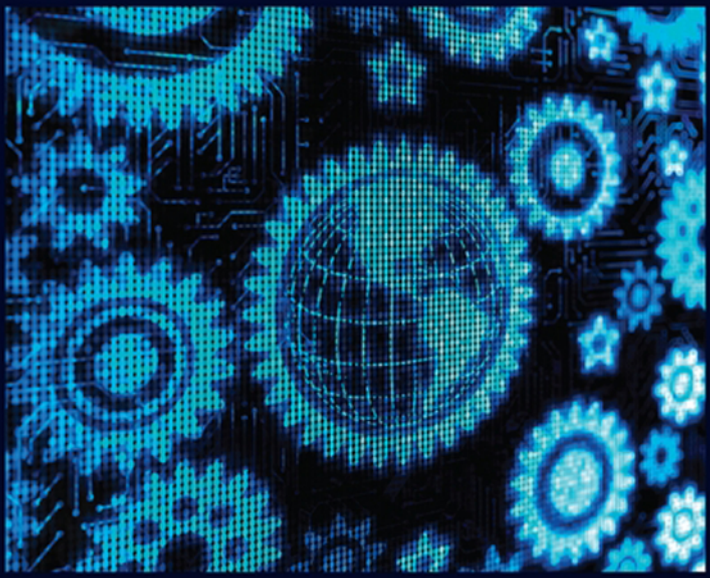
PREMIER REFERENCE SOURCE

Innovations in Embedded and Real-Time Systems Engineering for Communication



Seppo Virtanen

Innovations in Embedded and Real-Time Systems Engineering for Communication

Seppo Virtanen
University of Turku, Finland

Information Science
REFERENCE

Managing Director: Lindsay Johnston
Senior Editorial Director: Heather A. Probst
Book Production Manager: Sean Woznicki
Development Manager: Joel Gamon
Acquisitions Editor: Erika Gallagher
Typesetter: Deanna Jo Zombro
Cover Design: Nick Newcomer, Lisandro Gonzalez

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Copyright © 2012 by IGI Global. All rights reserved. No part of this publication may be reproduced, stored or distributed in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher. Product or company names used in this set are for identification purposes only. Inclusion of the names of the products or companies does not indicate a claim of ownership by IGI Global of the trademark or registered trademark.

Library of Congress Cataloging-in-Publication Data

Innovations in embedded and real-time systems engineering for communication / Seppo Virtanen, editor.
p. cm.
Includes bibliographical references and index.
Summary: "This book has collected the latest research within the field of real-time systems engineering, and will serve as a vital reference compendium for practitioners and academics"--Provided by publisher.
ISBN 978-1-4666-0912-9 (hardcover) -- ISBN 978-1-4666-0913-6 (ebook) -- ISBN 978-1-4666-0914-3 (print & perpetual access) 1. Embedded computer systems--Design and construction. 2. Real-time data processing. 3. Systems engineering--Technological innovations. I. Virtanen, Seppo.
TK7895.E42I532 2012
006.2'2--dc23
2012000026

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

The views expressed in this book are those of the authors, but not necessarily of the publisher.

Associate Editors

Sergey Balandin, *Nokia Research Center, Finland*
Jouni Isoaho, *University of Turku, Finland*
Gul Khan, *Ryerson University, Canada*
Dake Liu, *Linköping University, Sweden*
Jari Nurmi, *Tampere University of Technology, Finland*
Juha Plosila, *University of Turku, Finland*
Zili Shao, *Hong Kong Polytechnic University, Hong Kong*
Hannu Tenhunen, *Royal Institute of Technology (KTH), Sweden*
Andrey Terekhov, *Saint-Petersburg State University/Lanit-Tercom Inc., Russia*
Dragos Truscan, *Åbo Akademi University, Finland*

List of Reviewers

Tapani Ahonen, *Tampere University of Technology, Finland*
Jinian Bian, *Tsinghua University, China*
Michael Blumenstein, *Griffith University, Australia*
Dimitri Boulytchev, *Saint-Petersburg State University/Lanit-Tercom Inc., Russia*
Raja Datta, *Indian Institute of Technology (IIT) Kharagpur, India*
Peeter Ellervee, *Tallinn University of Technology, Estonia*
Abdeslam En-Nouaary, *Concordia University, Canada*
Fabrizio Ferrandi, *Politecnico di Milano, Italy*
Christos Grecos, *University of the West of Scotland, UK*
Martin Holzer, *Elektrobit, Austria*
Gert Jervan, *Tallinn University of Technology, Estonia*
Christian Poellabauer, *Notre Dame University, USA*
Paul Pop, *Technical University of Denmark, Denmark*
Martin Schoeberl, *Vienna University of Technology, Austria*
Tiberiu Seceleanu, *ABB Corporate Research, Sweden*
Tero Säntti, *University of Turku, Finland*
Tomi Westerlund, *University of Turku, Finland*
Colin Willcock, *Nokia-Siemens Networks, Germany*

Table of Contents

Preface xv

Chapter 1
Current Challenges in Embedded Communication Systems 1
Jouni Isoaho, University of Turku, Finland
Seppo Virtanen, University of Turku, Finland
Juha Plosila, University of Turku, Finland

Section 1
Technologies for Embedded Communication Systems

Chapter 2
Parallel Programming and Its Architectures Based on Data Access Separated Algorithm Kernels 23
Dake Liu, Linköping University, Sweden
Joar Sohl, Linköping University, Sweden
Jian Wang, Linköping University, Sweden

Chapter 3
Towards Sustainable Development of Energy-Aware Systems 43
Luigia Petre, Åbo Akademi University, Finland
Kaisa Sere, Åbo Akademi University, Finland

Chapter 4
A Multidimensional Software Cache for Scratchpad-Based Systems 59
Arnaldo Azevedo, Delft University of Technology, The Netherlands
Ben Juurlink, Technische Universität Berlin, Germany

Chapter 5
Automatic Generation of Memory Interfaces for ASIPs 79
David Kammler, RWTH Aachen University, Germany
Ernst Martin Witte, RWTH Aachen University, Germany
Anupam Chattopadhyay, RWTH Aachen University, Germany
Bastian Bauwens, RWTH Aachen University, Germany
Gerd Ascheid, RWTH Aachen University, Germany
Rainer Leupers, RWTH Aachen University, Germany
Heinrich Meyr, RWTH Aachen University, Germany

Section 2
Mobile Communication Applications

Chapter 6
System Architecture for 3GPP-LTE Modem using a Programmable Baseband Processor 102
Di Wu, Linköping University, Sweden
Johan Eilert, Linköping University, Sweden
Rizwan Asghar, Linköping University, Sweden
Dake Liu, Linköping University, Sweden
Anders Nilsson, Coresonic AB, Sweden
Eric Tell, Coresonic AB, Sweden
Eric Alfredsson, Coresonic AB, Sweden

Chapter 7
Joint Uplink and Downlink Performance Profiling of LTE Protocol Processing on a Mobile Platform..... 122
David Szczesny, Ruhr-Universität Bochum, Germany
Sebastian Hessel, Ruhr-Universität Bochum, Germany
Anas Showk, Ruhr-Universität Bochum, Germany
Attila Bilgic, Ruhr-Universität Bochum, Germany
Uwe Hildebrand, Comneon GmbH, Germany
Valerio Frascolla, Comneon GmbH, Germany

Chapter 8
Embedded Networks in Mobile Devices 141
Sergey Balandin, Nokia Research Center, Finland
Michel Gillet, Nokia Devices, Finland

Chapter 9
Implementation of FFT on General-Purpose Architectures for FPGA 156
Fabio Garzia, Tampere University of Technology, Finland
Roberto Airoldi, Tampere University of Technology, Finland
Jari Nurmi, Tampere University of Technology, Finland

Section 3
On-Chip Communication

Chapter 10
Performance Analysis of On-Chip Communication Structures under Device Variability..... 177
Faiz-ul Hassan, University of Glasgow, UK
Wim Vanderbauwhede, University of Glasgow, UK
Fernando Rodríguez-Salazar, University of Glasgow, UK

Chapter 11
Schedulability Analysis for Real Time On-Chip Communication with Wormhole Switching..... 198
Zheng Shi, University of York, UK
Alan Burns, University of York, UK
Leandro Soares Indrusiak, University of York, UK

Chapter 12
Modeling Communication in Multi–Processor Systems–on–Chip Using Modular Connectors..... 219
Leonidas Tsiopoulos, Åbo Akademi University, Finland
Kaisa Sere, Åbo Akademi University, Finland
Juha Plosila, University of Turku, Finland

Chapter 13
Generating Process Network Communication Infrastructure for Custom Multi-Core Platforms.....241
Peter Sørensen, Technical University of Denmark, Denmark
Jan Madsen, Technical University of Denmark, Denmark

Section 4
Formalisms and Methodologies for Embedded Communication Systems

Chapter 14
Joint Validation of Application Models and Multi-Abstraction Network-on-Chip Platforms..... 263
Sanna Määttä, Tampere University of Technology, Finland
Leandro Möller, Technische Universität Darmstadt, Germany
Leandro Soares Indrusiak, University of York, UK
Luciano Ost, Catholic University of Rio Grande do Sul, Brazil
Manfred Glesner, Technische Universität Darmstadt, Germany
Jari Nurmi, Tampere University of Technology, Finland
Fernando Moraes, Catholic University of Rio Grande do Sul, Brazil

Chapter 15
Hierarchical Agent Monitored Parallel On-Chip System:A Novel Design Paradigm and its Formal Specification 278
Liang Guang, University of Turku, Finland
Juha Plosila, University of Turku, Finland
Jouni Isoaho, University of Turku, Finland
Hannu Tenhunen, University of Turku, Finland

Chapter 16
Service-Oriented Development of Fault Tolerant Communicating Systems: Refinement Approach 297
Linas Laibinis, Åbo Akademi University, Finland
Elena Troubitsyna, Åbo Akademi University, Finland
Sari Leppänen, Nokia Research Center, Finland

Compilation of References 320

About the Contributors 341

Index 350

Detailed Table of Contents

Preface xv

Chapter 1
Current Challenges in Embedded Communication Systems 1
Jouni Isoaho, University of Turku, Finland
Seppo Virtanen, University of Turku, Finland
Juha Plosila, University of Turku, Finland

This article defines and analyses key challenges met in future embedded systems in networked multimedia and communication applications. Self-awareness, interoperability and embedded security are used to characterize different aspects of designing and implementing next generation embedded systems. The dynamic nature of applications and implementations as well as possible technological faults and variations need to be considered in system verification and modeling. A new design layer needs to be added to current NoC platforms in order to build procedures that take into account dynamic system reconfigurations, fault-tolerance aspects and flexible portability. Increased modularity and networked implementations create a need for trust management mechanisms between system components and technology for analyzing validity and correctness of received application and system configuration information.

Section 1
Technologies for Embedded Communication Systems

Chapter 2
Parallel Programming and Its Architectures Based on Data Access Separated Algorithm Kernels 23
Dake Liu, Linköping University, Sweden
Joar Sohl, Linköping University, Sweden
Jian Wang, Linköping University, Sweden

A novel master-multi-SIMD architecture and its kernel (template) based parallel programming flow is introduced as a parallel signal processing platform. The name of the platform is ePUMA (embedded Parallel DSP processor architecture with Unique Memory Access). The essential technology is to separate data accessing kernels from arithmetic computing kernels so that the run-time cost of data access can be minimized by running it in parallel with algorithm computing. The SIMD memory subsystem

architecture based on the proposed flow dramatically improves the total computing performance. The hardware system and programming flow introduced in this article will primarily aim at low-power high-performance embedded parallel computing with low silicon cost for communications and similar real-time signal processing.

Chapter 3
Towards Sustainable Development of Energy-Aware Systems 43
Luigia Petre, Åbo Akademi University, Finland
Kaisa Sere, Åbo Akademi University, Finland

Smart devices have pervaded our society and encouraged lifestyles that depend on them. One of the fundamental requirements for a successful dependency is that the general public be aware of the energy limitations of these devices and to stay in control of energy consumption. In this paper, the authors propose a formal specification method that takes energy into account. They propose two development approaches that can use these specifications to develop energy-aware systems in a sustainable manner.

Chapter 4
A Multidimensional Software Cache for Scratchpad-Based Systems 59
Arnaldo Azevedo, Delft University of Technology, The Netherlands
Ben Juurlink, Technische Universität Berlin, Germany

In many kernels of multimedia applications, the working set is predictable, making it possible to schedule the data transfers before the computation. Many other kernels, however, process data that is known just before it is needed or have working sets that do not fit in the scratchpad memory. Furthermore, multimedia kernels often access two or higher dimensional data structures and conventional software caches have difficulties to exploit the data locality exhibited by these kernels. For such kernels, the authors present a Multidimensional Software Cache (MDSC), which stores 1- 4 dimensional blocks to mimic in cache the organization of the data structure. Furthermore, it indexes the cache using the matrix indices rather than linear memory addresses. MDSC also makes use of the lower overhead of Direct Memory Access (DMA) list transfers and allows exploiting known data access patterns to reduce the number of accesses to the cache. The MDSC is evaluated using GLCM, providing an 8% performance improvement compared to the IBM software cache. For MC, several optimizations are presented that reduce the number of accesses to the MDSC.

Chapter 5
Automatic Generation of Memory Interfaces for ASIPs 79
David Kammler, RWTH Aachen University, Germany
Ernst Martin Witte, RWTH Aachen University, Germany
Anupam Chattopadhyay, RWTH Aachen University, Germany
Bastian Bauwens, RWTH Aachen University, Germany
Gerd Ascheid, RWTH Aachen University, Germany
Rainer Leupers, RWTH Aachen University, Germany
Heinrich Meyr, RWTH Aachen University, Germany

With the growing market for multi-processor system-on-chip (MPSoC) solutions, application-specific instruction-set processors (ASIPs) gain importance as they allow for a wide tradeoff between flexibility and efficiency in such a system. Their development is aided by architecture description languages (ADLs) supporting the automatic generation of architecture-specific tool sets as well as synthesizable register transfer level (RTL) implementations from a single architecture model. However, these generated implementations have to be manually adapted to the interfaces of dedicated memories or memory controllers, slowing down the design-space exploration regarding the memory architecture. To overcome this drawback, the authors extend RTL code generation from ADL models with the automatic generation of memory interfaces. This is accomplished by introducing a new abstract and versatile description format for memory interfaces and their timing protocols. The feasibility of this approach is demonstrated in real-life case studies, including a design space exploration for a banked memory system.

Section 2
Mobile Communication Applications

Chapter 6
System Architecture for 3GPP-LTE Modem using a Programmable Baseband Processor 102
Di Wu, Linköping University, Sweden
Johan Eilert, Linköping University, Sweden
Rizwan Asghar, Linköping University, Sweden
Dake Liu, Linköping University, Sweden
Anders Nilsson, Coresonic AB, Sweden
Eric Tell, Coresonic AB, Sweden
Eric Alfredsson, Coresonic AB, Sweden

The evolution of third generation mobile communications toward high-speed packet access and long-term evolution is ongoing and will substantially increase the throughput with higher spectral efficiency. This paper presents the system architecture of an LTE modem based on a programmable baseband processor. The architecture includes a baseband processor that handles processing time and frequency synchronization, IFFT/FFT (up to 2048-p), channel estimation and subcarrier de-mapping. The throughput and latency requirements of a Category four User Equipment (CAT4 UE) is met by adding a MIMO symbol detector and a parallel Turbo decoder supporting H-ARQ, which brings both low silicon cost and enough flexibility to support other wireless standards. The complexity demonstrated by the modem shows the practicality and advantage of using programmable baseband processors for a single-chip LTE solution.

Chapter 7
Joint Uplink and Downlink Performance Profiling of LTE Protocol Processing on a Mobile Platform..... 122
David Szczesny, Ruhr-Universität Bochum, Germany
Sebastian Hessel, Ruhr-Universität Bochum, Germany
Anas Showk, Ruhr-Universität Bochum, Germany
Attila Bilgic, Ruhr-Universität Bochum, Germany
Uwe Hildebrand, Comneon GmbH, Germany
Valerio Frascolla, Comneon GmbH, Germany

This article provides a detailed profiling of the layer 2 (L2) protocol processing for 3G successor Long Term Evolution (LTE). For this purpose, the most processing intensive part of the LTE L2 data plane is executed on top of a virtual ARM based mobile phone platform. The authors measure the execution times as well as the maximum data rates at different system setups. The profiling is done for uplink (UL) and downlink (DL) directions separately as well as in a joint UL and DL scenario. As a result, the authors identify time critical algorithms in the protocol stack and check to what extent state-of-the-art hardware platforms with a single-core processor and traditional hardware acceleration concepts are still applicable for protocol processing in LTE and beyond LTE mobile devices.

Chapter 8
Embedded Networks in Mobile Devices 141
Sergey Balandin, Nokia Research Center, Finland
Michel Gillet, Nokia Devices, Finland

The concept of a mobile phone has recently transformed into a new concept of mobile multimedia devices capable of performing multiple complex tasks and integrating multiple functionalities. It has resulted in a significant increase of device integration costs and complicated deployment of new technologies. Device integrator companies favor modularity everywhere possible, which results in a new trend toward networked architectures for the mobile devices. However, comparing to the best-known embedded network solutions, e.g., SoC and NoC, these architectures have unique constraints and requirements, which also are significantly different from the wide area networks. The main constraints are power consumption and having a modular architecture to allow reuse of the components. Transition to the new architectures for mobile devices is a time consuming task that requires the analysis of many solutions applied in other contexts, especially for embedded protocols, QoS and resource management. This article reviews the state of the art in embedded networks research and the key assumptions, restrictions and limitations faced by designers of embedded networks architectures for mobile devices.

Chapter 9
Implementation of FFT on General-Purpose Architectures for FPGA 156
Fabio Garzia, Tampere University of Technology, Finland
Roberto Airoldi, Tampere University of Technology, Finland
Jari Nurmi, Tampere University of Technology, Finland

This paper describes two general-purpose architectures targeted to Field Programmable Gate Array (FPGA) implementation. The first architecture is based on the coupling of a coarse-grain reconfigurable array with a general-purpose processor core. The second architecture is a homogeneous multi-processor system-on-chip (MP-SoC). Both architectures have been mapped onto two different Altera FPGA devices, a StratixII and a StratixIV. Although mapping onto the StratixIV results in higher operating frequencies, the capabilities of the device are not fully exploited. The implementation of a FFT on the two platforms shows a considerable speed-up in comparison with a single-processor reference architecture. The speed-up is higher in the reconfigurable solution but the MP-SoC provides an easier programming interface that is completely based on C language. The authors' approach proves that implementing a programmable architecture on FPGA and then programming it using a high-level software language is a viable alternative to designing a dedicated hardware block with a hardware description language (HDL) and mapping it on FPGA.

Section 3
On-Chip Communication

Chapter 10
Performance Analysis of On-Chip Communication Structures under Device Variability..... 177
Faiz-ul Hassan, University of Glasgow, UK
Wim Vanderbauwhede, University of Glasgow, UK
Fernando Rodríguez-Salazar, University of Glasgow, UK

On-chip communication is becoming an important bottleneck in the design and operation of high performance systems where it has to face additional challenges due to device variability. Communication structures such as tapered buffer drivers, interconnects, repeaters, and data storage elements are vulnerable to variability, which can limit the performance of the on-chip communication networks. In this regard, it becomes important to have a complete understanding of the impact that variability will have on the performance of these circuit elements in order to design high yield and reliable systems. In this paper, the authors have characterized the performance of the communication structures under the impact of random dopant fluctuation (RDF) for the future technology generations of 25, 18, and 13 nm. For accurate characterization of their performance, a Monte Carlo simulation method has been used along with predictive device models for the given technologies. Analytical models have been developed for the link failure probability of a repeater inserted interconnect which uses characterization data of all communication structures to give an accurate prediction of the link failure probability. The model has also been extended to calculate the link failure probability of a wider communication link.

Chapter 11
Schedulability Analysis for Real Time On-Chip Communication with Wormhole Switching..... 198
Zheng Shi, University of York, UK
Alan Burns, University of York, UK
Leandro Soares Indrusiak, University of York, UK

In this paper, the authors discuss a real-time on-chip communication service with a priority-based wormhole switching policy. The authors present a novel off-line schedulability analysis approach, worst case network latency analysis. By evaluating diverse inter-relationships and service attributes among the traffic flows, this approach can predict the packet network latency for all practical situations. The simulation results provide evidence that communication latency calculated using the real time analysis approach is safe, closely matching the figures obtained from simulation.

Chapter 12
Modeling Communication in Multi–Processor Systems–on–Chip Using Modular Connectors..... 219
Leonidas Tsiopoulos, Åbo Akademi University, Finland
Kaisa Sere, Åbo Akademi University, Finland
Juha Plosila, University of Turku, Finland

Formal methods of concurrent programming can be used to develop and verify complex Multi–Processor Systems–On–Chip in order to ensure that these systems satisfy their functional and communication

requirements. The authors use the Action Systems formalism and show how asynchronous communication of Multi–Processor Systems–on–Chip can be modeled using generic connectors composed out of simple channel components. The paper proposes a new approach to modeling generic and hierarchical connectors for handling the complexity of on–chip communication and data flow. The authors' goal is to avoid overloaded bus–based architectures and give a distributed framework. A case study presents the authors' modeling methodology.

Chapter 13
Generating Process Network Communication Infrastructure for Custom Multi-Core Platforms.....241
Peter Sørensen, Technical University of Denmark, Denmark
Jan Madsen, Technical University of Denmark, Denmark

We present an approach for generating implementations of abstraction layers implementing the communication infrastructure of applications modeled as process networks. Our approach is unique in that it does not rely on assumptions about the capabilities and topology of the underlying platform. Instead, a generic implementation is adapted to the particular platform based on information retrieved from analyzing the platform. At the heart of the approach is a novel method for analyzing the capabilities of custom execution platforms composed of components. The versatility and usefulness of the approach and analysis method is demonstrated through a case study.

Section 4
Formalisms and Methodologies for Embedded Communication Systems

Chapter 14
Joint Validation of Application Models and Multi-Abstraction Network-on-Chip Platforms..... 263
Sanna Määttä, Tampere University of Technology, Finland
Leandro Möller, Technische Universität Darmstadt, Germany
Leandro Soares Indrusiak, University of York, UK
Luciano Ost, Catholic University of Rio Grande do Sul, Brazil
Manfred Glesner, Technische Universität Darmstadt, Germany
Jari Nurmi, Tampere University of Technology, Finland
Fernando Moraes, Catholic University of Rio Grande do Sul, Brazil

Application models are often disregarded during the design of multiprocessor Systems-on-Chip (MPSoC). This is due to the difficulties of capturing the application constraints and applying them to the design space exploration of the platform. In this article we propose an application modelling formalism that supports joint validation of application and platform models. To support designers on the trade-off analysis between accuracy, observability, and validation speed, we show that this approach can handle the successive refinement of platform models at multiple abstraction levels. A case study of the joint validation of a single application successively mapped onto three different platform models demonstrates the applicability of the presented approach.

Chapter 15
Hierarchical Agent Monitored Parallel On-Chip System:A Novel Design Paradigm and its Formal Specification 278
Liang Guang, University of Turku, Finland
Juha Plosila, University of Turku, Finland
Jouni Isoaho, University of Turku, Finland
Hannu Tenhunen, University of Turku, Finland

In this paper, the authors present a formal specification of a novel design paradigm, hierarchical agent monitored SoCs (HAMSOC). The paradigm motivates dynamic monitoring in a hierarchical and distributed manner, with adaptive agents embedded for local and global operations. Formal methods are of essential importance to the development of such a novel and complex platform. As the initial effort, functional specification is indispensable to the non-ambiguous system modeling before potential property verification. The formal specification defines the manner by which the system can be constructed with hierarchical components and the representation of run-time information in modeling entities and every type of the monitoring operations. The syntax follows the standard set theory with additional glossary and notations introduced to facilitate practical SoC design process. A case study of hierarchical monitoring for power management in NoC (Network-on-chip), written with the formal specification, is demonstrated.

Chapter 16
Service-Oriented Development of Fault Tolerant Communicating Systems: Refinement Approach 297
Linas Laibinis, Åbo Akademi University, Finland
Elena Troubitsyna, Åbo Akademi University, Finland
Sari Leppänen, Nokia Research Center, Finland

Telecommunication systems must have a high degree of availability, that is, a high probability of correct and timely provision of requested services. To achieve this, correctness of software for such systems should be ensured. Application of formal methods helps increase confidence in building correct software. However, to be used in practice, formal methods should be well integrated into existing development process. In this paper, the authors propose a formal model-driven approach to development of communicating systems. The authors formalize and extend the Lyra approach—a top-down service-oriented method for development of communicating systems. Lyra is based on transformation and decomposition of models expressed in UML2. The authors formalize Lyra in the B Method by proposing a set of formal specification and refinement patterns reflecting the essential models and transformations of the Lyra phases. Moreover, this paper extends Lyra to integrate reasoning about fault tolerance in the entire development flow.

Compilation of References 320

About the Contributors 341

Index 350

Preface

The last few years have seen a complete renovation of the way the general public communicates. Not that long ago there was a clear separation of the environments for different types of communication: personal computers, laptops, and netbooks are traditionally used as the primary instruments for communication requiring data networks, for example browsing the World Wide Web and using electronic mail and social media. Television sets are considered the *de facto* environment for watching TV shows and movies, and phones for placing voice calls. In terms of telephony, already some time ago, the world has seen the shift from copper-wired household telephones shared by everyone in the family to personal wireless cellular phones that permit the user to make phone calls in practice everywhere and on the move. The development of cellular telephony first made it possible to start sending short messages with the phones and later enabled data connections to wireless mobile devices. As the capabilities of mobile communication improved, mobile telephone users developed a desire for more and more services for their devices. This has lead to the contemporary trends of convergence in the ways we communicate: first, one can see a user need for wireless communication with universal and always-accessible internet availability. Second, there is a desire for having uniform terminal equipment for most, if not all, communication types, and third, users want to have reasonably-sized battery-operated terminals with tolerable standby times. A modern hand-held mobile communication device is now a small embedded computer optimized for the different types of communication the user often engages in. Smartphones are nowadays used as personal calendars, game consoles, remote controls, music players, social media terminals, e-mail and Web clients, and even televisions, in addition to their original historical usage scenario of placing and receiving phone calls. This makes the systems resemble more computers and home entertainment systems than telephones. The convergence is not leading to a situation where a specific type of communication is taken completely away from its traditional environment, but it is leading more to providing an adequate secondary way of using a variety of communication applications in one device. It is still most users' preference to watch television broadcasts and movies on a large TV screen rather than on a small smartphone screen, but the ability to watch the news broadcast on-demand on a smartphone while on the move is a definite added value to traditional telephony. Similarly, a television user may want the ability to make video calls and browse Web pages using the television set while preferring to make video calls using a smart phone and to browse the Web using a computer.

The world is currently seeing the first stages of this paradigm shift in communication in many parts of the world. In the heart of this development is the active research performed in the field of embedded and real time communication systems. The disciplines of computer science, computer engineering, telecommunication, and communication engineering are well established and scientists, researchers, and industry professionals in these disciplines are numerous in all continents. The convergence of these disciplines into embedded and real-time communication systems is a natural development as can be

seen for example in the smartphone and tablet industries today. The field is interdisciplinary in scope, binding together research from the mentioned disciplines with focus on how they converge to embedded and real-time systems for the communication application domain.

In this development towards more computer-like mobile communication devices, that is, embedded and real-time communication systems, the system-level design process and the design of hardware and software components of the system are facing brand new challenges: despite the small size and reliance on battery power, the devices need to be able to perform ever more complex operations and tasks while maintaining a low enough pricing scheme to ensure adequate market interest. The manufacturers and their design teams are therefore constantly forced to find a balance between adequate performance of the device (the device may not be too slow but it should not provide excess processing performance beyond its application range), low enough manufacturing and design costs, and a short time-to-market. Inherent to all these development trends is the dramatically increased importance of system security (for example, security of the data stored in the device) and communication security (for example, connections through each available network need to be secure). These trends drive the processes of modeling and designing communication systems towards complex and secure communication-enabled embedded systems at an increasing speed.

Forthcoming research in embedded and real-time communication systems needs to target the challenges in future complex converged wireless systems by adventurous development and technological exploration, and experimentation with novel technologies, systems, and system design methodologies. Key research areas in this respect are embedded system design, communication system design, system-wide security, and hardware/software co-design, producing results that converge into novel technologies usable in future secure embedded communication systems. In Embedded system design the software design process needs to take into account special characteristics of the underlying hardware technology. Typically, the data processing resource and the storage resource are significantly more limited than in desktop computers. These limitations follow from the fact that the design goal in embedded systems is usually a small power-efficient device that is able to perform the planned application just within the given time and storage constraints and does not need to be easily extendable to more complex applications. Likewise, the hardware design process needs as input the assumed complexity of the application software to be run in the system so that the hardware design process would produce a device with adequate processing and storage resources. Clearly the hardware and software design processes depend on each other, and how each of them affects the other one is not trivial. A research discipline within embedded system design called hardware/software co-design emerged a decade ago to deal with the problems of interconnecting the two processes, partitioning the target application into hardware and software, and making the partitioning as late as possible to enable incremental addition of details to the specification of the target system independent of hardware and software design issues.

Ideally, the embedded system design process would start from an executable specification written in a compilable programming language. The original executable specification would later serve as a reference model for each stage of the design process. The specification would be refined throughout the design process, ultimately becoming a system model with enough detail for automatic hardware and software generation within given design constraints, producing an HDL hardware description (optionally based on a provided base hardware platform) and a software implementation for the produced hardware description. The resulting hardware/software partitioned model should then be verified by automated comparison to the original reference model. Although hardware/software co-design has been an active research discipline for more than a decade, and there are commercial toolsets for it in existence, the

ideal flow described above is not adequately achievable as many of the key transitions from higher design abstractions to lower ones still require significant amounts of manual work for optimal results, and therefore they are often seen to not reduce the designer's workload as much as one would hope. Alternatively, to achieve relaxed transitions from higher abstraction levels to lower ones, the designer would often be forced to use a very limited subset of the modeling language, which can easily be seen to severely limit the innovativeness of novel designs. One possible research aim to deal with these issues would be to develop hardware/software co-design methods and flows for embedded systems further towards the ideal flow outlined above, focusing especially on domain-specific solutions: for example, the embedded communication systems application domain.

In Communication system design, issues like quality of sound, spectrum utilization efficiency, energy used per amount of transmitted information, and digital transmissions have lead to a number of new technologies to be taken into account in the design process. Already a long ago it has been seen that frequency modulation (FM) improves the quality of analog audio transmissions, and it has been used for a number of radio communication systems, like FM radio and VHF radio systems. Currently, several radio systems originally launched as analog services, such as mobile phone and television networks, have been converted into digital transmission systems in many parts of the world. Also, newer services like wireless local and metropolitan area computer networks have been in the digital domain already since their introduction to general audiences. Common to these digital transmission systems is that in each of them, a digitized data stream is sent over a radio link. This digital data stream might correspond to voice audio, video signals, packet data, and so on. Not only is the data stream sent over the network digitally, but the actual radio device modulating the data for transmission over the air is digital nowadays. An increasing part of receivers are designed using pure digital technology, using either Application Specific Integrated Circuits (ASICs), Digital Signal Processors (DSPs), Field Programmable Gate Arrays (FPGAs), or a combination of these and general purpose processors. In these solutions, only the radio frequency parts work using analog technology; the intermediate frequency (IF) is fed into a fast analog-to-digital converter (ADC), and the rest of the processing is done digitally.

In designing small embedded communication devices that support multiple wireless technologies and also provide many other features unrelated to the communication tasks, the traditional solution of using general purpose programmable microprocessors (CPUs) with accompanying application-domain-specific ASICs (application-specific integrated circuits) is neither an efficient nor a flexible enough solution to handle the increasing demands. The lack of flexibility in such systems is mostly due to the often minimal programmability of ASICs; they are designed for a specific application and are either minimally or not at all capable of performing other similar applications. In addition to the lack of flexibility in such systems, devices consisting of general purpose processors and accompanying ASICs may require excess power (depending on the CPU used) or may cause unnecessary enlargement of overall device size due to the area occupied on the circuit board. One suggested solution to this could be to use FPGAs (field programmable gate arrays) that make true hardware reconfigurability possible. Ideally, with FPGAs, it could be possible to store multiple hardware configurations in the device memory or download new hardware configurations when necessary. This way, a new configuration could be taken into use whenever the system needs to switch from one application or wireless networking technology to another (for example, switching from WLAN to one of the fast 3G communication protocols or vice versa). The clear benefit of this technology is that overall device size could be very small since the application or technology specific circuitry (separate ASICs for each technology) would be replaced ideally by just one reconfigurable circuit. However, the "on-the-fly" reconfiguration of FPGAs as of today

is not usually seen feasible to facilitate a roaming user switching from one radio network to another. Efficient reconfigurability techniques should be explored and developed in the course of forthcoming research as the reconfigurability speed, and also, its usability and cost can be expected to improve to tolerable levels in the future.

Software Defined Radio is the term for software and hardware technologies that enable reconfigurable radio systems. SDR-enabled devices can be dynamically programmed to reconfigure the characteristics of the equipment. An even wider concept is Open Wireless Architecture (OWA), which by definition is an elastic, extensible, and open broadband platform that can support diverse communication standards and can integrate multiple wireless networks. To achieve this flexibility, OWA focuses on all aspects of a communication system such as the RF section, baseband processing, and application domain. Software Defined Radio (SDR) is one of the building blocks of OWA. The goal is that many systems could use a common hardware platform that could be configured in software to function in a desired way in multiple applications. Provided a well-defined base hardware platform, new radio systems could be implemented in a robust way utilizing the ideal principles introduced in SDR and OWA. A well-defined platform should both provide the functional blocks that the radio system requires (as a reconfigurable on-chip communication system) and enable data transports between the functional blocks. Adventurous research to find new technologies to design such SDR and OWA enabled platforms for future converged wireless platforms is an important research area for future embedded and real-time communication systems.

System and communication security is in the heart of the trends of having the ability to connect a single device to multiple networks that differ significantly from each other and on the other hand having small hand-held devices that automatically try to establish network connections whenever they detect a network service is available. In an operating environment like this, the importance of guaranteed data security, data and user authenticity, and data transmission security is becoming more and more crucial every day. Even if much of the information moving in the networks is public, there is always information that for varying reasons must be made unavailable to unauthorized actors on the network. The traditional wired networks could in principle be made totally secure by ensuring that non-authorized actors never have physical access to the network. However, this is seldom the case, and in a wireless environment, basically anyone near enough can be eavesdropping on the communication. Hence, methods are needed for making it impossible, or very difficult, for eavesdroppers to actually use the data being sent over the communications link. Also, as the embedded communication devices nowadays hold an increasing amount of personal information of the owner of the device that could be taken advantage of in identity thefts and financial frauds, the importance of protecting the data stored into the device from misuse for example in cases of device theft or device loss is of extremely high importance.

Traditionally, cryptographic systems relied on a simple algorithm and a lengthy encryption key, like for example in the “one-time-pad”-type encryption, where the algorithm is a simple bitwise XOR operation but the key is as long as the data to be encrypted. Modern cryptosystems have switched to algorithmic complexity and shorter, for example 256, 512, or 1024 bit keys, the length of which is independent from the length of the data to be encrypted. The goal is to have more manageable keys and a software implementation of the encryption algorithm. However, currently the speed of computing systems is forcing the keys to be made longer again to fight brute force attacks on the encryption algorithms. For example, in wireless local area networks, this problem has been seen already some time ago: the WEP encryption of such a network can be broken by brute force in a few minutes by means of cryptanalysis (although one must recall that the WEP encryption method is also flawed by design, resulting in its breaking being easier than without the design flaws). As encryption keys are made longer, and solid and safe encryption

algorithms are used, brute force attacks will take more and more time and potentially become unfeasible again. However, longer keys also generate considerably higher processing requirements for the encrypting and decrypting devices that more and more often today are battery-powered handheld devices. Clearly, one could argue that the history is repeating itself: a pure software implementation on a general purpose microprocessor might no longer be feasible in a hand-held system, and therefore, effort should be placed in researching encryption application-domain-optimized ASIP-type programmable processors that would move the most calculation-intensive parts of the encryption algorithm into domain-optimized hardware execution units of the processor.

The integration of state-of-the-art and future encryption features into embedded communication systems as outlined above is a very important and contemporary research area requiring focus from scientists and engineers in the discipline. Also, the development of novel cryptosystems is a well-established research discipline in the mathematical sciences, and analyzing the feasibility of novel and future cryptosystems for use in embedded communication systems, developing encryption application domain aware hardware platforms and technologies, and based on this work, integrating cryptographic features into embedded communication systems, are research areas requiring attention to facilitate secure embedded and real-time communication system design in coming years.

Convergence of the technologies described in the previous pages is the future of communication devices and their design. The emerging variety of wireless systems and the increase in features that are not directly related to the main communication task are a vital challenge for developers designing novel embedded communication devices. In terms of the communication part, the trend seems to be to adopt SDR and OWA methodologies, whereas the parts unrelated to communication are developed more like computer programs. Both parts of the system require more and more research effort also in the area of data and data transmission security to ensure safe storage of information and safe communication between communicating parties.

This book is a summation volume of articles published in the *International Journal of Embedded and Real-Time Communication Systems* (IJERTCS) in its first volume year, 2010. The journal has an interdisciplinary scope, binding together research from different disciplines, with focus on how the disciplines converge to embedded and real-time systems for the communication application domain. The subject coverage of the journal is broad, which enables a clear presentation of how the research results presented in the journal benefit the convergence of embedded systems, real-time systems, and communication systems. The journal is aimed to benefit scientists, researchers, industry professionals, educators, and junior researchers like PhD students in the embedded systems and communication systems sector. An important aim is to provide the target audience with a forum to disseminate and obtain information on the most recent advances in embedded and real-time communication systems research: to give the readers the opportunity to take advantage of the research presented in the journal in their scientific, industrial, or educational purposes.

As a journal in the focal point of disciplines such as computer science, computer engineering, telecommunication, and communication engineering, the *International Journal of Embedded and Real-Time Communication Systems* is positioned well to provide its readership with interesting and well-focused articles based on recent high-quality research. The journal's coverage in topics from embedded systems, real-time systems, to communications system engineering, and especially how these disciplines interact in the field of embedded and real-time systems for communication, offers its readership both theoretical and practical research results facilitating the convergence of embedded systems, real-time computing, and communication system technologies and paradigms.

This book begins with an introductory chapter by Jouni Isoaho, *et al.* In this chapter, titled "Current Challenges in Embedded Communication Systems," the technological challenges brought by future embedded communication systems are investigated based on current research and trends in the domain. The authors define the key challenges and proceed to proposing potential solutions to deal with the challenges through an analysis of the development trends and historical evolution of embedded systems. The proposed solutions are gathered around three key research areas and challenges for future embedded communication systems: technologies for self-aware autonomous systems, embedded security, and device interoperability. Solutions to the challenges are suggested to be found in the areas of verification and modeling methods, system implementation platforms, and security-enabled designs flows.

After the introductory chapter, the book is organized into four thematic sections. The section on **Technologies for Embedded Communication Systems** focuses on novel technological advances for embedded communication systems. Many different aspects of technological advances are covered: for example, programming techniques, energy efficiency, cache technology, and memory interfaces for embedded communication systems. The selection of topics clearly highlights the multidisciplinary nature of this research area; embedded and real-time communication systems are positioned in the focal points of research of several different Information Technology disciplines.

The second thematic section of this book is **Mobile Communication Applications**. As discussed in the previous pages, it is not only the physical device that constitutes the embedded communication system, but actually it is the applications and the application domain that define the features and functionality of the system. In this thematic section, a selection of important application areas of embedded communication systems are studied, most notably the Long-Term Evolution (LTE) high speed wireless network protocol processing in mobile devices, signal processing algorithms for air interfaces and networks built into devices to interconnect the internal modules and transport application data between the internal modules the device is made of, that is, embedded networks.

The third thematic section is **On-Chip Communication**, where focus is on the research problems regarding reliable communication inside a chip; for example, between the different on-chip cores that together form a Multi-Processor System-on-Chip (MPSoC). An on-chip communication implementation itself can be seen as an embedded and real-time communication system, but very often today, MPSoCs are an essential building part of some larger embedded communication system. The efficient and reliable implementation of on-chip communication is essential for ensuring excellence in device performance and variety in its operating capabilities, and is thus in the heart of the research areas needing scholarly focus for future communication systems and environments.

The last thematic section of the book is **Formalisms and Methodologies for Embedded Communication Systems**. This thematic section focuses on putting the technologies and applications together and making it possible to do high abstraction level design work and verify the correctness of the designs. A very important issue to take into account is the decision making at the methodological level and what kind of effects these decisions have in the lower levels of abstraction in the design. One could see the methodological aspects as the key to successful system integration of all the different technologies needed in a complex embedded communication system. In a modern design process, it is essential to have a well-documented high abstraction level design methodology accompanied with a reliable method for verifying the correctness of the design.

The first section of this book, Technologies for Embedded Communication Systems, comprises of four chapters (chapters 2-5). Chapter 2, by Dake Liu, *et al*., titled "*Parallel Programming and its Architectures Based on Data Access Separated Algorithm Kernels*," presents a novel master-multi-SIMD

architecture and its kernel based parallel programming flow as a parallel signal processing platform, *embedded Parallel DSP processor architecture with Unique Memory Access* (ePUMA). In this approach, data accessing kernels are separated from arithmetic computing kernels in an attempt to minimize the run-time cost of data access. The technique relies on running data access in parallel with algorithm computing. The memory subsystem architecture based on SIMD improves significantly the total computing performance in the presented approach. The hardware system and the programming flow specified for it are primarily targeted to high-performance embedded parallel computing in environments requiring low power consumption and silicon cost. The performed benchmarking shows promising results for the architecture, and its scalability is successfully tested with a configuration of one master and eight SIMD cores.

Chapter 3, "*Towards Sustainable Development of Energy-Aware Systems*," by Luigia Petre and Kaisa Sere, presents an energy-aware modeling and refinement framework for systems with mobile software and hardware resources. Nowadays, more and more effort in system design must be placed on energy consumption: device power consumption needs to be minimized to reduce energy costs and to conserve resources used to produce the energy, and on the other hand, to maximize standby times and recharging intervals. The authors propose two development approaches that can be used to develop energy aware systems in a sustainable manner.

In Chapter 4, written by Arnaldo Azevedo and Ben Juurlink, and titled "*A Multidimensional Software Cache for Scratchpad-Based Systems*," an extension of software caches is discussed. Software caches are used especially in embedded systems where the hardware-controlled caches are too inflexible, for example, regarding the replacement algorithms, prefetching policy, or cache partitioning. In many streaming applications, the working set is well predictable and can be managed using software. In scratchpad-based systems it is important to prefetch data before it is needed and to overlap data transfers with computation in order to hide the memory latency. The authors introduce Multidimensional Software Cache (MDSC) with 1-4 dimensions to mimic such data structures. The software cache overhead can be reduced by using information of the application's access characteristics to reduce the number of cache accesses.

Section 1 of this book ends with Chapter 5, by David Kammler, *et al.* The chapter is titled "*Automatic Generation of Memory Interfaces for ASIPs*." In Systems-on-Chip, the design and implementation of memory interfaces that results from high-level design decisions may be cumbersome and error-prone. In the worst case the implementations have to be manually adapted to the interfaces of dedicated memories or memory controllers, slowing down the design-space exploration regarding the memory architecture of Multi-Processor System-on-Chip (MPSoC) devices. By automating the generation of the memory interfaces required, the authors streamlined the design flow and increased the reliability of the resulting interface logic. The authors introduce a new abstract and versatile description format for memory interfaces and their timing protocols. The feasibility of their approach is proven by real-life case studies including a design space exploration for a banked memory system.

Section 2 of this book is titled Mobile Communication Applications, and it consists of four chapters (6-9). Chapter 6, written by Di Wu, *et al.,* begins this section and is titled "*System Architecture for 3GPP-LTE Modem Using a Programmable Baseband Processor*." This chapter, originally published as a research article in IJERTCS vol. 1 no. 3, was one of the receivers of the highly esteemed 2010 volume year Excellence in Research Journal Awards honoring innovative research and outstanding scholarship. The new 3GPP Long-Term Evolution (LTE) standard is a hot topic among the researchers and engineers in the embedded and real-time communication systems domain. It provides an upgrade path for high performance mobile data communications, but by doing that, it poses new challenges on

the implementation side. The authors address the LTE modem physical layer implementation by using a parallel processing architecture. Their architecture is an application-specific processor adopting the novel Single Instruction Multiple Tasking (SIMT) approach. In this work, the Multiple-Input Multiple-Output (MIMO) symbol detector and a parallel Turbo decoder have been implemented as configurable ASIC accelerator blocks, and the rest of the modem on the programmable processor. The authors conclude that the chosen architecture provides a feasible solution for the LTE CAT4 modem implementation.

Chapter 7 continues in the Long-Term Evolution (LTE) mobile communication domain. The chapter is titled "*Joint Uplink and Downlink Performance Profiling of LTE Protocol Processing on a Mobile Platform*" and is written by David Szczesny, *et al.* The authors address in particular the LTE protocol processing. They measure the Layer 2 protocol processing execution times and maximum data rates on a virtual ARM-based mobile phone platform. The profiling is done for uplink (UL) and downlink (DL) direction separately as well as in a joint UL and DL scenario. As a result, they identify time critical algorithms in the protocol stack and check to what extent state-of-the-art hardware platforms with a single-core processor and traditional hardware acceleration concepts are still applicable for protocol processing in LTE and beyond LTE mobile devices. Their conclusion is that such traditional platforms are incapable to achieve the highest data rates specified for LTE-Advanced downlink transmission. More sophisticated hardware accelerators for the L2 processing will be needed to supply enough computational power required in LTE and especially in next generation mobile devices.

In Chapter 8, Sergey Balandin and Michel Gillet focus on the current state-of-the-art in embedded networks research with the title "*Embedded Networks in Mobile Devices*." Mobile phones are no longer targeted solely for placing calls, but have become complex devices with a rich set of features for personal information management, multimedia processing, and multiple standard compliant communication and computer network connectivity. From the user perspective the development is intriguing as the smartphones of the near future will be able to perform many of the tasks currently executed with laptop computers; however, in terms of the device design and integration, vast challenges exist. One favored approach is to make designs as modular as possible to clarify integration and ease product differentiation. This development has lead to the need for designing embedded networks, network architectures, and specifications for interconnecting the modules of which the devices is comprised. The embedded operating environment poses challenges in the areas of power management and power consumption optimization, protocol design, quality of service, and data transport security.

Chapter 9, by Fabio Garzia, *et al*., concludes the second section of this book. In the chapter titled "*Implementation of FFT on General-Purpose Architectures for FPGA*," the authors evaluate Fast Fourier Transform (FFT) implementations on programmable and reconfigurable architectures. FFT and inverse FFT are intensively applied in modern communication systems based on Orthogonal Frequency Division Multiplexing (OFDM) as their air interface. One challenge is to implement the transforms as flexibly as possible to enable the Software-Defined Radio approach, which can be used to efficiently implement several communication receiver standards on a single device. The authors have characterized and compared such flexible implementations on multi-core and coarse-grain reconfigurable architectures implemented on a programmable logic device. Both of the parallel platforms show a considerable speed-up in comparison with the single-processor reference architecture. The speed-up is higher in the reconfigurable solution, but the MPSoC provides an easier programming interface that is completely based on C language. The authors conclude that their approach of implementing a programmable architecture on FPGA and then programming it using a high-level software language is a viable alternative to designing a dedicated hardware block with a hardware description language (HDL) and mapping it onto FPGA.

The third section of this book is On-Chip Communication, and it consists of four chapters (10-13). The section starts with chapter 10, written by Faiz-ul-Hassan, *et al.* The title of the chapter is "*Performance Analysis of On-Chip Communication Structures under Device Variability*." The increasing device variability in fine linewidth integrated circuits will affect not only the capabilities of the active computing subsystems but also on-chip communication characteristics. The authors have studied the impact of device variability on that issue concerning 13-25 nm processing technologies. They have critically examined the effect of device variability due to random dopant fluctuation (RDF) on the performance of the basic elements of on-chip communication, such as tapered buffer drivers with different tapering factor, repeaters of different sizes, and data storage registers. As a design methodology, scaling up of circuits in the critical paths can be employed to minimize the effects of device variability. However, the study results show that Network-on-Chip (NoC) link failure probability increases significantly with the increase of device variability, and it is a limiting factor in the maximum operating frequency of a synchronous link.

Chapter 11 is titled "*Schedulability Analysis for Real Time On-Chip Communication with Wormhole Switching*," written by Zheng Shi, *et al*. Network-on-Chip (NoC) architectures need to provide different levels of communication service to meet the needs of each component of the applications that are executed by the platform. For real-time application components, a scheduling strategy and an analysis approach for predicting whether all real-time packets are able to meet their timing bounds are required. The authors present an analytical method to predict an upper bound for the packet network latency in NoCs under any practical network traffic condition based on evaluating diverse inter-relationships and service attributes among traffic flows. A comparison to simulation results verifies that the communication latencies predicted using the authors' real-time analysis approach are safe and give results very close to ones obtained from simulations.

Chapter 12, written by Leonidas Tsiopoulos, et al., and titled "*Modeling Communication in Multi-Processor Systems-on-Chip using Modular Connectors*," presents an approach to formally model and verify the functionality of an asynchronous on-chip communication platform in terms of elementary communication components. The authors propose a new approach for modeling generic and hierarchical connectors for handling the complexity of on-chip communication and data flow. The communication infrastructure consists of a distribution of different channels composed into connectors, and further distribution of connectors. Their goals are to avoid overloaded bus-based architectures and to reach a distributed framework. The authors conclude that the approach is useful in modeling complex MPSoCs, and the designer can take advantage of the reusability of formally proven modular components.

The third section of this book ends with Chapter 13, written by Peter Sørensen and Jan Madsen. The chapter, titled "*Generating Process Network Communication Infrastructure for Custom Multi-Core Platforms*," focuses on the problem of automatically generating the communication infrastructure for applications modeled as process networks, targeted to custom execution platforms. The communication approach is based on abstraction layers implementing the channels and does not depend on any assumptions regarding the actual underlying platform. The approach utilizes a generic implementation that is adapted to the target custom platform built of reusable modules. The work includes an analysis method analyzing the low-level services available in the platform and for gathering the information required to synthesize access to the detected services. For the analysis, the platform is described in the service relation model formalism presented in this chapter. The approach also includes a procedure for analyzing user-provided process network mappings for feasibility, and in cases where feasibility cannot be ensured, the procedure attempts to extend the network and mapping so that it becomes feasible. The extension is

carried out by incorporating additional processes and channels into the network. The approach is tested with a case study demonstrating its capabilities.

The fourth and final section of this book is titled Formalisms and Methodologies for Embedded Communication Systems, and it consists of three chapters (14-16). The section starts with Chapter 14, written by Sanna Määttä, *et al.,* and titled "*Joint Validation of Application Models and Multi-Abstraction Network-on-Chip Platforms*." The authors propose an application modeling formalism for joint validation of application and platform models during the design of multiprocessor systems-on-chip (MPSoCs). Presently, there are difficulties in capturing application constraints and utilizing them in the process of platform design space exploration. The proposed approach assists designers in doing trade-off analyses between accuracy, observability, and validation speed, and is able to carry out successive refinement of platform models at multiple abstraction levels. The executable model based approach uses a back-annotation strategy for increased accuracy of application execution during the joint validation of application and platform models. The applicability of the approach is tested and demonstrated with a case study of a single application successively mapped onto three different platform models.

Chapter 15 is titled "*Hierarchical Agent Monitored Parallel On-Chip System: A Novel Design Paradigm and Its Formal Specification*" and is written by Liang Guang, *et al.* The authors present a formal framework to model the characteristics of reconfigurable agent based systems and the monitoring operations flowing between agents at different levels of hierarchy. The authors call their approach HAMSOC, hierarchical agent monitored SoC. HAMSOC is expected to offer resilience and adaptivity to each structural level of a parallel system of any size, this way achieving dependable performance, power efficiency, and all required system features. The approach seems to be quite promising for online monitoring services in massively parallel SoCs.

The fourth section ends with the last chapter of the book, chapter 16, written by Linas Laibinis, *et al.,* and titled "*Service-Oriented Development of Fault Tolerant Communicating Systems: Refinement Approach*." The authors present the formalization of a top-down service based development method supporting correctness preserving refinement and fault tolerance management of complex communication systems. Communication systems are required to have high availability, and to achieve this, system correctness must be ensured. In this chapter the authors focus on software correctness assurance. The authors formalize and extend a top-down service-oriented method for development of communication systems called Lyra. Lyra is based on models expressed in UML2. The authors state that the proposed development approach has a high degree of automation.

Seppo Virtanen
University of Turku, Finland

Chapter 1
Current Challenges in Embedded Communication Systems

Jouni Isoaho
University of Turku, Finland

Seppo Virtanen
University of Turku, Finland

Juha Plosila
University of Turku, Finland

ABSTRACT

This article defines and analyses key challenges met in future embedded systems in networked multimedia and communication applications. Self-awareness, interoperability and embedded security are used to characterize different aspects of designing and implementing next generation embedded systems. The dynamic nature of applications and implementations as well as possible technological faults and variations need to be considered in system verification and modeling. A new design layer needs to be added to current NoC platforms in order to build procedures that take into account dynamic system reconfigurations, fault-tolerance aspects and flexible portability. Increased modularity and networked implementations create a need for trust management mechanisms between system components and technology for analyzing validity and correctness of received application and system configuration information.

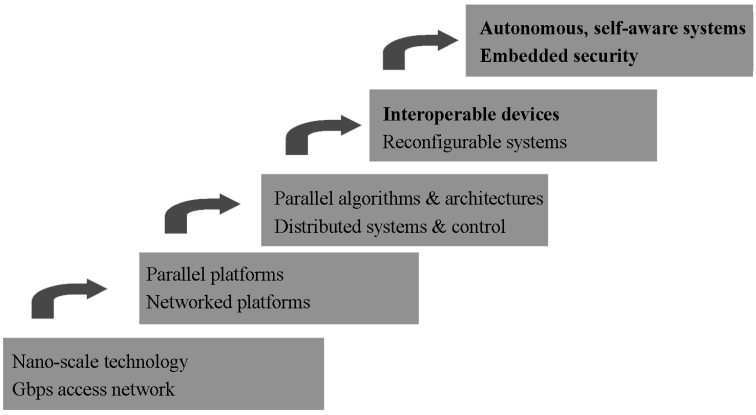
INTRODUCTION

Technological evolution towards the nanoregime has lead to remarkable improvements in capacity and performance of embedded systems. Processing capacity has increased a thousandfold since the early days of embedded systems, allowing their penetration to completely new application areas. The focus has changed from the simple control oriented tasks via peripheral circuitry to mobile and personal high-speed multimedia processing, often with network interfaces. Many of these applications may even require continuous access to very high speed networks everywhere. The

DOI: 10.4018/978-1-4666-0912-9.ch001

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 1. Derivation of challenges in embedded systems



changes in the roles of and the tasks performed by embedded systems create new challenges to be solved during the design process. The derivation of these challenges for embedded systems is presented in detail in Figure 1.

The implementations of future nanoscale electronic systems can be based either on advanced CMOS-style technologies ranging between 0.1 – 0.05 µm, or completely new, emerging technology approaches. In either case, the strategic issue is scalability towards and beyond gigascale integration in three domains:

a. architectural and system scalability with respect to complexity,
b. performance scalability with respect to geometrical scaling in underlying device and circuit structures, and
c. design effort scalability with respect to increased functionality.

Very high-speed access networks approaching gigabit speeds and providing continuous network access provide also a new dimension to system scalability. A system implementation can be composed of multiple networked embedded systems instead of being limited to a single network node. Scalability in all the three perspectives and a demand for networked multiparty connections define the key characteristics required of the platform solution. Thus, the platforms need to provide extensive parallel processing power with seamless network support. In order to benefit from such platforms properly, the system and algorithm level solutions need to provide extensive support for concurrency of operations.

Although quite many systems are basically sequential by their nature, their parallelism can still be increased at the system architecture level. In addition to different types of parallel techniques, also distributing the system functionality into modular self-contained components is vital in order to decrease the complexity of the design phase and to provide splitting of the system into multiple processing units and network nodes. Typically, system components are highly diverse by their functionality and interconnections, yet they must together compose coherent system functionality. Similarly, network nodes and their communication infrastructures might differ from each other heavily. They are often dynamic and time-dependent. Even parallel platforms themselves are seldom highly homogenous by structure. System reconfigurability is necessary to provide support for programming that is targeted to decreasing these dissimilarities and making the platform suitable for multiple applications, thus further increasing product life-times and design economy.

In the development towards more computer-like mobile communication devices, that is, embedded communication systems, the system-level design process and the design of hardware and software components of the system are facing brand new challenges: despite the small size and reliance on battery power, the devices need to be able to perform ever more complex operations and tasks while maintaining a low enough pricing scheme to ensure adequate market interest. The manufacturers and their design teams are therefore constantly forced to find a balance between adequate performance of the device (the device may not be too slow to annoy the user, but on the other hand it should not provide excess processing performance beyond its application range), low enough manufacturing and design costs and a short time-to-market.

Especially due to the usage of different network nodes for implementing embedded systems, device interoperability is a necessity. Device interoperability at the on-chip level is typically increased by utilizing standardized buses and interfaces. In device interoperability, the key focus is moving from inter-component issues onto component level suitability to different types of platforms. Furthermore, due to physical effects producing errors, fault-free pre-tested circuits will be increasingly difficult to manufacture and the process will be cost intensive. Instead of this very costly and time-consuming approach, the key problem to be solved is how to build robust, error-free and highly scalable systems when their basic building blocks can be defective due to static and dynamic errors or failures. These dependability issues together with efficient utilization of programmability and reconfigurability demand an increased amount of on-system intelligence. The issues can not anymore be solved during the actual design phase, but rather the system needs to operate increasingly autonomously and in a self-aware manner during its whole life-cycle. Finally, when considering the multiparty interaction and external networked interconnections, the last crucial issue is the information security aspect which we address in this article as embedded security. Security functionality needs to be an integral built-in part of future embedded systems. The dramatically increased importance of system security (e.g. security of the data stored in the device), implementation security (e.g. the hardware and software components are formally verified to function correctly) and communication security (e.g. connections through each available network need to be secure) drives the processes of modeling and designing communication systems towards complex communication and security enabled embedded systems at an increasing speed.

The article continues as follows: first we briefly define and analyze the key challenges for embedded systems: device interoperability, self-awareness and embedded security. Then we define modern embedded systems through their evolution history. Finally, the impacts of these challenges are discussed from the perspective of practical design issues.

CHALLENGES

The new challenges faced by developers of future complex converged wireless systems can only be met by adventurous development of technologies and technological exploration and experimentation of systems and system design methodologies. Research results from the areas of embedded system design, communication system design, system-wide security and hardware-software co-design need to be brought together to produce multidisciplinary results that converge into novel technologies usable in future secure embedded communication systems. The new challenges of *device interoperability*, *self-awareness* and *embedded security* are a well advised starting point for studying and analyzing the changes happening in the design process of embedded systems.

Device Interoperability

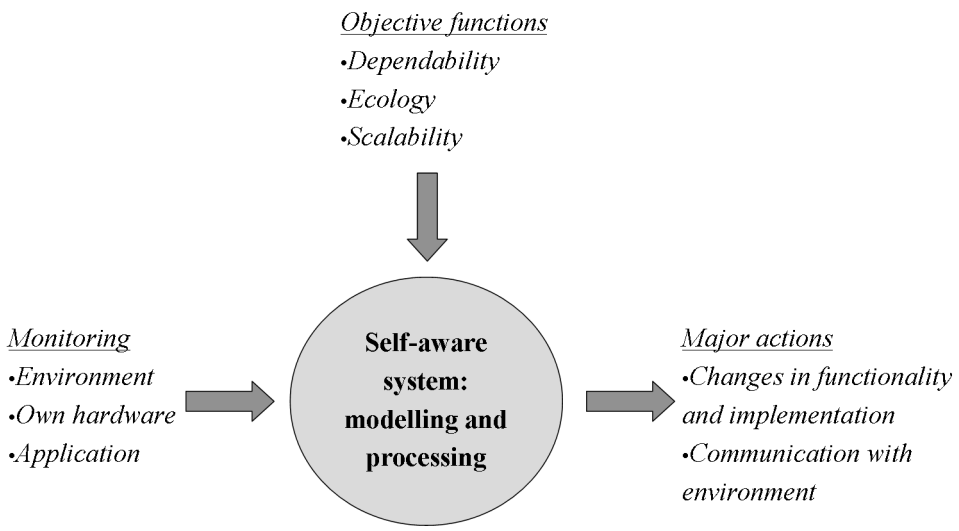
The device interoperability problem can be seen as a very simple and straightforward issue: a system is composed of several subcomponents and they all should work on several platforms. However, it has been a very complicated problem for system designers in embedded and electronic system design already for two decades. HDL based macro libraries and synthesis tools made technology independence a very important topic in hardware design in the early 1990's. Efficiently synthesizable code, and tool and technology independence require very exact design rules to be followed. However, the key problem is how to make these components operate together. For this purpose, standards for multiple interfaces and on-chip buses have been created.

Despite the standardization work, there still is the problem of the existence of several incompatible standards causing an increased complexity of design work. With embedded software, the key problem is how to write code in a way that correct functionality is ensured independent of the processors used and other resources available. Also, there is a need for ensuring adequate processing capacity, guaranteed service times and resource availabilities. In multiprocessor environments, typically processors are quite homogenous or at least their differences are known beforehand. The key problem is that processing in the system needs to be split among several processors instead of a single processor implementation. This means that the components of which the overall system is composed also need to be interoperable in order to allow arbitrary partitioning between different processors. Also, inter-component communication capacity and latency requirements need to be considered. In dynamically reconfigurable systems these two issues need to be considered automatically by self-design algorithms. In networked embedded systems the components can be distributed to separate nodes (independent networked devices) within a multi-standard communication infrastructure. Then the basic assumption is that the processor and the communication links are unknown during the design phase of the system.

