router for on-chip switched networks," in *ISSOC*, 2005, pp. 32–35.

[114] D. Kim, K. Kim, J.-Y. Kim, S. Lee, and H.-J. Yoo, "An 81.6 GOPS object recognition processor based on NoC and visual image processing memory," in *CICC*, Sep., pp. 443–446.

[115] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in *DAC*, 2005, pp. 559–564.

[116] S. Kim, C. Im, and S. Ha, "Schedule-aware performance estimation of communication architecture for efficient design space exploration," *IEEE Trans. VLSI Systems*, vol. 13, no. 5, pp. 539–552, May 2005.

[117] T. Kogel, R. Leupers, and H. Meyer, *Integrated System-level Modeling of Network-on-Chip enabled Multiprocessor Platforms*. Springer, 2006.

[118] K. Kozminski, "Benchmarks for layout synthesis - evolution and current status," in *DAC*, Jun. 1991, pp. 265–270.

[119] C. Kretzschmar, A. Nieuwland, and D. Muller, "Why transition coding for power minimization of on-chip buses does not work," in *DATE*, vol. 1, Feb. 2004, pp. 512–517.

[120] M. Kreutz, L. Marcon, L. Carro, N. Calazans, and A. Susin, "Energy and latency evaluation of NoC topologies," in *ISCAS*, vol. 6, May 2005, pp. 5866–5869.

[121] M. E. Kreutz, L. Carro, C. A. Zeferino, and A. A. Susin, "Communication architectures for system-on-chip," in *SBCCI*, Sep. 2001, pp. 14–19.

[122] P. Kukkala, M. Hännikäinen, and T. Hämäläinen, "Co-simulation of wireless local area network terminals with protocol software implemented in SDL," in *Euromicro DSD*, Aug. 2005, pp. 161–164.

[123] P. Kukkala, V. Helminen, M. Hännikäinen, and T. Hämäläinen, "UML 2.0 implementation of an embedded WLAN protocol," in *PIMRC*, Sep. 2004, pp. 1158–1162.

[124] P. Kukkala, M. Setälä, T. Arpinen, E. Salminen, M. Hännikäinen, and T. Hämäläinen, "Implementing a WLAN video terminal using UML and fully-

automated design flow," *EURASIP Journal on Embedded Systems*, no. Embedded Digital Signal Processing Systems, Jan 2007.

[125] A. Kulmala, "Multiprocessor system with general-purpose interconnection architecture on FPGA," Master's thesis, Tampere University of Technology, 2005.

[126] A. Kulmala, T. D. Hämäläinen, and M. Hännikäinen, "Comparison of GALS and synchronous architectures with MPEG-4 video encoder on multiprocessor system-on-chip FPGA," in *Euromicro DSD*, Sep. 2006, pp. 83–86.

[127] A. Kulmala, M. Hännikäinen, and T. D. Hämäläinen, "Reliable GALS implementation of MPEG-4 encoder with mixed clock FIFO on standard FPGA," in *FPL*, Aug., pp. 495–500.

[128] A. Kulmala, O. Lehtoranta, T. D. Hämäläinen, and M. Hännikäinen, "Scalable MPEG-4 encoder on FPGA multiprocessor SoC," *EURASIP Journal on Embedded Systems*, no. Field-Programmable Gate Arrays in Embedded Systems, Jun. 2006.

[129] A. Kulmala, E. Salminen, and T. D. Hämäläinen, "Prototyping and evaluating large system-on-chips on Multi-FPGA platform," in *SAMOS VII*, Jul. 2007, pp. 179–189.

[130] ——, "Distributed bus arbitration algorithm comparison on FPGA based MPEG-4 multiprocessor SoC," *IET Computers and Digital Techniques*, vol. 2, no. 4, pp. 314–325, Jul. 2008.

[131] ——, "Evaluating SoC network performance in MPEG-4 encoder," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, May 2008.

[132] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, K. Tiensyrjä, and A. Hemani, "A network on chip architecture and design methodology," in *VLSI*, April 2002, pp. 105–112.

