THESIS ON INFORMATICS AND SYSTEM ENGINEERING C72

System-Level Design of Timing-Sensitive Network-on-Chip Based Dependable Systems

MIHKEL TAGEL

TUT PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Engineering

Dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer and System Engineering on March 28, 2012.

Supervisor: Dr. Gert Jervan
Department of Computer Engineering
Tallinn University of Technology, Estonia

Opponents: Prof. Zebo Peng
Department of Computer and Information Science
Linköping University, Sweden

Dr. Leandro Soares Indrusiak
Department of Computer Science
University of York, United Kingdom

Defence of the thesis: May 14, 2012

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for any academic degree.

/Mihkel Tagel/

ISSN 1406-4731
ISBN 978-9949-23-263-5 (publication)
ISBN 978-9949-23-264-2 (PDF)

INFORMAATIKA JA SÜSTEEMITEHNIKA C72

Kiipvõrkudel põhinevate ajakriitiliste ja töökindlate süsteemide kõrgtaseme disain

MIHKEL TAGEL

TTÜ
KIRJASTUS

To my family

Abstract

Technology scaling into sub-nanometer range will have an impact on system-on-chip (SoC) manufacturing yield and quality. Smaller feature sizes allow more and more functionality to be packed to the same chip area – thus, the systems are getting increasingly complex. An example of a modern system-on-chip is a mobile phone that has to support different basebands, network protocols and multimedia formats. At the same time not only is increasing the computation requirements but also the amount of communication. Limited throughput and bus lengths have impact to the system design and to the number of modules. As technologies advance, the high degree of sensitivity to defects makes a SoC designer goal to design a fault-free system a very difficult task. A SoC designer has to assume that the manufactured devices might contain faults and an application, running on the system, must be aware that the underlying hardware is not perfect. Networks-on-chip (NoC) have been proposed as one of the alternatives to solve the on-chip communication scalability problems and to address dependability at various levels of abstraction. In a traditional SoC the components are interconnected via a central bus having essentially point-to-point connections (circuit switching). In networks-on-chip packet switching is being utilized. A data to be transmitted is divided into smaller blocks called packets. Depending on the NoC topology, routing algorithm and location of the components in the network, a packet could be delivered to the receiver via multiple network nodes. Therefore, communication modelling and synthesis plays an important role in the design of complex NoC-based timing-sensitive systems-on-chip. Trying to guarantee the observance of timing constraints without detailed know-how of communication transactions might lead to unexpected results.

This thesis concentrates on system-level design issues of NoC-based real-time systems. We present a method for communication synthesis and scheduling to take into account possible network resource conflicts and to calculate communication delays. Two global optimization techniques to improve the initial schedule and/or task mapping are described. Two extensions of the proposed communication model to synthesize communication at different granularity levels and to introduce communication interleaving modelling support are presented. Finally, we extend our work to handle a given number of transient and intermittent faults during task execution or data transmission on communication links. It is done by allocating slack time and scheduling the application with modified shifting-based scheduling algorithm.

Kokkuvõte

Pooljuhtide tootmistehnoloogiate arengul on otsene mõju kiipsüsteemide projekteerimisele. Transistoride mõõtmete vähenemine võimaldab mahutada sama suurele kiibipinnale üha rohkem funktsionaalsust. Kiipsüsteemide keerukust ilmestab näide, kus kaasaegne mobiiltelefon ei ole ainult helistamiseks, vaid peab toetama erinevaid võrgusagedusi, võrguprotokolle, multimeediaformaate jne. Seejuures ei ole kasvanud mitte ainult töötlemisressursi vajadus, vaid ka komponentide vahelise andmevahetuse hulk. Limiteeritud läbilaskevõime ja siinipikkused seavad piiri süsteemide kasvule ja täiendavate komponentide lisamisele. Teisalt võivad nanostruktuuridel põhinevaid kiipsüsteeme hakata häirima uut tüüpi vead ja rikked. Kiipsüsteemide projekteerija ei saa eeldada, et riistvara töötab veatult. Üheks võimalikuks suurte kiipsüsteemide tootmise, (taas)kasutatavuse ning arhitektuursete probleemide lahenduseks on kiipvõrkude ideoloogia. Traditsioonilises kiipsüsteemis on komponendid ühendatud keskse siini külge ning andmevahetuseks luuakse kahe komponendi vahel n-ö fikseeritud kanal (kanalkommutatsioon). Teised komponendid samal ajal andmeid vahetada ei saa. Kiipvõrkudes on aga kasutusel arvutivõrkudest tuntud pakettkommutatsioon. Sõnum jagatakse kindla suurusega pakettideks. Sõltuvalt kiipvõrgu topoloogiast, marsruutimisalgoritmist ning komponentide paiknemisest kiipvõrgus võib paketi saatmine toimuda läbi mitme erineva võrgusõlme (marsruuteri). Kommunikatsiooni modelleerimisel ja sünteesil on oluline roll kiipvõrkudel põhinevate kiipsüsteemide disainis. Ilma detailse arusaamata kiipidevahelisest kommunikatsioonist on raske hinnata süsteemide ajalist käitumist ning garanteerida nende vastavust nõuetele.

Käesolev doktoritöö keskendub süsteemitaseme disaini probleemidele kiipvõrkudel põhinevates ajakriitilistes kiipsüsteemides. Oleme välja pakkunud kommunikatsiooni modelleerimise ja sünteesi meetodi, mis võimaldab leida andmeülekandeks kuluva aja, võttes seejuures arvesse võimalikke võrgukonflikte. Kirjeldame pakutud kommunikatsiooni modelleerimise meetodi kombineerimist erinevate globaalsete optimeerimisalgoritmidega, eesmärgiga leida efektiivsem ülesannete planeering ja/või jaotus kiipsüsteemi protsessoritel. Pakume välja kaks kommunikatsioonimudeli täiendust, mis võimaldavad efektiivsemalt kasutada kiipvõrgu ressursse ja analüüsime mudeli keerukuse kasvu. Doktoritöö viimases osas kirjeldame meetodit, mis võimaldab rakendusel tolereerida etteantud arvu vigasid. Keskendume lühiajalistele vigadele, mis ei ole tootmisvead ning mis võivad esineda nii ülesannete täitmisel kui ka kommunikatsioonis komponentide vahel.

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Gert Jervan for guiding me through the Master's course to the end of the PhD. During this time I have gained a lot of invaluable experience. I would like to thank also Prof. Peeter Ellervee for helping us to set up the base framework for my PhD studies and being the co-supervisor. Special thanks go to Prof. Thomas Hollstein for giving valuable feedback for my PhD work, pointing out the areas that could be improved. Gert, Peeter and Thomas – thank you for proof reading this PhD thesis!

Moreover, I would like to acknowledge the organizations that have supported my PhD studies: Tallinn University of Technology (Department of Computer Engineering), National Graduate School in Information and Communication Technologies (IKTDK), Centre of Research Excellence in Dependable Embedded Systems (CREDES), European Regional Development Fund through the Centre for Integrated Electronic Systems and Biomedical Engineering (CEBE) and Estonian IT Foundation (EITSA).

Finally, I would like to thank my family for the patience and support – especially my wife Liis.

Mihkel Tagel

Tallinn, April 2012

List of Publications

Journals

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2012). Contention-aware scheduling for NoC based systems. *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)* [submitted for review].

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2011). System-level optimization of NoC-based timing sensitive systems. *Estonian Journal of Engineering*, 17(2), 158 - 168.

Tagel, M., Ellervee, P., & Jervan, G. (2010). System-Level Communication Synthesis and Dependability Improvements for Network-on-Chip Based Systems. *Estonian Journal of Engineering*, 16(1), 23 - 38.

Conferences

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2011). Contention-aware scheduling for NoC based systems. *Proceedings of the 29th Norchip conference*, (pp. 1-4). Lund, Sweden.

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2011). Communication modelling and synthesis for NoC-based systems with real-time constraints. *Proceedings of the 14th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems* (pp. 237 - 242). Cottbus, Germany.

Tagel, M., Ellervee, P., & Jervan, G. (2010). Design Space Exploration and Optimisation for NoC-based Timing Sensitive Systems. *Proceedings of the 12th Biennial Baltic Electronic Conference* (pp. 177 - 180). Tallinn, Estonia.

Tagel, M., Ellervee, P., & Jervan, G. (2009). Scheduling Framework for Real-time Dependable NoC-Based Systems. *Proceedings of the International Symposium on System-on-Chip* (pp. 95 - 99). Tampere, Finland.

Book chapter

Tagel, M., Ellervee, P., & Jervan, G. (2011). System-Level Design of NoC-Based Dependable Embedded Systems. Ubar, R.; Raik, J.; Vierhaus, H. T. (Eds.). In *Fault-Tolerance and Applications in System-on-Chip Design: Advancements and Techniques* (pp. 1 - 36). Hershey, Pennsylvania, USA: IGI Global.

List of Abbreviations

API	Application Programming Interface
ATE	Automated Test Equipment
B&B	Branch-and-Bound
BSC	Boundary Shift Code
CMOS	Complementary Metal-Oxide Semiconductor
CRC	Cyclic Redundancy Check
DAG	Directed Acyclic Graph
DAP	Duplicate Add Parity
DLS	Dynamic Level Scheduling
DSP	Digital Signal Processor
ECI	Error Capturing Instruction
EDA	Electronic Design Automation
ETF	Earliest Time First
ETG	Extended Task Graph
FDAR	Fault Diagnosis-And-Repair
FIFO	First-In First-Out
FT-CPG	Fault-Tolerant Conditional Process Graph
GA	Genetic Algorithm
GALS	Globally Asynchronous Locally Synchronous
GM	Group Migration
HM	Hamming Distance
HTML	HyperText Markup Language
IC	Integrated Circuit
IDMA	Locally organized packet identity (ID) division Multiple Access
IP	Intellectual Property
LS	List Schedule

MDR	Modified Dual Rail
MPSoC	Multiprocessor System-on-Chip
MTBF	Mean Time Between Failures
MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
NI	Network Interface
NMR	N-Modular Redundancy
NOC	Network-on-Chip
OCP-IP	Open Core Protocol International Partnership
OSM	Optimal Subset Mapping
QAP	Quadratic Assignment Problem
QOS	Quality of Service
RAM	Random Access Memory
RMU	Recovery Management Unit
RNI	Resource Network Interface
SA	Simulated Annealing
SBS	Shifting-based Scheduling
SER	Soft Error Rate
SOC	System-on-Chip
SPIN	Scalable, Programmable, Integrated Network
TDMA	Time Division Multiple Access
TDN	Temporally Disjoint Networks
TMR	Triple Modular Redundancy
VLSI	Very Large Scale Integration
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WCET	Worst Case Execution Time
XGFT	Fault Tolerant Extended Generalized Fat Tree
XHiNoC	Extendable Hierarchical Network-on-Chip
XML	Extensible Markup Language

List of Figures

Table of Contents

Chapter 1. Introduction

As technologies advance and semiconductor process dimensions shrink into the nanometer and sub-nanometer range, the high degree of sensitivity to defects begins to impact the overall yield and quality. The International Technology Roadmap for Semiconductors (2009) states that relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification, and test. Such a paradigm shift is likely forced by technology scaling that leads to more transient and permanent failures of signals, logic values, devices, and interconnects. In consumer electronics, where the reliability has not been a major concern so far, the design process has to be changed. Otherwise, there is a high loss in terms of faulty devices due to problems stemming from the nanometer and sub-nanometer manufacturing process. There has been a lot of research made on system reliability in different computing domains by employing data encoding, duplicating system components or software-based fault tolerance techniques. This research has mostly had either focus on low level hardware reliability or covered the distributed systems. Due to future design complexities and technology scaling, it is infeasible to concentrate only onto low level reliability analysis and improvement. We should fill the gap by looking at the application level. We have to assume that the manufactured devices might contain faults and an application, running on the system, must be aware that the underlying hardware is not perfect.

The advances in design methods and tools have enabled integration of increasing number of components on a chip. Design space exploration of such many-core systems-on-chip (SoC) has been extensively studied, whereas the main focus has been so far on the computational aspect. With the increasing number of on-chip components and further advances in semiconductor technologies, the communication complexity increases and there is a need for alternatives to the traditional bus-based or point-to-point communication architectures. Network-on-chip (NoC) is one of the possibilities to overcome some of the on-chip communication problems. In such NoC-based systems, the communication is achieved by routing packets through the network infrastructure rather than routing global wires. However, communication parameters (inter-task communication volume, link latency and bandwidth, buffer size) might have major impact to the performance of applications implemented on NoCs. Therefore, in order to guarantee predictable behaviour and to satisfy performance constraints, a careful selection of application partitioning, mapping and synthesis algorithms is required. NoC platform provides also additional flexibility to tolerate faults and to guarantee system

reliability. Many authors have addressed these problems but most of the emphasis has been on the systems based on bus-based or point-to-point communication (Marculescu, Ogras, Li-Shiuan Peh Jerger, & Hoskote, 2009). However, a complete system-level design flow, taking into account the NoC network modelling and dependability issues, is still missing.

Contributions of the thesis

This thesis concentrates on system-level design issues of network-on-chip based systems. It describes various methods for NoC architecture analysis and optimization, and gives an overview of different fault-tolerance methods. The main emphasis of the thesis is on the communication modelling and accurate communication scheduling taking into account possible network contentions.

The main contributions of this thesis are:

- Contention-aware scheduling method that takes into account network induced latencies and handles network resource conflicts (Tagel, Ellervee, Hollstein, & Jervan, 2011a; Tagel, Ellervee, Hollstein, & Jervan, 2011b). Communication modelling, synthesis and scheduling are all part of our system-level design flow. The produced schedules can be verified by executing the applications on a cycle-accurate NoC simulator.
- Two extensions of the proposed communication model to synthesize communication at different granularity levels (Tagel, Ellervee, Hollstein, & Jervan, 2012) and introduce communication interleaving modelling support. Both methods have focus on more effective scheduling of communication to increase the network utilization and to reduce the schedule length.
- Design optimization by branch-and-bound and simulated annealing techniques to improve the initial schedule and/or task mapping (Tagel, Ellervee, & Jervan, 2010).
- A system-level technique to tolerate transient and intermittent faults in tasks and communications. The fault-tolerance requirements are taken into account during the task and communication scheduling to allocate appropriate slack time to execute recovery actions (Tagel, Ellervee, & Jervan, 2011).

Organization of the thesis

This thesis is organized into six chapters. After the introductory chapter, the Chapter 2 analyses the problems related to the development of dependable systems-on-chip. It outlines challenges, specifies problems and examines the

work that has been done in different NoC research areas relevant to this thesis. It gives an overview of the state-of-the-art in system-level design of traditional and NoC-based systems and describes briefly various methods proposed for system-level architecture analysis and optimization, such as application mapping, scheduling, communication analysis and synthesis. The chapter gives also an overview of different fault-tolerance techniques that have been successfully applied to bus-based systems. It analyses their shortcomings and applicability to the NoC-based systems.

The Chapter 3 describes our system-level design framework. The basic definitions and problem formulation are given in this chapter. The system and network-on-chip architecture assumptions we have made in this thesis are described. We propose contention-aware scheduling algorithm for NoC-based systems with real-time constraints. The proposed approach captures both the end-point and the network contentions. It can be used for different topologies and different switching methods together with deterministic routing algorithms. The produced schedules are verified by executing the applications on cycle accurate XHiNoC simulator.

The Chapter 4 presents two extensions of our communication model described in Chapter 3. In the first part we explore the improvements in schedule length versus modelling complexity achieved by using packet-based communication synthesis and scheduling. In the second part of the chapter an extension to our communication model to support communication interleaving is described. We have a configurable number of separate flit queues that share equally the available bandwidth. During the scheduling we are trying to load-balance the queues and to mimic the operation of the XHiNoC routers. The goal is to utilize more effectively the available network resources and to reduce the schedule length.

The Chapter 5 presents branch-and-bound and simulated annealing optimization techniques in order to improve the initial schedule length or task mapping. It explores the trade-off between the amount of improvement gained versus time spent for calculation.

In the last chapter we describe a system-level technique to tolerate transient and intermittent faults in tasks and communication. The work is based on the shifting-based scheduling that we have extended to contain on top of the task dependability requirements also communication fault-tolerance requirements and integrated it with our contention-aware scheduling.

Chapter 2. Background and Related Work

Innovations in the chip design and in related domains have been motivated by technology scaling. One of the well-known formulations of the rapid technology scaling is Moore's law that states that the number of transistors on an integrated circuit (IC) will double approximately in every two years (Moore, 1975). The increase in number of logic gates that can be implemented on a single chip allows designing of circuits with broad functionality. Previously, a system that was implemented out of many discrete components on a board can now be integrated on a single chip. Such increased system complexity has motivated the shift from custom chip design to higher integration levels, standardized interfaces, intellectual property (IP) re-use, hardware/software co-design and fast prototyping. An IC that integrates all components of an electronic system on a single chip is called a system-on-chip (SoC). Figure 2.1 depicts an example of a typical bus-based SoC. It consists of one or more programmable cores, on-chip memory, signal processor and peripherals, all interconnected by an on-chip bus.

Consumer electronics is one of the driving forces of the system-on-chip design. While the time-to-market window is continuously getting shorter consumers have higher expectations on the functionality of devices. The devices have to support a wide range of multimedia codecs and connectivity standards. This has been made possible thanks to the re-configurability of system components. An example is software-defined radio (SDR). SDR is a

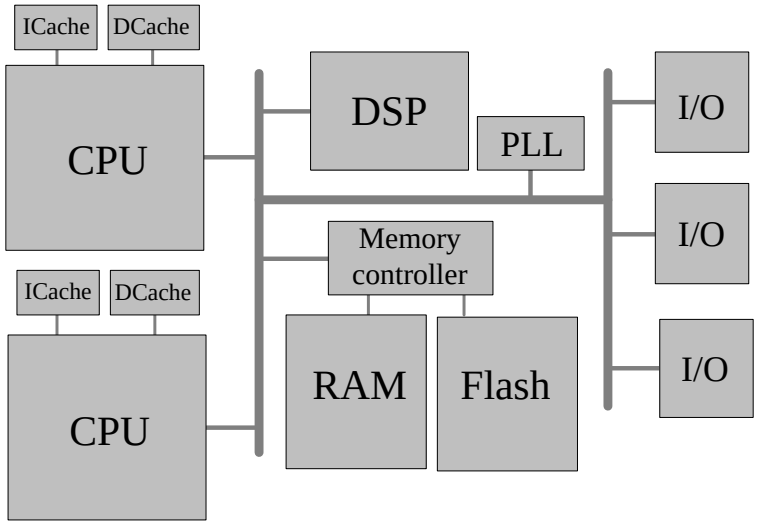


Figure 2.1. An example of a typical bus-based system-on-chip

communication system where typical fixed hardware components are implemented in software or re-configurable hardware. Such system can adaptively accommodate to certain operating situations. Another example of diverse requirements that a modern system-on-chip has to handle is high definition multimedia. Besides different encoding and decoding schemes, requiring more and more processing power, the on-chip interconnect has to handle increased bandwidth requirements with manageable communication latency.

In this chapter we first describe the design challenges that have emerged together with technology scaling and due to the increase of the design complexity. We give an overview of the key concepts and terminology of NoC-based systems-on-chip. The second part of this chapter is devoted to system-level design and dependability issues.

2.1. System-on-chip design challenges

The advances in design methods and tools have enabled integration of ever increasing number of components on the chip. Design space exploration of such many-core SoCs has been extensively studied, whereas the main focus has been so far on the computational aspects. With the increasing number of on-chip components and further advances in semiconductor technologies, the communication complexity increases and there is a need for alternatives to the traditional bus-based or point-to-point communication architectures. The main challenges that motivate the paradigm shift in the current SoC design methodologies are described below.

Power and thermal management

The power dissipation is one of the key issues in present day VLSI CMOS technology that limit the circuit performance (Sahoo, Datta, & Kar, 2011). In the previous decade the dynamic power dissipation minimization has mainly targeted the logic gates. However, technology scaling over the years has changed the situation – wires do not scale as rapidly as transistors. Interconnect wires account for a significant fraction (up to 50%) of the energy consumed in an integrated circuit and is expected to grow in the future (Raghunathan, Srivastava, & Gupta, 2003; Sahoo, Datta, & Kar, 2011). Feature size scaling increases power density on the chip die that in turn can produce an increase in the chip temperature. The rapidly increasing proportion of the consumer electronics market, represented by handheld and battery-powered equipment, also means that low power consumption has become a critical design requirement that must be addressed (Claasen, 2006).

Memory bandwidth and latency

Computational speeds will increase at a higher rate compared to the time to access off-chip memory. According to Chen & Kung (2008) this increasing gap between processor and memory speeds is a well-known problem, named the memory wall. In order to feed the computational engine, SoC designers have to take action, such as, integrating embedded memory into the same chip, or exploiting data access localities at the algorithm or software level.

Deep submicron effects and variability

Scaling of feature sizes in semiconductor industry has given the ability to increase performance while lowering the power consumption. However, with feature sizes below 40 nm it is getting hard to achieve a favourable trade-off between cost and performance (power consumption) of the designed devices (International Technology Roadmap for Semiconductors, 2009; Konstadinidis, 2009). The emergence of deep submicron noise in the form of cross-talk, leakage, supply noise, as well as process variations is making it increasingly hard to achieve the desired level of reliability while maintaining the constant improvement in performance and energy-efficiency (Shanbhag, Soumyanath, & Martin, 2000; Kahng, 2007). Interconnects also add a new dimension to the design complexity. As interconnects shrink and come closer together, previously negligible physical effects like crosstalk become significant (Hamilton, 1999; Ho, Mai, & Horowitz, 2001). To produce reliable products in the presence of high variability, reliability should be looked from the perspective of multiple disciplines, e.g. fabrication, circuit design, logic design and software (Chen & Kung, 2008).

Global synchrony

SoCs are traditionally based on a bus architecture where system modules exchange data via a synchronous central bus. An example is Advanced Microcontroller Bus Architecture (AMBA) bus. The AMBA bus was introduced by ARM in 1996 and is widely used as the on-chip bus in SoC designs (Shrivastav, Tomar, & Singh, 2011). When the number of components increases rapidly, we have a situation where the clock signal cannot be distributed over the entire SoC during one clock cycle. Ho et al. (2001) describe that while local wires scale in performance, global and fixed-length wires do not. Moreover, relative wire lengths are increasing compared to the chip area and the size of transistors. Optimization techniques, such as optimal wire sizing, buffer insertion, and simultaneous device and buffer sizing are solving only some of the problems. On-chip interconnect is becoming a complex circuitry in its own (Hamilton, 1999). Consequently, there is a need for an alternative way for providing scalable and efficient interconnects. *Globally asynchronous locally synchronous* (GALS) design approach has been proposed as a feasible solution for communication intensive complex SoCs. In 2000, Agarwal, Hrishikesh, Keckler, & Burger have examined the effects of technology scaling on wire

delays and clock speeds, and measured the expected performance of a modern microprocessor core in CMOS technologies down to 35 nm. Their estimation showed that even under the best conditions the latency across the chip in a top-level metal wire will be 12-32 cycles (depending on the clock rate). Jason Cong's simulations at the 70 nm level suggest that delays on local interconnect will decrease by more than 50 percent, whereas delays on non-optimized global interconnect will increase by 150 percent (from 2 ns to 3.5 ns) (Hamilton, 1999). A GALS system contains several independent synchronous blocks that operate using their own local clocks and communicate asynchronously with each other. The main feature of these systems is the absence of a global timing reference and the use of several distinct local clocks (or clock domains), possibly running at different frequencies (Iyer & Marculescu, 2002).

Productivity gap

Chip design has become so complex that designers need more education, and experience in a broad range of fields (device physics, wafer processing, analogue effects, digital systems) to understand how all these aspects come together. For the same reasons, designers need smarter tools that comprehend distributed effects like crosstalk (Hamilton, 1999). The complexity and cost of design and verification of multi-core products has rapidly increased to the point where developers devote thousands of engineer-years to a single design and still the products reach the market with hundreds of bugs (Allan, Edenfeld, Joyner, Kahng, Rodgers, & Zorian, 2002). If we look to consumer-electronics then the primary focus in CMOS process development has been on integration density. By having a greater functionality in a smaller area of silicon, the higher integration density and lower costs can be achieved. According to Claasen (2006) for consumer applications Moore's law may continue for as long as the cost per function decreases from node to node. To bridge the technology and productivity gap, the computation needs to be decoupled from the communication. The communication platform should be scalable and predictable in terms of performance and electrical properties. It should enable high intellectual property core reuse by using standard interfaces to connect IP-s to on-chip interconnect.

Verification and design for test

The increasing complexity of SoCs and the different set of tests required by deep submicron process technologies have increased test data volume and test time to the extent that many SoCs no longer fit comfortably within the capabilities of automated test equipment (ATE) (Claasen, 2006). As a result, the cost of test has been rapidly increasing. Due to process variability, the reliability of devices is not anymore only a concern of safety-critical applications but also a concern in consumer electronics. The products need to be designed to tolerate certain number of manufacturing (permanent) and transient faults.

2.2. Network-on-chip as a new design paradigm

To overcome some of the system-on-chip design challenges described previously the network-on-chip paradigm has been proposed. While computer networking techniques are well known already for many decades, the paradigm shift reached to the on-chips in the beginning of this millennium. The interconnection network is a shared resource that a designer can utilize. To design an on-chip communication infrastructure and to meet the performance requirements of an application, a designer has certain design alternatives that are governed by topology, switching, routing and flow control of the network. NoC provides communication infrastructure for resources. Resources can be heterogeneous. As depicted in Figure 2.2 a resource can be memory, processor core, DSP, re-configurable block or any IP block that conforms to the network interface (NI). Every resource is connected to a switch via resource network interface (RNI). Instead of dedicated point-to-point channels between two IP cores, the interconnection network is implemented as a set of shared routers and communication links between the routers. The way the routers are connected with each other defines the network topology. Data to be transferred between communicating nodes is called a message. As messages can have varying sizes it is infeasible to design routers to handle unbounded amounts of data. Instead, messages are divided into smaller bounded flow control units (packets, flits). The way a message is split and transferred through the routers is called switching. Usually there are alternative paths to deliver a message from source to destination. An algorithm to choose between such paths is called routing. A good routing algorithm finds usually minimal paths while avoiding deadlocks. Another alternative would be to balance the network load. Flow control handles network resource accesses. If a network is not able to handle the current

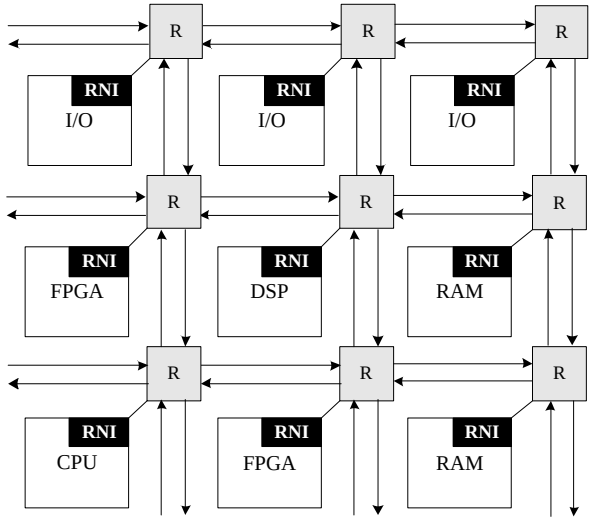


Figure 2.2. An example of a NoC-based system-on-chip

communication load the flow control might forward more critical messages while dropping or re-routing the non-critical ones. An effective network design maximises the throughput and decreases network latency and communication conflicts (Dally & Towles, 2004).

There have been several independent research groups introducing the computer networking ideas to systems-on-chip. In 2000, Guerrier and Greiner proposed a scalable, programmable, integrated network (SPIN) for packet-switched system-on-chip interconnections. They have been using fat-tree topology and wormhole switching with two one-way 32-bit data paths having credit-based flow control. They proposed a router design with dedicated input buffers and shared output buffers, estimated the router cost and network performance. The term "network-on-chip" was first used by Hemani et al. in 2000. The authors introduced the concept of re-configurable network of resources and its associated methodology as solution to the design productivity problem. In 2001, Dally and Towles proposed NoC as a general-purpose on-chip interconnection network to connect IP cores, replacing design-specific global on-chip wires. It was demonstrated that using a network to replace global wires has advantages in structure, performance and modularity. In the paper of Sgroi et al. in 2001 the authors state that the GigaScale Research Center (GSRC) suggests a layered approach similar to that defined for communication networks to address the problem of connecting a large number of IP cores. Additionally the authors described the need for a set of new generation methodologies and tools. In 2001, researchers from Philips Research presented quality of service (QoS) router architecture supporting both best-effort and guaranteed-throughput (Rijpkema, Goossens, & Wielage, 2001). In 2002, Benini and De Micheli formulated NoC as a new SoC design paradigm.

The NoC design paradigm has two good properties to handle the SoC design complexity when compared to bus-based systems – predictability and reusability. The communication throughput, design and verification time are easier to predict due to the regular structure of the NoC. One could connect to the network any IP component that has the appropriate network interface. The NoC paradigm does not set any limits to the number of components. The components and also the communication platform are reusable – the designer needs to design, optimise and verify them once. The layered network architecture provides the needed communication and network services enabling the functionality reuse (Jantsch & Tenhunen, 2003). During the years many NoC research platforms have been developed such as SPIN (Guerrier & Greiner, 2000), Nostrum (Kumar, et al., 2002), Proteo (Sigüenza-Tortosa & Nurmi, 2002), CHAIN (Felicijan, Bainbridge, & Furber, 2003), HERMES (Moraes, Calazans, Mello, Möller, & Ost, 2004), Xpipes (Bertozzi & Benini, 2004), Aethereal (Goossens, Dielissen, & Radulescu, 2005), MANGO (Bjerregaard & Sparso, 2005) and XHiNoC (Samman, Hollstein, & Glesner, 2008). Commercial NoC platforms include Arteris (Arteris, 2009), STNoC (STMicroelectronics, 2009), Silistix (Silistix, 2009) and Sonics (Sonics, 2009).

Current and future directions of on-chip networks include 3D NoCs (Banerjee, Souri, Kapur, & Saraswat, 2001; Feero & Pande, 2007; Pavlidis & Friedman, 2007; Murali, Seiculescu, Benini, & De Micheli, 2009) and optical interconnects (Haurylau et al., 2006). Both emerged in the end of 90's in various forms.

Comparison with bus-based systems and macro networks

Point-to-point connections (circuit switching), common to SoCs, are replaced in NoCs by dividing messages into packets (packet switching). Each component stores its state and exchanges data autonomously with others. Such systems are by their nature GALS systems, containing several independent synchronous blocks that operate with their own local clocks and communicate asynchronously with each other (Iyer & Marculescu, 2002). Having multiple different network routes available for the data transmission makes NoCs to be adaptive – to balance the network load, for instance. The communication platform limitations, data throughput, reliability and QoS are more difficult to address in NoC architectures than in computer networks. The NoC components (memory, resources) are relatively more expensive, whereas the number of point-to-point links is larger on-chip than off-chip. Typically, most of the macro-networks are designed to achieve the highest performance possible while the NoC designs have usually strict power budget constraints. On-chip wires are also relatively shorter than the off-chip ones, thus allowing a much tighter synchronization than off-chip. Additionally, while the macro-networks can handle broad number of applications the NoC-based system is typically designed for a specific domain. A NoC designer has usually more information about the system behaviour and the traffic patterns than a network engineer. On the one hand, only a minimum design overhead is allowed that is needed to guarantee the reliable data transfer. On the other hand, the on-chip network must handle the data ordering and flow control issues (Radulescu & Goossens, 2002). The packets might appear at the destination resource out of order – they need to be buffered and put into the correct order.

While the NoC design has several similarities with macro-networks, due to the aforementioned differences the methods cannot be directly applied to the NoC context.

2.3. Principles of Networks-on-Chip

To design an on-chip communication infrastructure and to meet the performance requirements of an application designer has certain design alternatives that are governed by topology, switching, routing and flow control of the network. In this section we provide an overview of the key concepts and terminology of NoCs.

2.3.1 Topology

Topology refers to the physical structure of the network (how resources and switches are connected to each other). It defines connectivity and routing possibilities between the nodes affecting therefore performance of the network and the design of the routers. Topologies can be classified into groups based on different criteria. One classification that can be found in the literature is direct and indirect topologies. In the direct network topology each processing core can exchange messages with any other processing core via the routers. Examples of direct topologies are meshes and tori. In the indirect network topology packets are switched indirectly via a series of intermediate switch stages until they reach the destination. Examples of indirect topologies are butterflies and fat trees.

Another classification of topologies is by their regularity – regular, application specific and hybrid (hierarchical). A regular topology is not the most efficient in terms of silicon area but allows easier routing algorithms and has better predictability. However, if some of the processing cores are not homogeneous the regular topology might be transformed into the application specific topology. The regularity aims for design reuse and scalability while application specific topologies target performance and power consumption. Hierarchical networks can be used to improve some of the NoC characteristics such as increase bandwidth, reduce average distance (hop count) and network latency. However, this comes at the increased cost and complexity of the network. To compare the various topologies different metrics have been proposed in the literature:

- **Degree** – number of communication links at each router node.
- **Hop count** – the number of hops or traversed links a message takes from source to destination. The maximum hop count represents the *diameter of the network*. The average minimum hop count represents the average hop count over all possible source-destination pairs in the network.
- **Maximum channel load** – represents the maximum number of bits per second that can be injected into the network before it saturates (maximum bandwidth the network can support).
- **Path diversity** – is described by the number of possible shortest paths from source to destination. A topology that provides multiple shortest paths between a source-destination pair has greater path diversity than a topology where there is only a single path available between source-destination. Path diversity relates to the terms of fault tolerance and load balancing.
- **Cost** – total number of physical links and routers in the network.

In this thesis we are using 2D-mesh topology due to its regularity, scalability and popularity in NoC research field: although, our approach is not limited to the

2D-mesh topology. Therefore in the next pages we will provide in addition to mesh-based topologies also an overview of alternative topologies.

Mesh-based topologies

According to survey of Salminen, Kulmala, & Hämäläinen (2008) 59% of NoCs implement mesh and torus topologies. Mesh and torus networks can be described as *k*-ary *n*-cubes, where *k* is the number of nodes in each dimension and *n* is the number of dimensions. They span from a range of networks from rings (*n = 1*) to binary *n*-cubes (*k = 2*), also known as hypercubes (Dally & Towles, 2001). Figure 2.3 depicts three examples of different mesh topologies. A router in 2-cube (2D) mesh contains five ports – east, north, west, south and local port. The local port is connected to the processing core. The other ports are connected with adjacent routers. For a 2D-mesh with *k* number of nodes in each dimension *n*, the total amount of nodes is $T_{nodes} = k^n$. In mesh topology the routers along the edge of the network have the network degree 2 while the rest of the routers have 4 as the network degree. It means that there is an imbalance of the bandwidth across network channels. The maximum hop count (diameter of the network) to the longest neighbour in 2-cube mesh is $H_{max}^{mesh} = 2(k - 1)$. An example on-chip multiprocessor system that uses mesh topology is Intel-Teraflops system (Vangal, et al., 2008). The 80 homogeneous computing elements are interconnected through NoC routers in the 8 × 10 2D-mesh network topology. The main differences between the mesh and torus topologies are the additional communication links connecting a node at the edge of the network with another node at the opposite edge in the same vertical or horizontal paths as depicted in the Figure 2.3b. The network degree of torus is 4 having *2n* links in each dimension. The maximum hop count of 2-cube torus $H_{max}^{torus} = (k-1)$ is two times smaller compared to 2-cube mesh.

Recent research in this area is devoted to 3-dimensional NoCs. Each router in 2D NoC is connected to a neighbouring router in one of four directions. Consequently, each router has five ports. Alternatively, in 3D-mesh NoC, the router typically connects to two additional neighbouring routers located on the adjacent physical planes (Pavlidis & Friedman, 2007).

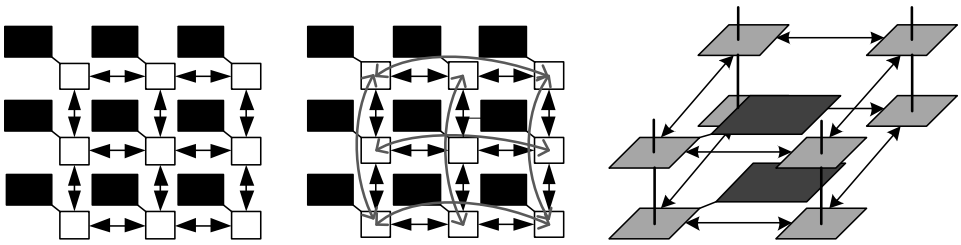


Figure 2.3. Regular topologies. Examples are (a) 4-ary 2-cube mesh, (b) 4-ary 2-cube torus, (c) 3D-mesh

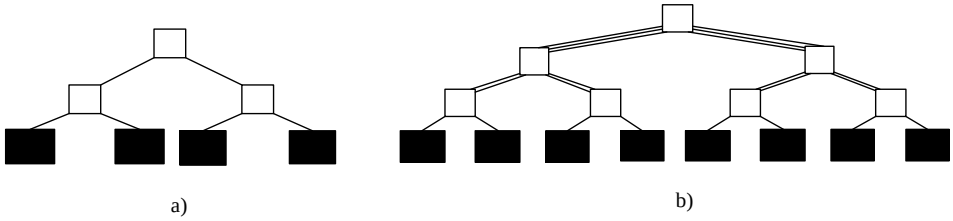


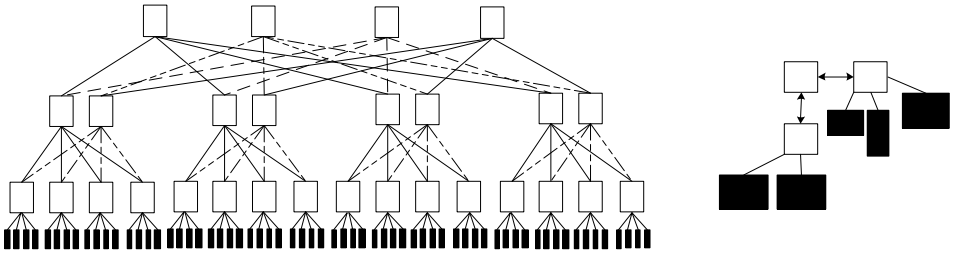
Figure 2.4. Tree-based topologies. Examples are a) binary tree b) fat-tree

Tree-based topologies

The *k*-ary tree and *k*-ary *n*-dimensional fat tree proposed by Adriahantenaina, Charlery, Greiner, Mortiez, and Zeferino (2003) are two alternative regular NoC topologies. In the 2-ary (binary) tree network, depicted in Figure 2.4a, every router is connected to one up-level router and two down-level routers. At the end of the binary-tree network, two computing elements can be connected to each router. Packet routing in a binary tree network is very simple. A direct routing method, reading the binary address from the packet header, can be used to route packets from the source to the destination node. Trees can be laid out in 2D with no wire crossing. One problem with trees is that links closer to the root carry more traffic than those at the lower levels. Solution to this problem is the fat-tree topology. Fat-tree, depicted in Figure 2.4b, is a topology based on a complete binary tree. A set of processing cores is located at the leaves of the fat-tree. The internal nodes of the tree contain routers. The capacity of channels increases as we go up the tree (Leiserson, 1985).