Self-Aware Autonomous Agent Based Systems

A self-aware system is monitoring its environment and its own state in order to optimize and match its functions to each situation. The key aspects of a self-aware system are presented in Figure 2. A system needs to monitor the state and the operation of its neighboring systems, the condition of its own hardware and the applications running on it. Based on monitored information and objective functions given, it supervises the processing and the execution of the application. Key objective functions for today's embedded systems can be seen as dependability, ecology and scalability. Dependability means that the system should work in a robust manner under possible hardware or software errors before, during and after reconfiguration, system and platform changes. Ecology means concepts like power consumption, product life-time and R&D costs. Power consumption minimization is mostly finding the most power-efficient implementation for each particular phase of operation, dynamic system or subsystem shutdowns and dynamically optimized communication patterns. Extending product life-times is one of the traditional key aspects of embedded systems due to their reconfigurable and reprogrammable implementation platforms. The target is that new components and specifications can easily be incorporated during product life time. Creating self-aware systems like this is an important goal for future embedded systems. R&D costs can be minimized by changing the design problem from system level back towards component level and moving design responsibility partially to self-design from the actual design phase. In order to make proper self-design decisions, different types of system models and cost functions need to be

Figure 2. A self-aware system



included in a self-aware system. For example, as the environment around the system is typically not fully explorable, it needs to be modeled based on partial information received by sensors and data inputs.

Embedded Security

System and communication security is in the heart of the trends of having the ability to connect a single device to multiple networks that differ significantly from each other and on the other hand having small hand-held devices that automatically try to establish network connections whenever they detect a network service is available. In an operating environment like this, the importance of guaranteed data security, data and user authenticity and data transmission security is becoming more and more crucial every day. Methods are needed for preventing eavesdroppers from using the data being sent over a communications link. Also, as the embedded communication devices nowadays hold an increasing amount of personal information of the owner of the device that could be taken advantage of in identity thefts and financial frauds, the importance of protecting the data stored into the device from misuse for example in cases of device theft or device loss is of extremely high importance. The third dimension to embedded security challenges for future devices is protection of the on-board data transfer system from attacks. Applying encryption usually is not enough to defer for example side-channel attacks that aim to break the encryption by for example analyzing the power consumption of a computation routine.

Modern cryptosystems like RSA (Rivest et al., 1977) and AES (NIST, 2001) rely on algorithmic complexity and relatively short, for example 256, 512 or 1024 bit, keys. The goal is to have manageable keys and a software implementation of the algorithm. However, the increasing processing speed of computing systems will force the keys to be made longer to fight brute force attacks on the encryption algorithms. For example in wireless local area networks (WLANs) the WEP encryption has been known to be weak already for years (Stubblefield et al., 2002), and currently it is common knowledge that it can be broken by brute force in less than an hour provided there is enough data traffic in the network to be used as source data for cryptanalysis.

As encryption keys are made longer, brute force attacks will take more and more time and potentially become unfeasible again. Longer keys also generate considerably higher processing requirements, however, for the encrypting and decrypting devices that are often battery-powered handheld devices. A pure software implementation on a general purpose microprocessor or single-algorithm ASIC co-circuitry might no longer be feasible, and effort should be placed in researching encryption application domain optimized ASIP-type programmable processors that would move the most calculation-intensive parts of the encryption algorithm into domain-optimized hardware execution units of the same processor. Because cryptosystems research is already a well established discipline with a good tradition, the key problem faced by researchers in this area can be identified more in the direction of developing encryption application domain aware reconfigurable and self-aware embedded communication system platforms and technologies that, as integral system components, will be feasible for current and future cryptosystems needed in communication.

DEFINITION OF EMBEDDED SYSTEMS AND DIMENSIONS

Through the history, many definitions for embedded systems have been given. However, they are all somewhat different. In this section we first study some of the existing definitions. Then we briefly look at the evolution of embedded systems from the introduction of first microprocessors to the future distributed networked embedded systems. Based on earlier definitions and our short historical analysis we define and characterize modern embedded systems in general.

The historical classification of processor architecture generations, or computer generations, is a topic that scientists and researchers easily debate on. Stallings (1996) suggests that there have been five generations of computers. The first three generations of these (vacuum tube computers 1946-1957, transistor circuit computers 1958-1964, integrated circuit (IC) computers 1965-1971) are agreed upon by most authors. According to Stallings, the division between the fourth and fifth generation is not clear, although the generation is suggested to have changed with the shift from large scale integration (LSI) based circuit technology (1972-1977) to very large scale integration (VLSI) based circuit technology (1978 onwards). Tanenbaum (1999) more or less agrees with the first three generations mentioned above, although he defines the third generation to be the combination of Stallings' generations 3 and 4 (IC and LSI). Tanenbaum defines the fourth generation to have begun in 1980 in the form of VLSI based computers. In addition, he defines an additional generation, called the zeroth generation for mechanical computers (1642-1945).

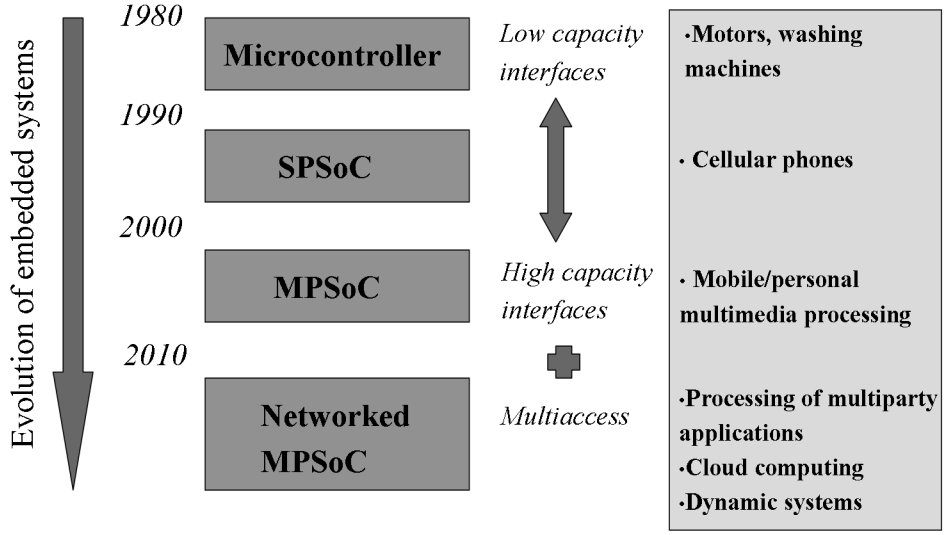
In both cases, the definitions of the most recent generation are already coming of age, and in our view the current state-of-the art can not anymore adequately be described merely by its scale of integration. We see that the current generation of computer architecture is that of firmware-programmable multi-core computation engines that can be general-purpose or customized for specific application domains. A comprehensive coverage of the current generation of processors is provided by Nurmi (2007).

Throughout their history, embedded systems have been designed around a central master processing element representing one of the aforementioned generations of computer architecture: starting from microcontrollers in the early age of embedded systems and reaching multi-processor systems-on-chip (MPSoC) of the current implementations, and in the near future networked MPSoCs as seen in Figure 3. Through time, the definition of embedded systems as well as their evolution (Figure 3) has also been tied to the contemporary computer architecture generation. In the 1980s, microcontrollers gained popularity as programmable single-chip and stand-alone sys-

tems with very versatile built-in peripherals that provided an integrated solution for many simple controlling tasks. Besides these controlling tasks, during the 1990's there was an increasing need for much heavier processing. For example, in cellular phones many algorithms were implemented using embedded signal processor cores. Embedded systems were no longer individual microprocessor or microcontroller chips, but merely cheap low-power embedded processor cores that could be integrated into the same chip or package during the end-product manufacturing. Wolf (1994) approached the definition of embedded systems from the viewpoint of design methodologies, specifically the concept of hardware-software co-design: an embedded system is typically built around a microprocessor to create special-purpose, cost-efficient products for a variety of applications ranging from very simple appliances to safety-critical controllers. The different emphases on system features such as size, power consumption, operating environment, processing capability and reliability place great demands on the overall design process of an embedded system. Well-defined design methodologies for embedded systems were seen as a vital research problem also by Gajski and Vahid (1995). The authors defined the functionality of an embedded system to be fixed and primarily pre-determined by the operating environment and the way the system interacts with it. The characteristics of embedded systems, such as different modes of operations within the device, rapid exception response capability and requirements for concurrent processing make the design process extremely challenging. Camposano and Wilberg (1996) described a typical embedded system as a device implementing dedicated functions such as controlling anti-blocking brakes, assembly lines or processing multimedia codecs. According to them, a typical embedded system is digital and consists of a processor core accompanied with a suitable set of peripherals, memories and embedded software that adapt the system to its particular function.

A similar description of embedded systems and embedded computing is given also by Stankovic et al. (1996), and the authors also make a prediction for the following decade. The prediction includes notions of increasing ubiquity, higher QoS demands and high-integrity real-time services. The authors also mention that in the future the need for highly distributed systems

Figure 3. Evolution of embedded systems



offering a variety of services is eminent. In our view this is actually a very interesting prediction, because it can be seen as one of the currently most intriguing novel research areas in embedded communication systems for the purposes of the future internet: distributed cloud-style computing applications and the increasing need of establishing and maintaining trust between systems using distributed computing resources and applications. An interesting extension of these issues is when the internet is put inside the device: the modules of which the embedded communication system is constructed need to establish trust relationships between each other while being able to perform a distributed computing function.

In the 2000's, the emphasis started to move towards much heavier processing with multicore processors and complex field-programmable gate arrays incorporating embedded core processors and memories in addition to reconfigurable logic facilities. These platforms accelerated the change from Single Processor System-on-Chips (SPSoC) to Multi-Processor System-on-Chips (MPSoC). Mobile systems using MPSoC solutions as the processing platform, and the highly improved communication capacity and access of today are paving the way towards the next step in the evolution of embedded systems: network-connected and network-distributed co-operative embedded systems based on MPSoC processing technology.

As a summary of the four generations of embedded systems as depicted in Figure 3, each generation is built around one or more programmable processing engines. Some of the engines are targeted core processors that can be firmly integrated at the system-on-chip (SoC) or the system-on-package (SoP) level. They interact with their environment with a high variety of ways and interfaces. Especially with early microcontroller based systems the role of the processing engines was clearly that of a stand-alone solution, acting as a concept to embed intelligence to the environment to which it was connected. In such a system the processing engines operate in the role of a master. Today an embedded system can be comprised of multiple co-operative embedded subsystems, interconnected using a dynamic communication infrastructure. Each of the subsystems acts either as a master or a slave, but they can still be seen independent in their operations. We can thus define an embedded system in the following way:

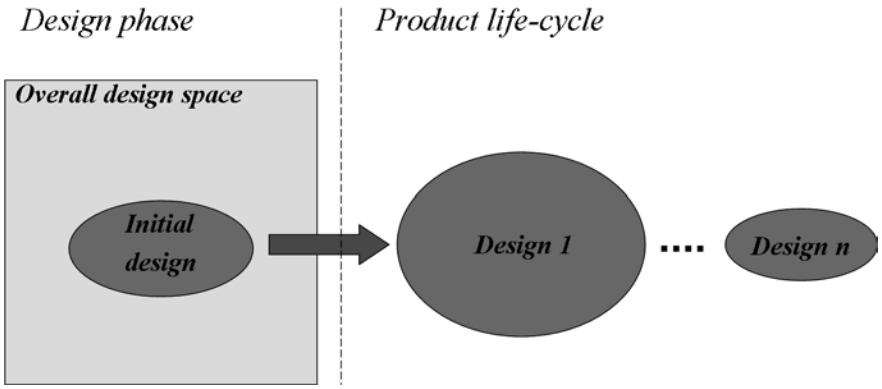
An embedded system has one or more programmable processing engines. The engines operate as an independent stand-alone system either in the role of a master or a slave and interact with their environment.

VERIFICATION AND MODELING

In order to match the challenges described in the previous sections it is essential to study them in more detail in relation to practical system design aspects. The target in self-awareness design is to build into the system the ability to optimize and match its functionality dynamically to each situation. In practice this means that self-design principles need to be adopted. The self-design approach is presented in Figure 4. Systematic support for this kind of an approach is not found in current design methodologies: the division of responsibilities between the actual design phase and the product life-cycle in self-design is very different from existing proposals. During the design phase, initial design is done and this is further redesigned, refined or fine-tuned by the system itself when necessary. This approach presents very high demands on system verification and modelling. The whole design space needs to be analyzed both functionally and physically before the designed system can be shipped out.

Verification of modern embedded communication systems is a tedious task. The verification gap, i.e. the gap between design capacity (functional complexity) and ability to verify, is getting larger and larger. The more parallel and heterogeneous the system is the more difficult the

Figure 4. Self-design approach



verification process becomes. Traditional simulation based verification methods rely on stimulus sets which exercise the functionality of a system model at different abstraction levels of interest. To reach an acceptable simulation (or test) coverage for a complex embedded system with a very large number of possible behavioral scenarios and a vast number of possible individual system states is a cumbersome process, taking much more time than the actual design phases.

Formal methods of parallel systems (Kordon & Lemoine, 2004; Gupta et al., 2004; Fitzgerald et al., 2005; Girault & Valk, 2003) provide means to manage parallelism, process synchronization, modularity, and other design complexity issues. They likewise provide mathematical means to enhance the verification process tackling the verification gap problem. Model checking, equivalence checking, and other types of formal verification are already widely used in design of hardware platforms (e.g. ASIC chips) of embedded systems (Kropf, 1999; Lam, 2005; Perry & Foster, 2005), but they are not yet that common, a standard procedure, in software industry despite the intense academic activity on the area during the recent decades. One reason for this difference between the hardware and software domains is the fact that hardware must be practically flawless while in software verification some compromises can be tolerated. This is because application programs, middleware, and operating systems can usually be updated or patched after their release if and when problems emerge in practice. However, releasing flawed systems cannot be considered an acceptable policy in general, especially in safety critical application areas such as human transportation, medicine, and nuclear power. Hence, there is a constant need for powerful formal methods facilitating efficient co-modeling and co-verification of complex embedded software and underlying computation platforms. The aim of such methods is to enable detection and removal of possible design errors and inefficiencies already at the early phases of the design process decreasing time-to-market and improving system reliability and quality in general.

Abstraction is the keyword in any formal methodology. To abstract away design details still capturing the essential functional aspects into a system model is the way to manage verification of a complex system as a whole. Formal modeling and verification is not a simple procedure: it requires a mindset different from traditional software or hardware engineering. A system model is written in a formal language and is composed of the models of the system's components accompanied by an abstraction of the environment with which the system is intended to interact. Such a model specifies the computational function of the system as well as the communication protocols between the

system's components and between the system and its environment. The formal model is augmented with a sufficient set of safety and liveness properties, logic expressions, which define the expected (correct) behavior of the system within the given environment, including properties dealing with the involved communication protocols. The properties are then verified either dynamically as a part of the simulation process or statically by a formal verification engine.

A class of formal methods focuses more on gradual development of systems, i.e. correctness-preserving stepwise refinement, rather than just verifying properties of system models at different levels of abstraction (Sekerinski & Sere, 1999; Back & von Wright, 1998; Abrial, 1996). Such methods target the whole design process from the initial abstract specification to the final implementable system model promoting a disciplined stepwise design flow which ensures that the final model is correct by construction, a correct implementation of the initial specification. Proof obligations are logic expressions which define mathematically the conditions for a correct transformation, including the properties that need to be preserved from a model to another. They are specified for each transformation or refinement step and proven correct by a theorem prover software. Alternatively, in simulation oriented semi-formal verification, a system model is compared with a more abstract version of the model to verify that the new more concrete model preserves the properties specified for the old one and that the responses of the two models are in accordance for any given stimulus set.

The idea of stepwise refinement can be utilized for example in parallelizing a sequential computational algorithm to improve system performance. The algorithm is gradually developed from the original sequential form to the final parallel form by increasing the level of parallelism at each refinement step. Correctness of the algorithm is preserved through the process. Another application of the same principle is to decompose or partition an initial unstructured system model stepwise into a hierarchy of parallel-operating subsystem models. At some point of such decomposition, separation of the software and hardware domains is carried out by creating, as a series of correctness-preserving refinement steps, a set of software/hardware interface models. They effectively represent the processors which run the processes represented by the subsystem models in the software partition. The hardware partition contains the models for the communication media, memories, and application specific circuit blocks, for example. In the case of a platform based design approach, the decomposition process is systematically directed to conform with a pre-defined computation platform model.

Refinement also enables formal development and verification of the involved communication protocols as a part of the design process. This is an important feature especially in design of communication-intensive systems. Initially a communication medium and a protocol using this medium are modeled in an abstract way (e.g. transaction level model). Then, by stepwise refining the models of the communicating parties and the communication medium between them, a concrete protocol model is gradually created. It could be, for instance, a formal implementable register transfer level model, proven correct by the refinement process.

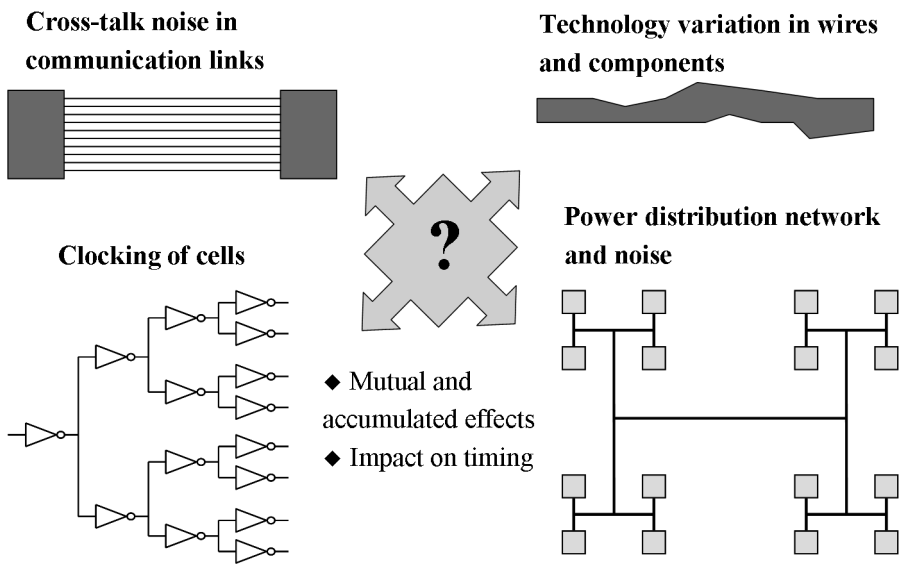
As the above examples demonstrate, refinement is a powerful methodology. Some may consider it too complicated or restrictive for practical use because of the proof obligations that need to be specified and proven for each design step. On the other hand, also in normal model checking there is the difficulty of finding and formulating a sufficient set of properties to be verified. These kind of problems are actually typical for any formal method and require a certain amount of expertise to master. Having said that, the theory and tools of the refinement oriented methods do need further development to enable more efficient

and flexible derivation of modern and forthcoming highly parallel embedded systems.

With appropriate extensions (Schneider, 2000; Tuominen et al., 2008; Tartanoglu et al., 2003), formal methods of parallel systems are capable of modeling and verifying not only functional (logical) system properties but also extra-functional properties such as timing, quality-of-service (QoS), energy, and dependability. All of the mentioned extra-functional properties are important from both hardware and software perspective, and they pose most intriguing challenge for the class of formal methods discussed here. Proving that the fine-grained time constraints of circuit components are met is crucial for stability of hardware. Meeting real-time constraints (operational deadlines), in turn, concerns the whole system, both software and underlying hardware. QoS is a key issue in networked communication systems such as networks-on-chip (NoC). Adding support for traffic congestion analysis and QoS management to a formal modeling and verification framework is indeed a very interesting challenge. The level of energy consumption is especially important for portable battery operated devices. Energy-aware formal modeling and analysis would enable early estimation of energy consumption, based on an abstract system model, and facilitate finding a system configuration which satisfies the given energy/power constraints. An energy-aware framework could be extended with support for formal temperature-aware modeling, making early detection of potential hot spots possible. Dependability is an essential property for any system and it is a fruitful application area for formal modeling and verification. For example, formal methods could be used to avoid deadlocks in networks and to reason about fault tolerance properties of systems. Reconfigurability is another interesting property which is partly related to dependability since reconfiguration can be used to enhance dependability. As reconfigurability is becoming increasingly relevant for embedded system design, it would be important to find ways to formally model and verify dynamically reconfigurable systems with time constraints attached to the modeled reconfiguration operations.

Physical design issues, like cross-talk noise in communication links, clocking structures, power distribution networks and their noise, and technological variations in wires and components have been thoroughly studied during the last twenty years (Bakoglu, 1990; Zheng et al. 2001). However, the vital issues are their mutual and accumulated effects, which are still mostly unknown issues (Figure 5). Only some preliminary studies have been published so far (for example Tuuna, Nigussie et al., 2008; Tuuna, Zheng et al., 2008). As the number of critical parameters needed in physical analyses is increasing heavily, the models are changing towards analytical and system level models. As technological tolerances and their impact on system functionality and performance seem to increase, the self-design approach is almost a necessity to make systems function as specified from the very beginning. This includes moving from the fully synchronous domain towards asynchronous, self-timed operations, which makes system testing and verification a more time-consuming task. In the link level, adjustable drivers and techniques need to be used. The link performance can not anymore be fully characterized, and the performance and the functionality need to first be measured and only then adjusted. This requires a completely new role for the analysis procedure. The overall design space under the worst case technological tolerances needs to be analyzed in order to ensure system functionality also after all possible design and implementation modifications by the self-design procedure. As solving these problems is going to be part of the self-design processing done during the life-time of the embedded system, in the near future they need to be considered in the embedded system design process as well as in the functional verification and modeling processes.

Figure 5. Physical analysis of mutual effects



SYSTEM IMPLEMENTATION PLATFORM

In embedded system design, the second key area to be dealt with is the system implementation platform including the firmware controlling the hardware. There the focus is the firmware layer added above the current embedded NoC platform layer as presented in Figure 6. The NoC platform provides communication and computing facilities. In the development towards more self-aware and autonomous functionality, we propose a new layer for designing system level intelligence. The agent abstraction is one way to address this approach (Russel et al., 2003; Rantala et al., 2007). Both system and application architectures are based on hierarchical co-operative agent systems. An agent abstraction within a platform is presented in Figure 7.

In general, an agent function together with an application function forms the component abstraction that interacts with the underlying IC environment through simple diagnostic and control devices and circuits. Self-awareness and the related models and monitoring techniques are the most critical issue in the process. In our case, this model includes a mechanism for adding dynamic reconfigurability properties to application abstraction, and further includes technology platform-level information for the physical refinement process. After initial design of the system, reconfiguration is needed for three purposes: capacity enhancement of system, fault-tolerance and possible online changes (system upgrades) in specification. Autonomous behavior will be created by combining the control abstraction and reconfigurability capacity to handle the dynamic behavior (for example, robustness towards faults) and performance optimization (for example, load balancing). The simplest solutions for implementation platforms are homogenous processor arrays with symmetric cells and interconnection infrastructures, but of course all reconfigurable and programmable parallel platforms can be used. Various platforms and techniques needed for them are presented in literature (Nurmi et al., 2004; Nurmi, 2007). However, homogeneous processing units and communication links facilitate self-design due to easier device interoperability when for example moving or splitting system function-

Figure 6. Adding a new layer to implementation platforms

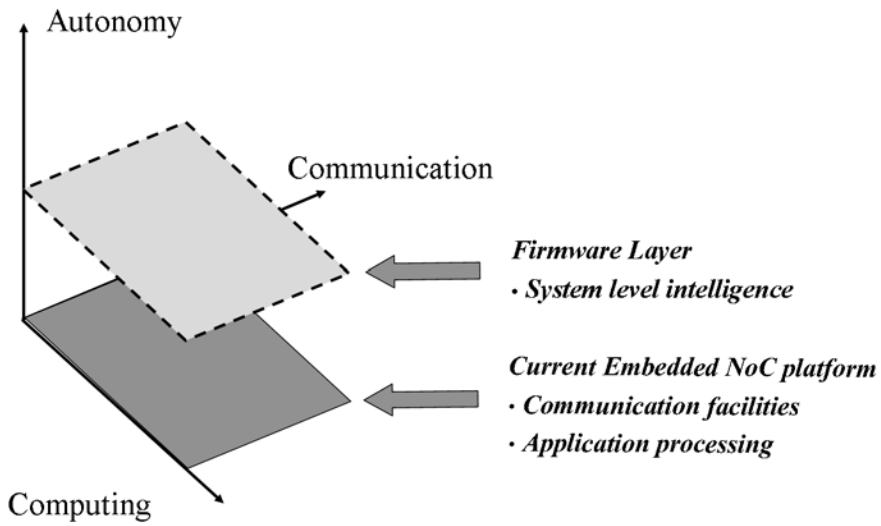
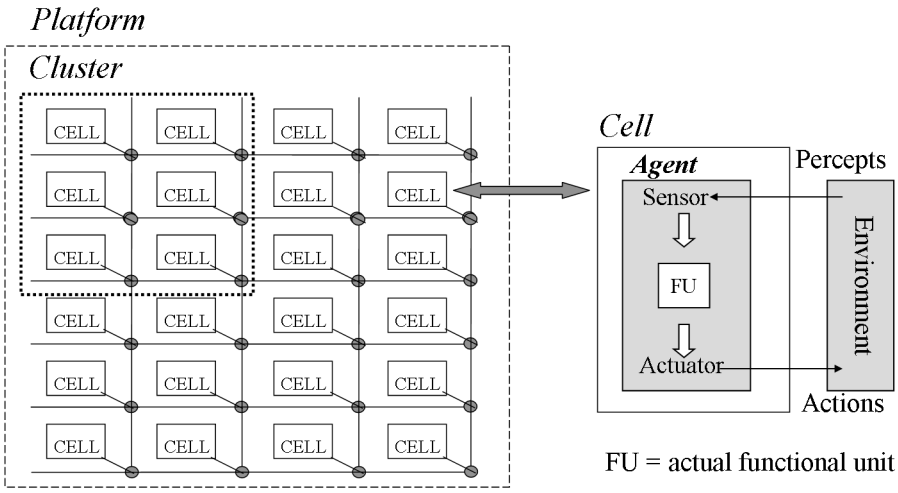


Figure 7. Three levels of platform hierarchy



ality between cells. This approach allows utilization of self-checking properties for avoiding system crashes due to failures or errors occurring during manufacturing or field operations. This agent/control layer is added on top of the current NoC platform approach to improve the modular nature of the system by incorporating autonomous, self-aware properties into component level in order to strengthen self-design and fault-tolerance aspects.

Buses are traditional communication backbones of embedded systems with multiple interacting components such as processors, memory modules, and application specific circuit blocks. Since a bus is a shared communication medium between the system's components, it provides a very cost-effective interconnect method. Buses are a practical medium for establishing device interoperability in MPSoC type embedded systems. Their obvious drawbacks are, for example, the lack of

parallelism in the presence of several bus masters as well as the limited scalability. Both of these problems can be alleviated in part by employing a hierarchical or layered bus architecture with multiple buses operating in parallel and interacting via bridge modules. Buses at different levels of such a hierarchy can either be identical or have different structure and performance characteristics.

Modern system-on-chip buses borrow their basic ideas from their off-chip counterparts. AMBA from ARM (2005) and CoreConnect from IBM (2006) are examples of popular on-chip buses. Typical features of such high-performance multimaster bus architectures include synchronous operation, a layered structure with separate buses for high-speed and slow-speed devices, separate write and read buses (at least at the high-speed layer), scalable data bit width, pipelined operation supporting overlapping consecutive transactions, support for transfer bursts of different sizes, centralized arbitration with programmable priorities, and a bus locking mechanism to enable exclusive accesses. The goal is to have a technology independent, synthesizable bus specification which enhances reusability of components, improves modularity, shortens time-to-market, and minimizes the communication infrastructure. Sonics Inc. has adopted a more aggressive approach in its Micronetwork architecture (Sonics, 2006), where the high-speed part of the interconnect is based on a low-latency crossbar fabric which enables a higher bandwidth than more conventional shared bus solutions like AMBA and CoreConnect. Slower peripheral devices, in turn, are connected with a flexible low-cost bus. Moreover, the Sonics architecture facilitates multiclocked design by providing an asynchronous bridge module with which two crossbar based subsystems operating at different clock frequencies can be connected together. Both open core protocol, i.e. OCP (2009), and AMBA protocol are supported.

EMBEDDED COMMUNICATION SYSTEM SECURITY

Security solutions in embedded systems that do not primarily deal with information security are easily viewed as add-on components to an otherwise complete design. For example, to implement data encryption, there are numerous approaches for specialized co-processors to efficiently handle a particular encryption scheme (for example Liu et al., 2008; Hodjat et al., 2005; Chaves et al., 2006; Zheng et al., 2008). The need for a co-processor that is capable of processing a particular encryption scheme could be seen as a sign of a trend of not including embedded security in the system design process: support for one encryption scheme is added as a co-processor, but if the target application changes or the required security algorithm is upgraded, the single-purpose co-processor may not scale up. This is similar to incorporating security measures to the system to compensate for a newly discovered problem. This type of strategy leads to problems with scalability and also potentially with overall robustness of the system.

There are also approaches where the security co-processor is programmable and supports multiple encryption methods, and potentially also supports yet undiscovered algorithms by means of programmability and optimized hardware, optimally even reconfigurability. Approaches in which the encryption co-processor is optimized for a specific application domain but not limited to a single encryption scheme are also in existence (for example Smyth et al., 2006; Hämäläinen et al., 2005; Han et al., 2008). This type of platform level security design is a significant improvement to scalability and upgradability in comparison to the single-scheme implementations. However, the type of platform solutions described here may still have severe limitations depending on the type of future upgrade encountered: without the option of hardware reconfigurability, the platform is able to execute only algorithms requiring similar (accelerated) operations as needed by the original

implementation, despite the potentially available programmability.

Optimally, security should be built into the system already from the specification stage as an integral component, making the security functionality an equal subsystem within the device and enabling for example more reliable system updates and reconfiguration for novel applications. As in any design process, design verification is a vital step also in an information security functionality enabled design process because the faults in electronic systems that can be exploited to gain access to the data stored in the system or the data communicated by the device are well documented (for example Bar-El et al., 2006).

The need for embedded system design processes integrating security into the design in the early phases of the project has been seen for some time (Kocher et al., 2004); however, a comprehensive embedded system design methodology for integrating security processing into the design process including the modeling and verification phases still remains to be seen. Methodological approaches that could readily scale up to facilitate incorporation of information security functionality to embedded communication system design processes are already in existence, however, although reports of using them to this extent are scarce. Examples of potentially extendable communication system design methodologies for this purpose could be for example the *TTA platform methodology for developing radio systems* suggested by Björkqvist and Virtanen (2006) and the *Development flow for wireless standards* suggested by Anwar et al. (2008).

We identify three functional areas within embedded communication systems where information security plays an important role and needs to be planned for already at the specification phase:

- Protection of data stored in the system
  - Battery powered mobile systems; for example, smartphones
- Protection of data transportation between communicating devices
  - Subsystems for external communication; for example, SDR functionality
- Protection of data transportation within the device
  - Subsystems for internal communication; for example, embedded networks

These three functional areas of embedded communication system security are not separate from each other but typically overlap and converge in system implementations. The fourth vital area of embedded communication system security is physical and mechanical security: for example, the casing needs to protect the internal components from the hazards existing in the operating environment. In this article, however, we focus on each of the aforementioned functional areas of embedded communication system security and will not cover physical and mechanical security.

Protection of Stored Data

Todays battery powered embedded communication systems are extremely likely to be aimed at users requiring highly portable personal devices for business and leisure. The devices, for example smartphones, are expected to provide a feature-rich application set with office, multimedia, network and personal information management applications. The device is expected to be able to connect to a variety of different network types, ranging from cellular telephone networks to open-access wireless computer networks and personal area networks, thus providing access to the internet and its services everywhere via the most suitable available network connection. This richness in features and capabilities has turned such devices into secondary laptops. However, in terms of information security, the users rarely treat the devices similarly as laptops but more like phones, rendering the stored information like

passwords, office application documents or other data vulnerable to anyone able to gain access to the device (Yap & Jones, 2007). Published tools and methods for extracting information stored in smartphones, supporting a variety of different brands and models, are also available, originally aimed for the purposes of digital forensics (for example Distefano & Me, 2008). Further, smartphones are a very desirable target for theft and are lost more easily than laptops due to the way they are used. Clearly, the data stored in a smartphone needs equal, if not better, protection in comparison to data stored in a laptop computer.

To prevent unauthorized access to the data stored in a mobile device like a smartphone, all data should always be encrypted, and the device should be password protected. Some solutions have been presented for the encryption functionality, for example by James (2007); however, the implementations are often device-specific add-on components, possibly protruding outwards from the casing. Further, because the encryption and decryption processes would need to always run in the system, there would be a cost to battery duration and device performance. Incorporating the encryption and decryption functionality at the specification level into the design might alleviate this problem, especially if a low-power implementation taking advantage of parallel processing could be utilized. Requiring the users to use a password in their devices would deviate from the conventional way of using telephones, possibly degrading usability. Biometric authentication, for example fingerprint readers found in many portable systems, could be a solution as an alternative form of authentication in addition to manual password entry.

Protection of Data Transport between Devices

Modern embedded systems often provide communication capabilities to enable information exchange between remote parties. The communication capabilities may be the main feature of the device, as is the case with for example mobile phones or smartphones, or they may be a secondary function that supports or provides added value to the main functionality of the device, for example wireless LAN support in a portable audio player for transporting media files, or a Universal Serial Bus (USB) terminal in a television for viewing multimedia material from external USB devices. In the latter case, we call the system an *embedded communication subsystem for external communication*.

Embedded systems designed for communication intensive applications often provide support for multiple networks and protocols, both wired and wireless. This approach is very beneficial to the end-user as it makes it possible to allow the device to choose the most cost-effective and/or the fastest available network for the purposes of the target communication application. The problem is that connecting to free and fast wireless services may pose great risks in terms of information security: the wireless transport may not be encrypted, and even if it is, the user has no knowledge of the network infrastructure and servers beyond the wireless link. For example, the wireless access point may be connected to a server that records all the plain-text data carried in higher level protocols after the encryption used in the wireless link has been decrypted. Also, depending on the wireless encryption scheme used, breaking the encryption or exploiting its known weaknesses for example during the handshake part of connection instantiation renders the connection vulnerable to eavesdropping. For example Haataja and Hyppönen (2008) and Sher and Magedanz (2007) have recently reported on the vulnerabilities found in different popular wireless network types and problems caused by their convergence.

For smaller devices, the embedded communication susbsystem would ideally be a single-chip system deriving from System-on-Chip (SoC), Network-on-Chip (NoC) and Software Defined Radio (SDR) technologies and hardware and

software reconfigurability. In addition to managing the communication networks and algorithms, some part of the subsystem should handle the information security functionality, highlighting once again the importance of incorporating this functionality deep into the design process already at the specification phase. An approach to the right direction in integrating security into the design is presented by Patel & Parameswaran (2008), where the design flow assigns one processor of an MPSoC system to execute information security functionality. Similar approaches may prove useful in developing methodologies for integrating security functionality into embedded communication subsystems for external connectivity.

Protection of Data Transport within a Device

Similarly as computers are nowadays networked, the internal organization of many modern embedded systems is networked. This strategy allows for a modular hardware structure, potentially making the integration of system components more straight-forward and less time consuming. Because different types of components have different data transportation needs, it is important to have a well designed interconnectivity subsystem, for example an embedded network, within the device. We call this an *embedded communication subsystem for internal communication*.

The subsystem for internal communication transports all data that needs to be passed from one system component to another during its processing by the system. For this reason, the internal communication subsystem needs to be protected from information security attacks. If the information is unencrypted and the casing makes it possible to gain physical access to the internals of the device, reverse engineering the system based on the transported data may become possible (and of course the transported data becomes compromised). Applying encryption alleviates this problem slightly. However, internal communication subsystems are still vulnerable for example to differential power analysis (DPA) attacks. DPA attacks are a form of so-called side-channel attacks (Tiri, 2007), where the attack is based on obtaining physical information on the system performing a cryptographic function rather than attempting to break the encryption. The physical information is then analyzed in the hope that it will reveal patterns in the system's execution of the cryptographic function and for example reveal the decryption key. DPA attacks focus on the power consumption of the system while it executes loops in an encryption algorithm. A DPA analysis can reveal the intermediate results of the calculations performed by the system in the different phases of the encryption algorithm. These types of attacks have been studied by researchers, and for example Bucci et al. (2008) have presented ways of enhancing such attacks. On the other hand, there is active on-going research aiming to find ways of circumventing DPA attacks. Goldwasser et al. (2008) have suggested the use of one-time programs, that is, programs that exist and execute in the system only once, to prevent the possibility of using DPA attacks against the system. Akavia et al. (2009) suggest a method for fighting another type of side-channel attacks, the so called memory attacks by simultaneous use of a cryptographic algorithm and specially formatted bitstrings. These solutions deal with particular problems and already implemented systems. Tiri & Verbauwhede (2006) suggest a design flow for secure integrated circuits, where the solution to the side-channel attack problem is moved to the circuit design flow. Such approaches are recommendable, and we wish to see even more extensive security-enabled embedded processor design methodologies in the nearest future.

CONCLUSION

In this article we studied and analyzed the key changes faced by design engineers and research-

ers in the design of future embedded systems for networked multimedia and communication applications. We derived the challenges in embedded systems starting from the technology basis, the nanoscale technologies and gigabit access networks to the new key focuses: self-awareness, interoperability and embedded security. As our second analysis approach we defined embedded systems based on their historical evolution in four development steps or generations. Finally, we discussed how the new challenges and the technological development impact practical design problems. The dynamic nature of applications and implementations as well as possible technological faults and variations need to be considered in system verification and modeling. The whole design space used by self-design procedures needs to be considered during the design phase. A new design layer needs to be added to current NoC platforms in order to build procedures that take into account dynamic system reconfigurations, fault-tolerance aspects and flexible portability. We described this autonomous or self-aware characteristic using an agent abstraction. Increased modularity together with networked implementations requires information security aspects to be considered, creating a completely new challenge for embedded system design. There is a clear need for building trust management mechanisms between system components, and for developing technologies for analyzing the validity and correctness of received application and system configuration information.

REFERENCES

Abrial, J.-R. (1996). *The B-book: Assigning programs to meanings*. Cambridge, UK: Cambridge University Press.

Akavia, A. Goldwasser & Vaikuntanathan, V. (2009). Simultaneous hardcore bits and cryptography against memory attacks. In O. Reingold (Ed.), *Theory of cryptography* (LNCS 5444, pp. 474-495).

Anwar, M. I., Virtanen, S., & Isoaho, J. (2008). A software defined approach to common baseband processing. *Journal of Systems Architecture*, *54*(8), 769–786. doi:10.1016/j.sysarc.2008.01.007

ARM. (2005). *AMBA*. Retrieved from http://www.arm.com

Back, R., & von Wright, J. (1998). *Refinement calculus: A systematic introduction*. New York: Springer-Verlag.

Bakoglu, H. B. (1990). *Circuits, interconnections, and packaging for VLSI*. Reading, MA: Addison-Wesley.

Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., & Whelan, C. (2006). The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, *94*(2), 370–382. doi:10.1109/JPROC.2005.862424

Björkqvist, J., & Virtanen, S. (2006). Convergence of hardware and software towards platforms for radio technologies. *IEEE Communications Magazine*, *44*(11), 52–57. doi:10.1109/MCOM.2006.248165

Bucci, M., Giancane, L., Luzzi, R., Marino, M., Scotti, G., & Trifiletti, A. (2008). Enhancing power analysis attacks against cryptographic devices. *IET Circuits. Devices and Systems*, *2*(3), 298–305. doi:10.1049/iet-cds:20070166

Camposano, R., & Wilberg, J. (1996). Embedded system design. *Design Automation for Embedded Systems*, *1*(1-2), 5–50. doi:10.1007/BF00134682

Chaves, R., Kuzmanov, G., Vassiliadis, S., & Sousa, L. (2006) Reconfigurable memory based AES co-processor. In *Proceedings of the International Parallel and Distributed Processing Symposium* (p. 8).

Distefano, A., & Me, G. (2008). An overall assesment of mobile internal acquisition tool. *Digital Investigation: The International Journal of Digital Forensics & Incident Response*, *5*, S121–S127.

Fitzgerald, J., Larsen, P., Muherjee, P., Plat, N., & Verhoef, M. (Eds.). (2005). *Validated designs for object-oriented systems*. London: Springer-Verlag.

Gajski, D. D., & Vahid, F. (1995). Specification and design of embedded hardware-software systems. *IEEE Design & Test of Computers*, *12*(1), 53–67. doi:10.1109/54.350695

Girault, C., & Valk, R. (Eds.). (2003). *Petri Nets for systems engineering: A guide to modeling, verification, and applications*. Berlin Heidelberg, Germany: Springer-Verlag.

Goldwasser, S., Kalai, Y. T., & Rothblum, G. N. (2008). One-time programs. In D. Wagner (Ed.), *Advances in Cryptology - CRYPTO 2008*. (LNCS 5157, pp. 39-56).

Gupta, R., Le Guernic, P., Shukla, S. K., & Talpin, J.-P. (Eds.). (2004). *Formal methods and models for system design: A system level perspective*. Boston: Kluwer Academic Publishers.

Haataja, K., & Hyppönen, K. (2008). Man-in-the-middle attacks on Bluetooth: A comparative analysis, a novel attack, and countermeasures. In *Proceedings of the 3rd International Symposium on Communications, Control and Signal Processing* (pp. 1096-1102).

Hämäläinen, H., Heikkinen, J., Hannikainen, M., & Hämäläinen, T. D. (2005). Design of transport triggered architecture processors for wireless encryption. In *Proceedings of the 8th Euromicro Conference on Digital System Design* (pp. 144-152).

Han, L., Han, J., Zeng, X., Lu, R., & Zhao, J. (2008). A programmable security processor for cryptography algorithms. In *Proceedings of 9th the International Conference on Solid-State and Integrated Circuit Technology* (pp. 2144-2147).

Hodjat, A., Hwang, D. D., Lai, B., Tiri, K., & Verbauwhede, I. (2005). A 3.84 gbits/s AES crypto processor with modes of operation in a 0.18 µm CMOS technology. In *Proceedings of the 15th ACM Great Lakes Symposium on VLSI* (pp. 60-63).

IBM. (2006). *CoreConnect*. Retrieved from http://www.ibm.com

James, P. (2007). Can SDV technology be utilized in a smartphone to prevent forensic analysis? In *Proceedings of the 5th Australian Digital Forensics Conference* (pp. 164-178).

Kocher, P., Lee, R., McGraw, G., Raghunathan, A., & Ravi, S. (2004). Security as a new dimension in embedded system design. In *Proceedings of the 41st Annual ACM IEEE Conference on Design Automation* (pp. 753-760).

Kordon, F., & Lemoine, M. (Eds.). (2004). *Formal methods for embedded distributed systems: How to master the complexity*. Boston: Kluwer Academic Publishers.

Kropf, T. (1999). *Introduction to formal hardware verification*. Berlin Heidelberg, Germany: Springer-Verlag.

Lam, W. K. (2005). *Hardware design verification: Simulation and formal method-based approaches*. Upper Saddle River, NJ: Pearson Education.

Liu, Y., Chen, P., Xie, G., Liu, Z., & Li, Z. (2008). The design of low-power asynchronous DES coprocessor for sensor network encryption. In. *Proceedings of the International Symposium on Computer Science and Computational Technology*, *2*, 190–193. doi:10.1109/ISCSCT.2008.228

NIST, National Institute of Standards and Technology. (2001). *Advanced encryption standard* (FIPS-197). Gaithersburg, MD: Author.

Nurmi, J. (Ed.). (2007). *Processor design - System-on-chip computing for ASICs and FPGAs*. Dordrecht, The Netherlands: Springer.

Nurmi, J., Tenhunen, H., Isoaho, J., & Jantsch, A. (Eds.). (2004). *Interconnect-centric design for advanced SoC and NoC*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

OCP. (2009). *OCP-IP*. Retrieved from http://www.ocpip.org

Patel, K., & Parameswaran, S. (2008). LOCS: A low overhead profiler-driven design flow for security of MPSoCs. In *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (pp. 79-84).

Perry, D. L., & Foster, H. D. (2005). *Applied formal verification*. New York: McGraw-Hill.

Rantala, P., Isoaho, J., & Tenhunen, H. (2007). Novel agent-based management for fault-tolerance in network-on-chip. In *Proceedings of 10th EUROMICRO Conference on Digital System Design* (pp. 551-555).

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120–126. doi:10.1145/359340.359342

Russel, S., & Norvig, P. (2003). *Artificial intelligence - a modern approach* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall.

Schneider, S. (2000). *Concurrent and real-time systems: The CSP approach*. Chichester, UK: John Wiley & Sons.

Sekerinski, E., & Sere, K. (1999) (Eds). *Program development by refinement: Case studies using the B method*. London: Springer-Verlag.

Sher, M., & Magedanz, T. (2007). 3G-WLAN convergence: Vulnerability, attacks, possibilities and security model. In *Proceedings of the 2nd International Conference on Availability, Reliability and Security* (pp. 198-205).

Smyth, N., McLoone, M., & McCanny, J. V. (2006). WLAN security processor. *IEEE Transactions on Circuits and Systems*, *53*(7), 1506–1520. doi:10.1109/TCSI.2006.877888

Sonics. (2006). *Micronetwork architecture*. Retrieved from http://www.sonicsinc.com

Stallings, W. (1996). *Computer organization and architecture - designing for performance*. Upper Saddle River, NJ: Prentice-Hall.

Stankovic, J. A., Burns, A., Jeffay, K., Jones, M., Koob, M., & Lee, I. (1996). Strategic directions in real-time and embedded systems. *ACM Computing Surveys*, *28*(4), 751–763. doi:10.1145/242223.242291

Stubblefield, A., Ioannidis, J., & Rubin, A. D. (2002). Using the Mantin, Shamir and Fluhrer Attack to break WEP. In *Proceedings of the 2002 Network and Distributed Systems Security Symposium* (pp. 17-22).

Tanenbaum, A. (1999). *Structured computer organization* (4th ed.). Upper Saddle River, NJ: Prentice-Hall.

Tartanoglu, F., Issarny, V., Levy, N., & Romanovsky, A. (2003). Formalizing dependability mechanisms in B. In *Proceedings of the ICSE'2003 Workshop on Architecting Dependable Systems.*

Tiri, K. (2007). Side-channel attack pitfalls. In *Proceedings of the 44th Annual Conference on Design Automation* (pp. 15-20).

Tiri, K., & Verbauwhede, I. (2006). A digital design flow for secure integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *25*(7), 1197–1208. doi:10.1109/TCAD.2005.855939

Tuominen, J., Westerlund, T., & Plosila, J. (2008). Power aware system refinement. *Electronic Notes in Theoretical Computer Science*, *201*, 223–253. doi:10.1016/j.entcs.2008.02.022

Tuuna, S., Nigussie, E., Isoaho, J., & Tenhunen, H. (2008). Analysis of delay variation in encoded on-chip bus signaling under process variation. In *Proceedings of the 21st IEEE International Conference on VLSI Design* (pp. 228-234).

Tuuna, S., Zheng, L.-R., Isoaho, J., & Tenhunen, H. (2008). Modeling of on-chip bus switching current and its impact on noise in power supply grid. *IEEE Transactions on Very Large Scale Integration (VLSI)*. *Systems*, *16*(6), 766–770.

Wolf, W. H. (1994). Hardware-software co-design of embedded systems. *Proceedings of the IEEE, 82(*7), 967-989.

Yap, L. F., & Jones, A. (2007). Profiling through a digital mobile device. In *Proceedings of the 5th Australian Digital Forensics Conference* (pp. 52-58).

Zheng, L.-R., & Tenhunen, H. (2001). Fast modeling of core switching noise on distributed LRC power grid in ULSI circuits. *IEEE Transactions on Advanced Packaging*, *24*(3), 245–254. doi:10.1109/6040.938290

Zheng, X., Liu, Z., & Peng, B. (2008). Design and implementation of an ultra low power RSA coprocessor. In *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing* (pp. 1-5).

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 1, edited by Seppo Virtanen, pp. 1-21, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Section 1
Technologies for Embedded Communication Systems

Chapter 2
Parallel Programming and Its Architectures Based on Data Access Separated Algorithm Kernels

Dake Liu
Linköping University, Sweden

Joar Sohl
Linköping University, Sweden

Jian Wang
Linköping University, Sweden

ABSTRACT

A novel master-multi-SIMD architecture and its kernel (template) based parallel programming flow is introduced as a parallel signal processing platform. The name of the platform is ePUMA (embedded Parallel DSP processor architecture with Unique Memory Access). The essential technology is to separate data accessing kernels from arithmetic computing kernels so that the run-time cost of data access can be minimized by running it in parallel with algorithm computing. The SIMD memory subsystem architecture based on the proposed flow dramatically improves the total computing performance. The hardware system and programming flow introduced in this article will primarily aim at low-power high-performance embedded parallel computing with low silicon cost for communications and similar real-time signal processing.

1. INTRODUCTION

The programming and the architecture of real-time parallel computing for on-chip multicore computers are based on either general computing solutions or custom solutions. General solutions, usually based on a cache-coherent programming model, are not low cost solution for real-time applications (Hennessey & Patterson, 2003). Custom solutions are application-specific and suitable only for a selection of applications, such as LeoCore

DOI: 10.4018/978-1-4666-0912-9.ch002

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

of Coresonic (Nilsson, Tell & Liu, 2008). Parallel programming based on architectures with local scratchpad memories associated with ultra large register files was proposed by Flachs et al., 2006, Khailany et al., 2008. A large register file supports flexible parallel programming and consumes much power. Parallel computing based on a VLIW DSP processor has been well used in industry (Tretter, 2003). However, VLIW based DSP processors cannot offer silicon efficiency and low power.

Currently master (host)-multi-SIMD based architecture is the main driver of embedded DSP computing. Several hundreds GOPS computing performance offers great opportunities for computationally demanding applications, yet some applications cannot be supported because of the high power consumption. A majority share of power is consumed during data access for parallel computing. Excessive and redundant parallel data access drives the clock frequency to a very high rate, so that the power consumption cannot be reduced by lowering the supply voltage.

1.1. Essential Glossary

OpenCL

OpenCL (Open Computing Language) (Khronos, 2008) is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors.

Kernel

The definition of a kernel by OpenCL: A *kernel* is a function declared in a *program* and executed on an OpenCL *device*. A *kernel* is identified by the __kernel qualifier.

From control complexity: A kernel is a subroutine executed independently in a SIMD or in an accelerator without interwork to its host machine or other SIMD.

From data complexity: Kernel is a computation that uses single the regular memory access pattern for each operand array (using only one addressing kernel / template).

From algorithm complexity: A kernel shall handle only one algorithm or part of an algorithm which can be implemented using only one loop.

Cluster

A *cluster* here consists of one master (host) machine and several SIMD machines.

Total data access cost in SIMD:

The run time cost of (1) loading data from the main memory to the SIMD local vector memory, (2) loading data from SIMD local vector memory to the vector register file, and (3) storing results from SIMD local vector memory to the main memory.

Data permutation:

The data permutation here in this article is used to select each piece of data in a vector and to store it in a memory block of the vector memory. It can be conducted during the data loading from the main memory to the local vector memory. The purpose of data permutation is to distribute data to different memory blocks in a vector memory so that multiple data values can be used in parallel simultaneously.

Conflict free memory access:

Based on data permutation, data is selected to be stored in different memory blocks. Multiple data can be accessed in parallel without conflict, facilitating parallel computing.

Separated data access kernel:

The data access kernel is separated from its original algorithm kernel. A kernel carries the data location information in the main memory and in the local vector memory. It also specifies the way

that the data in the main memory is collected and merged into one DMA transaction, and the way that the data shall be distributed to each block of the vector parallel memory.

Prolog and Epilog in host:

It is a part of a context; a *prolog* is used to introduce a kernel to be executed in a SIMD machine and an *epilog* is used to terminate a kernel executed in a SIMD machine.

Prolog and Epilog in SIMD:

The *prolog* is the initial part of a SIMD kernel program and the *epilog* is the finishing part of a SIMD kernel program. A prolog in SIMD introduces the regular parallel computing by aligning data and data access. An epilog in SIMD handles the final irregular part of a SIMD kernel after executing the regular parallel computing.

1.2. HW-SW Relations

A basic OpenCL program consists of three main parts: the top level code, the context code, and the kernel code. A kernel is further divided into the data access kernel and the algorithm kernel. The top-level code links all code parts together. The context code prepares and issues a kernel. A data access kernel is parallel code to prepare data required by the algorithm kernel. An algorithm kernel is parallel code executed in parallel with the data access kernel.

There are three hardware subsystems to run the code: the host machine running the top-level codes and context, the datapath of a SIMD machine running algorithm kernels, and the data access path in the SIMD machine running data access kernels. In the following sections, three hardware subsystems will be discussed based on the code allocated in the hardware in *Figure 1*.

2. MASTER-SIMD MACHINE ARCHITECTURE

A complete Master(host)-SIMD architecture can be illustrated in *Figure 2*.

In this figure, one master (host) machine can carry several SIMD machines. In this paper, one master machine can carry up to eight SIMD machines, according to benchmarks and to the task partitioning of ePUMA. The master machine should execute tasks that cannot be executed in parallel. Statistically, there are about 10% of the

Figure 1. HW allocation of tasks

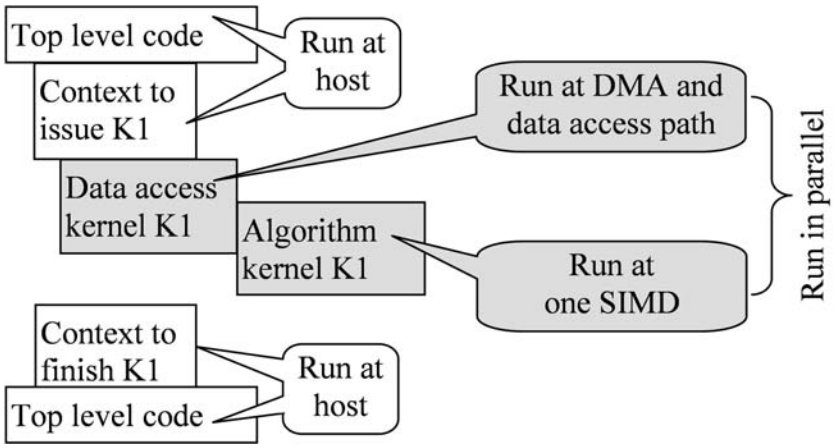
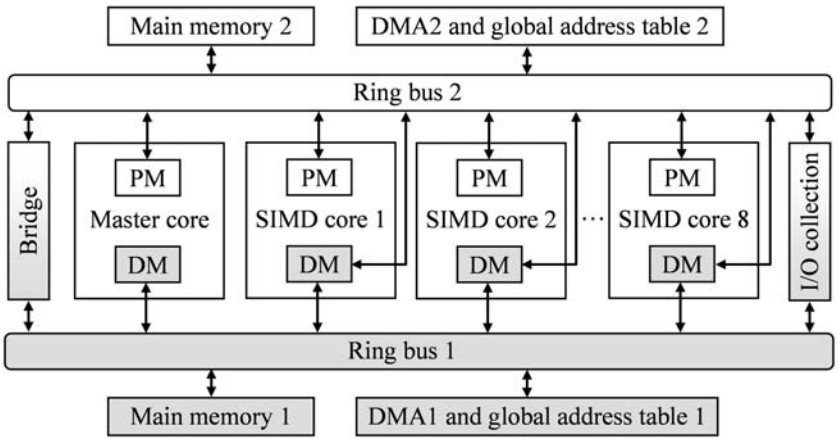


Figure 2. A cluster architecture, consisting of a Master and 8 SIMD machines



tasks in a normal application that cannot be executed in parallel.

In *Figure 2*, DM is a data memory or a vector data memory linked to the main memory 1 by ring bus 1. Ring bus 1 is controlled by the master controller and data transaction on ring bus 1 is conducted by the DMA 1 (direct memory access handler). PM is a program memory and a parameter memory linked to the main memory 2 by ring bus 2. Ring bus 2 is also used for broadcasting data from any SIMD machine to other SIMD machines when cache coherence is introduced.

In our paper, a cluster consists of one master machine and several SIMD machines. In most cases, the performance from a cluster is sufficient for most low power DSP applications. At the same time, the silicon cost of a cluster could reach the limit when the DM and PM in each machine are of average and reasonable size. There are cases where multiple clusters are integrated for applications with a very small data buffer and requiring unusually high performance. The following example in Figure 3 carries the integration of four clusters.

A simplified Master(host)-SIMD with only one SIMD machine is shown in Figure 4. There are two computing paths (sometimes called left and right paths, respectively): the load-store unit as the address generator for conflict-free parallel data access (the left path) and the SIMD datapath for parallel arithmetic computing (the right path). The address generator consists of addressing calculation logic including a permutation table. The permutation table will be used both for DMA transactions and for loading data to the vector register file. The research goal is to run the load-store unit in parallel with the datapath so that the total data access cost can be negligible.

3. MEMORY SUBSYSTEM

The memory subsystem of a SIMD machine consists of hardware (HW) and software (SW). The HW includes two paths: one from the main memory holding the data of the datapath consuming the data; the other one from the datapath generating results to the main memory storing result data. The SW consists of the code loading data, the code storing data, the control codes for DMA, and the bus configuration. Loading and storing data in a memory subsystem is illustrated in *Figure 5*.

The data access code in Figure 5 stands for the execution of loading and storing vector data. Arithmetic OP in Figure 5 shows the execution of the vector ALU instructions. There are three hops in the data access flow: the hop from the main memory to the local memory as a DMA transaction, the hop from the SIMD local vector

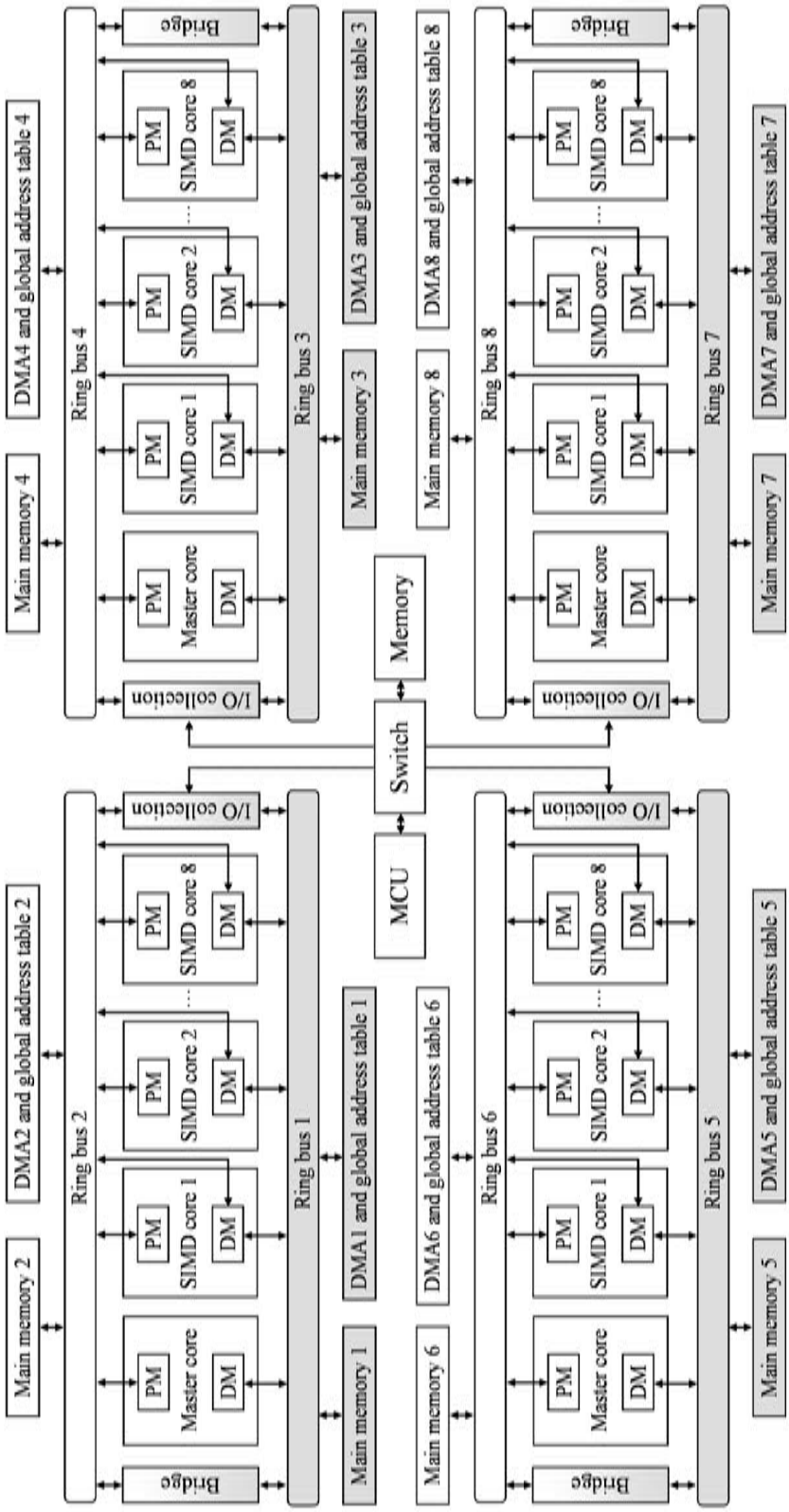


Figure 3. Multi-cluster integration

Figure 4. A simplified Master-SIMD architecture

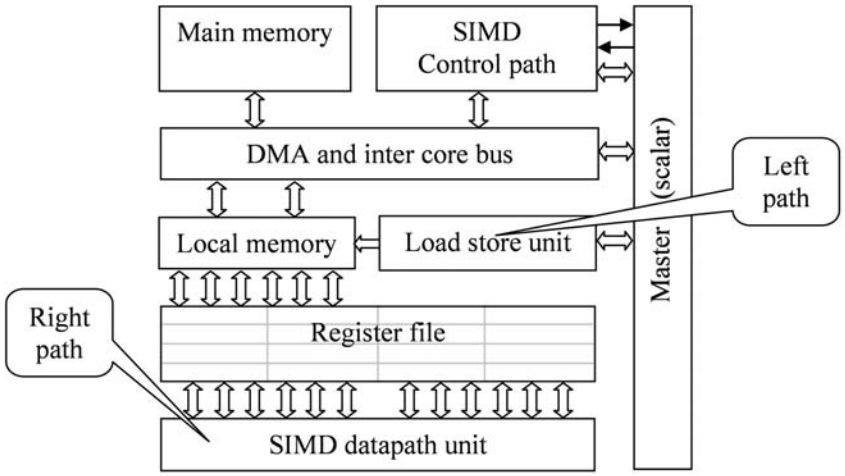
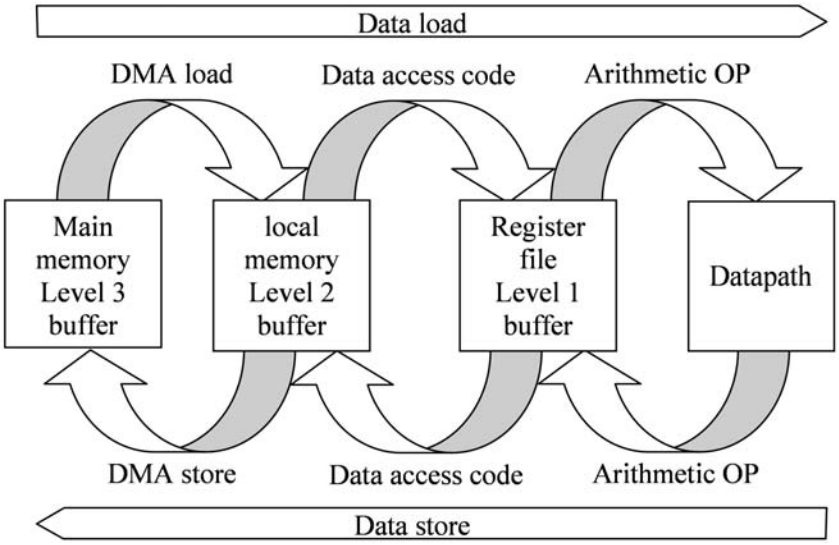


Figure 5. Load store data in a memory subsystem and memory hierarchy



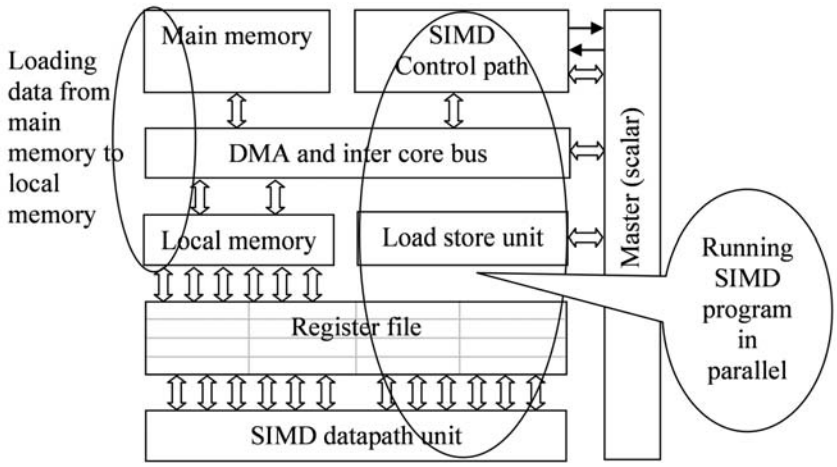
memory to the vector register file of the SIMD machine by executing the load-store code, and the hop from the vector register file to the ALU computing unit by executing the ALU codes.