[133] K. Kuusilinna, T. Hämäläinen, P. Liimatainen, and J. Saarinen, "Low-latency interconnection for IP-block based multimedia chips," in *PDCN*, Dec. 1998, pp. 411–416.

[134] K. Lahiri, A. Raghunathan, and S. Dey, "Evaluation of the traffic-performance characteristics of system-on-chip communication architectures," in *VLSI*, Jan. 2001, pp. 29–35.

[135] ——, "Design space exploration for optimizing on-chip communication architectures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 6, pp. 952–961, Jun. 2004.

[136] V. Lahtinen, "Design and analysis of interconnection architectures," Ph.D. dissertation, Tampere University of Technology, Jun. 2004.

[137] V. Lahtinen, T. Kangas, E. Salminen, K. Kuusilinna, and T. D. Hämäläinen, "Reducing SoC power consumption with generic interconnection components and TDMA-based arbitration," in *DDECS*, Apr. 2003, pp. 261–268.

[138] V. Lahtinen, K. Kuusilinna, T. Kangas, and T. Hämäläinen, "Interconnection scheme for continuous-media systems-on-chip," *Microprocessors and Microsystems*, vol. 26, no. 3, pp. 123–138, Apr. 2002.

[139] V. Lahtinen, E. Salminen, K. Kuusilinna, and T. Hämäläinen, "Comparison of synthesized bus and crossbar interconnection architectures," in *ISCAS*, vol. 5, May 2003, pp. 433–436.

[140] A. Lambrechts *et al.*, "Power breakdown analysis for a heterogeneous NoC platform running a video application," in *ASAP*, Jul. 2005, pp. 179–184.

[141] D. Lattard, E. Beigne, F. Clermidy, Y. Durand, R. Lemaire, P. Vivet, and F. Berens, "A reconfigurable baseband platform based on an asynchronous network-on-chip," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 223–235, Jan. 2008.

[142] H. G. Lee, U. Y. Ogras, R. Marculescu, and N. Chang, "Design space exploration and prototyping for on-chip multimedia applications," in *DAC*, Jul. 2006, pp. 137–142.

[143] K. Lee, S.-J. Lee, S.-E. Kim, H.-M. Choi, D. Kim, S. Kim, M.-W. Lee, and H.-J. Yoo, "A 51mW 1.6GHz on-chip network for low-power heterogeneous SoC platform," in *ISSCC*, Feb. 2004, pp. 152–158.

[144] K. Lee, S.-J. Lee, and H.-J. Yoo, "Low-power network-on-chip for high-performance SoC design," *IEEE Trans. VLSI Syst.*, vol. 14, no. 2, pp. 148–160, Feb. 2006.

[145] S. Lee, C. Lee, and H.-J. Lee, "A new multi-channel on-chip-bus architecture for system-on-chips," in *SOCC*, Sep. 2004, pp. 305–308.

[146] O. Lehtoranta, E. Salminen, A. Kulmala, M. Hännikäinen, and T. Hämäläinen, "A parallel MPEG-4 encoder for FPGA based multiprocessor SoC," in *FPL*, Aug. 2005, pp. 380–385.

[147] D. E. Lenoski and W.-D. Weber, *Scalable Shared-memory Multiprocessing*. Morgan Kaufman Publishers Inc., 1995.

[148] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, and D. Verkest, "Spatial division multiplexing: a novel approach for guaranteed throughput on NoCs," in *CODES+ISSS*, Sep. 2005, pp. 81–86.

[149] K. Leung and K. Yeung, "The design and implementation of a WWW traffic generator," in *ICPADS*, Jul. 2000, pp. 509–514.

[150] J. Leverich, H. Arakida, A. Solomatnikov, A. Firoozshahian, M. Horowitz, and C. Kozyrakis, "Comparing memory systems for chip multiprocessors," in *ISCA*, Jun. 2007, pp. 358–368.

[151] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier, "An architecture and compiler for scalable on-chip communication," *IEEE Trans. VLSI Syst.*, vol. 12, no. 7, pp. 711–726, Jul. 2004.

[152] P. Liljeberg, J. Plosila, and J. Isoaho, "Self-timed ring architecture for SOC applications," in *SOCC*, Sep. 2003, pp. 359–362.