Hybrid hierarchical topologies

A hybrid hierarchical network combines two or more sub-network topologies into one global interconnect architecture. Hierarchical networks are used to improve some of the NoC characteristics such as to increase bandwidth, reduce average distance and network latency. However, this increases complexity of the network – hierarchical topologies may require more complex routing algorithms and have increased wiring requirements. An example of a hierarchical topology is the fat-tree topology explained previously under tree-based topologies. Another example is the butterfly fat-tree where the number of switches converges to a constant depending on the number of levels (Pande, Grecu, Ivanov, & Saleh, 2003). An example of the butterfly fat-tree is depicted in Figure 2.5a. The Proteo NoC (Sigüenza-Tortosa & Nurmi, 2002) uses a hierarchical network topology that has a global bi-directional ring to connect several subnets together. The topology of each subnet is chosen to suit local traffic requirements.



a) b)

Figure 2.5. Other topologies. Examples are a) hierarchical butterfly fat-tree b) irregular network

Irregular and custom topologies

Irregular or custom network topologies are used to design network architectures in order to optimize the use of communication resources and to save power consumption. By using an irregular custom network topology (Figure 2.5b) the number of switches, used to design the network architecture, can be optimized. Accordingly, power dissipation and communication energy in the optimal number of switches can be reduced. The irregular customized networks are also suitable for embedded MPSoC (Multiprocessor System-on-Chip) applications, in which the IP components have different sizes (Samman F. A., 2010). The main drawback of the irregular/custom network topology is the complexity of the routing algorithm. Each routing algorithm of the irregular topology must be customized to avoid possible cyclic dependencies.

2.3.2 Switching method

Switching method determines how a message traverses its route. There are two main switching methods – circuit switching and packet switching. *Circuit switching* is a flow control that operates by first allocating channels to form a circuit from source to destination and then sending messages along this circuit (Figure 2.6a). After data transmission, the circuit can be de-allocated and released for other communications. Circuit switching is connection-oriented, meaning that there is an explicit connection establishment (Lu, 2007). In *packet switching* the messages are split into packets. Depending on a switching method, a packet can be further divided into smaller flow control units (*flits*). A packet consists usually of a *header*, a *payload* and a *tail*. The packet header contains routing information, while the payload carries the actual data. The tail indicates the end of a packet and can contain also error-checking code. Packet switching can be either connection-oriented or connection-less. In contrast to the connection-oriented switching, in the connection-less the packets are routed in a non-guaranteed manner. There is no dedicated circuit built between the source

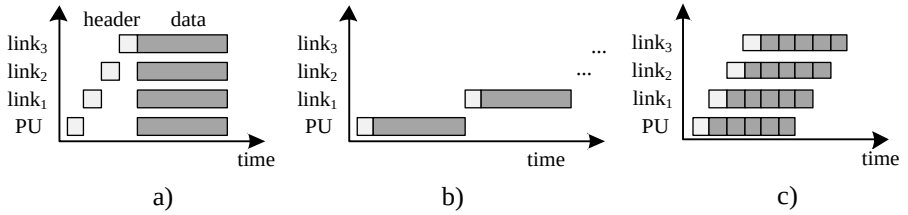


Figure 2.6. A comparison of different switching techniques: a) circuit switching b) store-and-forward c) wormhole (Ni & McKinley, 1993)

and the destination nodes. Most common packet switching techniques include store-and-forward, virtual cut-through and wormhole switching. Below we describe the aforementioned techniques together with virtual channels and wormhole flit-level cut-through switching methods.

Store-and-forward

Store-and-forward switching method is a well-known technique from the computer networking domain. In this approach when a packet reaches an intermediate node, the entire packet is stored in a packet buffer. The packet is forwarded to the next selected neighbour router after that router has an available buffer (Figure 2.6b). Store-and-forward is simple to implement but it has major drawbacks. First, it has to buffer the entire packet before forwarding it to the downstream router. This has a negative effect on router area overhead. Second, the network latency is proportional to the distance between the source and the destination nodes. The network latency of store-and-forward can be calculated according to Ni and McKinley (1993) as follows:

$$Latency_{store\text{-}and\text{-}forward} = (L/B)D \qquad (1)$$

where L is message size, B is channel bandwidth and D is distance in hops. The smallest flow control unit in store-and-forward switching method is a packet.

Virtual cut-through

To decrease the amount of time spent transmitting data Kermani and Kleinrock (1979) introduced the virtual cut-through switching method. In virtual cut-through a packet is stored at an intermediate node only if the next required channel is busy. The network latency of the virtual cut-through can be calculated as follows:

$$Latency_{virtual\ cut\text{-}through} = (Lh/B)D + L/B \qquad (2)$$

where Lh is size of the packet header. Usually the message size is times bigger than the packet header and therefore the distance D will produce a negligible effect on the network latency. The smallest flow control unit is a packet.

Wormhole switching

Wormhole switching operates like virtual cut-through but with channels and buffers allocated to flits rather than packets (Dally & Towles, 2004). A packet is divided into smaller flow control units called flits. There are three types of flits – *body*, *header*, and *tail*. The header flit governs the route. As the header advances along its specified route, the rest of the flits follow in a pipeline fashion. If a channel is busy, the header flit gets blocked and waits the channel to become available. Rather than collecting and buffering the remaining flits in the current blocked router, the flits stay in flit buffers along the established route. Body flits carry the data. The tail flit is handled like a body flit but its main purpose is to release the acquired flit buffers and channels. The network latency of wormhole switching can be calculated according to Ni and McKinley (1993) as follows:

$$Latency_{wormhole} = (Lf/B)D + L/B \qquad (3)$$

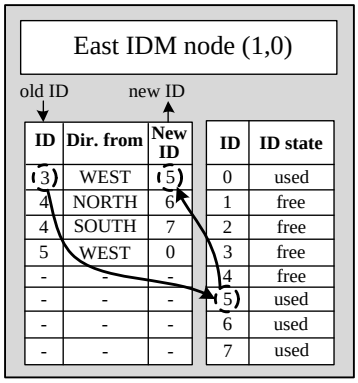
where *Lf* is size of the flit. In similar way to virtual cut-through distance *D* has not significant effect on the network latency unless it is very large. Wormhole switching is more efficient than virtual cut-through in terms of buffer space. However, this comes at the expense of some throughput since wormhole switching may block a channel mid-packet (Dally & Towles, 2004).

Virtual channel flow control

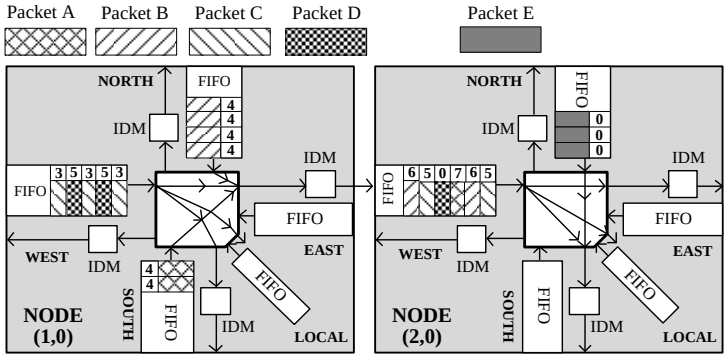
Virtual channel (VC) flow control associates several virtual channels (channel state and flit buffers) with a single physical channel. Virtual channels overcome the blocking problem of the wormhole switching by allowing other packets to use the channel bandwidth that would otherwise be left idle when a packet blocks (Dally & Towles, 2004). It requires an effective method to allocate an optimal number of VCs. According to Bjerregaard & Mahadevan (2006) VCs between 2 and 16 per physical channel have been typically proposed for NoCs. Allocating the virtual channels uniformly results in a waste of area and significant leakage power, especially at nanoscale (Bjerregaard & Mahadevan, 2006; Huang, Ogras, & Marculescu, 2007). Virtual channels increase total buffer counts and might result in power consumption that would exceed the target constraint for an embedded application. Moreover, VCs can increase delay in router's critical path due to extra arbitrations, thus it potentially affects the cycle time or pipeline depth of the router (Samman, Hollstein, & Glesner, 2011). Therefore, much more hardware-efficient router architectures are required in order to come up with cost-efficient solutions.

Wormhole flit-level cut-through switching method

Wormhole flit-level cut-through switching method is designed to overcome the *head-of-line blocking* problem, which commonly occurs when using traditional wormhole switching method. The problem is solved by allowing the flits of the competing wormhole messages to be interleaved at flit level in the same communication link without using virtual channels. In order to provide such



a) ID manager



b) ID-based flit flow

Figure 2.7. Conceptual view of the IDMA approach (Samman, Hollstein, & Glesner, 2008)

flexible communication media sharing, a concept of locally organized packet identity (ID) division multiple access (IDMA) method suitable for NoCs is being used. Local ID slots are distributed over every communication link, which can be attached to every flit of a packet or data stream as its local ID-tag (Samman, Hollstein, & Glesner, 2008).

Figure 2.7 depicts the conceptual view of the communication media sharing with local ID-tag management. The ID-manager (IDM) provides ID-slots for packets and will guarantee that different packets will have a different ID-tag. In the example depicted in Figure 2.7 the IDM provides 8 ID-tags (8 virtual space slots) for each communication link. It is important to note that the ID table size is configurable. The size depends on the required number of interleaved traffic flows. The IDM will manage the ID allocation before a new different packet enters the next FIFO buffer. Figure 2.7a illustrates the functionality of an IDM in accordance with packet flows in Figure 2.7b. The packets are classified based on their ID and from which in-port they have come from. For a new packet header

(Packet C from west in-port with ID 3), the IDM will search for a free ID. If the free ID has been found (i.e. ID 5), then old ID of the packet header (ID 3) is replaced by the new ID (ID 5), and the state of the ID is set to "used".

There is also a possibility that packets coming from different input ports have the same ID-tag, i.e. packet A from south and B from north with ID 4 (Figure 2.7b). The IDM will solve the situation in such way that the packets will have different IDs in the next FIFO (e.g. new ID 6 for Packet B and new ID 7 for Packet A). For payload flits following the header flit of the packets, their IDs will be replaced automatically by using lookup-table mechanism. If there are no more available IDs in the IDM, new packets cannot be forwarded into the output port. After the last (tail) flit of the packet flows through the router, all information related to its ID-tag will be deleted from the tables.

The IDMA approach is being used in Extendable Hierarchical Network-on-Chip (XHiNoC) simulator that is the main NoC simulator used in this PhD thesis. Our system-level design tool has been interfaced with XHiNoC being able to verify the produced schedules by executing the applications on the XHiNoC simulator. This method is explained in more detail in Chapter 3.

2.3.3 Routing

Routing algorithm determines the routing paths the packets may follow through the network. A good routing algorithm can be characterized by low communication latency, high network throughput, and low implementation cost in hardware. Features contributing to hardware cost are number of channels, buffers, and control logic. Features contributing to low latency and high throughput are freedom from deadlocks, freedom from livelocks, routing packets via shortest paths, load balancing the traffic and routing packets adaptively.

Deadlocks occur in an interconnection network when a group of packets is unable to progress because they are waiting on one another to release the resources, usually buffers or channels (Dally & Towles, 2004). Deadlocks have fatal effects on a network. Therefore deadlock avoidance or deadlock recovery should be considered for routing algorithms that tend to deadlock. *Livelocks* cause packets to move through the network, but they do not make progress toward their destinations (Dally & Towles, 2004). It can happen for example when packets are not allowed to take the shortest routes. Usually livelocks are being handled by allowing a certain number of misroutes after which the packet is discarded and need to be re-sent.

Routing algorithms can be divided into *unicast routing* and *multicast routing* based on the number of destinations. The unicast routing sends packets from a single source to a single destination node. In the multicast routing a single source packet is sent to multiple destination nodes. In computer networking these terms are referred to also as traffic or message types where the third type is broadcast traffic. In broadcast traffic a message is transferred to all receivers.

Another classification of routing algorithms is based on path diversity and adaptivity having *deterministic*, *oblivious* and *adaptive routing*.

Deterministic routing

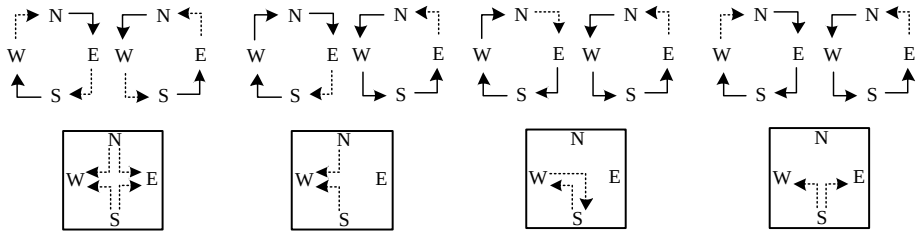
Deterministic routing chooses always the same path given the same source and destination node. An example of a deterministic routing algorithm is dimension-order XY routing. In XY routing the processing cores are numbered by their geographical coordinates. Packets are routed first via X- and then via Y-axis by comparing the source and destination coordinate. The pseudocode of XY routing is depicted in Figure 2.8. Deterministic routing has a small implementation overhead but it can cause load imbalance on network links. Deterministic routing cannot also tolerate permanent faults in a NoC and re-route the packets.

Oblivious routing

Oblivious routing considers all possible multiple paths from the source node to the destination but does not take the network state into account. Oblivious non-deterministic routing algorithms can distribute uniformly the communication load in situations where adaptive solutions are too expensive or slow. One of the first and well-known oblivious routing algorithms is Valiant's randomized routing algorithm (Valiant & Brebner, 1981). It has two phases. In the first phase a packet is sent from a source to a randomly chosen intermediate router. In the second phase the packet is delivered from the intermediate router to the destination. In the two phases an arbitrary routing algorithm can be used – for example for mesh or torus dimension-order routing algorithm could be used. Valiant's algorithm provides good worst-case performance at the expense of locality and low average-case throughput. Successive work in that area has focused in improving the locality and the average delays (Cho, Lis, Shim, Kinsy, & Devadas, 2009; Harsha, Hayes, Narayanan, Räcke, & Radhakrishnan, 2008).

```
XY_Routing (X_current, Y_current, X_destination, Y_destination)
1     X_offset = X_destination - X_current
2     Y_offset = Y_destination - Y_current
3     if X_offset > 0 then direction = EAST
4     else if X_offset < 0 then direction = WEST
5     else if X_offset = 0 and Y_offset > 0 then direction = NORTH
6     else if X_offset = 0 and Y_offset < 0 then direction = SOUTH
7     else direction = LOCAL
8     end if
end XY_Routing
```

Figure 2.8. Pseudocode of XY routing



a) X-First (XY) b) West-First (WF) c) Neg-First (NegF) d) North-Last (NL)

Figure 2.9. Turn models that can avoid deadlock configuration (Samman F. A., 2010)

Adaptive routing

Adaptive routing distributes the traffic dynamically in response to the network load. For example, it re-routes packets in order to avoid congested areas or failed links. Adaptive routing has been favourable providing high fault tolerance. According to Dally and Towles (2004) in contrast to fully adaptive routing there can be algorithms that target some specific network metrics such as minimal adaptive and load-balanced adaptive routing. A shortest route in a minimum adaptive routing algorithm is chosen from all possible candidates based on information about the network state in each hop. Load-balanced adaptive algorithm uses local network utilization information to make the routing decision trying to balance the traffic among the network links. The drawbacks of adaptive routing include higher modelling and implementation complexity.

Turn model

In order to avoid cyclic dependencies leading to a deadlock configuration and to increase adaptivity the turn model was proposed by Glass and Ni (1992). The model is based on analysing the directions in which packets can turn in a network and the cycles that the turns can form. The idea is prohibiting just enough turns to break all the cycles produced by routing algorithms. The work has presented examples of turn models for adaptive routing algorithms in 2D mesh-based interconnection networks. In a mesh-like network, there are four available turns at each clockwise and counter clockwise turns. Figure 2.9 presents four selected turn models that can be used to avoid a deadlock configuration (Samman F. A., 2010). The solid lines in the figure represent the allowed turns, and the dashed lines represent the prohibited turns. One of the well-known static routing algorithms that can be described by the turn model is the dimension-order routing algorithm. The turn model of the XY routing algorithm is presented in Figure 2.9a. As shown in the Figure 2.9a, four turns are prohibited to avoid a deadlock configuration, i.e. north-east, south-east, north-west and south-west turns. Because of the applied prohibited turns, routing algorithm is static whereby packets are always routed first to X-axis, then to Y-axis.

2.3.4 Flow control

Flow control deals with network load monitoring and congestion resolution. Due to the limited buffers and throughput the packets may be blocked and flow control decides how to resolve this situation. The flow control techniques can be divided into two – *bufferless* and *buffered flow control*. The bufferless flow control is the simplest in its implementation. In bufferless flow control there are no extra buffers in the switches. The link bandwidth is the resource to be acquired and allocated. There is need for an arbitration to choose between the competing communications. Unavailable bandwidth means that a message needs to be misrouted or dropped. Dropped message has to be resent by the source. Misrouting and message dropping both increase latency and decrease efficiency (throughput) of the network. Deflection routing is an example of the bufferless flow control. In deflection routing, an arbitrary routing algorithm chooses a routing path, while deflection policy is handling the resource contentions. In the case of network contention, the deflection policy grants link bandwidth to the higher priority messages and misroutes the lower priority messages. Deflection routing allows low overhead switch design while at the same time provides adaptivity for network load and resilience for permanent link faults.

In the buffered flow control, a switch has buffers to store the flow control unit(s) until bandwidth can be allocated to the communication on outgoing link. The granularity of the flow control unit can be different. In store-and-forward and virtual cut-through both the link bandwidth and buffers are allocated in terms of packets but in wormhole switching in flits. In buffered flow control, it is crucial to distribute the buffer availability information between the neighbouring routers. If buffers of the upstream routers are full, the downstream routers must stop transmitting any further flow control units. The flow control accounting is done at link level. The most common flow control accounting techniques are credit-based, on/off and ack/nack (Dally & Towles, 2004).

2.3.5 Quality of service

Quality of service (QoS) gives guarantees on packet delivery. The guarantees include correctness of the result, completion of the transmission, and bounds on the performance (Lu, 2007). The network traffic is divided usually into two service classes – *best-effort* and *guaranteed*. A best-effort service is connectionless. Packets are delivered when possible depending on the current network condition. A guaranteed service is typically connection-oriented. The guaranteed service class packets are prioritized over the best-effort traffic. In addition, guaranteed service avoids network congestions by establishing a virtual circuit and reserving the resources. It can be implemented for example by using multiple timeslots (Time Division Multiple Access, TDMA), IDMA or virtual channels.

2.3.6 Generic on-chip router architecture

The architecture of an on-chip router depends on various aspects – switching/routing method, flow control, quality of services requirements etc. These implementation choices may affect the processing delay of a router. As the router processing delay is part of our communication model it is important to have an overview of the generic on-chip router architecture. According to (Goossens, Dielissen, & Radulescu, 2005; Kumar et al., 2002; Guerrier & Greiner, 2000; Bertozzi & Benini, 2004; Bjerregaard & Mahadevan, 2006; Samman, 2010) a generic on-chip router contains the following components: first-in first-out (FIFO) buffer, arbiter, crossbar switch, routing engine and link controller.

First-in first-out buffer

First-in first-out buffer is used to buffer incoming and/or outgoing data in the switch. Some NoC architectures implement FIFO buffers either in input ports or in output ports to cut the buffering costs. When virtual channels are being used, each virtual channel implements a separate flit buffer. A virtual channel controller arbitrates among requesting packets and multiplexes virtual channels over the corresponding link on a flit-by-flit basis. However, allocating the virtual channels uniformly results in a waste of area and significant leakage power, especially at nanoscale (Huang, Ogras, & Marculescu, 2007).

Arbiter

Arbiter controls access and avoids conflicts in outgoing port that can happen when several packets/flits from incoming port request the same outgoing port. In literature several arbitration schemes have been proposed such as first-come first served, rate controlled, round-robin, priority-based, deadline-based, contention-aware and flit-by-flit rotating arbitration (Samman F. A., 2010).

Crossbar switch

Crossbar switches are the key building blocks of the NoCs. They form interconnects between input and output ports of the router. The individual switches connecting multiple inputs to multiple outputs are arranged in a matrix. Typically, all the input ports and the output ports have the same bit-width and operating frequency. If a crossbar has m inputs and n outputs then such crossbar has a matrix with $m \times n$ cross-points.

Routing engine

Routing engine is used to determine the routing path for incoming packets. The routing can be implemented as static table-based routing, routing algorithm described by a state machine or combination of both.

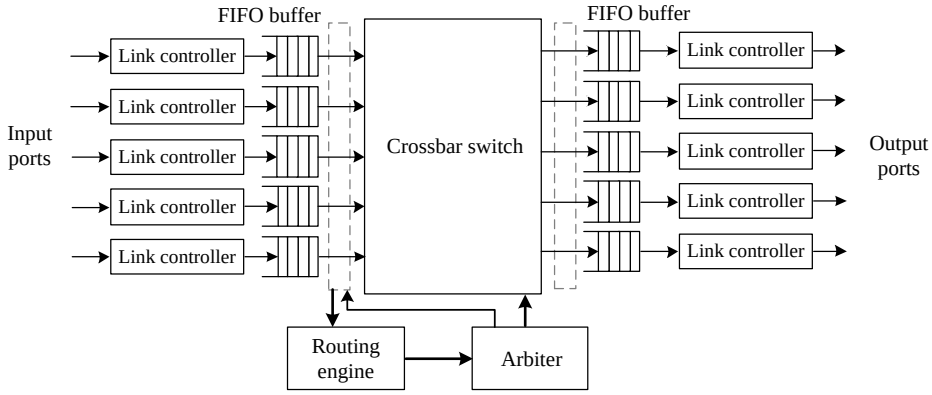


Figure 2.10. Generic router model

Link controller

The task of a link controller is to control data flow between output and input ports of adjacent routers. The most common flow control accounting techniques are credit-based, on/off and ack/nack (Dally & Towles, 2004). Data synchronization interfaces are also implemented in this unit to synchronize the data transmission from one switch to another one. Some data synchronization methods that can be implemented in the NoCs are source-synchronous, mesochronous, asynchronous queue-based, pipelined repeater-based or the handshake mechanism (Samman F. A., 2010).

Example generic router model

An example generic router model is depicted in Figure 2.10. The router has communication channels to four neighbouring routers and to local processing core. Each input port has its own separate FIFO buffer in which the incoming flit/packet is buffered. A 5x5 crossbar switch is used as the switching fabric in the router. There is a routing engine processing the packet at the head of the input FIFO to determine which output direction (i.e. north, east, west, south, local) the incoming packet/flit should be routed. Once the routing decision has been made the arbiter is requested to set up a path to the corresponding output channel. The crossbar arbiter maintains the status of current crossbar connection and determines whether or not to grant connection permission to the link controller. If there are multiple requests at the same time then the arbiter uses a policy to decide which input channel gets the access. Such simple and generic router model without virtual channels is being assumed also in our system-level design framework that is described in Chapter 3.

2.3.7 Further reading

There is a comprehensive survey of research and practices of networks-on-chip by Bjerregaard and Mahadevan (2006), survey of different NoC implementations by Salminen, Kulmala, and Hämäläinen (2008) and overview of outstanding research problems in NoC design by Marculescu et al. (2009).

2.4. System-level design

The technology scaling has enabled designers to pack more and more functionality onto a single chip. However, increasing complexity of systems, stringent time to market requirements and advances in the electronic design automation (EDA) tools have motivated the shift from lower levels of abstraction to the system level. The system-level design methodologies try to take into account important implementation issues already at higher abstraction levels. In this section we describe the key steps of a traditional system-level design flow and give an overview of the classical scheduling and mapping problem.

2.4.1 Traditional system-level design flow

Having its roots in the end of the 80's, system-level design is a hierarchical process that begins with a high-level description of the complete system and works down to fine grained descriptions of individual system modules (Stressing, 1989). Initially, the description of a system is independent from the implementation technology. There are even no details whether some component of the system should be implemented in hardware or in software. Therefore, early system description is more behavioural than structural, focusing on system functionality and performance specification rather than interconnects and modules. In addition to the system specification, it is important to have a possibility to verify the performance and functional specification. A specification at the system-level should be created in such a way that its correctness can be validated by simulation. Such a model is often referred to as simulatable specification. In addition, a model at the system-level should be expressed in a form that enables verification that further refinements correctly implement the model (Ashenden & Wilsey, 1998). Possible approaches include behavioural synthesis (correct by construction), and formal verification using model checking and equivalence checking (Ashenden & Wilsey, 1998). A third essential element of system-level design is the exploration of various design alternatives. For example, a system designer has choices whether to implement a function in hardware or in software, whether to solve it with sequential or parallel algorithm. The analysis of trade-offs between design alternatives is a key element of system-level design and shows the quality of the particular system-level design flow. It is important that a system-level design flow is

supported by system-level tools – simulators/verifiers, estimators and partitioners. The first system-level design tools were introduced in 1980 by Endot, a company formed out of the staff at the Case Western Reserve University (Stressing, 1989). The need for system-level design tools was motivated by the complexity of the aerospace and defence systems but it soon became apparent that these tools were applicable to design of complex digital hardware/software systems of any type (Stressing, 1989).

At the system-level, a system can be modelled as a collection of active objects that react to events, including communication of data between objects and stimuli from the surrounding environment. Abstractions are needed in a number of areas to make the system-level behavioural modelling tractable in the following views:

- abstraction of data,
- abstraction of concurrency, and
- abstraction of communication and timing (Ashenden & Wilsey, 1998).

Of course, the views of abstractions can be different, e.g., concurrency is replaced by calculation, and communication and timing are looked at separately (Jantsch, 2003).

The classical system-level design flow, depicted in Figure 2.11, consists of several consecutive design tasks with loopbacks to previous steps (Lagnese & Thomas, 1989). An input to the system-level design flow is a system specification that is represented in a formal way, e.g., dataflow or task graph. In the dataflow graph, the nodes represent operators and the arcs between them represent data and control dependencies like in task graphs. The operators are scheduled into time slots called control steps. Scheduling determines the execution order of the operators. The scheduling can be either static or dynamic. In the dynamic scheduling, the start times are obtained during execution (online) based on priorities assigned to processes. In the static scheduling, the start times of the processes are determined at the design time (off-line) and stored in the form of schedule tables. Scheduling sets lower limits on the hardware because operators scheduled into the same control step cannot share the hardware. Thus, scheduling has a great impact on the allocation of the hardware. After the scheduling the data-flow operators are mapped to the allocated hardware. If the hardware platform is given with the system specification then designer can also start first with the mapping and then perform the scheduling. Since both, mapping and scheduling, are computationally intensive tasks, the parallel execution of those design phases is extremely difficult. When the results of the system-level design flow do not satisfy the initial requirements, either the mapping or the scheduling of application's components should be changed. If no feasible solution is found, changes are needed in the system specification or in the architecture. After an acceptable schedule is found, lower abstraction-levels of hardware/software co-design will follow.

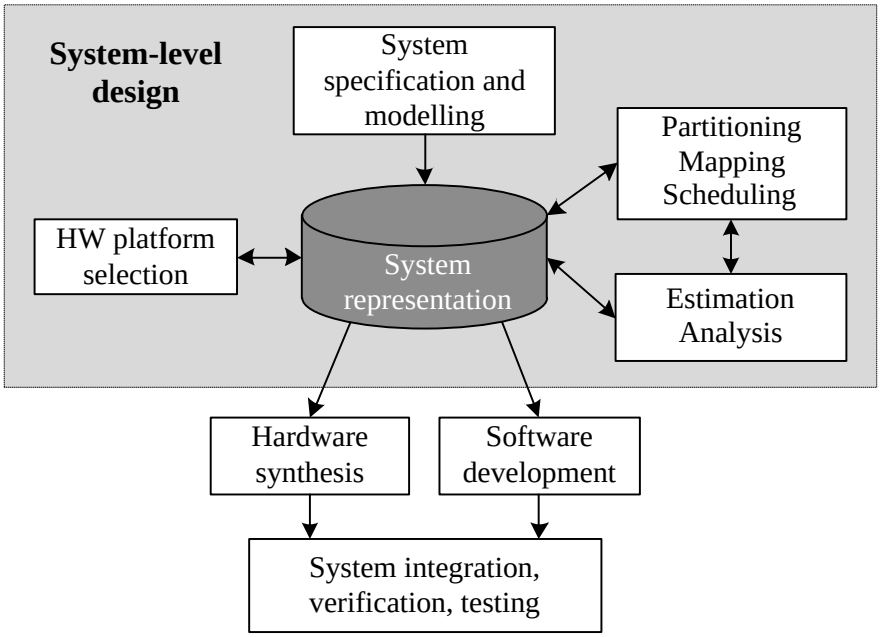


Figure 2.11. Classical system-level design flow

Refinement to a software implementation is facilitated by a system-level modelling language that is closely related to programming languages. In principle, both the hardware and software implementations could be expressed in the same language as the system-level model, thus avoiding semantic mismatches between different languages in the design flow (Ashenden & Wilsey, 1998). Some of the most common system-level design languages are StateCharts (Harel, 1987), Estelle (Budkowski & Dembinski, 1987), SDL (Færgemand & Olsen, 1994), CSP (Hoare, 1978) and SystemC (SystemC, 2009). Most recent and prominent of those is SystemC. SystemC is a C++ class library that can be used to create cycle-accurate models for software algorithms, hardware architectures and interfaces, related to system-level design (SystemC, 2009). C++ is a general-purpose programming language used in wide areas, such as application software, device drivers, embedded software or high-level synthesis (ISO/IEC 14882:2003 standard on the C++, 2003).

Most of modern embedded systems have both the hardware and software components. When designing such a system, it is important that both sides are developed not in an isolated but in an integrated manner. The generic hardware/software co-design methodology, as a part of the overall system design flow, supports concurrent development of software and hardware. Important tasks in such a development are co-simulation and co-verification. It should be noted that in many cases, systems have also analogue parts that should be designed concurrently with rest of the system (Gerstlauer, Haubelt, Pimentel, Stefanov, Gajski, & Teich, 2009).

2.4.2 Task mapping and scheduling

The term scheduling refers to a very generic problem that exists in different areas and domains. Jobs need to be processed in manufacturing, aircrafts are waiting for landing in a clearance list or customers are waiting in a restaurant to be served by tellers. In computer engineering tasks, that are part of an application (program), are waiting to be executed on a parallel computer. Whenever there is a choice of determining the order of execution of tasks and assigning them to processing entities we can call it the scheduling problem. In general, the scheduling problem assumes a set of resources and a set of consumers serviced by those resources according to a certain policy (El-Rewini, Ali, & Lewis, 1995). Typically the scheduling and the assignment (mapping) of tasks to processors is done in the same process. Although in some cases the static mapping can be also given. Then the problem is reduced to determining the execution order of the tasks on a processor. In the NoC domain we can talk about scheduling in terms of application scheduling, packet scheduling in real-time traffic, scheduling policy for routing control unit or scheduling in virtual channels. There is a wide range of scheduling problems and in the same way a considerable amount of scheduling methods to solve them.

Before looking for a suitable scheduling technique it is worth to classify the systems based on their predictability and timing requirements into real-time and non-real-time systems. Real-time systems can be further divided into three classes:

- **Hard real-time** – a violation in real-time constraints can lead to disastrous consequences such as loss of life or property. Hard real-time systems are time-constrained, cost-constrained and fault-tolerant.
- **Firm real-time** – infrequent constraint violations are tolerable but might degrade the system quality of service. The tasks that have missed their deadline are usually dropped because late results have little or no value.
- **Soft real-time** – constraints are less critical and can be violated. However, the violations are not desirable as they might lead to degraded quality of service. Multimedia systems have been historically considered as soft real-time systems. Although, with increased bandwidth and stringent timing requirements multimedia applications are considered nowadays rather part of firm real-time system class.

In this work we are interested in the class of hard and firm real-time systems implemented on network-on-chip based system-on-chip.

In the *pre-emptive scheduling* the execution of a running task can be interrupted at any time and resumed later. In *non-pre-emptive scheduling* a task may not be interrupted once it starts its execution. Scheduling can be performed

dynamically (*online scheduling*) or calculated offline (*static scheduling*). In static scheduling the scheduling process has complete knowledge of the task set and its constraints. It produces a single schedule that is fixed during the execution. Static scheduling has been preferable for scheduling applications on hard and firm real-time systems. Dynamic scheduling works with the active set of tasks. When new ready tasks arrive the schedule might change. According to Stankovic, Spuri, Di Natale, and Buttazzo (1995) offline scheduling is often equated to static scheduling that is wrong. In building a real-time system, offline scheduling analysis should be always done, regardless of whether the final runtime algorithm is static or dynamic.

Application consists of tasks and their precedence constraints. A classical way to represent such a model is using directed acyclic graph (DAG) called *task graph* (Marculescu, Ogras, Li-Shiuan Peh Jerger, & Hoskote, 2009). In a task graph each vertex represents a task and a directed edge a dependency between two tasks. A task represents a part of an application. The task granularity can vary – from a single operation to a function call. Weights can be associated with vertexes and edges that are representing either computation or communication costs. In classical scheduling inter-processor communication (IPC) cost has been typically considered as a constant and not included in the scheduling. In NoC-based system the communication delay does not depend only on the message size but also on the resource mapping and needs to be taken into account. The scheduling problem and related definitions are given and explained in more detail in Section 3.2.

According to El-Rewini, Ali, and Lewis (1995) the task scheduling problem is in many cases computationally intractable and belongs to the class of NP-hard or NP-complete problems. There are only three cases for which polynomial-time algorithms can be obtained: tree-structured task graphs on an arbitrary number of processors, interval orders on an arbitrary number of processors, and arbitrary task graphs on two processors. Since, these restrictions serve no practical use for our research we will review some heuristics and optimization techniques that can be applied in a general case.

Heuristics and optimization techniques

To find solutions for real-time scheduling problems heuristics can be used. A heuristic provides a schedule in less than exponential time but does not guarantee an optimal solution. A heuristic that can optimally schedule a particular application on a certain target machine may not produce optimal schedules for other task graphs on other machines (El-Rewini, Ali, & Lewis, 1995). One of the most popular general-purpose scheduling heuristic is list scheduling. In list scheduling each task is assigned a priority. Tasks that are ready to execute are sorted based on the priority in decreasing order and scheduled on "best" available processor. Various methods exist to calculate and assign priorities to tasks. Priority assignment results in different schedules as tasks are selected in a different order.

Two examples of dynamic list scheduling algorithms are earliest time first (ETF) by Hwang, Chow, Anger, and Le (1989) and dynamic level scheduling (DLS) by Sih and Lee (1993). In these algorithms the priorities of ready tasks are computed dynamically by evaluating the start time of every ready task on every processor. The ready task together with a processor that achieves the earliest start time is selected.

Another possibility to solve the scheduling or mapping problem in feasible time is to use optimization techniques. Some examples are branch-and-bound, simulated annealing and genetic algorithms. *Branch-and-bound* (B&B) is a general algorithm for finding optimal solutions for various computationally intensive optimization problems. Branch-and-bound consists of three main functions – branching, bounding and pruning. Branching describes the problem as a search tree whose nodes are subsets of given problem. Bounding calculates the upper and lower bounds that are used to evaluate a set of candidates and prune the ones that do not lead to an optimum. The main issues in branch-and-bound are to find the appropriate lower bound function and effective branching strategy. *Simulated annealing* (SA) is a probabilistic metaheuristic for the global optimization problem, locating a near-optimal solution in feasible time (Kirkpatrick, Gelatt, & Vecchi, 1983). Simulated annealing belongs to the class of non-optimal algorithms. It has been applied to several combinatorial optimization problems from various fields of science and engineering. *Genetic algorithms* (GA) are search algorithms that are based on the principles of evolution and natural genetics. A set of solutions to the problem is called a population. New solutions are generated by breeding and mutation. The idea comes from the theory of evolution where the most suited elements of the population are likely to survive and generating offspring, transmitting their biological inheritance to the next generation. Genetic algorithms operate through a straightforward cycle: creation of a population, evaluation of the elements, selection of the best elements and reproduction to create a new population (Kwok & Ahmad, 1999).

In this work we use some of the classical scheduling and optimization algorithms such as list scheduling, branch-and-bound and simulated annealing. For additional information, there is a comprehensive survey of static scheduling algorithms for allocating directed task graphs to multiprocessor systems by Kwok and Ahmad (1999).

2.5. Design issues of NoC-based systems

To overcome some of the system-on-chip design challenges, described in the first part of this chapter, the network-on-chip paradigm has been proposed. Networks-on-chip have brought new research topics that need to be explored to reach their full potential. In this section we give an overview of the key research challenges of NoC-based systems.

Communication modelling

NoC communication latency depends on various parameters such as topology, routing algorithm, switching method, etc., and need to be calculated after task mapping and before the task graph scheduling (Marculescu, Ogras, Li-Shiuan Peh Jerger, & Hoskote, 2009). In several research papers, the average or the worst case communication delay has been considered (Lei & Kumar, 2003; Marcon, Kreutz, Susin, & Calazans, 2005; Hu & Marculescu, 2005; Shin & Kim, 2004; Stuijk, Basten, Geilen, & Ghamarian, 2006; Shim & Burns, 2008; Shin & Kim, 2008). In many cases, it is an approximation that can be either too pessimistic (giving the upper bound) or too optimistic (by not scheduling explicitly the communication or not considering the communication conflicts). Therefore, an efficient system-level NoC design framework requires an approach for the communication modelling and synthesis to calculate the communication deadlines by taking into account also possible network conflicts. The Chapter 3 discusses the problem in more detail and proposes a system level model together with the contention-aware scheduling algorithm to tackle the problem.

Programmability

To write effective parallel embedded system software a programmer needs fast and reliable on-chip communication infrastructure with easy-to-use application programming interface (API) and predictable communication delays. The underlying architecture should provide the necessary services in a flexible way – have best-effort and guaranteed services, have possibility to measure the performance and debug the application.

Reliability and variability

With shrinking transistors and wire dimensions, reliability and variability have become significant challenges for IC designers (Owens, Dally, Ho, Jayasimha, Keckler, & Peh, 2007). Variability might stem from different sources – fabrication, environment and system load. There has been a lot of research made on system reliability in different computing domains by employing data encoding, duplicating system components or software-based fault tolerance techniques. This research have mostly had either focus on low level hardware reliability or covered the macro distributed systems. Due to future design complexities and technology scaling it is infeasible to concentrate only to low level reliability analysis and improvement in SoC design. We should fill the gap by looking at the application level reliability analysis and improvement. NoC platform provides flexibility to tolerate faults and guarantee system reliability. However, to meet the required level of dependability the designed system must be also predictable. Chapter 2.6 gives an overview of the issues related with dependable system-on-chip design in more detail. In Chapter 6 we describe an algorithm to synthesize dependable application schedules that are able to tolerate a given number of transient or intermittent faults.

System-level design

The system-level design flow for NoC-based systems follow the same principles as described previously. That is, the initial specification is modelled to estimate performance and resource requirements when using different architectural solutions. This includes platform selection, task mapping and task scheduling. The difference is that because of the rather complex communication behaviour between processing cores, communication mapping and scheduling between tasks should be addressed with care. The reason for that is rather simple – communication latencies may be unpredictable, especially when trying to apply dynamic task organisation. Therefore, the traditional scheduling techniques that are applicable to the hard real-time and distributed systems are not suitable as they address only the bus-based or point-to-point communication. A communication schedule could be extracted by simulating an application on a NoC simulator, but the simulation speed will be the limiting factor. Moreover, according to Henkel, Wolf, and Chakradhar (2004) simulating NoCs will become quite a challenge since NoCs will comprise many processing units connected through a complex communication infrastructure. New simulation strategies are needed such as „cycle-approximate“ that simulate the systems „sufficiently“ accurately with much higher simulation speeds than ordinary cycle-accurate simulators. System-level design for NoCs has one major difference when compared to the traditional system-level design – hardware platform is either fixed or has limited modification possibilities (Keutzer, Newton, Rabaey, & Sangiovanni-Vincentelli, 2000). Therefore the main focus is on the application design and task distribution between the processing cores.

Evaluation and benchmarking

Network-on-chip performance depends on the traffic workloads it needs to process. Since, implementing a real life application on a NoC is time consuming and complex, analytical or trace based model can be used instead to evaluate the network performance in early design phases. Traffic model refers to a mathematical characterization of a workload generated by an application. In the field of interconnection networks some common traffic patterns exists, which are employed also in the NoC domain. These traffic models cover the spatial distribution of messages.