3.1. The Goal of Designing our Memory Subsystem

The ultimate goal of the memory subsystem design is to minimize the total data access cost in SIMD. It is to completely hide the data access time cost so that the computing latency is equal to the cost of arithmetic computing. There are three steps to reach the goal illustrated in Figure 6 and Figure 8 (both figures were originally from Figure 4 in this paper).

The first step of data loading is to load data from the main memory to the local vector memory in a SIMD. It is usually a DMA transaction prepared by the master. While the DMA is loading

Figure 6. The first step to hide cost of data accessing



data, the SIMD machine can run a task in parallel if the local vector memory is not used by the running task. There are two ways to run the DMA loading in parallel with a task running in SIMD; One way is to use ping-pong memory, while data loading and data consuming by a task can be arranged in different physical memories. Another way is to design a relative large vector register file so that the register file can support data consumption in a relatively long period of time. In our project, relatively small ping-pong memories will be used. If the data width of the main memory is the same as the data width of the local vector memory, the permutation table can be used for both DMA loading and for loading data from the local vector memory to the vector register file. The address code of a load store instruction can simply be the address of the permutation table.

The second step is to load data from the local vector memory to the register file of the SIMD machine. There are two ways to run the register file loading in parallel with the running of a task in a datapath. One way is to issue two instructions per clock cycle, one instruction loads data to the register file, while another runs the arithmetic computing in the SIMD datapath. Another way is to load data to the vector register file, while a SIMD machine is running a SIMT task. Here SIMT stands for Single Instruction Multiple Tasks. A SIMT instruction is a task level instruction. The concept is discussed in Chapter 20 of the book Embedded DSP Processor Design (Liu, 2008). A SIMT task is usually a simple iterative loop function and is handled by a FSM in a datapath. (See Figure 7)

There are four physical circuit paths to allocate the four kinds of tasks in pipeline and parallel modes. The first path is the master; it prepares for a task including its DMA transaction, the PTB (permutation table) for the DMA transaction, the context to prepare for a task, and the context to terminate the task. The second path is the DMA hardware, including the main memory, and the ring bus. It offers data from the main memory to the local vector memory. The third path is the SIMD data access path, consisting of an address generator and a PTB. It moves data from the local vector memory to the VRF (vector register file). Finally, the SIMD datapath consumes data from VRF. By running four circuit paths in parallel, the data access latency can theoretically be hidden behind the arithmetic computation in the SIMD datapath. There are up to 8 SIMD data access paths and 8 SIMD datapaths. There are actually up to 18 circuit paths running in parallel when there is one master handling 8 SIMD machines.

Figure 7. Four hardware paths running tasks in parallel

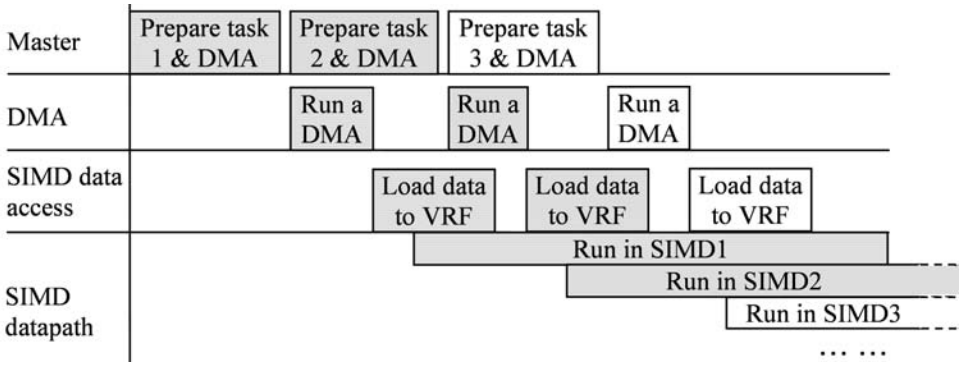
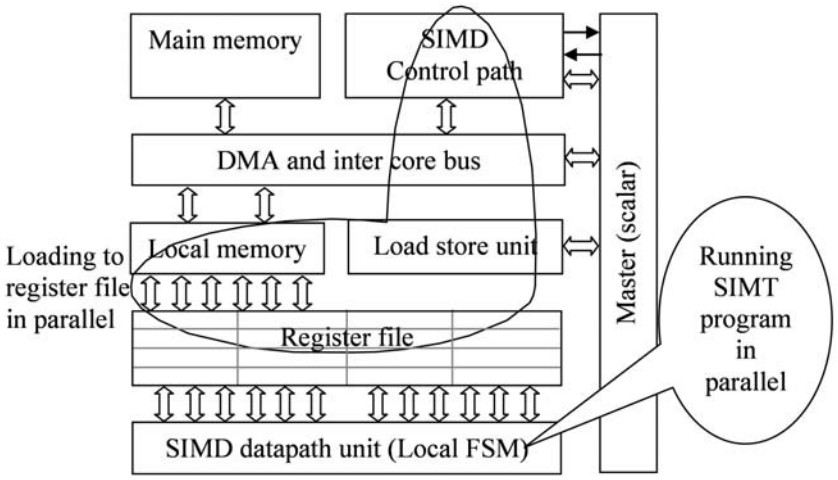


Figure 8. The second step to hide the cost of data accessing



The third step of data access is to store results and changed data to the main memory. It is also executed in parallel with the task running in the SIMD datapath. The way of parallel execution is similar to that discussed in Figure 6.

3.2. The Memory Hierarchy

The memory hierarchy is specified in Figure 5 and it is based on the OpenCL specification. The register files in the master and in the SIMD processors are the lowest-level computing buffer, and the data width of a register file is the width of a (vector) operand.

The local data memory is the second-level computing buffer. In the master (host) machine, there might be two data memories. In a SIMD machine, there is at least one local vector memory. The vector data width of a vector memory is the width of a vector operand. In our research, we define the operand width of 8×16bits=128bits. A local vector memory consists of 8 physical memory blocks. The data width in each memory block is 16 bits. The clock rate of a local (vector) memory is the machine clock rate. The program memory in a master or in a SIMD machine is on the same level of the local data memory. The master program memory can be a cache. The SIMD program memory is usually a simple single port

SRAM. The top-level computing buffer is the main memory and its clock rate is lower than the machine clock rate.

The physical link between the SIMD vector register file and its local vector memory is the register bus in a SIMD. The physical link between the SIMD local vector memory and the main memory is composed of a ring bus and a DMA handler.

3.3. Local Memory

The SIMD local vector memory is not a cache. It is a group of scratchpad memory blocks in which each memory block is a single port SRAM. A vector memory is illustrated in dashed line in Figure 9. In this example, 8 memory blocks compose a local vector memory.

In this figure, DMA moves data between the main memory and the local vector memory. When loading data to the local vector memory, a word can be stored to one of the eight SRAM blocks. The permutation hardware (the switch) selects one of eight SRAM blocks for storing one of the eight incoming data. The vector address generator tells which SRAM block to select according to the permutation table. The permutation table is prepared for an algorithm (in a SIMD task) by the master machine and sent to the SIMD machine before running the task.

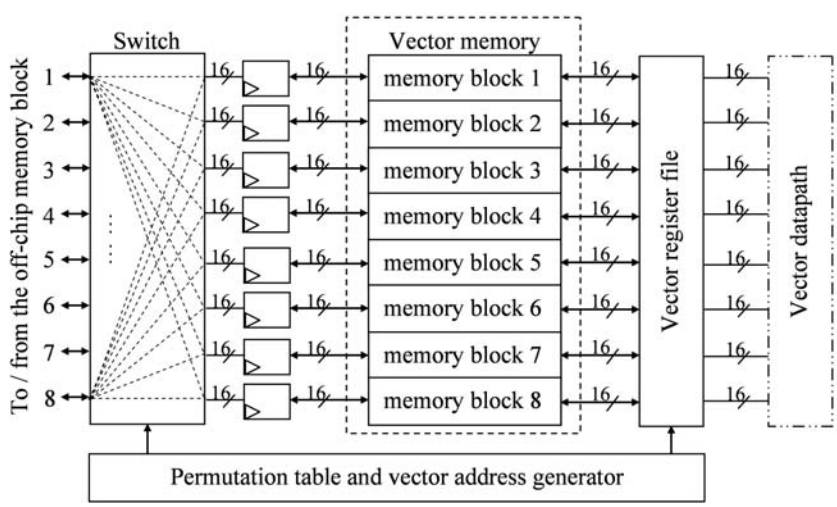
When loading data to the local vector memory according to the permutation table, the data in a vector is allocated in different memory blocks. A conflict-free parallel data access algorithm can therefore support a parallel execution of a DSP algorithm. The way a permutation algorithm is prepared will be discussed later in the paper.

3.4. When to Formulate Vector Data

The definition of vector data is that all data elements of the vector can be available in parallel simultaneously. A vector can be formulated at three times.

- **Case 1:** A vector can be formulated as a vector word in the local vector memory during the DMA data loading.
- **Advantage:** When accessing data between the local vector memory and the vector register file, eight memory blocks in the local vector memory use one address. The address coding size is short so that the silicon cost is lower and SIMD programming is easier. For example, a vector memory

Figure 9. Local vector memory and its peripherals



with 4k vector words needs only 12bits for address coding.

- **Drawback:** One cannot formulate vector data for both row and column computing at the same time. The row-column conversion in a vector register file will consume much time.
- **Case 2:** A vector is formulated in the vector register file while running vector loading from the local vector memory to the vector register file.
- **Advantage:** This is the most flexible solution. Different kinds of vectors for both row and column can be formulated based on one DMA transaction.
- **Drawback:** Addressing coding size is huge. Using the same example, a vector memory with 4k vector words needs 8×12bits = 96 bits for addressing. The silicon cost is higher, and the SIMD programming will be more difficult! However, if the permutation table can be fully used for loading data to the register file, the address coding size can be the size of the permutation table pointer plus the segment address of the data block, which are actually small.
- **Case 3:** A vector can be formulated in the local vector memory during the DMA data loading. Polymorphic addressing architecture will be used to minimize the address coding cost.
- **Advantage:** This is possibly the best solution with low addressing coding cost and the most flexible data access for both rows and columns.
- **Drawback:** New technology, methodology, tool-chain, and verification will be challenges. Extra cost of configuration will also be a problem. If we need to reconfigure the polymorphic architecture for each computing step, there will be no advantage. Fortunately, we do not need to reconfigure the polymorphic architecture while running a vector algorithm.

There are two extreme cases: One is the case for unpredictable data access when permutation cannot be executed during DMA loading. In this case, SIMD load-store instructions will be used to permute data when data is used. Data access induced latency will be the main computing latency. Another extreme case is to load and permute data of one DMA transaction for running multiple algorithms. The permutation algorithm will be very advanced and it will be the ultimate challenge of the future research.

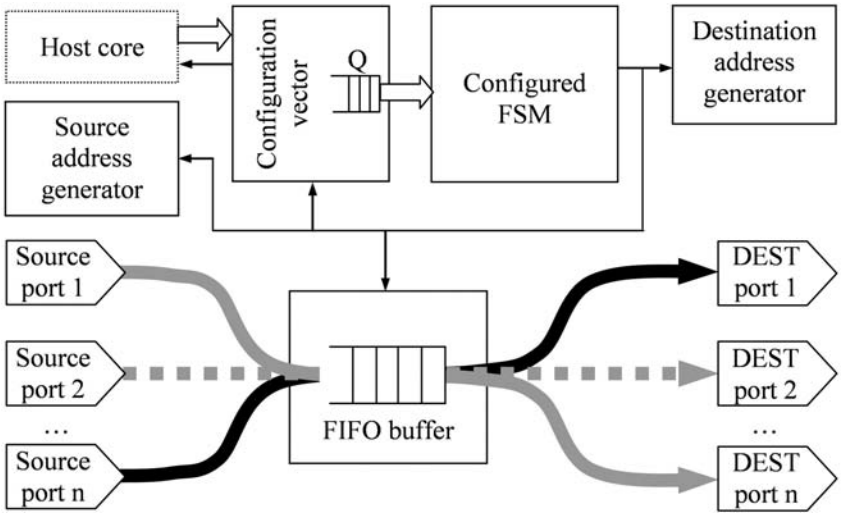
3.5. DMA

The DMA controller is a hardware module connecting two memories by sending data between them without the direct control of a processor. In most cases, one of the two memories is the *main* memory. A DMA task is initialized by the master machine and is terminated by the DMA itself. A DMA termination generates an interrupt to the DMA user.

A DMA task may also be initialized by a SIMD machine if tasks in several SIMD machines need cache coherence. In this case, a SIMD (the dirty data generator) initializes an interrupt to the DMA and to the SIMD machines involved in the cache coherence-based computing. The SIMD that is responsible for the dirty data sends both the dirty data and its address to the DMA. The DMA forwards the dirty data and its address to the SIMD machines interrupted.

The DMA HW block diagram is given in the Figure 10. In this figure, a master initializes a DMA transaction and sends the transaction control table to the DMA handler. At the same time, a permutation table associated with the DMA task is also formulated by the master and sent to the SIMD machine. The DMA hardware will be configured according to the transaction table when the transaction is issued. A source data memory and a destination data memory will be connected to the DMA and its ring bus. Addresses and clocks are supplied to the two memories according to the

Figure 10. DMA implementation of the proposed architecture



configuration. The data transaction will thus be executed. The data FIFO in DMA hardware is a data buffer for data format adaptation between the source memory and the destination memory.

There are opportunities to select, link, and pack multi data blocks located in the main memory into one DMA transaction while loading from the main memory to the local store via DMA. During the DMA transaction, the data blocks distributed in the main memory can be collected and packed into one DMA data packet. In this way, 2D image data can be collected for a 2D image computing.

The first step of an intelligent DMA transaction is to collect data blocks and pack them into one DMA data packet. Address translation will be conducted during the data packing as illustrated in (a). The second step of an intelligent DMA transaction consists of data permutation, data shuffling, and formulating vector words for further conflict free data access. Data permutation can be executed both during loading data from the main memory to the local store and during loading data to a vector register file using polymorphic addressing hardware.

3.6. Main Memory and System Level Bus

There are two main memories connected to two ring busses: one for data and one for storing codes, parameters, and data.

The ring bus is depicted in Figure 12. Figure 12 (a) illustrates the principle and the physical connections of reading from and writing to rings. The address bus is not specified in this figure. Figure 12 (b) exposes the possibility to isolate parts of the ring bus to smaller pieces to support multiple connections. Figure 12 (c) shows actually the structure of the two-ring bus. Two main memories (M1 and M2) are connected to the two-ring bus. A processor core C can be a SIMD machine or a host machine. It can be seen in Figure 12 (c) that each core (for example C1) is connected to two ring busses. Two main memories can therefore be used simultaneously. The bridge between the two ring busses is used to exchange data between the two main memories.

3.7. Vector Data Loading Path

The chain (or the path) of data loading and the related hardware is given in Figure 13.

Figure 11. Two intelligent DMA functions: linking table and permutation

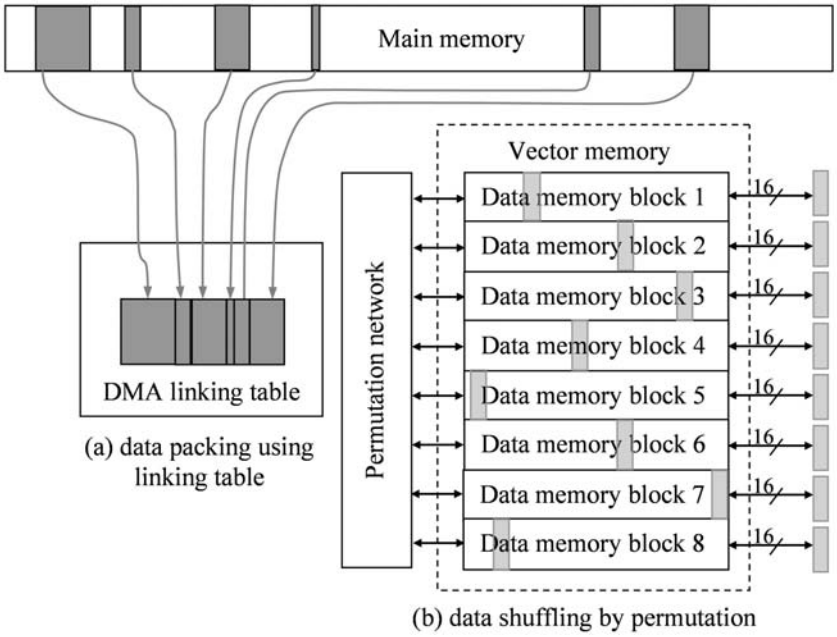
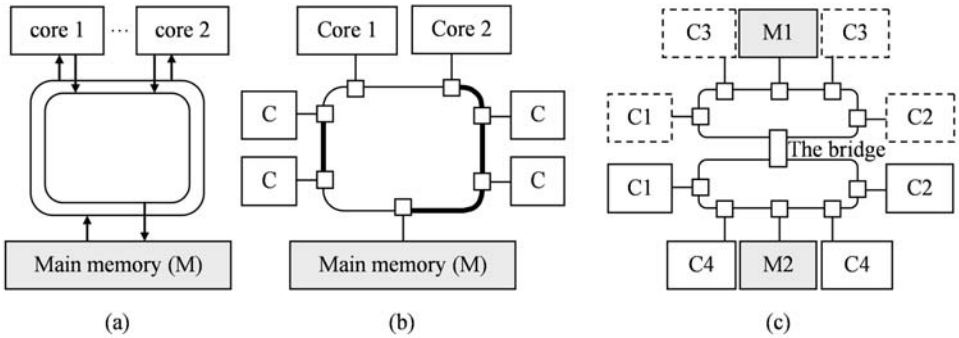


Figure 12. Ring bus and the main memories

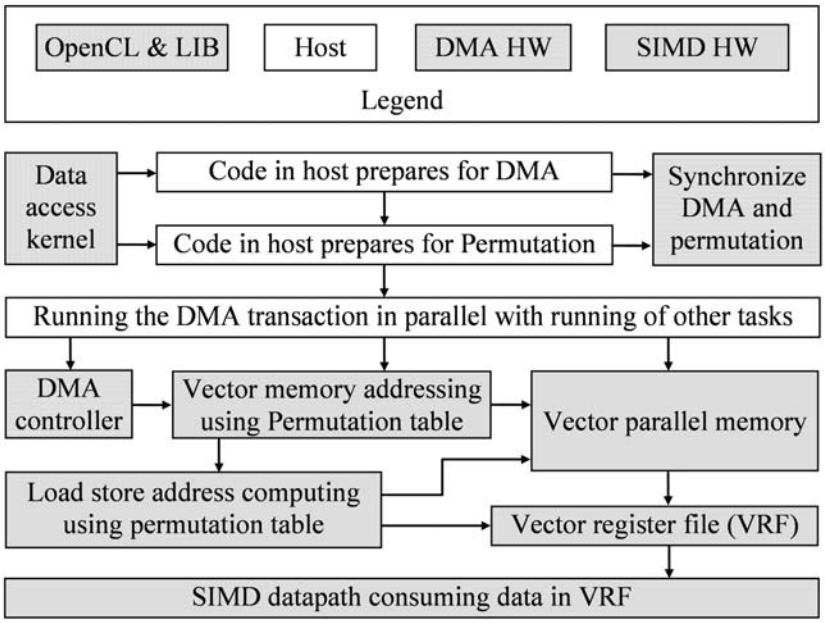


In our research, data accessing for computing an algorithm is separated from its kernel. An independent data access kernel can be selected and matched to an algorithm kernel during the parallel programming and compiling time. The purpose is to run the data access in parallel with the algorithm execution. A data access kernel consists of the DMA linking table (to collect multi-data blocks), the start address of each memory block in the local vector memory in the SIMD, and the permutation table (block name and offset address of each data word).

When the data access is predictable, the host machine can prepare for a DMA transaction by preparing a DMA task table and a permutation table. The permutation table is then sent to the SIMD machine via the control interface between the host and SIMD machines. The DMA task table is also sent to the DMA controller via the control interface to the DMA module.

When the data access path (addressing path or addressing generator) of the SIMD machine is locked by the DMA controller, the DMA starts its data transaction. During the transaction, the

Figure 13. Data loading process



addressing path in the SIMD machine loads a data word to a predefined position in a predefined vector word base on the permutation table. Data from DMA is therefore shuffled and distributed to different memory blocks in the local vector memory.

The formulated vectors in the local vector memory will be further loaded to the vector register file as vector operands for further computing. Finally, vector data in the vector register file is consumed by running SIMD ALU instructions.

When the data access is unpredictable, for example, the data access is completely depended on the current result of the algorithm execution, then the data access can not be executed as the background task in parallel by DMA. More explicitly, data load and store will be executed by the SIMD machine by running data access instructions while running the computing algorithm. In this case, the data access chain will be: (a.) SIMD machine calls for a DMA transaction, (b.) DMA simply sends a data block to SIMD, (c.) SIMD explicitly loads data from the vector memory to the vector register file by running load/store instructions.

3.8. Vector Data Store Chain

The writing chain from SIMD local vector memory to the main memory is illustrated in Figure 14.

The writing process is relatively simple. It uses the same DMA task flow mentioned in Figure 13 and the backward data flow in Figure 14.

3.9. SIMD HW Block Diagram

Finally the block diagram of the SIMD machine is given in Figure 15.

The SIMD datapath consists of a local FSM, which can handle local control and run a simple loop by itself. SIMD addressing path can be executed in parallel with the SIMD datapath and controlled by the local FSM in the datapath. This is the so called SIMT (Single Instruction Multiple Task) concept, see chapter 20 in (Liu, 2008). Under the control of the local FSM, the SIMD addressing path can handle data accesses when the SIMD datapath is running for the current algorithm under SIMT mode.

Figure 14. Data writing process

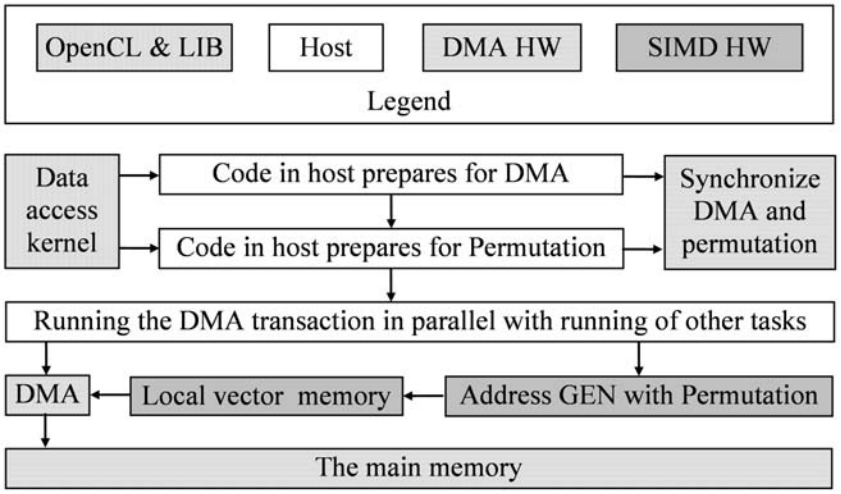
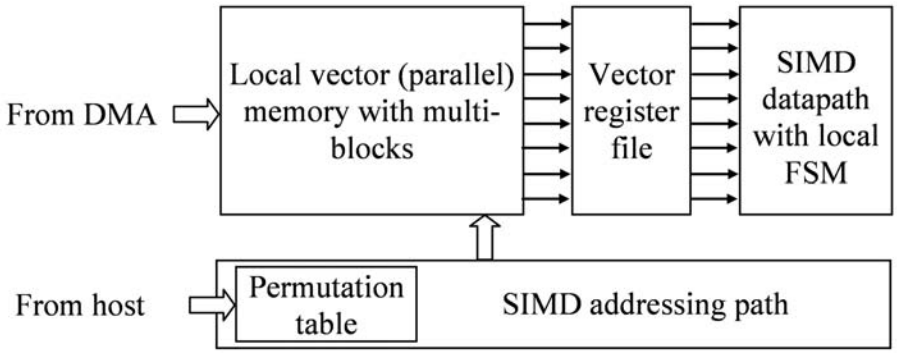


Figure 15. Data loading process in hardware



4. PROGRAMMING FLOW

Based on the *kernel*-based programming flow, kernel codes will be separated from the main program and will be allocated into SIMD. To make program kernels portable, kernels should be configurable, allowing to be used by multiple programs and applications. Kernels can be prepared by kernel developers or can be made by programmers. The style of a kernel based program is:

```
...
 Top level program running in host
   Context for issuing kernel 1 in
host
      Kernel 1 code in SIMD
    Context terminating kernel 1 in
host
  Top level program running in host
    Context for issuing kernel 2 in
host
      Kernel 2 code in SIMD
    Context terminating kernel 2 in
host
 ...
```

The basic flow and tool chain will be based on OpenCL. The main innovation of the programming

flow and the hardware architecture is the separation of parallel data access kernels from algorithm computing kernels. Double performance over hardware cost is expected from the separation.

The basic programming flow is depicted in Figure 16.

4.1. Parallel Task Identification

The parallel programming starts from the program SW specification. The specification could be paper-based document, early behavior C code, or MATLAB code. Parallel tasks can be identified during the code analysis. Code analysis can be conducted by reading the code or by using a code analyzer. A code analyzer and matcher can be used for code identification and matching the identified code to a kernel (Lundgren & Ödlund, 2007).

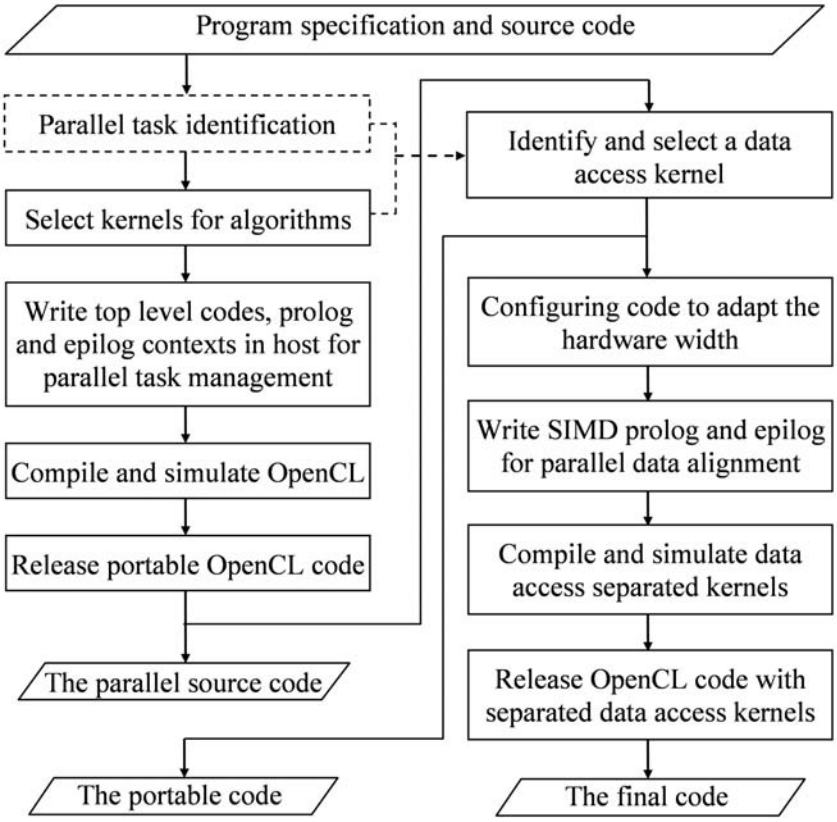
4.2. Algorithm Kernels

The coding template library consists of algorithm kernels. Kernels can be principally adapted from CUDA (NVIDIA, 2009) kernels based on an OpenCL kernel coding template. Kernels from STI CELL, Intel MMX, and MATLAB were also used as references. Kernels can also be made by programmers following the kernel development flow and kernel coding template. One way to identify suitable algorithm kernels for being accelerated was investigated by Skoglund (2007).

4.3. Data Access kernels

BACT is the Behavioral Address Coding Template, which is the original data access kernel for conflict-free parallel data access before hardware

Figure 16. Programming flow



configuration, see chapter 20 in (Liu, D. 2008). The data access kernel (Figure 17) is separated from its original algorithm kernel. A data access kernel carries the data location information in the main memory and in the vector parallel local store memory. It also specifies the way that the data in the main memory is collected and merged into one DMA transaction and the way that the data shall be distributed to each block of the vector parallel memory. A data access kernel may contain following information:

1. The location where is data in the main memory and where the data shall be stored in the vector parallel local store, the segment address of the local vector memory, and all offset addresses of all memory blocks in the vector parallel memory.
2. How data in the main memory is collected and merged into one DMA transaction.
3. How data is to be distributed to each block of the local vector memory so that data vectors can be formed and can be loaded to the vector register file in parallel.

In, data access is classified. BLAS stands for Basic Linear Algebra Subroutines, 1D, 2D, and 3D stand for one dimensional, 2 dimensional, and 3 dimensional, respectively. R2, R3, R4, and R5 stand for radix-2, 3, 4 and 5. I means to implement and N means not needed.

4.4. Hardware Configuration

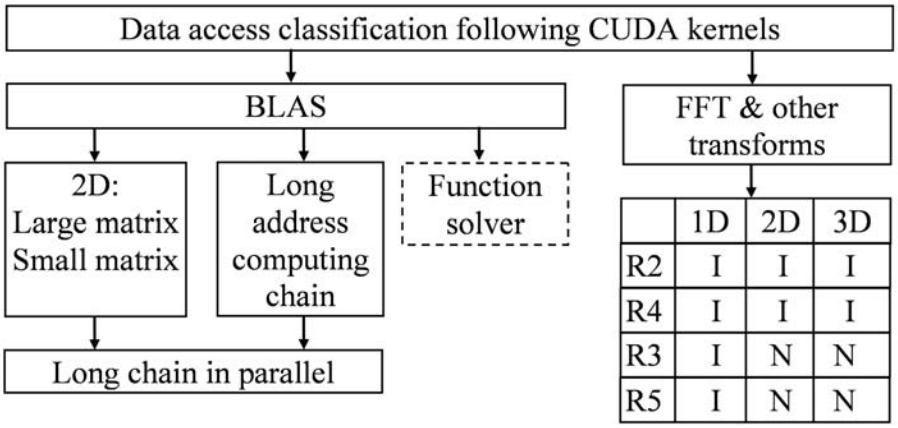
The hardware configuration on kernels means the translation of the kernel code from width-independent to width-dependent code. Here the width is the SIMD datapath width; it denotes the number of issues of a datapath. The width adaptation means actually to add *mode* computing to the original coding kernels. In an original coding kernel, there were only dimensions of the data sizes such as the size of the matrix or the size of the FFT. By adding mode computing, the size of the hardware (used as the *mode* value %) is taken into account for computing and especially for data access computing.

4.5. Prolog and Epilog

There are two levels of prolog and epilog codes. The top-level prolog and epilog codes are put in context for parallelization and for being executed in the host machine (initializing and termination a parallel job).

The low-level prolog and epilog codes are used for data alignment and pipeline installation while running a vector computing subroutine. It includes

Figure 17. Data access kernel classification



	1D	2D	3D
R2	I	I	I
R4	I	I	I
R3	I	N	N
R5	I	N	N

the irregular starting code, the set-up pipelines, the terminating pipelines, and the finishing code of the kernel body. Here, a kernel body stands for the regular part of a kernel. The low-level prolog and epilog will be executed in the SIMD machine.

4.6. Code Release

There will be three releases of codes. The first release will offer the source code of the parallel program, which can be ported to any parallel machine running OpenCL. The second release will be the portable code of our architecture based on the methodology of data access separation from OpenCL algorithm kernels. Finally, the third release is targeted only for a specific design with specific hardware configuration. Different releases offer different levels of code compatibilities.

5. CONFLICT FREE PARALLEL MEMORY ACCESS

Multiple data is needed simultaneously during parallel computing in a SIMD datapath. At least for each algorithm computing, it is essential to access to N data units from the local vector store with N memory blocks simultaneously without access conflict. Here a conflict happens when multiple data cannot be accessed simultaneously because they are stored in one physical memory block.

A data access kernel code supports conflict free parallel memory access when all simultaneously accessed data are allocated in different memory blocks. Conflict-free memory access is more important when vector register file for SIMD is not very large and data access patterns are not simply regular. For transform and matrix algorithms, conflict free parallel data access is even more important.

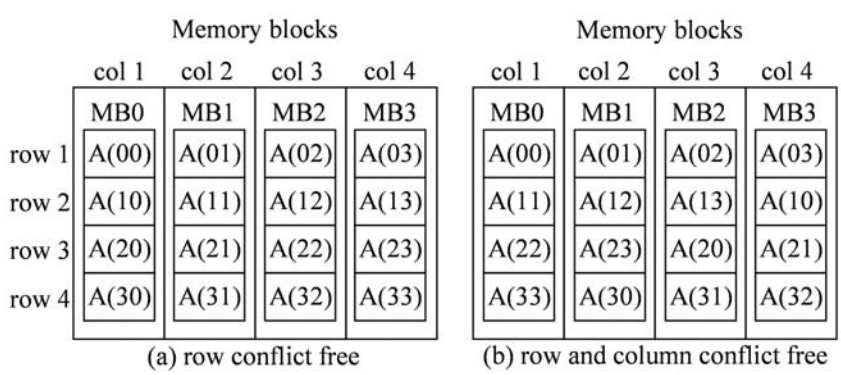
5.1. Data Permutation

As discussed before, data permutation can be applied when loading data from DMA into a vector local store. Data permutation for parallel memory access was discussed by Gössel et al. (1994). Lundgren and Ödlund (2007) investigated the feasibility of exposing memory patterns from code.

A coding template example depicted in this chapter supports row and column accesses in parallel. In Figure 18 (a), only rows can be accessed in parallel while loading data in memory blocks MB0 to MB3, if there is no permutation. By conducting permutation during loading data to MB0….MB3 in local vector memory in Figure 18 (b), both row and column can be accessed in parallel.

A BACT is specified for modeling a parallel addressing to adapt a parallel algorithm. The BACT models the address permutation while loading data to memory blocks. The BACT is

Figure 18. A data permutation example



coded as a kind of conflict-free memory access by specifying relative positions of each access. However, it is not yet adapted to hardware.

A BACT can be configured and adapted to target hardware, specified by the number of parallel memory blocks and the size of each block. After hardware adaptation, a BACT becomes a coding template of a physical vector memory, namely a Parallel Memory Coding Template (PMCT). Finally, a PMCT can be used for a specific algorithm with a specific physical vector memory address. An example of a BACT corresponding to Figure 18 (b) is:

```
int array A[N][N]
  /* i denotes a row and j denotes a
column */,
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
  {
    Row address = i;
    Column address = (i + j) % N;
  }
```

When it is adapted to a vector memory with four parallel memory blocks, the BACT is converted to PMCT.

```
int array A[N][N] /*i denotes a row
and j denotes a column*/,
For (i = 0; i < 4; i++)
  For (j = 0; j < 4; j++)
  {
    RA = I;              /* the row
address */
    CA = (i + j) mod 4;  /* the col-
umn address */
  }
```

Finally, the physical addresses as the permutation table for an algorithm running on hardware with the 4-block vector memory are:

```
int array A[N][N] /*i denotes a row
and j denotes a column*/,
int Segment /*starting address to
load the array*/
For (i = 0; i < 4; i++)
  For (j = 0; j < 4; j++)
  {
    CA= (i + j) mod 4;
    address = segment + 4*I + CA;/*
the final physical address */
  }
```

6. BENCHMARKING ON KERNEL BASED PARALLEL PROGRAMS

An 8-way conventional SIMD processor is used as the reference to compare the performance of our ePUMA, an 8-way SIMD with SIMT (Single Instruction Multi Task) architecture. As a measurement of efficiency we will use the ratio

$$R = \frac{total_cycles}{arithmetic_instructions}.$$

6.1. Matrix Multiplication

The number of arithmetic instructions used for a 64 × 64 matrix multiplication is 36864. This number is the same for all architectures. The results when using one 8-way SIMD can be seen in Table 1. We see that our ePUMA SIMT architecture is not that far from the theoretical limit. When using

Table 1. Matrix multiplication results

Architecture	Conventional SIMD	ePUMA	ePUMA – Overlapped DMA
Total cycles	275342	44043	42531
R	7.469	1.195	1.154
Speedup	1.00	6.25	6.47

interleaved DMA, we achieved a speed increase of 6.47 compared to the conventional SIMD architecture.

Here the overlapped DMA stands for the running of a DMA transaction overlapped with the SIMD computing. The results for using multiple processors can be seen in Table 2. Benchmarking of 64×64 matrix multiplication further on three machines, host with 2×8way-SIMD, host with 4×8way-SIMD, and host with 8×8way-SIMD, is shown in. We see that our architecture scales well, though it is not yet linear scaling. The total speedup compared to the conventional SIMD then grows to 46.4.

Examining the benefit of using our permutation tables, we look at the relative added cost of performing a parallel transpose on the matrix that will be accessed in column-major order. We see in that doing this adds about 11-12% to our computation time.

Table 2. Scalability

Processors	1	2	4	8
Speedup	1.000	1.973	3.832	7.176

Table 3. Using permutation tables vs parallel transpose

Processors	1	2	4	8
Permutation tables	1.0000	1.0000	1.0000	1.0000
Parallel transpose	1.1147	1.1148	1.1152	1.1164

Table 4. LR factorization results

Architecture	Conventional SIMD	ePUMA
Total cycles	82166	29435
R	4.656	1.668
Speedup	1.00	2.79

6.2. LR-factorization

We also examined how well 64×64 LR-factorization performed on our architecture. The number of arithmetic instructions is 17647. We can see the results in. While we do not get the same increased performance as for the matrix multiplication, a 2.79 times speedup is significant. Since LR-factorization traditionally scales well we believe that it will have similar speedups as the matrix multiplication when using multiple processors.

CONCLUSION

In this paper, we introduced our ePUMA architecture and programming methodology for parallel DSP computing. ePUMA is a host-SIMD based architecture with SIMT acceleration supporting separated data access computing kernels. Benchmarking shows exciting results for the ePUMA architecture. For example, by using our 8-way ePUMA SIMD machine, we gain a speedup of 6.47 for 64×64 matrix multiplication and 2.79 for 64×64 matrix LR-factorization comparing to conventional 8-way SIMD architecture. The architecture of one master and eight SIMD cores offers a speedup of 46.4 for matrix multiplication, demonstrating that ePUMA is a scalable architecture.

ACKNOWLEDGMENT

The project is supported by SSF, Swedish foundation for strategic research. Authors wish to thank the following people for useful discussions: Anders Wass and Peter Olanders of Ericsson, Christoph Kessler, Ingemar Ragnemalm, and Ali Saeed of Linköping University.

REFERENCES

Flachs, B., Asano, S., Dhong, S. H., Hofstee, H. P., Gervais, G., & Kim, R. (2006). The microarchitecture of the synergistic processor for a cell processor. *IEEE Journal of Solid State Circuits*, *41*(1), 63–70. doi:10.1109/JSSC.2005.859332

Gössel, M., Rebel, B., & Creutzburg, R. (1994). *Memory architecture and parallel access*. New York: Elsevier Science.

Hennessy, J. H., & Patterson, D. A. (2003). *Computer architecture - a quantitative approach* (3rd ed.). Morgan Kaufmann.

Khailany, B. K., Williams, T., Lin, J., Long, E. P., Rygh, M., & Tovey, D. W. (2008). A programmable 512 GOPS stream processor for signal, image, and video processing. *IEEE Journal of Solid-State Circuits*, *43*(1), 202–213. doi:10.1109/JSSC.2007.909331

Khronos. (2008). *OpenCL specification V1.0*. Beaverton, OR: Author.

Liu, D. (2008). *Embedded DSP processor design - application specific instruction set processors*. New York: Elsevier.

Lundgren, B., & Ödlund, A. (2007). *Expose of patterns in parallel memory access*. Unpublished master's thesis, Linköping University, Sweden.

Nilsson, A., Tell, E., & Liu, D. (2008). An 11mm^2 70mW fully programmable baseband processor for mobile WiMAX and DVB-T/H in 0.12µm CMOS. In *Proceedings of the IEEE International Solid State Circuits Conference* (pp. 266-268).

NVIDIA. (2009). *NVIDIA CUDA reference manual version 2.2*. Santa Clara, CA: Author.

Skoglund, B. (2007). *Code profiling as a design tool for application specific instruction sets*. Unpublished master's thesis, Linköping University, Sweden.

Tretter, S. A. (2003). *Communication system design using DSP algorithms: With laboratory experiments for the TMS320C6701 and TMS320C6711*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 1, edited by Seppo Virtanen, pp. 64-85, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 3
Towards Sustainable Development of Energy-Aware Systems

Luigia Petre
Åbo Akademi University, Finland

Kaisa Sere
Åbo Akademi University, Finland

ABSTRACT

Smart devices have pervaded our society and encouraged lifestyles that depend on them. One of the fundamental requirements for a successful dependency is that the general public be aware of the energy limitations of these devices and to stay in control of energy consumption. In this paper, the authors propose a formal specification method that takes energy into account. They propose two development approaches that can use these specifications to develop energy-aware systems in a sustainable manner.

INTRODUCTION

Smart devices communicating with each other are pervading our society. Their applicability spans a wide range of domains, from wireless sensors deployed in space or military zones to cameras, cars, and refrigerators. Transportation, communication, medicine, commerce, and banking are all areas increasingly embedding IT (Royal Academy, 2009), and gradually promoting the use of smart, typically small, IT-based devices. We are becoming dependent on a very flexible living manner in which plane tickets can be ordered from a mobile phone that can also perform Internet banking and in the near future will act as an electronic wallet. The importance and versatility of such smart devices require them to work continuously and at full capacity. These characteristics are usually referred by Laprie, Avizienis, and Kopetz (1992) as *reliability* - the property of a system to continuously provide service and *availability* - the readiness for correct service of the system.

DOI: 10.4018/978-1-4666-0912-9.ch003

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

One often overlooked factor required for reliable and available systems is that of *power* or *energy*. This factor could be overlooked in the past where most communicating devices were stationary and electrically plugged machines. However, it cannot be ignored in the current networking environment that becomes context- and location-aware, (Yan & Sere, 2004; Roman, Julien, & Payton, 2004), hence it involves mobile, unplugged devices functioning on batteries. These mobile devices have numerous user-friendly features (Michael, 2005) such as high resolution display, radio, camera, TV, GPS, etc. At the same time, a device should be as small as possible, leaving not much space for the battery in spite of the increased energy need of its various features.

We can regard energy in relation to smart devices in a multitude of ways. One aspect is that of being provided by a battery, hence the engineering challenge of fitting a (rechargeable) battery supporting the functionality of the device and thus enabling mobility. Based on this aspect, users can question how long can a certain battery-enabled device be used and how flexible this usage is. For instance, if an individual travels to a remote location and there takes a very special trip, this person may need for the trip only the camera and the phone functionality of a smart device. The rest of the features - software and hardware - should be turned off to save the battery. The current smart devices are sometimes so complex that they do not offer such simple choices to the user. Instead, they may be running some anti-virus software periodically, checking for updates, or loading and keeping a multimedia player ready to be used without a clear choice for the user on whether these features are needed or not. Yet another aspect concerning energy in relation to smart devices is that of deploying or developing products that consume no more than a certain amount of energy per time unit. Recharging a battery is another fundamental issue to consider as well. A clear implication of these energy-related aspects is that we need to take energy into consideration when developing and deploying smart devices: they need to be *energy-aware*.

We can view energy as a non-functional property of a system, and thus not needing to be addressed in the application developer's requirements. Instead, energy could be taken care of by a *network manager*, at a more specialized level often referred to as the *middleware level* (Petre, 2008). Energy consumption would thus be handled together with other properties concerning the proper functioning of smart devices, such as system security or resource availability. However, if we need to specify constraints on a system's overall energy consumption, we have to be able to model it early in the lifecycle, namely starting from the user requirements. This amounts to expressing both functional and non-functional properties at the same abstraction level, thus breaking some of the traditional encapsulation of the hardware and middleware specification. We claim that this is a necessary modeling choice for energy-awareness as well as beneficial for the efficiency of the developed systems and adhere to it in this paper. Energy becomes a feature of every single module, software or hardware, thus enabling a wide range of properties to be proved for the developed device.

Another key dimension in computing today is that of *sustainability*. In a strict sense, this refers to using resources without depleting them, thus falling along the power consumption and efficiency lines of research. However, as our society becomes more computing-dependent, we have become proficient in developing most various types and forms of software-based systems. The time is therefore ripe for trying to develop software-based systems in a *sustainable manner*, at various levels: less human resources and less time involved in planning, developing, and testing; less redundant and useless software; no incorrect development of requirements into software. These objectives can be approached by investing in the devise of more sophisticated methods and tools that take care of avoiding the undesired side-effects of developing software.

The famous line 14 of the driverless Parisian metro, developed in 1998 using the B-method (Abrial, 1996) is the first notable example of a sustainable development, reviewed by Lecompte (2008). The method used by Siemens for developing the software controlling the line 14 train ensured its correctness in a mathematical manner that effectively eliminated the unit testing from the software lifecycle. No human resources are now needed to operate the trains and in addition the trains are faster, hence fewer are needed in total. Sustainability in the human resource number, train number, and software lifecycle were achieved using a *formal method*, i.e., a fault-avoidance technique used for constructing trustworthy systems. Modeling based on a formal method means using mathematics in the specification and development of the systems, e.g., to make *sure* that the final result correctly implements the requirements of the system. In this paper we employ *action systems* (Back & Kurki-Suonio, 1983), a state-based formal method (related to the B-method) for distributed systems in order to model energy and sustainable development of energy-aware systems. In particular, we use a conservative extension of action systems named MIDAS, developed by Petre, Sere, and Waldén (2006) for modeling energy. Using this formalism, we can stepwise develop a system as well as prove various properties about it.

We proceed as follows. In Section 2 we present our energy-aware modeling language, in Section 3 we describe a small case study using the language, in Section 4 we outline two techniques suited for energy-aware development, in Section 5 we discuss related work and implications of the model, and in Section 6 we present some conclusions.

ENERGY-AWARE RESOURCES

In this section we present our approach to modeling energy-awareness, based on the MIDAS language - MIDdleware based on Action Systems. MIDAS is developed for handling resource functioning in networked applications; the central concept of MIDAS is that of a *resource*. Petre, Sere, and Waldén (2006) and Petre (2008) define a resource as a unit that has a location and other specific properties. While we still adhere to this approach, there is an important distinction we make here regarding the location of a resource. Due to their different perception and use of energy, software and hardware are distinguishable and have a different approach to location. Hardware could well be considered as the location of software; in this case, the (software) location domain is discrete and similar to the approach of Petre, Sere, and Waldén (2006) and Petre (2008). However, the location of hardware can be the whole tri-dimensional, continuous space. In this paper we adhere to the latter location approach and define *Space* to be the location domain for both software and hardware. This simplifies having two location domains to work with and, quite importantly, avoids modeling the identity of the hardware units as locations for the software units. While identity modeling has been done before in the action systems framework by Bonsague, Kok, and Sere (1998), it requires an elaborate construction that is not instrumental here.

In the following we model several resource types: *data*, *code*, and a combination of data and code called *computation unit*. We then distinguish between software and hardware resources.

SIMPLE RESOURCES

Data resources Let *Var* be a finite set of *variable names*. Data resource units (*variables*) are specified as quadruples (v, loc, Val, val) where $v \in Var$ denotes the name of the variable, $loc \subseteq Space \cup \{\lambda\}$, $\lambda \notin Space$ is either its location in the domain *Space* or a default location $\{\lambda\}$, *Val* is a nonempty set of *values* denoting the variable's type, and $val \in Val$ is the current value of the variable. We express the location of a variable named v by the expression $v.loc$ and the names of the variables located at a location α by the expression $\alpha.var$. The value

of a variable named v is given by the expression $v.val$. When $|v.loc|>1$, we say that the variable is *replicated*. The condition $|v.loc|>1$ implicitly excludes the case $v.loc=\{\lambda\}$. We assume that the type of a variable is unchangeable.

We consider here that data resources do not consume any energy, nor does storing them. Writing and reading data consumes energy like any code does.

Code resources Let *Act* be a finite set of *action names*, distinct from *Var*. Code resource units (called *actions*) are specified as quadruples (a,loc,A,rE) where $a \in Act$ is an informal, optional name for the action, $loc \subseteq Space \cup \{\lambda\}$, $\lambda \notin Space$ is either its location in the domain *Space* or a default location $\{\lambda\}$, A is its body, i.e., a statement that can model evaluation and updates of the variables, and rE models the energy requested by action a to execute. We denote the location of an action a by $a.loc$, the actions at location α by $\alpha.action$, and the energy requested by the action a to execute by $a.rE$. When $|a.loc|>1$, we say that the action is *replicated*. The condition $|a.loc|>1$ implicitly excludes the case $a.loc=\{\lambda\}$.

The name of an action is unchangeable. The body A of an action named a can model deadlocking and stuttering code, assignments, guarded code, sequential, conditional, and non-deterministic composition of a finite number of actions. Semantically, actions are defined using *weakest precondition (wp) predicate transformers* (Back & Sere, 1996). An action A can be executed only if a specific boolean condition $g(A)$ named *guard* holds; in this case we say that the action is *enabled*. Guards are formally defined using *wp* predicate transformers (Back & Sere, 1996).

Resource accessibility As the location of resources is specified, we allow for the situation in which resources that need each other for execution are not necessarily co-located, hence modeling their execution is not realistic. To avoid this, we define the set of accessible locations for each action a, denoted $cell(a)$. An action can execute only if all its required data is located in its cell; this condition is called *location guard*, denoted $lg(a)$. We define a stronger enabledness condition named $gd(a)$ of each action a as the conjunction of its guard $g(A)$ and its location guard $lg(a)$:

$$gd(a) = g(A) \wedge lg(a)$$

Computation Unit Resources

The state of a system is described using variables; more precisely, the current state of a system is given by the current values of the state variables. A computation unit resource consists of a global description of some behavior over the state variables in the location domain *Space* and is specified as

$$\mathsf{A} = |[\ \mathbf{exp}\, y; \mathbf{var}\, x; \mathbf{imp}\, z; \mathbf{do}\ [\,]_{i \in I}\ (a_i, A_i, loc_i, rE_i)\ \mathbf{od}\ ]| \quad (1)$$

The first three sections are for variable declaration and initialization, while the last describes the computation involved in A, when I is finite. We assume that x, y and z are sets of variables whose names are pairwise disjoint, i.e., the name of a variable is unique in a computation unit.

The **exp** section describes the finite set of *exported* variables y of A. These variables are defined by and can be used within A, as well as within other computation units that import them. As the exported variables can be imported by other units, their names are unchangeable. The **var** section describes the finite set of *local* variables x of A. These variables are defined by and can be used only within A. As the variables are local to A, their names can be changed. This change has to respect the requirement of unique names for variables in a computation unit and has to be propagated in all the action bodies that use the respective local variables. The **imp** section describes the finite set of *imported* variables z. These variables are specified by name and possibly desired locations of import or import type. They are used in A and are declared as exported in other computation units. As the imported variables refer to exported

variables of other computation unit, their names are unchangeable.

Execution model The imported and the exported variables form the *global* variables of A, used for communication between computation units. The **do... od** section describes the computation involved in A, modeled by a non-deterministic choice ([]) between actions a_i at certain locations, operating on local and global variables. First, the local and exported variables whose values form the state of A are initialized. Then, repeatedly, enabled actions at various locations in $\{a_i.loc\}_{i\in I}$ are non-deterministically chosen and executed, typically updating the state of A. Actions that do not access each other's variables and are enabled at the same time can be executed in parallel. This is possible because their sequential execution in any order has the same result and the actions are taken to be atomic. Atomicity means that, if an enabled action is chosen for execution, then it is executed to completion without any interference from the other actions of the system. The computation terminates if no action is enabled, otherwise it continues infinitely.

Energy-aware computation unit A computation unit A as in (1) contains a finite set of actions $\{a_i\}_{i\in I}$ executing over the state of A non-deterministically. We define the required energy of A, denoted A.*rE*, as the sum of the requested energy amounts of its enabled actions:

$$\mathsf{A}.rE = \Sigma_{i\in enI}\, a_i.rE,\ enI = \{i \in I \mid gd(a_i)\} \qquad (2)$$

The A.*rE* amount models the actual energy need of that computation unit, at the moment of evaluation.

Replicated resources Replicated resources can be created by using a specific action *copy*(*R*,*locations*), where the resource *R* can be either data or code. If a computation unit needs to replicate itself, then it can do so by using an action *copy*(*locations*) that replicates the whole computation unit to *locations*. A resource cannot be copied at a location where there already is a copy of *R*, a condition modeled by the location guard of the *copy* action (Petre, Sere, & Waldén, 2006). When updating replicated *data*, all the copies need to be updated simultaneously. This requires that all the copies of that data need to be located in the cell of the updating action and that they are all updated in the updating action body. Since actions are atomic, this ensures that no other actions can access the copies unless the updating action has been executed.

Homonym resources Networks can contain *data* bearing the same name even if they are not representing the same information. Such variables are called *homonym* and they differ from the replicated variables in that they are updated independently of each other. Computation units containing exported homonym variables cannot communicate with each other due to semantic reasons.

Resource mobility All types of resources can move through the network. This is described by the action

move(*R*, *source*, *destination*)

that checks in the location guard that there is a copy of *R* at *source* and there is no copy of *R* at *destination*. The resource *R* can be either data or code. If a computation unit needs to move as a whole, then it can do so by using an action

move(*source*, *destination*)

that moves the whole computation unit from *source* to *destination*. The location guard of the *move* action ensures also that *source* and *destination* are in the cell of *move* (Petre, Sere, & Waldén, 2006). If the resource to be moved is an action, then consequently the cell of this action will modify its contents due to the movement.

Resources and locations If all the computation unit components have the same location, then this location is propagated to the computation unit. In case the locations differ, the computation unit gets the default location $\{\lambda\}$. Thus, assume for *A* below we have $y_m.loc=\Phi_m$, $x_j.loc=\Psi_j$, and $a_i.loc=\Delta_i$:

$$\begin{aligned}\mathtt{A} = |\,[&\mathbf{exp}\{\ldots,(y_m,\Phi_m,\ldots),\ldots\};\\ &\mathbf{var}\{\ldots,(x_j,\Psi_j,\ldots),\ldots\};\\ &\mathbf{imp}\ z;\\ &\mathbf{do}[]_{i\in I}\left(a_i,\ \ldots,\ \Delta_i,\ \ldots\right)\mathbf{od}\\ &]\,|\end{aligned}$$

Then,

$$\begin{cases}\mathtt{A}.loc = \ \Phi_m, \Phi_m = \Psi_j = \Delta_i, \forall m, j, i\\ \qquad \{\lambda\}, \text{otherwise}\end{cases} \tag{3}$$

Here $\{\lambda\}$, $\lambda \notin Space$ stands for the default location of the computation unit A. The reverse relation of a computation unit propagating its location to its components holds in the following form: If A.*loc* ⊆ *Space*, then all the components of A have the same location A.*loc*. Yet, if A.*loc* = $\{\lambda\}$ then we cannot say anything about the locations of the components.

Different types of computation unit resources For code to get executed, it needs a hardware host to run upon. In its turn, a hardware host needs power supply to work, from a battery or an electric socket. Therefore, energy awareness makes us distinguish between the following computation units: *software* (or code), *hardware*, and *electrical socket*. The software and hardware computation units are the energy *consumers*, while the electrical sockets are the energy *suppliers*. A software unit differs from a hardware unit in that the latter has typically a battery and a connection to an electrical socket. In the following we introduce models for the electrical socket and hardware host computation units. The software computation unit is more general and has the form described in (1).

An *electrical socket* ES is defined as a computation unit of the form

ES =|[**exp** (*E*,*loc*,*Boolean*,*T*)]|

The fact that the socket is free to charge a device is modeled when the variable *E* evaluates to *T*. If the socket is already charging some device, then the variable *E* evaluates to *F*. We note that *E* has homonyms at various locations in *Space*, but it is a non-replicated variable, hence |*E.loc*|=1. This is due to the independent updates of the possible *E* homonyms.

A *hardware host* HW is defined as a computation unit having an exported variable *B* that models its battery level. Concretely, *B.val* models the amount of energy that hardware host has left. In addition, a hardware host has a connection modeled by an imported variable *E* to an electrical socket. The fact that the hardware host is connected or disconnected to an electrical socket is modeled by an exported boolean variable named *charge*. Moreover, HW has typically some actions act_i performing the required tasks of that unit. A device can connect to electricity or disconnect from electricity at any time. We model this with the action named *plug* below

plug:: *plug.loc*=*E.loc*→ *charge.val*:= ¬*charge.val*; *E.val*:= ¬*E.val*

We use the notation a_i::A_i instead of (a_i, A_i, loc_i, rE_i) when writing out computation units. The body of this action models the flip between plugging and unplugging the device from the electrical socket as well as the flip between the socket being free or occupied. The guard condition *plug.loc*=*E.loc* models that a hardware device can be (un)plugged only when it is co-located with an electrical socket. In this case, the location of the hardware host is given by *plug.loc*, which is one of its actions.

If the hardware host is charging, then the battery level should increase until it reaches its limit *BMAX*. While the battery charges, other actions act_i of that hardware resource should be able to execute; for this, charging the battery and executing other actions should be interleaved. As any action needs energy to execute, instead of executing the action a_i:: A_i we replace it with a_i below:

$$
\begin{aligned}
&a_i :: \mathit{if}\ \text{charge.val}\ \mathit{then}\ A_i\ \text{else}\\
&\qquad gd(a_i) \wedge B.val > threshold + a_i.rE\\
&\quad \rightarrow B.val := B.val - a_i.rE; A_i\\
&\text{fi}
\end{aligned}
\tag{4}
$$

The action a_i models that, if the hardware device is charging, then the executing code consumes energy from the electrical socket directly. If, however, the hardware host is not charging, then the power requested to execute the code is taken from the battery of the hardware host.

We model the charging process in a discrete manner here for simplicity. Namely, we denote by E_{unit} the smallest energy amount that the battery can be charged with at a time. The action *charging*, modeling the charging process is modeled as

$$Charging :: \text{charge}.val \wedge B.val < BMAX \rightarrow B.val := B.val + E_{unit}$$

We note that an action system-based model is able to handle continuous processes (Back, Petre, and Porres, 2001) as well.