[153] A. Lines, "Asynchronous interconnect for synchronous SoC design," *IEEE Micro*, vol. 24, no. 1, pp. 32–41, 2004.

[154] J. Liu, L.-R. Zheng, and H. Tenhunen, "Interconnect intellectual property for network-on-chip (NoC)," *Journal of Systems Architecture*, vol. 50, no. 2-3, pp. 65–79, Feb. 2004.

[155] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon, "Analyzing on-chip communication in a MPSoC environment," in *DATE*, Feb. 2004, pp. 752–757.

[156] R. Lu and C.-K. Koh, "Samba-bus: a high performance bus architecture for system-on-chips," in *ICCAD*, Nov. 2003, pp. 8–12.

[157] Z. Lu, A. Jantsch, E. Salminen, and C. Grecu, *Network-on-Chip Benchmarking Specification Part 2: Micro-Benchmark Specification Version 1.0*, OCP-IP, May 2008.

[158] Z. Lu, M. Zhong, and A. Jantsch, "Evaluation of on-chip networks using deflection routing," in *GLSVLSI*, May 2006, pp. 296–301.

[159] S. Mahadevan, F. Angiolini, M. Storgaard, R. G. Olsen, J. Sparsø, and J. Madsen, "Network traffic generator model for fast network-on-chip simulation," in *DATE*, vol. 2, Mar. 2005, pp. 780–785.

[160] T. Marescaux, E. Brockmeyer, and H. Corporaal, "The impact of higher communication layers on NoC supported MP-SoCs," in *NOCS*, May 2007, pp. 107–116.

[161] E. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," in *ITC*, Oct. 2002, pp. 519–528.

[162] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Evaluation of current QoS mechanisms in network on chip," in *Intl. Symposium on Soc*, Nov. 2006, pp. 115–118.

[163] M. Millberg, R. T. E. Nilsson, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip," in *DATE*, Feb. 2004.

[164] A. Mitra, M. Lajolo, and K. Lahiri, "SOFTENIT: a methodology for boosting the software content of system-on-chip designs," in *GLSVLSI*, Apr. 2005.

[165] S. Mohanty and V. Prasanna, "Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures."

[166] F. Mondinelli, M. Borgatti, and Z. Vajna, "A 0.13 um 1Gb/s/channel store-and-forward network on-chip," in *SOCC*, Sep. 2004, pp. 141–142.

[167] G. E. Moore, “Cramming more components onto integrated circuits,” *Electronics magazine*, pp. 114–117, Apr. 1965.

[168] ——, “No exponential is forever: but "Forever" can be delayed!” in *ISSCC, Digest of Technical Papers*, 2003, pp. 20–23.

[169] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, “Hermes: an infrastructure for low area overhead packet-switching networks on chip,” *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69–93, Oct. 2004.

[170] A. Morgenshtein, I. Cidon, A. Kolodny, and R. Ginosar, “Comparative analysis of serial vs parallel links in NoC,” in *Intl. Symposium on Soc*, Nov. 2004, pp. 185–188.

[171] R. Mullins, “Minimising dynamic power consumption in on-chip networks,” in *Intl. Symposium on Soc*, Nov. 2006, pp. 119–122.

[172] R. Mullins, A. West, and S. Moore, “The design and implementation of a low-latency on-chip network,” in *ASP-DAC*, Jan. 2006.

[173] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, L. Raffo, and G. de Micheli, “Designing message-dependent deadlock free networks on chips for application-specific systems-on-chips,” in *IFIP*, 2006.

[174] Nanoscale Integration and Modeling Group, “Predictive technology model,” Arizona State University, [online], http://www.eas.asu.edu/ ptm/, 2006.

[175] C. Neeb, M. Thul, and N. Wehn, “Network-on-chip-centric approach to interleaving in high throughput channel decoders,” in *ISCAS*, May 2005, pp. 1766–1769.

[176] E. Nilsson and J. Öberg, “PANACEA - a case study on the PANACEA NoC - a Nostrum network on chip prototype,” Royal Institute of Technology, Tech. Rep. 229, Apr. 2006.