- *Uniform random traffic* – each node sends messages to every other node with equal probability. It makes traffic uniformly distributed, balancing load even for topologies and routing algorithms that have normally poor load balancing characteristics.
- *Permutation traffic* – all the traffic from each source is directed to one destination. It can be represented by a permutation function $d = f(s)$ that maps a source to a destination. Examples of permutation traffic include bit permutations such as shuffle, transpose, bit reverse and digit permutations such as tornado and neighbour.

To evaluate the temporal characteristics of an interconnection network the injection rate can be adjusted to generate bursty traffic. One of the key parameters to adjust during benchmarking is the message size. Message size can be fixed, varied between different runs or computed based on some distribution function. For example, injecting packets into the network from a long message could affect the latency of some competing short messages, increasing the average latency. Therefore it is important to capture in a traffic model all aforementioned aspects of an application or to use application traces.

When we are looking at benchmark applications then usually existing benchmark suites for multiprocessor computing have been adjusted for NoC domain. Examples are SPLASH-2 (Woo, Ohara, Torrie, Singh, & Gupta, 1995) and PARSEC (Bienia, Kumar, Singh, & Li, 2008) that are referenced in many NoC-related research papers. Gratz and Keckler (2010) use SPLASH-2 and PARSEC as sources of realistic traffic based on which they propose a set of new workload characteristics. They show that these characteristics correlate more accurately with network congestions in realistic workloads than traditional metrics. However, according to (Marculescu, Ogras, Li-Shiuan Peh Jerger, & Hoskote, 2009; Grecu, et al., 2007; Owens, Dally, Ho, Jayasimha, Keckler, & Peh, 2007) there is still a lack of driving applications for network-on-chip evaluation and benchmarking. Classic benchmarks for multiprocessor systems are application-oriented, and cannot be used directly for communication-intensive architectures such as NoCs. Moreover, the nature of the applications running on NoC-based systems is expected to be more varied and heterogeneous compared to typical applications for multiprocessor computers (Grecu, et al., 2007). The diversity of network-on-chip platforms makes the situation even worse. It is hard to compare the results if underlying architecture is different. Fortunately, Open Core Protocol International Partnership (OCP-IP) has launched a network-on-chip benchmarking initiative. The first deliverable of the initiative is a white paper that outlines the essential features of a NoC benchmarking environment (Grecu, et al., 2007).

One recent research paper by Liu, et al. (2011) tries also to bridge the benchmarking gap. The authors present a realistic traffic benchmark suite, called MCSL, and describe the methodology used to generate it. The MCSL benchmark suite includes a set of realistic traffic patterns for 8 typical MPSoC applications and covers popular NoC architectures in various scales. MCSL captures not only the communication behaviours in NoCs but also the temporal dependencies among them. Each traffic pattern in MCSL has two versions, a recorded traffic pattern and a statistical traffic pattern. The former provides detailed communication traces for comprehensive NoC studies, while the latter helps to accelerate NoC explorations at the cost of accuracy (Liu, et al., 2011).

Due to the lack of established benchmark applications in NoC domain we have used in this thesis synthetic task graphs to evaluate different aspects of our approach. Using synthetic task graphs is a common way to demonstrate scaling and feasibility of the proposed approaches and effectively used by many authors

(Sinnen & Sousa, 2005; Hu & Marculescu, 2005; Manolache, Eles, & Peng, 2007).

2.6. Dependable systems-on-chip

In 1997, Kiang has depicted dependability requirements over the past several decades showing shift in the dependability demands from the product reliability into customer demands for more and more functionality. The percentage of hardware failures noted in the field is claimed to be minimal. However, technology scaling brings process variations and increases the number of transient faults, motivating the fault-tolerance design of systems (Constantinescu, 2003). A system-on-chip designer has to assume that the manufactured devices might contain faults and an application, running on the system, must be aware that the underlying hardware is not perfect. According to Wattanapongsakorn and Levitan (2000) a design framework that integrates dependability analysis into the system design process must be implemented. To date, there are very few such system design frameworks, and none of them provide support at all design abstraction levels in the system design process, including evaluations of system dependability.

System dependability can be described as quality-of-service having attributes reliability, availability, maintainability, testability, integrity and safety (Wattanapongsakorn & Levitan, 2000). Achieving a dependable system requires combination of a set of methods that can be classified into:

- *fault-avoidance* – how to prevent (by construction) fault occurrence;
- *fault-tolerance* – how to provide (by redundancy) service in spite of faults occurred or occurring;
- *error-removal* – how to minimize (by verification) the presence of latent errors;
- *error-forecasting* – how to estimate (by evaluation) the presence, the creation and the consequences of errors (Laprie, 1985).

In the next sections we will give an overview of classification of faults, describe the fault tolerance and some of its techniques to increase the system reliability.

2.6.1 Classification of faults

Different sources classify the terms fault, error, failure differently. However, in everyday life we tend to use them interchangeably. According to IEEE (2009) standard 1044-2009 of software anomalies, an error is an action which produces an incorrect result. A fault is a manifestation of the error in software. A failure is a termination of the ability of a component to perform a required action. A failure may be produced when a fault is encountered. In Koren and Krishna (2007) view a fault (or a failure) can be either a hardware defect or a software mistake. An error is a manifestation of the fault or the failure.

Software faults are in general all programming mistakes (bugs). Hardware faults can be divided into three groups: permanent, intermittent and transient faults according to their duration and occurrence.

- **Permanent faults** – are irreversible physical defects in hardware caused by manufacturing process variations or wearout mechanism. Once a permanent fault occurs it does not disappear. Manufacturing tests are used to detect permanent faults caused by the manufacturing process. Fault tolerance techniques can be used to achieve higher yield by accepting chips with some permanent faults that are then masked by the fault tolerance methods.
- **Intermittent faults** – occur because of unstable or marginal hardware. They can be activated by environmental changes, like higher temperature or voltage. Usually intermittent faults precede the occurrence of permanent faults (Constantinescu, 2003).
- **Transient faults** – cause a component to malfunction for some time. Transient faults are malfunctions caused by some temporary environmental conditions such as neutrons and alpha particles, power supply and interconnect noise, electromagnetic interference and electrostatic discharge (Constantinescu, 2003). Transient faults cause no permanent damage and therefore they are called soft errors. The soft errors are measured by soft error rate (SER) that is a probability of error occurrence.

2.6.2 Fault tolerance

Fault tolerance is an exercise to exploit and manage redundancy. Redundancy is the property of having more of a resource than is minimally necessary to provide the service. As failures happen, redundancy is exploited to mask or work around these failures, thus maintaining the desired level of functionality (Koren & Krishna, 2007).

Usually we speak of four forms of redundancy:

- **Hardware** – provided by incorporating extra hardware into the design to either detect or override the effects of a failed component. We can have
    - *static hardware redundancy* – objective to immediately mask a failure;
    - *dynamic hardware redundancy* – spare components are activated upon a failure of a currently active component;
    - *hybrid hardware redundancy* – combination of the two above.
- **Software** – protects against software faults. Two or more versions of the software can be run in the hope that that the different versions will not fail on the same input.
- **Information** – extra bits are added to the original data bits so that an error in the bits can be detected and/or corrected. The best-known forms of information redundancy are error detection and correction coding. Error codes require extra hardware to process the redundant data (the check bits).
- **Time** – deals with hardware redundancy, re-transmissions, re-execution of the same program on the same hardware. Time redundancy is effective mainly against transient faults (Koren & Krishna, 2007).

Metrics are used to measure the quality and reliability of devices. There are two general classes of metrics that can be computed with reliability models:

- the expected time to some event, and
- the probability that a system is operating in a given mode at time *t*.

The expected time to some event is characterized by mean time to failure (MTTF) – the expected time that a system will operate before a failure occurs. Mean time to repair (MTTR) is an expected time to repair the system. Mean time between failures (MTBF) combines the two latter measures and is the expected time that a system will operate between two failures:

$$MTBF = MTTF + MTTR \qquad (4)$$

The second class is represented by the reliability measure. Reliability, denoted by *R(t)*, is the probability (function of time *t*) that the system has been up continuously in the time interval $[t_0, t]$, given that the system was performing correctly at time t_0 (Smith, DeLong, Johnson, & Giras, 2000).

While general system measures are useful at system-level, these metrics may overlook important properties of fault-tolerant NoCs (Grecu, Anghel, Pande, Ivanov, & Saleh, 2007). For example, even when the failure rate is high (causing undesirable MTBF) recovery can be performed quickly on packet or even on flit level. Another drawback is related to the fact that generic metrics represent

average values. In a system with hard real-time requirements the NoC interconnect must provide QoS and meet the performance constraints (latency, throughput). Therefore specialized measures focusing on network interconnects should be considered when designing fault-tolerant NoC-based systems-on-chip. For example, one has to consider node connectivity that is defined as the minimum number of nodes and links that have to fail before the network becomes disconnected or average node-pair distance and the network diameter (the maximum node-pair distance), both calculated given the probability of node and/or link failure (Koren & Krishna, 2007). In 2007 Ejlali, Al-Hashimi, Rosinger, and Miremadi proposed performability metric to measure the performance and reliability of communication in joint view. Performability *P(L, T)* of an on-chip interconnect is defined as the probability to transmit *L* useful bits during the time *T* in the presence of noise. In presence of erroneous communication re-transmission of messages is needed which reduces probability to finish the transmission in a given time period. Lowering the bit-rate increases the time to transmit the messages but also increases probability to finish the transmission during the time interval. According to authors the performability of an interconnect that is used for a safety-critical application must be greater than $1\text{-}10^{-1}$.

2.6.3 Fault tolerance techniques

Fault tolerance has been extensively studied in the field of distributed systems and bus-based SoCs. In the paper of Miremadi and Torin (1995) the impact of transient faults in a microprocessor system is described. They use three different error detection mechanisms – *signature*, *watchdog timer*, and *error capturing instruction* (ECI) mechanism. Signature is a technique where each operation or a set of operations are assigned with a pre-computed checksum that indicates whether a fault has occurred during those operations. Watchdog Timer is a technique where the program flow is periodically checked for presence of faults. Watchdog Timer can monitor, for example, execution time of the processes or to calculate periodically checksums (signatures). In the case of ECI mechanism, redundant machine-instructions are inserted into the main memory to detect control flow errors.

Once a fault is detected with one of the techniques above, it can be handled by a system-level fault tolerance mechanism. In 2006, Izosimov described the following software based fault tolerance mechanisms: re-execution, rollback recovery with checkpointing and active/passive replication. Re-execution restores the initial inputs of the task and executes it again. Time penalty depends on the task length. Rollback recovery with checkpointing mechanism reduces the time overhead – the last non-faulty state (so called checkpoint) of a task has to be saved in advance and will be restored if the task fails. It requires checkpoints to be designed into the application that is not a deterministic task. Active and passive replication utilizes the spare capacity of other computational nodes. In

2007, Koren and Krishna described fault tolerant routing schemes in macro-distributed networks.

Similarly to distributed systems, NoC is based on a layered approach. The fault tolerance techniques can be classified by the layer onto which they are placed in the communication stack. We are, however, dividing the fault tolerance techniques into two bigger classes – system-level and network-level techniques. At the network level, the fault tolerance techniques are based, for example, on hardware redundancy, error detection/correction and fault tolerant routing. By system-level fault tolerance we mean techniques that take into account application specifics and can tolerate even unreliable hardware.

One of the most popular generic fault tolerance techniques is n-modular redundancy (NMR) that consists of *n* identical components and a voter to detect and mask failures. This structure is capable of masking *(n - 1)/2* errors having *n* identical components. The most common values for *n* are three (triple modular redundancy, TMR), five and seven capable of masking one, two and three errors, respectively. Because a system with an even number of components may produce an inconclusive result, the number of components used must be odd (Pan & Cheng, 2007). NMR can be used to increase both hardware and system-level reliability by either duplicating routers, physical links or running multiple copies of software components on different NoC processing cores.

Pande, Ganguly, Feero, Belzer, and Grecu (2006) propose a joint crosstalk avoidance and error correction code to minimize power consumption and increase reliability of communication in NoCs. The proposed schemes, Duplicate Add Parity (DAP) and Modified Dual Rail (MDR), use duplication to reduce crosstalk. Boundary Shift Code (BSC) coding scheme attempts to reduce crosstalk-induced delay by avoiding shared boundary between successive codewords. BSC scheme is different from DAP scheme, such that at each clock cycle, the parity bit is placed on the opposite side of the encoded flow control unit. Data coding techniques can be used in both inter-router and end-to-end communication. Dumitras and Marculescu (2003) propose a fast and computationally lightweight fault tolerant scheme for the on-chip communication, based on an error-detection and multiple-transmissions scheme. The key observation behind the strategy is that, at the chip level, the bandwidth is less expensive than in traditional networks because of the existing high-speed buses and interconnection fabrics that can be used for the implementation of a NoC. Therefore we can afford to have more packet transmissions in order to simplify the communication scheme and to guarantee low latencies. Dumitras and Marculescu call this strategy where IP's communicate using probabilistic broadcast scheme – on-chip stochastic communication. Data is forwarded from a source to destination cores via multiple paths selected by probability. Similar approach is proposed by Pirretti, Link, Brooks, Vijaykrishnan, Kandemir, and Irwin (2004) and by Murali, Atienza, Benini, and De Micheli (2006). Lehtonen, Liljeberg and Plosila (2009) describe turn models for routing to avoid deadlocks and increase network resilience for permanent faults. Kariniemi and Nurmi

(2005) present a fault tolerant Extended Generalized Fat Tree (XGFT) NoC that comes with a fault-diagnosis-and-repair (FDAR) system. The FDAR system is able to locate faults and re-configure routing nodes in such a way that the network can route packets correctly despite the faults. The fault diagnosis and repair is very important as there is only one routing path available in the XGFTs for routing the packets downwards from nearest common ancestor to its destination. Frazzetta, Dimartino, Palesi, Kumar and Catania (2008) describe an interesting approach where partially faulty links are also used for communication. For example, data can be transmitted via "healthy wires" on a 24-bit wide channel although the channel is before degrading 32-bit wide. Special method is used to split and resemble the flow control units. Zhang, Han, Xu, Li and Li (2009) introduce virtual topology that allows using spare NoC cores to replace faulty ones and re-configure the NoC to maintain the logical topology. A virtual topology is isomorphic with the topology of the target design but is a degraded version. From the viewpoint of programmers and application, they always see a unified virtual topology regardless of the various underlying physical topologies. Another approach is to have a fixed topology but remap the tasks on a failed core. Ababei and Katti (2009) propose a dynamic remapping algorithm to address single and multiple processing core failures. Remapping is done by a general manager, located on a selected tile of the network.

In Valtonen, Nurmi, Isoaho and Tenhunen (2001) view, reliability problems can be avoided with physical autonomy, i.e., by constructing the system from simple physically autonomous cells. The electrical properties and logical correctness of each cell should be subject to verification by other autonomous cells that could isolate the cell if deemed erroneous (self-diagnosis is insufficient, because the entire cell, including the diagnostic unit, may be defect). In 2007, Rantala, Isoaho and Tenhunen motivated the shift from low level testing and testability design into system-level fault tolerance design. They propose an agent-based design methodology that helps bridging the gap between applications and re-configurable architectures in order to address the fault tolerance issues. They add a new functional agent/control layer to the traditional NoC architecture. The control flow of the agent-based architecture is divided hierarchically to different levels. The granularity of functional units on the lowest level is small and grows gradually when raised on the levels of abstraction. For example the platform agent at the highest level controls the whole NoC platform while a cell agent monitors and reports status of a processing unit to higher level agents. Rusu, Grecu and Anghel (2008) propose a coordinated checkpointing and rollback protocol that is aimed towards fast recovery from system or application level failures. The fault tolerance protocol uses a global synchronization coordinator recovery management unit (RMU) which is a dedicated task. Any task can initiate a checkpoint or a rollback but the coordination is done each time by the RMU. The advantages of such an approach are simpler protocol (no synchronization is needed between multiple RMUs), less hardware overhead and smaller power consumption. The drawback is the single point of failure – the dedicated RMU itself.

As a conclusion, there are various techniques to increase NoC fault tolerance but most of the research has been so far dedicated to NoC interconnects or fault tolerant routing. With technology scaling transient faults play an important role. An application running on a NoC must be aware of transient faults and be able to detect and recover efficiently from them. In Chapter 6 we propose a system-level technique to tolerate a given number of transient and intermittent faults. We have extended the shifting-based scheduling approach proposed by Izosimov (2006). We have extended it with communication fault tolerance requirements and integrated it with our contention-aware scheduling.

2.7. Network-on-chip simulators

System simulation is an important part of the system-level design flow. It allows performing design space exploration and evaluation of performance and other system metrics. There are two classes of simulators used in NoC research – established tools that have been used in computer networking research and simulators that have been developed specifically for the NoC domain. Some examples of general-purpose network simulators are OPNET, OMNeT++ and NS-2. OPNET accelerates the research and development process for analysing and designing communication networks, devices, protocols, and applications (OPNET, 2011). OMNeT++ is a discrete event simulation environment. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, it is successfully being used in other areas like the simulation of complex IT systems, queuing networks or hardware architectures (OMNeT++, 2011). NS-2 is a discrete event simulator targeted at networking research, providing support for simulation of TCP, routing, and multicast protocols over wired and wireless networks (Nework simulator NS-2, 2011). All those simulators have found their usage also in NoC domain.

In the next pages we will give an overview of a selection of network-on-chip simulators that we have used in our research. Although, these simulators target the same NoC domain we can see the diversity in their functionality and implementation details. There is always a careful analysis needed before using a new NoC simulator. This includes for example NoC architecture parameters such as topology, routing algorithm, switching method and delay model of routers. All the NoC simulators below have been interfaced with our system-level design framework. The interfacing implementation details are given in Chapter 3.

2.7.1 Noxim

Noxim is a network-on-chip simulator developed in SystemC at the University of Catania (Italy). Noxim has a command line interface for defining several NoC architecture parameters. User can customize several parameters such as network

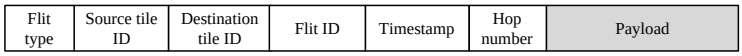
size, buffer size, packet size distribution, routing algorithm, path selection strategy, packet injection rate and different traffic distributions. The simulator allows NoC evaluation in terms of throughput, delay and power consumption. This information is delivered to the user both in terms of average and per-communication results. In detail, the user is presented with different evaluation metrics including the total number of received packets/flits, global average throughput, max/min global delay, total energy consumption and per-communications delay/throughput/energy (Palesi, Patti, & Fazzino, 2011). Noxim packet format is defined by two C++ structures *NoximPacket* and *NoximFlit*. Noxim packet consists of a specified number of flits. This determines the packet size. The abstract view of the Noxim flit format is depicted in Figure 2.12a. The payload size is 32 bits. Noxim provides a separate SystemC processing module where custom application simulation kernel logic can be implemented. Having a modular design where input configuration can be given via command line without re-compilation makes Noxim easy to use also for design space exploration (batch simulation).

2.7.2 NIRGAM

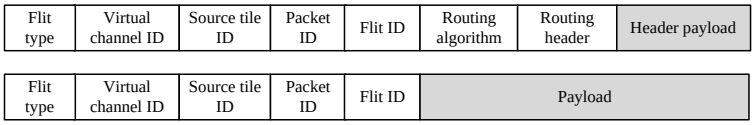
NIRGAM is a SystemC-based discrete event, cycle accurate simulator for research in network-on-chip. NIRGAM has been collaboratively developed between University of Southampton (UK) and Malaviya National Institute of Technology Jaipur (India). NIRGAM provides functionality for experimenting with different NoC topologies in terms of routing algorithms and applications (NIRGAM, 2011). It supports mesh and torus topologies. Routing algorithms include deterministic XY, adaptive odd-even and source routing. Constant bit-rate, bursty and trace-based traffic generators are included. User can configure additionally the clock frequency, buffer depths, flit size and number of virtual channels. The main configuration file captures the NoC architecture and simulation specific parameters, while the second configuration file describes the application mapping (tile ID and corresponding C++ shared library file). The simulator can output different performance metrics (latency and throughput) for a given set of choices. The abstract view of the NIRGAM flit formats are depicted in Figure 2.12b. The flit and payload sizes are configurable parameters. In similar way to Noxim the NIRGAM simulator is modular and does not require re-compilation of software when changing architecture parameters.

2.7.3 XHiNoC

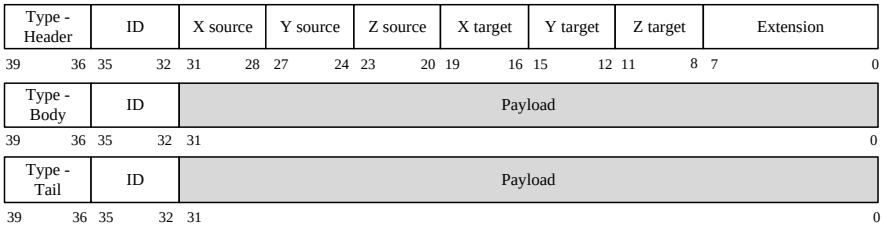
XHiNoC that stands for Extendable Hierarchical NoC is a re-configurable NoC with synchronous parallel pipeline router architecture. The XHiNoC is developed based on synthesizable modular VHDL objects at Darmstadt University of Technology (Germany). VHDL is a hardware description language for use in electronic systems design and verification (IEEE Standard 1076-2008 on VHDL language, 2008). XHiNoC provides a flexible shared communication



a) Noxim flit format



b) NIRGAM flit formats



c) XHiNoC flit formats

Figure 2.12. Flit formats of different NoC simulators

media by utilizing a unique concept of locally organized packet identity (ID) division multiple access (IDMA). Local ID slots are distributed over every communication link, which can be attached to every flit of a packet or data stream as its local ID-tag (Figure 2.12c). This switching technique has been described in more detail in Section 2.3.2. The core of the simulator is written in VHDL, while there is a SystemC wrapper on top of the XHiNoC that implements the core network interface functions such as send and receive. The router implementation in VHDL can be synthesized down to hardware to measure the area and power consumption. Different from Noxim and NIRGAM the XHiNoC co-simulation parameters cannot be specified via command line or configuration file. Each change in NoC architecture parameters requires changes in VHDL and SystemC interface files that connect together the routers/tiles. It is a tedious manual work. Fortunately, XHiNoC has been interfaced with ATLAS front-end that is able to generate the required VHDL/SystemC co-simulation files for selected NoC architecture parameters. The differences in technical parameters between the three described simulators are summarized in Table 2.1.

2.7.4 ATLAS

The ATLAS environment automates the various processes related to the design flow for some of NoCs that are part of the framework. Currently the ATLAS framework supports Hermes, Mercury and XHiNoC NoC implementations. The ATLAS design flow is composed by the following stages: NoC generation, traffic generation, simulation, performance and power evaluation. In the NoC

generation, the NoC parameters such as channel bandwidth, buffer depth, number of virtual channels, flow control strategies are configured. In the traffic generation, the traffic scenarios are generated to characterize the applications which execute on the NoC. In the simulation data is injected to the network according to generated traffic scenarios. In the performance evaluation, ATLAS can generate graphics, tables, maps and reports to help in the analysis of obtained results. The power evaluation gives estimates on NoC power usage (ATLAS, 2011).

2.8. Summary

In this chapter we have addressed several system-on-chip design challenges that have emerged together with technology scaling and due to increase of the design complexity. The main challenges are design productivity, system synchronization and global wiring. Wires do not scale as rapidly as transistors. Interconnect wires account for a significant fraction (up to 50%) of the energy consumed in an integrated circuit and is expected to grow in the future. Interconnects also add a new dimension to the design complexity. As interconnects shrink and come closer together, previously negligible physical effects like crosstalk become significant. Local wires scale in performance while global and fixed-length wires do not. When the number of components increases rapidly, we have a situation where the clock signal cannot be distributed over the entire SoC during one clock cycle. Network-on-chip is one of the possibilities to overcome some of the on-chip communication scalability problems. In NoC-based systems the communication is achieved by routing packets through the network infrastructure rather than routing global wires. To design an on-chip communication infrastructure and to meet the performance requirements of an application, a designer has certain design alternatives that are governed by topology, switching, routing and flow control of the network. In this chapter we have given an overview of the key concepts and terminology of NoCs. We have presented the classical system-level design flow and have described some of the well-known scheduling and mapping techniques. Communication parameters (inter-task communication volume, link latency and bandwidth, buffer size) might have major impact to the performance of applications implemented on NoCs. We have given an overview of the key research challenges of NoC-based systems. We have described the issues related to dependability of SoCs and gave a survey of different methods to increase fault tolerance of these systems. The last part of this chapter has been devoted to NoC simulators.

Table 2.1. Comparison of different NoC simulators

	Noxim	**NIRGAM**	**XHiNoC (+ ATLAS frontend)**
Topology	mesh	mesh, torus	mesh
Routing	XY, west-first, north-last, negative-first, odd-even, dyad, fully-adaptive, source routing	XY, odd-even, source routing	XY
Switching	wormhole	wormhole	IDMA
Traffic types	random, transpose1, transpose2, bit-reversal, butterfly, shuffle	constant bit rate, bursty, trace-based	random, hot-spot, custom traffic rules
Configurable options	buffer depth, packet injection rate, warmup period	clock frequency, buffer depth, flit size, number of virtual channels	Flit-width, buffer-depth
Performance analysis	total number of received packets/flits, global average throughput, max/min global delay, total energy consumption, per-communication statistics	average latency per packet, average latency per flit, average throughput (in Gbps) – all reported separately for each channel	throughput, latency distribution graphs, link analysis, latency analysis

Chapter 3. System-Level Design for NoC-Based Real-Time Systems

In the previous chapter an introduction to the classical system-level design methodology together with an overview of the design and dependability issues of NoC-based systems-on-chip has been given. The purpose of this chapter is to address in more detail the communication modelling problem of NoC-based real-time systems. First, we give a motivation why the inter-processor communication (IPC) needs to be modelled in NoCs. Second, we introduce our system-level design flow. We provide a description and definition of the system model and formulate the design problem. Next, we describe the communication synthesis method and demonstrate how it is being used during application scheduling. The last part of the chapter explains how our system-level design framework is interfaced with the XHiNoC simulator, followed by the experimental results.

The main results of this chapter have been published in the following papers:

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2011). Contention-aware scheduling for NoC based systems. *Proceedings of the 29th Norchip conference*, (pp. 1-4). Lund, Sweden.

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2011). Communication modelling and synthesis for NoC-based systems with real-time constraints. *Proceedings of the 14th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems* (pp. 237 - 242). Cottbus, Germany.

3.1. Motivation

If inter-processor communication would take zero cycles we could exploit all available parallelism at minimum cost. In reality, the performance of applications implemented on NoCs is influenced heavily by different NoC communication parameters, such as inter-processor communication volume, link latency, bandwidth and buffer size. There is a trade-off between the amount of parallelism utilized and overhead of the network communication. Keeping tasks mapped on a few processing cores reduces the amount of inter-processor communication but increases the schedule length. On conversely, mapping tasks on distant processing cores only to utilize the parallelism may not be the optimum solution. On top of the communication delay a message transfer in the network could experience unexpected network contentions. Therefore, in order to guarantee predictable behaviour and to satisfy performance constraints of

real-time systems, a careful selection of application partitioning, mapping and synthesis algorithms is required.

Several authors have investigated communication behaviour in NoC-based systems and considered average or worst case communication delay. Papers by Lei and Kumar (2003) and Marcon, Kreutz, Susin, and Calazans (2005) approximate the communication delay without considering the congestions. In the paper by Shin and Kim (2004) the worst case communication delay gets estimated by assuming that the worst case communication delay at each link is the time required to transfer all communication loads assigned to the link. As authors state, it is a pessimistic communication delay model. There are also papers that use architectural or network level means to hide the communication modelling complexities. In 2006, Stuijk, Basten, Geilen, and Ghamarian described a scheduling problem of time-constrained communication of a streaming application on a NoC. The approach uses time division multiple access (TDMA) channel access method with wormhole switching and tries to minimize resource (TDMA slot table) usage while guaranteeing communication latency. Complexity of maintaining TDMA slot table for each link and for any time interval is not addressed. Such an approach requires having more complex routers with TDMA support. In 2008, Shim and Burns proposed a priority-based pre-emptive arbitration with virtual channels and wormhole switching. This scheme allows authors to calculate upper bounds of network delay by considering direct interference from higher priority traffic-flows and indirect interference where two traffic-flows do not share any physical link but there is (are) intervening traffic flow(s) between the given two traffic-flows. This analysis gives pessimistic estimates that are used to guarantee delays of hard real-time traffic. The work of Lu (2007) similarly considers direct and indirect interference by building a congestion tree. An algorithm was developed by the author to estimate worst-case performance of real-time messages and to conduct the feasibility analysis. The analysis returns the pass ratio, i.e., the percentage of feasible messages, and network utilization of the feasible messages. Wormhole switching with virtual channels is a requirement. The author assumes that there is sufficient number of virtual channels available to service high-priority communication. Additionally, the pipeline latency is simplified on links so that the flits of a message will reserve all the link bandwidth along the message path simultaneously for the whole duration of its communication time. It can produce more pessimistic schedule as link bandwidth is reserved even when the virtual channel setup/tear-down is done gradually link-by-link. The work of Millberg, Nilsson, Thid, and Jantsch (2004) describes virtual channels implemented using a combination of looped containers and temporally disjoint networks (TDN). TDNs are results of the topology of the network and the number of buffer stages in the switches. Packets residing in the neighbouring time/space slots could be seen as being in different networks and therefore disjoint. Instead of scheduling the communication it is disjoint spatially. As the communication might not temporally conflict with each other this approach can possibly have waste of resources such as link bandwidth and switch buffers.

Our proposed method is an extension of the edge scheduling approach described for distributed systems by Sinnen and Sousa (2005). A similar approach, where communication edges of a task graph are mapped to the physical network topology can be found also in the NoC domain (Hu & Marculescu, 2005; Manolache, Eles, & Peng, 2007). In 2005, Hu & Marculescu have described in their scheduling heuristic a sub-procedure that calculates the data ready time of the tasks. According to presented pseudocode, a communication transaction is always scheduled to the earliest possible start time while reserving the whole path. In real network-on-chip implementations, link reservation and tear down is an incremental process. Moreover, on a routing path a message can experience network contentions delaying delivery of the data. It will lead to the communication schedule described by Sinnen & Sousa in 2005 as a nonaligned approach. The downside of the nonaligned approach is that it requires routers to have schedule tables so that they could reserve the needed bandwidth in advance for upcoming traffic. Otherwise, another communication, starting earlier and sharing the same communication link, could take the full bandwidth and invalidate possibly the rest of the schedule. In our approach both of the problems are considered.

Manolache, Eles, & Peng have proposed in 2007 a task graph extension with detailed communication dependencies employing virtual cut-through switching with deterministic dimension-order XY routing. We have generalized the proposed approach and made it compatible with different switching methods such as store-and-forward and wormhole switching. In contrast to many published work on NoC communication scheduling, our approach is minimalistic in a sense, that it will operate on a resource minimized NoC without virtual channels. Since virtual channels have been turned out to be extremely area consuming, for many applications using them is no option and much more hardware-efficient router architectures are required in order to come up with cost-efficient solutions (Samman, Hollstein, & Glesner, 2008).

3.2. System-level design flow and definitions

We are using a classical system-level design flow that we have extended to include the aspects of NoC communication modelling (Figure 3.1). Input to the system-level design flow is an application *A*, a NoC architecture *N* and an application mapping *M*. Important parts of the design flow are the communication synthesis and scheduling, that are performed together with task scheduling. The resulting schedule is verified by executing the application on the XHiNoC simulator. If simulation results do not correspond to scheduling results refinement of the NoC delay model may be needed. Optionally, we can perform a design optimization loop trying to improve the schedule and/or the task mapping.

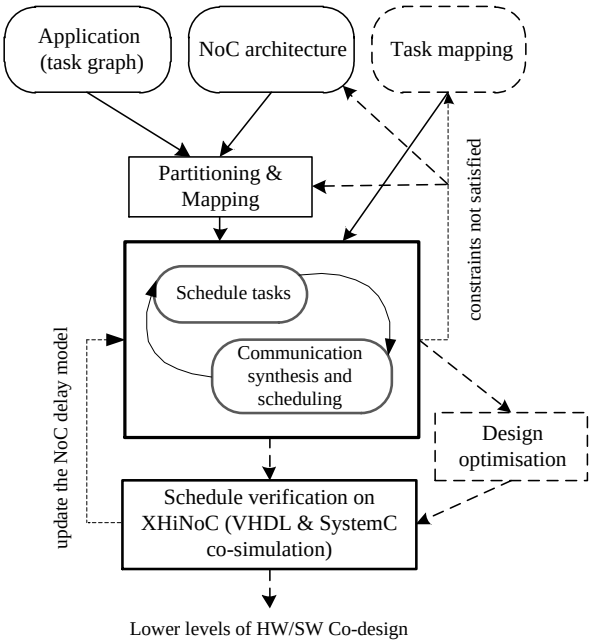


Figure 3.1. System-level design flow

Definition 3.1 (Task Graph). *An application A is specified as a directed acyclic graph A = (T, C, wcet, comm), where* $T = \{t_i \mid i = 1,\ldots,T\}$ *is set of vertices representing non-preemptive tasks and* $C = \{c_{i,j} \mid (i,j) \in \{1,\ldots,V\} \times \{1,\ldots,V\}\}$ *is a set of edges representing precedence constraints between tasks. Each task* t_i *is characterized by the Worst Case Execution Time (WCET)* $wcet_i$*, while each edge carries the communication cost* $comm_{i,j}$ *between the tasks* t_i *and* t_j*.*

Task WCET can be extracted from application execution trace or obtained via static analysis. The paper by Wilhelm, et al. (2008) describes various methods in detail. In this thesis we assume that the task graphs together with the computation and communication volumes are given. The set $\{t_p \in V : c_{p,i} \in C\}$ of all direct predecessors of t_i is denoted by $pred(t_i)$ and the set $\{t_s \in V : c_{i,s} \in C\}$ of all direct successors of t_i is denoted by $succ(t_i)$. A vertex without predecessors ($pred(t) = \varnothing$) is named source node and vertex without successors ($succ(t) = \varnothing$) is named sink node. We assume that the application has dummy source and sink vertices. Both vertices have $wcet = 0$. An edge $c_{i,j}$ that connects two tasks t_i and t_j represents either control flow dependency when the communication cost (message size) $comm_{i,j} = 0$ or communication when the $comm_{i,j} > 0$. However, if two tasks having control flow dependency are mapped to different computing resources, a message is sent from t_i to t_j to trigger its execution. Figure 3.2 depicts task graph representation of a robot control application extracted and visualized from MCSL benchmark suite (Liu, et al., 2011). The application consists of 88 tasks and has 131 edges.

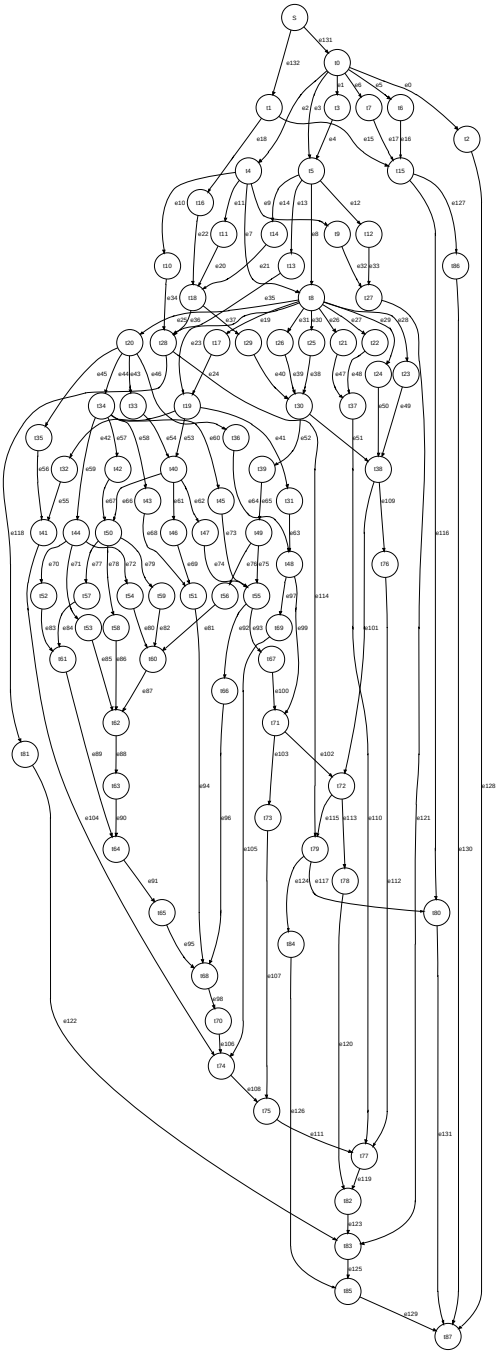


Figure 3.2. Task graph of a robot control application

Definition 3.2 (Network model). *The NoC architecture is modelled as a directed graph N = (R, L) where R = {r_k | k = 1,…,R} is a set of vertices (resources) and L = {$l_{k,l}$ | (k,l)∈ {1,…,R} x {1,…,R}} is a set of directed edges (communication links) connecting a pair of resources (k,l).*

An example of a 3x3 2D-mesh NoC architecture graph is depicted in Figure 3.3. As such network model captures the connectivity and routing possibilities between the nodes it is also referred to as topology graph in the literature (Sinnen & Sousa, 2005). Resources in the network model are routers, network interfaces and homogenous computational cores. In the NoC topology graph we do not distinguish separately between a network interface and a computational core – they are represented as one vertex. Communication links are all bi-directional, each direction having a separate edge in the network graph. The architecture is characterized by topology, routing algorithm, switching method and the delay model of the network components. The network delay model described in this work comprises of the router processing delay D_{router}, the channel transmission delay $D_{channel}$ and the path setup delay D_{setup}. The routing path from r_k to r_l is denoted by $r_{k,l}$ and $Z(r_{k,l})$ is set of links that make up the path $r_{k,l}$.