Summing up, a hardware host is modeled as the following computation unit:

$$
\begin{aligned}
\texttt{HW} = |[\ &\mathbf{exp}\left\{\left(B,\ ,\ ,0\right),\ (charge,\ ,\ ,F)\right\};\\
&\mathbf{imp}\,E;\\
&\mathbf{do}\,charging\ ::\\
&\quad charge.val\ /\backslash\ B.val\ <\ BMAX\ \rightarrow\\
&\qquad B.val\ :=\ B.val\ +\ E_{unit}\\
&\ []plug ::\ plug.loc =\ E.loc\ \rightarrow\\
&\quad charge.val\ := \neg charge.val;\\
&\quad E.val\ := \neg E.val\\
&\ []skip ::\ skip\\
&\ []a_i :: \text{if}\,charge.val\,\text{then}\,A_i\,\text{else}\\
&\quad gd\left(a_i\right)\ /\backslash\ B.val >\ threshold + a_i.rE\\
&\quad \rightarrow B.val\ :=\ B.val\ -\ a_i.r\text{E};A_i\\
&\quad \text{fi}\\
&\mathbf{od}\ ]|
\end{aligned}
\tag{5}
$$

where $threshold \geq skip.rE + plug.rE + charging.rE$ and we assume $a_i.rE<BMAX$. The action *skip* keeps HW running since it is always enabled. There are two aspects here that we need to observe. First, as the action *skip* is always enabled and the action systems execution model is non-deterministic, it can be the case that only *skip* is chosen for execution in the detriment of the other HW actions. This is perfectly acceptable in a high-level model, but needs to be refined in the concrete implementation where a certain fairness model will be introduced (Back & Kurki-Suonio, 1983). Second, we model smart devices that work continuously unless switched off and, without any enabled actions, the action system model of HW would terminate. Obviously, when this model is refined towards implementation, the action *skip* will be replaced with a concrete action needed for keeping the device on. More importantly, this action will have a stronger guard condition that will model the fact that a mobile device can work continuously as long as it has enough energy to consume in order to do so. Keeping the device working consumes some energy modeled as $skip.rE$; hence, *skip* cannot execute unless $charge.val \vee (B.val \geq skip.rE)$. The actions *charging* and *plug* will have similar guard conditions.

Parallel composition In order to model complex systems we need to compose computation units. This operation is described using the parallel composition operator.

Consider two computation units A and B as given below:

$$\mathsf{A} = |[\ \mathbf{exp}\,y;\ \mathbf{var}\,x;\ \mathbf{imp}\,z;\ \mathbf{do}\ []_{i\in I} a_i :: A_i\ \mathbf{od}\ ]|$$

$$\mathsf{B} = |[\ \mathbf{exp}\,v;\ \mathbf{var}\,w;\ \mathbf{imp}\,t;\ \mathbf{do}\ []_{j\in J} a_j :: A_j\ \mathbf{od}\ ]|$$

We *assume* that the local variables of A and B have distinct names: $x \cap w = \emptyset$. If this is not the case, we can always rename a local variable to meet this requirement. The exported variables declared in A and B are *required* to have distinct

names: $y \cap v = \emptyset$. The *parallel composition* A || B of A and B has the following form:

$$\mathsf{A} \parallel \mathsf{B} = |[\ \mathbf{exp}u;\ \mathbf{var}s;\ \mathbf{imp}r;\ \mathbf{do}\ [\,]_{i \in I \cup J} a_i :: A_i \mathbf{od}\ ]|$$

where $u = y \cup v$, $s = x \cup w$ and $r = (z \cup t) \setminus u$. The initial values and locations of the variables, as well as the actions in A || B consist of the initial values, locations, and the actions of the original systems, respectively. The well-definedness of A || B is ensured by the fact that all its variables have unique names. Thus, the exported and local variables of A and of B have distinct names, and moreover, the local variables of A can always be renamed in order not to be homonym with the exported variables of B (and vice versa). The binary parallel composition operator '||' is associative and commutative, and thus extends naturally to the parallel composition of a finite set of systems.

The parallel composition A || B defines a more complex computation unit that either A or B. The location of A || B is computer according to (3), while the requested energy of A || B is computer according to (2), where $enI = enI_1 \cup enI_2$. It is evident that $(\mathsf{A} \| \mathsf{B}).rE = \mathsf{A}.rE + \mathsf{B}.rE$ and this property extends to a finite number of units composing in parallel: $\mathsf{A}.rE = \mathsf{A}_1.rE + \ldots + \mathsf{A}_n.rE$. Parallel composition combines the variables of the component systems as well as their evolution. Since the actions in the parallel composition are the combined actions of the two systems, changes will occur differently. Thus, the behavior of A in the composition A || B is determined by its actions and by those actions of B referring to variables from A. Instead, the behavior of the system A when considered in isolation is only determined by its actions.

Connecting the resources In order to model the charging process, we compose the electrical socket ES and the hardware device HW in parallel and enable the communication among them: ES || HW.

Data and code cannot exist unless stored on a hardware host such as a memory stick, an I-phone, a laptop, etc. We assumed that the hardware does not consume any (noticeable) amount of energy for *storing* data or code. However, the hardware needs energy, from a battery or an electrical socket, in order to *execute* code. As we model code by actions, we therefore need to strengthen the location guards of code-related actions of a computing unit A in order to prevent executing them unless there is sufficient power for doing it. We also need to model that executing code consumes energy. We do this in two steps. First, the computation unit A that contains the actions a_i can compose in parallel with a hardware host HW: A || HW. The energy-awareness is modeled by using the global variables B and *charge* for every action a_i as shown in (4). Second, for any action a_i we strengthen $gd(a_i)$ with the conjunction $\{B, charge\} \subseteq (a_i.loc).var$. The action a_i now has the form:

$$\begin{aligned}
&\{B,\ charge\} \subseteq (a_i.loc).var \rightarrow \\
&\quad \text{if}\, charge.val\, \text{then}\, A_i \\
&\text{else}\ gd(a_i)\ \ /\backslash\ B.val > \ threshold + a_i.rE \\
&\quad \rightarrow B.val\ := \ B.val\ -\ a_i.r\mathrm{E}; A_i \\
&\quad \text{fi}
\end{aligned}$$

This means that there is a hardware host at the location of a_i with a battery and a charger. Due to the requirement of unique resource names at a given location, this is the only hardware host with the battery level given by $B.val$ and the charging status modeled by $charge.val$. This models that a given code can execute on only one hardware host at a given time.

CASE STUDY

In this section we use the energy-aware MIDAS to capture an example application.

Assume one walks in the Vigeland park in Oslo, well-known for its beautiful statues. While passing close by a restaurant, an SMS is received on

the mobile phone containing the lunch menu and the price of the menu. Not hungry yet, the person heads towards the statues area of the park. When sufficiently close to a statue, some information about that statue can be received on the phone, in the form of a message. The person can choose between an SMS, an audio message, or a video one. The person then checks the phone: is there enough free memory to get the information? is there enough energy in the battery to play the video? If the existing battery level is not sufficient, the person can check back to the restaurant for charging the phone at one of their electrical sockets. Provided one socket is free, one can charge the phone, (maybe also eat,) then get back to the statues and download and play the video, maybe check the audio message and then continue.

We use four components to model this application: the restaurant R*est*, one of its electrical sockets ES, one statue S*tatue*, and the phone P*h*. Their energy-aware MIDAS specifications are shown below:

$$\mathtt{R}est = \Big\|\mathbf{exp}\ \Big\{\big(SMS, foodloc, TEXT, x\big)\Big\}\Big\|$$

$$\mathtt{ES} = \Big\|\mathbf{exp}\ \Big\{\big(E, foodloc, Boolean, F\big)\Big\}\Big\|$$

$$\mathtt{S}tatue = \Big\|\mathbf{exp}\ \Big\{(SMS, foodloc, TEXT, x),\ \big(AMMS,\ statloc,\ MP3,\ z,\ size\big),\ \big(VMMS,\ statloc,\ MP4,\ v,\ size\big)\Big\}\Big\|$$

From the point of view of our application, the restaurant is seen as the provider of an SMS containing the lunch menu and the price of the menu. An electrical socket of the restaurant is modeled as a separate computation unit resource that is co-located with the restaurant.

We have shown in S*tatue* only a piece of code for the statue, which provides information on three media types. Typically, the statue in such an application will host a wireless sensor with hardware and software parts. For the person walking in the park these underlying structures are not of interest, and thus omitted from the specification. For the multimedia messages (audio and video) we also need to check their size, hence the added *size* field of the corresponding variables.

$$
\begin{array}{l}
\mathtt{P}h\ =|\,[\\
\quad \mathbf{exp}\Big\{\big(B,,,\big),\ (charge,,Boolean,F)\Big\}; \\
\quad \mathbf{var}\{\big(freetextmemory,,Boolean,\big), \\
\qquad \big(freeMMmemory,,,\big), \\
\qquad \big(newSMS,,TEXT,\big), \\
\qquad \big(newAMMS,,MP3,,size\big), \\
\qquad (newVMMS,,MP4,,size)\}; \\
\quad \mathbf{imp}\Big\{(E, SMS, AMMS,\ VMMS)\Big\}; \\
\quad \mathbf{do}\, charging :: \\
\qquad charge.val \wedge B.val\ <\ BMAX \\
\qquad \rightarrow\ B.val\ := B.val + E_{unit} \\
\quad [\,]plug ::\ plug.loc = E.loc \\
\qquad \rightarrow\ charge.val := \neg charge.val; \\
\qquad\quad E.val := \neg E.val \\
\quad [\,]skip ::\ skip \\
\\
\quad [\,]act ::\ if\, charge.val\ then\ A\ else \\
\qquad gd\big(a\big) \wedge B.val\ >\ threshold\ +\ a.rE \\
\qquad \rightarrow B.val\ :=\ B.val - a.rE; A\ fi \\
\quad [\,]move\big(initloc, foodloc\big) \\
\quad [\,]move\big(foodloc, statloc\big) \\
\quad [\,]move\big(statloc, foodloc\big) \\
\quad [\,]freetextmemory \\
\qquad \rightarrow\ newSMS.val\ :=\ SMS.val \\
\quad [\,]AMMS.size < freeMMmemory \\
\qquad \rightarrow\ newAMMS.val\ :=\ AMMS.val \\
\quad [\,]VMMS.size < freeMMmemory \\
\qquad \rightarrow\ newVMMS.val\ :=\ VMMS.val \\
\quad [\,]play[\,]\ldots \\
\quad \mathbf{od}]\,|
\end{array}
$$

The P*h* computation unit is a hardware host, and therefore it has the exported variables *B* and *charge*, the imported variable *E*, and the four actions shown in (5): *charging*, *plug*, *skip*, and *act*. Several local variables are shown: *freetextmemory* is a boolean variable modeling whether or not there is enough room left for new text messages; *freeMMmemory* models the free memory amount left for multimedia messages; *newSMS* models a placeholder for incoming text messages, while *newAMMS* and *newVMMS* are placeholders for incoming audio and video multimedia messages, respectively. We have distinct variables for the text messages and multimedia messages because the latter are much bigger than the former, while the former are (still) more likely to be sent from public places such as restaurants. However, this is just a choice in our example.

As actions we have three for moving the phone between an initial, the restaurant, and the statue locations, *initloc*, *foodloc*, and *statloc*, respectively. Then, if there is space for text messages (the variable *freetextmemory* evaluates to *T*), the phone can get the lunch menu SMS or an SMS about the statue. If there is enough space for multimedia messages, then they are similarly obtainable and then also played. Since the phone is a general hardware device, getting charged from the restaurant sockets is modeled by (5).

The application is modeled by the parallel composition of these four components:

R*est* || ES || S*tatue* || P*h*

We note that both the restaurant component Rest and the statue component S*tatue* export an *SMS* variable. This semantically means that the phone P*h* imports this variable only from one component at a time. The location guard of the action *freetextmemory* → *newSMS.val*:=*SMS.val* ensures that *SMS* is located in the cell of the action; if there are two *SMS* variables in the cell, then one of them is chosen and the corresponding SMS value is copied to the phone's local variable *newSMS*.

TOWARDS ENERGY-AWARE DEVELOPMENT

In this section we emphasize two different approaches with respect to energy-aware development. The first one follows a classical customer-developer approach in which the customer specifies the energy requirements for a certain product and the developer sets out to deliver them. The second one is more innovative in the sense that the customer requirement for the device energy is to be flexible, i.e., adaptable to various circumstances. Even though flexibility is a less precisely specified requirement, we argue that it can be achieved if energy is considered from the beginning of the development lifecycle and for the all involved components, hardware or software. In the following we describe in some detail the two approaches.

Energy-Constrained Development

This type of development is based on some initial, customer-dependent constraint such as "this device should consume no more than *x* amount of energy per time unit" and is relevant to a series of products such as wireless sensors deployed a harsh or hostile environment, devices deployed in space, etc. Recharging this type of devices is not an option, hence the need to be sure of their energy consumption for evaluating their life spans.

There are approaches to develop devices that adhere to a specified amount of consumed power, described, for instance in (Goldsmith, 2005, Chapter 16.6). In here we outline a software engineering method whose starting point is a constraint expressed as function on the energy consumption of the overall device. This method implies a top-down development where the energy consumed by a device is explicitly taken into consideration.

The case study collection of Abrial, Börger, and Langmaack (1996) showed that several formal frameworks, including action systems, use refinement and decomposition techniques frequently for developing systems. When discussing refinement

techniques for a computation unit A we refer to the *behavior* of A as the set of state sequences that correspond to all the possible executions of A. In this context, we say that the computation unit A is *superposition* refined by the computation unit B (Back & Sere, 1996) when the behavior of A is still modeled by B and the new behavior introduced to B does not influence or take over the behavior of A. This means that new variables and actions can be added to B, in addition to those of A, but in such a manner that they do not modify or take over the state evolution of A. We employ refinement together with parallel decomposition for the overall development process. Parallel decomposition refers to expressing a computation unit A as a parallel composition $A = A_1 \| \ldots \| A_n$ and is a useful structuring technique.

By combining refinement and parallel decomposition, we are able to develop a device while keeping the energy requirement valid throughout the process. When specifying the first model of the device to be built, we define the energy constraint as an invariant for that model. This means that the constraint must be respected by that model upon initialization as well as after each action is executed. We can express the energy required from a certain computation unit A (the initial device model) by A.*rE*, hence we can express that A.*rE* is within certain limits: $A.rE \in [min,max]$. Adding details to the model towards implementation is achieved using refinement and decomposition. The refinement method ensures that the property is then kept throughout the development. Parallel decomposition of the initial module into several submodules does not influence the property but instead the property is refined as well. If $A = A_1 \| \ldots \| A_n$, then $A.rE = A_1.rE + \ldots + A_n.rE$. The invariant $A.rE \in [min,max]$ is rewritten as $A_1.rE + \ldots + A_n.rE \in [min,max]$ and thus, while the energy consumption of each module is not fixed, the sum of all these figures has still to respect the initial limits. This is illustrated in Figure 1 where we sketch the development process. The process starts with establishing the energy constraint at the requirements phase in collaboration between the customer and the developer. Then, the requirements have to be captured in an initial design model where the constraint becomes an invariant and subsequently this initial model is developed towards implementation while the invariant is kept.

Energy-Flexible Development

While the previous method considers the device as a whole consuming energy, the one outlined in the following takes a different perspective. We con-

Figure 1. Energy-constrained development

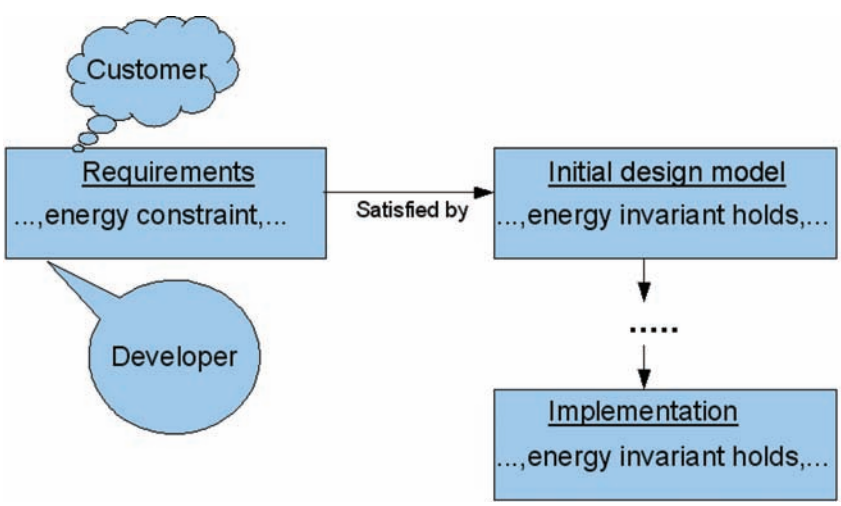
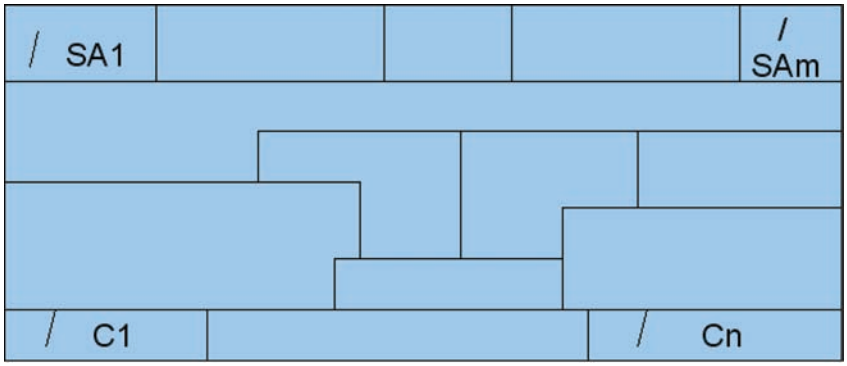


Figure 2. Energy-flexible development



sider a device as a sum of components - software and hardware - and impose that a user is able to switch them on and off to satisfy various energy requirements. This view is illustrated in Figure 2, where a generic device is seen as a sum of various components, such as the hardware parts $C_1, \ldots C_n$ and the software applications $SA_1, \ldots, SA_m$. These components have certain dependencies between them, for instance a software application cannot work unless some corresponding hardware parts are running.

Our claim is that, by establishing clear dependencies between which components depend on whom, various functioning patterns can be determined where not all the components need to be switched on and running all the time. The users can be given the choice of choosing which applications they need at some point and only the components that support the needed applications will be running. The rest can be conveniently switched off. In the figure we depicted that all the device components have a switch (/) that can be opened or closed.

Thus, each component of the device, software and hardware, has a certain amount of energy they consume. These amounts add up to the overall device-consumed energy per time unit, resulting in more battery time for fewer active components and vice versa, less battery time when more software application and correspondent hardware components are active.

If we consider a component of the device to be a computation unit in our modeling language, then this energy-flexible development can be modeled conveniently. Namely, we add to each computation unit a variable (*status*,,{*On*,*Off*},*Off*) and two actions *SwitchOn*:: *status.val*:=*On* and *SwitchOff*:: *status.val*:=*Off*. This allows the users to switch *On* and *Off* all the components, software and hardware. The enabledness condition of each action becomes stronger, namely the guard conditions of all the component actions *a:: A* become a conjunction of their previous guard condition *g*(*a*) and the condition *status.val*=*On*. Hence, an action cannot execute unless its computation unit is switched on.

RELATED WORK

The approach we take to energy-awareness in this paper is very different from the existing, energy-concerning literature. In general, the energy requirements of smart devices are considered at low abstraction levels. Rakhmatov and Vrudhula (2003) propose an analytical model to estimate the battery lifetime. Based on it, new algorithms are proposed for task ordering and voltage assignment that improve the battery usage, and thus prolong its lifetime. Power estimation techniques - where power is defined as the energy consumed per time unit - have been long proposed, both at

the gate-level estimation in VLSI designs (Najm, 1994) as well as at the high-abstraction level right above the Register Transfer Level (Landman, 1996). Hardware consumes a significant amount of energy due to the software it runs, and thus the same hardware configuration running different software implementations of the same functionality will have different energy consumption figures (Lafond, 2009). There is a considerable amount of work on optimizing the energy consumption of software by studying processor instructions (Seth, Keskar, & Venugopal, 2001), a family of processors (Russel & Jacome, 1998), or the memory management (Fan, Ellis, & Lebeck, 2001). It is also very useful to evaluate such figures for identifying the best high level software configuration. Simulations of different software architectures have been performed to evaluate the consumed energy of a system, for instance by Lafond and Lilius (2007) as well as by Lafond, Nybom, Björqvist, and Lilius (2008). For instance, the just-in-time Java compiler is faster than the interpreted execution mode but uses a bigger amount of memory, hence a tradeoff would need to be investigated for optimal energy consumption.

The type of energy-related research outlined above provides a very useful study of how energy is consumed and how the consumption could be optimized. However, there seems to be no approach to take energy into consideration at a very high abstraction level, such as devising devices that only consume a certain amount of energy. Also, the hardware and consequently the energy consumption are typically hidden from the user, when it can be through the user choices of applications to run that a portable device can save important amounts of energy. In this paper we outlined a high-level, software engineering method to developing devices of which energy consumption figures we are sure or devices with transparent availability of switching on and off the software and hardware components.

CONCLUSION

Energy consumption of devices is an important research topic. Various energy management techniques can affect the system reliability (Zhu, Melhem, & Mosse, 2004), heuristics for certain fault-tolerance algorithms can provide important energy savings (Unsal, Koren, & Krishna, 2004), while the battery limitation of energy can seriously affect the effectiveness of cryptographic protocols (Potlapally, Ravi, Raghunathan, & Jha, 2003) and thus compromise the security and privacy of devices. In this paper we addressed the topic of energy requirements of devices and made two important contributions. First, we proposed an energy-aware model that allows each software module and hardware component of a device to specify its energy requirements and battery amount, respectively. By modeling energy supplier entities (as electrical sockets), in addition to the software and hardware energy consumers, we obtain a complete view of how energy is affecting the operation of a software-embedding device. Second, we outlined two development approaches to obtain energy-aware systems. The first approach takes energy into account from the user requirements and based on the specified demands for energy consumption develops the device by keeping the energy consumption satisfied. The second approach amounts to developing flexible devices that a user can manipulate in terms of energy consumption. The methods could also be combined.

In terms of more concrete achievements, first we have identified the electrical sockets as the energy providers and the code and hardware resources as the energy consumers. This distinction implies that we have identified two network types: the electricity network and the resource network. Both networks are location-aware and possibly contain mobile entities. We have three network layers: first, the (fixed) locations in *Space* provide the infrastructure for all the resource types; second, the electricity network consists of simple sockets

with locations and since the sockets provide energy, they can be considered as *energy resources*; and third, the general resource network consists of data, code, computation unit or hardware resources with locations. These resources can charge when co-located with energy-provider sockets. Second, by modeling all resource types in MIDAS, we have a unitary notation for specifying, analyzing, and eventually refining the resources from specification to implementation. We can interrogate the locations (such as α) for various types of resource located there such as data (α.*var*), code (α.*action*), and hardware ($\exists\ B,charge \bullet \{B,charge\} \subseteq \alpha.var$). We can also interrogate the code (say *a*) for its requested power amount (*a.rE*) and the hardware host for its battery level and charging status, *B.val*, *charge.val*. We can check whether a certain location (say α) contains an electrical socket ($\exists E \in \alpha.var$) and if such a socket is free ($\exists E \in \alpha.var \bullet E.val$) or occupied ($\exists E \in \alpha.var \bullet \neg E.val$). Given all these query possibilities, the energy-aware applications seem numerous.

An overall feature of our approach is that we offer it via a formal method. There are several methods to treat energy consumption, for instance, via simulations to observe the best alternatives or via algorithms that optimize various aspects such as memory management or execution time. Formal methods are based on mathematical models and propose logic and algebraic inspired developments of the specifications. On one hand, this offers a solid validation of the proposed models and demonstrated properties. On the other hand, it offers a sustainable way of developing systems so that the final device is respecting the desired properties and thus may not need unit testing.

It is imperative that we can work towards obtaining more control over the applications that run on smart devices, especially in terms of energy consumption, so that we can better adapt them to our own needs. Our next step is to apply our proposed methods to develop small, smart devices that take energy into account.

REFERENCES

Abrial, J. R. (1996). *The B-book: Assigning Programs to Meanings*. Cambridge, UK: Cambridge University Press. doi:10.1017/CBO9780511624162

Abrial, J. R., Börger, E., & Langmaack, H. (1996). *Formal Methods for Industrial Applications - Specifying and Programming the Steam Boiler Control* (LNCS 1165). New York: Springer Verlag.

Back, R. J., & Kurki-Suonio, R. (1983). Decentralization of process nets with centralized control. In *Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (pp 131-142).

Back, R. J., Petre, L., & Porres, I. P. (2001). Continuous action systems as a model for hybrid systems. *Nordic Journal of Computing*, *8*(1), 2–21.

Back, R. J., & Sere, K. (1996). Superposition refinement of reactive systems. *Formal Aspects of Computing*, *8*(3), 324–346. doi:10.1007/BF01214918

Back, R. J., & von Wright, J. (1998). *Refinement Calculus: A Systematic Introduction*. New York: Springer-Verlag.

Bonsangue, M. M., Kok, J. N., & Sere, K. (1998). An Approach to Object-Orientation in Action Systems. In *Proceedings of Mathematics of Program Construction* (LNCS 1422, pp. 68-95). New York: Springer Verlag.

Fan, X., Ellis, C., & Lebeck, A. (2001). Memmory controller policies for DRAM power management. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*.

Goldsmith, A. (2005). *Wireless Communications*. Cambridge, UK: Cambridge University Press.

Lafond, S. (2009). *Simulation of Embedded Systems for Energy Consumption Estimation* (PhD Thesis). TUCS Dissertation Series, (113).

Lafond, S., & Lilius, J. (2007). Energy consumption analysis for two embedded Java virtual machines. *Journal of Systems Architecture*, *53*(5-6), 328–337. doi:10.1016/j.sysarc.2006.10.003

Lafond, S., Nybom, K., Björqvist, J., & Lilius, J. (2008). Receiver Coding Gain in DVB-H Terminals using Application Layer FEC Codes. In *Proceedings of the 3rd International Conference on Digital Telecommuincations (ICDT 2008)* (pp. 110-116). Washington, DC: IEEE.

Landman, P. (1996). High-level power estimation. In *Proceedings of International Symposium on Low-Power Electronics and Design* (pp. 29-35).

Laprie, J., Avizienis, A., & Kopetz, H. (Eds.), *Dependability: Basic Concepts and Terminology*. New York: Springer Verlag.

Lecomte, T. (2008). Safe and Reliable Metro Platform Screen Doors Control/Command Systems. In *Proceedings of the 14th International Conference on Formal Methods (FM 2008)* (LNCS 5014, pp. 430-434). New York: Springer-Verlag.

Michael, M. (2005). *Energy awareness for mobile devices*. Paper presented at the Research Seminar on Energy Awareness, University of Helsinki, Helsinki, Finland.

Najm, F. (1998). A survey of power estimationT Techniques in vlsi circuits. *IEEE Transactions on VLSI Systems*, *2*, 446–455. doi:10.1109/92.335013

Petre, L. (2008). Energy-Aware Middleware. In D. Bustard & R. Sterritt (Eds.), *Proceedings of the 15th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'08)* (pp. 326-334). Washington, DC: IEEE Computer Society Press.

Petre, L., Sere, K., & Waldén, M. (1999). A topological approach to distributed computing. In G. Stefanescu (Ed.), *Proceedings of the Workshop on Distributed Systems (WDS 99)* (ENTCS 28, pp. 97-118). Maryland Heights, MO: Elsevier Science.

Petre, L., Sere, K., & Waldén, M. (2006) A language for modeling network availability. In *Proceedings of the 8th International Conference on Formal Engineering Methods (ICFEM'2006)* (LNCS 4260, pp. 639-659). New York: Springer Verlag.

Potlapally, N., Ravi, S., Raghunathan, A., & Jha, N. K. (2003). Analyzing the energy consumption of cryptographic protocols. In *Proceedings of the 2003 international symposium on Low power electronics and design (ISLPED'03)* (pp 30-35). New York: ACM.

Rakhmatov, D., & Vrudhula, S. (2003). Energy management for battery-powered embedded systems. *Transactions on Embedded Computing Systems*, *2*(3), 277–324. doi:10.1145/860176.860179

Roman, C.-G., Julien, C., & Payton, J. (2004). A formal treatment of context-awareness. In M. Wermelinger & T. Margaria-Steffen (Eds.), *Proceedings of the 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004)* (LNCS 2984, pp. 12-36). New York: Springer Verlag.

Royal Academy of Engineering. (2009). *Engineering Values in IT*. London: The Royal Academy of Engineering, the Institution of Engineering and Technology and the British Computer Society. Retrieved August 1, 2009 from http://www.raeng.org.uk/news/publications/list/reports/Engineering_values_in_IT.pdf

Russell, J. T., & Jacome, M. F. (1998). Software power estimation and optimization for high performance, 32-bit embedded processors. In *Proceddings of the International Conference pn Computer Design (ICCD'98)* (pp. 328-333).

Seth, A., Keskar, R. B., & Venugopal, R. (2001). Algorithms for energy optimization using processor instructions. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*.

Tanenbaum, A. S. (2003). *Computer Networks* (4th ed.). Upper Saddle River, NJ: Prentice Hall/ Pearson.

Unsal, O. S., Koren, I., & Krishna, C. M. (2004). Towards energy-aware software-based fault tolerance in real time systems. In *Proceedings of the 2002 international symposium on Low power electronics and design* (pp. 124-129).

Yan, L., & Sere, K. (2004). A formalism for context-aware mobile computing. In *Proceedings of the Third International Symposium in Parallel and Distributed Computing/International Workshop on Algorithms, Models and Tools for Parallel Computing and Heterogenous Networks (ISPDC/HeteroPar'04)* (pp. 14-21). Washington, DC: IEEE Computer Society Press.

Zhu, D., Melhem, R., & Mosse, D. (2004). The effect of energy management on reliability in real-time embedded systems. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer Aided Design (ICCAD'04)* (pp. 35-40). Washington, DC: IEEE Computer Society.

This work was previously published in International Journal of Embedded and Real-Time Communication Systems (IJERTCS), Volume 1, Issue 2, edited by Seppo Virtanen, pp. 45-60, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 4
A Multidimensional Software Cache for Scratchpad-Based Systems

Arnaldo Azevedo
Delft University of Technology, The Netherlands

Ben Juurlink
Technische Universität Berlin, Germany

ABSTRACT

In many kernels of multimedia applications, the working set is predictable, making it possible to schedule the data transfers before the computation. Many other kernels, however, process data that is known just before it is needed or have working sets that do not fit in the scratchpad memory. Furthermore, multimedia kernels often access two or higher dimensional data structures and conventional software caches have difficulties to exploit the data locality exhibited by these kernels. For such kernels, the authors present a Multidimensional Software Cache (MDSC), which stores 1- 4 dimensional blocks to mimic in cache the organization of the data structure. Furthermore, it indexes the cache using the matrix indices rather than linear memory addresses. MDSC also makes use of the lower overhead of Direct Memory Access (DMA) list transfers and allows exploiting known data access patterns to reduce the number of accesses to the cache. The MDSC is evaluated using GLCM, providing an 8% performance improvement compared to the IBM software cache. For MC, several optimizations are presented that reduce the number of accesses to the MDSC.

INTRODUCTION

Most processors use a cache to overcome the memory latency. Some processors, however, employ software-controlled high-speed internal memories or *scratchpad* memories to exploit locality. Processors based on scratchpad memories are very efficient in terms of power and performance (Banakar et al., 2002). The power efficiency is due to the simple structure of the memory compared to caches. Scratchpad memories also have predictable latencies. These characteristics

DOI: 10.4018/978-1-4666-0912-9.ch004

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

make scratchpad memories a common choice for embedded processors.

Many kernels (e.g., multimedia kernels) have a working set that is predictable, which makes it possible to transfer data before the computation. It is often also possible to overlap computation with data transfers by means of a double buffering technique, where the data in one buffer is processed while the data for the next processing stage is fetched in another buffer. In scratchpad-based systems these data transfers usually need to be explicitly programmed using Direct Memory Access (DMA) requests. There are also many multimedia kernels, however, that process data that is known just before it is needed. This is the case, for instance, in the Motion Compensation (MC) kernel of H.264 video decoding. Only after Motion Vector Prediction it is possible to fetch the data necessary to reconstruct the frame. Other kernels have working sets that exceed the capacity of the scratchpad memory. This is the case in the Gray Level Co-occurrence Matrix (GLCM) kernel. It features a relatively random access that renders DMA requests for each individual access impractical. MC is an interesting kernel as its memory access pattern is similar to other important multimedia kernels such as texture mapping. GLCM features a fine-grain random access pattern that is representative of other tabulation algorithms, such as histogram.

Both kernels exhibit data locality that could be exploited by a cache. In MC the motion vectors are often closely related so that data that is (logically) adjacent to the reference area is needed to decode the next macroblock (MB). In GLCM the difference of adjacent pixels is often small so that the kernel accesses small parts of the GLCM matrix. In a scratchpad memory a cache can be emulated. This is often referred to as a software cache. Software caches, however, incur high overhead, representing up to approximately 50% (Gonzalez et al., 2008) of the total application execution time. Such high overheads could harm performance compared to hand-programmed, just-in-time DMA transfers. It is therefore necessary to reduce the number of cache accesses as much as possible. An additional feature of these as well as many other multimedia kernels is that they access 2- or higher-dimensional data structures and adjacent sub-rows are not consecutive in memory.

For such kernels we propose a Multidimensional Software Cache (MDSC). The MDSC stores 1- to 4-dimensional blocks (sub-matrices) and the cache is indexed by the matrix indices rather than a linear memory address. This approach both minimizes the memory transfer time and the number of cache accesses. The first is achieved by grouping memory requests, thereby reducing the overhead associated with memory requests. The latter is achieved by exploiting the multidimensional access behavior of the application.

Our experimental platform is the Cell processor. Implementing a software cache for the Cell processor is an active research topic (Balart et al., 2007; Lee et al., 2008; Chen et al., 2008). Balart et al. (2007) propose a compile time software cache with support for asynchronous transfers. The compiler uses asynchronous transfers to overlap memory transfers with computation. They report a speedup of 1.26 to 1.66 over synchronous transfers. Chen et al. (2008) propose a similar approach with support for runtime prefetching based on the access patterns. These works are complementary to our work since the MDSC can be used as the software cache implementation for the compiler.

The current version of the MDSC does not feature cache coherency. Currently it is not needed because either only read-only data is cached or efficient multicore kernel implementation avoids the need for coherency. However, cache coherency is an important feature for caches in a multicore environment. Lee et al. (2008) and Seo et al. (2009) propose a coherent shared memory interface for the Cell BE using software caches. It employs a software cache in the local store for page-level caching. It guarantees coherence at the page level and uses centralized lazy release coherency.

A static analysis tool for finding the best parameters for a software cache given an application is proposed in (Senthil et al., 2008). The tool uses traces with annotated memory accesses and bases its analysis on the frequency of cache accesses to a given cache line and the number of accesses between two accesses to the same cache line. A similar tool would be desirable for the MDSC as in this work we rely on exhaustive search to find the optimal parameters. The development of such a tool is future work.

Zatt et al. (2007) show that caching the MC reference area can save up to 60% bandwidth and more than 75% of the memory cycles compared to issuing a new request for each reference area. The presented solution is hardware specific, however, and therefore not sufficiently general/flexible to be implemented in a programmable embedded multimedia system.

This article makes the following contributions:

- An evaluation of the overhead incurred by a generic software cache for MC and GLCM.
- We propose the Multidimensional Software Cache (MDSC) that caches 1- to 4-dimensional blocks of data that are logically adjacent, thereby reducing the number of cache accesses and the DMA startup overhead.
- We determine the optimal parameters of the MDSC for MC as well as GLCM.
- For MC several optimizations are presented that reduce the number of accesses to the MDSC.
- We compare the performance of the MDSC to the performance achieved by hand-programmed DMAs as well as the heavily optimized IBM software cache. The experimental results show that for GLCM the MDCS provides an 8% performance improvement compared to the IBM software cache. For MC, the MDSC provides an average 65% improvement over just-in-time DMAs and 43% over the IBM software cache.

This article is organized as follows. First section describes the architecture of the Cell processor and evaluates the latency and throughput of DMA operations. The MDSC implementation, properties, and its application programming interface (API) are presented in the next section. It is followed by the description of the employed benchmarks and the MDSC optimizations for MC, and the presentation of the methodology used to evaluate the proposed software cache. Afterwards, the experimental results are presented and discussed. Finally, conclusions are drawn.

CELL PROCESSOR ARCHITECTURE

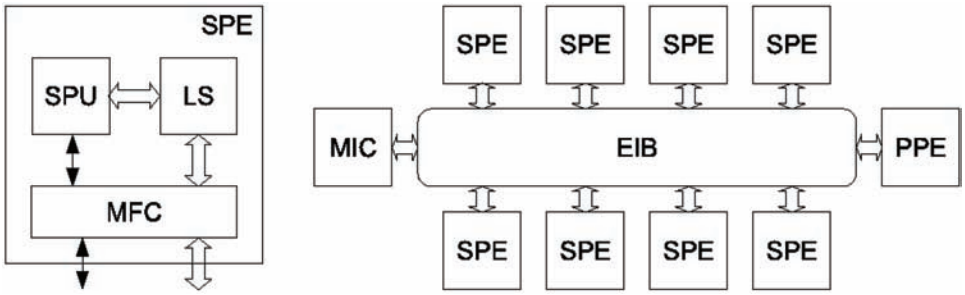
This section briefly describes the Cell processor. The main characteristics of the Cell processor are presented with the focus on the memory system. It also reports the memory latency for transferring data from main memory as a function of the size of the request and the number of requesting processing elements.

Cell Processor

The Cell Broadband Engine (Kahle et al., 2005; Gschwind et al., 2006) is a heterogeneous multicore processor designed for multimedia and game processing. It consists of one Power Processor Element (PPE) and eight Synergistic Processing Elements (SPEs) connected by the Element Interconnect Bus (EIB) that contains four 16B-wide data rings. A block diagram of the processor is depicted in Figure 1.

The PPE is a simplified version of the PowerPC processor family. It is based on IBM's 64-bit Power Architecture ("Power Architecture," 2010) with 128-bit vector media extensions. It is fully compliant with the 64-bit Power Architecture specification and can run 32-bit and 64-bit oper-

Figure 1. Cell Broadband Engine block diagram



ating systems and applications. The PPE is dual-threaded and has a two-way in-order execution pipeline unit with 23 stages. The PPE supports a conventional two-level cache hierarchy with 32KB L1 instruction and data caches and a 512KB unified L2 cache.

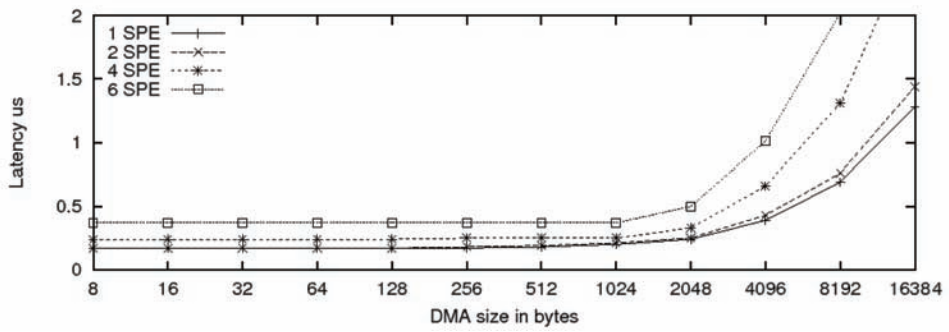
As depicted in Figure 1, each SPE contains a Synergistic Processing Unit (SPU), a Local Store, and a Memory Flow Controller (MFC). The Local Store is a 256KB scratchpad memory and the MFC is composed of a DMA engine, a memory management unit, and a bus interface. The SPUs are tailored for multimedia processing and are single-threaded, non-preemptive, two-way in-order processors. The register file consists of 128 128-bit wide registers. All instructions are Single-Instruction-Multiple-Data (SIMD) and they operate on 128-bit vectors with varying element width, i.e., 2 × 64-bit, 4 × 32-bit, 8 × 16-bit, 16 × 8-bit, or 128 × 1-bit. Data should be 128-bit aligned and there is no hardware support for scalar operations. The design decision of not supporting scalar and unaligned operations was taken to reduce the control complexity and to eliminate several stages from the critical memory access path (Gschwind et al., 2006).

An SPE can only access data and code stored in its 256KB Local Store. To access the external memory the SPU issue a DMA request to the MFC. There are four types of DMA requests: *put*, *get*, *putlist*, and *getlist*. A *put* request writes data from the Local Store to the external memory. A *get* copies data in the external memory to the Local Store. Requests can be grouped in a list with up to 1024 requests, and are issued by *putlist* and *getlist* requests. The DMA unit requests the data and sets a flag when the request is performed. Data and instructions are transferred in packets of at most 16KB and both the source and the target address must be 16B aligned. The DMA unit can handle up to 16 requests concurrently and data communication can be performed in parallel with computation. Double buffering can be employed to hide the DMA transfer latency. The Local Stores are mapped in the global memory address space to allow Local Store-to-Local Store communication, but this memory (if cached) is not coherent in the system.

DMA Latency

Figure 2 depicts the DMA latency as function of the DMA size. It can be seen that memory request latencies are approximately the same up to 1024 bytes for the Cell processor. It also depicts the DMA latency when several SPUs are communicating simultaneously. There is no difference in delay when 1 or 2 SPUs are fetching data because the Cell processor features a dual channel memory controller. However, a single SPU cannot make use of both channels simultaneously. Full bandwidth is achieved only when several SPUs are accessing the external memory simultaneously.

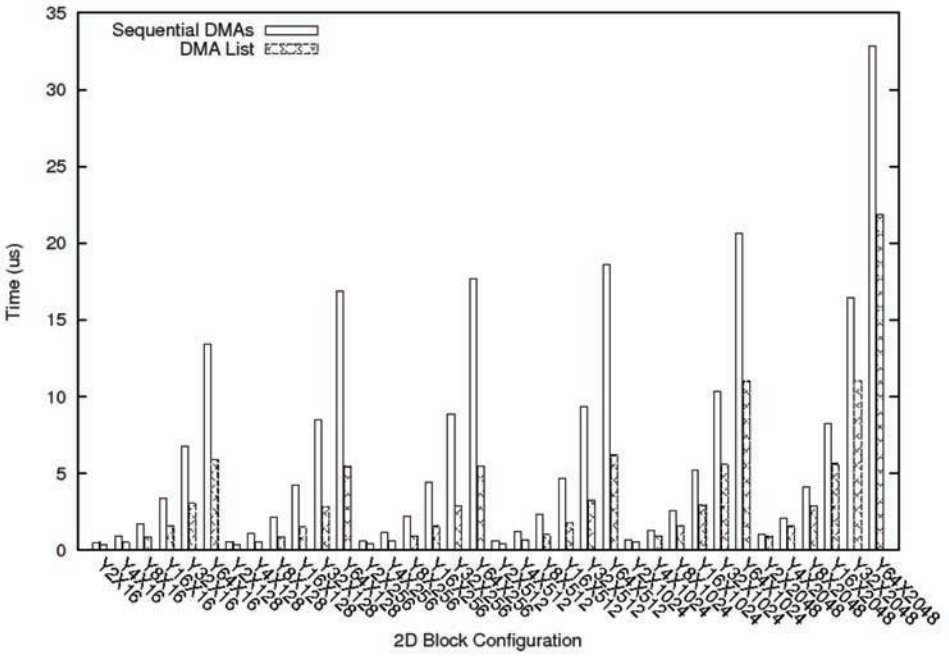
Figure 2. DMA latency as function size of the transfer size, for several SPEs communicating simultaneously



Several DMA operations can be grouped in a single DMA list operation in order to reduce the DMA startup cost. Figure 3 depicts the latency of DMA list operations for several numbers of requests (Y) and request sizes (X). For example, the label Y2X256 means that 2 DMA operations are grouped in a single DMA list operation, and that each operation fetches 256 bytes. We refer to the size of each individual DMA operation as the *line size*. For clarity, results for 64- and 128-byte lines have been omitted, as the results are very similar to the results for 32- and 256-byte lines. The latency for requesting the same block using several individual DMA requests is depicted for comparison.

The results show that requesting multiple lines reduces the request overhead. The average request time is reduced by 50% using DMA list, when

Figure 3. Latency of DMA list operation compared with a sequence of individual DMAs requests for the same 2D block configuration



compared with using sequential DMA requests. It ranges from 27% when fetching 2 lines to 69% for 64 lines.

MULTIDIMENSIONAL SOFTWARE CACHE

In this section we propose a Multidimensional Software Cache (MDSC). We start by highlighting the differences between hardware caches and software caches. Afterwards the reasons to shift from address indexed caches to data structure indexed caches are presented. Next, the arguments for a multidimensional software cache are given. With the underlining motives addressed, the proposed software cache (SC) structure is presented.

Scratchpad memories are more area and power efficient than hardware caches. However, they require additional programming effort as they need explicit commands to fetch data from the main memory. These commands can be automatically handled by compilers, but are usually handled by the programmer for better performance or because of lack of tools.

One option to increasing the efficacy of scratchpad based systems is to use a software cache. They increase the programmability and with software cache the limited size of the Local Store is less of a concern. Software caches, however, incur additional overhead, which can be prohibitive. This overhead is further increased if the cache does not match the application's data access pattern. This is the case when using a generic cache for MC and for other image processing applications, such as texture mapping.

Software caches provide the abstraction of a large, fast local memory to the programmer. This abstraction captures all data and instructions used in the program. Because of its generality, indexing by the memory address is a natural choice. However, because of this generality and because every access has to be done through the caches, it is not possible to exploit application-specific data locality.

Software caches for scratchpad based processors such as the IBM software cache ("Example Library," 2010) capture accesses to specific structures. Only the data structures that do not fit in the Local Store are likely to be accessed through the SC. Just like hardware caches, the IBM software cache uses the memory addresses of the data to index the cache. Once again it loses the opportunity to use the SC parameters information to exploit data locality. In this case it is critical, as accessing the cache implies runtime overhead. It is possible to exploit data locality to reduce SC access overhead, due to its characteristics. These characteristics include the parameters that are known at compile time and that SC captures only few specific data structures access.

The MDSC uses the indices of the accessed data structure to index the cache. So, instead of consulting the cache as *access_SC(&datastructure[i][j])*, we propose *access_MDSC(&datastructure, i, j)*. Although similar, the second makes explicit more information about the data structure and the access. Another characteristic of the MDSC is the ability to mimic in the SC the logical organization of the accessed data structure in the main memory. The cache blocks are 1- to 4-dimensional. A 2D cache can be used to store rectangular areas of images while a 3D cache can be used to store areas of a sequence of video frames. We cannot give an example where 4D cache blocks would be useful. However, because of the SIMD instructions set of the SPE, 4D cache blocks are just as efficient as 2D and 3D cache blocks.

This approach makes the MDSC differ from a regular cache in two ways. First, it differs in the tag calculation and the set allocation. In a regular cache the tag is the address shifted by the base-2 logarithm of the size of the cache line. For MDSC the tag is the concatenation of each index shifted by the base-2 logarithm of the size of its block dimension. The set is also calculated based on the indices. An exclusive or operation is performed

between each index value and its successor and the results are added together. The set is the sum modulo the number of sets in the MDSC. The second difference is the format and the load of the multidimensional block. The multidimensional block is formed by a group of cache lines gathered from memory according to the number and size of the dimensions of the MDSC. For a 2D MDSC, *n* cache lines represent a block. A strided access to the main memory is performed to load the consecutive lines. In our implementation a DMA list is created with a DMA request for each list entry. It is similar for 3 and 4-dimensional blocks where a 3D block is a collection of 2D blocks and a 4D blocks is a collection of 3D blocks.

This approach presents two advantages over regular caches. First, it reduces the memory latency by grouping several memory requests. Since, as was shown in the previous section, a single DMA list operation has a lower latency than several sequential DMAs. For regular caches, accessing a new image area would result in a new DMA request for each line of the new area being accessed. The second advantage is that the MDSC can be used to reduce the number of accesses to the SC. As the shape of the cache block is known, it can be used to access the data of a cache set without actually checking if the data is present in the cache. This can be done by simple pointer arithmetic. In other words, a single cache lockup is necessary for accessing an entire block.

Like a regular cache, the MDSC performs the following steps to access a block:

1. Check if the data is already present in Local Store. Each block in the cache is represented by a tag. The tag is formed by concatenating the indices of the data that is being accessed after dividing each index by the size of its respective block dimension. In the case of a set associative cache, a hash function is used to define in which set the block referred to by the indices is stored. The new formed tag is then compared with the other tags in the specific set in the tag array (in case of a set associative cache) or with all tags in the tag array (for a fully associative cache).
2. In case the tag is not present, decide where to place the new block. If the block is not currently stored in the cache it is first necessary to determine where it should be placed. The MDSC uses the next position of the last allocated block in a FIFO fashion, both for the set associative and fully associative configurations.
3. If out of space, decide which block to eject from Local Store. If the chosen place was already been used by another block that was modified, it needs to be written back to the main memory. The tag of the block being written back is separated according to the size of each dimensions of the block to recover the block address in the main memory.
4. If necessary, perform DMA operations. Issue DMA requests to, if necessary, copy the modified data back to main memory and to copy the new block from main memory to the Local Store. A MDA list is prepared one entry each line of the multidimensional block. With the list ready a *putlist* and/or *getlist* is issued. The process blocks until the DMA unit informs the SPU that the new block is present in the Local Store.
5. Perform memory access - A modulo operation is performed between each index of the request position with it's respectively dimension size. The result is used to calculate the position of the requested data in the cached block.

The associativity of the MDSC can be configured. It allows a fully or a set associative configuration and for static or dynamic implementation. A fully associative cache is possible when the number of cache blocks is small. For the fully associative configuration, the MDSC uses First-In-First-Out (FIFO) policy to replace blocks when the cache is fully utilized. The FIFO was selected

due its low implementation complexity. For set associative a round-robin mechanism is used to select the block to be replaced.

In the static implementation the MDSC parameters are constants and thus are known at compile time while the dynamic implementation allows the MDSC parameters to be modified at runtime. The former is more efficient than the latter as it allows optimization of the code for several parameter options. However, a dynamic configuration is necessary when the data to be cached can have different characteristics, such as the resolution of the video being decoded, and the MDSC needs to adapt to these characteristics.

The Application Programming Interface (API) consists of two functions: *access_sc(&datastructure, i, j)* and *accessp_sc(&datastructure, i, j)*. The function *access_sc* returns the data stored in the (i,j) position while the function *accessp_sc* returns its address (memory pointer). These functions checks if the 2D block which contains *datastructure[i][i]* is present in the software cache. If it is, this function returns immediately. If not, this function blocks until the 2D block which contains *datastructure[i][i]* is fetched from main memory to the software cache and then returns to the caller.

Because the MDSC uses matrix indices to index the cache, the boundaries of the data structure need to be specified. This can be done via macros in the static configuration, thus increasing the performance, or dynamically at runtime when using the dynamic configuration. Figure 4 depicts how the MDSC can be configured.

STUDIED APPLICATIONS AND MDSC ENHANCEMENTS

In this section, the applications used for the case study are presented and qualitative reasons are given why the studied applications could profit from a MDSC. First, the GLCM algorithm is presented. It is followed by a description of MC. For the MC kernel it is possible to exploit the access behavior. We will describe several enhancements that exploit this fact.

Figure 4. MDSC interface

```
#define CACHE_NAME gsc_y
#define CACHED_TYPE unsigned char
#define CACHE_TYPE          0     /* 0=read-only, 1=read-write              */
#define CACHE_STATS               /* Activates statistic collection         */
#define CACHE_FULL_ASSOC    0     /* 1=cache is fully associative           */
#define CACHE_LOG2NWAY      2     /* Log 2 number of ways                   */
#define CACHE_LOG2NSETS     5     /* Log 2 number of sets                   */
#define CACHE_DIM           3     /* Number of block dimensions             */

#define CACHE_X_LOG2SIZE    9     /* Log 2 first dimension (line size)      */
/* When accessing the MDSC, the first index must be between 0 and CACHE_X_RANGE */
#define CACHE_X_RANGE       1920
#define CACHE_LOG2_X_RANGE 11    /* Log2(CACHE_X_RANGE)

#define CACHE_Y_LOG2SIZE 4       /* Log 2 second dimension (number of lines)*/
#define CACHE_Y_RANGE 1088
#define CACHE_LOG2_Y_RANGE 11    /* Log2(CACHE_Y_RANGE)

#define CACHE_Z_LOG2SIZE 1       /* Log 2 third dimension                  */
#define CACHE_Z_RANGE 256
#define CACHE_LOG2_Z_RANGE 8     /* Log2(CACHE_Z_RANGE)                    */

#include <mdsc-api.h>
```

GLCM

The Gray-Level Co-occurrence Matrices (GLCM) is a tabulation of how often different combinations of pixel brightness values (gray levels) occur in an image. The second order GLCM considers the relationship between groups of two (usually neighboring) pixels in the original image. It considers the relation between two pixels at a time, called the reference and the neighbor pixel. The GLCM is useful to extract statistical characteristics of the image and is used in medical imaging and content based image retrieval (Shahbahrami et al., 2008). In this study, all 9 neighboring pixels are examined, as depicted by the pseudo code in Figure 5.

In this application, the source image being processed can be easily accessed through DMAs. The temporal locality of the image is very low and the spatial locality can be captured with DMAs. Also the DMA latency can be hidden using double buffering. However, the GLCM matrix is indirectly indexed and its size is 256KB, 256×256×4 bytes. This is the entire size of the Local Store and this is only for one color component of the image. Differently from the source image, it is not possible to determine in advance which position of the matrix will be accessed to make use of the DMAs. Using DMA requests to access each position of the matrix would lead to huge number of DMA requests that would slowdown the performance by 2 orders of magnitude, making this solution unfeasible.

Photos, however, usually exhibit large amounts of spatial redundancy, which is exploited by image compression algorithms. The same type of redundancy can be exploited here by caches. Because the change of color is usually smooth, two-dimensional portions of the GLCM matrix are likely to be accessed close in time, in other words, the spatial redundancy of photos is translated as temporal redundancy when updating the matrix.

H.264 Motion Compensation

Motion Compensation (MC) is the process of copying an area of the reference frame to reconstruct

Figure 5. Pseudo-code for GLCM

```
int GLCM[256][256];

for(i=1; i < img_height-1; i++)
  for (j=1; j < img_width-1; j++)
  {
      GLCM[ img[i][j] ][ img[i-1][j-1] ]++;
      GLCM[ img[i][j] ][ img[i-1][j ]  ]++;
      GLCM[ img[i][j] ][ img[i-1][j+1] ]++;

      GLCM[ img[i][j] ][ img[i][ j-1]  ]++;
      GLCM[ img[i][j] ][ img[i][ j+1]  ]++;

      GLCM[ img[i][j] ][ img[i+1][j-1] ]++;
      GLCM[ img[i][j] ][ img[i+1][j ]  ]++;
      GLCM[ img[i][j] ][ img[i+1][j+1] ]++;
  }
```

the current frame. For advanced video codecs such as H.264, both the reference frame and the Motion Vectors (MV) need to be calculated. In H.264, this process is known as Motion Vector Prediction (MVP) and is part of the MC. Only after the MVP it is possible to request the necessary data to reconstruct the frame. In H.264, MVs can span half of the vertical frame size and it is possible to have up to 16 frames as candidates for reference frame. This makes it impossible to speculatively load all possible areas in advance. In our Cell implementation of macroblock (MB) decoding, the MC kernel is the most time consuming, representing 62% of the total execution time. It requests the reference area through DMA transfers and waits until data is present in the Local Store. The rest of the execution time is spent in DMA data in and out (excluding the reference area) 14%, deblocking filter 17%, and Inverse Discrete Cosine Transform 7%. The memory requests represent 75% of the execution time of the MC kernel. These numbers show the importance of improving the performance of MC.

The unpredictability of the data accesses in MC causes two significant problems on scratchpad memory-based processors. The first problem is that the data transfer cannot be overlapped with the computation. The process has to wait for the data to be transferred to the scratchpad memory. Because H.264 allows very fine grained area to be copied, up to 4x4 pixels, the waiting time for the data can be significant. The second problem is that the data locality cannot be exploited. It is difficult to keep track of the memory area present in the scratchpad memory and new data must be requested for each macroblock (MB) partition. Because the MVs are usually small and not randomly distributed, the same area can be copied several times. Zatt et al. (2007) show that caching the reference area can save up to 60% bandwidth and more than 75% of the memory cycles.

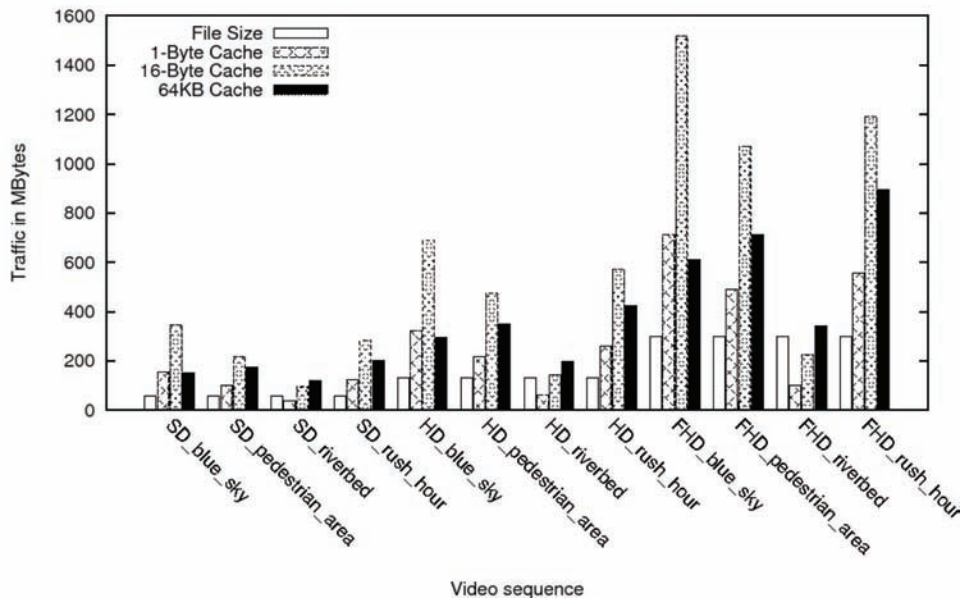
First, the data locality exhibit by H.264 MC is investigated. H.264 sequences from HDVideoBench (Alvarez et al., 2007) are used as input for the experiments. Each video sequence consists of 100 frames in standard (SD), high-definition (HD), and full high-definition (FHD) resolutions at 25 frames per second.

To evaluate the data locality, the number of bytes requested from memory is measured. For the measurement, the motion vectors and reference indices are extracted from the encoded sequences for each MB partition. Because of the MC quarter-pixel precision, adjacent additional areas need to be fetched from the memory. For vertical filtering, five extra pixels are required for each line, while for horizontal filtering, five extra lines are required. Details of the MC implementation can be found in (Azevedo et al., 2007).

A tool was developed to translate the extracted MVs to memory requests in the DineroIV (Edler & Hill, 2010) cache simulator input format. DineroIV was used to report the requested number of bytes for each sequence. Three simulations were performed and the results are reported in Figure 6. The first simulates a 1-byte cache to depict the temporal data reuse of MC. The second simulation reports the data traffic for a 16-byte cache with a 16-byte line size. The third simulation reports the data traffic for a 64KB direct mapped cache with 64-byte cache lines.The size of the original uncompressed sequence is presented as file size. The difference between the first and second simulations with the original file size shows the amount of data being reused.

The results show that the sequences exhibit data locality. In case of the 1-byte cache, the data locality is temporal, while in the second case, of the 16-byte cache, the reported data locality is both spatial and temporal. The Riverbed sequences do not exhibit a lot of data reuse. These sequences use mostly intra prediction MBs that uses neighboring pixels to predict the area to be reconstructed, thus not making use of MC. MC references about twice the volume of data of the original sequence (1-byte cache). But, because of memory alignment constraints, the actual volume of transferred data is about 3.5 times the volume

Figure 6. Data locality in MC



of the original sequence (as shown by the 16-byte cache result). The 64KB cache reduces the volume of data transferred by 34% compared to the 16-byte cache. It reduces the data volume to 2.3 times of the original sequence. This indicates that the cache is capturing part of the data locality of the MC. It can be improved as it is an unified cache capturing the three color component access, thus increasing conflicts.

The MDSC reference frame number and the vertical and horizontal coordinates of the MV. This access method enables to exploit the access pattern as it exposes pattern specific information. Each block of the MDSC is an x×y rectangular area of a frame. The *x* and *y* values and their ranges are configurable at runtime.

Because of the data locality characteristics of MC, a fully associative configuration was selected, as it reduces the number of conflicts. A fully associative cache is possible because of the small number of blocks present in the implementation, as will be shown in the next section. To support different video resolutions, the MC has to use a dynamic configuration of the MDSC.

MC Enhancements

The video frames are stored in YCbCr format instead of the RGB format, and each component is stored in a separated data structure. To increase the compaction, the color components (Cb and Cr) are subsampled 1:4 as they are less perceptive to the human eye. The MVs are the same for all components, but, because of the subsampling, they need to be adjusted for the Cb and Cr components. The MDSC configuration exploits this feature and checks and requests at once all components. This reduces the number of accesses to the MDSC by a factor of 3 and overlaps the memory requests, thus reducing the memory latency.

Four additional enhancement strategies that are employed to reduce the number of accesses to the MDSC are described below. Each strategy builds upon the strategies presented above it.

Extended_X: To reduce the number of accesses to the software cache, an extended line technique was implemented based on the technique described in (Azevedo et al. (2007)). The maximum line size to be accessed is 21 pixels.

This consists of the 16 pixels of the maximum MB partition plus 5 extra pixels for quarter-pixel filtering. These pixels can be spread over three 128-bit words. In this technique not only the pixels corresponding to 2D cache block are fetched, but also the 32 pixel columns to the right of the block. Adding these as extra columns for each cache line guarantees that all the data that need to be filtered are present in the cache. This reduces the number of cache accesses per line from 2, or occasionally 3, to 1. Note that this implies that some pixels can be present twice in the cache, once as part of the macroblock it pertains to and once stored with the macroblock to the left of its macroblock. When accessing the cache, however, the first one is returned. Because only read-only data is cached, this does not cause inconsistency problems.