[177] J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, Eds., *Interconnect-Centric Design for Advanced SoC and NoC*. Kluwer Academic Publishers, 2004.

[178] *Open Core Protocol Specification, Release 2.0*, OCP-IP Alliance, Portland, OR, 2003.

[179] S. Ogg, E. Valli, C. D'Alessandro, A. Yakovlev, B. Al-Hashimi, and L. Benini, "Reducing interconnect cost in NoC through serialized asynchronous links," in *NOCS*, May 2007, pp. 219–219.

[180] U. Y. Ogras, J. Hu, and R. Marculescu, "Key research problems in NoC design: a holistic perspective," in *CODES*, Sep. 2005, pp. 69–75.

[181] U. Ogras, R. Marculescu, H. G. Lee, and N. Chang, "Communication architecture optimization: making the shortest path shorter in regular networks-on-chip," in *DATE*, Mar. 2006, pp. 6–10.

[182] H. Orsila, E. Salminen, and T. D. Hämäläinen, *Global optimization: Focus on Simulated annealing*. ITECH, 2008, ch. Best Practices for Simulated Annealing in Multiprocessor Task Distribution Problems.

[183] H. Orsila, E. Salminen, M. Hännikäinen, and T. D. Hämäläinen, "Optimal subset mapping and convergence evaluation of mapping algorithms for distributing task graphs on multiprocessor SoC," in *Intl. Symposium on Soc*, Tampere, Finland, Nov. 2007.

[184] J. Owens, W. Dally, R. Ho, D. Jayasimha, S. Keckler, and L.-S. Peh, "Research challenges for on-chip interconnection networks," *IEEE Micro*, vol. 27, no. 5, pp. 96–108, Sep-Oct. 2007.

[185] J. Palma, L. Indrusiak, F. Moraes, A. Ortiz, M.Glesner, and R. Reis, "Evaluating the impact of data encoding techniques on the power consumption in networks-on-chip," in *Emerging VLSI Technologies and Architectures*, Mar. 2006, pp. 2–3.

[186] D. Pamunuwa, J. Öberg, L. R. Zheng, M. Millberg, A. Jantsch, and H. Tenhunen, "Layout, performance and power trade-offs in mesh-based network-on-chip architectures," in *VLSI-SOC*, Dec. 2003, pp. 362–267.

[187] P. P. Pande *et al.*, "Design, synthesis, and test of networks on chips," *IEEE Des. Test Comput*, vol. 22, no. 5, pp. 404–413, Aug. 2005.

[188] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Design of a switch for network on chip applications," in *ISCAS*, 2003, pp. 217–220.

[189] ——, "Switch-based interconnect architecture for future systems on chip," in *SPIE, VLSI Circuits and Systems*, vol. 5117, May 2003, pp. 228–237.

[190] P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *IEEE Trans. Computers*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[191] J. Paul, D. Thomas, and A. Bobrek, "Scenario-oriented design for single-chip heterogeneous multiprocessors," *IEEE Trans. VLSI Syst.*, vol. 14, no. 8, pp. 868–880, Aug. 2006.

[192] V. F. Pavlidis and E. G. Friedman, "3-D topologies for networks-on-chip," *IEEE Trans. VLSI Syst.*, vol. 15, no. 10, pp. 1081–1090, Oct. 2007.

[193] L.-S. Peh and W. J. Dally, "A delay model for router micro-architectures," *IEEE Micro*, pp. 26–34, Jan/Feb. 2001.

[194] S. Penolazzi and A. Jantsch, "A high level power model for the Nostrum NoC," in *Euromicro DSD*, Aug. 2006, pp. 673–676.

[195] A. D. Pimentel, "The Artemis workbench for system-level performance evaluation of embedded systems," *Int. Journal of Embedded Systems*, vol. 1, no. 7, 2005.

[196] A. D. Pimentel, S. Polstra, F. Terpstra, A. van Halderen, J. E. Coffland, and L. O. Hertzberger, "Towards efficient design space exploration of heterogeneous embedded media systems," in *SAMOS*, Jul. 2002, pp. 57–63.