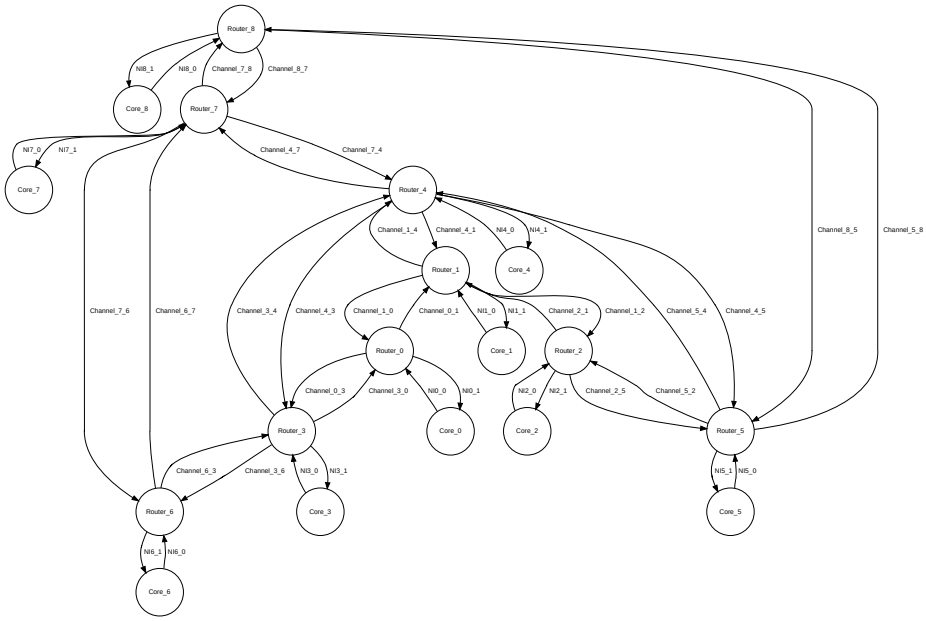


Figure 3.3. A 3x3 2D-mesh network-on-chip graph representation

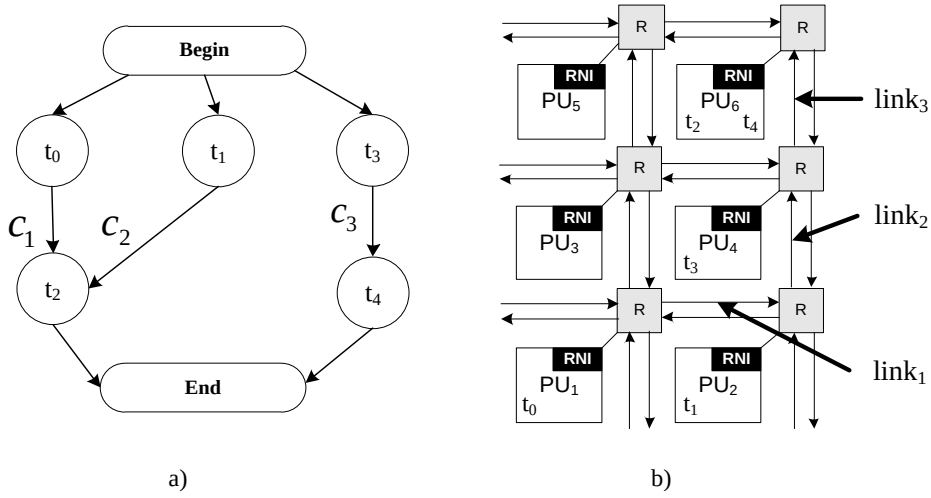


Figure 3.4. An example task graph and its mapping to a 2x3 2D-mesh NoC

Definition 3.3 (Mapping). *Function $M(T \rightarrow R)$ associates each task t_i from the set $T = \{t_i \mid i = 1,...,T\}$ with a NoC computing resource r_k that is selected from the set $R = \{r_k \mid k = 1,...,R\}$.*

An example task graph consisting of 5 tasks and its mapping to a 2x3 mesh NoC is depicted in Figure 3.4. Task mapping can be given as input to the system-level design flow or we can perform design space exploration trying to find a mapping that satisfies the design constraints. Applying design optimization techniques to our system model are described in more detail in Chapter 5.

Definition 3.4 (Extended Task Graph). *Given an application A, a network model N and a mapping function M the result of the communication synthesis is an extended task graph (ETG) where the communication edges between the tasks are transformed into a sequence of nodes (communication sub-graph) representing the flow control units.*

For each communication link the message is passing a new vertex is added into the extended task graph. There is an example ETG depicted in Figure 3.5. The communication synthesis process is explained in more detail in Chapter 3.4.

Assumptions on architecture

Our work focuses on the system-level design of network-on-chip based real-time systems-on-chip. The results produced by the system-level methods must be predictable and meet the design requirements. To guarantee the predictability and to handle system-level modelling complexity, we assume throughout this

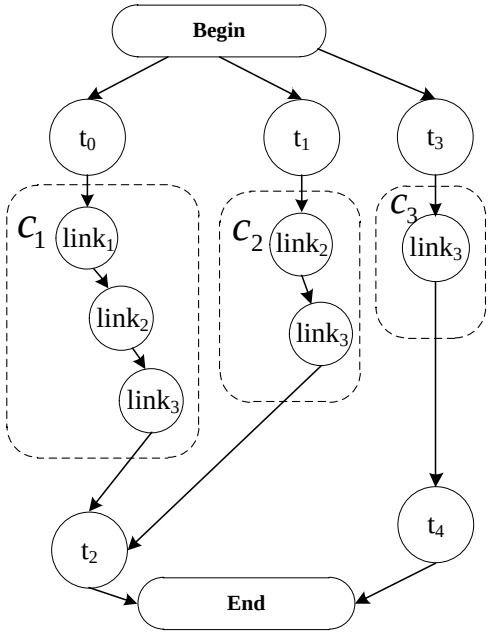


Figure 3.5. Extended task graph

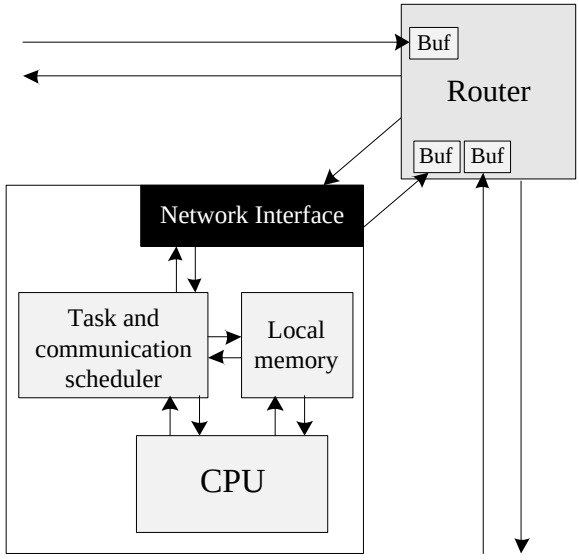


Figure 3.6. Local processing core

thesis the following NoC architecture. In our system model we assume that each processing core is controlled by a scheduler that takes care of the task execution on the core and schedules the message transfers between the tasks. Otherwise a task that completes earlier than its WCET and starts a message transfer could lead to an unexpected network congestion and have a fatal effect on the global execution schedule.

We are employing static scheduling to obtain performance characteristics of an application and resulting schedule. It is important to note that in this work our main focus is not on the schedulability analysis but on a method for fast and efficient system analysis. The full application schedule is calculated offline. A partial schedule that consists of events related only to a specific processing core is stored in each local scheduler memory. Figure 3.6 depicts an extract of a NoC router and its connected core.

We assume that the size of an input buffer is one flow control unit (packet/flit). No output buffering is used. The input buffer of one flow control unit together with its incoming communication link can be considered as one shared resource during communication synthesis. Head-of-line blocking and deadlocks are avoided by construction – communication messages are scheduled to avoid resource conflicts and messages are injected into the network deterministically, only at respective scheduled time moments. As mentioned in Chapter 2.3.3, adaptive routing provides features such as load-balancing and increased failover. However, it is more hard to guarantee deadlock freedom and performance constraints. To have a predictable communication model we assume the use of deterministic routing algorithms. In our experiments we are using dimension-order XY routing.

3.3. Sources of network contention

The NoC platform introduces communication latency that depends not only on the message size but also on the resource mapping and needs to be taken into account. It means that communication modelling must take into account both spatial and temporal information. Otherwise one will not be able to analyse the impact of network conflicts on application temporal behaviour. Since virtual channels have been turned out to be extremely area consuming, for many applications using them is no option and much more hardware-efficient router architectures are required in order to come up with cost-efficient solutions (Samman, Hollstein, & Glesner, 2011). Furthermore, we assume to have a NoC architecture with best-effort services without packet interleaving. As such communication infrastructure is hardware efficient but does not guarantee communication delays it is important for a predictable design flow to explicitly synthesize and schedule the communication. A communication schedule could be extracted by simulating the application on a NoC simulator, but the simulation speed will be the limiting factor.

A message transfer between two communicating tasks goes through a set of stages at which network resources must be acquired. Lack of available resources at any point of time causes network contentions. We can distinguish two types of contentions (Sinnen & Sousa, 2005):

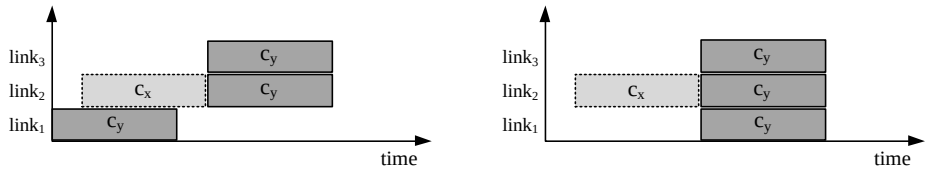
- **end-point contention** – refers to the contentions in network interfaces. Only a limited number of communications can pass from the processing core into the network and vice versa;
- **network contention** – is caused by the limited number of resources within the network. To successfully handle this kind of contention in scheduling, an accurate model of the NoC is required.

All limited network resources (network interface, routers) are part of our system model, therefore we are able to capture both the end-point and the network contentions.

The proposed communication scheduling method is similar to the edge scheduling approach described for distributed systems by Sinnen and Sousa (2005). In edge scheduling, communication resources are treated like processors in the sense that only one communication can be active on each resource at a time. Thus, edges are scheduled onto the communication links for the time they occupy them. Sinnen and Sousa (2005) also describe two possible alternatives for determining scheduling times on a route with contentions:

- **nonaligned approach** – on link $link_2$, c_y is delayed until the link is available (Figure 3.7a);
- **aligned approach** – alternatively, c_y can be scheduled later on all links (Figure 3.7b). For example it starts on all links after communication c_x finishes on $link_2$.

According to Sinnen and Sousa the nonaligned approach allows determination of the start and finish time successively for each link on the route. The downside of the nonaligned approach is that it requires routers to have schedule tables so that they could reserve the needed bandwidth in advance for upcoming traffic. Otherwise, another communication, starting earlier and sharing the same communication link, could take the full bandwidth and invalidate possibly the rest of the schedule.



a) nonaligned view b) aligned view

Figure 3.7. Edge scheduling on a route with contention.

In our proposed approach communication is modelled as part of the extended task graph. It has several advantages – there is no need for a separate communication model and communication is embedded in a natural way into the task graph. Communication synthesis and scheduling can be applied directly on the extended task graph model. Moreover the ETG carries scheduling data that is needed in later phases to output the schedule for simulation.

3.4. Communication synthesis

As already mentioned in the introduction of this chapter, communication synthesis plays an important role in our system-level design flow. In short, it transforms a regular task graph edge into a communication sub-graph based on the communication links a message traverses. We get an extended task graph containing information about both tasks and communications, enabling uniform treatment of computation and communication. In this section we describe the communication synthesis in detail.

In Figure 3.8, an example task graph (Figure 3.8a) and its mapping onto five processing units (Figure 3.8b) has been presented. Task t_0 is mapped onto PU_1, t_1 onto PU_2 etc. It can be seen in Figure 3.8b that communication c_1 (from t_0 to t_2) takes three links ($link_1$, $link_2$, $link_3$) while c_2 (from t_1 to t_2) takes two links ($link_2$, $link_3$). For the sake of simplicity the communication link between the network interface and the router is omitted in this example but always included in the real model. The physical links, which the communication traverses, are shared resources. It means that in addition to calculating the communication latencies we need to have a method to take into account the network conflicts as well. To determine the schedule length for a given mapping the communication needs to be synthesized and scheduled together with the tasks.

For each communication link the message is passing a new vertex is added into the extended task graph (Definition 3.4 in Section 3.2). Figure 3.8 depicts the synthesis process for the communication c_1 (between tasks t_0 to t_2). The communication passes three links ($link_1$, $link_2$, $link_3$). In Figure 3.8c the communication c_1 is on $link_1$ and this is represented in the extended task graph by adding a corresponding vertex. In the next step, depicted in Figure 3.8d, the communication c_1 has advanced to $link_2$. A corresponding vertex is added into the extended task graph that captures also the precedence constraint of the messages on the communication links. We cannot schedule the message on $link_2$ before it has been scheduled on $link_1$. The process is repeated until the whole routing path of the message has been traversed and complete communication sub-graph built.

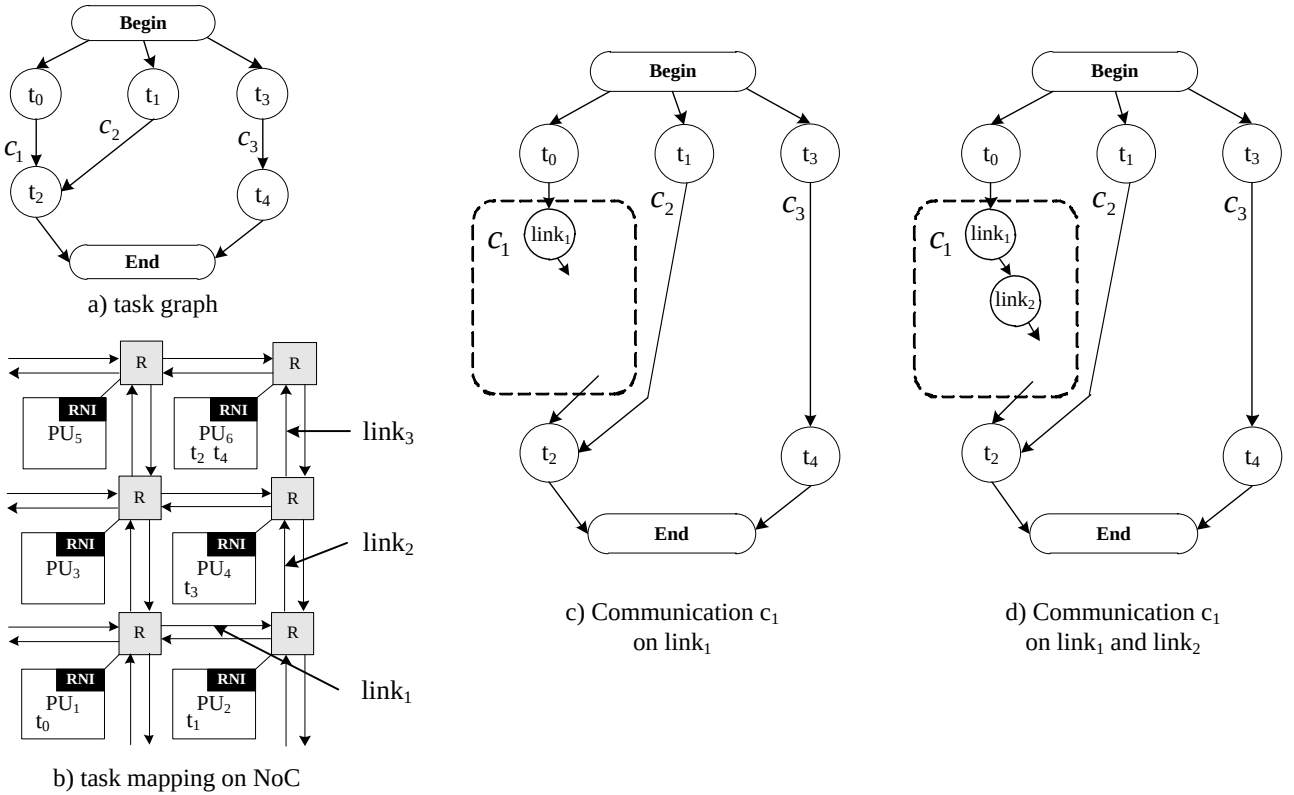


Figure 3.8. Two example steps of communication synthesis

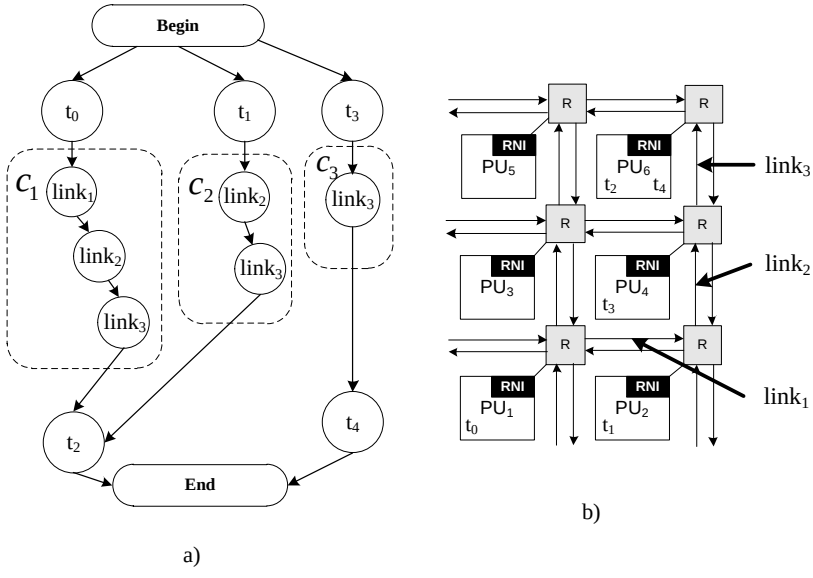


Figure 3.9. Fully synthesized task graph and its mapping

Figure 3.9a depicts the result of the communication synthesis process for communications c_1, c_2 and c_3. Those communication vertexes are scheduled together with the tasks. The communication can be represented in the extended task graph on various granularity levels. In the previous example a vertex in the ETG represented a whole message. In this case the graph complexity depends on the number of tasks, routing algorithm R_{algo}, NoC size and mapping. This is represented by the function $G_{complexity} = (A, N, M, R_{algo})$. An analysis is given in the experimental results (Chapter 3.7.1) on the scaling of the approach for different application and NoC sizes. A vertex that is being added into the task graph during communication synthesis could represent in a similar way also a packet or a flit. It gives more flexibility during scheduling at the cost of increased task graph size. Chapter 4.1 analyses such trade-off in more detail.

3.5. Contention-aware scheduling

Scheduling determines the order of execution of tasks on processing cores. While a mapping captures a task spatial assignment, the scheduling captures the temporal behaviour of an application by assigning each task a start time (t_{start}). The result of the process is a schedule S that is characterized by the schedule length S_{length} (also referred to as makespan). Schedule length is the maximum time from all processing cores required to finish the task execution. In our work a schedule is called feasible if it satisfies all precedence and task WCET constraints.

Definition 3.5 (Contention-aware scheduling problem). *Given an application A = (T, C, wcet, comm), a network model N = (R, L) and a mapping function M(T→R) our goal is to take into account network resource conflicts and schedule both – communication messages and tasks. The resulting schedule S must be feasible and cycle-accurate to be executed on a NoC simulator.*

Our proposed approach can be used with an arbitrary scheduling algorithm, although in this work we have chosen list scheduling that has been enhanced to support the communication model. List scheduling is a greedy heuristic using a priority list and precedence constraints to schedule the tasks and minimise the schedule length. List scheduling is simple to implement and it has relatively short schedule calculation time.

The pseudocode of contention-aware scheduling algorithm is depicted in Figure 3.10. We start by calculating task priorities (line 1) and add the source task into the ready task list. As a *priority function* for the list scheduling we are calculating the task mobility Mob_i that is difference between task ASAP (As-Soon-As-Possible) and ALAP (As-Late-As-Possible) schedule. We sort the ready task list by mobility (line 5) and schedule a ready task (line 6). We capture the messages (line 7) that need to be transferred. We insert eligible successor tasks into ready-to-schedule task list (line 8) and remove the scheduled task from the list (line 9). After all ready tasks have been scheduled and outgoing communication captured, we will sort the messages by their ASAP schedule (line 11) and start the communication synthesis and scheduling for the messages in the list. The process is repeated until all tasks and communication messages

```
Contention-aware scheduling
1     calculate task priorities
2     add task with pred(t) = ∅ into ready task list
3     while tasks not scheduled ≠ ∅
4       for each task in ready task list do
5         sort ready task list based on mobility
6         schedule ready task
7         add first vertex of outgoing communication(s) into comm. list
8         add eligible successor tasks to ready task list
9         remove scheduled task from ready list
10      end for
11      sort communication list based on ASAP schedule
12      for each communication message in comm. list do
13        schedule communication message on the whole route
14        add eligible successor tasks to ready task list
15        remove scheduled message from comm. list
16    end for
17    end while
End Contention-aware scheduling
```

Figure 3.10. Pseudocode of contention-aware scheduling

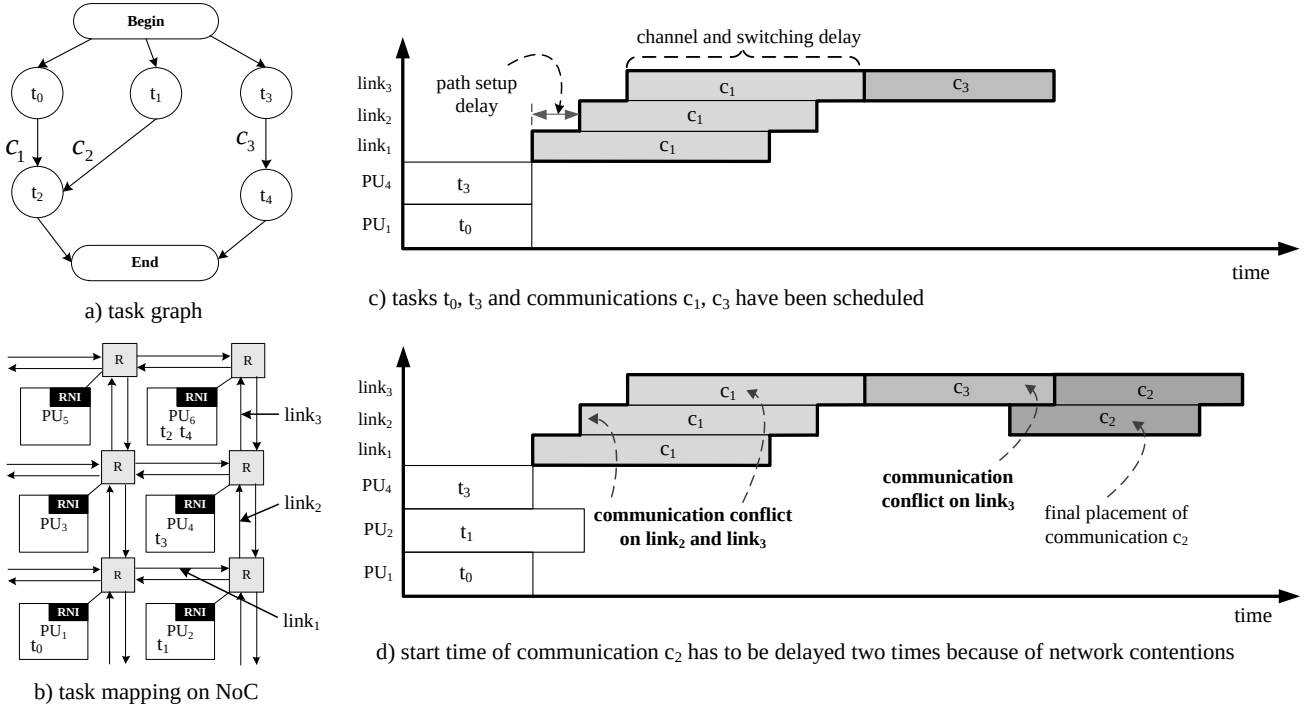


Figure 3.11. An example of contention-aware communication scheduling

have been scheduled. In the next sections we will explain in more detail how the communication is scheduled and how the network contentions are handled. Additionally, we will describe two modelling aspects that affect the schedule length.

Communication scheduling with network resource handling

One of our goals is to take into account the network conflicts that occur due to limited resources in the network. The next example illustrates the contention-aware scheduling process, having focus on communication scheduling and describes how the resource conflicts are handled.

Figure 3.11a depicts a task graph with respective mapping in Figure 3.11b. In Figure 3.11c tasks t_0, t_3 and communications c_1 and c_3 have been scheduled. In Figure 3.11d we have scheduled the task t_1 and we start to schedule the communication c_2. Usually we would schedule the communication right after the task has finished its execution. We can see in Figure 3.11d that for c_2 there is a communication conflict in $link_2$ and $link_3$. We would have to delay start of the communication. Even if we delay the start time of c_2 in $link_2$ there will be another conflict between c_2 and c_3 on $link_3$. Therefore, we need to delay the c_2 on $link_3$ to wait for available channel bandwidth. This is done by finding the maximum schedule time on $link_3$ and scheduling the communication $c_2^{link3} = max(link_3^{time})$. After the communication c_2 has been scheduled on $link_3$ the schedule start time of the same message on $link_2$ need to be updated respectively. It is needed to avoid the problem of nonaligned approach described in the beginning of this chapter. The communication delay (ci^{CD}) of a message is calculated based on the following formula:

$$c_i^{CD} = (D_{router} + D_{channel}) \times number\_of\_flits \qquad (5)$$

where D_{router} is the router switching delay and $D_{channel}$ the channel delay. The communication start time on the next link is represented by the formula:

$$c_{i\ next}^{start\ time} = c_i^{end\ time} + D_{setup} \qquad (6)$$

where D_{setup} is the path setup delay. After the communication has been scheduled on the links the formula ($c_i^{endtime} - c_i^{starttime}$) gives us the total communication delay of c_i.

The pseudocode of the communication scheduling process is summarized in Figure 3.12. It is a detailed description of the line 13 in the main contention-aware scheduling algorithm, depicted in Figure 3.10. Input to the procedure is a task graph edge between two vertices. The edge is synthesized into communication sub-graph (line 1) according to the communication links the message traverses. Once the input edge is replaced by a number of communication vertexes (that are mapped to the communication links) the communication scheduling starts. First, we get the schedule time of the communication link where the ready-to-schedule communication vertex is mapped. Second, we find the maximum schedule time from predecessor vertexes

(lines 5-13). If the predecessor vertex is a task then we schedule the communication at the end of the task execution. If the predecessor vertex represents communication we will take into account the path setup delay and shift the start of the communication accordingly. We will determine the final communication start time by comparing the schedule time from communication link and predecessor vertexes and choose the maximum (lines 15-21). If we faced a network contention on the current communication link and delayed the start time of the vertex then we need to update the communication start time of all related predecessor communication vertexes (line 22). It is needed to avoid the problem of non-aligned approach described in the beginning of this chapter.

Figure 3.13 depicts the resulting schedule of the example application (Figure 3.11a). We can see that if we would not have taken into account the network conflicts we would have got a shorter schedule length that could have led to unexpected results during the application execution.

```
ScheduleCommunication(e)
1  first vertex of sub-graph  = transform communication edge e  into sub-graph
2  add into readyToSchedule list the first vertex of communication sub-graph
3  while readyToSchedule ≠ ∅, i = 0 do
4     linkSchedTime = get max schedule time from comm. link the vertex c_i is mapped
5     for each incoming edge i_j of vertex c_i do
6        predecessor vertex c_j^pred = get source vertex of edge i_j
7        if (c_j^pred is a regular task) then
8           maxTimeFromPredecessor = schedule end time of c_j^pred
9        else
10          maxTimeFromPredecessor = get comm. start time on previous link + routing
11          delay
12       end if
13    end for
14
15    if linkSchedTime <  maxTimeFromPredecessor then
16       commStartTime =  maxTimeFromPredecessor
17    else
18       commStartTime = linkSchedTime
19    end if
20
21    commEndTime = commStartTime + communication delay of c_i
22    back annotate schedule end time for predecessor comm. vertexes if needed
23    add successor vertexes and remove scheduled message c_i from readyToSchedule
24 end while
end ScheduleCommunication
```

Figure 3.12. Pseudocode of communication scheduling

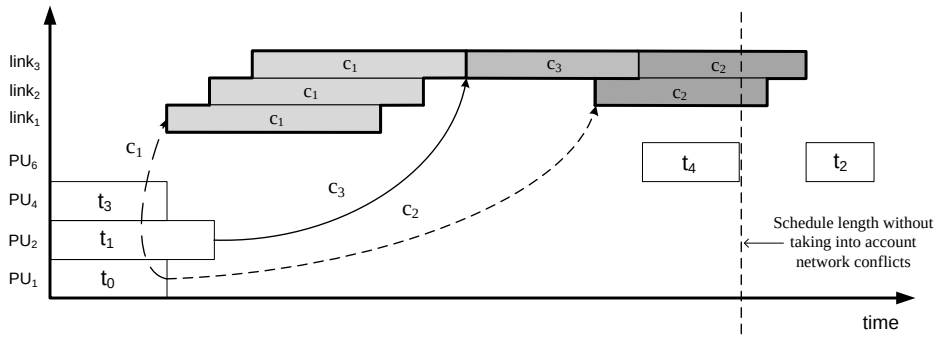


Figure 3.13. Resulting schedule

Priority function

The quality of schedules produced by list scheduling depends on the priority function. To calculate the ASAP and ALAP schedule we perform *unconstrained scheduling* that is known term in high-level synthesis. In general, we allocate starting time of tasks under the assumption that unlimited amount of resources are available. It will help us to compute the earliest time (lower bound) and the latest time (upper bound) a task can start its execution. While in classical ASAP and ALAP algorithms communication cost is omitted we have tried to include it in our modified algorithm. We assume that network resources are unconstrained. It means that the network has enough bandwidth to process the messages without contentions. Depending on application mapping, routing algorithm and switching method we can count the number of hops and calculate the communication latency without contentions.

The pseudocode of ASAP algorithm with communication is given in Figure 3.14. We start from the source node and move towards the sink node. The algorithm starts with finding the source node that has no predecessors. The source node is assigned the starting time 0 and added into the ASAP schedule. Then we perform a set of operations (lines 4-15) in a loop until all tasks have been scheduled. We find a task whose predecessors are all scheduled. We loop through all the predecessor tasks to find the earliest time we could schedule the current task. During that process we calculate the communication delay (lines 7-9) between the task and its predecessor if they are mapped on different processors. We store the earliest start time (lines 10-13) that we have found taking into account predecessor task end time and communication delays.

The pseudocode of ALAP algorithm with communication is given in Figure 3.15. In contrast to ASAP algorithm we start from the sink node and move towards the source node. The algorithm starts with finding the sink node that has

ASAP with communication (task graph *A*)

```
1     task t = get source task without predecessors (pred(t) = ∅)
2     schedule_ASAP(t) = 0; A = A - t
4     while A ≠ ∅
4        get task t_i whose predecessors are all scheduled
5        WCET_max = 0
6        for each predecessor task t_j of task t_i described by edge e_j do
7           if (e_j^communication size > 0 and t_j^task mapping != t_i^task mapping)
8              communication delay CD_j = calculate communication delay(e_j)
9           end if
10          if (t_j^start time + t_j^WCET + CD_j) > WCET_max)
11             WCET_max = t_j^start time + t_j^WCET + CD_j
12          end if
13       end for
14       schedule_ASAP(t_i)  = WCET_max
15       A = A - t_i
14    end while
End ASAP with communication
```

Figure 3.14. Pseudocode of ASAP scheduling with communication

ALAP with communication (task graph *A*)

```
1     task t = get sink task without successors (pred(t) = ∅)
2     schedule_ALAP(t) = schedule_ASAP(t) + t^WCET; A = A - t
3     while A ≠ ∅
4        get task t_i whose successors are all scheduled
5        WCET_max = ∞
6        for each successor task t_j of task t_i described by edge e_j do
7           time_max = schedule_ALAP(t_j)
8           if (e_j^communication size > 0 and t_j^task mapping != t_i^task mapping)
9              communication delay CD_j = calculate communication delay(e_j)
10          end if
11          time_max = time_max - CD_j
12          if (time_max < WCET_max)
13             WCET_max = time_max
14          end if
15       end for
16       schedule_ALAP(t_i)  = WCET_max - t_i^WCET
17       A = A - t_i
18    end while
End ALAP with communication
```

Figure 3.15. Pseudocode of ALAP scheduling with communication

no successors. The sink node is assigned the maximum starting time, which is sum of its minimum starting time (ASAP schedule) and its WCET (line 2). Then we perform a set of operations (lines 4-17) in a loop until all tasks have been scheduled. We find a task whose successors are all scheduled. We loop through of its successor tasks to find the latest time we could schedule the chosen task. During that process we take into account the communication delay (lines 8-10) between the chosen task and its successor if they are mapped on different processors.

Finally we calculate the *mobility* Mob_i for each task t_i that is $Mob_i = ALAP(t_i) - ASAP(t_i)$.

To see how much influence our modified ASAP and ALAP algorithm has on application schedule length we have performed experiments with ten synthetic task graphs having varying average vertex degree and computation-to-communication ratio (CCR) mapped on a 6x6 2D-mesh NoC. Figure 3.16 depicts the difference in list schedule length for two experiment sets where task priorities are calculated using the classical ASAP and ALAP and our modified algorithm. Even our modified ASAP and ALAP algorithm modifies the task mobility values (priorities) causing possibly different order of tasks in the schedule it does not have as much effect as we have expected. However, it can have more influence when the application is communication dominated.

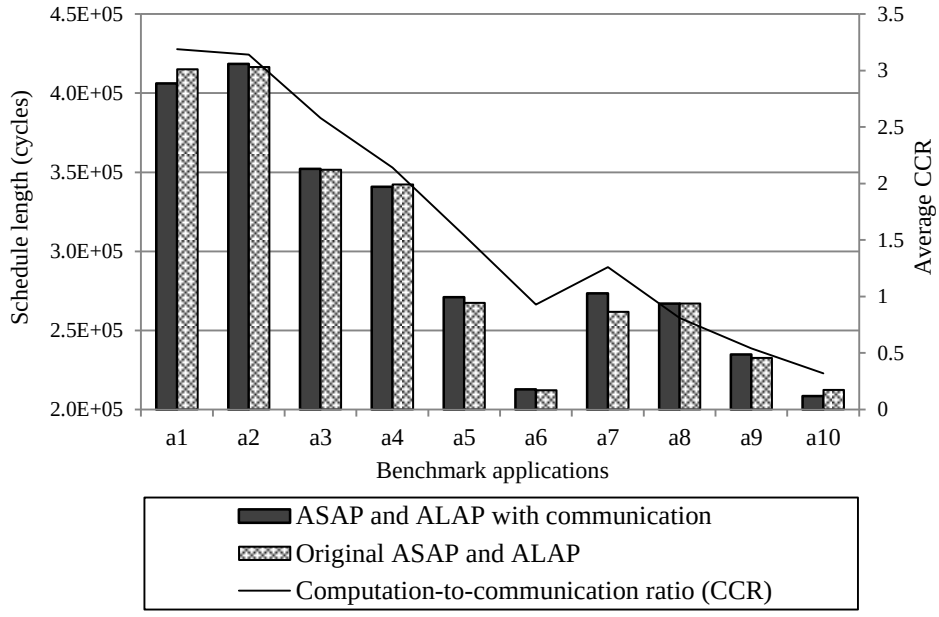


Figure 3.16. Schedule length comparison of original and extended ASAP and ALAP

Schedule holes

During scheduling, tasks that have higher priority are scheduled first. However, tasks take typically different amount of time to execute. As scheduling is performed incrementally we have multiple options when to schedule the outgoing messages of tasks. If we would schedule a task and then all the related messages then we could have a situation where the network is idle because of another task that has a higher priority and a longer execution time. In this section we will describe the problem of schedule holes and propose a solution.

There is an example of such situation depicted in Figure 3.17a. The task t_2 has a higher priority than task t_1. The communication c_2 has been scheduled to the end of the task t_2 execution (Figure 3.17a). In the next step the task t_1 is scheduled and while the communication c_2 is already in the schedule then the communication c_1 is scheduled at the end of the current schedule (Figure 3.17a). In Figure 3.17b the problem is solved by first scheduling the tasks, secondly sorting the messages by the ASAP schedule and then scheduling the messages. The messages are sorted to increase the network utilization and to avoid *schedule holes* (idle timeslots) that are caused by incremental scheduling.

To show the possible improvement we have made similar experiments as described previously in Figure 3.16 with ten synthetic task graphs having varying average vertex degree and computation-to-communication ratio, mapped on a 6x6 2D-mesh NoC. The results are depicted in Figure 3.18. It can be seen that we are able to get shorter schedules for all the ten task graphs by scheduling first the ready tasks and then the communication messages sorted by the ASAP schedule.

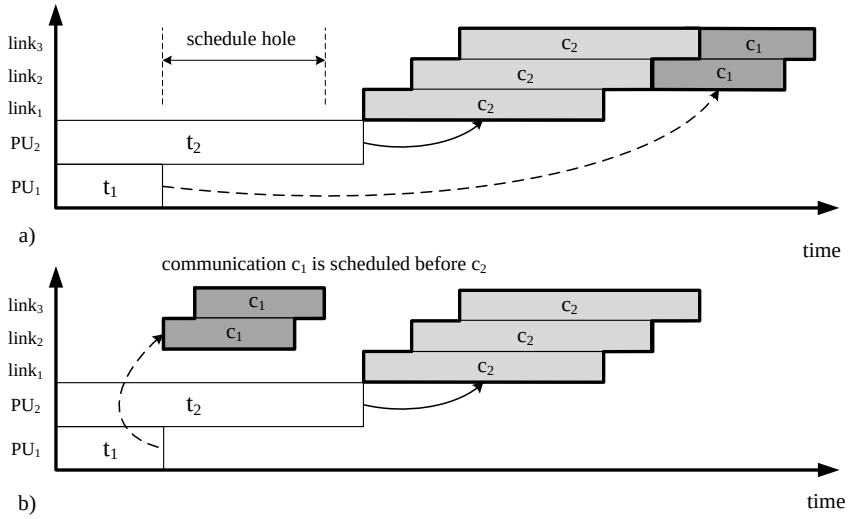


Figure 3.17. An example of a schedule hole

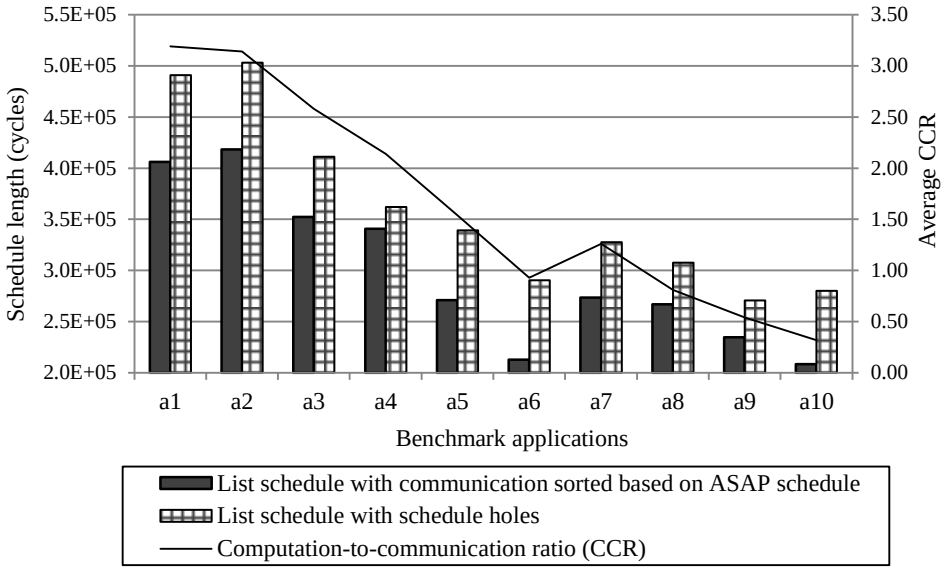


Figure 3.18. Comparison of communication scheduling schemes

3.6. Simulation environment

One of the goals of our approach is to produce schedules that are predictable and repeatable during the execution of the application on the NoC (e.g. no flits are lost, schedule length is met). We have interfaced our system-level design tool and written a SystemC event-driven application simulation kernel for Noxim, NIRGAM and XHiNoC network-on-chip simulators. An overview of these simulators has been given in Chapter 2.7. In this section we describe the SystemC application simulation kernel for the cycle-accurate XHiNoC simulator. The simulation kernel is similar for all the three NoC simulators. The main purpose of the XHiNoC is to provide a flexible shared communication media by utilizing a concept of locally organized packet identity (ID) division multiple access (IDMA). Local ID slots are distributed over every communication link, which can be attached to every flit of a packet or data stream as its local ID-tag (Samman, Hollstein, & Glesner, 2011). The core of the simulator is written in VHDL, while there is a SystemC wrapper on top of the XHiNoC that implements the core network interface functions such as send and receive.