Extended_XY: This technique is an extension of the Extended_X technique and can be applied only when the vertical span of the block is equal or larger than 32 lines, as the number of line needs to be a power of 2. Because the maximum MB partition plus the additional area are 21 lines long in the vertical direction, just two accesses to the MDSC are sufficient to guarantee that the data are present in the cache. Only the first and last lines of the partition need to be accessed. The border between the two cached areas is found by masking the y coordinate of the MV with the height of the block.

SIMD: Since the Cell SPE is a SIMD architecture, a natural step to improve performance is to vectorize the tag search. Each tag is a 32-bit integer and the SPE allows vector operations with four 32-bit words. In this optimization, four positions of the tag array are compared simultaneously with the searched tag. Once the tag is found, each of the four positions of the tag array is compared individually to find the block index.

Fixed: As previously stated, the parameters for the MDSC are configurable at runtime. In this version, the cache parameters were fixed, meaning that loop boundaries are known at compile time. This allows for certain loop optimizations to be performed, including the elimination of branches and loop unrolling.

EXPERIMENTAL METHODOLOGY

This research focuses on the performance of the cache access functions. Because of that, the kernels that access the cache will be measured. For the MC, the access to the reference area is evaluated. For the GLCM, the whole function is measured. This is because the GLCM only performs a load, an add, and a store for each position. The images are loaded through explicit DMA transfers and do not make part to the GLCM calculation.

The measurements were performed on a Sony Playstation 3 (PS3). The Cell in the PS3 has 6 out of the 8 SPEs available. One SPE is disabled for redundancy purposes and another one is used by the system for resources access management. Another important characteristic is that the PS3 has only 256MB of RAM. This small amount of memory causes memory swaps with the disk. For this reason, the MC kernel was modified to access only 5 frames, which corresponds to the number of frames in the decoder frame buffer. Otherwise the DMA transfer time is doubled due to memory (de)allocation routines by the OS.

To measure the performance of the kernels, SPU hardware decrementers were used. The decrementer runs at a smaller frequency than the processor itself. In our case, it runs at 78.8 MHz, which is 40 times slower. This approach is not suitable for fine grain profiling, but is sufficiently accurate to measure the performance of functions. The *spu_read_decrementer* and *spu_write_decrementer* intrinsics are used to access the decrementer.

EXPERIMENTAL RESULTS

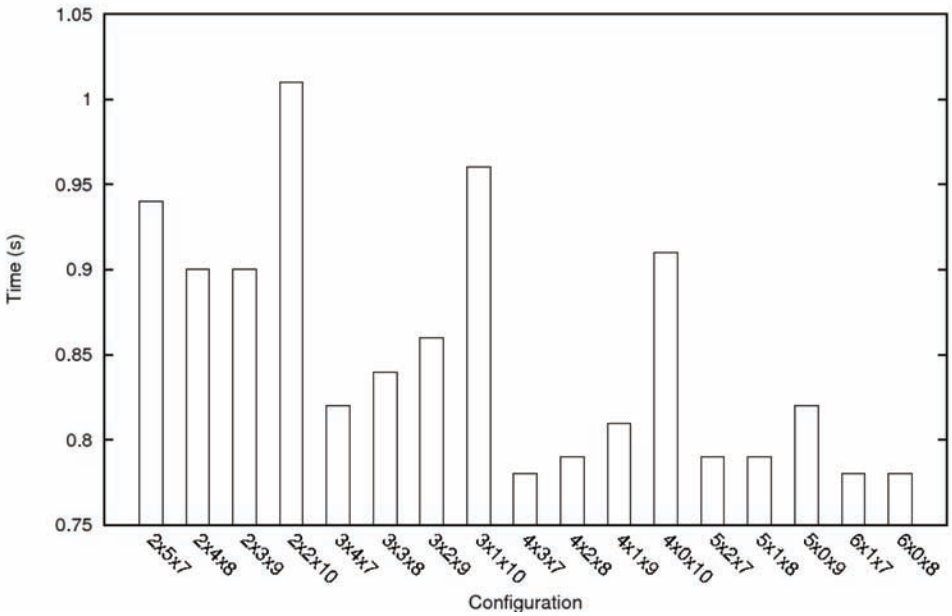
The HDVideoBench (Alvarez et al., 2007) is used as benchmark for the experiments. Each video sequence is composed by 100 frames in standard (SD), high-definition (HD), and full high-definition (FHD) resolutions at 25 frames per second.

All results were obtained using a single SPE. The experiments were not performed using several cores because the parallelization strategy would influence the results and would change the focus of the work. Both kernels can run in a multicore environment without cache coherency. The GLCM kernel could have a separated matrix for each core and process a slice of the frame. After finishing the processing, the matrices would need to be added together for the final result. For MC the cache is a read-only, thus is does not need cache coherency to work in a multicore environment.

GLCM Results

To generate input for GLCM, the first frame of each HDVideoBench sequence was transformed into an RGB image and each component was processed. Several configurations of a 4-way set associative 64KB MDSC were tested to determine the optimal configuration. The number of sets ranged from 4 to 64, the number of lines from 1 to 32, and the line size from 128 to 1024 bytes. Figure 7 depicts the results of all possible configurations that respect the 64KB cache size. In this figure each bar is labeled as S×L×B, where *S* is the base-2 logarithm of the number of sets, *L* is the base-2 logarithm of the number of lines in a 2D block, and *B* is the base-2 logarithm of the line size in bytes. For instance, 4×3×7 denotes the configuration with 16 sets, 8 lines, and 128 bytes per line. In other words, the block size of this MDSC configuration is 8×128. As shown in the figure, the GLCM performs better with a higher number of sets with a shorter line size. Because of these characteristics a fully associa-

Figure 7. Time taken by the GLCM kernel for several MDSC configurations



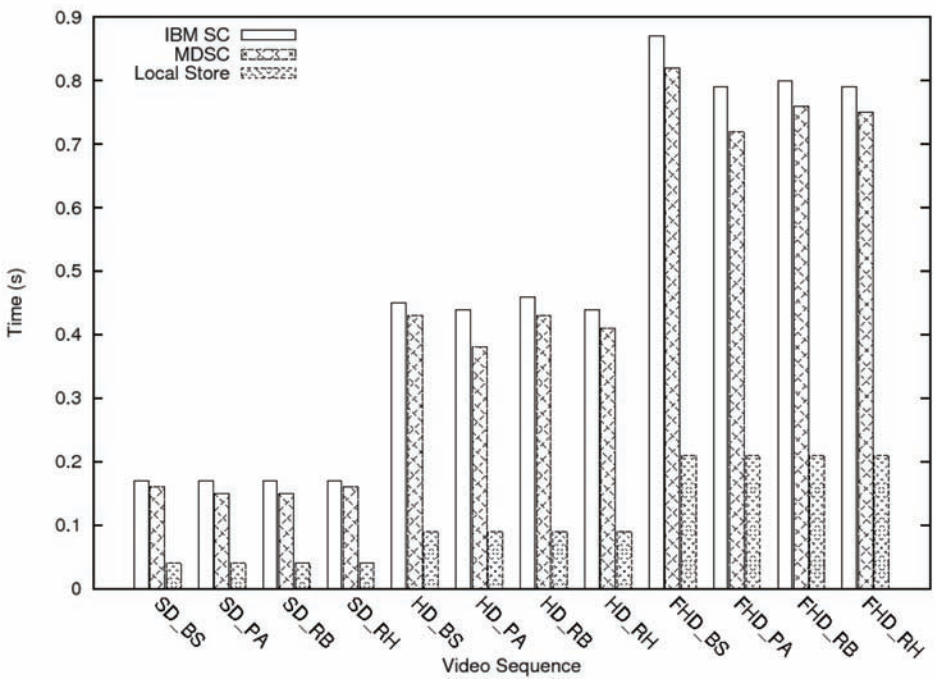
tive MDSC was not evaluated as it requires the opposite characteristics to perform well.

The best performing MDSC configuration consists of 64 sets and uses a block size of 1×256 bytes (6×0×8). Surprisingly, 1-dimensional blocks yield the highest performance. The reason for this is the latency for strided access. The reduction in miss rate, when increasing the block height, is not larger than the increase in the latency for longer DMA list request. However, the performance of the configurations 6×1×7 and 4×3×7 that uses blocks of 2×128 bytes and 8×128 are less than 1% lower than the best configuration.

Figure 8 compares the performance of the optimal MDSC configuration to the optimal configuration of the IBM SC. Experimentally we determined that the optimal 64KB IBM SC configuration for GLCM is 4-way set-associative, consists of 128 sets, and uses a line size of 128 bytes. For comparison, the time taken by the GLCM kernel when the GLCM matrix fits in the Local Store is also depicted. To achieve this, the image color resolution had to be quantitized to 6 bits.

Compared to the IBM SC, the MDSC provides an 8% improvement on average. This performance improvement is due to the lower miss rate achieved by the MDSC. For example, for the FHD BlueSky sequence (denoted FHD/BS in Figure 8, the MDSC incurs a miss rate of 2.4%, while the IBM SC incurs a miss rate of 2.6%. This 0.2% difference in miss rate translates to an 8% increase in memory requests by the IBM SC compared to the MDSC. It also increases the number of times the miss handling code of the software cache is executed. The miss handling code is much more time demanding than the hit branch as it has to choose a block to replace, calculate the block memory address based on the old tag, and issue a request for the new block.

Figure 8. Time taken by the GLCM kernel for several video sequences when the optimal IBM SC configuration is employed (IBM SC), when the optimal MDSC configuration is employed (MDSC), and when the GLCM matrix would fit in the Local Store



The MDSC set hash function, based on indices instead of linear addresses, reduces the number of conflict misses compared to the IBM SC. The MDSC hash function more equally distributes the number of accesses over the sets. For example, for FHD BlueSky the average deviation of the number of accesses to each set of the MDSC is reduced by 18% when compared with the IBM SC with the same configuration. With a better distribution of accesses the number of replacements is lower, which reduces the miss rate.

Compared to when the GLCM matrix would fit in the Local Store, which is included only for comparison, the MDSC incurs a slowdown of 3.75 on average. Considering that the GLCM kernel consists of simple processing and the fact that an access to the MDSC takes around 20 cycles, the MDSC (as well as the IBM SC) quite efficiently bridge the memory gap.

MC Results

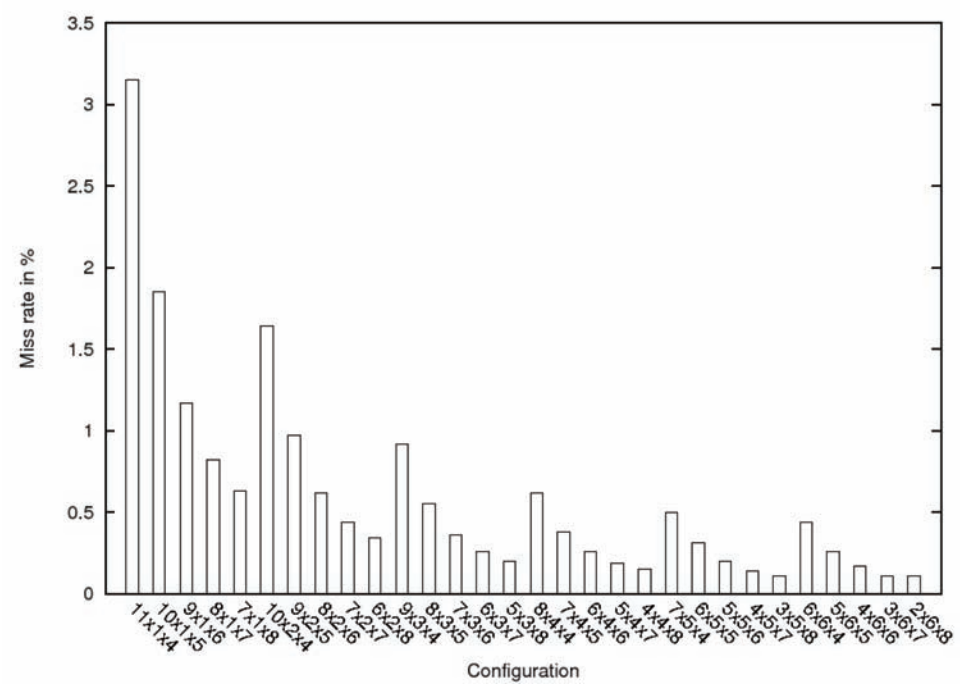
In order to determine the optimal 2D block size of the MDSC for the MC kernel, the design space was explored. The MVs from the HDVideoBench sequences were used as input. MVs and reference indices are extracted from the encoded sequences for each MB partition. Blocks of size $2^n \times 2^m$ were tested, for n between 1 and 6, and for m between 5 and 8. For each block size, the miss rate was calculated. The size of the MDSC was fixed at 96KB, 64KB for the Y components and 16KB for the Cb and Cr components each. As noted in Studied Applications section, the MDSC for MC is fully associative. None of the enhancement discussed in MC Enhancements were considered in this exploration.

Figure 9 depicts the miss rate for each design point. It uses the same labeling style as previous figures with base-2 logarithm of the number of blocks in the cache, number of lines per block, and line size in bytes. Not surprisingly, the miss rate decreases when the 2D block size increases. The results show that the 8×64×128 blocks exhibit a miss rate of 0.11%, and that the 8×32×256 and 4×64×256 exhibit the same miss rate. The 32×256 block was selected because, as depicted in Figure 2, fetching 256 bytes is as efficient as fetching fewer bytes. 32 rows also allow the use of the Extended_XY enhancement methodology. The 0.11% miss rate reduces the total number of DMA transfers to 32% of the implementation using hand programmed DMA requests.

A similar exploration was performed for the IBM SC. The IBM SC was configured to be 4-way set associative and the line size was varied from 16 to 256 bytes. As for the MDSC, a 64KB cache was used for the Y components and two 16KB caches for both the Cb and Cr components. It should be noted that unlike the MDSC the IBM SC uses three separated caches, one for each component. The IBM SC uses the Round-Robin replacement policy. The best performing SC configuration uses 256-byte lines and has a miss rate of 8.6% for Y component. Because the IBM SC can use only 1-dimensional blocks while the application processes 2-dimensional blocks, the IBM SC miss rate is much higher that the miss rate of the MDSC.

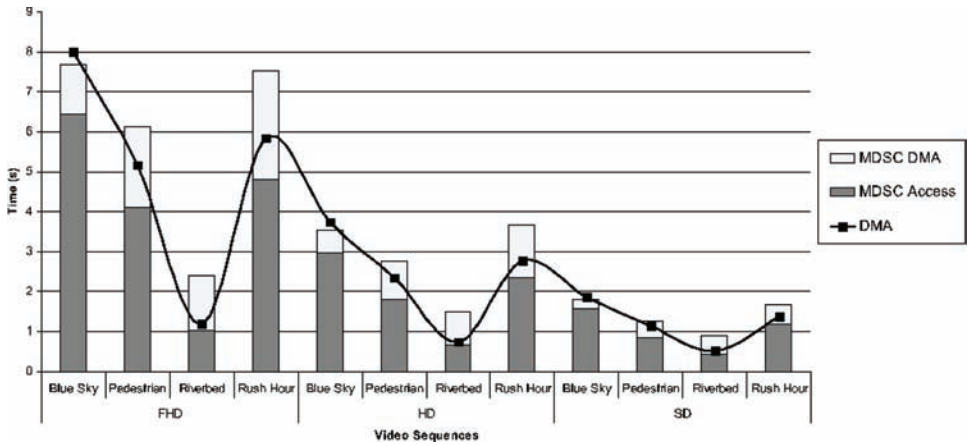
Figure 10 breaks down the time taken by the MC kernel when the baseline MDSC (without enhancements) is employed into time needed to access the MDSC and time required for the DMA transfers. For comparison, the time taken by a version that does not use a software cache but fetches the reference areas from main memory using explicit, hand-programmed DMA transfers is also included and labeled *DMA*. The results include the time for frame border detection, the time to fetch the additional quarter-pixel area, the time to fetch additional Cb and Cr components, and the handling of 128-bit alignment constraints. The border detection and the alignment calculation are included because in the DMA time because they are overlapped with memory transfers and account for less than 1% of the total DMA time. The baseline MDSC implementation performs an MDSC access for every 16-byte quadword. It is

Figure 9. Miss rate incurred by the MC kernel for different configurations of a 96KB MDSC



clear from the figure that the majority of the time is spent accessing the cache rather than transferring data. This is because for every MDSC access the index has to be calculated and the tag has to be compared to the tags stored in the software cache. This overhead is relatively time consuming compared to the time taken by the DMA transfers. Furthermore, for all but one sequence, the version that uses hand-programmed DMAs is faster than the version that employs the baseline MDSC. The used sequences are composed of 100 frames and are 4 seconds long (25 fps). The proposed

Figure 10. Breakdown of the time taken by the MC kernel for different input sequences when the baseline MDSC is employed and the time taken when explicit, hand-programmed DMA transfers are used



enhancements exploit the known memory access behavior to reduce this overhead.

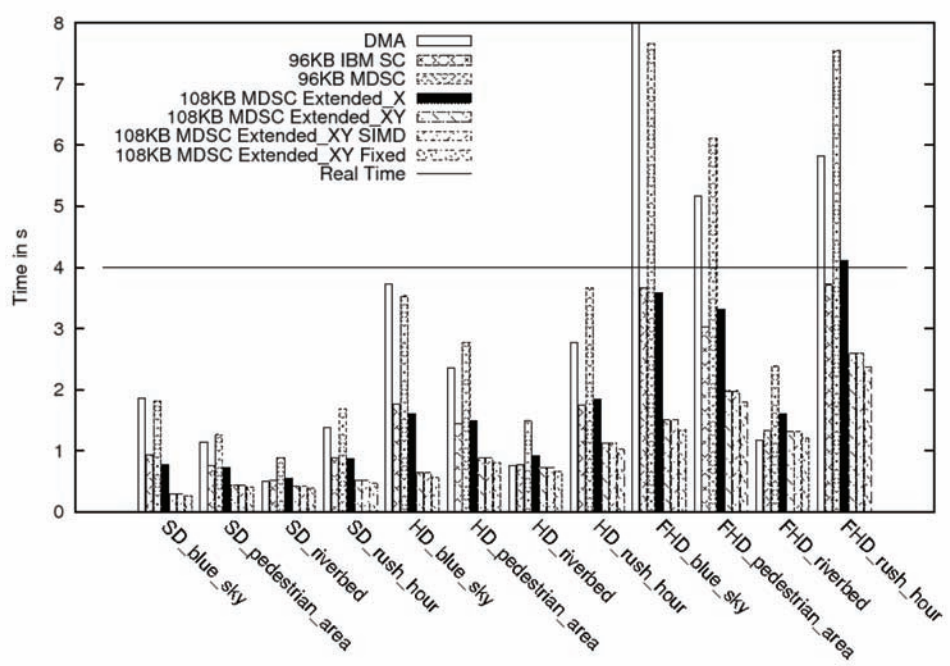
Figure 11 compares the performance of the direct DMA version of MC, the IBM SC, the baseline MDSC, and the MDSC extended with the enhancements described in Section MC Enhancements. It depicts the time in seconds to fetch the reference area from main memory to the SPE scratchpad. Our baseline for comparison is the DMA version of MC. The line labeled *Real Time* depicts the performance required for real time processing. As in the previous experiment, the DMA include border and alignment handling while the other versions depict the time required for MC only.

When the number of MDSC accesses is reduced with the Extended_X technique, an average 25% improvement over the DMA version is achieved. The Extended_X technique reduces the number of MDSC accesses by a factor of 2, because it ensures that when the first pixel in a line is present in the MDSC, the entire line is present. Checking only the presence of the first and last line of the MB partition, as is done in the Extended_XY technique, results in only two MDSC accesses per MB partition. This substantially increases the efficiency of the MDSC implementation and achieves an average 60% execution time reduction compared to the direct DMA version of the code.

The SIMD version of the MDSC does not provide an additional performance improvement. Its overhead cancels the benefits because of the small number of 2-dimensional blocks (eight) in the implemented MDSC. Fixing the parameters of the MDSC so that certain compiler optimizations can be performed yields an additional 5% execution time reduction, leading to a total average performance improvement of 65%. Fixing the parameters of the MDSC is similar as using the MDSC static configuration.

Compared to the IBM SC, the MDSC with the Extended_XY enhancement is 37% more perfor-

Figure 11. Time taken by MC for the direct DMA version, the IBM SC, the MDSC, and the various MDSC enhancements. The input sequences are 100 frames and 4 seconds long



mance efficient. The performance improvement increases to 43% when comparing the IBM SC with the Fixed implementation of the MDSC. This improvement is mainly due to the fact that because the MDSC uses 2-dimensional blocks and because the Extended_XY technique ensures that the entire reference area is included in at most two MDSC blocks, two MDSC accesses are sufficient to determine if the reference area is in cache, whereas the IBM SC requires at least one access for every line in the reference area. Also, the MDSC exploits the relationship between the Y, Cr and Cr components to reduce the number of access. If an area is present in the Y cache it then it is present in the Cb and Cr caches, thus only the address calculation is required to access Cr and Cr data.

Overall, the results show that software caches can exploit the data locality exhibited by MC. To obtain actual performance improvements, however, the number of access needs to be minimized. Furthermore, the MDSC allows reducing the number of accesses more than 1-dimensional cache organizations such as the IBM SC, thereby yielding higher overall performance.

CONCLUSION

In this article a Multidimensional Software Cache has been proposed for systems based on scratchpad memories such as the Cell processor. The objectives of the MDSC, next to increasing the programmability, are to exploit the data locality that cannot easily be exploited by hand-programmed DMAs, to reduce the DMA startup overhead by employing list DMAs instead of several sequential DMAs, and to minimize the number of cache accesses by using large, multidimensional blocks. Furthermore, the cache is indexed by the indices of the base element of the block rather than the memory address, which allows reducing the number of conflict misses. The proposed software cache organization has been evaluated for GLCM and H.264 MC, which is representative of many other multimedia kernels.

The GLCM uses indirect addressing, but because the difference between adjacent pixels is usually small, it exhibits data locality. Somewhat surprisingly, the MDSC configuration that yields the highest performance uses 1-dimensional blocks. However, the performance of two configurations was less than 1% lower. This indicates that in an organization where the two channels could be used simultaneously, the benefits of a 2-dimensional block would be more pronounced. Compared to the heavily optimized IBM software cache, the MDSC improves performance by 8%. The indexing function of the MDSC reduces the number of conflicts and accounts for the performance improvement.

For MC, first the data locality that it exhibits has been analyzed. This analysis shows that MC has a significant amount of data locality that could be exploited by a (software) cache. Then the data access pattern of MC has been evaluated to design a software cache that exploits it. The proposed software cache stores frame areas instead of blocks of consecutive memory locations. In other words, it uses 2-dimensional cache blocks. Enhancements have been proposed to reduce the number of accesses to the MDSC and its associated overhead.

For MC, the experimental results show that without tuning the software cache to the application, the performance degrades compared to an implementation that uses hand-programmed DMAs and does not attempt to exploit the data locality. This performance degradation is the result of the access overhead to the software cache to check for the presence of the desired data. The enhancements proposed in order to reduce the number of accesses to the MDSC achieve an average 65% performance improvement over the hand-programmed DMA implementation. For only one sequence (Riverbed), the MDSC did not attain a performance improvement over the DMA version. The reason for this is the lack of data locality in the Riverbed sequence. The software

cache overhead can be reduced by using information of the application's access characteristics to reduce the number of cache accesses. Compared to the IBM Cell software cache, the MDCS provides an improvement of 43%, averaged over all video sequences.

The performance of the MDSC reflects the machine organization used for this study. The Cell processor has two memory channels, but a single SPU cannot use both channels. The access latency does not change when one or two SPUs are accessing data simultaneously, as depicted in Figure 2. As future work, the impact of the MDSC on a processor that can communicate via multiple channels simultaneously will be studied. It is expected that this will increase the performance improvements achieved by the MDSC. Also, a lightweight hardware accelerator for software caches will be investigated that reduces the overhead of the MDSC without significantly increasing the area or power consumption.

REFERENCES

Alvarez, M., Salami, E., Ramirez, A., & Valero, M. (2007). HD-VideoBench: A benchmark for evaluating high definition digital video applications. In *Workload Characterization* (pp. 120-125). Washington, DC: IEEE Computer Society. DOI:10.1109/IISWC.2007.4362188

Azevedo, A., Zatt, B., Agostini, L., & Bampi, S. (2007). MoCHA: A bi-predictive motion compensation hardware for H.264/AVC decoder targeting HDTV. In *Circuits and Systems* (pp. 1617-1620). DOI: 10.1109/ISCAS.2007.378828

Balart, J., Gonzalez, M., Martorell, X., Ayguade, E., Sura, Z., & Chen, T. (2007). A novel asynchronous software cache implementation for the Cell-BE processor. In Proceedings of *Languages and Compilers for Parallel Computing. Urbana (Caracas, Venezuela)*, *IL*, 125–140. doi:. doi:10.1007/978-3-540-85261-2_9

Banakar, R., Steinke, S., Lee, B., Balakrishnan, M., & Marwedel, P. (2002). Scratchpad memory: A design alternative for cache on-chip memory in embedded systems. In *Proceedings of Hardware/Software Codesign*, Estes Park (pp. 73-78). New York: ACM. DOI:10.1145/774789.774805

Chen, T., Zhang, T., Sura, Z., & Tallada, M. G. (2008). Prefetching irregular references for software cache on Cell. In *Code Generation and Optimization* (pp. 155-164). New York: ACM. DOI: 10.1145/1356058.1356079

Edler, J., & Hill, M. D. (2010). *Dinero IV trace-driven uniprocessor cache simulator*. Retrieved January 28, 2010, from http://pages.cs.wisc.edu/~markhill/DineroIV/

Example Library API Reference. (2010). Retrieved January 28, 2010, from https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/3B6ED257EE6235D900257353006E0F6A/$file/SDK_Example_Library_API_v3.0.pdf

Gonzalez, M., Vujic, N., Martorell, X., Ayguade, E., Eichenberger, A. E., Chen, T., et al. (2008). Hybrid access-specific software cache techniques for the Cell BE architecture. In *Parallel Architectures and Compilation Techniques* (pp. 292-302). New York: ACM. DOI: 10.1145/1454115.1454156

Gschwind, M., Hofstee, H., Flachs, B., Hopkins, M., Watanabe, Y., & Yamazaki, T. (2006). Synergistic processing in Cell's multicore architecture. *IEEE Micro*, *26*(2), 10–24..doi:10.1109/MM.2006.41

Kahle, J., Day, M., Hofstee, H., Johns, C., Maeurer, T., & Shippy, D. (2005). Introduction to the Cell multiprocessor. *IBM Journal of Research and Development*, *49*(4), 589–604. doi:10.1147/rd.494.0589

Lee, J., Seo, S., Kim, C., Kim, J., Chun, P., Sura, Z., et al. (2008). COMIC: A coherent shared memory interface for Cell-BE. In *Parallel Architectures and Compilation Techniques* (pp. 303-314). New York: ACM. DOI: 10.1145/1454115.1454157

Power Architecture Version 2.02. (2010). Retrieved January 28, 2010, from http://www-106.ibm.com/developerworks/eserver/library/es-archguide-v2.html

Senthil, G., Gudla, S., & Baruah, P. K. (2008). Exploring software cache on the Cell BE processor. In *High Performance Computing* (p. 5).

Seo, S., Lee, J., & Sura, Z. (2009). Design and implementation of software-managed caches for multicores with local memory. In *High Performance Computer Architecture* (pp. 55-66). DOI:10.1109/HPCA.2009.4798237

Shahbahrami, A., Borodin, D., & Juurlink, B. (2008). *Comparison between color and texture features for image retrieval*. Circuits, Systems and Signal Processing.

Zatt, B., Azevedo, A., Agostini, L., Susin, A., & Bampi, S. (2007). Memory hierarchy targeting bi-predictive motion compensation for H.264/AVC decoder. In *VLSI* (pp. 445-446). Washington, DC: IEEE Computer Society. DOI:10.1109/ISVLSI.2007.64

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 4, edited by Seppo Virtanen, pp. 1-20, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 5
Automatic Generation of Memory Interfaces for ASIPs

David Kammler
RWTH Aachen University, Germany

Bastian Bauwens
RWTH Aachen University, Germany

Ernst Martin Witte
RWTH Aachen University, Germany

Gerd Ascheid
RWTH Aachen University, Germany

Anupam Chattopadhyay
RWTH Aachen University, Germany

Rainer Leupers
RWTH Aachen University, Germany

Heinrich Meyr
RWTH Aachen University, Germany

ABSTRACT

With the growing market for multi-processor system-on-chip (MPSoC) solutions, application-specific instruction-set processors (ASIPs) gain importance as they allow for a wide tradeoff between flexibility and efficiency in such a system. Their development is aided by architecture description languages (ADLs) supporting the automatic generation of architecture-specific tool sets as well as synthesizable register transfer level (RTL) implementations from a single architecture model. However, these generated implementations have to be manually adapted to the interfaces of dedicated memories or memory controllers, slowing down the design-space exploration regarding the memory architecture. To overcome this drawback, the authors extend RTL code generation from ADL models with the automatic generation of memory interfaces. This is accomplished by introducing a new abstract and versatile description format for memory interfaces and their timing protocols. The feasibility of this approach is demonstrated in real-life case studies, including a design space exploration for a banked memory system.

INTRODUCTION

Nowadays, the market for multi-processor system-on-chip (MPSoC) solutions is expanding dramatically. Often, the development of such an MPSoC includes the design of new processor architectures which are tailored to a particular application. A common technique to develop these application-specific instruction-set processors (ASIPs) is the use of an architecture description language (ADL) (Fauth, Van Praet, & Freericks, 1995; Hadjiyiannis, Hanono, & Devadas, 1997; Halambi et al., 1999; Hoffmann, Meyr, & Leupers, 2002; Leupers

DOI: 10.4018/978-1-4666-0912-9.ch005

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

& Marwedel, 1998; Mishra & Dutt, 2008; Rajesh & Moona, 1999). Present ADL tool suites enable the automatic generation of a fully synthesizable hardware description language (HDL) model of the architecture on register transfer level (RTL) (Basu & Moona, 2003; Hadjiyiannis & Devadas, 2003; Mishra, Kejariwal, & Dutt, 2003; Schliebusch, Meyr, & Leupers, 2007). Optimizations and standard processor features like debug mechanisms are supported by the automatic generation process making the generated model suitable for the final implementation. This decreases development time drastically, thereby allowing the designers to concentrate on the actual architectural features on the high abstraction level of ADLs rather than spending time on detailed modeling on RTL. Accesses to memories and busses are usually modeled as abstract function calls in order to e.g., transfer the address or data. Neither the definition of the interface pins nor a highly accurate description of the timing protocol are required on this level. However, both are mandatory for an accurate implementation on RTL. Adding this low-level information to the ADL model by specifying the pins and their usage directly is no option. This would cause an overhead and lower the abstraction level of the ADL model inadequately, making it complex, hard to maintain and prohibit a fast design-space exploration. Especially, attaching different memory types to the processor would result in many changes to the model, thus slowing down dramatically the exploration of memory architectures which is of special importance for the development of tailored ASIPs. Moreover, the speed of the generated processor simulator would decrease with the increasing detail of the model.

In this paper we present a solution for modeling memory interfaces in ADLs avoiding the previous drawbacks. The contribution of this paper is the development of a new abstract and versatile description format for memory interfaces, which is external to the actual ADL model of the processor core. Such a memory-interface description (MID) covers the definition of memory ports including their pins as well as the timing protocol including the usage of address/data busses with fixed timing or handshake mechanisms. Keeping the MID separate from the ADL model has several advantages:

Orthogonalization of processor model and memory-interface description: Separating the ADL model of the processor from the MID allows for a fast design-space exploration of the ASIP on ADL level and a pin-accurate implementation on RTL at the same time with the same ADL model.

Reuse of MIDs: Once specified, MIDs for a dedicated memory or memory system can be reused for other architectures or if several identical memories are attached to a single processor.

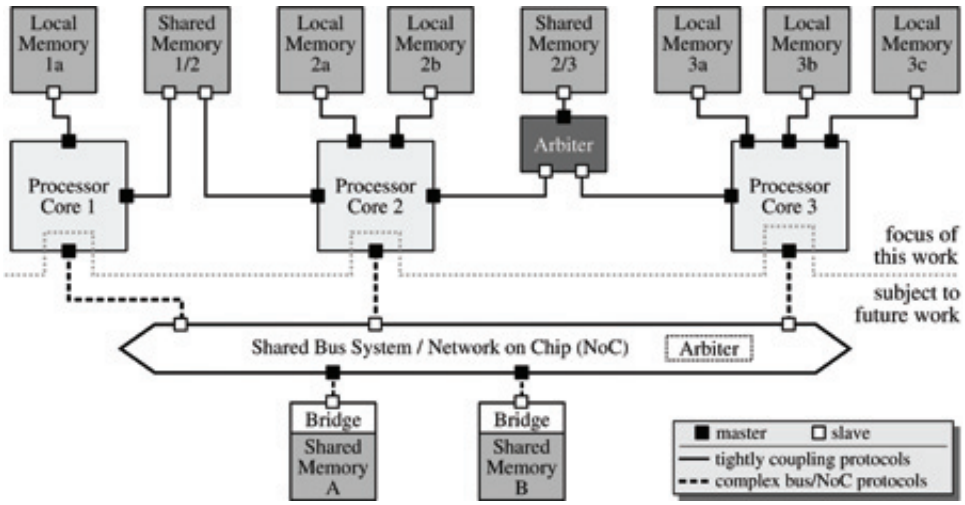
Rapid exploration of different memory architectures: Different memory architectures, including their physical parameters (e.g., area, timing, etc.), can be explored easily by selecting different MIDs for the HDL-code generation.

Maintaining simulator performance: Since the model is not extended with pin level details, the simulator performance is not affected. Nevertheless, the MID can still be used by other tools of the ADL tool suite if required.

Independence of memory vendors: Designers can more easily switch to other vendors with comparable memories by simply modifying or replacing MIDs. Changes to the ADL model in order to adapt the interface are not required.

The MID is defined so that the description of a wide range of real-world memory interfaces is possible. Even memory systems with cache hierarchy are supported as long as an HDL description is available. Focus is put on on-chip memories, which are tightly coupled with the processor cores, as this is usually the preferred option for fast hardware accelerators, such as most ASIPs. Slower accesses via more complex shared buses (e.g., AMBA) are not directly targeted in this work as depicted in Figure 1. This approach, however, still allows the connection to shared memories utilizing either more than one memory port or a dedicated arbiter as shown in the upper half of the figure.

Figure 1. System context of this work: tightly coupled memories in embedded systems



Our automatic memory interface generation is integrated into the LISA ADL development framework (Hoffmann, Meyr, & Leupers, 2002), which has been developed at the Institute for Integrated Signal Processing Systems (ISS), RWTH Aachen University and is now commercialized by CoWare (CoWare Inc., 2009).

The rest of the paper is organized as follows. After a discussion of the related work, an overview of features of existing memory interfaces and timing models for memory accesses is given. Then, the general approach for the automatic generation of memory interfaces is presented. The format of our MID is introduced afterwards. The way the generation process is integrated into our framework is shown and the structure of the generated hardware (HW) is discussed. Native support for banked memory systems is highlighted afterwards. Finally, several case studies are presented to show the feasibility and efficiency of our approach, followed by the summary and conclusion.

RELATED WORK

In the following we give a brief overview on those ADLs and tool suites reported to be used for HW implementation and focus on their capabilities regarding memory interfaces. A detailed discussion of prominent ADLs can be found in Mishra and Dutt (2008). ADLs can be classified into three categories regarding their nature of information: *structural*, *behavioral*, and *mixed*.

Structural ADLs describe architectural structures in terms of components and their connectivity. MIMOLA (Leupers & Marwedel, 1998) is an example for this class of ADLs. Since its abstraction level is close to RTL, it provides a close link to gate-level synthesis tools but as a result also requires detailed modeling of the memory interface and transfers via this interface.

Behavioral ADLs concentrate on the instruction-set architecture and neglect detailed HW structures. Among these languages, ISDL (Hadjiyiannis, Hanono, & Devadas, 1997) has been used for HDL-code generation (Hadjiyiannis & Devadas, 2003), however no information about the implementation of memory accesses in the resulting Verilog description is available.

Mixed ADLs contain structural information as well as details on the instruction set, like e.g. nML (Fauth, Van Praet, & Freericks, 1995). nML models are processed by the commercial HDL generator GO (Target Compiler Technologies, 2009).

Through application programming interfaces (APIs), users can plug in their own HDL implementations of the memory architecture (Goossens et al., 2006). However, no detailed information on this process is publicly available. Sim-nML (Rajesh & Moona, 1999), a derivative of nML, has also been used for generating synthesizable Verilog models (Basu & Moona, 2003). Here, the processor model is expected to work with a simple single-port external memory. EXPRESSION (Halambi et al., 1999) is another mixed ADL used for HDL-code generation (Mishra, Kejariwal, & Dutt, 2003) by exploiting predefined functional blocks provided as VHDL code. For the implementation of memory accesses, appropriate functions for several types of memories, such as SRAM or DRAM, are used. No information about the resulting VHDL code is available, as the library of predefined functional blocks cannot be obtained publicly. As described in Mishra, Mamidipaka, and Dutt (2004), EXPRESSION can also be used for the co-exploration of processor architectures and memory subsystems. The main goal is the optimization of the resulting compilers. However, no HDL code is generated based on this co-exploration. LISA (Hoffmann, Meyr, & Leupers, 2002) enables a similar exploration of the memory system (Braun et al., 2003) and has been used for the generation of optimized RTL implementations of processors (Schliebusch, Meyr, & Leupers, 2007). However, in the latter publication, the memory system exploration is not combined with the actual architecture implementation on RTL and the support for memory interfaces and protocols is limited to simple predefined interfaces with a fixed protocol.

This work aims at overcoming the described limitations of available solutions by extending HDL-code generation with support for arbitrary memory interfaces and protocols on a cycle- and pin-accurate level without sacrificing the high abstraction level of the ADL model.

MEMORY SYSTEMS

In order to enable the automatic generation of memory interfaces, present memory systems first have to be categorized based on their individual features and characteristics. To this end, over 25 different types of memories and memory controllers from 14 different vendors have been investigated and abstracted. This section first describes the most important aspects of available memory systems regarding their interface. The features that need to be supported by the HDL-code generation are identified. Moreover, a general timing model for memory accesses is introduced.

Memory Types and Features

In the following, most important memory types and features are discussed. The supported subset of features is summarized in Table 1.

Asynchronous/synchronous memories: The timing of *asynchronous* memories is specified by time intervals and relative signal changes. A clock is not required. For *synchronous* memories, all actions are aligned to a clock signal. The values of interface signals are latched internally at the (typically rising) edge of the clock. While this permits easy synchronization between a clocked processor architecture and a memory system, it also means that the reaction to a control signal cannot happen before the next clock cycle. In order to obtain a higher data throughput, some memory systems also utilize the falling edge of a clock signal to align data transfers (double data rate, DDR) (JEDEC, 2008). The automatic generation of memory interfaces focuses on the support of synchronous memory systems as these are prevalent in the on-chip embedded domain. Since all synchronous memories investigated (except those featuring DDR) utilize the rising clock edge, the automatic memory-interface generation currently focuses on this case. Therefore, a direct support of asynchronous memories or DDR is currently not available. These types of memories

Table 1. Memory types and features

Memory Type/Feature	Target for HDL-Code Generation
asynchronous memories	no (only with 3rd-party controller)
synchronous memories (rising clock edge)	yes
synchronous memories (falling clock edge)	no
static memories	yes
dynamic memories	no (only with 3rd-party controller)
double data rate (DDR)	no (only with 3rd-party controller)
pipelined memories	yes
burst access (definite length)	yes
burst access (indefinite length)	yes
data masking	yes
flushing of accesses	yes

can only be accessed via an appropriate 3rd-party memory controller.

Static/dynamic memories: *Static* memories retain their data contents as long as the memory is supplied with power. On the contrary, *dynamic* memories require a periodic refresh to be issued. In the embedded domain, dynamic memories are typically shared and accessed off-chip via shared buses. Therefore, refresh mechanisms are not targeted directly by the memory-interface generation. However, in the uncommon case it is necessary, dynamic memories could be connected using the memory-interface generation via appropriate 3rd-party controllers (e.g., OpenCores, 2009; Northwest Logic, 2009; HiTech Global LLC, 2009).

Pipelined memories: In case of a *pipelined* memory, a new access can be initiated before the previous access is finished. This behavior is e.g. shown by standard synchronous SRAMs and supported by the HDL-code generation.

Burst access: A common technique to increase data transfer rates of memories are *burst accesses*. During a burst access, multiple data words are automatically transferred in subsequent cycles without additional address transfers. The *burst length* of a memory access specifies the number of consecutive data words that are transferred during a burst access. The automatic memory-interface generation differentiates between two supported types of burst lengths. Memories can support a fixed set of *definite burst lengths* to select from. Other memory systems, especially when using memory controllers, can support the usage of an arbitrary burst length. Then, an interface control signal indicates the end of the burst access. This is referred to as *indefinite burst length*.

Data masking: When transferring a new data word to a memory system during a write access, it can be required to replace only specific bits/bytes at the specified word address. This feature is called *data masking* and is fully supported by the HDL-code generation.

Flushing of memory accesses: Some memories allow canceling pending memory accesses. In the following we will refer to this feature as *flushing* of memory accesses, which is supported by the HDL-code generation.

Terms and Definitions

The memory system communicates with connected HW components via an interface, which is made up of *pins*. In the scope of this paper, the term pin does not necessarily correspond to a package pin of an external memory. Since memories in SoCs are often placed on the same die as the processor

core, a pin may also correspond to internal signals that are used for the communication between the processor and the memory system. Note, that in this context a single pin can refer to signals with more than one bit.

A subset of the pins of the interface, which offers all the required functionality to perform an access to the memory system, is referred to as *port*. The interface of a memory system may consist of one or more ports, typically acting independently of each other. A *read port* only supports read accesses to a memory system, whereas a *write port* allows for write accesses only. If both types of accesses are supported by the same port, it is referred to as *read-write port*.

Memory-Access Timing Model

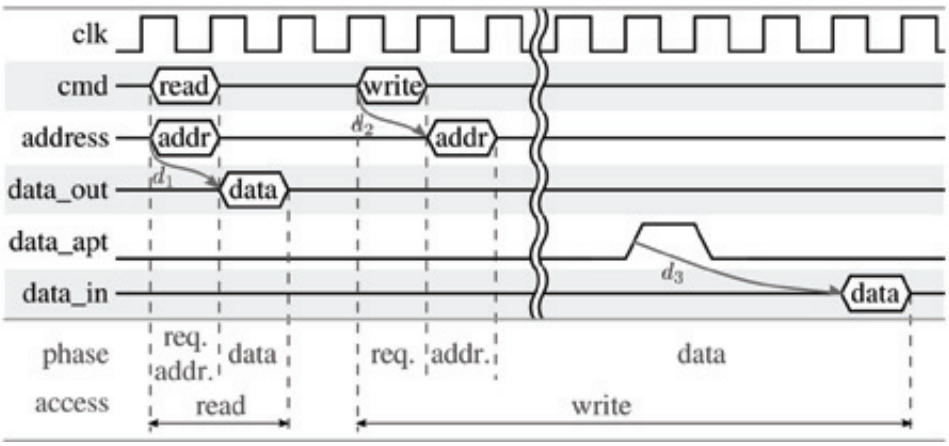
Independently of the actual specifications of a memory interface, each access to a memory system can be split into three different phases (Figure 2). During the *request phase*, the access is started and communicated to the memory system, typically by using control signals of the memory interface. The address to be accessed is forwarded to the memory system during the *address phase*. Finally, the data transfer is performed during the *data phase*. Individual phases may overlap, e.g., request and address may be provided at the same time (read access, Figure 2), or require additional delays between them (write access, Figure 2). In order to abstract from the timing varieties of different memory systems, each phase is divided into individual atomic elements as described in the following.

Assignments: An assignment represents the action of a signal or pin being set to a specified value. A typical assignment is the assertion of a control signal for a single cycle in order to request an access to a memory system. The transfer of address and write data are special types of assignments. Reading data from a memory system is also considered as an assignment, even if in this case data is read from the memory system, and thus the direction of the assignment is reversed.

Delay cycles: During a read or write access, memory systems can require a *fixed* number of delay cycles between individual assignments (arrows d_1 to d_3 in Figure 2). A particular delay always consists of the same number of cycles. Otherwise it would not be possible to determine the end of a delay without any handshake mechanisms.

Handshakes: Memory accesses may require a *variable* number of delay cycles. Handshake mechanisms are used to model such access protocols. A handshake can be described as a specific combination of values of interface signals used to inform about the status of a memory system.

Figure 2. Example for the memory access timing model



In Figure 2, for instance, a handshake (*data_apt*) is used to indicate that the memory accepts write data after a predefined delay d_3.

A VERSATILE MEMORY-INTERFACE APPROACH

On the ADL level, memory accesses are initiated via a dedicated API. This memory API covers all the features and aspects described in the previous section. It is based on the cycle-accurate API presented by Braun et al. (2003). Due to the large variety in the timing protocols, the API has been specifically designed in order to allow distributing the individual phases (request, address, data) of a memory transfer over the pipeline stages of an architecture. The phases are initiated by dedicated non-blocking API calls, because blocking calls would require implicit control flow generation in case of incomplete or unsuccessful execution. Therefore, the API calls immediately return values indicating the success or type of failure of the triggered operation. The designer has to model explicitly how the design behaves in case of a failure (e.g., stalling the pipeline in case of a failed data transfer). As a result, it is the designer's duty to invoke a certain access phase in the proper cycle (e.g., no read buffers are automatically inserted). Despite moderately increasing the design effort, this gives the designer the full freedom on how to deal with the control flow and how to tailor it to the specific needs of the target architecture.

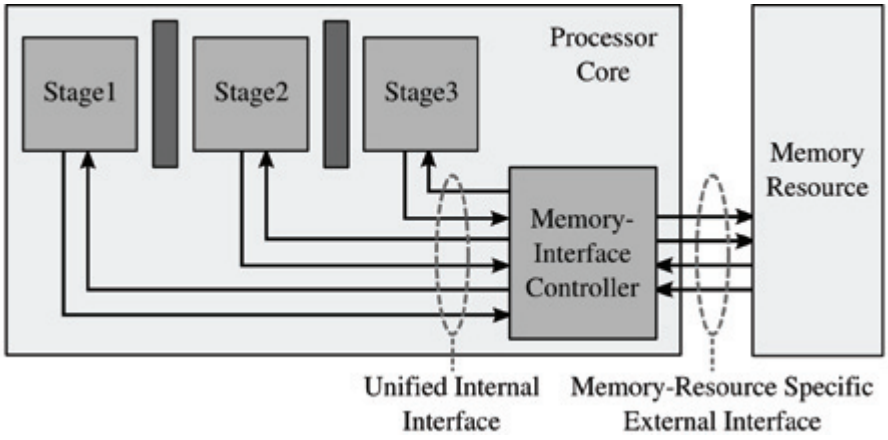
In order to map high-level accesses via the memory API to low-level HW structures, first a general set of signal types is identified, that enables to cover all memory API functionalities including request initialization, address transfer, data transfer (burst and non-burst), control and status propagation. Then, during the HDL-code generation process, the memory API function calls are used to derive the signals actually required in the specific case, which finally build up a *unified internal interface*. The 19 predefined signal types required in general are listed in Table 2 including their directions (from memory perspective). The bit widths of signals carrying address, data, burst length and data mask depend on the characteristics of the memory system. All other signals are single-bit signals. The specific signals which are finally selected for the interface depend on the type of the corresponding memory accesses in the ADL model. A detailed description of the actual protocol is omitted here as the mapping from the API to the internal interface signals is straightforward.

The unified internal interface, flexible and adapted to its individual usage, is not memory specific and does not allow to directly connect the target memory resource. Therefore, it needs to be mapped to a *memory-resource specific external interface*, which is defined by the pins and available ports of the targeted memory resource and independent of the memory accesses through-

Table 2. Internal interface signal types

Signal Type	Dir.	Signal Type	Dir.	Signal Type	Dir.
enable read	in	write data	in	busy	out
enable write	in	write data enable	in	address request	out
burst length	in	data mask	in	write data request	out
burst length enable	in	read data	out	read data valid	out
		confirm read	in	address timing	out
address	in			data timing	out
address enable	in	flush	in	last data	out

Figure 3. Mapping the ADL-derived internal interface to a memory-resource specific external interface



out the processor model. As depicted in Figure 3, this mapping is performed by a *memory-interface controller*, which encapsulates all the knowledge about the external interface. For each attached memory resource a dedicated memory-interface controller is instantiated.

The interface pins, which connect the memory to the memory-interface controller, are defined by their name, bit width and direction (input or output). Apart from these *interface pins*, the external interface may contain *clock, reset* and *unused pins*. The latter are required in order to define the complete interface of the memory system even if it is only used partially.

For HDL-code generation, the exact setup of pins and ports composing the external memory interface needs to be covered by a memory-interface description (MID), which is discussed in the following.

MEMORY-INTERFACE DESCRIPTION

In order to connect the generated HDL model of the processor core seamlessly to an existing memory resource, the interface and protocol need to be known explicitly during the HDL-code generation process. However, detailed information of pins, ports, timing protocol, etc. is not available on ADL level. Actually, the interface information is not architecture but memory dependent. Therefore, we propose to capture MIDs in dedicated external files not directly belonging to the ADL model and thereby orthogonalizing processor model and MID. Such MIDs need elements to capture two different kinds of information: *structural* and *behavioral elements*.

Structural MID Elements

Describing *structural* elements is straightforward since it requires only the definition of the pins and ports of a memory interface. As detailed in the previous section, for each pin a name, bit width and direction need to be specified. A port groups a set of pins that provides all the functionality required for an access to the memory system. However, additional semantic information also needs to be covered in the port description in order to identify the purpose of each pin. Therefore, previously defined interface pins or bit ranges of a pin can be declared as *data bus*, *address bus* or *data mask bus*. From the direction of the interface pin declared as data bus, the port type (read, write, read/write) can be derived directly. Apart from these specifications, the type (definite or indefinite) and the supported length of *bursts* need to be defined as well. A structural element

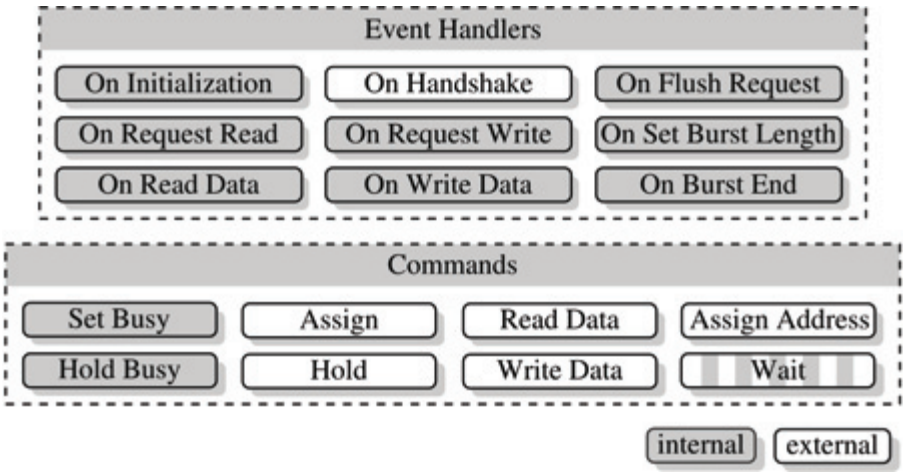
used by many interface descriptions on RTL is a VHDL generic or Verilog parameter as they offer a way to parameterize a component. Thus, the MID allows setting specific values for these *generics*. Additionally, certain resource parameters defined in the ADL model can be propagated via special generics to set e.g., the bit widths of the address and data buses.

Behavioral MID Elements

Behavioral elements of an MID are needed in order to describe the timing of actions and reactions of the memory system. This means for the memory-interface controller, that certain actions need to be triggered under certain conditions in order to map the internal interface protocol to that of the external interface and vice versa. In order to abstract the description of this mapping process, *events* and *commands* are used in the MID. Events describe certain conditions of the internal and external interface, that lead to the execution of actions encapsulated as a list of commands. It is worth mentioning that commands are not necessarily executed immediately when a certain event occurs. Depending on the timing protocol of the memory, a delayed execution of commands may be required. Both events and commands can be related to either the *internal* or the *external* interface. Internal events and commands need to cover all the functionality of the internal interface (and therefore also of the memory API on ADL level) in a rather abstract manner in order to allow for a convenient description without details of the internal-interface protocol. In contrast to that, external events and commands have to act at a pin-accurate level. Figure 4 shows all event handlers and commands that have been identified according to these criteria.

Details on the *internal events* are given in the following. *On initialization* events are triggered immediately after releasing the reset, providing a way to implement an initialization phase, if required by the target memory system. *On request read/write* are events occurring during the request phase of a memory access, while *on read/write data* events correspond to the data phase. The *on flush request* event can be used to cancel pending memory requests. In order to react on a change of the length of burst transfers the *on set burst length* event is provided. With the *on burst end* event the end of a burst transfer can be detected. This is of special importance for memories offering an indefinite burst length. In order to enable an individual and independent treatment of different memory ports (if there is more than one),

Figure 4. List of event handlers and commands



event handlers can be bound to a previously defined port.

External events, in contrast to internal events, need to capture conditions on a lower level, namely reacting on changes of pin values of the external interface. For this purpose only a single event handler is specified, the *on handshake* event. In order to allow for complex conditions, handshakes contain several *handshake criteria* that can be combined by logical conjunction or disjunction. The condition of a *handshake criterion* can be a certain value of a pin or bit range of a pin. Additionally, *handshake criteria* can further contain a list of criteria or even refer to other handshakes enabling the description of complex nested conditions.

Internal commands describe actions to feed back information of the current memory status to the processor core. In most cases the status information can be generated automatically within the memory-interface controller by knowing the exact protocol and therefore timing of the external interface. However, one piece of status information often depends directly on external information from the memory system and cannot be generated automatically: the *busy status*. Especially for dynamic memories encapsulated by a memory controller or other complex memory systems, the status of the memory determines whether new requests can be accepted or not. In order to deal with this issue, two commands are available: *set busy* and *hold busy*. Both types differ in their temporal impact on the value of the internal control signal. The *set busy* command assigns a value for "busy" or "not busy" for a single cycle, while the *hold busy* command modifies its default value which is held until changed by another *hold busy* command.

External commands refer to value assignments to signals of the external interface. The value of an external interface pin can be changed by using the command types *assign* and *hold*, where the *assign* command assigns a specific value to a pin for a single cycle and the *hold* command modifies its default value. The commands *assign address*, *read data* and *write data* trigger more direct interactions between the external interface and the internal interface. While *assign address* triggers the assignment of the address provided by the processor to the memory, the *read data* and *write data* commands specify the data transfers during the data phase of memory accesses. Due to its similarity to *assign address* and *write data*, the *read data* command is classified as external command although the actual direction of data is inverted.

Wait commands belong to both classes, internal and external commands, as they describe delays by means of number of clock cycles that can affect both sides. Note, that due to the usage of *wait* commands an order within the command list of an event handler is defined.

Due to the hierarchical nature of an MID (especially for the description of handshakes), an XML-based document is used. Approximately 30 lines of XML code are required to describe synchronous pipelined memories, while less than 100 lines are sufficient for more complex memory interfaces including burst support and an initialization phase. A minimum basic yet fully functional example of an MID file for a single-port SSRAM with one cycle latency is given in Figure 5.

Limitations of the Memory-Interface Description

The current list of events and commands is focused on low-level protocols typically used for the access to embedded memories. More complex protocols used by shared buses (e.g., AMBA) require extensions. A major extension is the propagation of configuration commands from the pipeline over the internal to the external interface and complex responses in the opposite direction. However, simple handshake-based protocols connecting e.g., to an arbiter granting access to a shared memory can already be supported, as long as a binary *busy* response is sufficient.

Figure 5. MID file example for a single-port SSRAM

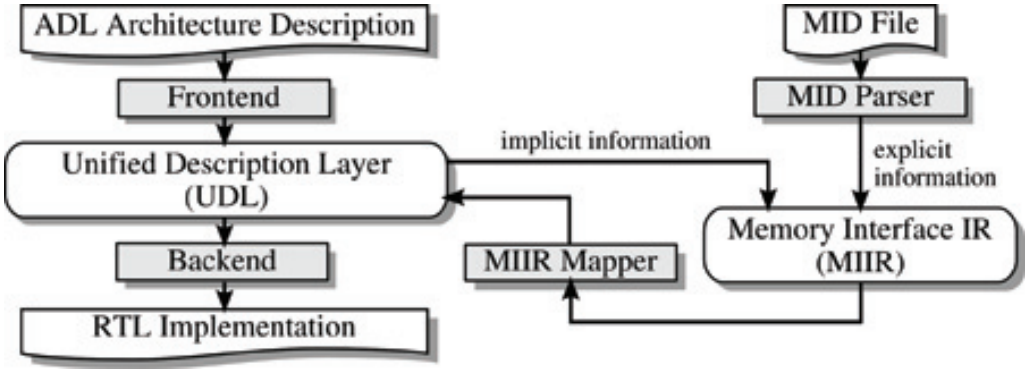
```
<MemoryInterface ComponentName="SyncMem">
    <ClockPin Name="clk"/>
    <ResetPin Name="rst"/>
    <Pin Name="rw_addr"  Dir="In"  Width="12"/>
    <Pin Name="data_in"  Dir="In"  Width="32"/>
    <Pin Name="data_out" Dir="Out" Width="32"/>
    <Pin Name="ew"       Dir="In"  Width="1"  Default="0"/>
    <Port Name="rw_port">
        <AddressBusPin Pin="rw_addr" />
        <DataBusPin    Pin="data_in" />
        <DataBusPin    Pin="data_out"/>
    </Port>
    <OnRequestWrite Port="rw_port">
        <Assign        Pin="ew"  Value="1"/>
        <AssignAddress Port="rw_port"/>
        <WriteData     Port="rw_port"/>
    </OnRequestWrite>
    <OnRequestRead Port="rw_port">
        <AssignAddress Port="rw_port"/>
        <Wait          Cycles="1"    />
        <ReadData      Port="rw_port"/>
    </OnRequestRead>
</MemoryInterface>
```

For evaluation purposes, we extended the MID elements with preliminary structural elements for a *configuration command bus*, a *configuration value bus* and a *status bus*. With the appropriate commands to access these buses and the additional event handlers *on provide address*, *on provide write data*, *on confirm read* and *on set bus configuration*, simple OCP-like protocols can already be supported. Prototypes of such bus configurations show promising results. Hence, the proposed approach appears to be also appropriate for more complex bus topologies. However, this is beyond the scope of this paper.

AUTOMATIC MEMORY-INTERFACE GENERATION

The HDL-code generation framework presented by Schliebusch, Meyr, and Leupers (2007) evolves around an intermediate representation, named unified description layer (UDL), which operates on an abstraction level between that of ADLs and RTL. According to common software design techniques, the ADL code is parsed via a frontend and RTL code can be produced in different HDLs by dedicated backends. The automatic memory-interface generation is integrated in the HDL-code generation process via the UDL structures. First, all the required information needs to be gathered to build up the structure of the memory-interface controller. On the one hand, the MID file is parsed to generate the *explicit* information about the memory structure. On the other hand, the UDL contains the memory-access information from the processor. This information *implicitly* refers to the internal interface and scope of operation of the memory-interface controller. The implicit and explicit information are combined to generate the memory-interface intermediate representation (MIIR). During this process, memory accesses are automatically mapped to the available memory ports. Finally, the MIIR is mapped to components of the UDL and thereby integrated into the existing framework. The whole process is shown in Figure 6.