[197] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas, "Calibration of abstract performance models for system-level design space exploration," *Journal of VLSI Sig. Process. Systems for Signal, Image, and Video Technology*, 2007.

[198] F. Poletti, A. Poggiali, D. Antonio, L. Benini, P. Marchal, M. Loghi, and M. Poncino, "Energy-efficient multiprocessor systems-on-chip for embedded computing: Exploring programming models and their architectural support," *IEEE Trans. Comput.*, vol. 56, no. 5, pp. 606–621, May 2007.

[199] A. Pullini, F.Angiolini, S. M. D. Atienza, G. De Micheli, and L. Benini, "Bringing NoCs to 65 nm," *IEEE Micro*, vol. 27, no. 5, pp. 75–85, Sep.-Oct 2007.

[200] R. Rabenseifner, S. R. Tiyyagura, and M. Müller, "Network bandwidth measurements and ratio analysis with the HPC challenge benchmark suite (HPCC)," in *EuroPVM/MPI, LNCS 3666*, Sep. 2005, pp. 368 – 378.

[201] A. Radulescu, J. Dielissen, S. Pestana, O. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens, "An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 4–17, Jan. 2005.

[202] R. M. Ramanathan, "Intel multi-core processors making the move to quad-core and beyond," Intel Corporation, white paper, Oct. 2006.

[203] A. Rasmus, A. Kulmala, E. Salminen, and T. D. Hämäläinen, "IP integration overhead analysis in system-on-chip video encoder," in *DDECS*, Krakow, Poland, Apr. 2007, pp. 333–336.

[204] T. Richardson, C. Nicopoulos, D. Park, V. Narayanan, Y. Xie, C. Das, and V. Degalahal, "A hybrid SoC interconnect with dynamic TDMA-based transaction-less buses and on-chip networks," in *VLSI design*, Jan. 2006.

[205] J. Riihimäki, P. Kukkala, E. Salminen, M. Hännikäinen, K. Kuusilinna, and T. Hämäläinen, "Practical distributed simulation of a network of wireless terminals," in *Intl. Symposium on Soc*, Tampere, Finland, Nov. 2004, pp. 49–52.

[206] J. Riihimäki, E. Salminen, K. Kuusilinna, and T. Hämäläinen, "Parameter optimization tool for enhancing on-chip network performance," in *ISCAS*, May 2002, pp. 61–64.

[207] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielisen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade offs in the design of a router with both guaranteed and best-effort services for network on chip (extended version)," *IEE Proc. Computers and Digital Techniques*, vol. 150, no. 5, pp. 294–302, Sep. 2003.

[208] T. Rintaluoma, O. Silven, and J. Raekallio, "Interface overheads in embedded multimedia software," in *SAMOS VI*, Jul. 2006, pp. 5–14.

[209] P. Ross, "Why CPU frequency stalled," *IEEE Spectrum*, p. 72, Apr. 2008.

[210] D. Rostislav, V. Vishnyakov, E. Friedman, and R. Ginosar, "An asynchronous router for multiple service levels networks on chip," in *ASYNC*, Mar. 2005, pp. 224–229.

[211] K. K. Ryu and V. J. Mooney III, "Automated bus generation for multiprocessor SoC design," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 11, pp. 1531–1549, Nov. 2004.

[212] I. Saastamoinen, M. Alho, and J. Nurmi, "Buffer implementation for Proteo network-on-chip," in *ISCAS*, 2003, pp. 113–116.

[213] M. Saldaña, L. Shannon, and P. Chow, "The routability of multiprocessor network topologies in FPGAs," in *SLIP*, Mar. 2006, pp. 49–56.

[214] E. Salminen, "Interface design for multiple processors in a system-on-chip video encode," Master's thesis, Tampere University of Technology, 2001.

[215] T. Salminen and J.-P. Soininen, "Evaluating application mapping using network simulation," in *Intl. Symposium on Soc*, Nov. 2003, pp. 27–30.

[216] H. Samuelsson and S. Kumar, "Ring road NoC architecture," in *Norchip*, Nov. 2004, pp. 16–19.