XHiNoC allows a configurable number of ID slots for packet interleaving on each communication link. As in our system model we assume no flit interleaving we have configured XHiNoC ID table size to be one bit. It means that one ID-slot is reserved for flow control/deadlock management, leaving one ID-slot for data transmission. If the network model is not precise or schedule is not produced correctly then the simulation will reveal this – if there are no free ID

slots available the XHiNoC will start dropping the flits to avoid deadlocks in the network.

Application is scheduled with our system-level design tool. The resulting schedule is written into an intermediate format (Figure 3.19) that is read by the XHiNoC SystemC application simulation kernel interface. The intermediate format describes send/receive events, their timing and dependencies. Each SystemC processing core has a scheduler that maintains the local task queue and updates it based on received data. A task can be ready only when it has received all input data. Once activated, a task takes WCET amount of time to execute. Data is injected to the network only at respective scheduled time periods or as soon as possible if that time moment is passed.

Format of conditional send transactions:

- *processor ID (integer)* – source processor/router number;
- *communication start cycle (long integer)* – global time when the message needs to be injected into the network if pre-conditions have been met (number of pre-conditions equals zero);
- *delay before sending (long integer)* – time the simulation kernel needs to wait until it can inject the data into the network (if number of pre-conditions equals zero). It is being used in alternative experiments where dynamic communication schedule is simulated;
- *transaction ID (long integer)* – transaction identifier, used in receiver side to match pre-conditions;
- *number of flits (integer)* – number of flits to inject into the network;
- *target address (integer)* – receiver processor/router number;
- *number of pre-conditions (integer)* – number of pre-conditions that have to be met before the data can be injected into the network on its scheduled start cycle.

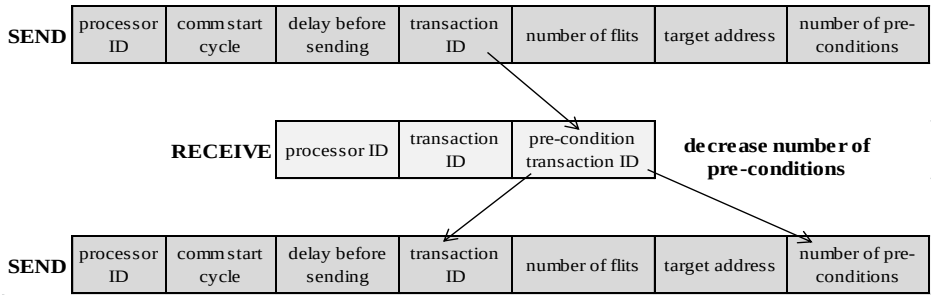


Figure 3.19. Data model of event based simulation

Original – no compaction

CPU 1 - SEND	
Destination CPU	Transaction ID
4	10
3	11
4	12

CPU 2 - SEND	
Destination CPU	Transaction ID
4	13
5	14
5	15

CPU 4 - RECEIVE	
Incoming transaction ID	
10	
13	

Modified – relative transaction IDs

CPU 1 - SEND	
Destination CPU	Transaction ID
4	1
3	1
4	2

CPU 2 - SEND	
Destination CPU	Transaction ID
4	1
5	1
5	2

CPU 4 - RECEIVE	
Incoming transaction ID	**Sender of flits**
1	**1**
1	**2**

Figure 3.20. An example of original and relative transaction IDs

Format of receive transactions:

- *processor ID (integer)* – receiver processor/router number;
- *transaction ID* – sender transaction ID;
- *pre-condition transaction ID* – transaction ID for which the number of pre-conditions should be decremented.

The key requirement of our application simulation kernel is the ability to determine at the receiver side the message transaction ID. Based on the received message ID the related pre-condition transaction ID is looked up from the receive-transactions-list and number of pre-conditions decremented. Data from a respective successor communication can be injected into the network on its scheduled start cycle only when the number of the message pre-conditions equals zero.

Initially, each XHiNoC 32-bit data-body and tail flit was tagged with a 20-bit message transaction ID and a 12-bit flit sequence number. Message transaction ID is needed for local schedulers to identify the received data. The flit sequence number is required by the ATLAS tool to perform performance and power analysis based on the simulation results. Such a global transaction ID numbering scheme wastes useful storage space and is not enough for big applications that require more than 20-bit transaction ID storage. To compact the data we use relative transaction IDs. At the sender side we keep a separate transaction numbering for each destination. The receive transaction format needs to be extended with the sender info. An example of original and relative transaction ID numbering scheme is depicted in Figure 3.20. The new numbering scheme occupies fewer bits in payload, leaving more storage space for application data.

3.7. Experimental results

In order to evaluate different aspects of our approach, we have run experiments with a set of synthetic task graphs mapped on different NoC sizes. Using synthetic task graphs is a common way to demonstrate scaling and feasibility of the proposed approaches (Sinnen & Sousa, 2005; Hu & Marculescu, 2005; Manolache, Eles, & Peng, 2007). The Figure 3.21 depicts our system-level design toolchain. The shapes with solid lines represent our contribution. We have a synthetic task graph generator where we can specify number of tasks, maximum number of outgoing edges for any vertex in the task graph, range of task WCET etc. The generated synthetic task graph is written into XML (extensible markup language) format that is read as an input by our system-level design tool. The system-level design tool is a C++ application that has its own configuration file where we can specify which types of algorithms to execute,

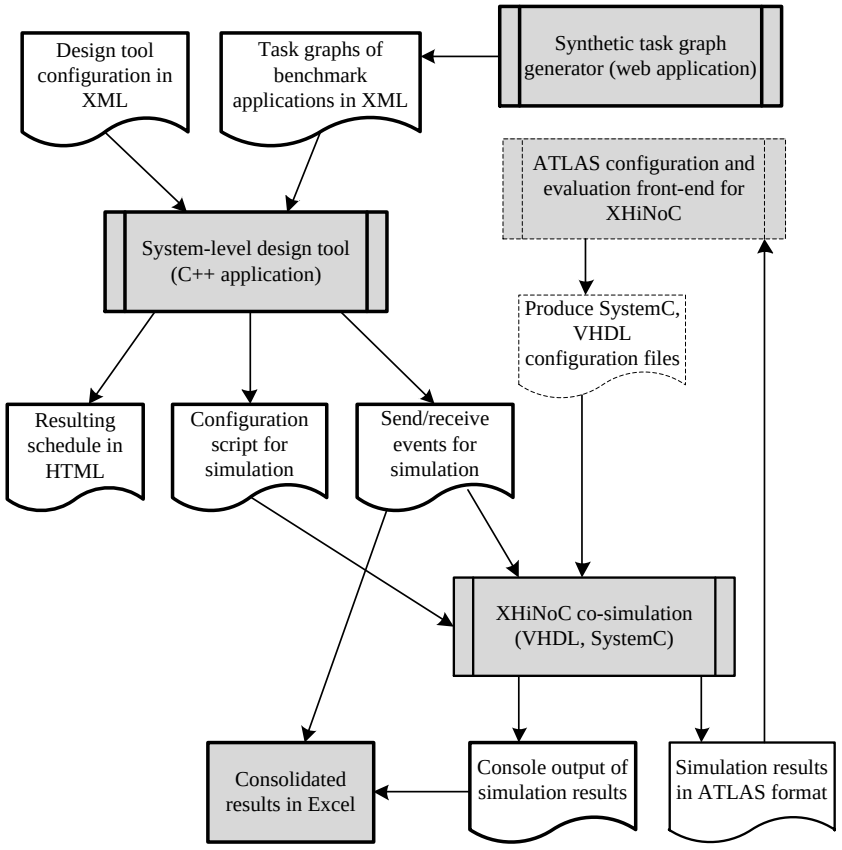


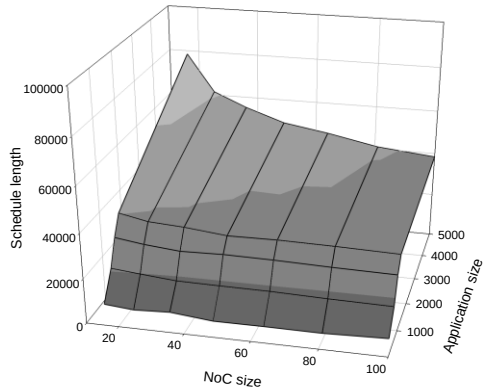
Figure 3.21. High-level view of our system-level toolchain

which benchmark application to model etc. It is the core part of our toolchain and covers most of the system-level design flow steps depicted in Figure 3.1. The system-level design tool generates several output files such as resulting schedule in HTML format, send/receive event files for NoC application simulation kernel etc. The XHiNoC platform is configured using the ATLAS configuration front-end. As a result we have proper XHiNoC VHDL and SystemC files that correspond to given architecture parameters. During the simulation the XHiNoC SystemC channels log timing information for each flit in ATLAS compatible format that can be used later for evaluation by the ATLAS tool. Additionally, we can capture the timing information of send/receive transactions and compare it to the static schedule produced by our system-level design tool. This is useful for debugging purposes when the simulation results do not correspond to the design tool results.

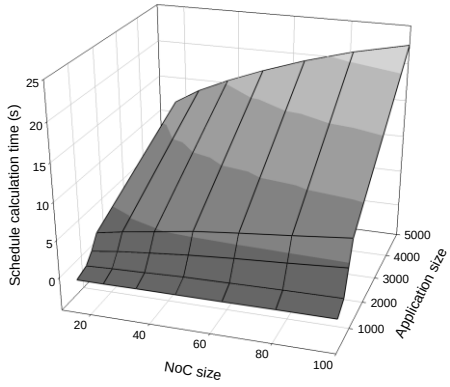
3.7.1 Complexity of the model

In the first set of experiments our goal is to illustrate the complexity of the communication model in respect to the application and network-on-chip size. The architecture parameters are varied together with the application size to show the scaling of our approach. We are using applications consisting of 100, 500, 750, 1000, and 5000 tasks mapped on different NoC sizes. These benchmark applications have been generated using in-house random task-graph generator that is part of our toolchain. The edge (communication transaction) count in the extended task graph is 244, 2488, 7490, 9691 and 25279 respectively. The initial mapping of the application has been given. The NoC architecture parameters are chosen as follows: link bit-width and flit size 32 bits, channel delay 1 cycle, switching delay 4 cycles and path setup delay 8 cycles. The experiments are performed on a computer with Intel L2400 processor (1.66 GHz) and 1 GB of available physical RAM (random access memory).

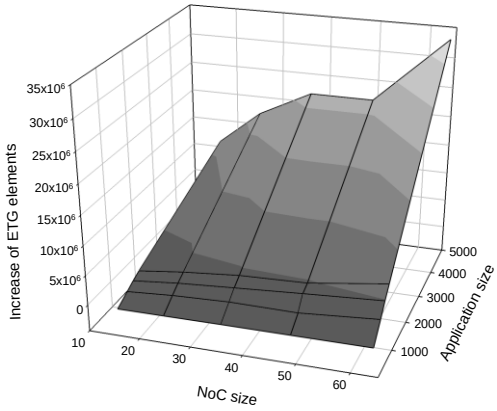
Our first experiment shows the NoC size impact on the schedule length and the calculation time for different application sizes. The results are depicted in Figure 3.22. The NoC size represents the total number of processing cores and the application size the total number of tasks. On the one hand, the more computational units we have available, the shorter schedule are we able to produce (Figure 3.22a). On the other hand, it takes more calculation time to model and synthesize the communication on a bigger NoC (Figure 3.22b). As communication sub-graph complexity depends on the application mapping and the number of communication links it traverses, we have measured in Figure 3.22c the NoC size impact on the number of task graph elements (vertexes and edges). The bigger the NoC and the more communication we have the larger the modelling overhead. However, the biggest application with 5000 tasks and 25279 communication transactions has a scheduling time 12 seconds, which is feasible for design space exploration.



a) Schedule length



b) Schedule calculation time



c) NoC size impact on extended task graph complexity

Figure 3.22. NoC and application size impact on modelling speed and complexity

3.7.2 Simulation results

In previous experiments our focus has been on the scaling of the approach for different application and NoC sizes. In the second set of experiments our goal is to verify the schedules produced by our system-level design tool by executing them on the cycle-accurate XHiNoC simulator. We have created another set of benchmark applications using our in-house random task-graph generator. Each of the ten applications (a1 - a10) consist of 2500 tasks with varying average vertex degree (1.5 - 5) and computation-to-communication ratio (CCR) (0.1 - 3.2). The applications are mapped to 2x2, 4x4 and 6x6 2D-mesh NoC. The NoC architecture parameters are chosen as follows: link bit-width and flit size 36 bits, channel delay 1 cycle, routing delay 4 cycles and path setup delay 8 cycles. The parameters used in the design tool are the same used to perform simulations with the XHiNoC. The experiments are performed on a computer with Intel Core i5-520M (2.40 GHz) processor and 4 GB of available physical RAM. Modelsim 6.5c has been used for running XHiNoC VHDL and SystemC co-simulation.

The experimental results show that a static schedule, being executed on the XHiNoC simulator, has no flit loss and the schedule deviation is at maximum one clock cycle. If we relax the send time of messages and send data as soon as possible, then the communication fails as network conflicts appear. When XHiNoC ID table size is set to one bit then XHiNoC can support only one communication in any router/network interface at a time.

Figure 3.23 depicts the average speedup compared to the simulation. Modelling time represents the time needed for synthesizing the communication and scheduling the application. Simulation time represents the time to simulate the generated static schedule. The modelling time is in average 28 times faster than the simulation time having at maximum one clock cycle deviation in schedule length, as mentioned previously. Therefore, the presented model is feasible for design space exploration and the results are repeatable during simulation.

The total amount of additional memory needed for locally storing the partial schedules is depicted in Figure 3.24. The memory overhead is affected by the CCR rather than the size of the NoC. It means that the model complexity does not explode with the size of the NoC. Using a larger NoC gives the possibility to distribute the load more evenly, reduce network contentions and schedule length.

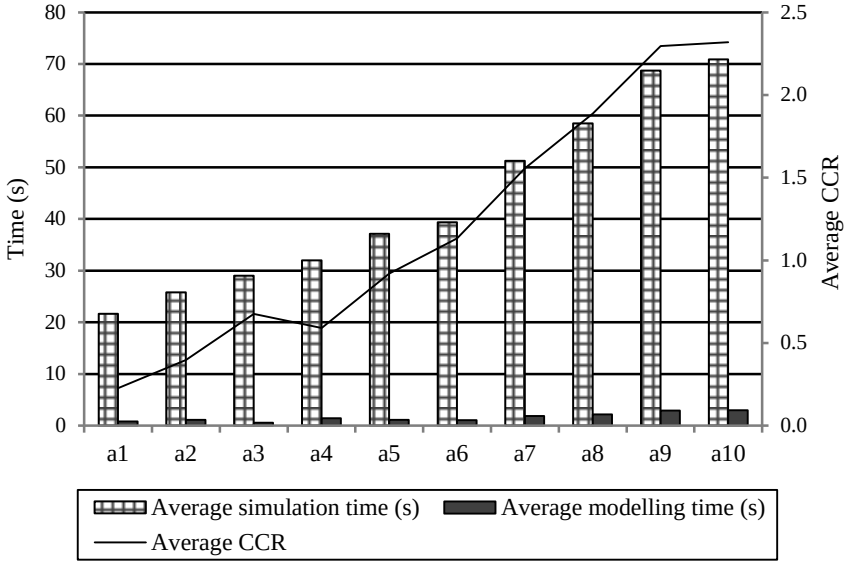


Figure 3.23. Modelling speedup compared to simulation

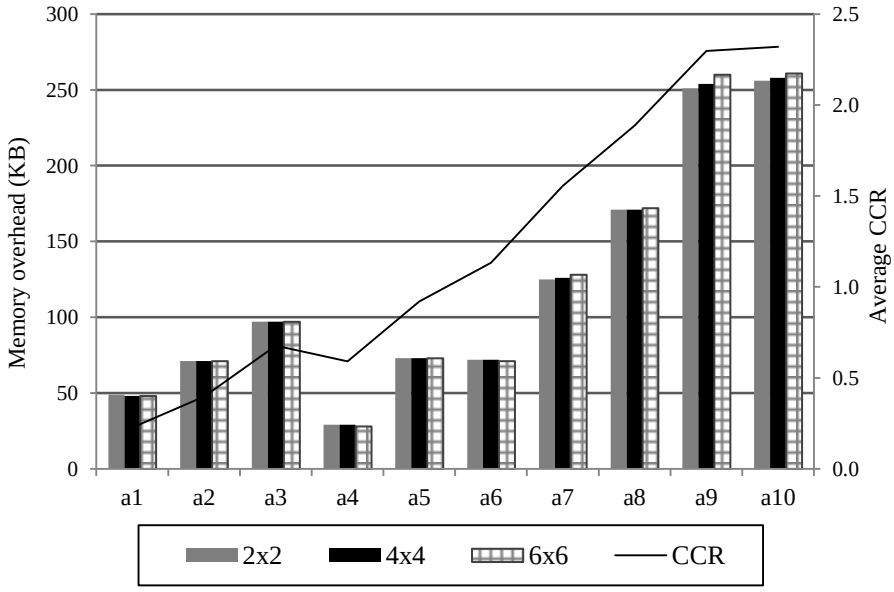


Figure 3.24. Schedule table memory overhead for different NoC sizes

3.8. Conclusions

In this chapter we have introduced our system-level design flow. We have provided description and definition of the system model and have formulated the design problem. We have described our communication synthesis method and demonstrated how it is being used during application scheduling. An edge in the task graph, connecting two tasks that have been mapped to different processing cores, is transformed into a series of vertexes (communication sub-graph) representing the network resources that need to be acquired during scheduling. Contention-aware scheduling uses the extended task graph as its input. It schedules the tasks and communication in a way that precedence constraints are met and network conflicts are avoided. The last part of the chapter explains how our system-level design tool is interfaced with the XHiNoC simulator, followed by the experimental results. The experimental results show that a static schedule, being executed on the XHiNoC simulator, has no flit loss and the schedule deviation is at maximum one clock cycle. Moreover, in our experiments the modelling time has been in average 28 times faster than the simulation time. Therefore, the presented method is feasible for design space exploration and the results are repeatable during simulation.

Chapter 4. Extensions of the Communication Model

In the previous chapter detailed description of our communication synthesis and scheduling approach has been presented. We have identified some areas that should be explored for possible improvements related to communication modelling. In this chapter we introduce two extensions of the proposed communication modelling approach to utilize the network-on-chip more effectively and to introduce the support for communication interleaving.

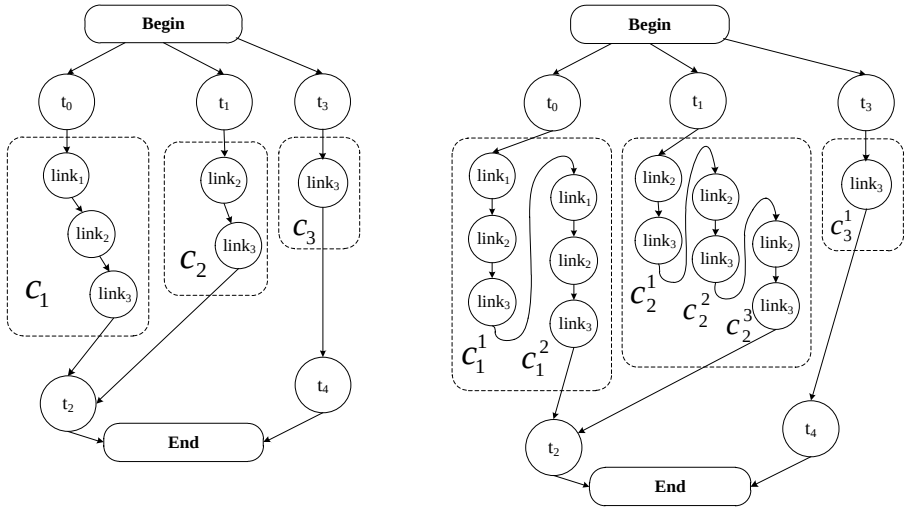
4.1. Packet-based schedules

In this section we will explore the trade-off between increased modelling complexity and the improvements gained from packet-based schedules. First, we will motivate the work by explaining, what are the possible areas of improvement when using packet-based communication modelling approach. Second, we will describe the packet-based communication synthesis and scheduling in detail, followed by the experimental results. The results of this section have been published in the following paper:

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2012). Contention-aware scheduling for NoC based systems. *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)* [submitted for review].

4.1.1 Motivation

Communication can be represented in the extended task graph on various granularity levels – at message, packet and flit level. In the previous chapter communication has been modelled as transmission of messages. For each message traversing a communication link a new vertex is added into the extended task graph. It means that each vertex in the communication sub-graph represents a data transmission on a physical communication link. A message is sent as a stream of flits. If some NoC implementation does not support sending a whole message as a series of flits we could let the NoC internally packetize the message. We would only have to take into account the additional amount of data to form each packet header. The experimental results in the previous chapter have shown that the approach is feasible for scheduling of real-time NoC-based systems. The approach is also feasible for design-space exploration due to its modelling speed and accuracy. However, due to the fact that in our NoC model we assume no data interleaving, reserving bandwidth for a long message could



a) message-based communication

b) packet-based communication

Figure 4.1. Message- and packet-based communication synthesis

affect negatively other traffic flows waiting for available resources. If we would model data transmission as a series of ordered but disjoint packets, we could have more flexibility for communication scheduling, possibly increasing network utilization and decreasing schedule length.

4.1.2 Packet-based schedules

In the previous chapter a communication vertex in the ETG represents a whole message. The extended task graph complexity depends on the number of tasks *A*, routing algorithm R_{algo}, NoC size *N* and the mapping *M*. This is represented by the function $G_{complexity} = (A, N, M, R_{algo})$. We have demonstrated that the approach scales well for larger applications and NoCs (Chapter 3.7.1). However, in previously presented method there is a lack of possibility to pre-empt a communication transmission. To increase communication scheduling flexibility we could split a message into one or several packets and to schedule the packets independently, preserving their order. The overall concept of communication synthesis and scheduling remains the same. A new design parameter is packet size P_{size}. A message is divided into *n* packets depending on the P_{size}. For each packet traversing a communication link a vertex is added into the extended task graph. Additionally, separate packets from the same message are connected by an edge to preserve their order in the communication scheduling.

First, we will have a look how the change in communication modelling granularity affects the communication synthesis process. The message- and packet-based communication synthesis is depicted side-by-side in Figure 4.1. In message-based communication synthesis (Figure 4.1a) a vertex in the

communication sub-graph represents a whole message. In packet-based communication synthesis (Figure 4.1b) the communication c_1 is split according to chosen P_{size} into two packets (c_1^1; c_1^2), communication c_2 into three packets (c_2^1; c_2^2; c_2^3) while data for c_3 fits into one packet (c_3^1). The *n*-th packet of communication c_i is labelled as c_i^n. For the sake of simplicity this example does not contain any actual communication amounts and the division into packets is illustrative. The packets are connected with an edge to preserve their order during communication scheduling (Figure 4.1b).

Definition 4.1 (Packet-based communication synthesis problem). *Given an application A, a network model N, and a mapping function M the result of packet-based communication synthesis is an extended task graph where the communication edges between tasks are transformed into sequences of packets (communication sub-graph) depending on the packet size* P_{size}.

Before we describe the changes needed in the contention-aware scheduling algorithm, presented in Chapter 3, we look at the flexibility that is available when utilizing packet-based communication modelling. We have a task graph depicted in Figure 4.2a that we are scheduling on a 2x3 2D-mesh NoC (Figure 4.2b). In this example our main goal is to explain the concept and difference between the two communication modelling granularity levels, not to find the optimal schedule. The task mapping, based on which the communication synthesis has been performed, is depicted in Figure 4.2b. In Figure 4.2c we can see the message-based schedule. The communication has been scheduled in the following order – c_1, c_3 and c_2. In Figure 4.2d the corresponding packet-based schedule is depicted. Let us assume that the scheduling process has scheduled first the packet c_1^1. As scheduling is done packet-by-packet then let us assume that in the next iteration the scheduler has decided to schedule the packet c_3^1. Next, the remaining packet of communication c_1 (c_1^2) has been scheduled. We can see that by scheduling the communication packet-by-packet and re-ordering their transmission on communication links we are able to get a shorter schedule length. However, even for this small example the number of communication sub-graph elements has increased approximately 2.5 times. This is a trade-off that we have to take into consideration. The graph complexity of packet-based communication modelling depends additionally on the packet size $G_{complexity} = (A, N, M, R_{algo}, P_{size})$. If the packet size is equal to the maximum size of any communication transfer – $P_{size} = max(c_i^{size})$ – then the packet-based model complexity is the same as for message-based.

There are a few changes that are needed to be made to the contention-aware scheduling (Figure 3.10) and to the communication synthesis and scheduling (Figure 3.12) algorithms. We will describe only the main differences. The detailed explanation of the contention-aware scheduling algorithm has been presented in the previous chapter. The pseudocode of the packet-based contention-aware scheduling algorithm is depicted in Figure 4.3.

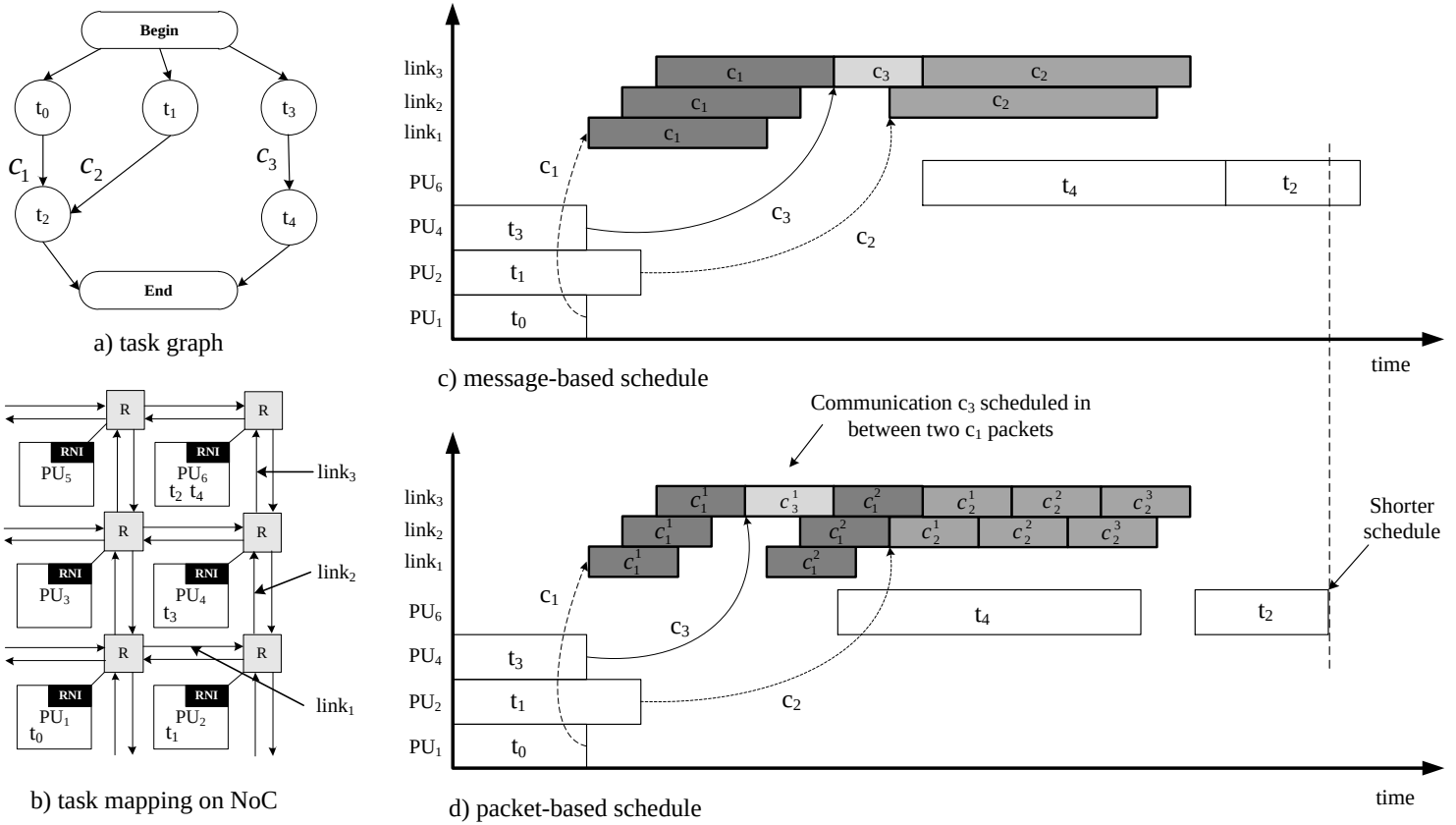


Figure 4.2. Comparison of message- and packet-based schedules

```
Contention-aware scheduling
1     calculate task priorities
2     add task with pred(t) = ∅ into ready task list
3     while tasks not scheduled ≠ ∅
4       for each task in ready task list do
5         sort ready task list based on mobility
6         schedule ready task
7         transform outgoing communication of scheduled task into packets
8         add the first vertex (packet) into communication list
9         add eligible successor tasks to ready task list
10        remove scheduled task from ready list
11      end for
12      for each packet in communication list do
13        sort communication list based on ASAP schedule
14        schedule packet on the whole route
15        add eligible successor packet or tasks to ready task or comm. list
16        remove scheduled packet from communication list
17    end for
18  end while
End Contention-aware scheduling
```

Figure 4.3. Pseudocode of packet-based contention-aware scheduling

```
SchedulePackets(first vertex)
1   add first vertex (packet) of communication sub-graph into readyToSchedule list
2   while readyToSchedule ≠ ∅,  i = 0  do
3     linkSchedTime = get max schedule time from comm. link the vertex c_i is mapped
4     for each incoming edge i_j of vertex c_i do
5       predecessor vertex c_j^pred = get source vertex of edge i_j
6       if (c_j^pred is a regular task) then
7           maxTimeFromPredecessor = schedule end time of c_j^pred
8       else
9           if (packet number of first vertex != packet number of c_j^pred) then
10              isFirstPacket = TRUE
11          end if
12          maxTimeFromPredecessor = get comm. start time on previous link + routing
13          delay
14      end if
15    end for
16    if  (isFirstPacket) then
17      commStartTime = linkSchedTime
18    end if
19    if  linkSchedTime <  maxTimeFromPredecessor then
20      commStartTime =  maxTimeFromPredecessor
21    else
22    commStartTime = linkSchedTime
23    end if
24    commEndTime = commStartTime + communication delay of packet c_i
25    back annotate schedule end time for predecessor comm. vertexes if needed
26    add successor vertexes and remove scheduled packet c_i from readyToSchedule
27  end while
end SchedulePackets
```

Figure 4.4. Pseudocode of packet-based communication scheduling

In line 7 (Figure 4.3), after a task has been scheduled, we transform a communication edge into a communication sub-graph that contains a series of packets. In line 8 we add the first vertex of the communication sub-graph into the communication list. After the ready tasks have been scheduled, we start scheduling of the packets in the communication list (lines 12-17). In each iteration one packet on the whole communication path is scheduled. Therefore, in the beginning of a new iteration packets are sorted in the communication list by their ASAP schedule. If we have scheduled the last vertex of current packet then we check the type of the successor vertex (line 15). If it is a task and if eligible we add it into the ready task list. If it represents a communication then the successor packet is added into the communication list.

The Figure 4.4 depicts the pseudocode of packet-based communication scheduling. The main difference is in line 9 where we check whether the packet being currently scheduled is the first packet of the message. If it is a next packet of the same message then we need to make sure that we schedule it right after the first packet on the communication link even if we have information that the predecessor vertex has been scheduled to a future date. We want to pipeline the messages, not to wait until the whole packet has been transferred and then start the next transmission.

4.1.3 Experimental results

In the experimental results our focus is on the improvement of schedule length we are able to gain by packet-based schedules in relation with the size of the packet and increase of the modelling complexity. The NoC architecture parameters are chosen as follows: link bit-width and flit size 36 bits, packet header 20 bits, channel delay 1 cycle, switching delay 4 cycles and path setup delay 8 cycles. The experiments are performed on a computer with Intel Core i5-520M (2.40 GHz) processor and 4 GB of available physical RAM.

In the first experiment we have used the task graph *a1* from the set of benchmarks described in Chapter 3.7.2. The application has been mapped to a 8x8 2D-mesh NoC. The average computation-to-communication ratio (CCR) is 4.2 for the given application and mapping. Figure 4.5 depicts comparison between message-based and packet-based schedules for different packet sizes. We can see that packet-based schedules are in average 10% shorter (from 7% up to 19%). We have run the same experiments with the rest of the benchmark applications. In Figure 4.6 we depict the average improvement of packet-based schedules in relation with computation-to-communication ratio. We can see in Figure 4.6 that the amount of improvement in schedule length depends on the computation-to-communication ratio. The higher the CCR the more improvement we can gain in schedule length by modelling communication in packets.

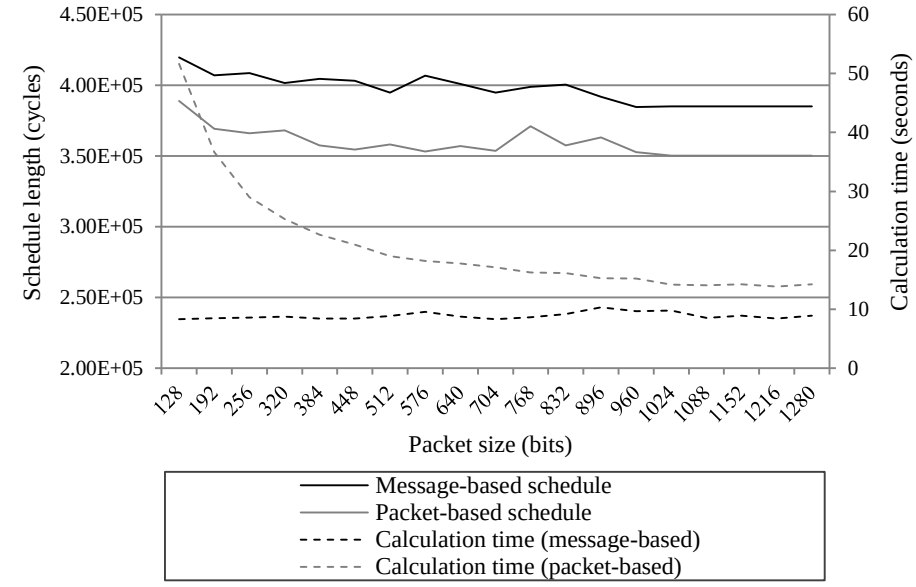


Figure 4.5. Message-based versus packet-based schedule for different packet sizes (benchmark application a1)

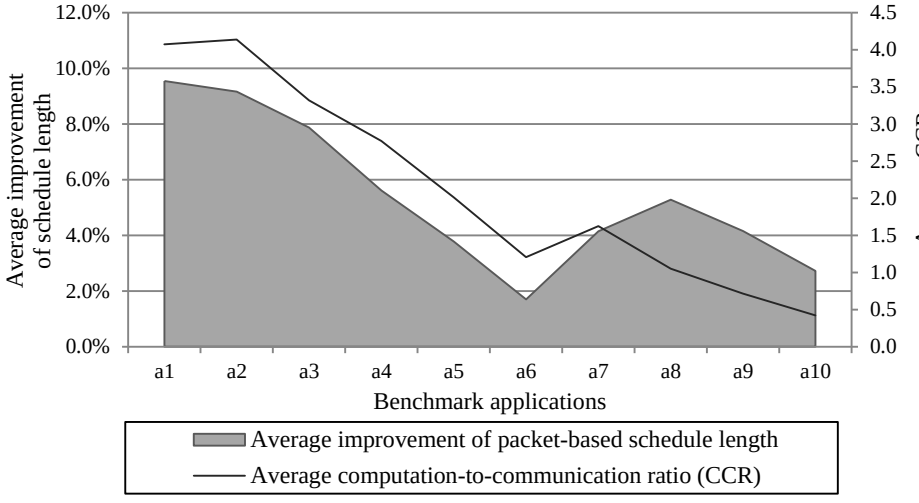


Figure 4.6. Packet-based schedule improvement in relation to CCR

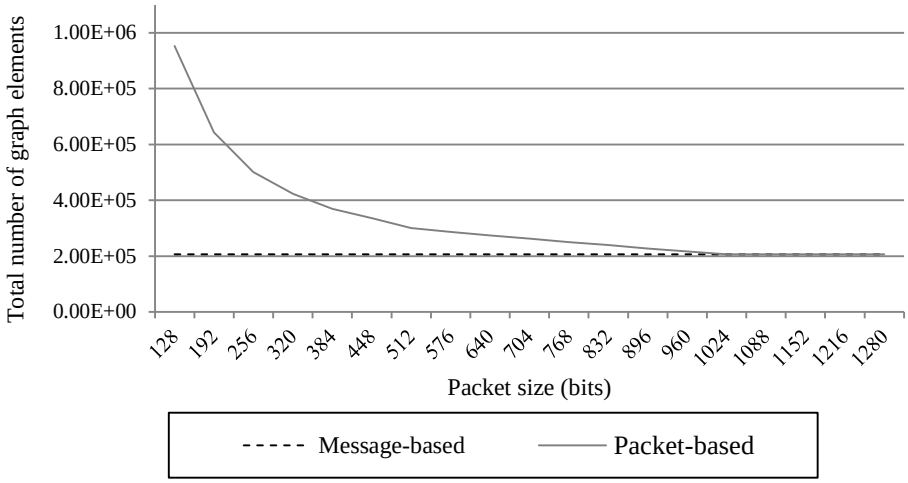


Figure 4.7. Comparison of message- and packet-based communication modelling complexity (benchmark application a1)

In the second set of experiments we are analysing the packed-based communication model complexity. As we have mentioned previously the packet-based communication model complexity is affected by the size of the packet. This has been confirmed by the experimental results. There is a sharp increase in communication synthesis and schedule calculation time if we go below the packet size of 512 bits. The results for benchmark application *a1* are depicted in Figure 4.7. The total number of graph elements (vertexes and edges) goes up sharply below 512 bits while it converges at 1024 bits with the message-based model. It means that for this benchmark application a good packet-size trade-off is between 512 and 1024 bits. If the packet size is bigger than 1024 bits it has no effect as each communication fits into a 1024-bit packet. Similar results have been retrieved with the rest of the benchmark applications.

4.2. Support for communication interleaving

In this section we will propose an extension to our communication model to support communication interleaving. The goal is to utilize more effectively the available network resources and to reduce the schedule length. First, we will give the motivation of the work. Second, we will explain in detail how multiple flit queues are being used to schedule the communication, followed by experimental results.