Figure 6. HDL-code generation flow



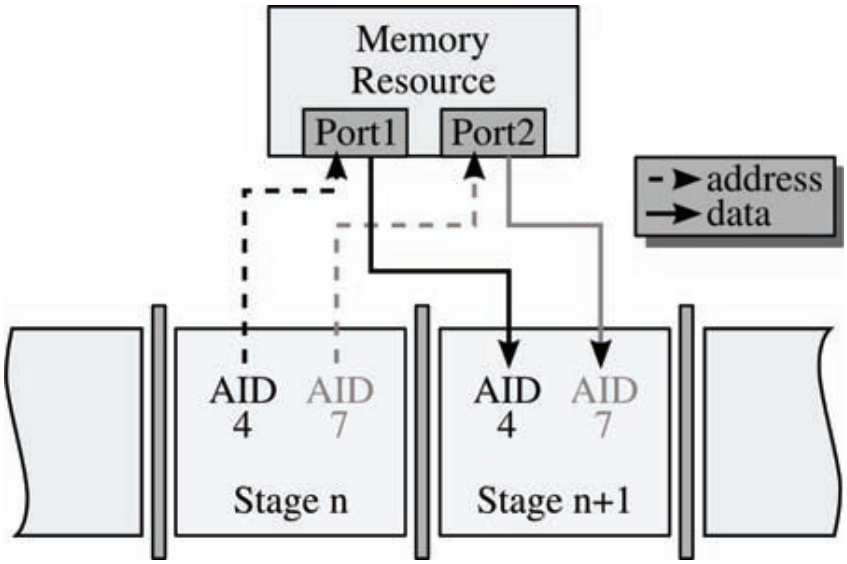
Mapping Memory Accesses to Memory Ports

During the HDL-code generation process, the individual memory accesses to a memory resource have to be assigned to the available *N* memory ports. This process is referred to as *port mapping*. In our approach, the port mapping is performed automatically, which is a non-trivial process in case of more than one physical memory port. The whole process can be separated into four steps:

i) *Access identification*: Accesses split over the pipeline into separate access phases have to be identified. This information can only be given by the designer. Thus, all memory API calls for the different access phases of a single access need to provide a unique access identity number (AID). Figure 7 depicts a simple example for two read accesses with separated address and data phases.

ii) *Consistency checks:* Checking the individual memory accesses for consistency focuses on tool usability. Particularly, the availability

Figure 7. Exemplary usage of access identity numbers (AIDs)



of AIDs enables the detection of potential modeling mistakes, for instance if not all required phases of an access are implemented in the processor model.

iii) *Identification of concurrent accesses:* Accesses have to be mapped to the available number of memory ports. For solving this problem, it is important to obtain the knowledge about possible concurrency (*conflicts*) of the various accesses and their phases. This knowledge can be mapped onto an *access-phase conflict graph* with vertices representing memory accesses and edges representing possible concurrent execution (Figure 8a). As described by Schliebusch, Meyr, & Leupers (2007), this access-phase conflict graph can be efficiently extracted on UDL level. Based on the resulting graph, an *access conflict graph G* can be derived (Figure 8b).

iv) *Assigning accesses to physical memory ports:* Assigning accesses to a single type of ports (read, write or read/write) corresponds to the *Maximum Induced N-Partite Subgraph* problem. A solution minimizes the number of conflicts between accesses on one memory port. However, the port types cannot be mapped separately, because the read/write type overlaps with both read and write types. For this reason, one single graph *G'* is used for the mapping of all ports (regardless of their types). *G'* results from extending *G* by a special fully connected subgraph *P* which consists of conflicting nodes representing the available memory ports (Figure 8c). Furthermore, *G'* is extended by conflict edges between read ports in *P* and write accesses as well as conflict edges between write ports in *P* and read accesses (Figure 8d). With *G'*, the port mapping again corresponds to the *Maximum Induced N-Partite Subgraph* problem with the additional constraint, that the induced subgraph *must* contain the edges added during the creation of *G'*. First, the graph *G'* is colored to get the minimum number of conflict-free subgraphs. Thus, all accesses of one subgraph can be mapped to the same memory port. In the best case, *G'* is already N-partite, which guarantees a conflict free port mapping. Otherwise, a heuristic is applied, where the subgraphs (without port nodes) are assigned to memory ports one by one in descending order of the number of vertices (accesses). The port for a particular subgraph is chosen by the minimization of

Figure 8. Mapping the access-phase conflict graph to an access conflict graph and an access/port conflict graph

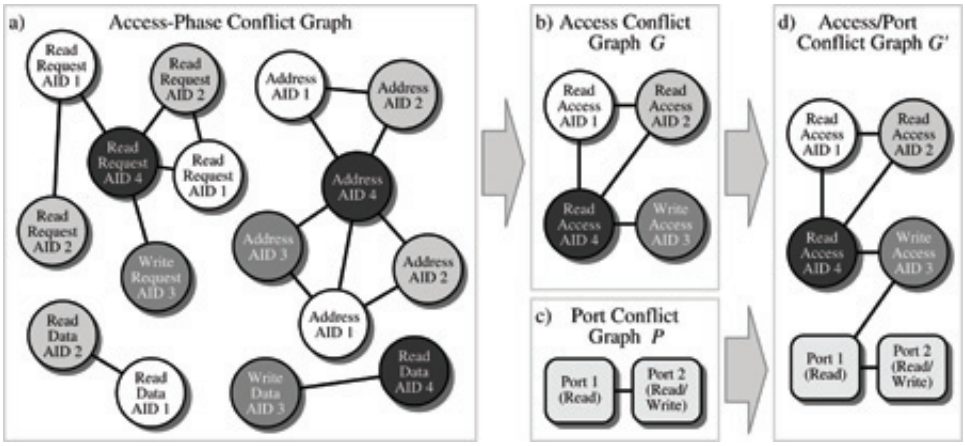
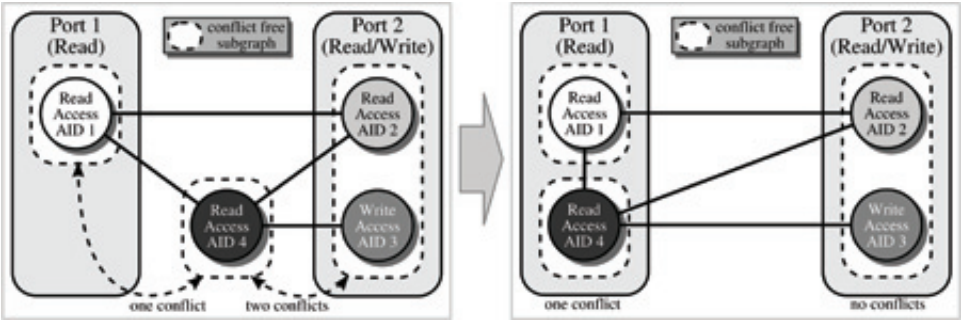


Figure 9. Example for the heuristic utilized during port mapping



conflicts between the current subgraph and all subgraphs already assigned to a port in previous steps. Figure 9 shows an example for such a step mapping the access with AID 4 to port 1.

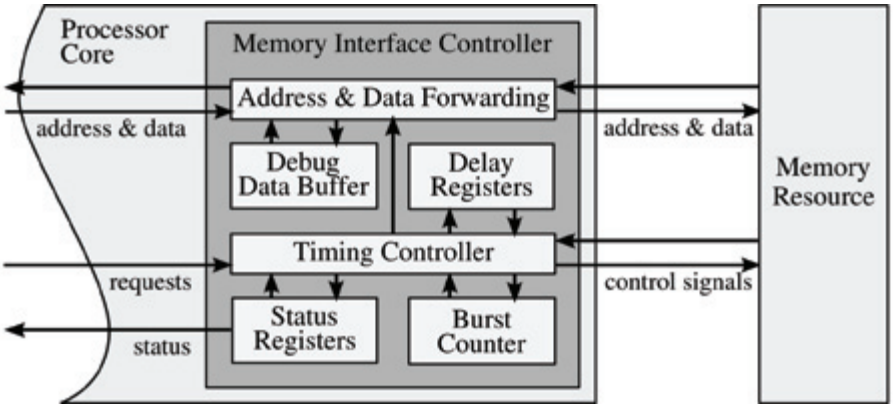
During HDL-code generation the designer is informed about the final port mapping. In the rare case that the mapping process is not successful, i.e., accesses are mapped to a wrong port, the process can be manually forced to map certain accesses to the same port. This can simply be achieved by using the same AID for accesses that ought to be mapped to the same port.

Mapping MIIR Structures to UDL Structures

The main component to be instantiated in the UDL is the memory-interface controller. The targeted structure of this controller is depicted in Figure 10. The behavioral elements of an MID allow to describe the behavior of the memory interface in a functional way without details about HW structures. Therefore, a direct mapping of the MIIR is not possible. In the following we describe how MIIR structures are mapped to the interface controller.

The *timing controller* and the *delay registers* build up the actual state machine of the memory-interface controller. The corresponding parts of the MID are described with event handlers and command lists. A command list can contain *wait*

Figure 10. Memory-interface controller implementation



commands to implement delay cycles which can be realized in the following two ways.

i) *Shift registers:* Shift registers implement a timer by bitwise shifting from MSB to LSB every cycle. The timer is initialized by setting the MSB to 1. When the 1 reaches a certain position of the register, dedicated actions can be triggered. This implementation allows to pipeline the individual commands of an event handler as delayed commands remain scheduled even if the event is triggered again before their execution. This is of special importance for pipelined memories.
ii) *Counter Registers:* Counters require less register cells and a single incrementer for implementation making them especially attractive for long delays. However, they do not allow pipelined scheduling. For triggering delay-based events, comparators are needed.

In principle, all commands except *wait* can be directly translated to signal assignments on RTL, which are executed conditionally. The resulting if-blocks make up the actual *timing controller* and the *address* and *data forwarding*. In order to observe the actual execution condition of a command, the command list of an event handler is first divided by its *wait* commands into *command collections* with the same execution condition. The conditions refer either directly to the event or to a check of the counter status in case of delayed commands. Thereby, each command collection is mapped to a single if-block of the *timing controller* description.

Some features supported by the HDL-code generation require the implementation of additional components, like a *burst counter* for keeping control of burst accesses and *status registers* to propagate status information to the internal interface.

Support for Other Processor Features

A special component is the *debug data buffer*, which is instantiated when the processor core is generated with debug mechanism (see Schliebusch, Meyr, and Leupers (2007) for details). This mechanism can cause the whole core to pause execution. As memories typically do not support such a feature directly and continue their operation in this situation, incoming data may need to be buffered in order not to get lost. For non-pipelined memories, the required buffer size is one data word per memory port. In case of pipelined memories the size of the buffer needs to be scaled linearly with the pipeline depth. This example shows another general advantage of *automatic* memory-interface generation, as the functionality of other automatic features of the HDL-code generator can be maintained without any manual code modifications.

Dynamic Mapping of Memory Accesses

The presented ADL-based generation of memory interfaces utilizing MIDs already facilitates the design-space exploration for ASIPs significantly. Different types of memories can be easily connected to the same processor core without modification of the actual ADL model by simply selecting different MIDs for the HDL-code generation process. However, a limitation of this approach results from the fact that memory accesses are statically mapped to memory ports during HDL-code generation of the architecture. This limitation appears to be irrelevant for most memory systems. However, if memory accesses need to be distributed at runtime to different memories, it can slow down the design-space exploration, because this distribution needs to be modeled explicitly. The utilization of memory banks as a throughput increasing alternative to multi-port memories is a good example for this kind of

dynamic distribution. Assuming that the memory banks assemble a continuous addressable memory space, the actual bank to be accessed depends on the address which is only known at runtime. Like multi-port memories, these systems require conflict resolution, because parallel accesses can only be executed if they belong to different banks. However, these conflicts cannot be resolved at design time due to their dependency on the application. Conflicts, which are solvable at compile-time, can be resolved in software. However, conflicts occurring at runtime can be very costly or even impossible to solve in software. Therefore, we introduce an extension of our tool flow to address runtime conflict resolution by means of a hardware unit. By automatically generating this unit, we avoid the tedious explicit description of the conflict resolution in the ADL model and enable an independent implementation of the processor pipeline and the banked memory system.

The Memory-Access Unit

The automatically generated memory-access unit (MAU) is placed between the memory system and the pipeline. Memory accesses are requested concurrently from the pipeline resulting in multiple independent read or write connections (unidirectional) between the pipeline and the MAU. The MAU distributes memory accesses from the pipeline to the addressed memory bank. This distribution can be configured to operate on any combination of bits selected from the actual address. This means for a memory system with B banks, that e.g. taking the MSBs for bank selection would result in a segmentation of B equally sized spaces. Contrariwise, when considering the LSBs, the address space is word-wise interleaved over the banks such that the actual banks are addressed by taking the address modulo B. The advantage of this approach is an efficient selection of the addressed memory bank. However, it limits B to be a power of 2, if the whole address space should be utilized.

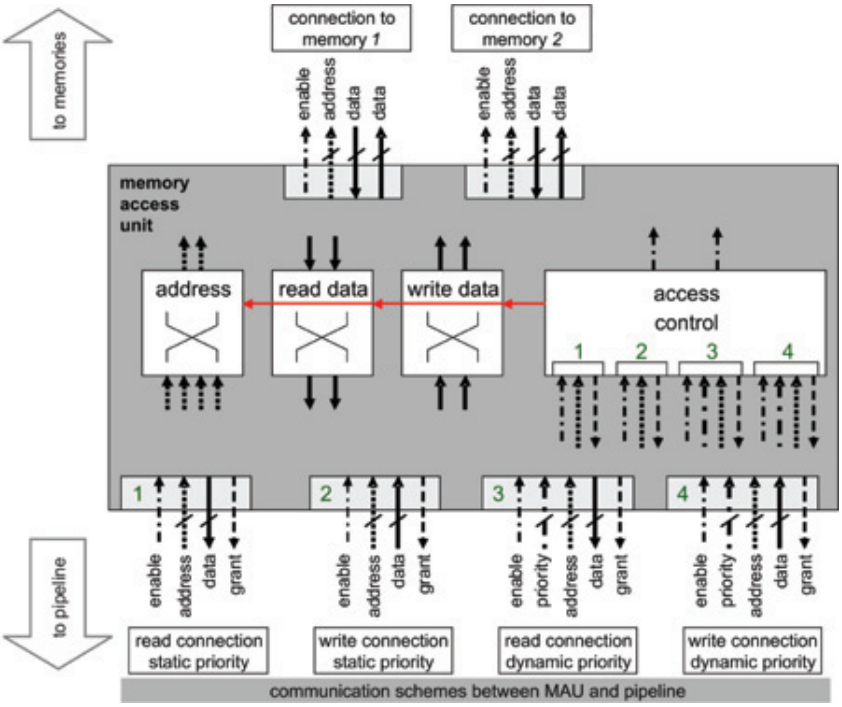
Access conflicts are resolved by granting or rejecting individual accesses. Therefore, a simple handshaking protocol is used between the pipeline and the MAU, which is able to confirm a request within the same cycle in order not to cause additional delays. The actual decision about granting an access can be based on static priorities of accesses (configured by the designer in a sorted list of AIDs) or dynamic priorities indicated by additional communication lines. In the latter case, the designer is responsible for setting the correct dynamic priorities in the ADL model. Figure 11 shows the interconnect of an exemplary MAU utilizing two memory banks.

One advantage of this mechanism is that for the core the memory space remains unchanged, regardless of the actual number and structure of memory banks. This enables an independent implementation and modification of the base architecture and the banked memory system. Therefore, the designer has full control over the architecture and its data and control flow when modeling the maximum number of parallel and independent accesses explicitly. No implicit control flow or dependencies are inserted. Any reasonable number of memory banks can be attached to the core without modifications to the ADL model. This enables a rapid exploration of various configurations of memory banks. Of course, it is still the designer's task to utilize the inherent parallelism of the banked system.

The MAU implementation strongly depends on the actual number and type of accesses asserted from the pipeline and the attached memories. Therefore, we included the automatic generation of MAUs on RTL in our framework. Additionally, functional SystemC simulation models of the MAU are available in order to ease the design process on system level.

The support for a rapid exploration is particularly important since the costs (e.g., area, signal propagation delay) for the MAU and the memory subsystem naturally vary with changing parameters such as the number of memory

Figure 11. Interconnect of a memory-access unit (MAU) with data segmentation on two memory banks



banks, memory/silicon technology, etc. In our experiments, the costs of this hardware-based approach lie mainly in a moderate area increase. Depending on the use case, MAUs have shown to consume an area of 1 to 4 kGates. The critical path is compromised for rather complex setups only. In the case-study section, we show the results of such a design-space exploration of an ASIP for cryptographic pairings taking advantage of the presented approach.

CASE STUDIES

In order to show the flexibility and efficiency of our approach, we first extended a simple RISC core with various data memory interfaces described in MIDs. Please note, that certain features of an interface (e.g., burst transfer) also need to be reflected by the instruction set of the architecture. Table 3 shows the required code sizes and area consumption of four different test cases for generated controllers between the internal and the external memory interface. The first test case utilizes a typical synchronous SRAM with a single read-write port and one cycle latency. For the second test case we used a simple handshake-based protocol allowing the core to connect to an arbiter granting access to a shared memory. The "full featured" test cases describe virtual memory interfaces introduced to test all other supported features of our approach like write mask, initialization phase and burst access. Two versions of this interface are listed, one with five different definite burst lengths and one with indefinite burst length. As can be seen, the size of an MID file is factor 5-10 smaller than the corresponding VHDL implementation of the generated interface controller. Aside from the code-size advantage of a single MID file, it is important to note, that–opposed to manually written memory interfaces–the interface controller can be regenerated automatically when the ASIP architecture (and therefore the internal interface) changes. The area consumption for any

Table 3. Generated HDL-code size and area consumption of exemplary memory-interface controllers

Interface Type	MID-file size (lines)	HDL-code size (lines)	Size of con-troller (gates)
single-port SSRAM	31	290	232
simple handshake	40	329	228
full featured, def. burst	87	700	824
full featured, indef. burst	67	705	722

of the test cases is less than one kGate. Synthesis results have been obtained with Synopsys Design Compiler using a 130 nm standard cell library with a supply voltage of 1.2 V. The maximum frequency for all RISC variants is 450 MHz.

Besides these synthetic case studies, two real-life ASIP examples from different domains are introduced in the following in order to show the effectiveness of our approach. While the first example comes from the image-processing domain and utilizes a tailored memory architecture with 4 dedicated memories, the second example shows an ASIP from the cryptography domain taking advantage of the previously introduced data-segmentation technique.

An ASIP for Retinex Image Processing

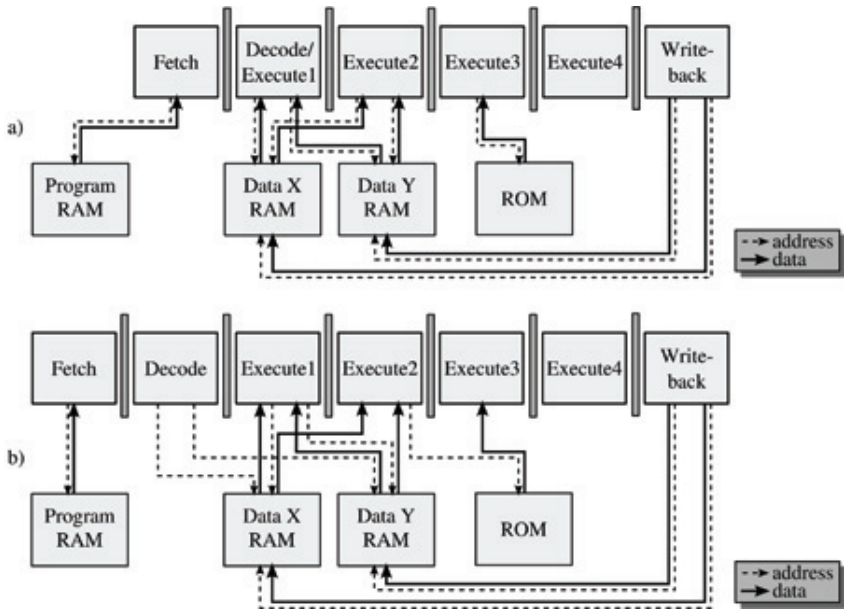
Retinex algorithms aim at the enhancement of images and videos taken under bad lighting conditions considering aspects of the human eye physiology. This target application imposes strict constraints on the data processing. Therefore, the ASIP developed by Saponara et al. (2007) required a careful co-exploration of data path, control path and memory interface. The ASIP is designed with 1 program, 2 data memories (X and Y) and a ROM. It uses complex memory-to-memory instructions asserting up to 3 concurrent memory accesses in order to speed up data processing. To simplify matters, the development initially started using ideal memories without delay cycles (Figure 12a) although SSRAMs with one delay cycle were targeted. After the initial design-space exploration for the data path, the memory accesses have been modeled for the real, cycle accurate SSRAMs. This required the separation of address and data phases for read accesses and their distribution to different pipeline stages, entailing an extension of the pipeline from 6 to 7 stages (Figure 12b). The proposed framework enabled to apply and verify these fundamental architectural modifications in a few hours only, whereas a manual implementation on RTL would have required several days.

An ASIP for Cryptographic Pairings at high Security Levels

The ASIP presented by Kammler et al. (2009) aims at the acceleration of high-security cryptographic pairings. Applications of such pairings include for example identity-based encryption and short digital signatures. A survey of various applications that would be impossible or very hard to realize without pairings is presented by Boneh (2007). Cryptographic pairings are based on elliptic curves, which entail calculations on large word width, if high security is targeted. In this publication, 256-bit Barreto-Naehrig curves (Barreto & Naehrig, 2006) are used resulting in numerous modular 256-bit multiplications and additions in order to compute a pairing. Therefore, this ASIP has been equipped with special scalable units for fast modular multiplications and additions on large word widths. These units perform the arithmetic in multiple cycles while performing 32-bit memory accesses in parallel to fetch the 256-bit operands word by word. Initial implementations revealed a bottleneck in the memory throughput.

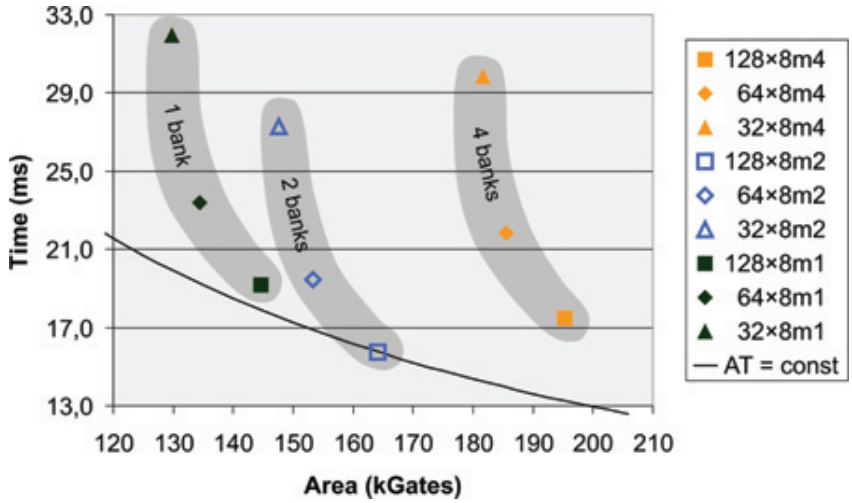
Due to the regular structure of the large operands, the data memory has been segmented into banks to increase the memory throughput. By automatically generating MAUs, different design

Figure 12. Memory architecture of the Retinex ASIP a) with ideal memories b) with real SSRAMs



variants with various memory bank configurations could be explored rapidly. Figure 13 shows the area consumption and the execution time for nine design variants executing the "Optimal Ate" pairing (Vercauteren, 2008). The variants naming convention follows the structure <multiplier bit width>m<number of memory banks>. The usage of a MAU with dynamic conflict resolution for concurrent memory accesses introduces a deterministic runtime degradation compared to an ideal conflict free scenario. However, for the investigated cryptographical applications, the observed runtime increase is less than 2% for a worst case collision scenario and thus considered acceptable.

Figure 13. Area-time trade-off for different ASIP variants (Optimal Ate pairing)



All syntheses have been performed with Synopsys Design Compiler utilizing a 130 nm standard cell library operating at a supply voltage of 1.2 V. Area numbers include the data-memory banks. As can be observed from the figure, the product of area and time (AT product) decreases with increasing size of the modular-multiplier unit. The exploration of different numbers of memory banks nicely shows the need for the rapid design space exploration: The design with the overall smallest AT product is 128×8m2–in spite of the area overhead due to two data memories. Up to this point, the critical path delay is not determined by the MAU, varying only slightly between 2.96 ns and 3.09 ns for the design points with one and two memory banks. However, increasing the number of memory banks further to four leads to an increased overall execution time due to a higher critical path delay of 3.69 ns. This is a result of the required distribution of accesses with dynamic priorities to four banks which lead to a decreased maximum clock frequency.

This case study shows that the memory system of ASIPs needs to be considered carefully and can limit the efficiency of an ASIP design. It is important to note, that this exploration result significantly depends on the used memory/silicon technology. Our approach enables the designer to quickly explore different alternatives and scale the memory throughput according to his needs.

CONCLUSION

In this paper we presented a new abstract and versatile description for memory interfaces covering the definition of memory ports including their pins as well as the timing protocol. It has proven to support a wide range of real-world interfaces for tightly coupled on-chip memories. Special support of banked memory systems eases their exploration. Our automatic memory-interface generation for HDL-code generation is integrated into the LISA ADL development framework (Hoffmann, Meyr, & Leupers, 2002).

Future work considers an extension of support for tool-aided exploration of memory systems with increased data throughput. Further investigations focus on the identification of additional events and commands in order to describe not only memory but also shared bus interfaces.

ACKNOWLEDGMENT

The Retinex ASIP design has been developed with M. Cassiano from the University of Pisa, now with ST Microelectronics, Italy, as well as S. Saponara and L. Fanucci from the University of Pisa. The authors would like to give thanks for their excellent cooperation. The authors would also like to thank D. Zhang, H. Scharwaechter, M. Langenberg, and D. Auras from the RWTH Aachen University as well as P. Schwabe from the Eindhoven University of Technology for their contributions to the ASIP design for cryptographic pairings.

REFERENCES

Barreto, P. S. L. M., & Naehrig, M. (2006). Pairing-friendly elliptic curves of prime order. In

Basu, S., & Moona, R. (2003). High level synthesis from Sim-nML processor models. In *Proceedings on the 16th International Conference on VLSI Design* (pp. 255260). Washington, DC: IEEE.

Boneh, D. (2007). A brief look at pairings based cryptography. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science – FOCS 2007* (pp. 19-26). Washington, DC: IEEE.

Braun, G., Wieferink, A., Schliebusch, O., Leupers, R., Meyr, H., & Nohl, A. (2003). Processor/memory co-exploration on multiple abstraction levels. In *Proceedings of the Design Automation and Test in Europe Conference and Exhibition* (pp. 966-971).

CoWare Inc. (2009). *CoWare Processor Designer*. Retrieved November 14, 2009, from http://www.coware.com/products/processordesigner.php

Fauth, A., Van Praet, J., & Freericks, M. (1995). Describing instruction set processors using nML. In *Proceedings of the European Design and Test Conference* (pp. 503507). Washington, DC: IEEE.

Goossens, G., Lanneer, D., Geurts, W., & Van Praet, J. (2006). Design of ASIPs in multi-processor SoCs using the Chess/Checkers retargetable tool suite. In *Proceedings of the International Symposium on System-onChip* (pp. 14). Washington, DC: IEEE.

Hadjiyiannis, G., & Devadas, S. (2003). Techniques for accurate performance evaluation in architecture exploration. *IEEE Transactions on Very Large Scale Integration (VLSI). Systems*, *11*(4), 601615.

Hadjiyiannis, G., Hanono, S., & Devadas, S. (1997). ISDL: An instruction set description language for retargetability. In *Proceedings of the 34th Design Automation Conference* (pp. 299302). Washington, DC: IEEE.

Halambi, A., Grun, P., Ganesh, V., Khare, A., Dutt, N., & Nicolau, A. (1999). EXPRESSION: a language for architecture exploration through compiler/simulator retargetability. In D. Borrione & R. Ernst (Eds.), *Proceedings of the Design Automation and Test in Europe Conference and Exhibition* (pp. 485490). Washington, DC: IEEE.

HiTech Global LLC. (2009). *DDR SDRAM Controller IP Core, Product Specification*. Retrieved November 14, 2009, from http://www.hitech-global.com/ipcores/ddrsdram.htm

Hoffmann, A., Meyr, H., & Leupers, R. (2002). *Architecture Exploration for Embedded Processors with LISA*. Boston: Kluwer Academic Publishers.

JEDEC. (2008). *JEDEC standard JESD79-3C: DDR3 SDRAM*. Arlington: JEDEC Solid State Technology Association.

Kammler, D., Zhang, D., Schwabe, P., Scharwaechter, H., Langenberg, M., & Auras, D. (2009). Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves. In Clavier, C., & Gaj, K. (Eds.), *Cryptographic Hardware and Embedded Systems - CHES 2009* (pp. 254–271). Heidelberg, Germany: Springer. doi:10.1007/978-3-642-04138-9_19

Leupers, R., & Marwedel, P. (1998). Retargetable code generation based on structural processor description. *Design Automation for Embedded Systems*, *3*(1), 75108. doi:10.1023/A:1008807631619

Mishra, P., & Dutt, N. (Eds.). (2008). *Processor Description Languages*. Burlington, VT: Morgan Kaufmann Publishers.

Mishra, P., Kejariwal, A., & Dutt, N. (2003). Rapid exploration of pipelined processors through automatic generation of synthesizable RTL models. In *Proceedings of the 14th IEEE International Workshop on Rapid Systems Prototyping* (pp. 226232). Washington, DC: IEEE.

Mishra, P., Mamidipaka, M., & Dutt, N. (2004). Processor-memory coexploration using an architecture description language. *ACM Transactions on Embedded Computing Systems*, *3*(1), 140162. doi:10.1145/972627.972634

Northwest Logic. (2009). *DDR SDRAM Controller Core, Product Specification*. Retrieved November 14, 2009, from http://www.nwlogic.com/docs/DDR_SDRAM_Controller_Core.pdf

OpenCores. (2009). *OpenCores DDR SDRAM Controller Core Project*. Retrieved November 14, 2009, from http://www.opencores.org/project,ddr_sdr

Preneel, B., & Tavares, S. (Eds.). (2005). *Selected Areas in Cryptography - SAC* (pp. 319–331). Heidelberg, Germany: Springer.

Rajesh, V., & Moona, R. (1999). Processor modeling for hardware software codesign. In *Proceedings of the 12th International Conference on VLSI Design* (pp. 132137). Washington, DC: IEEE.

Saponara, S., Fanucci, L., Marsi, S., & Ramponi, G. (2007). Algorithmic and architectural design for real-time and power-efficient Retinex image/video processing. *Journal of Real-Time Image Processing*, *1*(4), 267283. doi:10.1007/s11554-007-0027-z

Schliebusch, O., Meyr, H., & Leupers, R. (2007). *Optimized ASIP Synthesis from Architecture Description Language Models*. New York: Springer.

Target Compiler Technologies. (2009). *IP Designer*. Retrieved November 14, 2009, from http://www.retarget.com/products/ipdesigner.php

Vercauteren, F. (2008). Optimal pairings. *Cryptology ePrint Archive, Report 2008/096.* Retrieved November 20, 2009, from http://eprint.iacr.org/2008/096

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 3, edited by Seppo Virtanen, pp. 1-23, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Section 2
Mobile Communication Applications

Chapter 6
System Architecture for 3GPP-LTE Modem using a Programmable Baseband Processor

Di Wu
Linköping University, Sweden

Dake Liu
Linköping University, Sweden

Johan Eilert
Linköping University, Sweden

Anders Nilsson
Coresonic AB, Sweden

Rizwan Asghar
Linköping University, Sweden

Eric Tell
Coresonic AB, Sweden

Eric Alfredsson
Coresonic AB, Sweden

ABSTRACT

The evolution of third generation mobile communications toward high-speed packet access and long-term evolution is ongoing and will substantially increase the throughput with higher spectral efficiency. This paper presents the system architecture of an LTE modem based on a programmable baseband processor. The architecture includes a baseband processor that handles processing time and frequency synchronization, IFFT/FFT (up to 2048-p), channel estimation and subcarrier de-mapping. The throughput and latency requirements of a Category four User Equipment (CAT4 UE) is met by adding a MIMO symbol detector and a parallel Turbo decoder supporting H-ARQ, which brings both low silicon cost and enough flexibility to support other wireless standards. The complexity demonstrated by the modem shows the practicality and advantage of using programmable baseband processors for a single-chip LTE solution.

DOI: 10.4018/978-1-4666-0912-9.ch006

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

INTRODUCTION

3GPP Long-Term Evolution (LTE) is the forthcoming radio access technology which incorporates Orthogonal Frequency Division Multiple Access (OFDMA) as the multiple-access scheme in downlink. Compared to existing Code Division Multiple Access (CDMA) based technologies (e.g., WCDMA, CDMA2000 and TD-SCDMA), LTE is more optimized for data service and achieves higher spectral efficiency. Furthermore, LTE brings the scenario of Frequency Division Duplexing (FDD) and Time Division Duplexing (TDD). Hence it is very likely that LTE will be deployed to replace existing WCDMA, CDMA2000 and TD-SCDMA in the future for the ever-increasing data traffic.

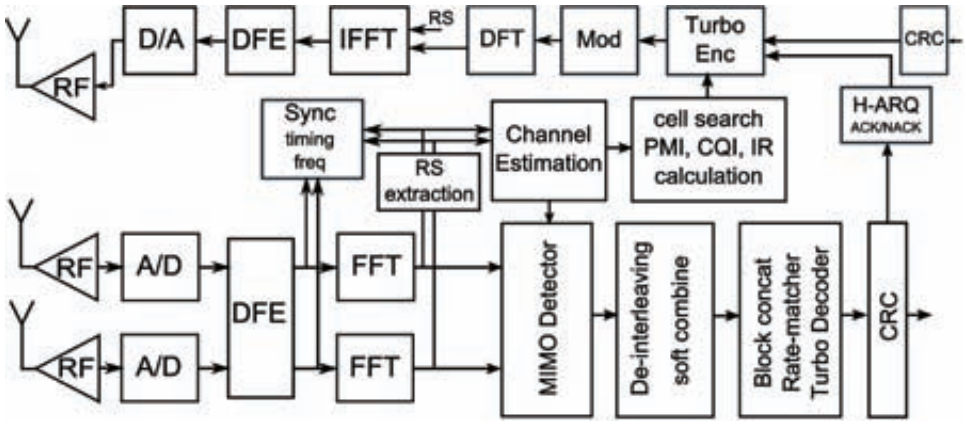
As the standardization of 3GPP LTE is still ongoing, the features defined will keep evolving along with the time as what has happened with 3G standards. Therefore, it is essential to design a scalable platform which supports the change of standard specifications without the redesign of hardware. The downlink receiver chain of an LTE modem is depicted in Figure 1.

There are a number of design challenges for LTE such as frequency synchronization, channel estimation, Multiple-Input Multiple-Output (MIMO) detection, Hybrid Automatic Repeat ReQuest (H-ARQ) and high-throughput forward error correction (FEC) decoding. Meanwhile, in order to cover legacy standards (e.g., WCDMA and DVB) and other new standards (e.g., WiMAX), either multiple Application Specific Integrated Circuits (ASIC) modems (one for each standard) have to be integrated into a chip or a programmable hardware which can handle multiple standards (Nilsson, Tell, & Liu, 2009) has to be used. The first solution not only consumes a significant amount of hardware, it also requires more integration work. The second solution is called software-defined radio (SDR) which exploits the similarity among different signal processing tasks to allow hardware multiplexing. SDR with an efficient architecture only consumes slightly more hardware while being able to support multiple standards, compared to single-standard ASIC solutions.

In Berkmann et al. (2008), implementation issues of an LTE modem is presented with insight to both the algorithms and their implementation cost estimation. However, to the best knowledge of the authors, detailed information of Software Defined Radio (SDR) based LTE modems are not yet available in literatures. In this paper:

- The architecture and implementation results of an LTE category 4 modem based on a novel programmable baseband pro-

Figure 1. Functional flow of an LTE Modem (physical layer only)



cessor, LeoCore of Coresonic (Nilsson, Tell, & Liu, 2009), is presented, which is the first SDR based LTE modem presented in the literature with architectural and performance information.

- The implementation of a MIMO detector that supports both MMSE and a novel low-complexity close-Maximum-Likelihood (close-ML) MIMO detection method is presented.
- The implementation of a parameterized parallel Radix-2 Turbo decoder that supports all block sizes is presented.
- The link-level performance of the complete LTE receiver is presented with various signal distortions taken into consideration. The degradation due to errors introduced in different processing stages is presented in the simulation result.

OVERVIEW OF 3GPP-LTE FEATURES

The LTE modem presented in this paper meets the physical layer requirements listed in Table 1.

OFDMA and SC-FDMA

Being significantly different from 3GPP WCDMA standards which use CDMA, LTE adopted OFDMA as the downlink access scheme and single-carrier Frequency division multiple access (SC-FDMA) as the uplink transmission scheme. The major difference between OFDMA and SC-FDMA is that the latter introduces one extra Discrete Fourier Transform to mitigate the peak-to-average power ratio problem.

Multi-Antenna

Multi-antenna (MIMO) technologies have been incorporated into LTE. As defined in 3GPP (2008), spatial multiplexing (SM) and transmit diversity (space-frequency block coding, SFBC) are the two MIMO schemes used in different scenarios. As depicted in

Figure 2, the antenna mapping consists of two parts namely layer mapping and pre-coding. The former multiplexes the modulated symbols belonging to one or two code words into different number of layers (or code blocks) to transmit. The latter load symbols from each layer and maps them to different antennas and subcarriers. Only two antennas are considered by UE CAT1-4. In OFDMA systems such as LTE, the general transmission model of each subcarrier is:

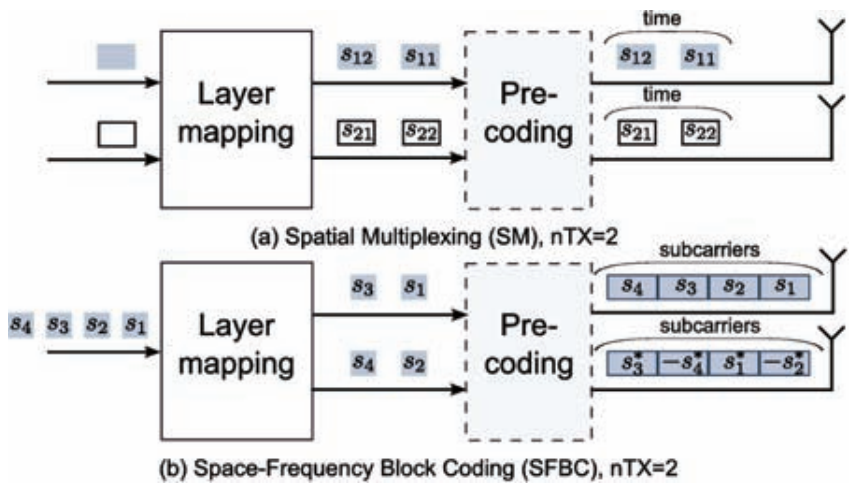
$$r = \mathrm{H}s + n \tag{1}$$

Where H is the frequency domain channel matrix, s and r are in respect the transmitted and received symbol vector, and n is the noise added by the channel.

Table 1. Supported LTE UE categories. 3GPP (2008)

UE Category (CAT)	**1**	**2**	**3**	**4**
Supported Bandwidth	1,4,3,5,10,15,20 MHz			
Antenna Configurations	Up to 2x2 SM and SFBC			
Num of Layers for SM	1	2		
Max num of Soft-bits	250368	1237248	1237248	1827072
DL peak rate (Mbit/s)	10	50	100	150
UL peak rate (Mbit/s)	5	25	50	50

Figure 2. Downlink multi-antenna transmission schemes



SYSTEM ARCHITECTURE

The processing flow of an LTE physical downlink shared-channel (PDSCH) which is the major part of a LTE receiver is illustrated in Figure 1. Note the modem implementation presented in this paper only includes the physical layer assuming higher layers (e.g., Radio Link Control and Media Access Control) are handled by one or two microprocessors. However, in this paper, higher layers are out of the scope of the presented work. The key modules in the flow are presented in the following sections.

Digital Front-End (DFE)

The DFE unit is the bridge between the Analog Front-End (AFE) and Digital BaseBand (DBB) part in the wireless systems. The function of DFE is usually gain control, sample rate conversion, pulse shaping, matched filtering and sometimes phase adjustment. Generally speaking, it is mainly a block of digital filters. Although from a functionality perspective the DFE is rather simple compared to other baseband blocks which diversify according to various standards, it still consumes a significant portion of the die area and power. A configurable DFE which supports multiple standards is included in the baseband processor.

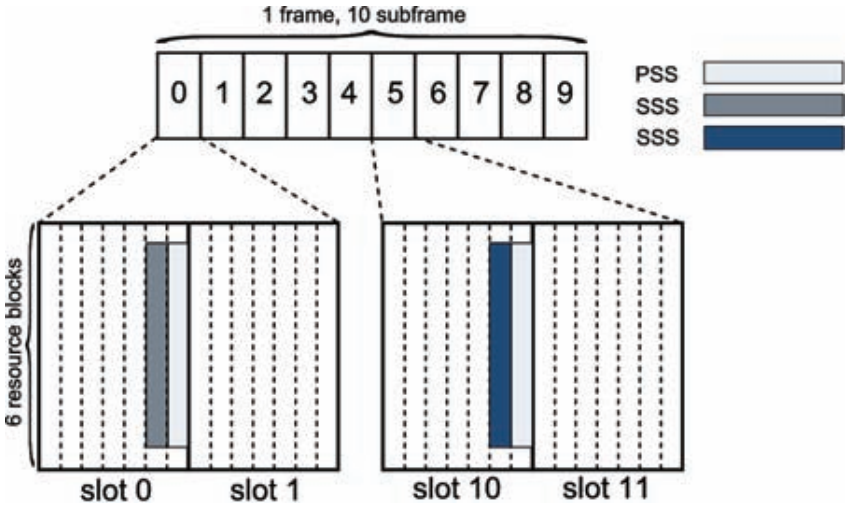
Synchronization

Being an OFDM system, LTE is sensitive to Carrier Frequency Offset (CFO) which causes inter-carrier interference. By utilizing the method in Speth et al. (2001), timing synchronization and fractional CFO estimation can be performed. Integer frequency offset estimation is then applied to remove misalignment which is integer multiple of the subcarrier spacing. Thereafter, even though most of the CFO has already been compensated for, there is still a Residual Frequency Offset (RFO) due to estimation errors. RFO is estimated based on the frequency-domain symbols after FFT.

Similar to existing cellular systems such as WCDMA, in LTE, the same cell search procedure is needed for the UE to access a certain cell. In LTE, two specially defined broadcasting signals namely the Primary Synchronization Signal (PSS) and Secondary Synchronization Signal (SSS) are used for cell search. As illustrated in

Figure 3, PSS and SSS are transmitted twice every frame (10μs). SSS is placed in the preceding symbol of PSS so that coherent detection is allowed. Similar to WCDMA, the same PSS is

Figure 3. Synchronization signal structure of FDD-LTE



transmitted in a certain cell while two different SSS are transmitted within one frame. Similar to WCDMA, PSS can be used to determine the slot boundary and SSS can be used to determine the frame boundary. Reference signals (RS) that are scattered in both time and frequency domains can be used to track the residual frequency offset. It is also possible to use other methods to perform synchronization.

In Figure 4, Figure 5, and Figure 6, the equalized symbols in a link-level LTE simulator are plotted. The simulator parameters are provided in table 3. As depicted in Figure 4, without frequency synchronization, the equalized symbols are serious rotated by the CFO which makes it impossible to perform accurate de-mapping according to the 64QAM constellation. In Figure 5, the fractional and integer frequency offset has already been corrected with only the residual offset left. The figure shows that the equalized symbols have been substantially de-rotated and are separable with mild effort. However, they are still noticeably rotated clockwise by a few degrees. As shown in Figure 6, after residual frequency correction, the equalized symbol are rotated counter clockwise to reassemble the 64QAM constellation. Most of them are within the boundaries that can be used to de-map the symbols to transmitted bits.

Channel Estimation

Like many other OFDM system (e.g., Digital Video Broadcasting), pilot symbols (which is called reference signals in LTE) are inserted during subcarrier mapping in both time and frequency directions. The major difference here is the use of multiple antennas. Being different from the typical "MIMO-OFDM" channel estimation problems (Li, Seshadri, & Ariyavisitakul, 1999) in academia, the reference signals transmitted from multiple antennas are orthogonal to each other which means the channel impulse response between different Tx-Rx antenna pairs can be separately estimated. This avoids the high complexity of real "MIMO" channel estimation at the cost of lower spectrum efficiency due to the "silent" subcarriers carrying nothing.

Detection

For MIMO systems, a major challenge is the separation and detection of the transmitted symbols at the receiver (Wu, Larsson, & Liu, 2009).

Figure 4. Equalized symbols with freq offset (LTE, 64QAM, CFO: 36 kHz)

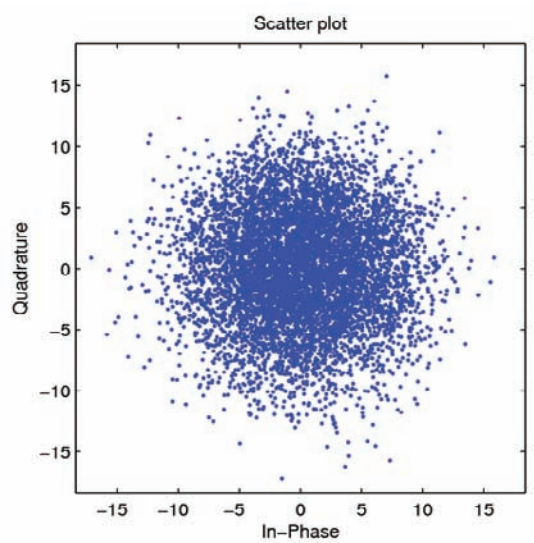


Figure 5. Equalized symbols with only residual freq offset (LTE, 64QAM, CFO: 36 kHz)

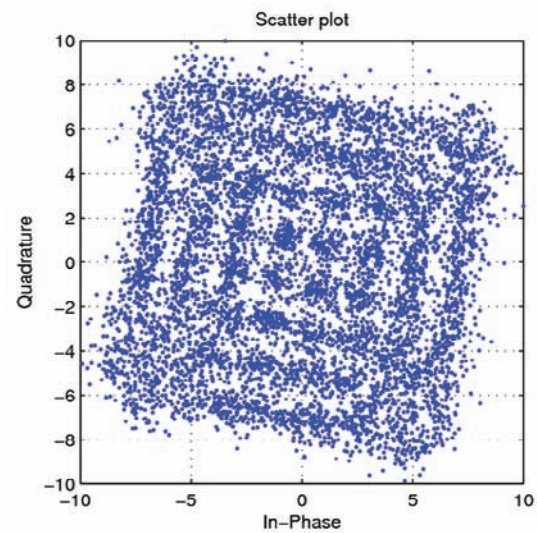
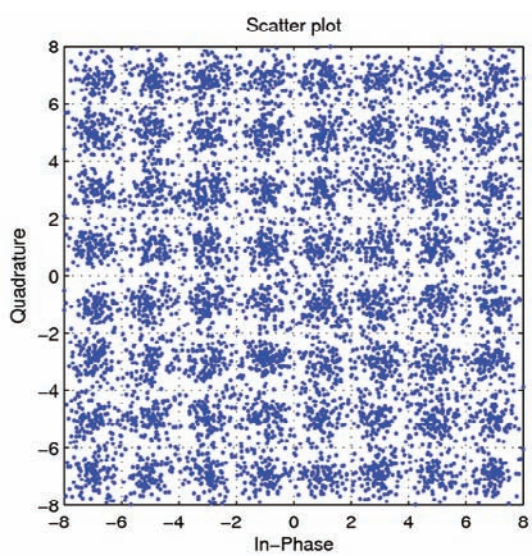


Figure 6. Equalized symbols with residual freq offset corrected (LTE, 64QAM CFO: 36 kHz)



Due to the amount of operations involved, such as matrix inversion, it cannot be handled by the baseband processor itself. Among different detection algorithm, Maximum Likelihood (ML) detection is an optimum detector that computes:

$$L\left(b_i \middle| r\right) = \log\left(\frac{\sum_{s:b_i(s)=1} \exp\left(-\frac{1}{N_0}\left\|r - \mathrm{H}s\right\|\right)^2}{\sum_{s:b_i(s)=0} \exp\left(-\frac{1}{N_0}\left\|r - \mathrm{H}s\right\|\right)^2}\right) \tag{2}$$

Here "$s{:}b_i(s)=\beta$" means all s for which the ith bit of s is equal to β.Computing (2) requires enumeration of the entire set of possible transmitted vectors. The complexity of doing this is usually not affordable for implementation in practice. However since ML provides the best theoretical performance, it is commonly used as a benchmark when comparing other algorithms. Linear detection such as Minimum Mean-Square-Error (MMSE) has very low complexity. MMSE detection is defined as follows:

$$MMSE:\ \hat{s}_{MMSE} = \left(\mathrm{H}^{\mathrm{H}}\mathrm{H} + \sigma^2\mathrm{I}\right)^{-1}\mathrm{H}^{\mathrm{H}}r \tag{3}$$

Being the most widely used detection scheme, MMSE detection involves matrix inversion which

can be efficiently handled through direct inversion with sufficient numerical stability (Eilert, Wu, & Liu, 2008). Although MMSE detection has low complexity which makes it attractive for real-time implementation, it is inferior in performance compared to more advanced detection methods such as Burg et al. (2005) and Barbero and Thompson (2006), especially when the channel is slow-fading (Wu, Larsson, & Liu, 2009). As a trade-off between detection performance and implementation complexity, close-ML soft-output detectors such as Wu, Larsson, & Liu (2009) and Zimmermann et. al. (2008) can be used. In Wu, Larsson, and Liu (2009), a fixed complexity close-ML detector is presented which substantially reduces the complexity by only partially enumerating the symbols selected for exact marginalization. More importantly, it has a fully parallel structure which makes it suitable for parallel implementation. Hence it is chosen for this implementation.

In Wu, Eilert, and Liu (2009), a low-complexity close-ML MIMO detection scheme for high-order modulation is proposed called MFCSO for Modified FCSO. Taking a 2 × 2 MIMO system as an example, considering each complex-valued symbol as one layer, only one of them is exactly marginalized with the other approximately marginalized (using DFE hard-decision). The channel rate processing of MFCSO involves the QR decomposition (QRD) of two 2×2 channel matrices which are $\mathbf{H}_1$ and $\mathbf{H}_2$where

$$\mathbf{H}_1 = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \mathbf{H_2} = \begin{bmatrix} h_{12} & h_{11} \\ h_{22} & h_{21} \end{bmatrix} \tag{4}$$

The QRD generates an upper triangular matrix R, and a unitary matrix Q so that

$$\begin{aligned} \mathbf{H}_1 &= \mathbf{Q}_1\mathbf{R}_1 \\ \mathbf{H}_2 &= \mathbf{Q}_2\mathbf{R}_2 \end{aligned} \tag{5}$$

The detection procedure for 2×2 SM presented in Wu D., Eilert, J., & Liu, D. (2009) is in the following:

Linear detection in (6) or (5) is carried out to estimate the 2×1 initial symbol vector

$$\hat{s}_{init} = \min_{\hat{s}_{init,k} \in \Im} \left\| \mathbf{H}_1 s - r \right\|^2 \tag{6}$$

Here s is the transmitted symbol vector, within which, s_k is the kth symbol.

- For each initially estimated symbol $\hat{s}_{init,k}$, $k \in \{1,2\}$, a candidate set ξ_k is created. $\Im_k$ contains N lattice points close to $\hat{s}_{init,k}$ In this paper, it is decided that N = 16 for 64-QAM and N = 9 for 16-QAM.
- First s_2 is chosen as the top-layer symbol. In order to perform DFE, needs to be computed. The same operation is needed once again when $s1$ is chosen as the top layer later.

$$\tilde{r} = \mathbf{Q}_1^H r \tag{7}$$

- For the nth constellation point $\zeta_n \in \Im_2$, its effect on e$r1$ will have to be canceled out.

$$\tilde{r}_1 = \tilde{r}_1 - \mathbf{R_1}(1,2)\zeta_n \tag{8}$$

Based on ζ_n, the partial Euclidean distance

$$\delta_n = \left\| \mathbf{R_1}(2,2)\zeta_n - \tilde{r}_2 \right\|^2 \tag{9}$$

is computed for the top-layer.

- DFE is applied to detect the other layer. Using back-substitution, $\hat{s}_1$ can be estimated from

$$\hat{s}_1 = \arg\min_{\hat{s}_1 \in \xi} \left\| \mathbf{R_1}(1,1)\hat{s}_1 - \tilde{r}_1 \right\|^2 \quad (10)$$

- The estimated $\hat{s}_1$ together with $\hat{s}_2 = \zeta_n$ form a complete possible transmitted symbol vector $\hat{s}$, based on which, an accumulated full Euclidean distance can be computed.

$$\delta_n = \delta_n + \left\| \mathbf{R_1}(1,1)\hat{s}_1 - \tilde{r}_1 \right\|^2 \quad (11)$$

- In total, there will be N different δ_n computed when is chosen as the top layer. Then s_1 is chosen as the top-layer symbol as well. Based on $\mathbf{Q}_2$, $\mathbf{R}_2$ and $\hat{s}_{init,k}$, the same procedure needs to be done once again to computed another N different δ_n. Hence for the 2×2 system, $2N$ different δ_n values need to be computed. They are used to update the LLR values in the end.

FEC and H-ARQ

In order to provide reliability as well as mobility of the data transmission over radio channels, Forward-Error Correction (FEC) codes such as Turbo code and Convolution code are used in LTE. These add redundant information before transmitting the data, and they are indispensable to achieve robust data transmission. Meanwhile, H-ARQ is used together with FEC to improve the throughput by retransmitting corrupted packets. Similar to WiMAX, two soft-combining methods namely Chase Combining (CC) and Incremental Redundancy (IR) are used to combine the LLR of a retransmitted packet with its previously received copies. In CC, the base station retransmits the same packet when it receives the NACK signal from UE, and UE combines the LLR information generated by the detector with those of the initially received packet. While in IR, each retransmission carries a new set of parity bits to be combined with the initially received systematic and a few parity bits. Since the amount of operations involved in Turbo decoding is significant, it has always been the bottleneck of the baseband processing which is latency constrained. Meanwhile, H-ARQ requires a large soft buffer in proportion to the data rate. The soft-buffer sizes of different UE categories are defined in Table 1. Both the Viterbi and CRC decoders are also implemented as accelerators attached to the processor.

Others

Other things such as cell search and the computation of Pre-coding Matrix Indication (PMI), Rank indication (RI) and Channel Quality Indication (CQI) at the UE side also involve significant amount of operations. These are handled by the baseband processor.

Functional Mapping

The radio signals received by the radio frequency front-end will be down converted to analog baseband signals, and then converted to digital baseband signals by the analog-to-digital converters (A/D). The digital front-end (DFE) applies filtering to the baseband signals and FFT is applied to convert the time-domain signal into frequency-domain where channel estimation and symbol detection occur. In LTE, the channel estimator will use the reference signals (RS) to estimate the MIMO channel matrices H for all data subcarriers. The estimated H and received symbol vector r belonging to the current user will be extracted for MIMO detection later. All the work above is handled by the baseband processor. The cycle cost of tasks (for 20~MHz bandwidth, 2×2 SM) mapped on LeoCore is presented in Table 2.

The **H** matrices will be passed to the MIMO detector together with the received symbols extracted from data subcarriers. The coefficients will be fed to the detector to compute the LLR soft-output $L\left(b_k^i\right)$. For example, in case of MMSE

Table 2. Cycle cost of UE receiver

Function	Million Cycles/Second
Timing & Frac. Freq. Sync.	8
Integer Freq. Sync	12
Residual Freq. Sync	18
FFT	88
Channel Estimation	80
PMI, CQI & RI Calculation	40

Table 3. Simulation parameters

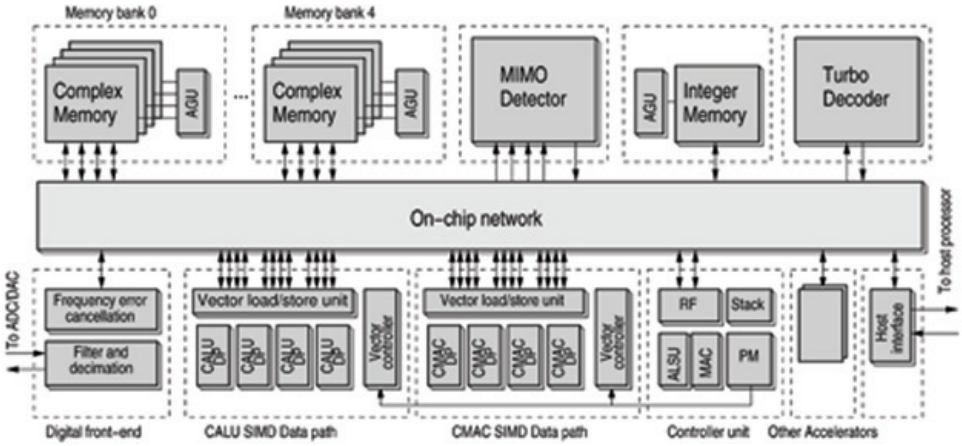
Function	Description
CQI	14
Modulation and Coding	64-QAM, 6/7 Turbo
System bandwidth	5 MHz
Channel model	Urban Micro
UE speed	3km/h
Carrier frequency offset	39kHz
Channel estimation	Least Squares
H-ARQ	Chase Combining, 3 retr
Turbo iterations	8 max, early stopping

detection is used, the detector loads the computed coefficient matrices W from the memory and multiplies W with the received symbol vectors r to compute $\hat{s}$ and to de-map $\hat{s}$ to LLR values. The LLR values will be passed to the Forward-Error-Correction (FEC) part where it will be de-interleaved and processed by the channel decoder to generate the final hard-decision of the transmitted bits. The FEC part mainly contains the soft-buffer for Chase Combining in H-ARQ and a parallel Turbo decoder which delivers a high throughput which is more than 170 Mbit/s.

BASEBAND PROCESSOR

In literature, a large number of programmable baseband processors targeting software defined radio have been proposed such as the ADRES architecture in Bougard et al. (2008) and the SODA architecture in Woh et al. (2008). The major goal of these architectures is to allow silicon efficiency to be achieved while maintaining the flexibility. In this paper, the LTE modem is based on a programmable baseband processor targeting multi-standard radio baseband processing (Nilsson, Tell, & Liu, 2009). It adopted a novel architecture namely Single Instruction Multiple Tasking (SIMT) and an architecture overview is presented in Figure 7.

Figure 7. Architecture of the LTE baseband SoC



SIMT Architecture

The major components in the SIMT architecture are: SIMD vector execution units, memory banks, the on-chip network, accelerators and an integer controller core. The SIMT processor uses vector instructions that operate on large data-sets in SIMD execution units. The key idea in the SIMT architecture is to issue only one instruction each clock cycle but still allow several operations to execute in parallel since vector instructions may run for several clock cycles on the SIMD units. This approach results in a degree of parallelism equivalent to a VLIW processor without the need for the large control-path overhead.

In this way the fact that modem processing to a large extent consists of operations on large vectors of data, is used to get a processor with high utilization of execution units and low control overhead. For example the integer data-path could execute control tasks while the CMAC performs one layer of an FFT and the CALU performs pilot tone extraction.

To be able to take full advantage of the SIMT architecture, several key architecture components are necessary: efficient vector execution units, a matching memory system and a controller core capable of managing several threads efficiently. The SIMT architecture utilizes multiple complex valued SIMD execution clusters such as complex MACs and complex ALUs. Programmable co-processors and accelerators, in this case the MIMO detector and the Turbo decoder, can also be attached to the on-chip network and be seamlessly integrated in the processor architecture.

The processor is controlled by the controller core, which includes the program memory, instruction issue logic and functions for multi-context support. The controller core executes all control flow functions as well as integer based instructions.

Memory System and On-Chip Network

To enable several concurrent vector operations, a distributed memory system is used where the memory is divided into several memory banks, with individual address generation units (AGU). This arrangement in conjunction with the on-chip network improves the power efficiency of the memory system. This also increases the throughput of the processor as multiple address calculations can be performed in parallel.

The on-chip network is realized as a restricted crossbar switch which is under direct software control. Hence no arbitration is necessary and the performance is fully predictable. This allows software tools to use static scheduling of both network transactions as well as for vector instructions.

Co-processors attached to the on-chip network can utilize the same addressing modes as the vector instructions. In this way OFDMA pilot extraction can be done in parallel with MIMO symbol detection and channel estimation.

The memory system in the SIMT processor instance used in this paper has a memory bandwidth of 1024 bits per clock cycle for the complex valued memory system and 80 bits per clock cycle for the integer memory system.

Configurable Accelerators

Although many baseband processing tasks can be mapped to the programmable processor mentioned in earlier sections, there are still a few tasks which need to be implemented in dedicated hardware to meet the performance and power constraints. For the UE category 4 requirement, two accelerators are added to the on-chip network. They are controlled by a number of control registers which can be set by the baseband processor.

MIMO SYMBOL DETECTORS

As presented in earlier sections, MIMO detection is one of the most performance demanding tasks in the receiver. There is a gap between linear detectors (e.g., MMSE) and close-ML detectors (e.g., MFCSO in Wu, Larsson, & Liu, 2009) with respect to performance and implementation cost. In order to allow different MIMO detection algorithms to be incorporated into the same hardware, both an MMSE MIMO detector and a combined MMSE/MFCSO MIMO detector have been designed for the modem. As presented in Wu, Eilert, & Liu (2009), the former one supports only MMSE detection based on the method presented in Eilert, Wu, and Liu (2008) for 2×2 SM and up to 64-QAM modulation schemes. Implemented using ST 65nm CMOS process, it only consumes $0.14mm^2$ area. In comparison, the combined detector supports real-time MFCSO detection for 2×2 SM and up to 64-QAM modulation schemes. As presented in Eilert, Wu, & Liu (2008), an MFCSO detector contains an MMSE detector and since the MMSE mode will consume substantially lower power than the MFCSO mode, the detector is switched to MMSE mode when the UE enters power-saving mode. The combined dual mode MMSE/MFCSO detector consumes less than $0.3mm^2$ area which is only 2 times that of the MMSE-only detector. Running at 200~MHz clock, both detectors can support real-time detection of full 20MHz band. For low-end modems with a focus on connectivity with minimum cost (e.g., smartphones), the MMSE-only detector can be used. For high-end modems, where performance is the key issue, the dual mode MIMO detector should be integrated. Both detectors support the detection of all MIMO schemes defined in LTE.