[217] A. Sangiovanni-Vincentelli, "Quo vadis, SLD? reasoning about the trends and challenges of system level design," *Proc. IEEE*, vol. 95, no. 3, pp. 467–506, Mar. 2007.

[218] S. Santi, B. Lin, L. Kocarev, G. M. Maggio, R. Rovatti, and G. Setti, "On the impact of traffic statistics on quality of service for networks on chip," in *ISCAS*, May 2005, pp. 2349–2352.

[219] S. Sathe, D. Wiklund, and D. Liu, "Design of a switching node (router) for on-chip networks," in *ASIC*, Oct. 2003, pp. 75–78.

[220] D. Sengupta and R. Saleh, "Generalized power-delay metrics in deep submicron CMOS designs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 1, pp. 183–189, Jan. 2007.

[221] M. Setälä, P. Kukkala, T. Arpinen, M. Hännikäinen, and T. D. Hämäläinen, "Automated distribution of UML 2.0 designed applications to a configurable multiprocessor platform," in *SAMOS VI*, Jul., pp. 27–38.

[222] J.-S. Shen, K.-C. Chang, and T.-F. Chen, "On a design of crossroad switches for low-power on-chip communication architectures," in *ISCAS*, May 2006.

[223] D. Sigüenza-Tortosa, T. Ahonen, and J. Nurmi, "Issues in the development of a practical NoC: the Proteo concept," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 95–105, Oct. 2004.

[224] K. Skadron, M. Martonosi, D. August, M. Hill, D. Lilja, and V. Pai, "Challenges in computer architecture evaluation," *IEEE Computer*, vol. 36, no. 8, pp. 30–36, Aug. 2003.

[225] J.-P. Soininen, A. Jantsch, M. Forsell, A. Pelkonen, J. Kreku, and S. Kumar, "Extending platform-based design to network on chip systems," in *VLSI Design*, Jan. 2003, pp. 401–408.

[226] *Sonics MicroNetworks Technical Overview Revision*, A21-1 ed., Sonics Inc., June 2000.

[227] V. Soteriou, N. Eisley, H. Wang, B. Li, and L.-S. Peh, "Polaris: A system-level roadmapping toolchain for on-chip interconnection networks," *IEEE Trans. VLSI Syst.*, vol. 15, no. 8, pp. 855–868, Aug. 2007.

[228] V. Soteriou, H. Wang, and L.-S. Peh, "A statistical traffic model for on-chip interconnection networks," in *MASCOTS*, Sep. 2006, pp. 104–116.

[229] SPIRIT Schema Working Group, "Spirit-user guide v1.0," [online] http://www.spiritconsortium.org/, Dec. 2004.

[230] Standard Performance Evaluation Corporation, "OpenMP benchmark suite," [online], http://www.spec.org/omp2001/, May 2005.

[231] D. Stroobandt, P. Verplaetse, and J. van Campenhout, "Generating synthetic benchmark circuits for evaluating CAD tools," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 9, pp. 1011–1022, Sep. 2000.

[232] D. Sylvester and K. Keutzer, "Impact of small process geometries on microarchitectures in systems on a chip," *Proc. IEEE*, vol. 89, no. 4, pp. 467–489, Apr. 2001.

[233] R. Thid, M. Millberg, and A. Jantsch, "Evaluating NoC communication backbones with simulation," in *Norchip*, Nov. 2003, pp. 27–30.

[234] R. Thid, I. Sander, and A. Jantsch, "Flexible bus and NoC performance analysis with configurable synthetic workloads," in *Euromicro DSD*, 2006, pp. 681–688.

[235] T. Valtonen, T. Nurmi, J. Isoaho, and H. Tenhunen, "An autonomous error-tolerant cell for scalable network-on-chip architectures," in *Norchip*, Nov. 2001, pp. 198–203.

[236] S. Vangal *et al.*, "An 80-tile Sub-100-W TeraFLOPS processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[237] V. Varadarajan and R. Mittra, "Finite-difference time-domain (FDTD) analysis using distributed computing," *IEEE Microwave and Guided Wave Letters*, vol. 4, no. 5, pp. 144–145, May 1994.