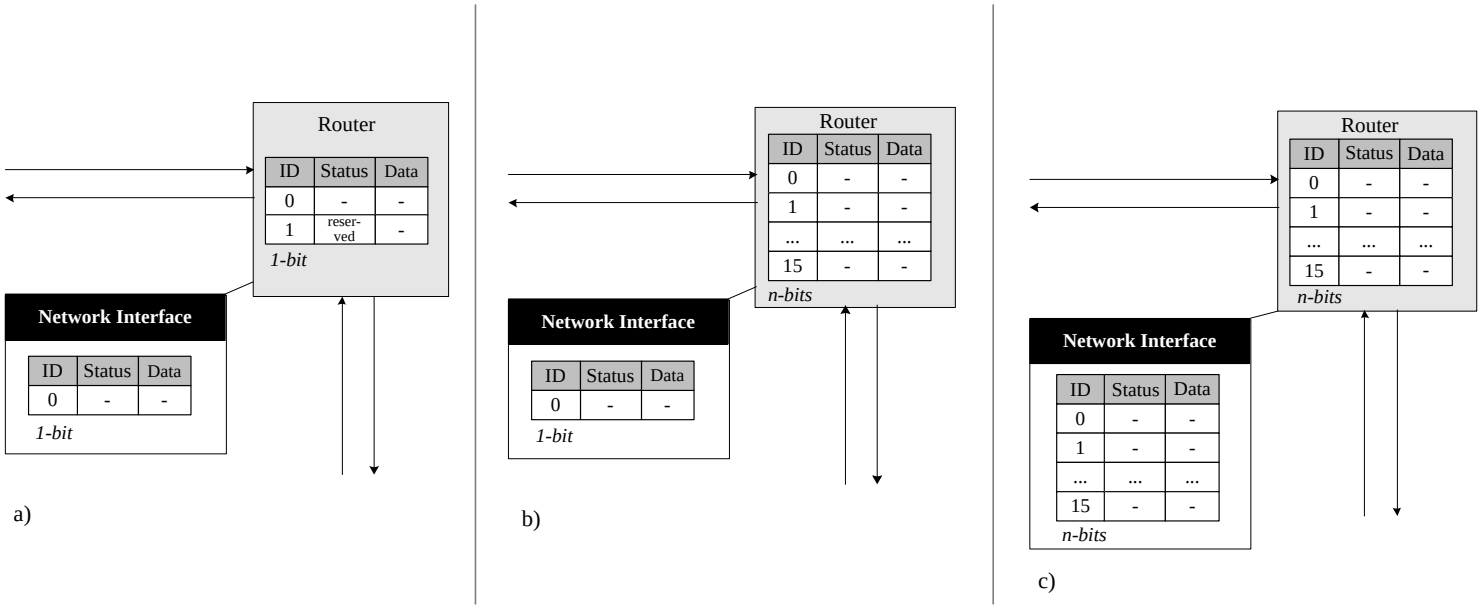


Figure 4.8. Three schemes of flit queues in network interface and XHiNoC router

4.2.1 Motivation

So far we have assumed that we have a network-on-chip with simple and hardware efficient routers and no communication interleaving. For example, in the experiments performed in Chapter 3.7.2 we have set the XHiNoC ID table size to be one bit – one ID-slot is reserved for deadlock management leaving only one ID-slot for data transmission. Figure 4.8a depicts the schematic view of such setup. If the schedule is not produced correctly then the XHiNoC will start dropping the flits. Experimental results in Chapter 3.7.2 showed that our contention-aware scheduling results are accurate having at maximum one clock cycle deviation and no lost flits. If the scheduler does not follow the communication schedule and injects data into the network as soon as possible then the XHiNoC soon runs out of available ID-s and starts dropping the flits. We got an interesting result if we increased the number of available ID-s in the router and run again the same experiment. The schematic view of such setup is depicted in Figure 4.8b. As XHiNoC routers had now resources (ID-slots) to interleave the flits we were able to get a shorter application execution time. It is important to note that the local network interface was trying to inject data to the network utilizing the full available bandwidth. However, at some point contentions occurred decreasing the injection rate of the network interface. For some applications we were able to achieve up to 30-40% shorter execution time but at the cost of increased average delay. Moreover, the results were unpredictable. Therefore, we have extended our model to include the communication interleaving aspect that will be described in the next section.

4.2.2 Multiple flit-queues

We have extended our model, described in Chapter 3, by a configurable number of flit queues. We assume that the bandwidth is equally shared between *n* separate flit queues. It means that the communication delay will be also *n* times higher for any message, even when there could be time periods where a router has temporarily resources available for full bandwidth. It is a trade-off between modelling complexity and accuracy. During the scheduling we are trying to load-balance the queues and to mimic the operation of the XHiNoC routers. The number of flit-queues is the same as the ID table size in the XHiNoC. Figure 4.8c depicts the schematic view of such setup. It is important to note that we are using the best-effort router implementation and assume no special hardware for guaranteed services. The design problem we are trying to solve in this section is formulated as follows.

Definition 4.2 (Contention-aware scheduling with communication interleaving). *Given an application A, a network model N, and a mapping function M our goal is to load-balance communication between n separate flit queues so that the schedule length* S_{length} *would be minimized in respect to the schedule without communication interleaving.*

```
ScheduleCommunicationWithInterleaving(first vertex, max number of IDs)
1     add first vertex of communication sub-graph into readyToSchedule list
2     while readyToSchedule ≠ ∅,  i = 0  do
3        for each incoming edge i_j of vertex c_i do
4           predecessor vertex c_j^pred = get source vertex of edge i_j
5           if (c_j^pred is a regular task) then
6                 maxTimeFromPredecessor = schedule end time of c_j^pred
7           else
8                 maxTimeFromPredecessor = get comm. start time on previous link + routing
9                 delay
10          end if
11       end for
12
13       fq = list of flit queues of output channel where the c_i is mapped
14       for each flit queue fq_k  and foundAvailableIDSlot == FALSE
15          if (fq_k time advanced <= maxTimeFromPredecessor) then
16                foundAvailableIDSlot = TRUE
17                idSlotNumber = k
18                linkSchedTime = fq_k time advanced
19          else
20                k = k + 1
21          end if
22       end for
23
24       if (foundAvailableIDSlot == FALSE) then
25          if (fq size == 0) then
26                idSlotNumber = 0
27                linkSchedTime = maxTimeFromPredecessor
28          else if (idSlotNumber < max number of IDs) then
29                linkSchedTime = maxTimeFromPredecessor
30          else
31                minAdvancedTime = find the queue from fq with minimum advanced time
32                linkSchedTime = minAdvancedTime
33                idSlotNumber = ID slot number of the queue with minAdvancedTime
34          end if
35       end if
36
37       if  linkSchedTime <  maxTimeFromPredecessor then
38          commStartTime =  maxTimeFromPredecessor
39       else
40          commStartTime = linkSchedTime
41       end if
42       commEndTime = commStartTime + communication delay of message c_i
43       fq[idSlotNumber] time advanced = commEndTime
44       back annotate schedule end time for predecessor comm. vertexes if needed
45       add successor vertexes and remove scheduled message c_i from readyToSchedule
46    end while
end ScheduleCommunicationWithInterleaving
```

Figure 4.9. Pseudocode of communication scheduling with interleaving

To maintain the complexity of the graph model, a vertex in the extended task graph represents the whole message transfer on a communication link (like in Chapter 3). The task scheduling process is the same as shown in Figure 3.10. To support communication interleaving we need to extend the communication scheduling and the SystemC application simulation kernel. Figure 4.9 depicts the modified communication scheduling pseudocode with interleaving support. Input to the procedure is the first communication vertex of the message to be scheduled and the maximum number of IDs we have allocated for this application. We need to check that we are not allocating more ID-slots than we have configured in the XHiNoC. In the previous and the following text we are using the terms "number of flit queues" and "number of ID slots" interchangeably. First, we find the schedule time from predecessor vertexes (lines 3-11). Second, we go through all the flit queues of the respective router output channel (lines 14-22). We check whether there is a previously allocated flit queue that does not overlap with communication start time of the current message. If we have found such queue then we store its number (line 17) and the schedule time of that flit queue (line 18). If we have not found a suitable queue from previously allocated ones then we have several options to consider (lines 24-35). If the list that stores the flit queues is empty then we allocate the first queue (lines 26-27). If the list of flit queues is not empty and we have not found a suitable from them, then we will check whether we could allocate a new one (lines 28-29). This relates to our goal to load-balance the communication between the number of flit queues that we have specified. If we have allocated the maximum number of flit queues then we will look for a flit queue that has the minimum schedule time and choose that one (lines 31-33). We will determine the final communication start time by comparing the schedule time from communication link and predecessor vertexes and choose the maximum (lines 37-41). If we faced a network contention on the current communication link and delayed the start time of the vertex then we need to update the communication start time of all related predecessor communication vertexes (line 44).

The SystemC application simulation kernel described in Chapter 3.6 has been modified as follows. Each local network interface contains information how many separate flit queues have been allocated for the application (Figure 4.8c). When sending flits, the SystemC application simulation kernel will first try to find either an empty flit queue or find a flit queue that has the fewest flits inside and insert flits into the respective queue. The network interface send process is sending flits in round-robin fashion from each queue. If a corresponding queue is empty then the process will wait for one cycle and proceeds with the next queue on the next cycle. Such a scheme allows flit interleaving while maintaining predictability of scheduling results. However, not all available network bandwidth might be utilized. The IDMA switching scheme is different from virtual channels with TDMA. There is no fixed time slot allocated for a communication all over the routing path. By utilizing IDMA switching XHiNoC router will adjust the bandwidth allocated for a communication dynamically

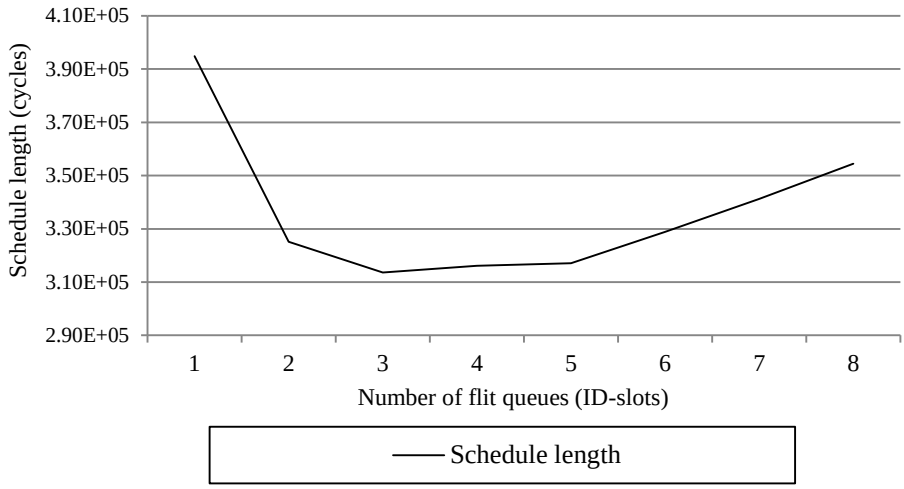


Figure 4.10. Schedule length versus number of IDs (benchmark application a1)

based on the current load. Our extension is more pessimistic and similar to virtual channels with TDMA.

4.2.3 Experimental results

In the experimental results our first goal is to confirm whether the produced schedules with interleaving support are repeatable during application execution on the XHiNoC. Second, we want to find out the difference between our conservative schedule and the schedule where communication is injected as soon as possible. The NoC architecture parameters are chosen as follows: link bit-width and flit size 36 bits, packet size 512 bits, packet header 20 bits, channel delay 1 cycle, switching delay 4 cycles and path setup delay 8 cycles. The experiments are performed on a computer with Intel Core i5-520M (2.40 GHz) processor and 4 GB of available physical RAM. Modelsim 6.5c has been used for running XHiNoC VHDL and SystemC co-simulation.

In the first set of experiments we have used the task graph *a1* from the set of benchmarks that we have described in Chapter 3.7.2. The application has been mapped on a 8x8 2D-mesh NoC. The average computation-to-communication ratio (CCR) is 4.2 for the given application and mapping. Experimental results in Figure 4.10 show that increasing the number of available ID-slots for given application decreases the schedule length (up to 20%) to a point after which the schedule length starts to increase again. It means that even if communication interleaving increases the average communication delay we are able to more effectively use the bandwidth and get a shorter schedule. If there are more than 5 flit-queues then we are allocating resources for them but these resources are not effectively utilized by the application. The schedule length matches the results

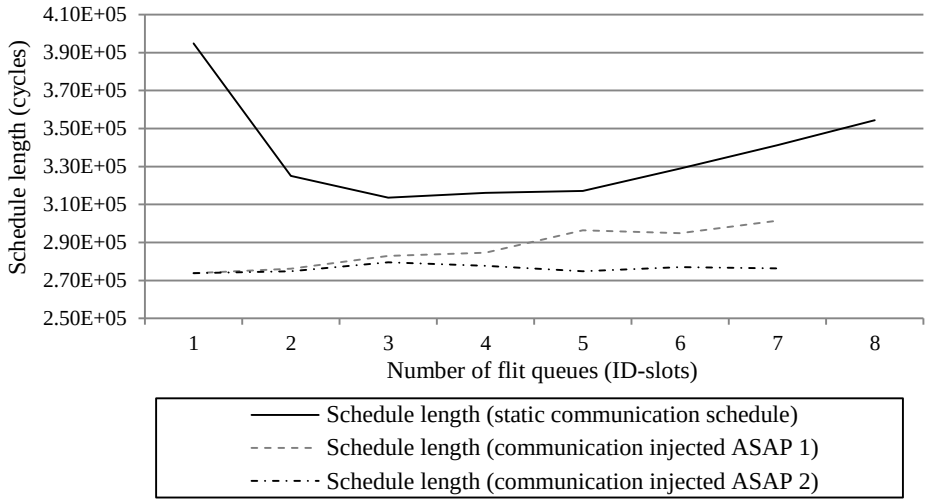


Figure 4.11. Results of different application execution scenarios (benchmark application a1)

from application execution on the XHiNoC simulator. Moreover, there are no lost flits. Similar results are observed with the rest of the benchmark applications. Optimal number of flit-queues for this set of benchmark applications is between 2 and 4 giving in average 20% of improvement in schedule length. Schedule calculation time is not affected by the number of flit-queues.

In the second set of experiments we have selected the same benchmark application *a1* but made changes to the SystemC application simulation kernel. Our goal is to find out the difference in schedule length and in average communication delay between our conservative communication schedule versus the schedule where data is injected into the network as soon as possible. We have produced the same static schedule with our system-level design tool as in previous experiments – that is, the communication is injected into the network only at respective scheduled time moments. We have made two different types of modifications. In both cases the communication is injected into the network as soon as possible. We do not wait for a scheduled start time of a communication. The difference is how the network interface is sending flits. In the first case, depicted by a grey dotted line in Figure 4.11, if a flit queue is empty then the send process waits for one cycle and proceeds with the next queue. In the second case, depicted by a black dotted line in Figure 4.11, if a flit queue is empty the process will find another queue that has some flits to send. In the first case we can see in average 13% improvement in schedule length and in the second case 17% improvement. However, it comes at the cost of increased average delay. Moreover, with such scheme the results are not predictable and do not correspond to the modelling results. It is because there is no application level

injection control or communication scheduling. Data is injected into the network as soon as possible, hoping that the network can cope with the load.

4.3. Conclusions

In this chapter we have presented two extensions of our communication model. In the first part we explored the improvements in schedule length versus modelling complexity achieved by using packet-based communication synthesis and scheduling. To increase communication scheduling flexibility a message is split into one or several packets and packets are scheduled independently, preserving their order. For each packet traversing a communication link a new vertex is added into the extended task graph. Additionally, separate packets from the same message are connected by an edge to preserve their order in the communication scheduling. Experimental results show that by synthesizing and scheduling communication at packet granularity we are able to achieve in average 10% shorter schedules at the cost of increased modelling complexity (calculation time and memory requirements).

In the second part of the chapter we have described an extension to our communication model to support communication interleaving. We have a configurable number of flit queues. We assume that the bandwidth is equally shared between the flit queues. During the scheduling we are trying to load-balance the queues and to mimic the operation of the XHiNoC routers. The number of flit-queues is the same as the ID table size in the XHiNoC. Experimental results have shown that increasing the number of available ID-slots for an application decreases the schedule length to a point after which the schedule length starts to increase again. We have been able to achieve in average 20% of improvement in schedule length using communication interleaving compared to the non-interleaving schedules described in Chapter 3.

Chapter 5. Design Optimization Techniques

Networks-on-chip are flexible communication platforms where computation is decoupled from communication. The available flexibility increases the amount of design options that need to be explored. Depending on the level of freedom we are interested in two options to optimize the design – to explore different application mappings or to improve the schedule based on given mapping.

The goal of this chapter can be formulated as follows. Given an application *A*, a NoC architecture *N* and a mapping function $M(T \rightarrow R)$ our goal is to perform design space exploration and optimization of the initial design with respect to the schedule length and to produce a set of feasible near-optimum design options. At the same time, we are also measuring the calculation time that is used to define a trade-off between the improvement gained during optimization and the time spent for calculation. Our goal is not to propose any enhancements to optimization methods but to show that our communication modelling and synthesis approach can be applied to arbitrary scheduling or optimization method.

The main results described in this chapter have been published in the following conference paper:

Tagel, M., Ellervee, P., & Jervan, G. (2010). Design Space Exploration and Optimisation for NoC-based Timing Sensitive Systems. *Proceedings of the 12th Biennial Baltic Electronic Conference* (pp. 177 - 180). Tallinn, Estonia.

5.1. Motivation

According to Marculescu, Ogras, Li-Shiuan Peh Jerger, and Hoskote (2009) application mapping has a major impact on schedule length, NoC performance and power consumption. Mapping tasks to processing cores is similar to the quadratic assignment problem (QAP) that is known to be computationally intractable (Garey & Johson, 1979). For example, mapping 10 tasks onto 10 NoC cores has in total *3 638 800* combinations (*10!*). If one evaluation would take *0.1* milliseconds then we would need approximately one year to accomplish this task. To run an exhaustive search to find a global optimum is infeasible for complex problems. Heuristics shall be used to approximate the optimum in reasonable amount of time. Figure 5.1 depicts one of possible taxonomies of classical optimization algorithms proposed by Talbi and Muntean in 1993. Besides list scheduling that is the basis of our contention-aware scheduling, we

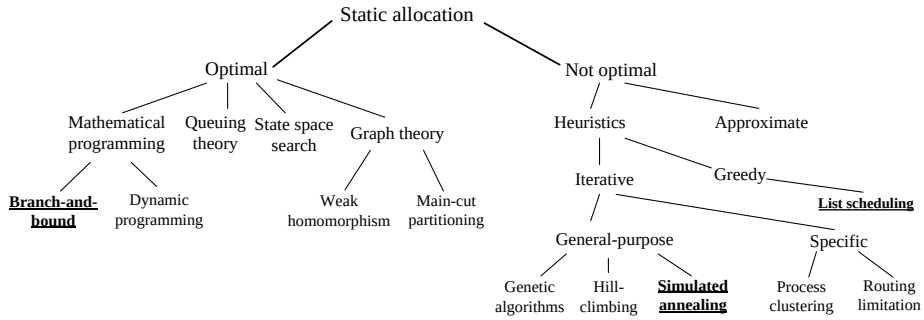


Figure 5.1. Taxonomy of optimization algorithms (Talbi & Muntean, 1993)

have chosen two well-known algorithms to show that our communication modelling approach can be applied to an arbitrary optimization algorithm. We are exploring the trade-off between time to calculate the schedule versus improvement gained. Additionally, we are interested to see how much influence gives the application mapping or the schedule improvement to the resulting schedule length.

Various optimization techniques have been described by several authors to solve the mapping or scheduling problem in the NoC domain. Orsila, Salminen, Hännikäinen, and Hämäläinen (2007) proposed an algorithm called optimal subset mapping (OSM) that rapidly evaluates the task mapping space. It takes a random subset of tasks and finds the optimum mapping in that subset by trying all possible mappings (brute-force search). The experimental results are compared to simulated annealing (SA) and group migration (GM) algorithms. Lu, Xia, and Jantsch (2008) present a cluster-based simulated annealing algorithm that combines clustering technique with simulated annealing to map processing cores onto 2D-mesh NoCs. Clustering is used to greatly simplify the mapping problem as mapping is done cluster wise instead of node wise. Hu and Marculescu (2003) show the impact of different task mappings to the communication energy consumption and present a branch-and-bound algorithm to solve the problem under tight performance constraints. What is interesting is the authors' conclusion – they are able to generate more efficient designs with less computation time by their branch-and-bound based approach compared to simulated annealing. In a follow-up paper Hu and Marculescu in 2005 proposed an energy-aware scheduling method to statically map and schedule application-specific communication transactions and computation tasks onto heterogeneous network-on-chip architectures. The approach is based on the idea of slack-budgeting, which allocates more slack to those tasks whose mapping onto processing cores has a larger impact on energy consumption and performance of the application. The authors use a search and repair method to improve the initial schedule and to fix the missed deadlines. This overview shows that general

purpose optimization algorithms and custom heuristics have been successfully applied to the NoC domain to solve the scheduling or mapping problem. However, different assumptions on underlying NoC architecture and communication modelling do not allow the algorithms to be used one-to-one in our system-level design framework. General purpose optimization algorithms require some additional analysis and tweaking for each concrete problem and system model.

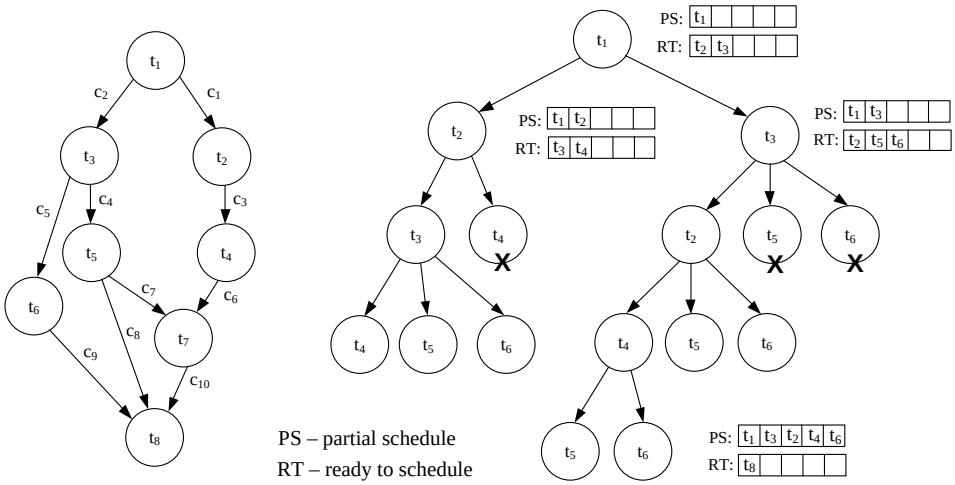
In Chapter 3.5 contention-aware scheduling (based on list scheduling) is used to schedule an application with given mapping. List scheduling is a greedy heuristic using a priority list and precedence constraints to schedule the tasks and minimise the schedule length. The algorithm is straightforward to implement and it has a low calculation time. It is a good option for design space exploration. However, to get an optimum solution, one would need to schedule all possible task sequences reaching in worst case *n!* permutations. In this chapter we are describing the use of branch-and-bound and simulated annealing optimization techniques in our system-level design framework. We are exploring the trade-off between improvement in schedule length and time spent for calculation. Second, we are analysing how much influence gives the application mapping or the schedule improvement to the resulting schedule length.

5.2. Schedule optimization with branch-and-bound

Branch-and-bound (B&B) is a general algorithm for finding optimal solutions for various computationally intensive optimization problems. Branch-and-bound consists of three main functions – branching, bounding and pruning. Branching describes the problem as a search tree whose nodes are subsets of given problem. Bounding calculates upper and lower bounds that are used to evaluate a set of candidates and prune the ones that do not lead to an optimum. Figure 5.2 depicts an illustration of the branch-and-bound process. An example task graph is transformed into a B&B search-tree where some nodes have been pruned as they do not lead towards the optimum solution.

The optimization problem we are trying to address in this section with branch-and-bound can be formulated as follows.

Definition 5.1 (Branch-and-bound schedule optimization problem). *Given an application A = (T, C, wcet, comm), a network model N = (R, L) and a mapping function M(T→R) the branch-and-bound optimization problem is used to determine an optimum feasible schedule S for application A on network-on-chip N.*



a) example task graph

b) partial branch-and-bound search tree

Figure 5.2. Branch-and-bound example

The pseudocode of branch-and-bound algorithm is depicted in Figure 5.3. Each B&B tree node contains a partial schedule (constructed so far) and a list of ready tasks as depicted in Figure 5.2b. We are choosing a node to branch based on the *best-first* strategy. For this we are sorting all the partial schedules by length and choosing the first (best) one. We have also evaluated a *breadth-first search*, but it did not improve the calculation speed. Each ready task of a node being branched will be added into the B&B search tree and stored also in the partial schedule list. There is an example in Figure 5.2b. We start with task t_1 – the partial schedule contains only t_1 and task t_1 activates tasks t_2 and t_3 (ready task list). When we are branching from task t_1 then we add into B&B search tree all the task t_1 ready task list contents – tasks t_2 and t_3. The new node t_2 in B&B search tree contains a partial schedule t_1, t_2 and it activates the tasks t_3 and t_4. Such a branching strategy avoids infeasible solutions and reduces the number of calculations. The partial schedule length is calculated based on the order of tasks in the partial schedule. The rest of the tasks are scheduled by list scheduling that gives us a tight upper bound.

The lower bound is calculated by similar approach as in the paper of Rahman and Chowdhury (2009):

$$LB = \frac{\sum WCET_t}{\sum cores} + partial\ \ schedule\ \ length \qquad (7)$$

where the total WCET of unscheduled tasks, divided by the number of processing cores, is added to the partial schedule length. After that, we evaluate

```
Branch-and-Bound(ETG)
1  currentSolution = calculate initial solution with list scheduling
2  globalBest = currentSolution
3  while solutionList ≠ ∅ do
4   sort solutions in solutionList by priority function (best-first)
5   take the best from solutionList and branch
6
7   for each branched node in solutionList do
8     calculate upper and lower bounds for branched nodes
9     if(upper bound < globalBest) then
10        globalBest = upper bound
11
12    if(upper bound > globalBest or lower bound > upper bound) then
13       prune the node
14   end for
15
16 end while
end Branch-and-Bound;
```

Figure 5.3. Pseudocode of branch-and-bound

all candidate solutions. We prune the nodes that have its lower bound higher or equal to the global best upper bound. The process continues until there are no more nodes to expand.

An example of the branch-and-bound calculation process is depicted in Figure 5.4. The application consists of 100 tasks mapped on a 6x6 2D-mesh NoC. List schedule length is 9226 cycles while branch-and-bound result gives almost 15% of improvement (7882 cycles). However the calculation time increases 71 times from 0.12 seconds to 8.53.

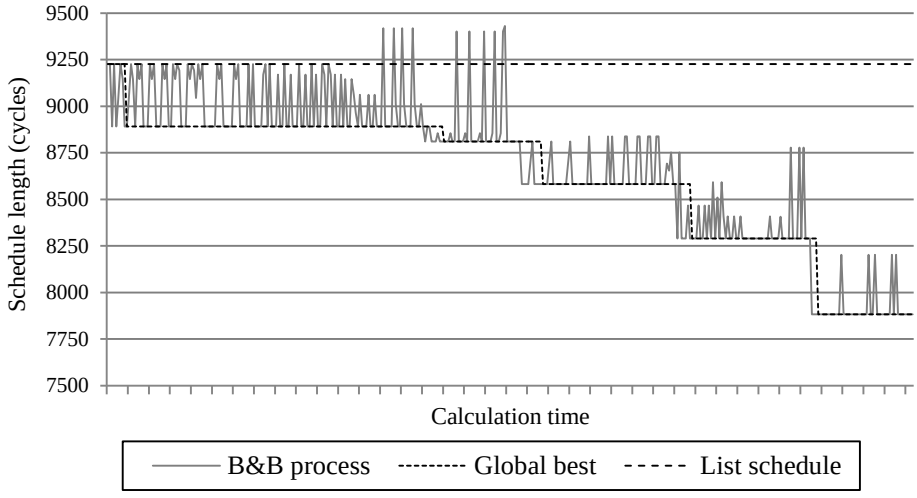


Figure 5.4. An example of branch-and-bound result conversion

5.3. Design optimization with simulated annealing

Simulated annealing (SA) is a probabilistic metaheuristic for the global optimization problem, locating a near-optimal solution in feasible time (Kirkpatrick, Gelatt, & Vecchi, 1983). Simulated annealing belongs to the class of non-optimal algorithms (Figure 5.1). It has been applied to several combinatorial optimization problems from various fields of science and engineering. These problems include for example travelling salesman problem (TSP), quadratic assignment, graph partitioning or linear arrangement. One of the main features of simulated annealing is the hill climbing capability – instead of accepting only improved solutions, bad solutions are accepted also with certain probability. That probability is gradually lowered, while finally only improved solutions are accepted. If an analogy is found from a problem at hand the simulated annealing is quite effective and simple to implement. However, it might need more time to tweak the initial parameters for a given problem.

5.3.1 Simulated annealing basics

The simulated annealing comprises of creating an initial solution that will be annealed by generating moves in the neighbourhood. Result (cost) of a move is calculated based on a target function and compared to the currently known best value. The cooling schedule controls the decrease of temperature, which has an effect on the move energy. The process continues until a termination condition has been met. The pseudocode of simulated annealing is depicted in Figure 5.5. In the next sections we will describe the main parts of the simulated annealing.

Initial parameters

In simulated annealing it is important to find feasible values for initial parameters: initial temperature (related with initial acceptance probability) and cooling schedule. When the initial temperature is too high, many bad uphill moves might be accepted driving SA far away from reasonable solution space. When it is too low, SA might not reach solutions which require more energy to cross higher hills to reach a global optimum. To estimate the initial temperature, we use a similar approach as proposed by Ledesma, Avina, and Sanchez in 2008. First, we create an initial solution. Next, we generate n number (e.g. 100 times) of random moves and record difference between initial solution and the random move. We calculate the average difference ($diff_{avg}$) of all random moves. The initial temperature can be estimated by the following formula:

$$T_{initial} = (1/-\log(p_{initial})) \times diff_{avg}^{2} \qquad (8)$$

where $p_{initial}$ is the initial probability to accept uphill moves. In our experiments we have used $p_{initial} = 0.9$.

```
Simulated Annealing (alpha, T_initial, N_allowed_temperatures, N_allowed_peturbations)
1   Construct initial solution S_current;
2   Current temperature T = T_initial;
3   Global best solution S_global = ∞;  N_temperatures = 0;
4
5   While N_temperatures < N_allowed_temperatures  Do Begin
6     Number of peturbations N_peturb = 0;
7     While N_peturb < N_allowed_peturbations Do Begin
8         Generate randomly a neighbouring solution S_neighbor
9         Calculate S_neighbor schedule length
10        ΔE = S_neighbor – S_current;
11
12        If ΔE<0 Then
13            S_current = S_neighbor;  N_peturb = 0;
14        Else
15          Generate random value r = uniform_random(0, 1);
16          If r ≤ e^(-ΔE/T) Then
17            S_current = S_neighbor;   N_peturb = N_peturb - 1;
18          Else
19            N_peturb = N_peturb + 1;
20          End If;
21        End If;
22    End;
23
24    If S_current <= S_global Then
25        S_global = S_current;  N_temperatures = N_temperatures + 1;
26    Else
27        N_temperatures = 0;
28    End If;
29    Set T = alpha x T;
30  End;
End Simulated Annealing;
```

Figure 5.5. Pseudocode of simulated annealing

Cooling schedule and acceptance criterion

We have used an exponential cooling schedule described by the following equation:

$$T_{new} = alpha \times T \quad (9)$$

where *alpha* is a constant (in our experiments usually between 0.7 and 0.96) and *T* current temperature.

An acceptance criterion is needed to overcome local optimums and accept uphill moves. One of the most common is the Metropolis acceptance criterion:

$$p_{acceptance} = e^{-\Delta E/kT} \quad (10)$$

where ΔE is object function difference between modified solution and current one, *T* current temperature and *k* Boltzman's constant (Metropolis, Rosenbluth, & Rosenbluth, 1953). The acceptance probability ($p_{acceptance}$) is compared to a random value generated uniformly in the range of *[0, 1]*. The probability accepting uphill moves will decrease with the temperature and/or with the increase of ΔE. Therefore, selection of initial temperature, probability and cooling schedule is important for the performance of SA.

Neighbourhood search

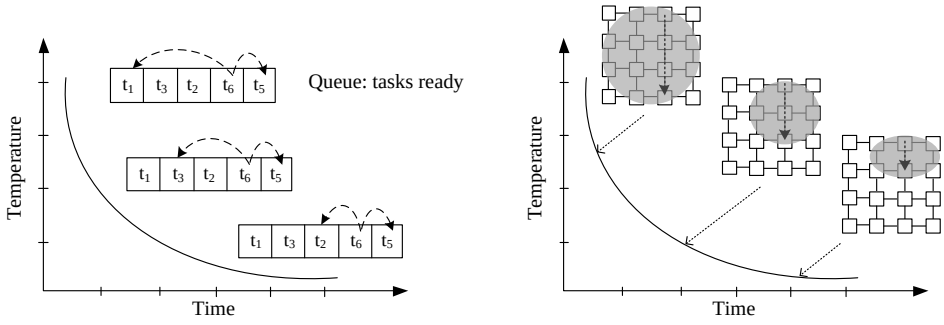
Depending on the optimization problem there can be different approaches to create neighbouring solutions. For example for schedule optimization task priority could be increased changing its position in the schedule. To optimize task mapping a randomly chosen task could be re-mapped to another processing core. It is important to keep in mind that a neighbouring solution should not restrict another move to revert the change in a later phase. Otherwise the process could be stuck in nearby local optimum and not be able to cross the hill.

Termination condition

In literature various termination conditions can be found. In our approach we specify the number of temperature levels ($N_{allowed\_temperatures}$) we are annealing at minimum, until we stop the process. The counter is incremented when we have found a better or equal solution compared to previous one. The counter is reset to zero when a higher result is found as we might have recovered from a local optimum and it would need further examination. At each temperature we are creating at minimum $N_{allowed\_peturbations}$ random neighbourhood solutions. The counter is incremented when an uphill move is rejected and decremented when a solution is accepted. The idea is essentially to keep the process on the same energy level until it stabilizes and only then lower the temperature.

5.3.2 Schedule optimization with simulated annealing

The schedule optimization problem we are trying to solve with simulated annealing can be formulated as follows.



a) shift tasks in ready queue

b) temperature controlled mapping

Figure 5.6. Two types of neighbourhood moves in simulated annealing

Definition 5.2 (Simulated annealing schedule optimization problem). *Given an application A = (T, C, wcet, comm), a network model N = (R, L) and a mapping function M(T→R) the simulated annealing optimization problem is to minimize the schedule length and to determine a near optimum feasible solution.*

When optimising the schedule (having fixed mapping) with simulated annealing, a neighbouring solution is created by selecting randomly a task and randomly increasing or decreasing its priority in the ready list. It is illustrated in Figure 5.6a. For this we are recording during scheduling the tasks that are ready at the same time. The distance a random task can be shifted in the queue is controlled by the following formula:

$$Distance_{allowed} = Size_{ready\_queue} \times (T / T_{initial}) \qquad (11)$$

where $Size_{ready\_queue}$ is the size of ready list, T is current temperature and $T_{initial}$ initial temperature.

Figure 5.7 depicts example results of simulated annealing schedule optimization having the same application and platform as with branch-and-bound. The uphill-downhill moves that are distinctive for simulated annealing are clearly illustrated in the figure. The simulated annealing schedule length is 7988 cycles, which is 13% better than a schedule produced by list scheduling (9226 cycles) and near to the branch-and-bound result (7882 cycles). The SA calculation time (96 seconds) is 800 times higher than for list scheduling and 11 times higher compared to B&B.

5.3.3 Task mapping optimization with simulated annealing

When schedule optimization does not give the required amount of improvement we can go for design space exploration and find an alternative mapping that might give better results. As the search space would be unreasonable big for

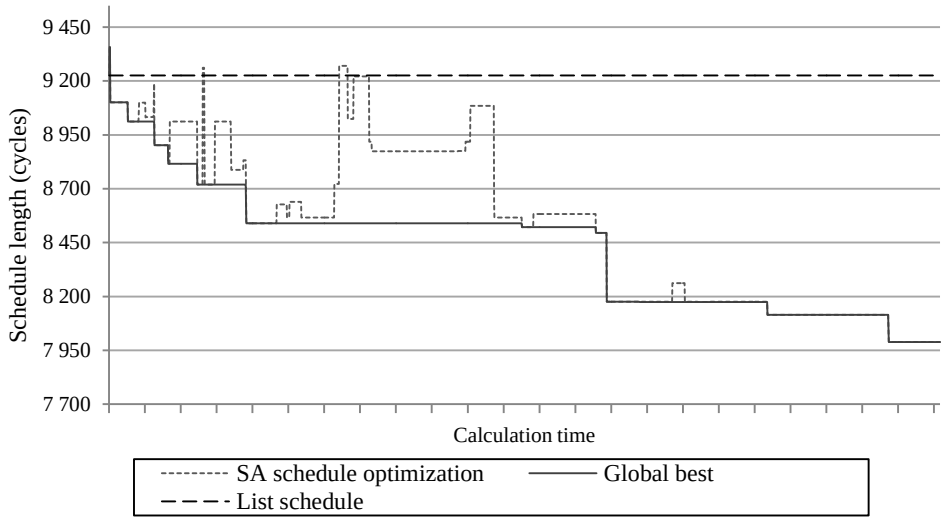


Figure 5.7. Simulated annealing schedule optimization

branch-and-bound we are using the simulated annealing to solve this optimization problem.

Definition 5.3 (Simulated annealing mapping optimization problem). *Given an application A = (T, C, wcet, comm), a network model N = (R, L) and an initial mapping function M(T→R) the simulated annealing optimization problem is to re-map the tasks in order to minimize the schedule length and to determine a near optimum feasible solution.*

Design space exploration is performed by selecting randomly a task and re-mapping it to another processing core (Figure 5.6b). The distance between original and re-mapped core (number of hops) is controlled by the cooling:

$$Distance_{allowed} = Number\_of\_hops \times (T / T_{initial}) \qquad (12)$$

where *Number_of_hops* is a random number in a range that depends on the current task location and the size of the NoC, T is current temperature and $T_{initial}$ initial temperature. In the beginning of the process, the distance can be higher while eventually reaching one hop (nearby cores). To evaluate cost of the move we need to perform each time communication synthesis and schedule the tasks and the communication. It is because our communication synthesis approach assumes that it is known which network resources a message traverses in its path. If that information changes re-synthesis of communication is needed.

Figure 5.8 depicts results of the simulated annealing mapping optimization. It can be seen that the mapping optimization gave the best schedule length 7264 cycles, compared to B&B: 7882 cycles. However, the calculation time was around 11 times higher (96 seconds) compared to B&B. As simulated annealing produces near optimum results we have confirmed the quality of the achieved

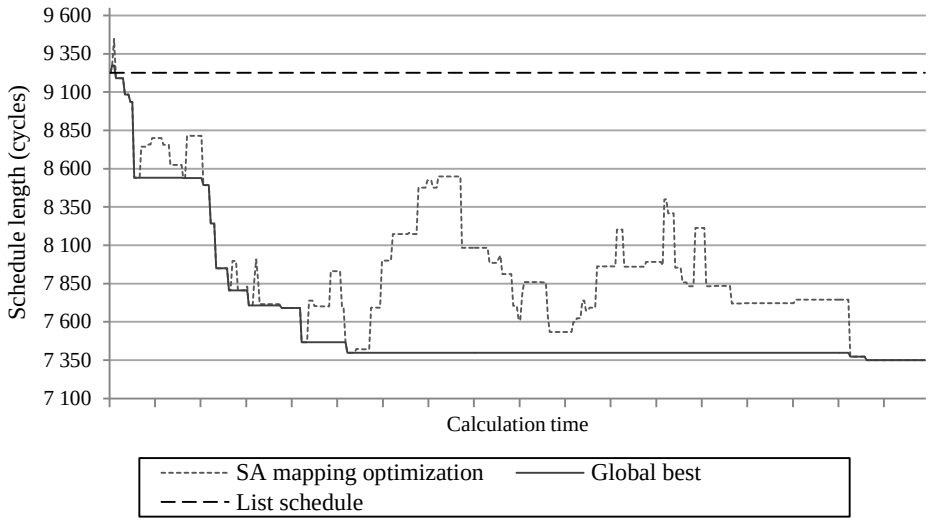


Figure 5.8. Simulated annealing mapping optimization

result by performing an experiment were the same application with 100 tasks and 30 arbitrary mappings were given and mapping optimization was run again with simulated annealing. An average schedule length of 7324 cycles was achieved, which is less than 1% difference compared to the previous best result. It shows that general purpose heuristics can be used effectively to solve specific design space exploration problems.