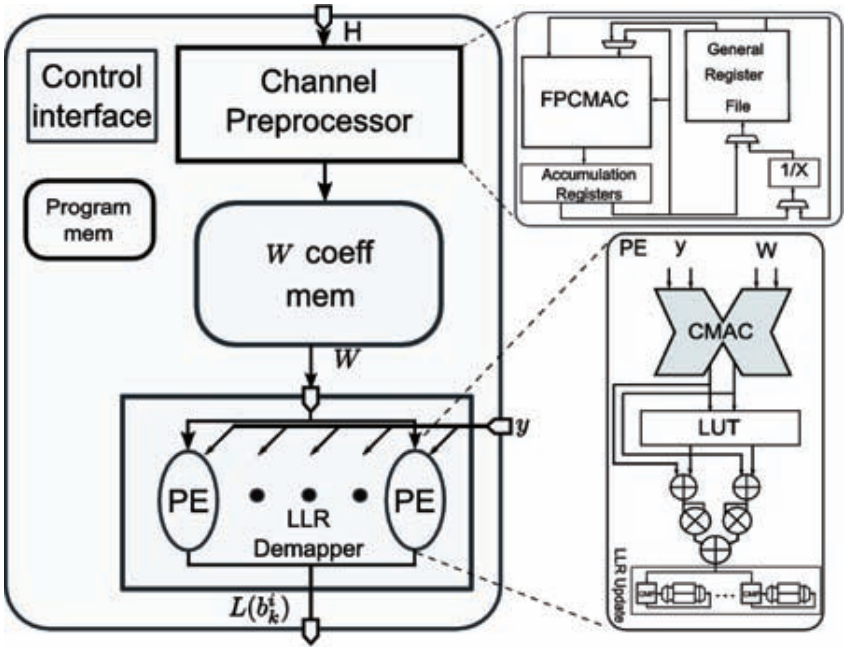
The block diagram of the dual mode detector is depicted in Figure 8. The detector contains two parts, the channel preprocessor unit (ChPU) and the LLR demapper. The ChPU handles channel-rate processing tasks such as computation of the MMSE coefficient matrix in (3), which is done once the estimated channel is updated. The computed coefficient matrices W will be stored in the coefficient buffer and fed to the LLR demapper as input. As depicted in Figure 8, the ChPU contains two Complex-valued Multiply-and-ACcumulate (CMAC), an inverse-square-root unit and a 32bit register file containing 24 registers. The ChPU is a programmable unit controlled by microcode. The method presented in Eilert, Wu, and Liu (2008) has been used to compute W, and the Modified Gram-Schmidt method in Golub and Van Loan (1996) is used to compute Q and R matrices in (5). These computed coefficient matrices will be stored in the coefficient buffer and fed to the LLR de-mapper as input. The LLR de-mapper computes the LLR values using the MFCSO method presented in Wu, Larsson, & Liu (2009)

$$\mathrm{W}_{\mathrm{MMSE}} = \left(\mathrm{H}^{\mathrm{H}}\mathrm{H} + \sigma^2\mathrm{I}\right)^{-1}\mathrm{H}^{\mathrm{H}} \tag{12}$$

$$L\left(b_k^i\right) = -\frac{1}{\sigma^2}\left\{\min_{1\in\zeta:b_k^i=0}\delta - \min_{1\in\zeta:b_k^i=1}\delta\right\} \tag{13}$$

The LLR de-mapper contains a number of processing elements (PE) which can utilize the parallelism in the algorithm. The computed LLR values $L\left(b_k^i\right)$ can be either directly passed to the channel decoder or combined with previously stored LLR values in the soft-buffer for H-ARQ. Since the processing in the LLR demapper is at symbol rate which is much higher than the channel-rate processing in the ChPU, a fully pipelined architecture is used in DU to allow the computation of 16 different δ_n in (11) to be finished within 16 clock cycles. DU is configured by a control register. The MMSE mode can be used in power saving mode to reduce the power consumption with a loss of detection performance. 16-bit fixed-point datatype with proper scaling is adopted in LLR demapper, the output LLR values are quantized to be 6-bit signed integers. The

Figure 8. Block diagram of the dual-mode MIMO detector



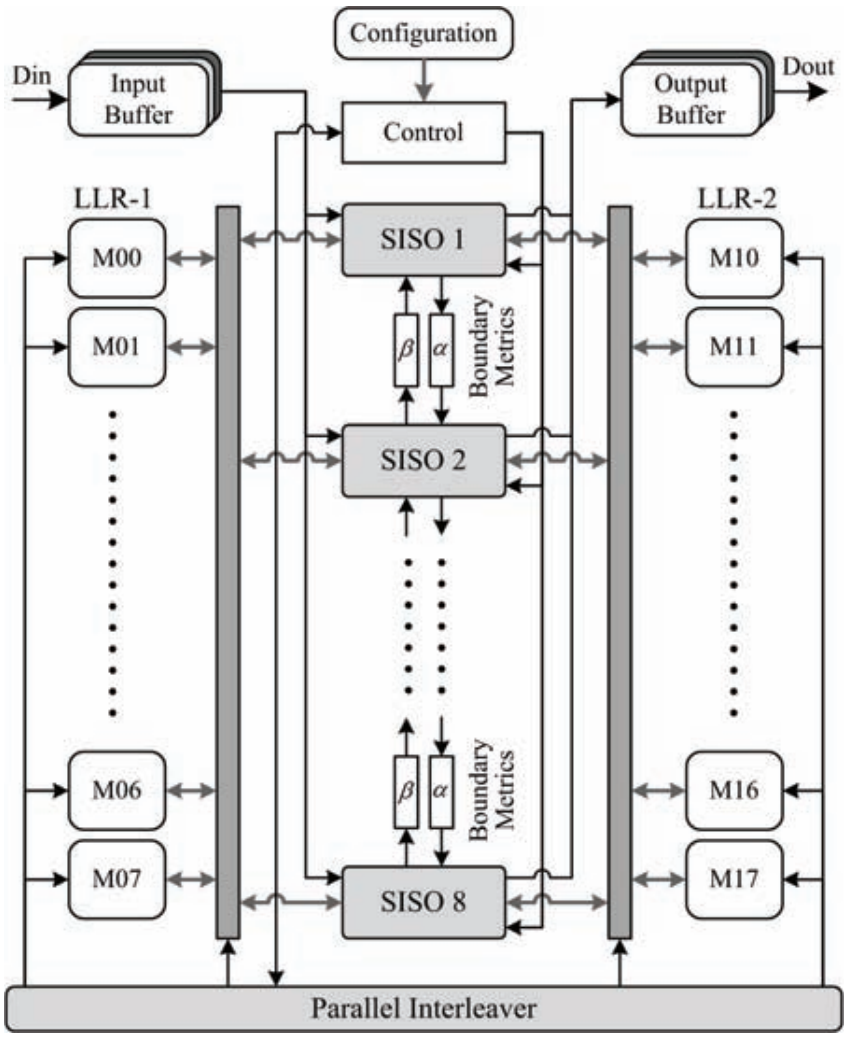
number of PE is decided at design time according to the processing load and latency analysis.

PARALLEL TURBO DECODER

According to the definition of LTE parameters in Table 1, up to 150Mbit/s peak data rate for downlink needs to be supported. With the state-of-the-art Turbo decoder (e.g., Benkeser et al., 2009), that contains a single soft-input soft-output (SISO), the maximum throughput that can be achieved with a clock frequency of 300MHz is around 25Mbit/s. Furthermore, the SIMD architecture based turbo decoder (Shin & Park, 2007) is not silicon efficient. Hence a new parallel Turbo decoder is needed to supply higher data rates. Since the variation of Turbo decoding procedure among different standards is small, a parameterized Turbo decoding accelerator is sufficient to achieve enough flexibility while maintaining area and power efficiency. In order to supply 150Mbit/s peak data rate required by LTE UE category 4, a parallel Turbo decoder with eight SISO units has been designed with its architecture depicted in Figure 9. As defined in 3GPP (2008b), the maximum code block size is 6144 in LTE which will incur very high latency when serial Turbo decoder is used. Fortunately, a contention-free interleaver has been introduced in LTE (3GPP, 2008b) which allows multiple SISO to access different sub-blocks in parallel. A super windowing strategy (Yoon & Bar-Ness, 2002) is adopted in this paper to speed up the decoding.

The challenge when using parallel window is the absence of initial values at the border of each parallel window. As depicted in Figure 9 and Figure 10, a method namely Next Iteration Initialization (NII) (Dingninou, Raouafi, & Berrou, 1999) is used to estimate the initial values both sliding windows and parallel windows using the corresponding state metrics produced at previous iteration. This requires extra storage of α and β values. These values are saved in a first-in first-out (FIFO) between neighboring SISO units and the local buffer inside each SISO. Therefore, the reduction of computational latency is at the cost

Figure 9. Block diagram of the parallel turbo decoder

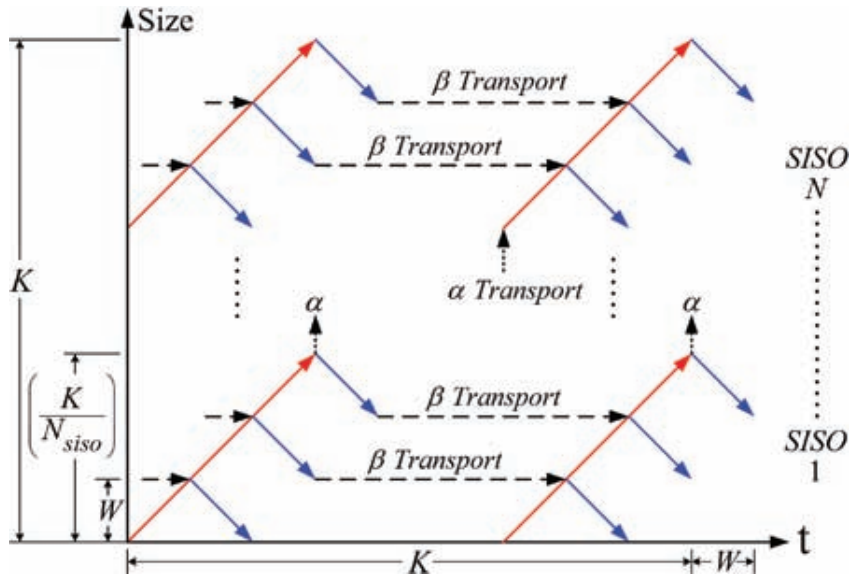


of increased storage. Fortunately, the storage overhead is limited as only the state metric (α and β) values at the beginning and end of sliding windows and parallel windows need to be saved.

SISO Unit

SISO unit is the major block in a Turbo decoder. The input to the SISO are: systematic a-priori information λa_k^s, systematic intrinsic information λi_k^s and parity intrinsic information λi_k^p. In each SISO, classical sliding-window processing is applied to reduce the latency. In this paper, the size of the sliding window S_{sw} is 64. Each SISO unit consists of α, β units for calculating state metric, γ unit for calculating branch metric and Log-Likelihood Ratio (LLR) units (Figure 11). Radix-2 log-Max decoding is used with the scaling of the extrinsic information which allows a close log-MAP decoding performance to be achieved. The scaling factor ranges from 0.6 to 0.8. During the forward recursion the result from α and γ units will be saved into a stack memory, from where; the β unit and LLR unit will read in a reversed order to compute the backward metrics. LLR unit computes the extrinsic information $\Lambda(d_k)$ and the

Figure 10. Scheduling chart of a half iteration

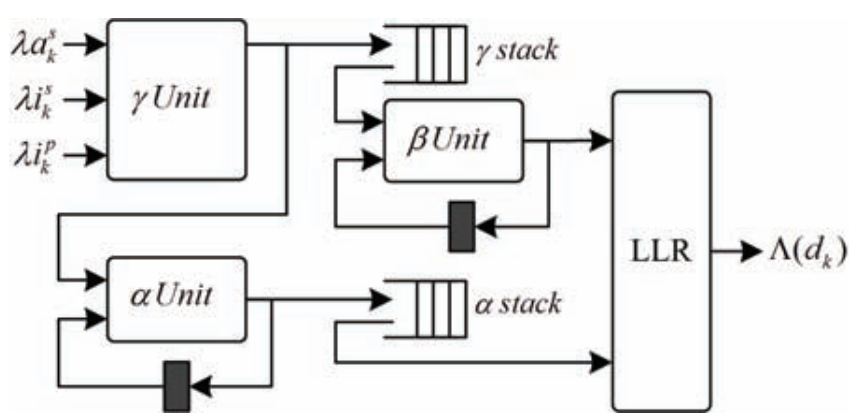


soft-output based on the result from α unit and β unit. The output of all the SISO units, is then passed through the switch network to different memory banks for later access by the SISO units in the next iteration.

Parallel Interleaver

The internal interleaver involved with the Turbo Code in 3GPP-LTE is based on quadratic permutation polynomial (QPP) $A_{(x)} = (f_1x + f_2x^2) \bmod K$, where $x = 0, 1, 2, \ldots (K-1)$, K is the block size and the polynomial provides deterministic interleaver behavior for different block sizes with appropriate values of f_1 and f_2. It allows conflict-free parallel access, and also exhibit good distance properties. To support the parallel Turbo decoder, a configurable parallel interleaver is adopted. It can generate eight addresses in parallel for the data access of eight SISO units. Direct implementation of QPP is in-efficient due to multiplications, modulo function and bit growth problem. The simplified hardware solution is to use recursive approach adopted in Asghar and Liu (2008) and Asghar et al.

Figure 11. Block diagram of the SISO unit



(in press). The QPP can be re-written for recursive computation as $A_{(x+1)} = (A_{(x)} + g_{(x)}) \bmod K$, where $g_{(x)} = (f_1 + f_2 + 2.f_2.x) \bmod K$, which can also be computed recursively as $g_{(x+1)} = (g_{(x)} + 2.f_2) \bmod K$. Owing to the parallelism inherited in the QPP interleaver, the generation of parallel interleaver addresses requires 32 additions. However, for the sake of hardware reuse, part of the basic interleaver unit can be shared to generate multiple addresses at the same time. The optimized hardware uses 18 additions in total to generate 8 parallel interleaver addresses, thus saving 14 additions. The hardware for parallel interleaver address generation for LTE is shown in Figure 12.

Due to recursive nature of QPP interleaver each sub-block computed by a processor *Pi* needs some starting point i.e., S1 … S8. The starting value for each of the processor can be computed by using the index $m = 0, 1, \ldots (N_{SISO} - 1)$ and sub-block size $K_{sub_blk} = K/N_{SISO}$. The starting address computation for data index $(m \cdot K_{sub_blk})$ can be computed by:

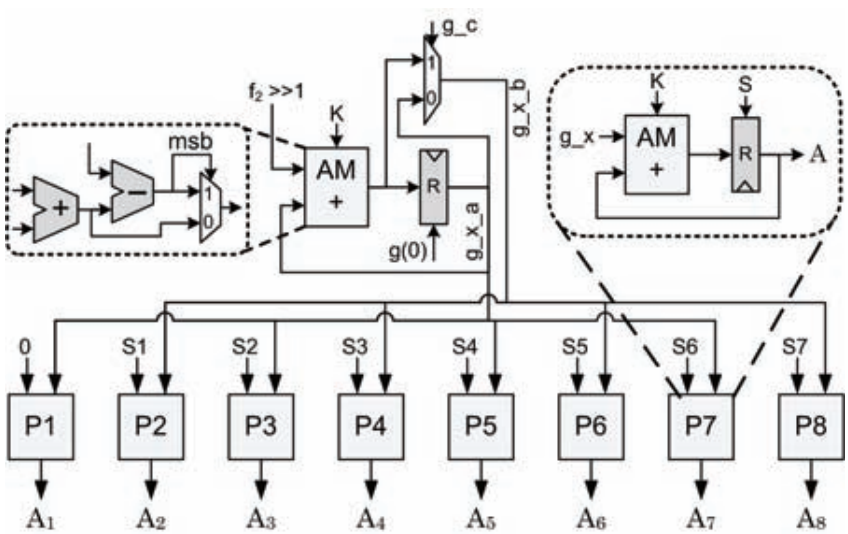
$$S(m) = \left(f_1 \cdot m \cdot K_{sub_blk} + f_2 \cdot \left(m \cdot K_{sub_blk} \right)^2 \right) \bmod K \quad (14)$$

In simplified form it can be written as $S(m) = \left(S(m-1) + S'(m) \right) \bmod K$, where $S'(m) = f_1 \cdot K_{sub_blk} + (2m-1) \cdot f_2 \cdot (K_{sub_blk})^2$. The term $S'(m)$ is not hardware efficient when computing it on-the-fly. A LUT is needed to achieve the faster switching among different block sizes; however, if fast switching among different block sizes is not needed then it can be computed on-the-fly through multiple recursive computations.

H-ARQ

In LTE, two soft-combining methods namely Chase Combining (CC) and Incremental Redundancy (IR) are supported. In this paper, only CC is considered. The base station retransmits the same packet when it receives the NACK signal from UE, and UE combines the LLR information generated by the detector with those of the initially received packet. The gain of H-ARQ is at the cost of larger soft buffer used to store the demodulated soft information. In order to support the buffer size as large as 1.8Mbits, high-density memory such as 1T-SRAM needs to be used. Since

Figure 12. Parallel interleaver architecture



1T-SRAM is different from the technology used for the decoder, it is not included in the results of this paper.

PERFORMANCE ANALYSIS

In order to carry out both fast prototyping and verification of the 3GPP LTE modems, a complete physical layer simulation chain has been developed in Matlab and C. Combining with an LTE signal generator; it allows both quantitative performance evaluation and conformance testing of the chip. The simulation chain includes a transmitter conforming to 3GPP technical spec (e.g., 3GPP, 2008; 3GPP, 2008b). The 3GPP SCME model (Baum et al., 2005) is used as the channel model.

In the simulation done for this paper, 8000 subframes are simulated. To compare the performance of MFCSO and MMSE detection, 2×2 SM is used. No close-loop pre-coding is assumed in this paper. At most three retransmissions are allowed in CC based H-ARQ. The throughput figures are calculated based on the method in Mehlfuhrer, Caban, and Rupp (2008). Simulation parameters are listed in Table 3.

- In Figure 13 and Figure 14, simulation settings represented in acronyms such as EC (estimated channel), PC (perfect channel), CFO (with carrier frequency offset), RFOC (residual frequency offset cancellation), PT (parallel log-Max Turbo) and ST (serial log-Max Turbo). The result shows the degradation caused by approximation (e.g., parallel windowing and sliding windowing) introduced in the parallel Turbo decoder is rather small (1 Mbit/s difference in throughput) when compared with the serial log-Max Turbo decoder.
- Severe performance degradation incurs when the residual frequency offset is not corrected. When it is corrected by RFOC, the loss due to CFO is negligible in this implementation.
- The MFCSO detector substantially outperforms the MMSE detector (more than 50% higher throughput) in 2×2 SM.

Figure 13. Frame-error-ratio (2×2 SM, CQI=15, 5 MHz band)

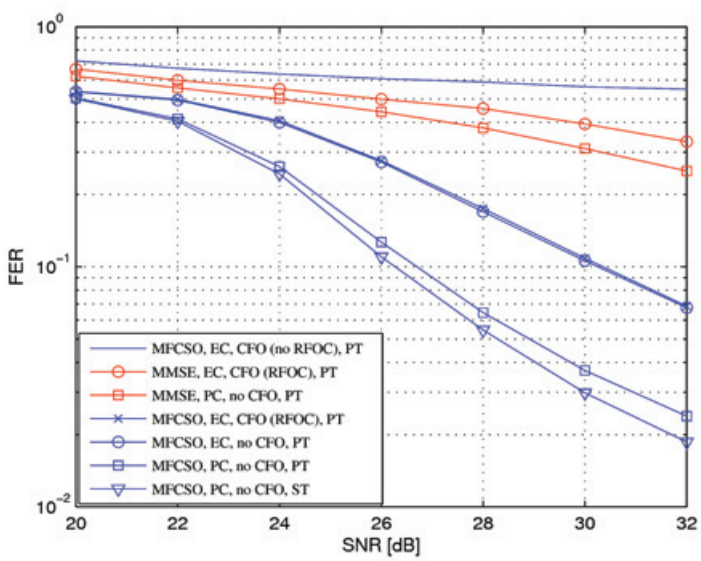
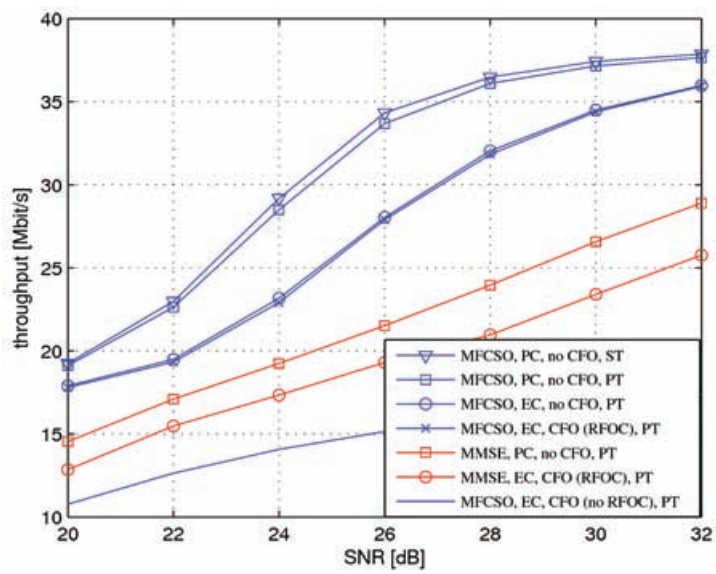


Figure 14. Coded throughput (2×2 SM, CQI=15, 5 MHz band)



- The least squares based channel estimation has limited performance which implies the need of more advanced channel estimation (e.g., cascaded Wiener filters).

AREA ESTIMATE

Both the MIMO detector and Turbo decoder have been implemented using 65nm CMOS process from STMicroelectronics. After placement and routing, the logic of the MIMO detector consumes 0.2mm^2 for 20MHz bandwidth when running at 300MHz. The Turbo decoder easily runs at 250MHz with a core area of 0.7mm^2. The throughput of the Radix-2 parallel Turbo decoder can be computed as:

$$T = \frac{S_{blk,\max} \times f_{clk}}{\left(S_{pw,\max} + S_{sw} + C_{extra}\right) \times 2 \times N_{ite}} \qquad (15)$$

Where $S_{pw,\max} = S_{blk,\mathrm{mix}}/N_{siso}$. Here $S_{blk,\max} = 6144$ is the maximum number of bits per codeword defined in 3GPP, 2008b; $N_{siso} = 8$ is the number of SISO units, $S_{sw} = 64$ is the sliding window size, $C_{extra} = 10$ is the number of overhead cycles, $f_{clk} = 250MHz$ is the clock frequency and N_{ite} is the number of decoding iterations. The throughput of the decoder is 152Mbit/s in case $N_{ite} = 6$. In high SNR region, early stopping can effectively reduce the number of iterations needed to four. This gives a throughput of 228Mbit/s. Theoretically, for 20MHz bandwidth with 2×2 Spatial Multiplexing and 64-QAM, the required throughput of Turbo decoding is 176Mbit/s, which means the presented implementation is sufficient for CAT4 PDSCH decoding. Owing to the scalability of the on-chip network in Nilsson, Tell, and Liu (2009) the Turbo decoder can be easily wrapped and integrated with the programmable platform. The ASIC implementation in 65nm CMOS resulted in 0.7mm^2 including memories at depicted in Table 4. Compared to the prior-art presented in Benkeser et al. (2009), this result shows that work in this paper achieves more than 10 times data throughput compared to Benkeser et al. (2009) while achieving higher silicon efficiency. Based on 65 nm CMOS technology, the baseband processor (including MIMO DFE, Viterbi decoder and Turbo

Table 4. Comparison of ASIC implementation results

Turbo Decoder	**Our Work**	Benkeser et. al. (2009)
Technology	**CMOS 65nm**	CMOS 130nm
Supply Voltage	**1.2V**	1.2V
Logic Size	**70kgate**	44.1kgate
Total Memory	**250kb**	120kb
Total Area	**0.7mm²**	1.2mm²
Working Frequency	**250MHz**	246MHz
Throughput	**152Mbps**	10.8Mbps
Silicon Area/Throughput	**0.005 mm²/Mbps**	0.11 mm²/Mbps

Encoder etc.) consumes 270 kgates. Including the 1T-SRAM soft buffer defined in Table 1, the pads and the pad rings, the total area of the CAT4 LTE modem (physical layer only) baseband chip is estimated to be 8 mm^2. When running at 250 MHz clock frequency, the processing capacity of PDSCH on the full 20 MHz band with 2×2 SM and 64-QAM modulation (which gives a data rate up to 120 Mbit/s at 28 dB) is supported.

OTHER ISSUES

Nevertheless, in order to fully exploit the advantage of programmable architectures, a good programming model, a friendly tool-chain and a highly skillful R&D team which is familiar with the new design flow are indispensable. Hence, whether to choose the ASIC of ASIP solution depends on the competence of the R&D team and the business model (e.g., number of legacy Intellectual Properties (IP), time-to-market etc).

CONCLUSION

This paper presents the system architecture of a 3GPP LTE CAT4 modem which is based on a programmable baseband processor, a combined MMSE/MFCSO MIMO detector and a parallel Turbo decoder. The simulation results have validated the baseband algorithms chosen to be implemented. The cost and performance analysis of the implementation shows that the mixture of ASIC accelerators, a programmable processor and an efficient memory subsystem provides a feasible solution for 3GPP LTE modem implementation. The result presented in this paper also shows that LeoCore processor occupies small area, compared to the accelerators and memories.

ACKNOWLEDGMENT

The work of D. Wu, J. Eilert, R. Asghar and D. Liu was supported in part by the European Commission through the EU-FP7 Multi-base with Ericsson AB, Infineon Austria AG, IMEC, Lund University and KU-Leuven. The authors would like to thank Christian Mehlführer and the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms at Vienna University of Technology, for contributions on the LTE simulation chain.

REFERENCES

Asghar, R., & Liu, D. (2008, May). Dual standard re-configurable hardware interleaver for turbo decoding. In *Proceedings of ISWPC*, Santorini, Greece (pp. 768-772).

Asghar, R., Wu, D., Eilert, J., & Liu, D. (in press). Memory Conflict Analysis and Implementation of a Re-configurable Interleaver Architecture Supporting Unified Parallel Turbo Decoding. *Journal of Signal Processing Systems*. doi:.doi:10.1007/s11265-009-0394-8

Barbero, L. G., & Thompson, J. S. (2006, June). Rapid prototyping of a fixedthroughput sphere decoder for MIMO systems. In *Proceedings of IEEE ICC*.

Baum, D. S., Salo, J., Milojevic, M., Kyoti, P., & Hansen, J. (2005, May). *MATLAB implementation of the interim channel model for beyond-3G systems (SCME)*.

Benkeser, C., Burg, A., Cupaiuolo, T., & Huang, Q. (2009). Design and Optimization of an HSDPA Turbo Decoder ASIC. *IEEE Journal of Solid-state Circuits*, *44*(1), 98–106. doi:10.1109/JSSC.2008.2007166

Berkmann, J., Carbonelli, C., Dietrich, F., Drewes, C., & Xu, W. (2008). On 3G LTE Terminal Implementation − Standard, Algorithms, Complexities and Challenges. In *Proceedings of IEEE IWCMC*.

Bougard, B., et al. (2008). A Coarse-Grained Array based Baseband Processor for 100Mbps+ Software Defined Radio. In *Proceedings of the Design, Automation and Test in Europe*.

Burg, A. (2005). VLSI implementation of MIMO detection using the sphere decoding algorithm. *IEEE Journal of Solid-state Circuits*, *44*(7), 1566–1577. doi:10.1109/JSSC.2005.847505

Dingninou, A., Raouafi, F., & Berrou, C. (1999). Organisation de la memoire dans un turbo decodeur utilisant l'algorithm SUB-MAP. In *Proceedings of Gretsi* (pp. 71-74).

Eilert, J., Wu, D., & Liu, D. (2008). Implementation of a Programmable Linear MMSE Detector for MIMO-OFDM. In *Proceedings of IEEE ICASSP*.

Golub, G. H., & Van Loan, C. F. (1996). *Matrix Computations* (3rd ed.). Baltimore: Johns Hopkins University Press.

3GPP. (2008, September). *Tech. Specification 36.211 V8.4.0, E-UTRA; Physical Channels and Modulation*.

3GPP. (2008b, September). *Tech. Specification 36.212 V8.4.0, E-UTRA; Multiplexing and Channel Coding*.

Li, Y. G., Seshadri, N., & Ariyavisitakul, S. (1999). Channel Estimation for OFDM Systems with Transmitter Diversity in Mobile Wireless Channels. *IEEE Journal on Selected Areas in Communications*, *17*(3).

Mehlfuhrer, C., Caban, S., & Rupp, M. (2008). *Experimental Evaluation of Adaptive Modulation and Coding in MIMO WiMAX with Limited Feedback*. EURASIP Journal on Advances in Signal Processing.

Nilsson, A., Tell, E., & Liu, D. (2009). An 11 mm^2 70 mW Fully-Programmable Baseband Processor for Mobile WiMAX and DVBT/H in 0.12μm CMOS. *IEEE Journal of Solid-state Circuits*, *44*(1), 90–97. doi:10.1109/JSSC.2008.2007167

Shin, M. C., & Park, I. C. (2007). SIMD processor-based Turbo decoder supporting multiple third-generation wireless standards. *IEEE Transaction on VLSI*, *15*, 801–810. doi:10.1109/TVLSI.2007.899237

Speth, M., Fechtel, S. A., Fock, G., & Meyr, H. (2001). Optimum receiver design for OFDM-based broadband transmission - part II: a case study. *IEEE Transactions on Communications*, *49*(4), 571–578. doi:10.1109/26.917759

Woh, M., et al. (2008). From SODA to scotch: The evolution of a wireless baseband processor. In *Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture*.

Wu, D., Eilert, J., & Liu, D. (2009). Evaluation of MIMO Symbol Detectors for 3GPP LTE Terminals. In *Proceedings of EUSIPCO 2009*.

Wu, D., Larsson, E. G., & Liu, D. (2009, April). Implementation Aspects of Fixed-Complexity Soft-Output MIMO Detection. In *Proceedings of IEEE VTC-Spring*.

Wu, D., Larsson, E. G., & Liu, D. (2009, April). Implementation Aspects of Fixed-Complexity Soft-Output MIMO Detection. In *Proceedings of IEEE VTCSpring*.

Yoon, S., & Bar-Ness, Y. (2002). A Parallel MAP Algorithm for Low Latency Turbo Decoding. *IEEE Communications Letters*, *6*, 288–290. doi:10.1109/LCOMM.2002.801310

Zimmermann, E., Fettweis, G., Milliner, D. L., & Barry, J. R. (2008, January). Parallel Smart Candidate Adding Algorithm for Soft-output MIMO Detection. In *Proceedings of the International Conference on Source and Channel Coding*, Ulm, Germany.

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 3, edited by Seppo Virtanen, pp. 44-64, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 7
Joint Uplink and Downlink Performance Profiling of LTE Protocol Processing on a Mobile Platform

David Szczesny
Ruhr-Universität Bochum, Germany

Attila Bilgic
Ruhr-Universität Bochum, Germany

Sebastian Hessel
Ruhr-Universität Bochum, Germany

Uwe Hildebrand
Comneon GmbH, Germany

Anas Showk
Ruhr-Universität Bochum, Germany

Valerio Frascolla
Comneon GmbH, Germany

ABSTRACT

This article provides a detailed profiling of the layer 2 (L2) protocol processing for 3G successor Long Term Evolution (LTE). For this purpose, the most processing intensive part of the LTE L2 data plane is executed on top of a virtual ARM based mobile phone platform. The authors measure the execution times as well as the maximum data rates at different system setups. The profiling is done for uplink (UL) and downlink (DL) directions separately as well as in a joint UL and DL scenario. As a result, the authors identify time critical algorithms in the protocol stack and check to what extent state-of-the-art hardware platforms with a single-core processor and traditional hardware acceleration concepts are still applicable for protocol processing in LTE and beyond LTE mobile devices.

INTRODUCTION

Next generation mobile communication systems like 3GPP's (3rd Generation Partnership Project) Long Term Evolution (LTE) will provide higher data rates at simultaneously reduced latency to account for new features and services like video streaming or online gaming over wireless links (Dahlman et al., 2008). Consequently, the radio architecture will experience another increase of its processing complexity (Berkmann et al., 2008) that already today amounts to several giga operations per second in latest feature-rich mobile handsets (Berkel, 2009).

DOI: 10.4018/978-1-4666-0912-9.ch007

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

The workload and thus the power consumption are dominated by the RF front-end and the digital baseband processing (Hausner & Drewes, 2008). For this reason many researchers are focusing on circuits and algorithms that are located in the LTE physical layer (Manolakis et al., 2008) where higher layer information is sometimes adopted for link level performance simulations (Martín-Sacristán et al., 2009). Nevertheless, more attention should be drawn to the data plane of higher protocol layers, since its processing demand will also scale with the increased data rates in LTE and even more in LTE-Advanced.

Traditionally, the protocol processing is realized in software and executed on embedded processors like the ARM family (Lueftner et al., 2007). Only cryptographic algorithms, which are characterized by a high computational effort, are offloaded to hardware in mobile platforms (Gehrmann & Stahl, 2006). It is expected that even more protocol stack functionality will be supported by hardware in future in order to identify suitable system architectures that offer the required performance while keeping the processor clock frequency and thus the energy budget at a reasonable level. This is of particular importance in mobile devices with a limited battery lifetime. Therefore, a thorough investigation of the processing demand is mandatory for an effective hardware/software partitioning of the LTE protocol stack.

Another related aspect in the design of upcoming mobile phone platforms deals with the question how such a hardware/software co-design can be done efficiently. Especially for a performance analysis on system level several commercial and academic design tools exist (Gajski et al., 2009). One approach comprises the use of a virtual system prototype (Cockx, 2000; Eckart & Schnieringer, 2006; Brandenburg et al., 2007). This is a software image of the hardware platform that allows for an accurate performance analysis of the software stack in an early design phase before the final silicon is available.

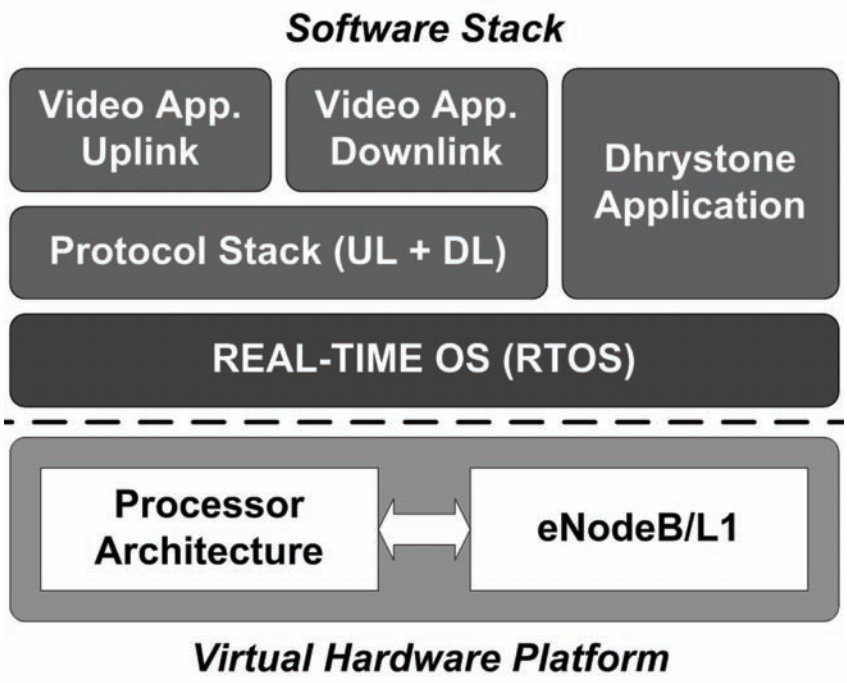
In this article we analyze the performance of LTE protocol processing on an ARM based mobile phone platform by applying a virtual system prototype (VSP) whose structure is shown in Figure 1. The protocol stack model in the software stack contains the most complex part of the LTE layer 2 (L2) data plane in uplink (UL) and downlink (DL) direction. Video applications are used to generate the data load in both directions during simulations, while the Dhrystone application runs in parallel to the video processing in the protocol stack for a more realistic scenario. Both applications run on top of a real-time operating system (RTOS). The virtual hardware platform, however, consists of the processor architecture and a base station/physical layer peripheral (eNodeB/L1). The latter emulates a communication between a base station and the analyzed mobile device. It therefore enables appropriate simulations in a fully controlled and closed environment where the protocol stack can be analyzed on transport block level at different transmission conditions associated with different computational demands on the terminal side.

The article is organized as follows: We first describe the virtual hardware platform and give details on the processor architecture and the eNodeB/L1 peripheral. Afterwards, an insight to the LTE L2 UL/DL protocol stack model and an introduction to the freeRTOS™ real-time operating system are provided. Then we briefly explain the profiling workflow, followed by the presentation of the simulation results. Finally, we give a conclusion in the last section.

VIRTUAL HARDWARE PLATFORM

A virtual hardware prototype is used in this work for agile investigations and execution of an LTE protocol stack model. Implementation, simulation and analysis of such a software model of a system on chip (SoC) are carried out using tools

Figure 1. Virtual system prototype

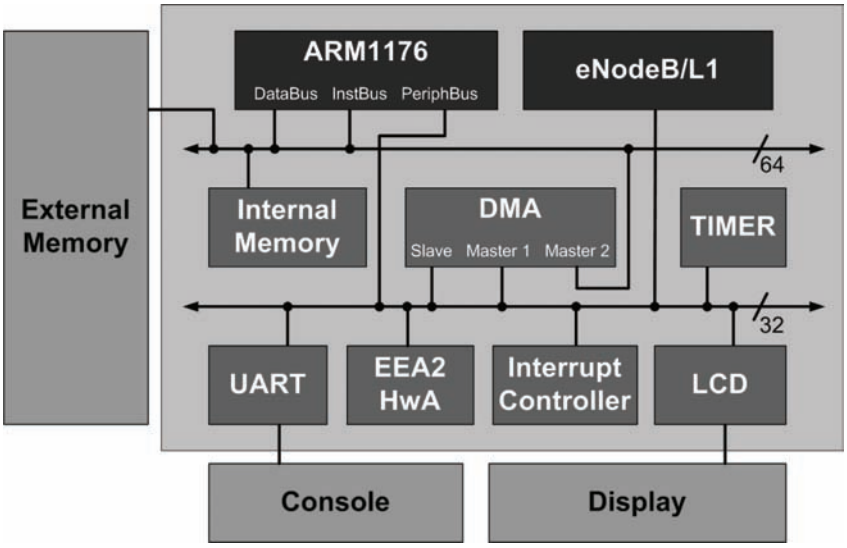


provided by VaST Systems Technology Corporation (Roan, 2006).

Processor Architecture

The processor architecture is a virtual prototype of a state-of-the-art mobile phone platform. An excerpt of it, with components in focus of this work, is illustrated in Figure 2. The platform is based on an ARM1176 embedded processor (ARM, 2008) which provides AMBA AXI bus interface ports. Detailed information about this bus protocol can be found in (ARM, 2004). A fast internal memory as well as an external memory with higher read and write latencies are connected to the processor via a 64-bit instruction and data bus. In addition, a 32-bit peripheral bus connects several devices to the core. All platform interrupts are combined and handled in an interrupt controller. Operating system interrupts are created by the timer. The eNodeB/L1 peripheral acts as a physical layer and base station emulator, generating and processing LTE transport blocks. A Direct Memory Access (DMA) controller reduces the processor load during the copy operation of received transport blocks from the eNodeB/L1 peripheral to the external memory before the LTE protocol stack processing in DL direction is accomplished. This is signaled with the start address of the actual transport block to the protocol stack which in turn triggers the protocol processing. On the other hand, generated transport blocks are copied with the DMA controller from external memory to the eNodeB/L1 peripheral in UL direction. Ciphering with the Advanced Encryption Standard (AES) (NIST, 2001), specified by the 3GPP in (3GPP, 2009a), is supported in the hardware platform and in the protocol stack by the EEA2 hardware accelerator (EEA2 HwA). In combination with the DMA controller the EEA2 HwA represents a conventional hardware acceleration concept (Silven & Jyrkkä, 2007). Furthermore, two user interfaces are integrated in the platform: A Universal Asynchronous Receiver Transmitter (UART)

Figure 2. Virtual prototype of an ARM1176 based mobile hardware platform



connected to a console and an LCD controller with an emulated display. During the simulation of LTE communication, a video is transmitted from eNodeB/L1 peripheral to the mobile platform and displayed using the LCD controller.

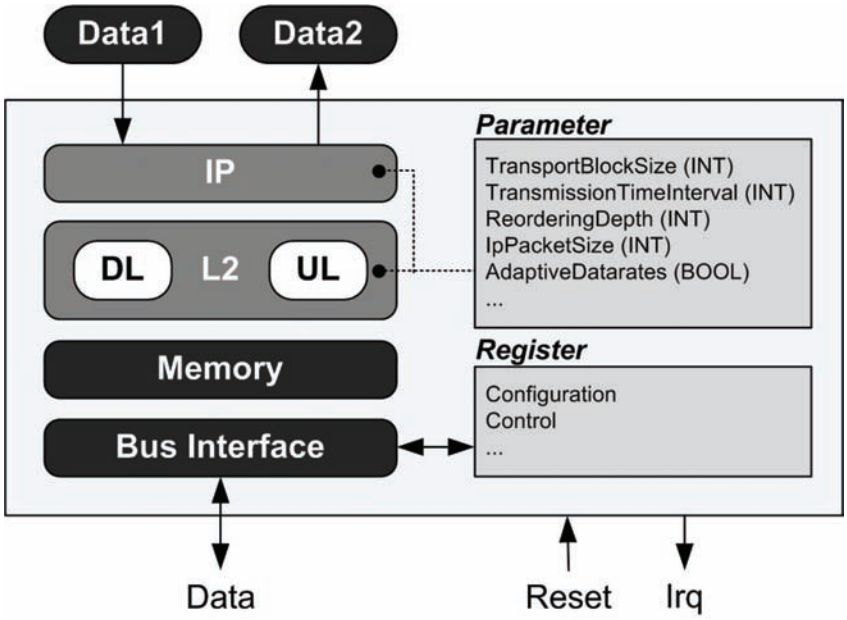
eNodeB/L1 Peripheral

The simplified functionality of an LTE physical layer and of a base station additionally including other entities of the core network are emulated by the eNodeB/L1 peripheral that allows for LTE protocol investigations on transport block level in the closed virtual prototyping environment. Implementation and integration in the hardware platform is carried out using the C programming language and Architectural Modeling Programming Interface (AMPI), a proprietary C library for hardware modeling, provided by VaST.

An overview of the eNodeB/L1 peripheral is depicted in Figure 3. The functional core consists of a simplified IP layer and an LTE L2 data plane protocol processing unit for UL and DL directions. The IP layer features a file interface for data input and data output. In addition, the data output can be redirected for playback to an emulated display in case that video data is received from the simulated mobile terminal. In the L2 DL component, LTE transport blocks are generated from IP data and buffered in the internal eNodeB/L1 memory. On the other hand, the L2 UL processes receive LTE transport blocks, which are also available in the internal peripheral memory, before obtaining IP packets forwarded for further processing to the IP layer. The internal eNodeB/L1 memory as well as the registers for control, configuration, etc. are accessed via the integrated bus interface. In addition a reset port and an interrupt (Irq) port are implemented whereas the latter signals the completion of each created transport block available in the peripheral's internal memory. In order to account for different transmission conditions associated with different processing demands on the mobile terminal side, the IP and the L2 UL/DL protocol processing components are configured via parameters from the virtual prototyping environment before simulation.

The configuration parameters in the eNodeB/L1 peripheral mainly influence the data flow in DL direction as illustrated in Figure 4. The data flow starts with the generation of IP packets in the IP layer. For that purpose, an IP header is cre-

Figure 3. eNodeB/L1 physical layer and base station emulator for LTE transport block generation and processing in UL and DL direction



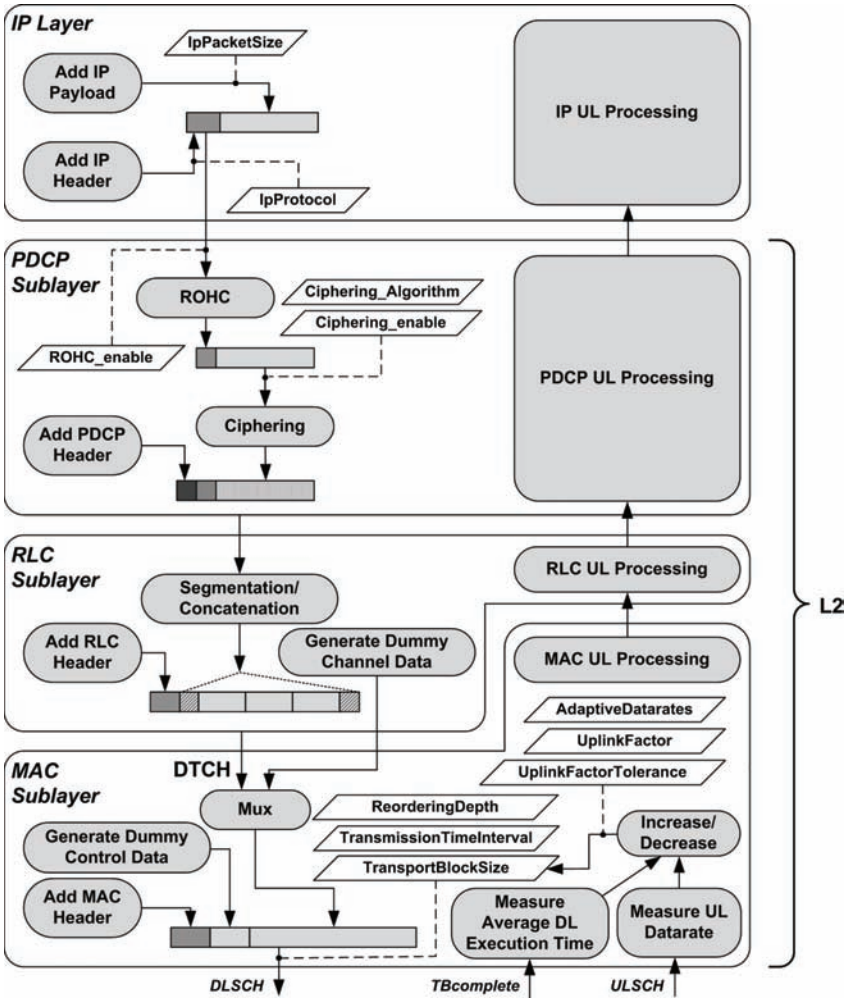
ated and arbitrary data from a video file is used as the IP payload. The total IP packet size and the IP protocol are configured via the *IpPacketSize* and *IpProtocol* parameter, respectively. Afterwards the generated IP packets are forwarded to the L2 DL unit that generates 3GPP Rel.8 compliant transport blocks (3GPP, 2009b, 2009c, 2009d) and stores them in the internal peripheral memory. The L2 DL unit is subdivided into three sublayers: Packet Data Convergence Protocol (PDCP), Radio Link Control (RLC) and Medium Access Control (MAC).

The processing in the L2 DL component starts in the PDCP sublayer. Before adding PDCP headers, the IP headers are compressed with the Robust Header Compression (ROHC) and the IP packet is encrypted depending on the *Ciphering_Algorithm* parameter by applying one of the provided ciphering algorithms. Although the eNodeB/L1 peripheral supports both algorithms which are specified by the 3GPP for ciphering in LTE (SNOW 3G and AES), only the AES is used in this work. Furthermore, it is possible to de-/activate the ROHC (*ROHC_enable*) and the ciphering (*Ciphering_enable*).

After the processing, PDCP packets are forwarded to the RLC sublayer. Here, the data transfer is carried out on the Dedicated Traffic Channel (DTCH). The data size of each RLC packet is typically determined by the physical layer transmission conditions. In our setup, however, the RLC packet size is simply 100 bytes smaller than the complete transport block size, since the main focus of this work is the profiling of the data plane without considering the physical layer and any physical channel information. Each RLC packet is filled with complete or segmented PDCP packets in the segmentation/concatenation unit. Segments occur either at the beginning and/or at the end of RLC packets. After adding an RLC header, dummy data for other logical channels is generated in order to increase the header processing complexity in the MAC sublayer and hence provide a more realistic scenario.

All logical channel data is multiplexed to the transport block payload in the MAC sublayer.

Figure 4. Data flow and parameters in the eNodeB/L1 peripheral



Furthermore, dummy MAC control data is appended to the beginning of the transport block and a MAC header is added to finalize the transport block generation. In addition to parameters for the transport block size (*TransportBlockSize*) and the transport block generation period (*TransmissionTimeInterval*), a reordering buffer is used to account for packet loss and retransmission in wireless links. The reordering buffer size is specified by the *ReorderingDepth* parameter. Generated transport blocks are copied in reverse order from the reordering buffer to the internal eNodeB/L1 Downlink Shared Channel (DLSCH) peripheral memory for software access.

Furthermore, the eNodeB/L1 peripheral offers a facility to determine the maximum UL and DL data rates. The parameter *AdaptiveDatarates* de-/activates the maximum data rates measurement. In case that both parameters for the UL data rate, *UplinkFactor* and *UplinkFactorTolerance*, are deactivated, the maximum DL data rate is determined without the consideration of the concurrently achievable UL data rate. In this mode, the current average execution time per transport block is measured during the simulation and the DL data rate is adapted accordingly by increasing/decreasing the transport block size. The DL execution time is the simulation time after transport block

generation until the completion of the transport block processing. The latter is signaled by the protocol stack by writing a predefined value to the *TBcomplete* register. The eNodeB/L1 peripheral computes permanently a moving average value for the current average DL execution time per transport block and decreases the transport block size if the average execution time per transport block exceeds the transmission time interval. The transport block size remains stable if the average execution time per transport block is within a 50 µs range below the assigned transmission time interval and the transport block size is increased in case that the average execution time per transport block falls below. After stabilization of the data rate the simulation stops and the maximum achieved DL data rate is plotted.

In order to measure the maximum data rates for a joint UL and DL scenario, the desired ratio between the UL and the DL data rate and the allowed tolerance in percent are configured with the *UplinkFactor* and *UplinkFactorTolerance* parameters, respectively. Then, the eNodeB/L1 peripheral measures the incoming UL data rate which is additionally taken into account for the adaptation of the DL data rate with regard to the setting of the described parameters and determines the maximum feasible UL and DL data rates.

SOFTWARE STACK

The software stack consists mainly of an LTE protocol stack model that is implemented on top of the freeRTOS™ real-time operating system. The protocol stack communicates with two video applications for sending and receiving video data. In addition, a Dhrystone benchmark application is executed in parallel in order to emulate the load of other applications and hence to achieve a more representative simulation scenario.

Protocol Stack Model

The protocol stack model implements the most complex and therefore execution time critical part of the LTE L2 UL/DL data plane. Thereby, mainly the inverse functionality of the L2 UL and DL parts of the eNodeB/L1 peripheral is implemented. The control plane is not considered in this work because it would have only a marginal impact on the profiling results. Compared to the data plane, only small amounts of data are processed less frequently in the control plane resulting in a significantly lower computational effort.

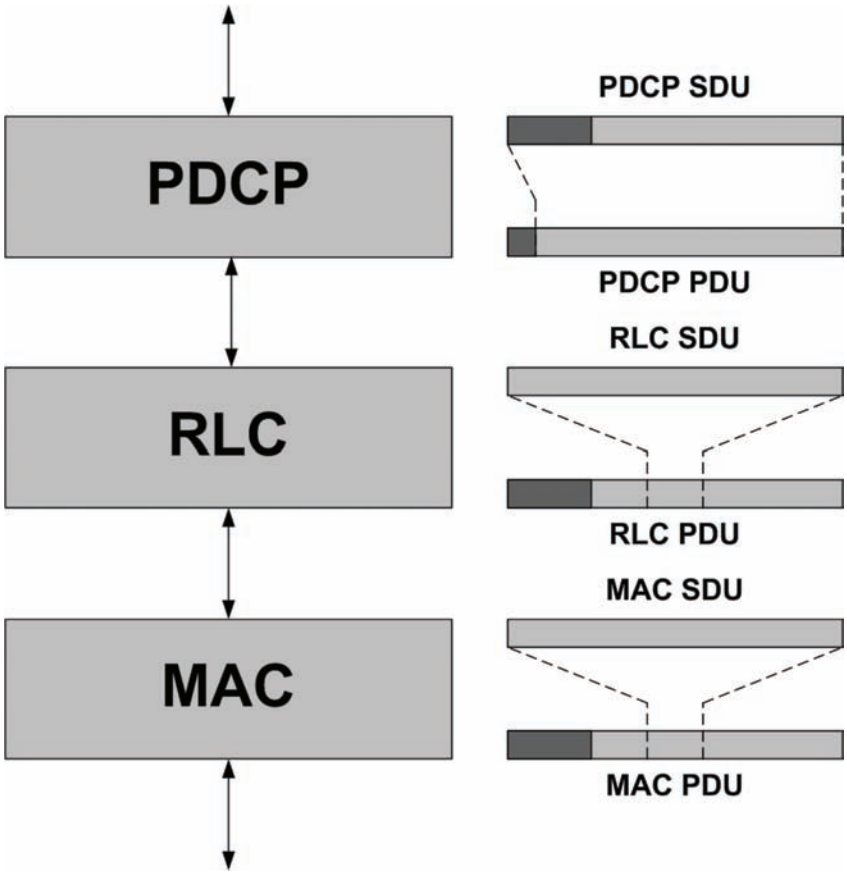
Three sublayers form the LTE L2 data plane: MAC, RLC and PDCP. Figure 5 shows the data structures within the LTE L2 data plane that is described for the DL direction in the following. Each MAC protocol data unit (PDU) comprises several MAC service data units (SDUs) which are transferred after MAC completion to the RLC sublayer. Here MAC SDUs are called RLC PDUs which similarly contain some RLC SDUs that represent PDCP PDUs in the PDCP sublayer. Finally, IP packets (PDCP SDUs) are obtained after processing in the PDCP sublayer and forwarded to higher protocol layers.

Input and output to the protocol stack model and communication between the sublayers within the protocol stack model are realized by pointers, so that the data in form of transport blocks remains in the original memory region. This avoids inefficient copying of large memory blocks which significantly reduces the execution time.

In the following, the implementation of each sublayer is described in detail.

1) **MAC:** In DL direction, MAC SDU information in form of data lengths and channel identification numbers is extracted during header processing applied on MAC PDUs. Additionally, the address offsets identifying the MAC SDU positions in the transport block are calculated. Based on this data, each

Figure 5. Overview of the LTE protocol stack model



MAC SDU is assigned to the corresponding logical channel signal in the demultiplexing module and forwarded to the RLC sublayer (see Figure 6, bottom). In UL direction, logical channel data from the RLC sublayer is multiplexed into the payload of a MAC PDU before the MAC header is generated and the resulting transport block is copied to the physical layer interface.

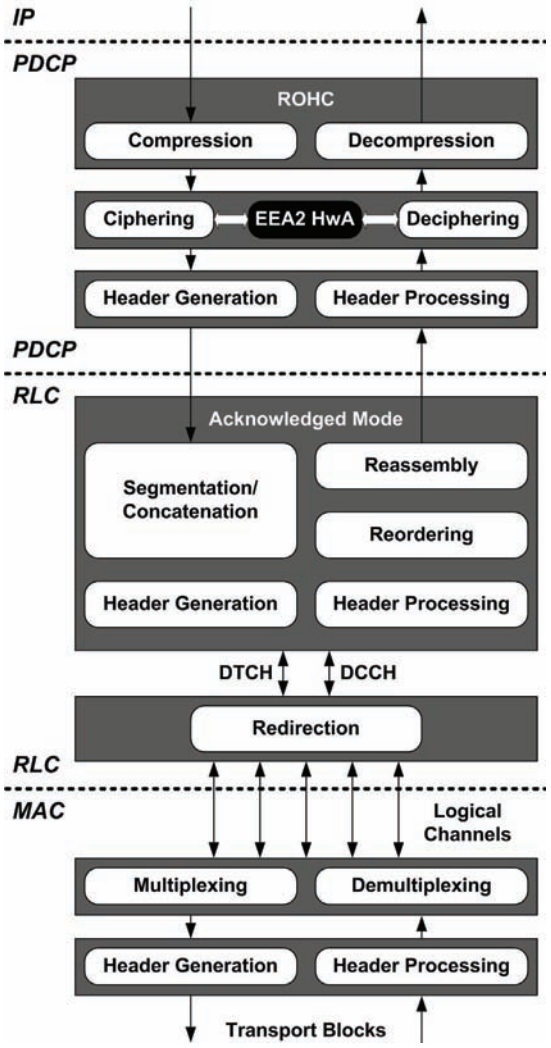
2) **RLC:** Three different modes of operation are provided by the RLC sublayer: Transparent Mode (TM), Unacknowledged Mode (UM) and Acknowledged Mode (AM). In the TM mode PDUs are directly forwarded to the PDCP sublayer without any processing in the RLC sublayer in order to achieve low latencies on the Broadcast Channel (BCCH), the Common Control Channel (CCCH) and the Paging Control Channel (PCCH). This dummy channel information generated in the eNodeB/L1 peripheral is directly discarded in the redirection unit of the protocol stack model. Furthermore, the UM and AM modes are specified by the 3GPP for the DTCH channel. Besides additional features like RLC packet resegmentation support in the AM mode, the main difference between AM und UM is the packet acknowledgement capability.

Because of its higher complexity, and accordingly higher impact on the processing demand on mobile terminal side, only the AM mode is implemented in the protocol stack model (see

Figure 6, middle). Therefore the communication between the MAC and the RLC sublayer is carried out on the DTCH and on the Dedicated Control Channel (DCCH), whereas the latter contains dummy control information that is discarded in DL direction. The UL processing in the RLC sublayer is almost identical to the DL processing in the eNodeB/L1 peripheral; hence the following description focuses on the DL direction of the RLC sublayer. Dynamic memory allocation provided by the operating system is used to store the decoded header data for each RLC PDU as well as all included RLC SDUs in linked lists for further processing in the protocol stack model. Based on the sequence numbers of the received RLC PDUs, the reordering is carried out with a simple insertion sort algorithm. Finally, segments of RLC SDUs need to be reassembled in the corresponding entity before being transmitted to the PDCP sublayer. For that purpose, the DMA controller is used to copy the appropriate segments to one continuous memory region representing complete RLC SDUs.

3) **PDCP:** Like the UL part in the RLC sublayer, the UL processing in the PDCP sublayer is almost equal to the DL processing in the eNodeB/L1 peripheral with only two differences. First, no parameters are adopted to control the UL PDCP processing on the mobile terminal side and second, ciphering is carried out in the EEA2 hardware accelerator. Hence, the following detailed description of the PDCP sublayer in the protocol stack model focuses on the DL direction as well. Each received PDCP PDU is processed in the PDCP sublayer in the following order: header processing, deciphering and ROHC decompression (see Figure 6, top). After decoding the header information, the encrypted payload of each PDCP PDU is deciphered using the EEA2 hardware accelerator. In addition to the cipher key and the PDCP PDU header information, a hyper frame number (HFN) is maintained in the protocol stack model needed for a correct decryption of the received data. For the deciphering, the data is copied to the EEA2 hardware accelerator by using a DMA controller and the decryption is started after the configuration of the appropriate registers. An interrupt signals the completion and triggers the reverse DMA copy of the deciphered data to the external memory. For

Figure 6. Implemented functionality of the MAC (bottom), the RLC (middle) and the PDCP sublayer (top) in the LTE L2 protocol stack model



measurements of the pure software performance, the deciphering can be deactivated before compilation. As the last computing step in the PDCP sublayer, the ROHC decompression is applied on the compressed IP header. For this purpose a modified and optimized version of the free implementation of ROHC available at (ROHC, 2003) is integrated in the protocol stack model. One major modification is the separation between the decompressed IP header and the payload, which avoids the inefficient creation of a continuous IP packet in the memory. Instead of that, memory is allocated only for the decompressed header whereas the payload remains in the original memory region. The extracted IP packets are forwarded to a very simple model of an IP layer, where only the checksum and the destination address are validated. Afterwards, the received video data is displayed using the LCD controller.

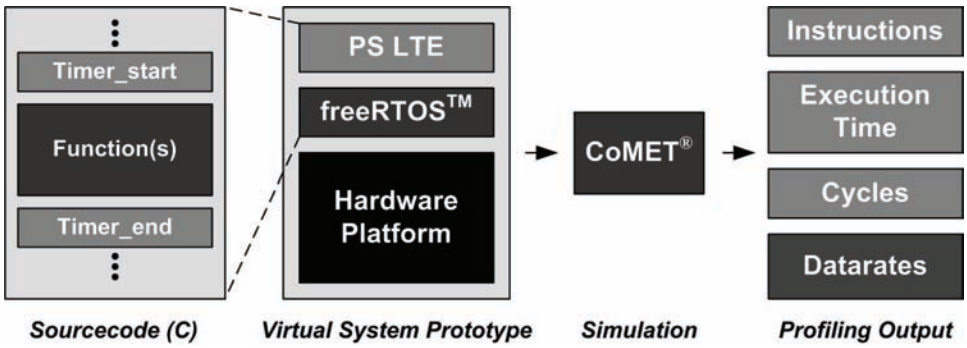
freeRTOS™

The open-source real-time operating system freeRTOS™ supports tasks and co-routines. Inter task communication and synchronization is established via queues, semaphores and mutexes. Furthermore, the kernel can be configured for preemptive, cooperative or hybrid mode. More detailed information can be found in (FreeRTOS, 2003). A real-time operating system in general is mainly characterized by low interrupt and task switching latencies required by real-time applications like an LTE protocol stack. The LTE L2 protocol stack model in this work is subdivided into six tasks with different priorities implemented on top of freeRTOS™. Each protocol sublayer is separated in two tasks for UL and DL processing, respectively. All DL tasks are executed on highest system priority level followed by the UL tasks, the video application tasks with integrated IP layer functionality and the Dhrystone benchmark task emulating the load of other applications executed in parallel. The kernel runs in the preemptive mode enabling task activation by receiving signals on queues. For instance a signal from the eNodeB/L1 interrupt handler activates the MAC DL task with the highest priority which first of all utilizes the DMA controller for copying the transport blocks to the external memory. After the completion of the MAC DL processing, a queue message containing for instance a pointer and the size of the data for further processing invokes the next task in the protocol stack and so forth.

PROFILING

A timing analysis of software execution is called performance profiling. The resulting identification of execution time critical algorithms is mandatory for an effective hardware/software partitioning in mobile embedded systems where real-time constraints have to be met. The virtual system prototype of the mobile platform is profiled with the CoMET® engineering environment from VaST (see Figure 7). VaST's API provides an interface from the software executed on a virtual processor architecture to the simulator which allows for exact timing measurements. In order to measure the execution time of determined parts in the protocol stack, timer functions are placed directly in the C code enclosing the dedicated functionality. In addition we use a Tool command language (Tcl) script, starting simulations with different parameters and extracting the timer values from the CoMET® output files afterwards. The evaluation is performed automatically by calculating the minimum, the average and the maximum values for the execution time, the number of instructions and clock cycles of target functions. Furthermore, the eNodeB/L1 peripheral allows for an analysis of the maximum data rates during simulation.