[238] S. Vassiliadis and I. Sourdis, "Reconfigurable fabric interconnects," in *Intl. Symposium on Soc*, Nov. 2006, pp. 41–44.

[239] P. Vellanki, N. Banerjee, and K. S. Chatha, "Quality-of-service and error control techniques for mesh-based network-on-chip architectures," *Integration, the VLSI Journal*, vol. 38, no. 3, pp. 353–382, Jan. 2005.

[240] F. R. Wagner, W. O. Cesário, L. Carro, and A. A. Jerraya, "Strategies for the integration of hardware and software IP components in embedded systems-on-chip," *Integration, the VLSI Journal*, vol. 37, no. 4, pp. 223–252, Sep. 2004.

[241] I. Walter, I. Cidon, R. Ginosar, and A. Kolodny, "Access regulation to hot-modules in wormhole NoCs," in *NOCS*, May 2007, pp. 137–148.

[242] M. Wang, N. H. Chan, S. Papadimitriou, C. Faloutsos, and T. Madhyastha, "Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic," in *International Conference on Data Engineering*, Feb. 2002, pp. 507–516.

[243] N. Wang, A. Sanusi, P. Zhao, S. Mohamed, and M. A. Bayoumi, "PMCNOC: A pipelining multi-channel central caching network-on-chip communication architecture design," in *SiPS*, Oct. 2007, pp. 487–492.

[244] R. Weicker, “An overview of common benchmarks,” *IEEE Computer*, vol. 23, no. 12, pp. 65–75, Dec. 1990.

[245] A. Weiss, “Dhrystone benchmark - history, analysis, "scores" and recommendations,” EEMBC Certification Laboratories, white paper, Nov. 2002.

[246] P. Wielage and K. Goossens, “Networks on silicon: blessing or nightmare?” in *Euromicro DSD*, Sep. 2002, pp. 196–200.

[247] D. Wiklund, “Development and performance evaluation of networks on chip,” Ph.D. dissertation, Linköping university, Apr. 2005.

[248] D. Wiklund and D. Liu, “Switched interconnect for system-on-a-chip designs,” in *IP2000*, Oct. 2000, pp. 198–192.

[249] W. Wolf, A. Jerraya, and G. Martin, “Multiprocessor system-on-chip(MPSoC) technology,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1701–1713, Oct. 2008.

[250] P. Wolkotte, G. Smit, G. Rauwerda, and L. Smit, “An energy-efficient reconfigurable circuit-switched network-on-chip,” in *IPDPS*, Apr. 2005, p. 155a.

[251] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, “The SPLASH-2 programs: characterization and methodological considerations,” in *ISCA*, Jun. 1995, pp. 24–36.

[252] Xin Wang, T. Ahonen, and J. Nurmi, “Applying CDMA technique to network-on-chip,” *IEEE Trans. VLSI Syst.*, vol. 15, no. 10, pp. 1091–1100, Oct. 2007.

[253] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, “Methodology for design, modeling, and analysis of networks-on-chip,” in *ISCAS*, May 2005, pp. 1778–1781.

[254] ——, “A design methodology for application-specific networks-on-chip,” *ACM Trans. Embedded Computing Systems*, vol. 5, no. 2, pp. 262–280, May 2006.

[255] T. Ye, L. Benini, and G. de Micheli, “Analysis of power consumption on switch fabrics in network routers,” in *DAC*, Jun. 2002, pp. 524–529.

[256] C. A. Zeferino, M. E. Kreutz, L. Carro, and A. A. Susin, "Models for communication tradeoffs on systems-on-chip," in *IP based design*, Oct. 2002, pp. 394–400.

[257] C. Zeferino, M. Kreutz, and A. Susin, "RASoC: a router soft-core for networks-on-chip," in *DATE*, vol. 3, Feb. 2004, pp. 198–203.

[258] C. A. Zeferino, M. E. Kreutz, L. Carro, and A. A. Susin, "A study on communication issues for system-on-chip," in *SBCCI*, Sep. 2002, pp. 121–126.

[259] H. Zhang, M. Wan, V. George, and J. Rabaey, "Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs," in *Workshop on VLSI*, Apr. 1999, pp. 2–8.