5.4. Experimental results

To show scaling of the branch-and-bound and simulated annealing optimization algorithms we have chosen synthetic task graphs, containing 100, 500, 750 and 1000 tasks. The NoC platform we have used in this set of experiments is a 6x6 2D-mesh, link bit-width of 32 bits, packet-size of 512 bits, and dimension-order XY routing. The computing resources are homogeneous – the task WCET is the same on all resources. The tests were performed on a computer with Intel L2400 CPU (1.66 GHz), 1 GB of available physical RAM. As simulated annealing depends heavily on the quality of random number generator we have run 20 tests in a batch varying the random number generator seed. We depict the minimum, maximum and average result. The SA initial parameters were chosen as described in Section 5.2. The initial task mapping was given.

The results in Table 5.1 show that all of the optimization techniques have produced a better schedule than the list scheduling (LS). However, this is achieved at the expense of increased calculation time. For applications with 750 and 1000 tasks, the schedule optimization with SA was not able to produce a result in feasible time. When we compare different optimization techniques it

can be seen, that branch-and-bound has given 10-15% of improvement with lowest calculation time. As we create only valid solution branches in the B&B this reduces rapidly the number of iterations reaching the solution faster than simulated annealing. The simulated annealing performance could be increased by a more efficient way to generate the neighbourhood where a change of existing solution is created. Mapping improvement with SA reached a similar schedule optimization as B&B, but the calculation time explodes with the increase of application size. It is important to note, that in the mapping optimization communication must be synthesized for each modified solution. In the schedule optimization, it is needed only to re-schedule the tasks and communication that is less time consuming.

5.5. Conclusions

One of the main goals of this chapter is to show that arbitrary scheduling or optimization technique can be used together with our contention-aware scheduling method. We have presented branch-and-bound and simulated annealing optimization techniques in order to improve the initial schedule and/or task mapping. Branch-and-bound is a general algorithm for finding optimal solutions for various computationally intensive optimization problems. Simulated annealing is a probabilistic metaheuristic for the global optimization problem, locating a near-optimal solution in feasible time. We have used branch-and-bound to improve the initial schedule. Simulation annealing has been used to improve the initial schedule and task mapping. The experimental results show, that our communication modelling technique can be used with both of the presented optimization methods, gaining improvement in schedule length by re-ordering or re-mapping the tasks.

Table 5.1. Comparison of optimization techniques

Number of tasks	Schedule length, cycles								Calculation time, minutes			
	List schedule	Branch-and-bound	SA			Mapping with SA			LS	B&B	SA	Mapping SA
			Min	*Avg*	*Max*	*Min*	*Avg*	*Max*				
100	9 226	7 882	7 988	8 488	8 719	7 264	7 367	7 998	0.003	0.1	1.3	1.8
500	18 496	16 728	18 019	18 112	18 206	16 916	17 522	18 265	0.03	13.7	34.7	18.2
750	32 127	28 932	NA	NA	NA	28 150	29 217	30 193	0.05	35.5	NA	33.6
1000	36 772	33 103	NA	NA	NA	33 642	33 733	34 086	0.10	85.9	NA	267.3

Chapter 6. System-Level Fault Tolerance Improvements

In the previous chapters we have had focus on predictable and effective system-level design of real-time NoC-based SoCs. The objective has been to optimize the designs in terms of schedule length and mapping. The assumption so far has been that the network-on-chip is able to provide the required communication services at any given (scheduled) time. According to Murali et al. (2005) and Constantinescu (2003) shrinking feature sizes towards nanometer scale cause power supply and threshold voltage to decrease, consequently making wires unreliable. The wires will be increasingly susceptible to different noise sources such as crosstalk, coupling noise, soft errors and process variations. To design a predictable and reliable system one would need to analyse the sources of faults and the probability of their occurrence. Based on the analysis fault-avoidance or fault tolerance techniques should be used to increase the reliability of the system.

Part of the results described in this chapter have been published in the following book chapter:

Tagel, M., Ellervee, P., & Jervan, G. (2011). System-Level Design of NoC-Based Dependable Embedded Systems. Ubar, R.; Raik J.; Vierhaus, H. T. (Eds.). In *Fault-Tolerance and Applications in System-on-Chip Design: Advancements and Techniques* (pp. 1 - 36). Hershey, Pennsylvania, USA: IGI Global.

6.1. Motivation

Faults in semiconductor devices can be divided based on their occurrence into three classes – permanent, transient and intermittent. As described in Section 2.6.1 permanent faults cause permanent malfunctioning of system components, while transient and intermittent faults appear for a short period of time. According to Sosnowski (1994) and Constantinescu (2003) improvements in semiconductor design and manufacturing processes have significantly decreased the occurrence of permanent faults. Conversely, the technology scaling has increased process variability causing higher number of transient and intermittent faults – due to which the device performance might vary in space and time. Sources of variability are transistors, interconnects, and the operating environment (Nikolic, Park, Kwak, & Giraud, 2011). Therefore, with current and future technology nodes variability and reliability are important issues to be addressed together. One way to handle the problem is to keep a relatively high design margin to guarantee the required yield. Another way is to increase the system reliability by employing data encoding, duplicating system components

or software-based fault-tolerance techniques. There are several techniques that we discussed in Chapter 2.6.3, which concentrate on low level reliability analysis and improvement in SoC design. We want to fill the gap between the application and the underlying network-on-chip and bring the reliability analysis to the system level together with the communication aspects.

Murali et al. (2005) proposed three error detection and correction schemes, end-to-end flow control (network level), switch-to-switch flow control (link level) and combined approach that can be used to protect NoC communication links from transient faults. In the end-to-end error detection scheme parity or cyclic redundancy check (CRC) codes are added to packets while in switch-to-switch scheme the error detection code is added either to flits or packets. In the end-to-end error detection scheme the receiver network interface sends acknowledgement to the sender if the received data was correct or requests to re-send it in case it was not able to correct the faulty data. If a switch detects an error on a packet's header flit, it drops the packet. In the switch-to-switch error detection scheme the faulty flits or packets are re-transmitted between the switches. In the hybrid single-error-correcting and multiple-error detecting scheme the receiver corrects any single bit error on a flit. When multiple bit errors occur there is end-to-end re-transmission of data from the sender network interface necessary. The authors are interested in power consumption overhead when implementing such error detection schemes. The work focuses only in network and link level. As the authors state in the conclusions further work in NoCs should include investigating the effects of application- and software-level reliability schemes and developing online adaptation capabilities to increase fault-tolerance of NoC-based systems. In 2007, Rantala, Isoaho and Tenhunen motivated the shift from low level testing and testability design into system-level fault-tolerance design. They propose an agent-based design methodology that helps bridging the gap between applications and re-configurable architectures in order to address the fault-tolerance issues. They add a new functional agent/control layer to the traditional NoC architecture. The control flow of the agent-based architecture is divided hierarchically to different levels. The granularity of functional units on the lowest level is small and grows gradually when raised on the levels of abstraction. For example the platform agent at the highest level controls the whole NoC platform while a cell agent monitors and reports status of a processing unit to higher level agents. Rusu, Grecu and Anghel (2008) propose a coordinated checkpointing and rollback protocol that is aimed towards fast recovery from system or application level failures. The fault-tolerance protocol uses a global synchronization coordinator Recovery Management Unit (RMU) which is a dedicated task. Any task can initiate a checkpoint or a rollback but the coordination is done each time by the RMU. The advantages of such an approach are simple protocol, no synchronization is needed between multiple RMUs, less hardware overhead and power consumption. The drawback is the single point of failure – the dedicated RMU itself. Zhang, Han, Xu, Li and Li (2009) introduce virtual topology that allows using spare NoC cores to replace faulty ones and re-configure the NoC to

maintain the logical topology. A virtual topology is isomorphic with the topology of the target design but is a degraded version. From the viewpoint of programmers and application, they always see a unified virtual topology regardless of the various underlying physical topologies. Ababei and Katti (2009) propose a dynamic remapping algorithm to address single and multiple processing core failures. Remapping is done by a general manager, located on a selected tile of the network.

In this chapter we present a system level approach to handle a given number of transient or intermittent faults. The fault-tolerance requirements are taken into account during the task and communication scheduling to allocate appropriate slack time to execute recovery actions. It is an application level software-based approach to increase the reliability of NoC-based real-time systems taking into account the communication aspects. In literature different fault models exist for on-chip interconnect and system components. Therefore, first we describe our system model with dependability requirements to put the work into a concrete context. Second, we introduce shifting-based scheduling (SBS) to schedule applications with dependability requirements, followed by experimental results.

6.2. System model with dependability requirements

As the basis of this work we are using the same system model and NoC architecture as described in Chapter 3. However, dependability brings new aspects that we have to take into consideration. We assume that the system is able to detect transient or intermittent faults and perform a corrective action. These faults can appear during task execution and data transmission. A transient fault in a processing node can be detected with special techniques such as watchdogs or signatures that are easy to implement and have a low overhead. The time to detect a fault is represented by *error-detection overhead* t_{det}. Once a fault is detected, inputs of the process will be restored and the task can be re-executed k times. The time to recover the inputs is characterized by the *recovery overhead* t_{rec}. For the sake of simplicity we assume in the remaining chapter that the error detection overhead t_{det} is included in task worst-case execution time.

We are using an end-to-end re-transmission scheme where the sender adds CRC error detection code to the original packet and the receiver checks the received data for correctness. If a faulty and unrecoverable packet is received at the destination the source task is requested to re-send the data. A faulty packet can be re-sent r times. The error-detection t_{det} overhead is a constant for data communication while the recovery overhead t_{rec} depends on the task mapping. In the current approach we do not handle delay faults. We assume that the task is executed in its WCET timeframe and the communication is transmitted during the scheduled time period. However, delay faults causing deadline misses can be caught by the online scheduler that exists in each processing core. Figure 6.1 depicts an example of a task re-execution and a packet re-transmission scenario. Task t_1 experiences a fault that is detected. The initial inputs are restored and

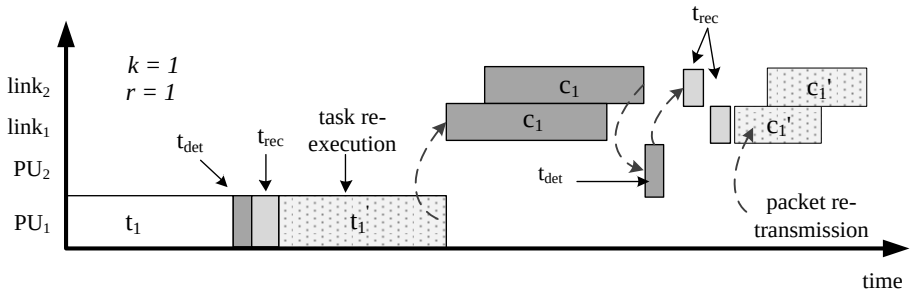


Figure 6.1. An example of re-execution and re-submission

task t_1 is re-executed. Task t_1 will send the result c_1 to processing unit PU_2. The processing unit PU_2 will detect the fault in one of received packets and will request to re-send it. Finally, the faulty packet c_1 is re-sent. The example schedule of the application is able to handle one task re-execution and one packet re-transmission. However, there can be more than one flipped bit in a task or a packet. Therefore, the error detection technique should be carefully selected to detect single or multiple faults in the data.

We have used the contention-aware scheduling described in Chapter 3 and combined it with shifting-based scheduling (SBS) (Izosimov, 2006) to produce schedules that meet the fault-tolerance requirements set by the designer at the system level. The scheduling is performed offline. The resulting schedule is stored in each processing core schedule table. The schedule table contains only the fault-free execution scenario and additionally information of how many times a task could be re-executed or a packet re-sent to still keep the deadlines. There is an online scheduler in each processing core that takes care of maintaining the current execution scenario. The input problem we are solving can be summarized as follows.

Definition 6.1 (Task graph with dependability requirements). *A task graph with dependability requirements corresponds to the Definition 3.1 having in addition information about k number of task re-executions and r number of packet re-transmissions an application is able to tolerate. The input contains also the CRC bit-length and polynomial.*

6.3. Fault-tolerant application schedules

The work of Izosimov (2006) describes system-level scheduling and optimizations of fault-tolerant embedded systems in bus-based systems. The work describes various techniques at the system level such as re-execution, rollback recovery with checkpointing and active/passive task replication. However, their work considers faults only in computational tasks. The

communication fault-tolerance is not a part of their system model. It is assumed that the faults occurring during data transmission are handled by the communication subsystem by the use of error correction codes or through hardware replication of the bus. The hardware overhead of such scheme would be hard to justify for a NoC-based system. The number of network resources (communication links, routers) would be much higher than in a bus-based system. We could apply one of the software fault-tolerance techniques, proposed by Izosimov (2006) for bus-based systems, to the NoC domain but the complexity increases when we try to manage slack time across all communication links and communication flows. Moreover, in a bus-based system the task mapping does not have such influence on communication delays as in a NoC-based system. Therefore, we need a method that would take into account both fault-tolerance and communication aspects of NoCs, trading-off some system performance for modelling speed.

The basis of our work is shifting-based scheduling approach proposed by Izosimov (2006). We have extended it to contain on top of the task dependability requirements also the communication fault-tolerance requirements and integrated it with our contention-aware scheduling. Shifting-based scheduling is an extension of the transparent recovery against single faults. A fault occurring on one computation node is masked to other computation nodes. It has impact only on the same computation node. According to Izosimov (2006) providing fault containment, transparency can potentially improve testability, debugability and increase determinism in fault-tolerant applications. A good property of shifting-based scheduling is that the start time of communication is fixed (frozen). It reduces the modelling complexity and fits into our communication model where communication is scheduled at fixed time periods. Moreover, we do not need a global real-time scheduler to synchronize a local recovery event with other cores in the case of fault occurrence. A downside is that SBS cannot trade-off transparency for performance – in case of fault-free execution scenario the slack periods are left in the schedule because the communication is scheduled to start at fixed time periods.

The scheduling problem we are solving with SBS can be formulated as follows.

Definition 6.2 (Shifting-based scheduling problem). *Given an application A = (T, C, wcet, comm) with dependability requirements k and r, a network model N = (R, L) and a mapping function M(T→R) we are interested to find a schedule S such that the worst-case end-to-end delay is minimized and the transparency requirements with frozen communication are satisfied*.

In 2006, Izosimov proposed a Fault-Tolerant Conditional Process Graph (FT-CPG) to represent an application with dependability requirements. FT-CPG captures alternative schedules in the case of different fault scenarios. Graphically FT-CPG is a directed acyclic fork-and-join graph where each branch corresponds to a change of condition. However, for SBS the full FT-CPG graph is not necessary to generate – instead, all possible execution scenarios are

```
Shifting-based-scheduling(A, N, k, r)
1    calculate task priorities
2    add task with pred(t) = ∅ into ready task list
3    while tasks not scheduled ≠ ∅
4      for each task t_current in ready task list do
5        calculate recovery slack sl_current for task t_current based on k
6        schedule t_current with recovery slack sl_current
7        add first vertex of outgoing communication message into comm. list
8        remove scheduled task t_current from ready list
9      end for
10     sort communication list based on ASAP schedule
11     for each communication message c_current in comm. list do
12       schedule communication message c_current on the whole route
13       for r packets to be tolerated do
14         for each traversed comm. link of c_current from dest. to source do
15           reserve communication bandwidth for recovery request
16         end for
17         for each traversed comm. link of c_current from source to dest. do
18           reserve communication bandwidth for re-submission of a packet
19         end for
20       end for
21     end for
22     add eligible successors of scheduled tasks to ready task list
23     sort ready task list based on mobility
24   end while
end Shifting-based-scheduling
```

Figure 6.2. Extended shifting-based scheduling algorithm

considered during scheduling. Therefore the graph complexity (number of vertexes and edges) stays the same as for contention-aware scheduling while there is a slight increase in the number of scheduling steps.

The pseudocode of extended shifting-based scheduling algorithm is depicted in Figure 6.2. Input for the SBS is an application *A*, a NoC architecture *N* with a mapping function *M*, a number of transient faults *k* to be tolerated in any processing core and a number of faulty packets *r* to be tolerated during data transmission. The main part of the pseudocode follows the contention-aware scheduling algorithm to schedule the tasks and communication. First, priorities of tasks are calculated based on mobility and the source task (*pred(t)* = ∅) is put into the ready to schedule list (lines 1-2). Scheduling loop (lines 3-24) is processed until all tasks and communication messages have been scheduled. Before a task is scheduled we need to calculate the recovery slack (line 5). It is calculated in three steps as described below:

1. The idle time t_{idle} between the task being scheduled $t_{current}$ and the last scheduled task t_{last} on the same processor is calculated as:

$$t_{idle} = t_{current} - t_{last} \qquad (13)$$

2. Initial recovery slack $sl_{initial}$ of task $t_{current}$ is calculated as:

$$sl_{initial} = k \ x \ (t_{current}^{WCET} + t_{rec}) \qquad (14)$$

where k is number of required recovery events, $t_{current}^{WCET}$ worst-case execution time of a task being scheduled and t_{rec} time needed to restore the initial inputs. Recovery overhead t_{rec} has a constant value.

3. We calculate the slack difference sl_{diff}

$$sl_{diff} = sl_{last} - t_{idle} \qquad (15)$$

where sl_{last} is the slack time of last scheduled task. The recovery slack $sl_{current}$ of task $t_{current}$ is changed if recovery slack of previous task sl_{last} subtracted with the idle time t_{idle} (Formula 15) is larger than the initial slack $sl_{initial}$. Otherwise initial recovery slack is preserved. This is summarized by the Formula 16.

$$sl_{current} = \begin{cases} sl_{initial}, if & sl_{diff} < sl_{initial} \\ sl_{diff}, if & sl_{diff} > sl_{initial} \end{cases} \qquad (16)$$

SBS is adjusting the recovery slack to accommodate recovery events of tasks mapped to the same processing core and will schedule communication to the end of the recovery slack. There is an example of fault-tolerant schedule depicted in Figure 6.3. Tasks t_1 and t_2 have a recovery slack that can accommodate one of the two tasks re-execution ($k = 1$). The figure also depicts the worst-case scenario for task t_1 in case of $k = 1$ transient faults.

So far we described how shifting-based scheduling approach can be used to calculate recovery slack for tasks mapped on the same core. The start time of the communication messages is frozen (fixed) – they are scheduled at the end of the task(s) recovery slack. Next, we will look the extension of the SBS algorithm to handle the communication fault-tolerance.

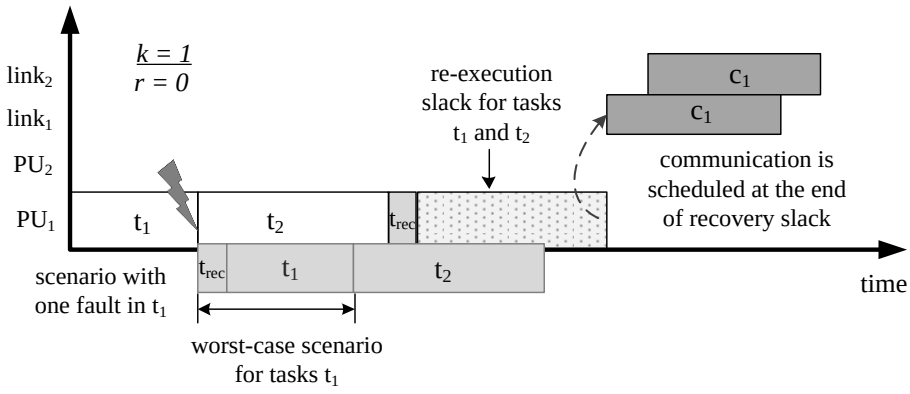


Figure 6.3. An example of SBS application schedule

Communication fault-tolerance

In Chapter 3 we have described the contention-aware communication synthesis and scheduling. In this chapter we will extend it to include the fault-tolerance aspects. We can model the communication at different levels of granularity – message or packet level. In message-based communication a stream of flits or packets that make up a message are scheduled as one piece. In packet-based communication model packets that make up a message might not be scheduled all to consecutive time periods. Chapter 4.1 described the trade-off and advantages of packet-based schedules versus message-based schedules. However, when we talk about communication fault-tolerance we assume that the data is sent in a form of packets even when the NoC supports sending the whole message as a stream of flits. There is an analogy with rollback recovery with checkpointing. In that technique the time needed for task re-execution can be reduced by taking checkpoints at specified time and restoring the last non-faulty state of the failing task. Several approaches exist for distributing the checkpoints. These can be inserted into places where saving of the process state is the fastest. This is application-specific and requires knowledge on application behaviour. Another approach is to insert the checkpoints systematically – for example at equal intervals. Splitting the message into smaller units (packets) and checking each received packet can more effectively identify the part of the faulty data and require less time to re-send it.

One of the main assumptions in our system-level fault-tolerance technique is the ability to detect faults. We protect packets by adding cyclic redundancy code proposed by Peterson and Brown in 1961. The basic idea of CRC is to divide a message by a fixed binary number (*CRC polynomial*) and to make the remainder from this division the checksum. Upon receipt of a message, the receiver can perform the same division and compare the remainder with the checksum. CRC is characterized mainly by the size of the CRC checksum and the polynomial. For interconnection networks, the property of interest is also the Hamming Distance (HD) that is the minimum possible number of bit inversions that must be injected into a message to create an error that is undetectable by the checksum. The paper of Koopman and Chakravarty in 2004 analyses this problem and gives an overview of different CRC sizes/polynomials and their hamming distances.

In our approach each packet contains CRC error detection code and we can specify the number of packets (parameter r) we want the application to be able to re-send. CRC increases slightly amount of transmitted data but it has no major effect on the schedule length as we see later in the experimental results. Figure 6.4 depicts two communication scenarios. In both examples communication is modelled at message granularity. It means that a message is scheduled as one piece to consecutive time periods. Figure 6.4a depicts the communication schedule without a recovery slack. In Figure 6.4b we have set the requirement to tolerate transient faults in one packet ($r = 1$) in each communication message and reserve the network resources accordingly. From this small example we are

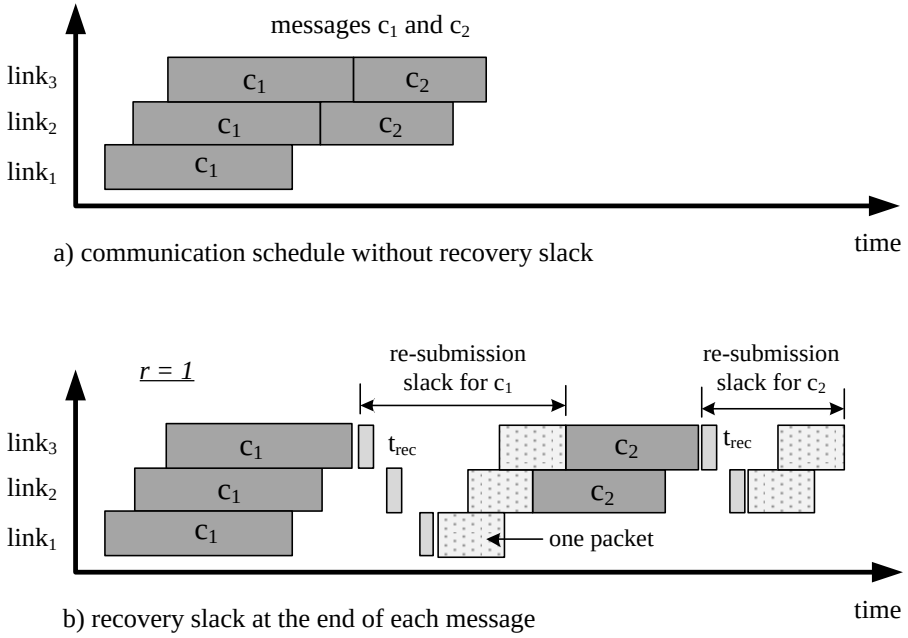


Figure 6.4. Different communication scenarios

able to see the complexity we have to handle and how the communication delays affect the slack allocation.

Lines 13-21 of shifting-based scheduling pseudocode in Figure 6.2 depict the slack allocation for communication fault-tolerance. We reserve separately recovery slack for a 2-flit packet (lines 14-16) that is sent from receiver to the sender noting the failed packet identifier and requesting it to be re-sent. We are using another CRC polynomial and CRC size to protect and to correct faults in relatively short failed packet identifier. It is because there is no additional re-submission scheme for those flits. The sender must be able to understand which packet to re-send. Separate slack time in the schedule is reserved for the re-submitted packet (lines 17-19). This process is iterated *r* number of times. There is a trade-off between packet size and amount of CRC data versus schedule length. Experimental results of this chapter contain a comparison of schedule length having different fault-tolerance requirements and different packet sizes.

Extraction of runtime schedules

Shifting-based-scheduling provides a trade-off between performance and complexity of the scheduling (in terms of memory, calculation time). SBS does not require complete scheduling of all possible execution scenarios. It relies on an online scheduler to maintain the current execution schedule and to switch to a contingency schedule. It fits together with our system-level model and application simulation kernel described in Chapter 3.6. We would need

additionally to store in each local scheduler how many times a task can be re-executed and a packet to be re-sent.

If a fault is detected at the run-time of an application, local scheduler will switch to a contingency schedule by re-executing the failed task and delaying the start time of related successor tasks on the same processing core. The outgoing messages will be sent at their respective scheduled time moments. At the receiver side we store for each incoming message a number of packets that the received message should consist of. Information about faulty received messages can be sent out by receiver to the sender at the end of each message transmission. The receiver will send out a special message noting the transaction identifier of a faulty packet. The sender receives the request and re-sends the faulty packet. This process can be iterated *r* times that correspond to the number of re-submission events our application is expected to handle.

6.4. Experimental results

We have performed the experiments with the same set of synthetic task graphs as described in Chapter 3. Each of the ten benchmark applications (*a1* - *a10*) consist of 2500 tasks with varying average vertex degree (1.5 - 5) and computation-to-communication ratio (0.1 - 3.2). The NoC architecture parameters are chosen as follows: link bit-width and flit size 36 bits, header size 20 bits, channel delay 1 cycle, switching delay 4 cycles and path setup delay 8 cycles. The experiments are performed on a computer with Intel Core i5-520M (2.40 GHz) processor and 4 GB of available physical RAM.

Packet size versus SBS schedule length

In the first set of experiments we have chosen the benchmark application *a1* consisting of 2500 tasks mapped on a 8x8 2D-mesh NoC. We have set the dependability parameters $k = 0$ and $r = 1$. The CRC size is between 9 to 12 bits depending on the size of the packet. Moreover, packet re-transmission slack period depends on the packet size. The hamming distance is 4, which is the minimum possible number of bit inversions that must be injected into a packet to create an error that is undetectable by corresponding CRC sequence. The different CRC sizes and related polynomials are described in the paper by Koopman and Chakravarty (2004).

Our goal is to explore how the packet size, that includes also error detection overhead, affects the computation-to-communication ratio and the schedule length. The results are depicted in Figure 6.5. We can see that the packet size has a big influence on the schedule length. There is an analogy with rollback recovery with checkpointing. Splitting a message into smaller units (packets) can more effectively identify the part of the faulty data and would require less time to re-send it. However, we can see in the Figure 6.5 that if the

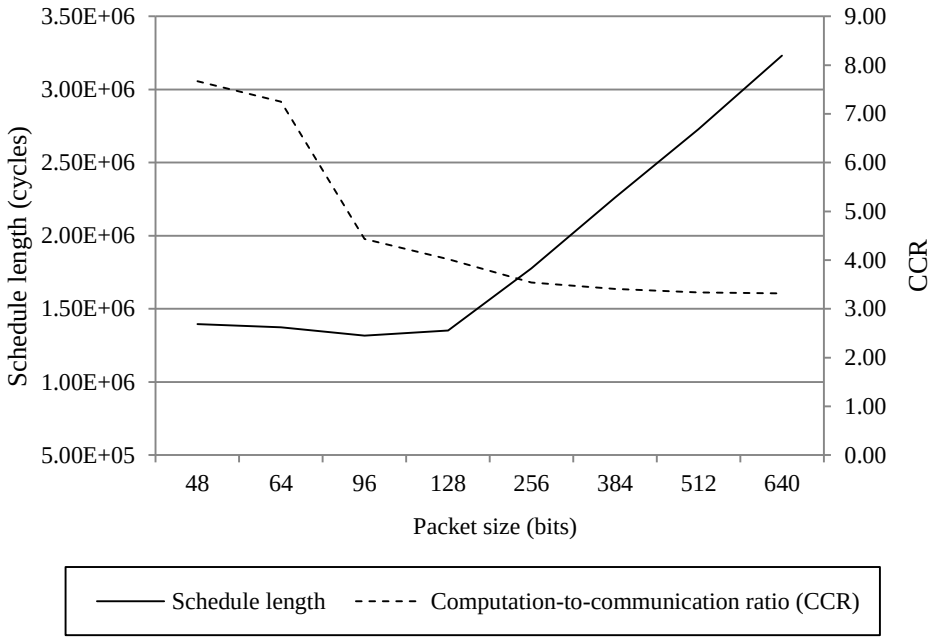


Figure 6.5. SBS schedule length depending on packet size (r = 1)

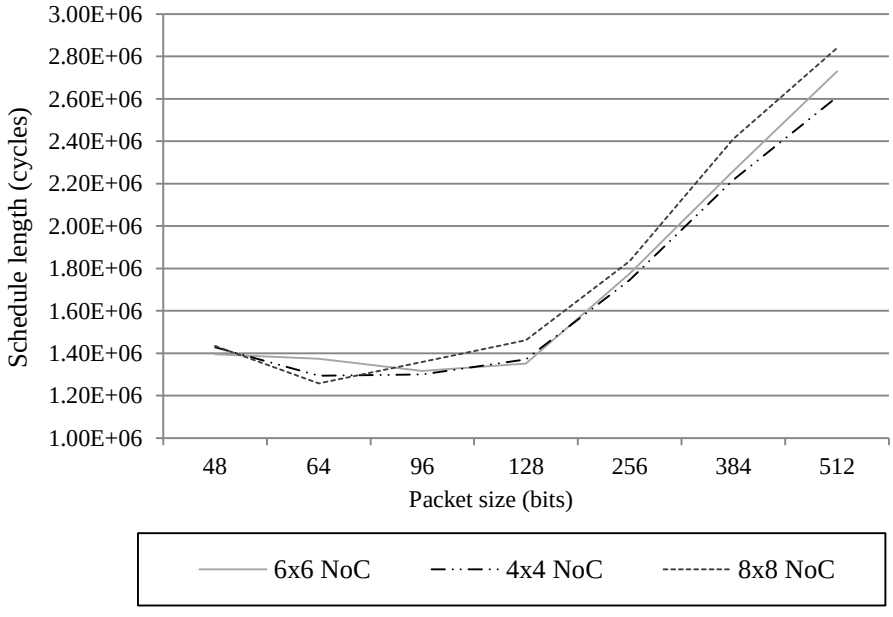


Figure 6.6. SBS schedule length for different NoC and packet sizes (r=1)

packet size goes below 96 bits the CCR jumps up and starts affecting the schedule length negatively. We have performed similar experiments with benchmark application *a1* for different NoC sizes to see how the NoC size affects the schedule length. The results are depicted in Figure 6.6. We can see the similar trend as in our first experiment. To confirm these findings we have run experiments also with the rest of the 9 benchmark applications (*a2*-*a10*) and the conclusion is that a favourable packet size for SBS is between 96 and 128 bits.

Dependability parameters

In the second set of experiments we are changing the SBS dependability parameters *k* (number of task re-executions) and *r* (number of packet re-transmissions). Our goal is to find out how the combined fault-tolerance in tasks and communications affects SBS schedule length and modelling speed. We have mapped application *a1* from our synthetic task graph set to a 8x8 2D-mesh NoC and varied the parameter *k* and *r* values. We have chosen packet size 128 bits that we have found out to be suitable from previous experiments.

The results are depicted in Figure 6.7. The labels in the graph x-axis show the respective *k* and *r* parameter values. We can see that increasing the *k* parameter value by one produces approximately 1.5 times longer schedules. We see a sharp increase in the schedule length when the number of packet re-transmissions (parameter *r* value) is increased. The schedule length increases approximately 3.5 times compared to the initial schedule without recovery slacks. The communication dependability overhead will start to dominate. The overhead is affected by the task mapping – the end-to-end re-transmission scheme reserves slack time for each communication link the packet traverses. As in previously described model we assume no flit interleaving no other communication is allowed to be sent on those communication links. The experimental results of this set of tests are summarized in Table 6.1. The SBS algorithm has virtually no impact on the schedule calculation speed compared to the case where no dependability is selected and original contention-aware scheduling has been used. However, a designer needs to select suitable *k* and *r* parameter values to find a trade-off between schedule length and dependability improvement.

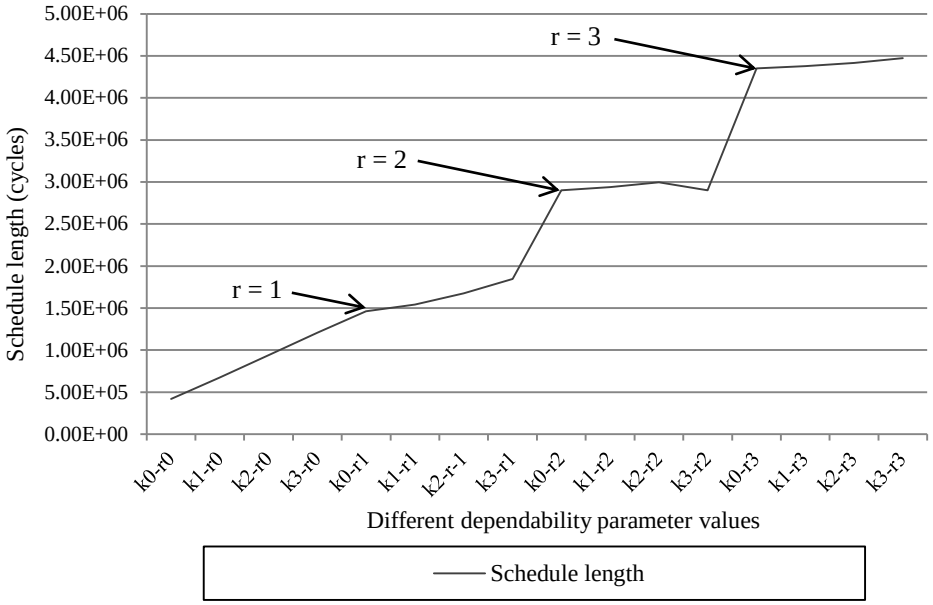


Figure 6.7. SBS schedule length for different dependability parameters

Table 6.1. Shifting-based scheduling – performance/dependability trade-off

Level of dependability		**Schedule length (cycles)**	**Increase of initial schedule length (x times)**	**Modelling time (seconds)**
r	*k*			
* Initial schedule length without dependability and no CRC in communication				
0	**0**	*** 419 697**	**1.0**	**11.9**
	1	674 448	1.6	12.0
	2	938 482	2.2	11.7
	3	1 204 977	2.9	11.7
1	**0**	**1 461 567**	**3.5**	**12.5**
	1	1 543 059	3.7	12.5
	2	1 674 128	4.0	12.4
	3	1 846 625	4.4	12.5
2	**0**	**2 900 645**	**6.9**	**12.9**
	1	2 938 034	7.0	12.6
	2	2 998 153	7.1	12.9
	3	2 900 645	6.9	13.0
3	**0**	**4 350 754**	**10.4**	**13.6**
	1	4 379 552	10.4	13.3
	2	4 418 198	10.5	13.9
	3	4 472 463	10.7	13.4

SBS mapping optimization by simulated annealing

As we saw previously the SBS schedule length is affected in a large scale by the number of packet re-transmissions (parameter *r*). Increasing the parameter *k* value by one produced approximately 1.5 times longer schedules. In both cases the results depend on the task mapping. If we would map the tasks to nearby cores we might get shorter packet re-submission slacks at the cost of more network contentions. We have used simulated annealing task mapping optimization, described in Chapter 5.3, to explore the amount of improvement we could gain out of task re-mapping. We try to minimize the network traffic by mapping the tasks only to nearby processing cores finding a trade-off between parallelism and inter-processor communication. The simulated annealing algorithm and the selection of initial parameters are described in Chapter 5.3. In this set of experiments the temperature in simulated annealing algorithm controls the distance were a task can be re-mapped. The cost of the move (recovery slack and the schedule length) is calculated by the extended shifting-based scheduling algorithm. The results are depicted in Figure 6.8. The x-axis describes different combinations of parameter *k* and *r* values. The y-axis depicts the schedule length. We can see that by optimizing the initial mapping we were able to gain up to 10% of schedule length improvement at the cost of increased calculation time (around 100 times higher). It is not as much improvement as we have expected. By increasing the number of packet re-transmissions (*r*) the schedule length is still affected mostly by the communication slack. One of the problems

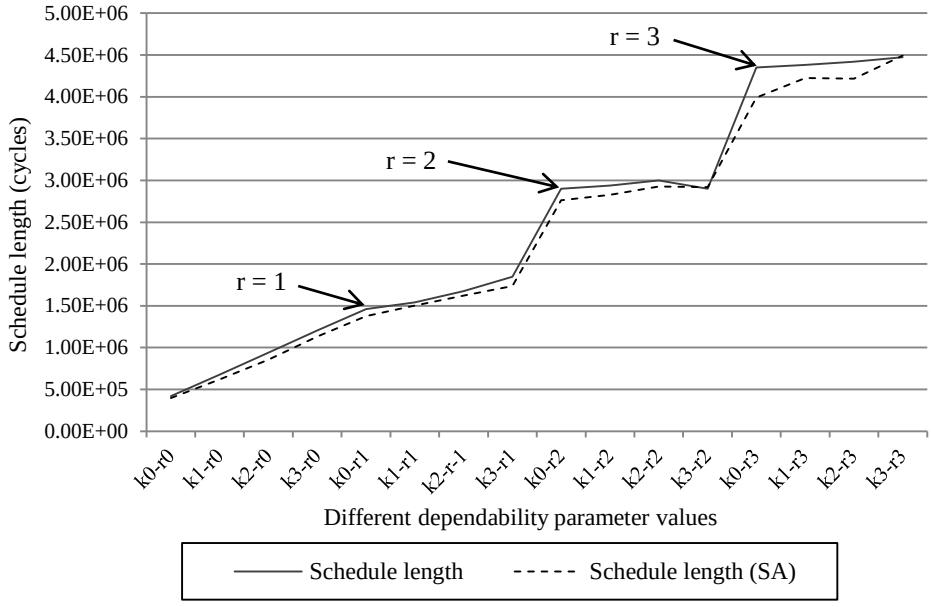


Figure 6.8. Mapping optimization for different dependability parameters

is in the deterministic XY routing that does not balance the network load between the communication links. A future work is to find a better deterministic routing algorithm that is able to distribute the network load more evenly.

6.5. Conclusions

In this chapter we have described a system-level technique to tolerate transient and intermittent faults in tasks and communications. We assume that a system is able to detect transient or intermittent faults and perform a corrective action. We are using an end-to-end re-transmission scheme where the sender adds CRC error detection code to the original packet and the receiver checks the received data for correctness. If a faulty and unrecoverable packet is received at the destination the source task is requested to re-send the data. The fault-tolerance requirements are taken into account during the task and communication scheduling to allocate appropriate slack time to execute recovery actions. For these purposes we have extended the shifting-based scheduling approach proposed by Izosimov (2006). We have extended it to contain on top of the task dependability requirements also communication fault-tolerance requirements and integrated it with our contention-aware scheduling. If a fault is detected at run-time of an application, local scheduler will switch to a contingency schedule by re-executing the failed task and delaying the start time of related successor tasks on the same processing core. The experimental results show that when scheduling application with modified shifting-based scheduling the packet size has a big influence on the schedule length. There is an analogy with rollback recovery with checkpointing. Splitting the message into smaller packets can more effectively identify the part of the faulty data and would require less time to re-send it. The modified SBS algorithm has virtually no impact on the schedule calculation speed compared to the case where no dependability is selected and original contention-aware scheduling has been used. However, a designer needs to select suitable SBS dependability parameter values to find a trade-off between schedule length and improvement in system fault-tolerance.

Summary

Conclusions

This thesis concentrates on system-level design issues of real-time NoC-based systems-on-chip. Networks-on-chip have been proposed as one of the alternatives to solve the on-chip communication scalability problems and to address dependability at various levels of abstraction. Communication modelling and synthesis plays an important role in the design of such complex systems. Trying to guarantee the observance of timing constraints without detailed know-how of communication transactions might lead to unexpected results.

We propose our system-level design framework for real-time network-on-chip based systems-on-chip. In the framework an application is described by a directed acyclic graph (task graph). Each task has given worst case execution time. Graph edges carry the amount of data that need to be exchanged between given pair of tasks. Once the tasks have been mapped to corresponding processing cores in NoC we perform communication synthesis – for each communication link the message traverses, a new vertex is added into the task graph transforming it into an extended task graph. The extended task graph is scheduled using contention-aware scheduling method. The resulting schedule is verified by executing the schedule on a NoC simulator. We have presented two extensions of the proposed communication model to synthesize communication at different granularity levels and to introduce communication interleaving support. We have described two optimization techniques to improve the initial schedule and task mapping. Finally, we have extended our work to handle given number of transient or intermittent faults during task execution and data transmission on communication links. It is done by allocating slack time and scheduling the application with modified shifting-based scheduling algorithm.

To summarize, the main contributions of the thesis are:

- a formal graph-based method for communication modelling at system-level;
- contention-aware scheduling of real-time NoC-based systems;
- possibility to model communication at different granularity (message, packet) levels;
- modelling and scheduling communication with interleaving support;
- design optimization by branch-and-bound and simulated annealing global optimization techniques;

- modified shifting-based scheduling technique to increase dependability of application schedules taking into account also communication aspects and reliability;
- system-level design tool that is interfaced with several NoC simulators.

Future work

The future work includes several directions. One of the directions is dependability of NoC-based SoCs. We could duplicate application tasks and run them in parallel on multiple nodes to decrease schedule length and to increase fault-tolerance of the systems. At the same time the communication modelling plays an important role. Efficient heuristics are needed to duplicate and map tasks to NoC architecture. More practical work includes implementing and performing experiments with different topologies and deterministic routing algorithms. XY deterministic routing algorithm, used in this thesis, is simple to implement but does not distribute the network load evenly.

References

ISO/IEC 14882:2003 standard on the programming language C++. (2003).

IEEE Standard 1076-2008 on VHDL language. (2008).

Arteris. (2009). Retrieved from http://www.arteris.com/

International Technology Roadmap for Semiconductors. (2009). Retrieved from http://www.itrs.net

Silistix. (2009). Retrieved from http://www.silistix.com/

Sonics. (2009). Retrieved from http://www.sonicsinc.com/

STMicroelectronics. (2009). Retrieved from http://www.st.com

SystemC. (2009). Retrieved from http://www.systemc.org

ATLAS. (2011). Retrieved from https://corfu.pucrs.br/redmine/projects/atlas/wiki

Nework simulator NS-2. (2011). Retrieved from http://isi.edu/nsnam/ns/

NIRGAM. (2011). Retrieved from http://nirgam.ecs.soton.ac.uk/

OMNeT++. (2011). Retrieved from http://www.omnetpp.org/

OPNET. (2011). Retrieved from http://www.opnet.com/solutions/network_rd/modeler.html

Ababei, C., & Katti, R. (2009). Achieving network on chip fault tolerance by adaptive remapping. *Proceedings of the 23rd IEEE international symposium on Parallel and Distributed Processing*, (pp. 1-4). Rome, Italy.

Adriahantenaina, A., Charlery, H., Greiner, A., Mortiez, L., & Zeferino, C. (2003). SPIN: a scalable, packet switched, on-chip micro-network. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, (pp. 70-73). Munich, Germany.

Agarwal, V., Hrishikesh, M., Keckler, S., & Burger, D. (2000). Clock rate versus IPC: the end of the road for conventional microarchitectures. *Proceedings of the 27th international symposium on Computer Architecture*, (pp. 248-259). Vancouver, BC, Canada.

Allan, A., Edenfeld, D., Joyner, J. W., Kahng, A. B., Rodgers, M., & Zorian, Y. (2002, January). 2001 technology roadmap for semiconductors. *IEEE Computer, 35*(1), 42-53.

Ashenden, P., & Wilsey, P. (1998). Considerations on system-level behavioural and structural modeling extensions to VHDL. *Proceedings of the International Verilog HDL Conference and VHDL International Users Forum*, (pp. 42-50). Santa Clara, CA, USA.

Banerjee, K., Souri, S., Kapur, P., & Saraswat, K. (2001). 3-D ICs: a novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. *Proceedings of the IEEE, 89 (5)*, pp. 602-633.

Benini, L., & De Micheli, G. (2002, Jan.). Networks on chips: a new SoC paradigm. *IEEE Computer, 35*(1), 70-78.

Bertozzi, D., & Benini, L. (2004). Xpipes: a network-on-chip architecture for gigascale systems-on-chip. *IEEE Circuits and Systems Magazine, 4*(2), 18-31.

Bienia, C., Kumar, S., Singh, J., & Li, K. (2008). The PARSEC benchmark suite: characterization and architectural implications. *Proceedings of the 17th international conference on Parallel Architectures and Compilation Techniques*, (pp. 72-81). Toronto, Ontario, Canada.

Bjerregaard, T., & Mahadevan, S. (2006). A survey of research and practices of network-on-chip. *ACM Computing Surveys, 38*(1).

Bjerregaard, T., & Sparso, J. (2005). A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. *Proceedings of the Design, Automation, and Test in Europe, 2*, pp. 1226-1231. Munich, Germany.

Budkowski, S., & Dembinski, P. (1987). An introduction to estelle: a specification language for distributed systems. *Computer Networks and ISDN Systems, 14*(1), 3-23.

Chen, Y.-K., & Kung, S. (2008). Trend and challenge on system-on-a-chip designs. *Journal of Signal Processing Systems, 53*(1-2), 217-229.

Cho, M., Lis, M., Shim, K., Kinsy, M., & Devadas, S. (2009). Path-based, randomized, oblivious, minimal routing. *Proceedings of the 2nd International Workshop on Network on Chip Architectures*, (pp. 23-28). New York, NY, USA.

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

Constantinescu, C. (2003). Trends and challenges in VLSI circuit reliability. *IEEE Micro, 23*(4), pp. 14-19.

Dally, W. J., & Towles, B. (2001). Route packets, not wires: on-chip inteconnection networks. *Proceedings of the Design Automation Conference*, (pp. 684-689). Las Vegas, NV, USA.

Dally, W. J., & Towles, B. (2004). *Principles and practices of interconnection networks*. Morgan Kaufman Publishers.

Dumitras, T., & Marculescu, R. (2003). On-chip stochastic communication. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, (pp. 790-795). Munich, Germany.

Ejlali, A., Al-Hashimi, B., Rosinger, P., & Miremadi, S. (2007). Joint consideration of fault-tolerance, energy-efficiency and performance in on-chip networks. *Proceedings of the Design, Automation and Test in Europe Conference and Exposition*, (pp. 1-6). Nice, France.

El-Rewini, H., Ali, H., & Lewis, T. (1995). Task scheduling in multiprocessing systems. *Computer, 28*(12), 27-37.

Færgemand, O., & Olsen, A. (1994). Introduction to SDL-92. *Computer Networks and ISDN Systems, 26*, 1143-1167.

Feero, B., & Pande, P. (2007). Performance evaluation for three-dimensional networks-on-chip. *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, (pp. 305-310). Porto Alegre, Brazil.

Felicijan, T., Bainbridge, J., & Furber, S. (2003). An asynchronous low latency arbiter for Quality of Service (QoS) applications. *Proceedings of the 15th international conference on Microelectronics*, (pp. 123-126).

Frazzetta, D., Dimartino, G., Palesi, M., Kumar, S., & Catania, V. (2008). Efficient application specific routing algorithms for NoC systems utilizing partially faulty links. *Proceedings of the 11th Euromicro conference on Digital System Design Architectures, Methods and Tools*, (pp. 18-25). Parma, Italy.

Garey, M., & Johson, D. (1979). *Computers and intractability: a guide to the theory of NP-completeness.* New York: W.H.Freeman Publishers.

Gerstlauer, A., Haubelt, C., Pimentel, A., Stefanov, T., Gajski, D., & Teich, J. (2009). Electronic system-level synthesis methodologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28*(10), 1517-1530.

Glass, C., & Ni, L. (1992). The turn model for adaptive routing. *Proceedings of the 19th annual international symposium on Computer Architecture*, (pp. 278-287). Queensland, Australia.

Goossens, K., Dielissen, J., & Radulescu, A. (2005). Æthereal network on chip: concepts, architectures, and implementations. *IEEE Design and Test of Computers, 22*(5), 414-421.

Gratz, P., & Keckler, S. (2010). Realistic workload characterization and analysis for networks-on-chip design. *Proceedings of the 4th workshop on Chip Multiprocessor Memory Systems and Interconnects.*

Grecu, C., Anghel, L., Pande, P., Ivanov, A., & Saleh, R. (2007). Essential fault-tolerance metrics for NoC infrastructures. *Proceedings of the 13th IEEE international On-Line Testing Symposium*, (pp. 37-42). Heraklion, Crete, Greece.

Grecu, C., Ivanov, A., Pande, R., Jantsch, A., Salminen, E., Ogras, U., et al. (2007). Towards open network-on-chip benchmarks. *Proceedings of the 1st international symposium on Networks-on-Chip*, (pp. 205-205). Princeton, New Jersey, USA.

Guerrier, P., & Greiner, A. (2000). A generic architecture for on-chip packet-switched interconnections. *Proceedings of the Design, Automation, and Test in Europe*, (pp. 250-256). Paris, France.

Hamilton, S. (1999). Taking Moore's law into the next century. *IEEE Computer, 32*(1), 43-48.

Harel, D. (1987). Statecharts: a visual formalism for computer systems. *Science of Computer, 8*(3), 231-274.

Harsha, P., Hayes, T., Narayanan, H., Räcke, H., & Radhakrishnan, J. (2008). Minimizing average latency in oblivious routing. *Proceedings of the 19th annual ACM-SIAM symposium on Discrete Algorithms*, (pp. 200-207). Philadelphia, PA, USA.

Haurylau, M., Chen, G., Chen, H., Zhang, J., Nelson, N., Albonesi, D., et al. (2006). On-chip optical interconnect roadmap: challenges and critical directions. *IEEE Journal of Selected Topics in Quantum Electronics, 12*(6), 1699-1705.

Hemani, A., Jantsch, A., Kumar, S., Postula, A., Öberg, J., Millberg, M., et al. (2000). Network on chip: an architecture for billion transistor era. *Norchip.* Turku, Finland.

Henkel, J., Wolf, W., & Chakradhar, S. (2004). On-chip networks: a scalable, communication-centric embedded system design paradigm. *Proceedings of the 17th international conference on VLSI Design*, (pp. 845-851). Princeton, NJ, USA.

Ho, R., Mai, K., & Horowitz, M. (2001). The future of wires. *Proceedings of the IEEE, 89 (4)*, pp. 490-504.

Hoare, C. A. (1978). Communicating sequential processes. *Communications of the ACM, 21*(11), 934-941.

Hu, J., & Marculescu, R. (2003). Energy-aware mapping for tile-based NoC architectures under performance constraints. *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, (pp. 233-239). Kitakyushu, Japan.

Hu, J., & Marculescu, R. (2005). Communication and task scheduling of application-specific networks-on-chip. *Computers and Digital Techniques, 152*(5), 643-651.

Huang, T.-C., Ogras, U., & Marculescu, R. (2007). Virtual channels planning for networks-on-chip. *Proceedings of the 8th international symposium on Quality Electronic Design*, (pp. 879-884). San Jose, CA, USA.

Hwang, J., Chow, Y., Anger, F., & Le, C. (1989). Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal of Computing, 18*(2), 244-257.

Izosimov, V. (2006). Scheduling and optimization of fault-tolerant distributed embedded systems. Sweden: Linköping University, Doctoral dissertation.

Iyer, A., & Marculescu, D. (2002). Power and performance evaluation of globally asynchronous locally synchronous processors. *Proceedings of the 29th annual international symposium on Computer Architecture*, (pp. 158-168). Anchorage, Alaska, USA.

IEEE standard classification for software anomalies. (Jan. 7 2010). 1-15. IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993).

Jantsch, A. (2003). *Modeling embedded systems and SoCs - concurrency and time in models of computation.* Morgan Kaufmann.

Jantsch, A., & Tenhunen, H. (2003). In *Networks on chip* (pp. 9-15). Kluwer Academic Publishers.

Kahng, A. (2007). Key directions and a roadmap for electrical design for manufacturability. *Proceedings of the 37th European Solid State Device Research Conference*, (pp. 83-88). Washington, DC, USA.

Kariniemi, K., & Nurmi, J. (2005). Fault tolerant XGFT network on chip for multi processor system on chip circuits. *Proceedings of the international conference on Field Programmable Logic and Applications*, (pp. 203-210). Tampere, Finland.

Kermani, P., & Kleinrock, L. (1979). Virtual cut-through: a new computer communication switching technique. *Computer Networks, 3*, 267-286.

Keutzer, K., Newton, A., Rabaey, J., & Sangiovanni-Vincentelli, A. (2000). System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19*(12), pp. 1523-1543.

Kiang, D. (1997). Technology impact on dependability requirements. *Proceedings of the 3rd IEEE international Software Engineering Standards Symposium and Forum*, (pp. 92-98).

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Konstadinidis, G. (2009). Challenges in microprocessor physical and power management design. *Proceedings of the international symposium on VLSI Design, Automation and Test*, (pp. 9-12). Hsinchu, Taiwan.

Koopman, P., & Chakravarty, T. (2004). Cyclic redundancy code (CRC) polynomial selection for embedded networks. *Proceedings of the 2004 international conference on Dependable Systems and Networks*, (pp. 145-154). Florence, Italy.

Koren, I., & Krishna, C. (2007). *Fault-tolerant systems.* Morgan Kaufmann.

Kumar, S., Jantsch, A., Millberg, M., Öberg, J., Soininen, J. P., Forsell, M., et al. (2002). A network on chip architecture and design methodology. *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, (pp. 105-112). Pittsburgh, PA, USA.

Kwok, Y.-K., & Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys, 31*(4), 406-471.

Lagnese, E., & Thomas, D. (1989). Architectural partitioning for system level design. *Proceedings of the 26th Conference on Design Automation*, (pp. 62-67).

Laprie, J.-C. (1985). Dependable computing and fault tolerance: concepts and terminology. *Proceedings of the 15th international symposium on Fault-Tolerant Computing*, (pp. 2-11).

Ledesma, S., Avina, G., & Sanchez, R. (2008). Practical considerations for simulated annealing implementation. In C. Tan, *Simulated Annealing.* InTech.

Lehtonen, T., Liljeberg, P., & Plosila, J. (2009). Fault tolerant distributed routing algorithms for mesh networks-on-chip. *Proceedings of the international symposium on Signals, Circuits and Systems*, (pp. 1-4). Iasi, Romania.

Lei, T., & Kumar, S. (2003). A two-step genetic algorithm for mapping task graphs to a network on chip architecture. *Proceedings of the Euromicro Symposium on Digital System Design*, (pp. 180-187). Belek-Antalya, Turkey.

Leiserson, C. E. (1985). Fat-trees: universal networks for hardware efficient supercomputing. *IEEE transactions on Computers, C-34*(10), 892-901.

Liu, W., Xu, J., Wu, X., Ye, Y., Wang, X., Zhang, W., et al. (2011). A NoC traffic suite based on real applications. *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, (pp. 66-71). Chennai, India.

Lu, Z. (2007). Design and analysis of on-chip communication for network-on-chip platforms. Stockholm, Sweden: KTH, Doctoral dissertation.

Lu, Z., Xia, L., & Jantsch, A. (2008). Cluster-based simulated annealing for mapping cores onto 2D mesh networks on chip. *Proceedings of the 11th IEEE workshop on Design and Diagnostics of Electronic Circuits and Systems*, (pp. 1-6). Bratislava, Slovakia.

Manolache, S., Eles, P., & Peng, Z. (2007). Fault-aware communication mapping for NoCs with guaranteed latency. *International Journal of Parallel Programming, 35*(2), 125-156.

Marcon, C., Kreutz, M., Susin, A., & Calazans, N. (2005). Models for embedded application mapping onto NoCs: timing analysis. *Proceedings of the 16th IEEE international workshop on Rapid System Prototyping*, (pp. 17-23). Montreal, Canada.

Marculescu, R., Ogras, U., Li-Shiuan Peh Jerger, N., & Hoskote, Y. (2009). Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28*(1), 3-21.

Metropolis, N., Rosenbluth, A., & Rosenbluth, M. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics, 21*(6), 1087–1092.

Millberg, M., Nilsson, E., Thid, R., & Jantsch, A. (2004). Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, (pp. 890-895). Paris, France.

Miremadi, G., & Torin, J. (1995). Evaluating processor- behaviour and three error-detection mechanisms using physical fault-injection. *IEEE Transactions on Reliability, 44*(3), 441-454.

Moore, G. (1975). Progress in digital integrated electronics. *Proceedings of the International Electron Devices Meeting*, (pp. 11-13). Washington, D.C., USA.

Moraes, F., Calazans, N., Mello, A., Möller, L., & Ost, L. (2004). HERMES: an infrastructure for low area overhead packet-switching networks on chip. *The VLSI Integration, 38*(1), 69-93.

Murali, S., Atienza, D., Benini, L., & De Micheli, G. (2006). A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. *Proceedings of the 43rd annual Design Automation Conference*, (pp. 845-848).

Murali, S., Seiculescu, C., Benini, L., & De Micheli, G. (2009). Synthesis of networks on chips for 3D systems on chips. *Proceedings of the 14th Asia South Pacific Design Automation Conference*, (pp. 242-247). Yokohama, Japan.

Murali, S., Theocharides, T., Vijaykrishnan, N., Irwin, M., Benini, L., & De Micheli, G. (2005). Analysis of error recovery schemes for networks on chips. *IEEE Design & Test of Computers, 22*(5), 434 - 442.

Ni, L., & McKinley, P. (1993). A survey of wormhole routing techniques in direct networks. *IEEE Computer, 26*(2), 62-76.

Nikolic, B., Park, J.-H., Kwak, J., & Giraud, B. (2011). Technology variability from a design perspective. *IEEE Transactions on Circuits and Systems, 58*(9), 1996-2009.

Orsila, H., Salminen, E., Hannikainen, M., & Hamalainen, T. (2007). Optimal subset mapping and convergence evaluation of mapping algorithms for distributing task graphs on multiprocessor SoC. *Proceedings of the 2007 international Symposium on System-on-Chip*, (pp. 1-6). Tampere, Finland.

Owens, J., Dally, W., Ho, R., Jayasimha, D., Keckler, S., & Peh, L.-S. (2007). Research challenges for on-chip interconnection networks. *IEEE Micro, 27*(5), 96-108.

Palesi, M., Patti, D., & Fazzino, F. (2011). *Noxim: the NoC simulator*. Retrieved from http://sourceforge.net/projects/noxim/

Pan, S.-J., & Cheng, K.-T. (2007). A framework for system reliability analysis considering both system error tolerance and component test quality. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, (pp. 1-6). Nice, France.

Pande, P., Ganguly, A., Feero, B., Belzer, B., & Grecu, C. (2006). Design of low power & reliable networks on chip through joint crosstalk avoidance and forward error correction coding. *Proceedings of the 21st IEEE international symposium on Defect and Fault Tolerance in VLSI Systems*, (pp. 466-476). Arlington, Virginia, USA.

Pande, P., Grecu, C., Ivanov, A., & Saleh, R. (2003). Design of a switch for network on chip applications. *Proceedings of the international symposium on Circuits and Systems, 5*, pp. 217-220. Bangkok, Thailand.

Pavlidis, V., & Friedman, E. (2007). 3-D topologies for networks-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 15*(10), 1081-1090.

Peterson, W., & Brown, D. (1961). Cyclic codes for error detection. *Proceedings of the Institute of Radio Engineers, 49*(1), 228-235.

Pirretti, M., Link, G., Brooks, R., Vijaykrishnan, N., Kandemir, M., & Irwin, M. (2004). Fault tolerant algorithms for network-on-chip interconnect. *Proceedings of the IEEE Computer Society Annual Symposium on VLSI: Emerging Trends in VLSI Systems Design*, (pp. 46-51). Lafayette, LA, USA.

Radulescu, A., & Goossens, K. (2002). Communication services for networks on chip. *Proceedings of the SAMOS conference*, (pp. 275-299).

Raghunathan, V., Srivastava, M., & Gupta, R. (2003). A survey of techniques for energy efficient on-chip communication. *Proceedings of the 40th Design Automation Conference*, (pp. 900-905). Anaheim, CA, USA.

Rahman, M., & Chowdhury, M. (2009). Examining branch and bound strategy on multiprocessor task scheduling. *Proceedings of the international conference on Computer and Information Technology*, (pp. 162–167). Dhaka, Bangladesh.

Rantala, P., Isoaho, J., & Tenhunen, H. (2007). Novel agent-based management for fault-tolerance in network-on-chip. *Proceedings of the 10th*

Euromicro Conference on Digital System Design Architectures, Methods and Tools, (pp. 551-555). Lübeck, Germany.

Rijpkema, E., Goossens, K., & Wielage, P. (2001). A router architecture for networks on silicon. *Proceedings of the 2nd Workshop on Embedded Systems.*

Rusu, C., Grecu, C., & Anghel, L. (2008). Communication aware recovery configurations for networks-on-chip. *Proceedings of the 14th IEEE international On-Line Testing Symposium*, (pp. 201-206). Rhodes, Greece.

Sahoo, S., Datta, M., & Kar, R. (2011). An efficient dynamic power estimation method for on-chip VLSI interconnects. *Proceedings of the 2nd international conference on Emerging Applications of Information Technology*, (pp. 379-382). Kolkata, West Bengal, India.

Salminen, E., Kulmala, A., & Hämäläinen, T. D. (2008). *Survey of network-on-chip proposals*. Retrieved from http://www.ocpip.org/uploads/documents/OCP-IP_Survey_of_NoC_Proposals_White_Paper_April_2008.pdf

Samman, F. A. (2010). Microarchitecture and implementation of networks-on-chip with a flexible concept for communication media sharing. Darmstadt, Germany: Technische Universität Darmstad, Doctoral dissertation.

Samman, F. A., Hollstein, T., & Glesner, M. (2011). New theory for deadlock-free multicast routing in wormhole-switched virtual-channelless nNetworks-on-chip. *IEEE Transactions on Parallel and Distributed Systems, 22*(4), 544-557.

Samman, F., Hollstein, T., & Glesner, M. (2008). Multicast parallel pipeline router architecture for network-on-chip. *Proceedings of the Design, Automation and Test in Europe*, (pp. 1396-1401). Munich, Germany.

Sgroi, M., Sheets, M., Mihal, A., Keutzer, K., Malik, S., Rabaey, J., et al. (2001). Addressing the system-on-a-chip interconnect woes through communication-based design. *Proceedings of the 38th Design Automation Conference*, (pp. 667-672). Las Vegas, NV, USA.

Shanbhag, N., Soumyanath, K., & Martin, S. (2000). Reliable low-power design in the presence of deep submicron noise. *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, (pp. 295-302). Rapallo, Italy.

Shim, Z., & Burns, A. (2008). Real-time communication analysis for on-chip networks with wormhole switching networks-on-chip. *Proceedings of the 2nd IEEE international symposium on Networks-on-Chip*, (pp. 161 - 170). Newcastle University, UK.

Shin, D., & Kim, J. (2004). Power-aware communication optimization for networks-on-chips with voltage scalable links. *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, (pp. 170-175). Stockholm, Sweden.

Shin, D., & Kim, J. (2008). Communication power optimization for network-on-chip architectures. *Journal of Low Power Electronics, 2*(2), 165-176.

Shrivastav, A., Tomar, G., & Singh, A. (2011). Performance comparison of AMBA bus-based system-on-chip communication protocol. *Proceedings of the international conference on Communication Systems and Network Technologies*, (pp. 449-454). Katra, India.

Sigüenza-Tortosa, D., & Nurmi, J. (2002). Proteo: a new approach to network-on-chip. *Proceedings of the IASTED international conference of Communication Systems and Networks.* Malaga, Spain.

Sih, G., & Lee, E. (1993). A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems, 4*(2), 175-187.

Sinnen, O., & Sousa, L. (2005). Communication contention in task scheduling. *IEEE Transactions on Parallel and Distributed Systems, 16*(6), 503-515.

Smith, D., DeLong, T., Johnson, B., & Giras, T. (2000). Determining the expected time to unsafe failure. *Proceedings of the 5th IEEE international symposim on High Assurance Systems Engineering*, (pp. 17-24).

Sosnowski, J. (1994). Transient fault tolerance in digital systems. *IEEE Micro, 14*(1), 24-35.

Stankovic, J., Spuri, M., Di Natale, M., & Buttazzo, G. (1995). Implications of classical scheduling results for real-time systems. *Computer, 28*(6), 16-25.

Stressing, J. (1989). System-level design tools. *Computer-Aided Engineering Journal*, 44-48.

Stuijk, S., Basten, T., Geilen, M., & Ghamarian, A. (2006). Resource-efficient routing and scheduling of time-constrained streaming communication on networks-on-chip. *Proceedings of the 9th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, (pp. 45-52). Dubrovnik, Croatia.

Zhang, L., Han, Y., Xu, Q., Li, X. w., & Li, H. (2009). On topology reconfiguration for defect-tolerant NoC-based homogeneous manycore systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 17*(9), pp. 1173-1186.

Tagel, M., Ellervee, P., & Jervan, G. (2009). Scheduling framework for real-time dependable NoC-based systems. *Proceedings of the International Symposium on System-on-Chip*, (pp. 95-99). Tampere, Finland.

Tagel, M., Ellervee, P., & Jervan, G. (2010a). Design space exploration and optimisation for NoC-based timing sensitive systems. *Proceedings of the 12th Biennial Baltic Electronic Conference*, (pp. 177-180). Tallinn, Estonia.

Tagel, M., Ellervee, P., & Jervan, G. (2010b). System-level communication synthesis and dependability improvements for network-on-chip based systems. *Estonian Journal of Engineering, 16*(1), 23-38.

Tagel, M., Ellervee, P., & Jervan, G. (2011). System-level design of NoC-based dependable embedded systems. In R. Ubar, J. Raik, & H. T. Vierhaus, *Fault-tolerance and applications in system-on-chip design:*

advancements and techniques (pp. 1-36). Hershey, Pennsylvania, USA: IGI Global.

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2011a). Communication modelling and synthesis for NoC-based systems with real-time constraints. *Proceedings of the 14th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, (pp. 237-242). Cottbus, Germany.

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2011b). Contention aware scheduling for NoC-based real-time systems. *Proceedings of the 29th Norchip conference*, (pp. 1-4). Lund, Sweden.

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2011c). System-level optimization of NoC-based timing sensitive systems. *Estonian Journal of Engineering, 17*(2), 158-168.

Tagel, M., Ellervee, P., Hollstein, T., & Jervan, G. (2012). Contention-aware scheduling for NoC based systems. *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)*, [submitted for review].

Talbi, E.-G., & Muntean, T. (1993). Hill-climbing, simulated annealing and genetic algorithms: a comparative study and application to the mapping problem. *Proceeding of the 26th Hawaii international conference on System Sciences*, (pp. 565-573). Maui, Hawaii USA.

Valiant, L., & Brebner, G. (1981). Universal schemes for parallel communication. *Proceedings of the 13th annual ACM symposium on Theory of Computing*, (pp. 263-277). Milwaukee, Wisconsin, USA.

Valtonen, T., Nurmi, T., Isoaho, J., & Tenhunen, H. (2001). An autonomous error-tolerant cell for scalable network-on-chip architectures. *Proceedings of the 19th IEEE NorChip Conference*, (pp. 198-203). Stockholm, Sweden.

Vangal, S., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., et al. (2008). An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits, 43*(1), 29–41.

Wattanapongsakorn, N., & Levitan, S. (2000). Integrating dependability analysis into the real-time system design process. *Proceedings of the Annual Reliability and Maintainability Symposium*, (pp. 327-334).

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., et al. (2008). The worst-case execution time problem — overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems, 7*(3), 36:1-36:53.

Woo, S., Ohara, M., Torrie, E., Singh, J., & Gupta, A. (1995). The SPLASH-2 programs: characterization and methodological considerations. *In Proceedings of the 22nd international symposium on Computer Architecture*, (pp. 24-36). Santa Margherita Ligure, Italy.

Curriculum Vitae

Personal data

Name: Mihkel Tagel

Date and place of birth: 22.12.1981, Estonia

Nationality**:** Estonian

Contact information

Address: Raja 15, 12618 Tallinn

Telephone: +372 620 22 51

E-mail address: mihkel.tagel@ati.ttu.ee

Education

2006 –	Ph.D. student in Department of Computer Engineering, Tallinn University of Technology (TUT)
2004 – 2006	M.Sc. in Computer Engineering, TUT
2000 – 2004	Diploma in Computer Engineering, TUT

Career

2007 –	Tallinn University of Technology, Department of Computer Engineering, Researcher
2003 –	TD Baltic AS, IT manager

Scientific activities

2007 – Member of IEEE Computer Society

Defended theses

Mihkel Tagel (2006). Deterministic Traffic Generator for Network-on-Chip Simulator. Master of Science in Computer Engineering, Tallinn University of Technology, Department of Computer Engineering. Supervisors: Gert Jervan, Peeter Ellervee.

Main areas of scientific work/current research topics

Testability and reliability of Network-on-Chips.

Other research projects

Design of Reliable Embedded Systems

Hardware functional verification and debug

Awards

2009 EITF “Tiger University” scholarship for ICT PhD students

Elulookirjeldus

Isikuandmed

Ees- ja perekonnanimi: Mihkel Tagel

Sünniaeg ja -koht: 22.12.1981, Eesti

Kodakondsus**:** eestlane

Kontaktandmed

Aadress: Raja 15, 12618 Tallinn

Telefon: +372 620 22 51

E-posti aadress: mihkel.tagel@ati.ttu.ee

Hariduskäik

2006 –	doktorant, Arvutitehnika instituut, Tallinna Tehnikaülikool (TTÜ)
2004 – 2006	tehnikateaduste magister, arvuti- ja süsteemitehnika eriala, TTÜ
2000 – 2004	diplom, arvutisüsteemide eriala, TTÜ

Teenistuskäik

2007 –	Tallinna Tehnikaülikool, Arvutitehnika instituut, erakorraline teadur
2003 –	TD Baltic AS, IT juht

Teadustegevus

2007 –	IEEE Computer Society. Liige

Kaitstud lõputööd

Mihkel Tagel (2006). Deterministlik võrguliikluse generaator kiipvõrgu simulaatorile. Magistrikraad. Tallinna Tehnikaülikool, Arvutitehnika instituut. Juhendajad: Gert Jervan, Peeter Ellervee

Teadustöö põhisuunad

Kiipvõrgul põhinevate kiipsüsteemide veakindluse tõstmine, süsteemitaseme disain.

Teised uurimisprojektid

Töökindlate sardsüsteemide disain

Riistvara funktsionaalne verifitseerimine ja silumine

Teaduspreemiad

2009 EITSA Tiigriülikooli stipendium IKT doktorantidele Eesti avalik-õiguslikes ülikoolides

DISSERTATIONS DEFENDED AT TALLINN UNIVERSITY OF TECHNOLOGY ON *INFORMATICS AND SYSTEM ENGINEERING*

1. **Lea Elmik**. Informational Modelling of a Communication Office. 1992.

2. **Kalle Tammemäe**. Control Intensive Digital System Synthesis. 1997.

3. **Eerik Lossmann**. Complex Signal Classification Algorithms, Based on the Third-Order Statistical Models. 1999.

4. **Kaido Kikkas**. Using the Internet in Rehabilitation of People with Mobility Impairments – Case Studies and Views from Estonia. 1999.

5. **Nazmun Nahar**. Global Electronic Commerce Process: Business-to-Business. 1999.

6. **Jevgeni Riipulk**. Microwave Radiometry for Medical Applications. 2000.

7. **Alar Kuusik**. Compact Smart Home Systems: Design and Verification of Cost Effective Hardware Solutions. 2001.

8. **Jaan Raik**. Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams. 2001.

9. **Andri Riid**. Transparent Fuzzy Systems: Model and Control. 2002.

10. **Marina Brik**. Investigation and Development of Test Generation Methods for Control Part of Digital Systems. 2002.

11. **Raul Land**. Synchronous Approximation and Processing of Sampled Data Signals. 2002.

12. **Ants Ronk**. An Extended Block-Adaptive Fourier Analyser for Analysis and Reproduction of Periodic Components of Band-Limited Discrete-Time Signals. 2002.

13. **Toivo Paavle**. System Level Modeling of the Phase Locked Loops: Behavioral Analysis and Parameterization. 2003.

14. **Irina Astrova**. On Integration of Object-Oriented Applications with Relational Databases. 2003.

15. **Kuldar Taveter**. A Multi-Perspective Methodology for Agent-Oriented Business Modelling and Simulation. 2004.

16. **Taivo Kangilaski**. Eesti Energia käiduhaldussüsteem. 2004.

17. **Artur Jutman**. Selected Issues of Modeling, Verification and Testing of Digital Systems. 2004.

18. **Ander Tenno**. Simulation and Estimation of Electro-Chemical Processes in Maintenance-Free Batteries with Fixed Electrolyte. 2004.

19. **Oleg Korolkov**. Formation of Diffusion Welded Al Contacts to Semiconductor Silicon. 2004.

20. **Risto Vaarandi**. Tools and Techniques for Event Log Analysis. 2005.

21. **Marko Koort**. Transmitter Power Control in Wireless Communication Systems. 2005.

22. **Raul Savimaa**. Modelling Emergent Behaviour of Organizations. Time-Aware, UML and Agent Based Approach. 2005.

23. **Raido Kurel**. Investigation of Electrical Characteristics of SiC Based Complementary JBS Structures. 2005.

24. **Rainer Taniloo**. Ökonoomsete negatiivse diferentsiaaltakistusega astmete ja elementide disainimine ja optimeerimine. 2005.

25. **Pauli Lallo.** Adaptive Secure Data Transmission Method for OSI Level I. 2005.

26. **Deniss Kumlander**. Some Practical Algorithms to Solve the Maximum Clique Problem. 2005.

27. **Tarmo Veskioja**. Stable Marriage Problem and College Admission. 2005.

28. **Elena Fomina**. Low Power Finite State Machine Synthesis. 2005.

29. **Eero Ivask**. Digital Test in WEB-Based Environment 2006.

30. **Виктор Войтович**. Разработка технологий выращивания из жидкой фазы эпитаксиальных структур арсенида галлия с высоковольтным p-n переходом и изготовления диодов на их основе. 2006.

31. **Tanel Alumäe**. Methods for Estonian Large Vocabulary Speech Recognition. 2006.

32. **Erki Eessaar**. Relational and Object-Relational Database Management Systems as Platforms for Managing Softwareengineering Artefacts. 2006.

33. **Rauno Gordon**. Modelling of Cardiac Dynamics and Intracardiac Bio-impedance. 2007.

34. **Madis Listak**. A Task-Oriented Design of a Biologically Inspired Underwater Robot. 2007.

35. **Elmet Orasson**. Hybrid Built-in Self-Test. Methods and Tools for Analysis and Optimization of BIST. 2007.

36. **Eduard Petlenkov**. Neural Networks Based Identification and Control of Nonlinear Systems: ANARX Model Based Approach. 2007.

37. **Toomas Kirt**. Concept Formation in Exploratory Data Analysis: Case Studies of Linguistic and Banking Data. 2007.

38. **Juhan-Peep Ernits**. Two State Space Reduction Techniques for Explicit State Model Checking. 2007.

39. **Innar Liiv**. Pattern Discovery Using Seriation and Matrix Reordering: A Unified View, Extensions and an Application to Inventory Management. 2008.

40. **Andrei Pokatilov**. Development of National Standard for Voltage Unit Based on Solid-State References. 2008.

41. **Karin Lindroos**. Mapping Social Structures by Formal Non-Linear Information Processing Methods: Case Studies of Estonian Islands Environments. 2008.

42. **Maksim Jenihhin**. Simulation-Based Hardware Verification with High-Level Decision Diagrams. 2008.

43. **Ando Saabas**. Logics for Low-Level Code and Proof-Preserving Program Transformations. 2008.

44. **Ilja Tšahhirov**. Security Protocols Analysis in the Computational Model – Dependency Flow Graphs-Based Approach. 2008.

45. **Toomas Ruuben**. Wideband Digital Beamforming in Sonar Systems. 2009.

46. **Sergei Devadze**. Fault Simulation of Digital Systems. 2009.

47. **Andrei Krivošei**. Model Based Method for Adaptive Decomposition of the Thoracic Bio-Impedance Variations into Cardiac and Respiratory Components. 2009.

48. **Vineeth Govind**. DfT-Based External Test and Diagnosis of Mesh-like Networks on Chips. 2009.

49. **Andres Kull**. Model-Based Testing of Reactive Systems. 2009.

50. **Ants Torim**. Formal Concepts in the Theory of Monotone Systems. 2009.

51. **Erika Matsak**. Discovering Logical Constructs from Estonian Children Language. 2009.

52. **Paul Annus**. Multichannel Bioimpedance Spectroscopy: Instrumentation Methods and Design Principles. 2009.

53. **Maris Tõnso**. Computer Algebra Tools for Modelling, Analysis and Synthesis for Nonlinear Control Systems. 2010.

54. **Aivo Jürgenson**. Efficient Semantics of Parallel and Serial Models of Attack Trees. 2010.

55. **Erkki Joasoon**. The Tactile Feedback Device for Multi-Touch User Interfaces. 2010.

56. **Jürgo-Sören Preden**. Enhancing Situation – Awareness Cognition and Reasoning of Ad-Hoc Network Agents. 2010.

57. **Pavel Grigorenko**. Higher-Order Attribute Semantics of Flat Languages. 2010.

58. **Anna Rannaste**. Hierarcical Test Pattern Generation and Untestability Identification Techniques for Synchronous Sequential Circuits. 2010.

59. **Sergei Strik**. Battery Charging and Full-Featured Battery Charger Integrated Circuit for Portable Applications. 2011.

60. **Rain Ottis**. A Systematic Approach to Offensive Volunteer Cyber Militia. 2011.

61. **Natalja Sleptšuk**. Investigation of the Intermediate Layer in the Metal-Silicon Carbide Contact Obtained by Diffusion Welding. 2011.

62. **Martin Jaanus**. The Interactive Learning Environment for Mobile Laboratories. 2011.

63. **Argo Kasemaa**. Analog Front End Components for Bio-Impedance Measurement: Current Source Design and Implementation. 2011.

64. **Kenneth Geers**. Strategic Cyber Security: Evaluating Nation-State Cyber Attack Mitigation Strategies. 2011.

65. **Riina Maigre**. Composition of Web Services on Large Service Models. 2011.

66. **Helena Kruus**. Optimization of Built-in Self-Test in Digital Systems. 2011.

67. **Gunnar Piho**. Archetypes Based Techniques for Development of Domains, Requirements and Sofware. 2011.

68. **Juri Gavšin**. Intrinsic Robot Safety Through Reversibility of Actions. 2011.

69. **Dmitri Mihhailov**. Hardware Implementation of Recursive Sorting Algorithms Using Tree-like Structures and HFSM Models. 2012.

70. **Anton Tšertov**. System Modeling for Processor-Centric Test Automation. 2012.

71. **Sergei Kostin**. Self-Diagnosis in Digital Systems. 2012.