Figure 7. Profiling workflow of a virtual mobile platform using CoMET®



RESULTS

Simulation and profiling of the VSP is carried out in two scenarios, a timing analysis and a data rate analysis. First, the execution times for the UL and DL protocol stack processing are measured for a representative hardware configuration. Second, the maximum achievable protocol transfer rates are determined for different processor speeds and transmission conditions.

Timing Analysis

The platform is configured via parameters in the VSP before simulation and measurement. The ARM1176 processor clock and the bus clocks are set to 450 MHz and 200 MHz, respectively. This represents slightly increased clock frequencies compared to the state-of-the-art mobile platform presented in (Hessel et al., 2008). Furthermore, the processing time per byte of the EEA2 hardware accelerator for the de-/ciphering is set to 10 ns according to realistic timing of the hardware implementations presented in (Hessel et al., 2009). A DL data rate of 100 MBit/s is generated by the eNodeB/L1 peripheral with transport block sizes of 100 kbits and a transmission time interval (TTI) of 1 ms. The parameter for the reordering depth is configured with a value of two, resulting in a sequence order of 2, 1, 4, 3, 6, 5 and so forth at transport block generation, leading to a higher processing complexity in the LTE protocol stack. In this work we measure the average execution times per transport block of the L2 data plane, the sublayers and the functions within the sublayers in the LTE protocol stack model by simulating the system for 1 s at 8 kB, 16 kB, 32 kB and 64 kB data and instruction cache sizes in the ARM1176 processor. Because of operating system functionality, the average processing time values of protocol stack components are not necessarily obtained by adding up the average processing time values of all their subcomponents.

For the timing analysis we investigate three different scenarios:

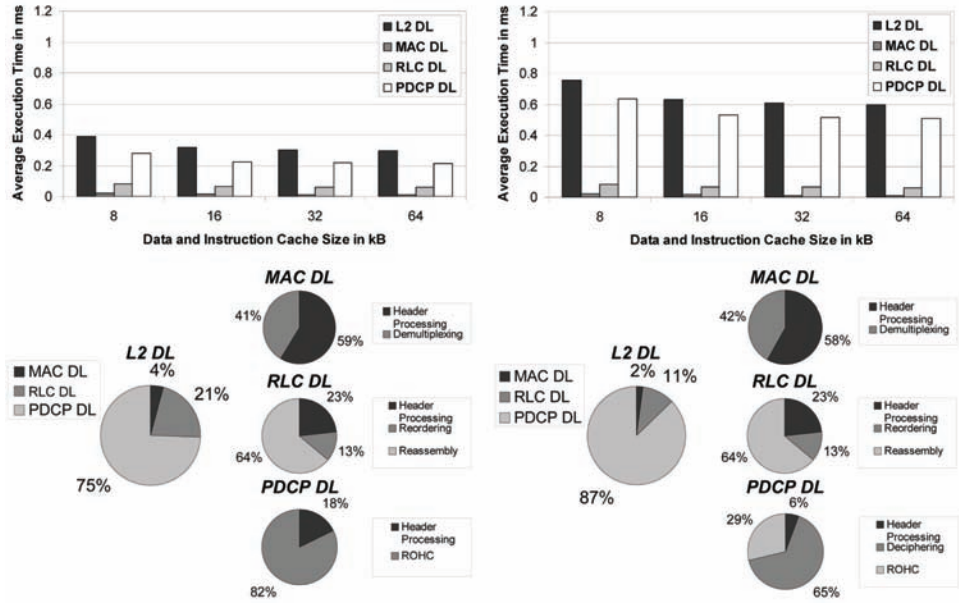
- **Scenario I (DL):** Only the DL tasks and the Dhrystone benchmark application task are activated in order to measure the execution times in DL direction.
- **Scenario II (UL):** Only the UL tasks and the Dhrystone benchmark application task are activated in order to measure the execution times in UL direction.
- **Scenario III (UL & DL):** The full software stack functionalities are simulated in order to measure the total impact on the average execution times in DL direction.

Additionally, the measurements are accomplished in two variations: de-/ciphering disabled or enabled. The former is used to present profiling

results of pure software execution without hardware support for the de-/ciphering in the EEA2 peripheral. The timing analysis of the complete UL and DL data plane in the LTE protocol stack model is carried out in the de-/ciphering enabled mode. Finally, it should be noted, that the software stack is compiled with a GCC version 4.3.2 cross compiler at the highest optimization level (-O3) in all measurement setups. Even though the complete binary size is about 5% bigger compared to compilation with the medium optimization level (-O2), the measured processing times are approximately 5% faster for 8 kB caches and about 1% faster for 64 kB caches. The space optimization option (-Os) even produces a slightly increased binary size with lower performance compared to the medium optimization level.

Scenario I (DL): The average execution time results and their relative distributions in the LTE L2 protocol stack model in DL direction with de-/activated decryption are illustrated in Figure 8. With the deciphering disabled in the protocol stack model, an increase of the data and instruction cache sizes to 64 kB leads to a reduction of the average execution time per transport block of the whole L2 DL from 391 µs to 293 µs resulting in a speedup of approximately 25%. Increasing the cache sizes beyond 32 kB only leads to a marginal performance improvement of about 2%. Consequently, the relative distribution of the average execution time among the L2 DL subcomponents and of the functions within each subcomponent is shown for the reasonable test case with 32 kB data and instruction cache sizes. The computational effort of the L2 DL is mainly consumed by the PDCP sublayer at 75%, followed by the RLC and the MAC sublayers at 21% and 4%, respectively. The high processing demand in the PDCP sublayer is mainly caused by the ROHC decompression (82%) and by the header processing (18%) measured in total for all PDCP PDUs contained in one transport block. Therefore, the ROHC decompression represents the most computational intensive software algorithm in

Figure 8. Average L2 DL execution times in the LTE protocol stack model and their relative distributions at 32 kB data and instruction cache sizes for the entire L2 and the sublayers in detail: deciphering disabled (left), deciphering enabled (right)



the LTE protocol stack model in DL direction with a total demand of 62% of the entire L2 DL execution time.

In the correspondent measurements carried out in DL direction for the deciphering enabled scenario, the average L2 DL execution time per transport block is higher compared to the previously presented measurements because of the additional processing demand for the deciphering in the protocol stack model (606 µs for 32 kB cache sizes). Moreover, the dependency of the PDCP sublayer execution time on the total L2 DL execution time is increased to 87%. The deciphering consumes thereby 65% of the average PDCP sublayer execution time per transport block followed by the ROHC decompression and the header processing at 29% and 6%, respectively. Although the deciphering is carried out in the hardware accelerator, bus demanding data transfers are required for the copying of the data to and from the peripheral, strongly affecting the execution time. With regard to the LTE requirements with a maximum DL data transfer rate of 100 MBit/s, where transport blocks are received and processed periodically in a transmission time interval of 1 ms, the complete L2 DL data plane processing demands already about 60% and up to approximately 75% of the available execution time depending on the different data and instruction cache size configurations in the ARM1176 processor.

Scenario II (UL): Comparable results are obtained from the profiling of the LTE protocol stack model in UL direction (see Figure 9). The total L2 UL average execution time per transport block nearly corresponds in the ciphering disabled (285 µs at 32 kB cache sizes) and ciphering enabled (567 µs at 32 kB cache sizes) scenarios to the measurement results obtained for the whole L2 in DL direction. Moreover, the computational effort of the PDCP sublayer is again mainly responsible for the entire L2 UL average execution time per transport block with a total demand of 86% and 93% for the ciphering disabled and ciphering enabled scenario, respectively. The UL functionality in the LTE data plane is mainly the inverse operation of the LTE DL; hence the corresponding UL algorithms and protocol stack components have comparable computational efforts. The most execution time demanding software algorithm is the ROHC compression and requires about 80% of the entire L2 UL execution time.

Scenario III (UL & DL): In a real environment the LTE protocol stack communicates simultaneously in UL and DL directions with the base station, but a reasonable measurement of the execution times can only be carried out by activating the L2 UL or DL separately. However, setting all DL tasks to higher priorities than all other tasks running in parallel in the system including the UL tasks allows for a timing analysis of the L2 in DL direction in such a realistic scenario. The influence of the UL processing on the DL processing in the protocol stack is illustrated in form of a relative slowdown of the entire L2 DL average execution time per transport block at different data and instruction cache sizes in Figure 10.

The slowdown of the L2 DL caused by parallel execution of the L2 UL decreases with rising cache sizes for both deciphering configurations. The data and instruction cache is permanently filled with data corresponding to the currently executed task. Consequently, smaller caches provide especially in a multi task environment higher cache miss ratios whereas every cache miss leads to a slow memory access and thus to a reduced execution time. Furthermore, the impact of the L2 UL on the L2 DL execution times is higher in the deciphering enabled mode compared to the deciphering disabled mode. The reasons are more complexity in the protocol stack model and the concurrent and mutual exclusive usage of the hardware accelerator for the EEA2 de-/ciphering for both transfer directions. In the case of 32 kB data and instruction cache sizes, the average L2 DL execution time per transport block consumes in the realistic scenario about 66% of

Figure 9. Average L2 UL execution times in the LTE protocol stack model and their relative distributions at 32 kB data and instruction cache sizes for the entire L2 and the sublayers in detail: ciphering disabled (left), ciphering enabled (right)

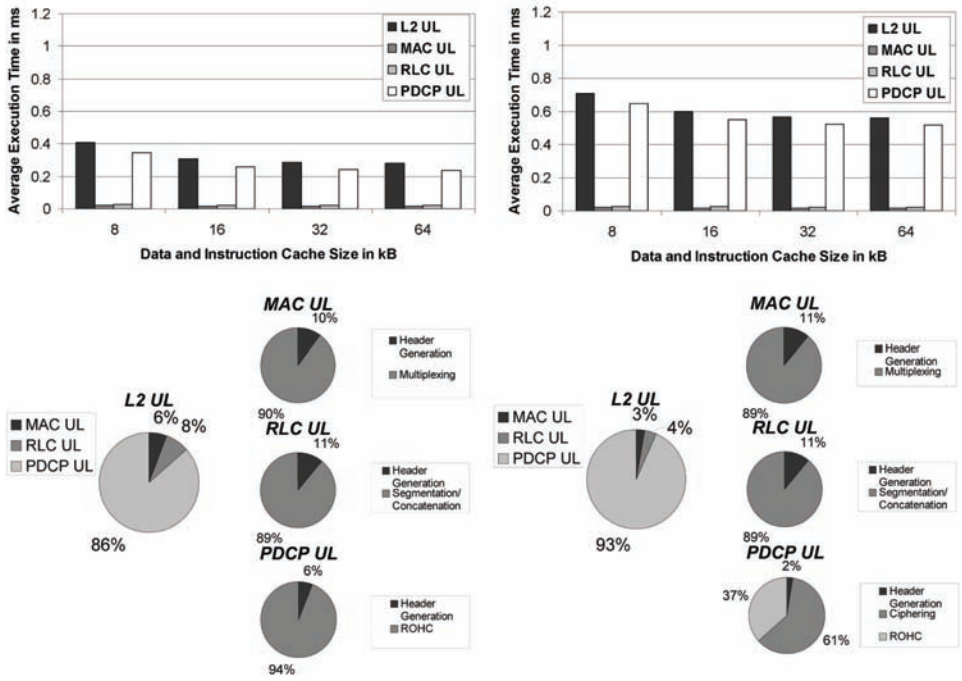
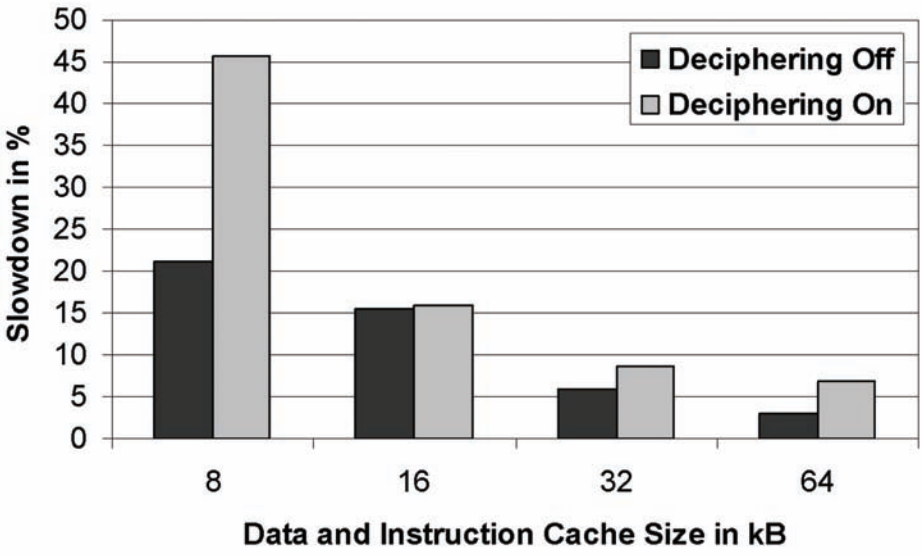


Figure 10. Relative slowdown of the average L2 DL execution time due to the parallel execution of L2 UL



the available processing time of 1 ms determined by the TTI.

Data Rates Analysis

For the data rates analysis the same hardware platform configuration is used. The simulations are carried out at different data and instruction cache sizes. Moreover, the de-/ciphering algorithms are enabled in all measurements.

First, the maximum data rates in DL direction are measured at different IP packet sizes varied from 100 bytes to 2000 bytes in 100 byte steps as depicted in the top diagram in Figure 11. The maximum data rates heavily depend on the IP packet size which in turn affects the number of PDCP PDUs located in a transport block. In this work, a fixed IP packet size is used in every simulation run. Hence, small IP packet sizes increase and big IP packet sizes decrease the number of concatenated PDCP PDUs per transport block. Because every PDCP PDU needs to be processed in the PDCP sublayer representing the most computational intensive protocol stack subcomponent, the entire protocol stack processing time per transport block grows strongly with the number of PDCP PDUs which results in a reduction of the maximum achievable data rate. LTE DL data rates of 100 Mbit/s can be achieved only for IP packet sizes equal or greater than 700 bytes and 64 kB caches.

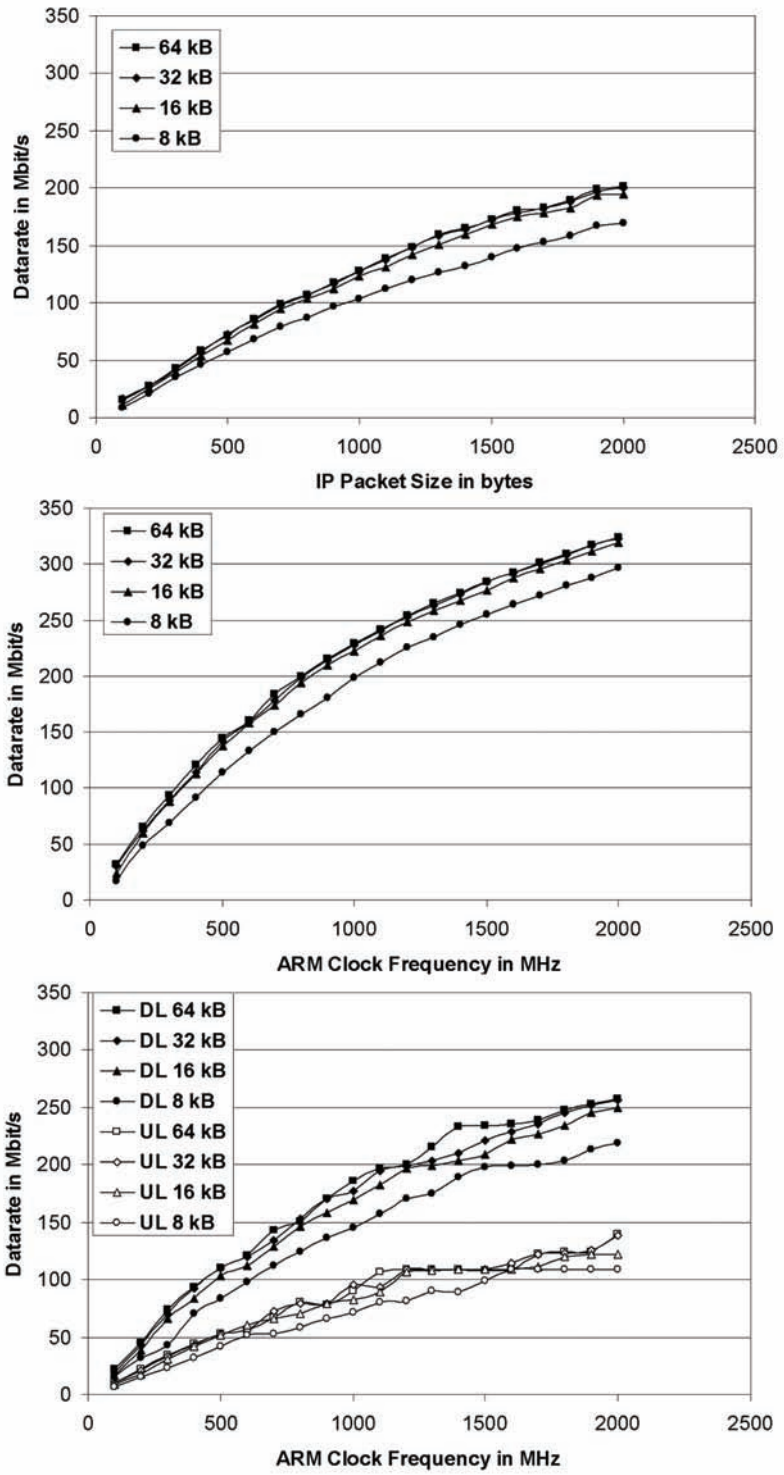
In the second setup, the data rates analysis is accomplished at different ARM1176 processor frequencies and a fixed IP packet size of 1000 bytes for the DL direction only and for both UL and DL directions communicating simultaneously with the eNodeB/L1 peripheral (see Figure 11 middle and bottom). The bus clocks are configured according to a factor of 2.25 between processor and bus clock frequency derived from the original hardware platform setup. Furthermore, for data rate simulations with parallel execution of the UL and DL in the protocol stack, the system stabilizes and generates the data rate output by reaching half of the maximum DL data rate in UL direction with an allowed tolerance of 10%. With regard to recent 3GPP specifications, the maximum data rates in DL direction are twice as high as the data rates in UL direction in LTE (100 Mbit/s DL, 50 Mbit/s UL) and in LTE-Advanced (1 Gbit/s DL, 500 Mbit/s UL) systems (Parkvall et al., 2008). In the most powerful hardware configuration with an ARM1176 processor clock of 2 GHz and 64 kB data and instruction cache sizes, the maximum data rate in DL direction amounts to approximately 320 Mbit/s. By running both protocol stack link directions simultaneously, maximum data rates of about 256 Mbit/s in DL direction and about 140 Mbit/s in UL direction are reached.

CONCLUSION

In this article, we present a detailed performance profiling of the LTE L2 protocol processing in mobile devices. We identify time critical stack functions and analyze the applicability of a conventional hardware accelerator using a single-core ARM11 processor in LTE and beyond LTE terminals. This is done with a virtual system prototype (VSP) of a state-of-the-art mobile hardware platform. In addition, the VSP contains a configurable base station and physical layer emulator for realistic data transmission in uplink (UL) and downlink (DL) direction. The software stack comprises a protocol stack model that represents the most execution time critical part of the LTE L2 data plane. The encryption algorithms are thereby accelerated by hardware.

A timing analysis of the pure software implementation (without deciphering) shows that most of the entire L2 processing time is consumed by the ROHC algorithms with a total demand of about 80% for the UL compression and about 60% for the DL decompression. This highlights the potential for performance improvement given by an acceleration of the ROHC algorithms, e.g. by hardware

Figure 11. Maximum DL data rates at different IP packet sizes (top) and maximum data rates at different ARM1176 processor frequencies in DL direction (middle) and simultaneously achieved in UL and DL directions (bottom)



support. With activated deciphering, however, the complete L2 DL processing per transport block requires already up to three-fourths of the available execution time. Regarding a simultaneous data transmission in both directions, an additional slowdown of up to 45% can even be observed.

The maximum performance of such a conventional mobile phone platform is demonstrated by the data rates analysis at different IP packet sizes and processor clock frequencies. At reasonable clock frequencies of about 450 MHz (Berkel, 2009), maximum LTE DL data rates of 100 Mbit/s can be achieved only with limitations on the IP packet size. Even with an unrealistic hardware configuration (i.e. processor clock frequencies of up to 2 GHz and 64 kB caches), only maximum DL data rates of approximately 320 Mbit/s are achieved. This would be still far below LTE-Advanced DL data rates of up to 1 Gbit/s.

Taking into consideration the overall processing demand of a complete protocol stack (e.g. considering the control plane and upper layers), it becomes clear that a mobile hardware platform even with a traditional hardware accelerator at reasonable processor clock frequencies is inappropriate for LTE terminals. Hence, instead of this conventional concept more sophisticated hardware accelerators (Szczesny et al., 2009) for the L2 processing will be needed to supply enough computational power required in LTE and especially in next generation mobile devices.

ACKNOWLEDGMENT

The authors acknowledge the excellent cooperation with all project partners within the EASY-C project and the support by the German Federal Ministry of Science and Education (BMBF). Further information is available on the project website: http://www.easy-c.de.

REFERENCES

ARM. (2004). *AMBA AXI Protocol Specification*. Retrieved October 31, 2009, from http://infocenter.arm.com

ARM. (2008). *ARM1176JZF-S Processor Technical Reference Manual*. Retrieved October 31, 2009, from http://infocenter.arm.com

Berkmann, J., Carbonelli, C., Dietrich, F., Drewes, C., & Xu, W. (2008). On 3G LTE Terminal Implementation - Standard, Algorithms, Complexities and Challenges. In *Proceedings of the International Wireless Communications and Mobile Computing Conference (IWCMC '08)* (pp. 970-975). Washington, DC: IEEE.

Brandenburg, M., Schollhorn, A., Heinen, S., Eckmüller, J., & Eckart, T. (2007). From Algorithm to First 3.5G Call in Record Time - A Novel System Design Approach Based on Virtual Prototyping and its Consequences for Interdisciplinary System Design Teams.

Cockx, J. (2000). Efficient Modeling of Preemption in a Virtual Prototype. In *Proceedings of the 11th International Workshop on Rapid System Prototyping (RSP 2000)* (pp. 14-19). Washington, DC: IEEE Computer Society.

Dahlman, E., Parkvall, S., Sköld, J., & Beming, P. (2008). *3G Evolution: HSPA and LTE for Mobile Broadband*. Oxford, UK: Academic Press.

Eckart, T., & Schnieringer, M. (2006). Development and Verification of Embedded Firmware using Virtual System Prototypes. In *Proceedings of the International Symposium on System-on-Chip (SoC 2006)* (pp. 1-1). Washington, DC: IEEE.

FreeRTOS. (2003). *The FreeRTOS™ Project*. Retrieved October 31, 2009, from http://www.freertos.org

Gajski, D. D., Samar, A., Gerstlauer, A., & Schirner, G. (2009). *Embedded System Design: Modeling, Synthesis and Verification*. New York: Springer.

Gehrmann, C., & Ståhl, P. (2006). Mobile Platform Security. *ERICSSON REVIEW. The Telecommunications Technology Journal*, *84*(2), 59–70.

3GPP. (2009a). *3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3GPP System Architecture Evolution (SAE): Security Architecture (Release 9)*. Retrieved October 31, 2009, from http://www.3gpp.org/ftp/Specs/html-info/33401.htm

3GPP. (2009b). *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Medium Access Control (MAC) Protocol Specification (Release 9)*. Retrieved October 31, 2009, from http://www.3gpp.org/ftp/Specs/html-info/36321.htm

3GPP. (2009c). *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Radio Link Control (RLC) Protocol Specification (Release 8)*. Retrieved October 31, 2009, from http://www.3gpp.org/ftp/Specs/html-info/36322.htm

3GPP. (2009d). *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Packet Data Convergence Protocol (PDCP) Specification (Release 8)*. Retrieved October 31, 2009, from http://www.3gpp.org/ftp/Specs/html-info/36323.htm

Hausner, J., & Drewes, C. (2008). Cellular Phones: Future Implementation Trends. *Journal of RF-Engineering and Telecommunications*, *62*(7-8), 185–190.

Hessel, S., Bruns, F., Bilgic, A., Lackorzynski, A., Härtig, H., & Hausner, J. (2008). Acceleration of the L4/Fiasco Microkernel Using Scratchpad Memory. In *Proceedings of the First Workshop on Virtualization in Mobile Computing (MobiVirt '08)* (pp. 6-10). New York: ACM Press.

Hessel, S., Szczesny, D., Traboulsi, S., Bilgic, A., & Hausner, J. (2009). On the Design of a Suitable Hardware Platform for Protocol Stack Processing in LTE Terminals. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering (CSE '09)* (pp. 1-8). Washington, DC: IEEE Computer Society.

In Proceedings of the Design. *Automation & Test in Europe Conference (DATE '07)* (pp. 1-3). Washington, DC: IEEE.

Lueftner, T., Berthold, J., Pacha, C., Georgakos, G., Sauzon, G., & Hoemke, O. (2007). A 90-nm CMOS Low-Power GSM/EDGE Multimedia-Enhanced Baseband Processor with 380-MHz ARM926 Core and Mixed-Signal Extensions. *IEEE Journal of Solid-state Circuits*, *42*(1), 134–144. doi:10.1109/JSSC.2006.886528

Manolakis, K., Ibing, A., & Jungnickel, V. (2008). Performance Evaluation of a 3GPP LTE Terminal Receiver. In *Proceedings of the 14th European Wireless Conference (EW '08)* (pp. 1-5). Berlin: VDE Verlag.

Martín-Sacristán, D., Monserrat, J. F., Cabrejas-Penuelas, J., Calabuig, D., Garrigas, S., & Cardona, N. (2009). On the Way towards Fourth-Generation Mobile: 3GPP LTE and LTE-Advanced. *EURASIP Journal on Wireless Communications and Networking,* 1-10.

NIST. (2001). *FIPS Publication 197: Advanced Encryption Standard (AES), National Institute of Standards and Technology (NIST)*. Retrieved October 31, 2009, from http://csrc.nist.gov/publications/PubsFIPS.html

Parkvall, S., Dahlman, E., Furuskar, A., Jading, Y., Olsson, M., Wanstedt, S., & Zangi, K. (2008). LTE-Advanced - Evolving LTE towards IMT-Advanced. In *Proceedings of the 68th IEEE International Vehicular Technology Conference (VTC 2008)* (pp. 1-5). Washington, DC: IEEE.

Roan, R. L. (2006). Optimizing Architectures for Performance and Area Using Virtual System Prototypes. *Embedded Technology Journal*. Retrieved October 31, 2009, from http://www.embeddedtechjournal.com/articles_2006/20061107_vast.htm

ROHC. (2003). *A Free Implementation of ROHC – Robust Header Compression*. Retrieved October 31, 2009, from http://rohc.sourceforge.net

Silven, O., & Jyrkkä, K. (2007). Observations on Power-Efficiency Trends in Mobile Communication Devices. *EURASIP Journal on Embedded Systems,* 8-17.

Szczesny, D., Hessel, S., Bruns, F., & Bilgic, A. (2009). On-the-fly Hardware Acceleration for Protocol Stack Processing in Next Generation Mobile Devices. In *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '09)* (pp. 155-162). New York: ACM.

van Berkel, C. H. (2009). Multi-Core for Mobile Phones. In *Proceedings of the Design, Automation & Test in Europe Conference (DATE '09)* (pp. 1260-1265). Washington, DC: IEEE.

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 4, edited by Seppo Virtanen, pp. 21-39, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 8
Embedded Networks in Mobile Devices

Sergey Balandin
Nokia Research Center, Finland

Michel Gillet
Nokia Devices, Finland

ABSTRACT

The concept of a mobile phone has recently transformed into a new concept of mobile multimedia devices capable of performing multiple complex tasks and integrating multiple functionalities. It has resulted in a significant increase of device integration costs and complicated deployment of new technologies. Device integrator companies favor modularity everywhere possible, which results in a new trend toward networked architectures for the mobile devices. However, comparing to the best-known embedded network solutions, e.g., SoC and NoC, these architectures have unique constraints and requirements, which also are significantly different from the wide area networks. The main constraints are power consumption and having a modular architecture to allow reuse of the components. Transition to the new architectures for mobile devices is a time consuming task that requires the analysis of many solutions applied in other contexts, especially for embedded protocols, QoS and resource management. This article reviews the state of the art in embedded networks research and the key assumptions, restrictions and limitations faced by designers of embedded networks architectures for mobile devices.

INTRODUCTION

Mobile industry is in the mist of a radical transformation and shift of paradigm from monolithic device that provides certain application to a modular solution that addresses all user needs. Recent development of various technologies, for example, media and multimedia convergence, easy access to internet and its services, GPS and indoor positioning, smart spaces concept, advances in the low power CMOS technologies, memory integration and high bandwidth radios, are blurring more and more, from a user perspective, the differences between a laptop, a mobile handset, photo and video cameras and other devices, which only a few years back were seen as independent

DOI: 10.4018/978-1-4666-0912-9.ch008

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

consumer electronics. At the same time, it is apparent from a technical perspective that the lines separating these areas are more blurred, which makes it difficult to predict with any accuracy how the mobile devices will look in 5-10 years and what will be their architecture.

The handset manufacturers now face a situation when there is essentially no standardized system interconnect or multi-purpose hardware interface, which could be the basis of a scalable and future proof architecture designed for ultra low power. It is important to note that having to deal with many different interfaces has a price—a need to use more complex chips with a large number of pins and additional functions for sharing pins make their usage mutually exclusive.

The monolithic nature of the currently available solutions is rooted in the trend to integrate as much as possible in one chip for increasing physical efficiency and reducing costs. But this approach has shown already its limits and lack of flexibility, as the complexity of monolithic chips steadily increase when combined with the fact that there is no multipurpose hardware interface resulting in a rigid architecture. When it takes 2 to 3 years for setting down the requirements and have the chip in a product, the user expectations have often changed significantly during that period. Moreover, one may need to wait several years to be able to have a new monolithic chip solution that fulfills the ever-changing needs.

An example of such mobile device is shown in Figure 1, where the main chip IC #0 includes many different functionalities and has only application specific hardware interfaces for camera, display, mass storage and various other components or ICs shown as App X to Z. When there is a need to add a new functionality, e.g., "Space travel", there are essentially only 3 possibilities:

1. Fit this new functionality at the last minute in the main chip IC #0, which is rarely a commercially viable solution for the chip manufacturer.
2. Use a proprietary hardware interface, which is simpler for the chip manufacturer, but obviously not very interesting for a mobile device integrator.
3. Try to hack a solution by re-using some existing hardware interfaces that were not designed for such purpose. This approach is rarely to succeed as in the long run the architectural disadvantages outnumber the short term benefits.

As a first step solution that simplifies the problem, dies stacking has been used in the mobile industry. However, there is no standard way to stack dies and no standard hardware interface for it. As a consequence, this step cannot answer alone the lack of architectural flexibility and sourcing flexibility.

If we look at laptops and netbooks, they have very strong standardized system interconnect and multipurpose hardware interfaces inherited from the PC world. These interconnects have often a bus or acyclic network topology as shown in Figure 2. Owing to the PC background, none of these interfaces was done for ultra low power. Their overall architecture of laptops or netbooks

Figure 1. One futuristic mobile device with a monolithic architecture

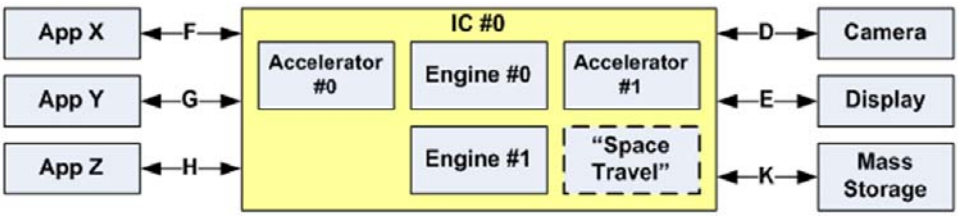
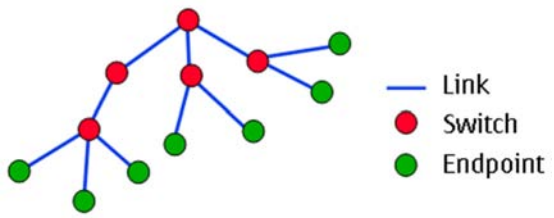


Figure 2. An example of acyclic network topology



is essentially dictated by the system interconnect, which is the opposite situation found in mobile devices, since none is available. But it might be interesting to mention that the big players in PC industry are also heavily working on integrating more functionality into a single chip.

By now, it has been clearly recognized that the mobile industry would greatly benefit from introducing an optimized interface technology for mobile devices similar in principle to ISA and PCI Express in the PC world. However, the requirements for such an interface are significantly different from the ones, which lead to the design of ISA or PCI Express. The main reason for that is naturally to be found in the intrinsic difference of environment in mechanical and electrical terms. For example, low power consumption is an essential technical requirement, as the target is to have mobile devices, which can operate several days or weeks on a single charge of the battery and be safe for the user.

Since by its nature mobile devices are mechanically compact, the low EMC/EMI emissions are paramount to avoid interferences with the antennas, as well as very low power per bit as the battery volume is limited. A significant part of the silicon chip cost and mobile device assembly work is linked to the number of pins, so the higher ratio of bandwidth per pin the better.

Another important issue is the large number of players in the mobile industry, which forces any generic interface to support, enable and match different architectures and different modular design techniques, so there is a strong requirement to enable various approaches to design modular architectures.

An important aspect is, of course, software integration and maintenance costs. The lack of standardized interface also means there is not really a unique way to handle interfaces from a software/kernel point of view. Indeed, the low-level drivers for communication between software and hardware of a given interface have too often to be tailor made virtually for every new interface.

To mitigate those issues, we have decided to address the problem from the networked protocols architecture point of view. This approach is very attractive for designers since by definition it is modular, communication centric and has a well-layered architecture.

This article provides an overview of the state of art in the embedded networks research and general background for our studies, key assumptions, restrictions and limitations that we faced at the beginning of designing the embedded networks architecture for mobile devices. The article is targeted to present our view on why the use of embedded networks is a key for further success of mobile device design. We also discuss the requirements and expectations for the dream architecture of mobile devices and map this perfect architecture to the currently available solutions to see the most critical issues that must be studied extra carefully in the architecture design. As a conclusion, we summarize our key findings that have to be taken into account while building embedded networks for mobile devices and provide references to the relevant standards in the field.

DESIGN OF MOBILE DEVICES

Definition of the Addressed Problem

Mobile devices are mass products, so a key criterion of device manufacturing is to be able to produce them at the lowest possible cost. In the context of this article, we set the focus and

scope only to the technical aspects impacting the resulting production costs. However, even with this scope definition the costs analysis is an extremely complex task, especially so that many factors are simply kept secret. To simplify the discussion, we start from easily measurable facts and their induced costs.

The first critical aspect is the pin count of Integrated Circuits (IC). With a constraint that mobile devices must be mechanically small, the IC packages must be small and thin. High pin counts and small packages invariably mean high pins or balls or studs density. To support these high densities, more advanced packaging technologies must be used, which increases the cost and also reduces the yield, which increases the costs even further. However, the cost increase does not stop at the IC manufacturing, as the pin count also impacts the cost of Printed Wiring Board (PWB), as the higher connection density means that more layers need to be used in the PWB. It also means that smaller vias demanding more expensive technologies, i.e., laser drilled buried or blind vias, have to be used. In addition, more layers in a PWB imply thinner individual layers, as the maximum thickness of PWB must be kept small. Thinner layers lower the characteristics impedances of the PWB traces, which introduce, for example, EMI/EMC issues and impedance matching issues for the high-speed traces and again increase the cost, e.g., as special shielding may be necessary. Higher pin density also has an impact on the yield and return rate of mobile devices, since making the soldering more difficult and soldering points more fragile. Also CMOS technologies are shrinking faster than packaging technologies. Thus, more complex ICs with fewer pins seems to be a more cost optimized solution.

Secondly, the number of different hardware interfaces for the main IC of a mobile device is huge, as currently there are dedicated hardware interfaces defined specifically for each application area, e.g., camera, display, modem, flash memories, GPS, radios, keyboard, SIM, audio, RAM and system control. Also for each application, usually there are alternatives, which must be supported to make the IC a viable product on the market. And since there is no multi-purpose standard hardware interface in the industry, the introduction of new features, requirements or applications often lead to the need to define yet another specific interface. Design, standardization, support, maintenance and verification of steadily growing set of interfaces require significant investments of time, money and best engineering resources. The amount of time needed to develop, standardize and adopt a new interface bound to a particular application is in average 2 to 4 years, and sometimes even more, depending on the interface complexity. It is then crucial for the industry to reduce the time needed to develop interfaces for new applications. The simplest way to achieve this is by using simple electrical IOs with poor bandwidth per pin ratio. However, the bandwidth requirements for interfaces are ever increasing, driven by the increased requirements set by the peripheral embedded devices, such as a megapixel camera race, faster flash technologies and HD support. As a consequence, if simple CMOS-like electrical IOs are used, it leads to an increase in the number of pins.

Thirdly, the system level design and integration induce great costs, and mistakes done at this phase create high risk to the product. When analyzing the basic requirements to keep costs under control, to have small package, maintain many different hardware interfaces and use small number of pins, we end up with a contradiction of requirements. The only solution is to multiplex the pins in such a way that the same pin supports 2 to 4 mutually exclusive configurations, and as a consequence use of some combinations of hardware interfaces may not be possible at all. Since only hardware interfaces rarely just happen to have the same number of pins, the result is a very complex scheme of sharing pins, which makes definition of a complete configuration of such IC extremely time consuming and error prone. Even worst, it

often sets artificially additional constraints on the configurations, which can be supported by IC.

The fourth aspect is the software integration. Today, many different hardware interfaces are used and they are defined and developed in different standardization committees and industry alliances. As a result, there is essentially no common software architecture for interface usage and the software integration becomes a tedious task since no commonalities exist. Part of the issue is the increased complexity of the mobile devices, which currently include a number of processors, which makes tracing and debugging software very tedious.

Finally, we are confronted with the monolithic architecture of the mobile devices, since no multipurpose hardware interface is defined. To illustrate this point let us take an example of a mobile device consisting of 2 movable parts; assuming that these parts can either rotate around an axis or multiple axes: this mechanical form factor is sometimes called "clamshell" or "flip form". Taking into account the constraints of size and volume, and that a mechanical structure of this kind is rather bulky compared to the size of IC, the area for placing the mobile device functionality has to be divided between the mobile device parts. At the same time these parts have to exchange a significant amount of data and control information for proper operation, which leads to the need for a significant amount of electrical connections, at least one for each type of interface used between the parts, which results in a complex and costly electro-mechanical structure. Thus, the fact that an interface can be used only for a single application leads either to increase the number of connections between the device parts or limits number of possible low cost partitioning.

A monolithic architecture essentially provides only one way to reduce costs - by extensive use of "one-chip solutions". But the time needed to develop complex one-chip solutions is increasing continuously, leading to a longer delivery time of the new technologies to the product. Even more, if a given technology misses the time window to be included in a given generation of the one-chip solution, it likely means that it will have to wait for the next generation to be included, or even can be completely dropped out of consideration.

Why Embedded Networks

The previous subsection discuses 5 technical aspects influencing the device cost. In this section, we take these aspects one by one and see what technical solutions can be used for reducing the cost. Let us start by the cost impact of IC with high pin counts.

At the electrical level, the only option to reduce number of pins is by increasing the throughput of a single pin. Electrically, it means moving away from wide parallel interface to narrow parallel interfaces with faster clock rate (i.e., HyperTransport) or to serial interfaces (i.e., USB). The fact that the electrical IOs needed for such purpose are more complex to develop and standardize, unavoidably forces the reuse of the same solution for several applications. If several applications reuse the same IOs, which are the PHY level from the protocol point of view, it is very tempting from an architecture standpoint to reuse also other low protocol layers, e.g., Link Layer. This observation results in a first small step towards embedded network for mobile devices.

The next issue is that currently there are too many hardware interfaces. By choosing to use more advanced PHYs, i.e., narrow parallel or serial IOs, the number of different HW interfaces should decrease and consequently should strengthen the reuse of the low protocol layers. The abusive multiplexing of pins should automatically be eased by the possibility to reduce the number of hardware interfaces and pins. The next logical step is to drive even further the architecture of mobile devices towards fewer hardware interfaces, where each interface type is designed to serve few different applications. When looking at the operation of a mobile device, it then means that

data will have to be exchanged between some of the hardware interfaces and it drives the need for a networked solution.

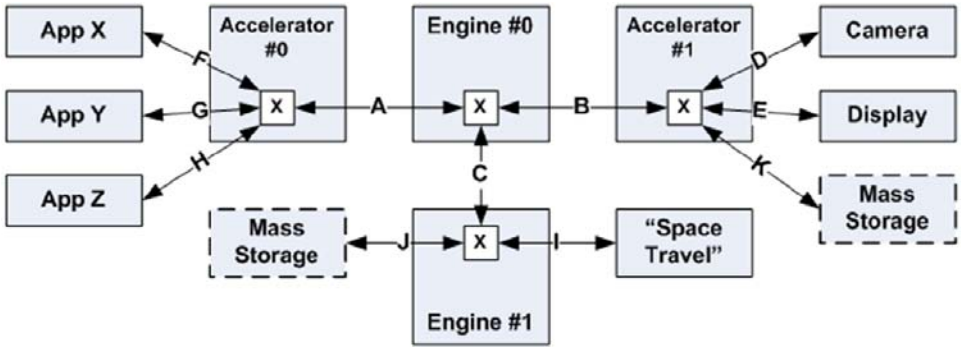
From a system level design and software integration point of view, moving from a monolithic architecture with many interfaces towards a networked architecture with few interfaces opens many opportunities. By being communication centric, the networked architecture simplifies both system and software integration by reusing the same communication layer for many applications. The hardware interface with its protocols and software are becoming a commodity and a developer of new applications can completely focus on the added value of this new application. To see what it means, let us redraw the monolithic architecture presented in Figure 1 by replacing it with a networked architecture as it is illustrated by Figure 3. The provided functionalities are identical in both cases. Also, we purposely avoided to show the physical component in this figure, since this same futuristic mobile device will be used later to exemplify certain physical configurations.

OVERVIEW OF THE EXISTING EMBEDDED NETWORKS

Unfortunately, the mobile industry currently does not have a system interconnect similar to ISA (IEEE, 1990), USB (USB, 2009), PCI (PCI-SIG, 2008) or PCI Express (PCI-SIG, 2009). Although to be absolutely accurate, there is a special USB version or extension of USB called USB HSIC, which is a chip-to-chip USB done by replacing the analog PHY by a set of CMOS wires, but we will discuss later some reasons why it does not fulfill our needs. The fact that no generic multipurpose hardware interface exists has strong historical reasons. The need to optimize device performance and power as much as possible often leads to a set of local optimizations, which does not necessary mean a globally optimized system. But there are also strong commercial reasons: when a company has a sizeable share of a market, having a proprietary interface is a clear locking point making harder for customers to change providers. Back to the technical aspects, the consequence of this is the undisputed dominance of monolithic architectures in current mobile devices, which include extension busses, mainly available as I2C (Philips Semiconductors, 2000) and SPI (Schwerdtfeger, 2000), which provide bandwidth of at most some Mbps. So, we can conclude that the current situation contradicts our original idea of easy and modular expansion of the device functionality. Let us see which upcoming technologies might solve the problem.

Indeed, in the mobile industry there is today a plethora of different interfaces used each for a single purpose. Even worse, if we take the display case as an example, at least half a dozen of

Figure 3. One futuristic mobile device based on embedded networks



different incompatible interfaces are defined. In comparison, in the PC world we can find a number of well know busses and embedded networks that have been used for many years, e.g.: ISA, PCI, USB, FireWire (LaCie, 2009), PCI-Express, SATA (SATA-IO, 2009) and eSATA. For something like USB, it is unclear whether it should really be counted as an embedded network technology, but here we assume that it is.

The most interesting standards from the point of view of this study are PCI-Express and USB. Both standards were designed with different requirements in mind and do not directly fit the mobile industry. On the other hand, USB is currently very popular as an external connection interface for mobile devices and other peripherals. Especially the USB 3.0 standard might carve out a niche in the mobile industry if it will be aligned with the currently developed core mobile device technologies. PCI-Express, as its name indicates, was designed and optimized to be backwards compatible with PCI, which was essential for its success in the PC world. But since the PCI interface is not used in mobile devices to connect external peripherals, the acceptance of PCI-Express for mobile devices is doubtful, especially that compromises had to be made to ensure its backwards compatibility to PCI. However, it is good to keep an eye on it as part of the trend of convergence of technologies with the PC industry.

Another very interesting technology has been developed by the space industry. SpaceWire is a standard for high-speed links and onboard networks for use in spacecraft. The standard is free for download from the ESA web page (ESA, 2009) and after a thorough study and modeling effort we have found that this technology has strong potential for intra- and inter-device communication. However, a number of restrictions made it sub-optimal for mobile devices. Among the most critical limiting factors one can mention that their PHY uses DS coding, which does not scale well in terms of single link bandwidth; the standard had minimal support for Network layer functionality and the Transport layer was not defined. The standard did not have Quality of Service (QoS) support; and finally uncertainty about the future of the technology forced us to drop it from the list of main technology candidate for mobile devices. Later, we got to know that the SpaceWire community is working on those missing aspects. It is clear that our requirements have many in common with the space industry, which may come as a surprise, but also there are big differences, which are likely irreconcilable.

As a result, the development of a set of new standards for the mobile industry has been started by the MIPI alliance (MIPI, 2009). One of these standards was targeting a PHY technology with 4 pins for a bi-directional link following a dual-simplex configuration with 1 Gbit/s bandwidth or more and ultra-low power consumption. There was no PHY solution that met these requirements, so the development of a new PHY was started. First, a more optimum signaling for PHY was proposed. The idea was to lower the swing, which is possible since there is no need to reach several meters inside a mobile device. As a result of the pre-study, the targeted solution had to have Bit Error Rate (BER) of 10^{-12} (which means 1 error every 15 minutes at link speed of 1 Gbit/s) for chip-to-chip connections. To allow lower power consumption, the BER could even be lowered to 10^{-9}. Given the choice of PHY and its BER, it was not possible to ignore or neglect the possibility of having errors during transmission as often done for simple CMOS hardware interface, i.e., SPI and some NoCs. As a consequence, the related protocol stack solution, called UniPro, has been introduced.

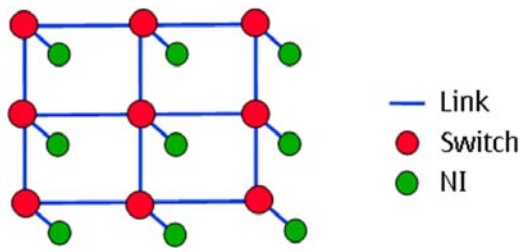
UniPro provides mechanisms for detecting errors and recovering from them, as well as many other capabilities such as node discovery, QoS and network management. It is important to mention that the current UniPro proposal combines point-to-point flow control and error recovery with the end-to-end flow control. UniPro provides many opportunities for the efficient handling of intra-

and inter-device connectivity. For example, in the intra-device case with relatively low BER and a small network radius, it makes more sense to use an end-to-end recovery scheme plus compensational buffers at the end points, while for the inter-device scenario a point-to-point recovery is more efficient.

The above discussed standard allows the integration of the mobile device core and the surrounding equipment by using the low level functionality provided by the standard. To address it, the idea of a wireless extension of our embedded network solution has recently emerged, but has not yet been discussed in a broad community. From the high-level point of view, this extension leads to the creation of a device federation concept, which would allow all surrounding devices to become logical sub-modules of the internal network of the mobile device. The provision of such a low-level device interconnect will allow to significantly speed-up communication and reduce power consumption, but any potential pitfalls of this approach are still to be discovered and investigated. But to simplify the story, hereafter we will limit the scope of this article to wired embedded networks, where the number of nodes in the network is in the order of tenth of nodes.

Another interesting technology to mention is the Network on Terminal Architecture (NoTA, 2009). NoTA is a modular service-based interconnect-centric architecture framework for embedded devices, with the basic underlying paradigm similar to what is used for Web Services. In the core of NoTA is a Device Interconnect Protocol (DIP) that can be implemented for various physical interfaces ranging from MIPI high-speed serial interfaces to wireless transports, e.g., Bluetooth. NoTA core, DIP and related system services are now open for all interested parties. A number of ideas of various services and solutions on top of NoTA have already been proposed. A number of related publications are available, and a good general overview and further references can be found in Lappetelainen (2008).

Figure 4. A classical NoC with a mesh topology



In the embedded networks pertinent to the mobile devices, there is arguably a clean cut between in one hand NoCs, e.g., Æthereal (Goossens, 2005), Xpipe (Dall'Osso, 2003), Nostrum (Millberg, 2004) and SPIN (Adriahantenaina, 2003); and, on the other hand, inter-chip solutions, e.g., USB, RapidIO® and PCI Express®. This clean cut can be seen and approached from different angles. (See Figure 4)

If we start by the supported topologies, arguably the most common for NoC is a mesh topology. Since a 2D mesh is very common in NoC, much work has been done to compare the efficiency and cost of various routing algorithms, e.g., in (Neuenhahn, 2008). Other topologies like thorus, 3D meshes are also attractive for NoCs. But by their very essence, it tends to focus the work around homogenous systems. In an off-chip embedded network, links are very expensive and a mesh topology is then out of the question: topologies are very often acyclic trees, e.g., USB. A notable exception is SpaceWire, where the link redundancy is critical due to specifics of space applications. But nonetheless, heterogeneous network is the norm for chip-to-chip. At first sight, this may seem a detail, but NoCs are usually designed with very little buffering in the switch or router or routing-switch. This approach is essentially made possible by the dense level of interconnections offered by meshes between NIs and switch: the number of switch is identical to the number of NIs, at least in a 2D mesh. For off-chip embedded network, the ratio between switch and nodes is about 1 to 3 as a general rule.

So in effect, fewer routes are available in an off-chip setup, which makes contention a more difficult issue. It also rules out the usage of some interesting routing algorithms, e.g., all variations of XY-routing.

Another fundamental point is that most NoCs are architected around the concept of flits, which defines the atomic unit of data transported across the network. A flit can usually carry 16, 32 or 64 bits of data. Of course, this has a huge impact on how the whole NoC stack is defined, i.e., it orients the choices and definition of the flow control on the link (if used), the error detection and recovery mechanism (if used), the routing and the amount of buffering needed in switches. For off-chip case, we do not have that luxury: the atomic unit of transport is a packet and its size is few hundred byte (e.g., in UniPro) to few Kbytes (e.g., in USB). There are 2 main reasons for this:

- The bigger is a packet, the more efficient usage of the off-chip PHYs is achieved
- The overhead of the header and footer is smaller compared to the payload for larger packets.

Essentially, it means that Data Link Layer defined for NoC cannot be applied directly for chip-to-chip. To allow efficient reuse of some of these concepts, they need to be applied in different ways.

Quality of Service is another key issue. A very elegant and efficient way for QoS handling was presented in Æthereal. The main idea is to use TDMA scheme where all switch/router are globally synchronized defining a very precise global notion of time. Then the QoS scheme, which they call guaranteed services (GS), can be achieved by avoiding contention by design for every single flit. A flit can then traverse the network ballistically. This leads to bound latencies, but it also aids in the compositional design of NoCs since different data flows are isolated from each others. Unfortunately, in our case when we have to encompass also chip-to-chip, this type of approach cannot be applied directly, as the main different lies in the fact that the global notion of time cannot be made as precise as in NoCs.

Another approach presented in (Evain, 2006) seems to be a better fit, since it is more inline with the inherent GALS nature of chip-to-chip interconnect. Nonetheless, more work is required to see how the proposed solution would interact with the necessary error correction mechanisms of chip-to-chip case. Also the cost impact must be analyzed since we have packets of few 100 bytes, compared to flits in NoCs.

Also there are other alternatives proposed in the context of NoC, one of them is discussed in (Marescaux, 2007). Much work has been done to find method, design flows to map and create a specific NoC optimized for certain applications or use cases, e.g., by (Goossens 2, 2005). For the chip-to-chip case, we do not have the luxury to dimension the network for a precise given usage. Different ICs are taken from the shelf and only the physical arrangement of ICs leave a bit of latitude to optimize the network. These show that the approach for chip-to-chip QoS must be different from approaches, known by the authors, used in NoC and must support more runtime mechanism for QoS.

After covering NoCs, we can look the existing chip-to-chip embedded networks. Most chip-to-chip interconnects that have QoS support are host centric. This simplifies QoS provision, but it is also clear that the host centric nature of these solutions is very restrictive and does not give much opportunity for power optimization, compared to the solutions without host centralization.

What became clear from our study is that the target solution must be a compromise between existing NoCs and chip-to-chip interconnect solutions. Nonetheless, it's very interesting to see that this hybrid interconnect do share many common design goals, requirements with NoCs presented in (Benini, 2002).

INTERCONNECT ARCHITECTURE FOR MOBILE DEVICES

Sketch of the Architecture

Based on the discussion started in previous chapter, we can sketch the dream architecture for a mobile device. Since the smallest amount of pins can be achieved by using embedded clock differential serial links, we assume it as a granted feature of our ideal PHY. This gives the best J/bit when transmitting data and assuming that the link is switched-off when not in use. Other advantages of serial links are the fact that with proper line coding and clocking strategy, a single link can go up to 5Gbit/s even in a very noisy environment as a mobile device, including EMI/EMC, power noise due to DC to DC converters, digital noise with high speed logic. This is made even worst due to the lack of proper grounding in battery powered devices. With proper coding, it wields the possibility for optical extension as well. As a result, it will solve problem of EMC/EMI and simplify mechanics because of the smaller minimal bend radius of optical fibers compared to copper, micro-coax or flex cables.

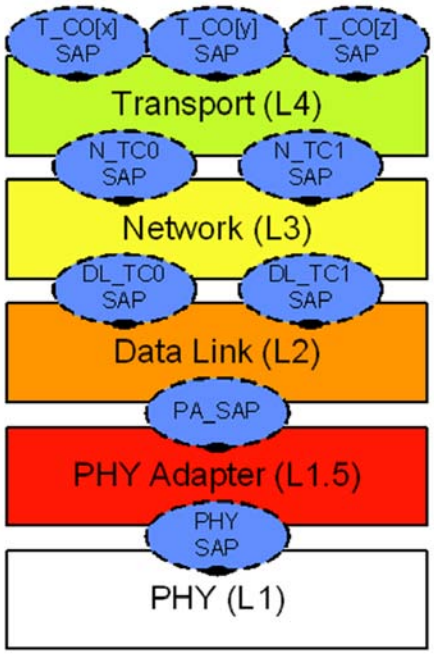
The main problem is to design PHY that can provide low power consumption for a wide range of bandwidth, having scalable implementation, so that when high speed is not needed it should allow scaling down the same PHY, while keeping this PHY interoperable with full fledged high speed PHYs. A great work in this direction is in progress in the PHY WorkGroup of the MIPI alliance.

Such PHY is ideal for chip-to-chip, but not optimal for die-to-die connections. Our goal is to define a complete solution, which can encompass both chip-to-chip and die-to-die. The reason for this is simple as at the same time it will enable to take a risk in introducing new technologies by adding new ICs or chips by using the chip-to-chip PHY; when the technology is proven to be efficient, it will also give the opportunity to bring it easily to wider usage, e.g., by dies stacking or direct on-chip integration on the same die.

It leads to the necessity of having an embedded network solution able to use very efficiently embedded clock differential serial links, but also efficiently support other type of PHYs, i.e., a die-to-die interconnect. When looking at the core concept of most NoCs, the flit, it is clear that NoC technologies cannot be applied easily to a chip-to-chip environment using this type of chip-to-chip PHYs. On the contrary, packet oriented embedded network solutions can use much more efficiently this type of PHYs. Since the costs and power consumption are much higher for off chip solutions in comparison to inside a chip or between dies solutions, it is advantageous to optimize the embedded network for inter-chip usage. The protocol stack of such embedded network is shown in Figure 5, where a layer L1.5 was introduced to simplify dealing with various PHYs.

Error detection and error recovery are two very important aspects. In the noisy environment found

Figure 5. PHY independent generic embedded network stack



in mobile devices, a PHY will experience a non-negligible bit error rate, which cannot be then ignored. This leads to a set of constraints how the ideal solution can be build in protocol terms.

Finally, we cannot forget power consumption. It is well known fact that battery capacity is increasing slower than the internal energy demand. The energy density achieved by current batteries seems to reach a practical limit and no major breakthrough seems to be expected. It is then essential to make all what is possible to save energy by any means. One of the directions is to go away from the host centric architecture when all communications have to go via a central host. Probably the best known example of this is USB. Again, embedded networks have natively a more peer-to-peer approach, which gives the possibility to have a more distributed, communication centric solution. And of course, we also should carefully address the downsides of non-host centric embedded networks. For example, QoS support is a strong requirement and it is more challenging to achieve than in a host centric system.

Usage Scenarios

To concretely show the opportunities offered by the selected approach, let us discuss few real life examples: the case of camera and display in the context of HD multimedia content; then the case of mass memory storage; and finally the case of software debugging and tracing.

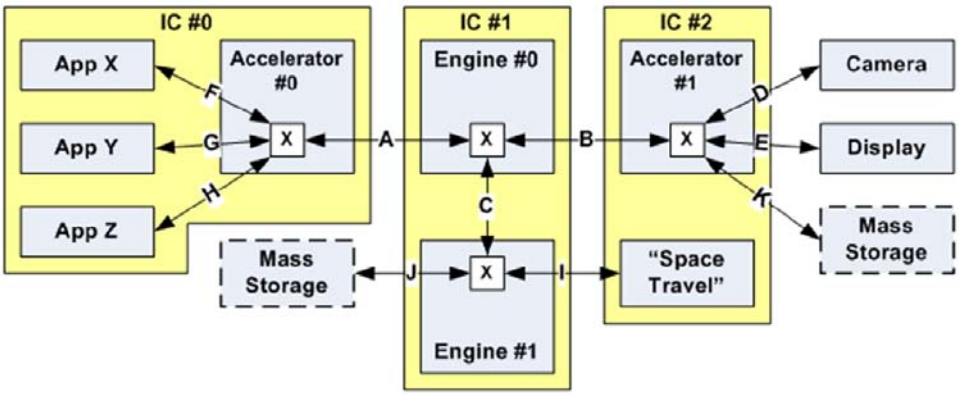
With the fast development of media and internet convergence, delivery of high quality multimedia content to mobile user has become more important than ever. Recently HD content took off and is getting more and more relevant every passing days. To support HD content on mobile devices, there are quite some technical issues to be addressed, i.e., the computing power needed to support HD content is high, e.g., when driving TV; the power consumption induced by such a use case is also very high. In these conditions, the proposed architecture allows many possible optimizations:

1. If the main CPU of the mobile device is not sufficient, an external dedicated IC can be used for handling HD multimedia content and serving a display and a camera.
2. If the main CPU has enough computing power and can handle HD content at low power, the same display and camera could be directly attached to the main IC.
3. If the main CPU is powerful enough to render HD content on the local display, but not sufficient to drive a HD TV set, the combination of the above stated solutions is possible.

Here, we make an assumption that displays and cameras would be directly interfaced to the device embedded network. The first scenario is interesting even if the main CPU does have enough computing power, but the dedicated IC could implement some novel solution, e.g., as presented in (Ahonen, 2004) which would lead to reduction of the overall power consumption. The first scenario is presented in Figure 6, where the physical chip IC #2 handles HD content, chip IC #1 is the main chip containing two CPUs, while IC #3 is an accelerator chip handling three different applications. The mass storage is drawn as a dashed box to show that we have two options where to attach it.

Since the HD content must be store somewhere, the case of mass memory storage is also interesting to address. We have basically three possible scenarios on how to handle HD multimedia, the usage of the mass storage must be near optimal for the 3 scenarios presented above. Here again, we take the advantage of the embedded network and have the mass storage using it. The mass storage can be attached to the multimedia IC handling HD content if that is where the highest throughput required from the mass storage. Alternatively, it can be attached directly to the main CPU. Better yet, the configuration can be changed

Figure 6. One possible physical configuration of our futuristic mobile device



without any software impact from one product to another depending on the targeted market or usage: this allows either power or performance optimization according to the main use case of a product.

Earlier we talked about software integration, tracing and software debugging. We can reuse the network to transport debug and trace information, which reduce the cost and simplify the infrastructure needed for this purpose. If QoS is provided on such non-centric embedded network, we can avoid the Heisenberg uncertainty principle that states that if we try to observe or debug a system, we will change its behavior making the debugging and tracing pointless. QoS enables us to isolate the different flows, and thus to limit and control the level of interference between the actual behavior to be observed and the debug/trace information.

QoS vs Resource Management

In the context of NoC, QoS have been well studied (Goossens, 2005; Evain, 2006; Marescaux, 2007), but unfortunately many of those findings are not directly applicable to our domain comprising of chip-to-chip and die-to-die connections. Even if the QoS issues would be solved, there is still a resource management problem that has to be carefully addressed. As an example, let us consider a flash memory. Standard QoS mechanism will guarantee a given throughput in the network between an application and the flash memory. But if the throughput of the flash memory is smaller than the throughput provided by the network, QoS will help only partially in providing deterministic behavior for applications using that shared resource. To truly solve the software and system level integration, the standard QoS definition must be extended to support mechanisms to also enable guarantees in the usage of shared resources, not usage limited only to resources within an embedded network. Extending QoS in this way would allow the possibility to do runtime optimizations that provide, for example:

- Better quality for one application to compared to another in using any shared resource in the network.
- Better control and runtime optimization of the power consumption, when taking power as one of the shared resources.

Indeed, we believe that this is a bigger issue than in NoC cases, because NoC can be readily optimized for a given purpose. A chip-to-chip embedded network cannot be optimized for one given purpose, but each node in the network must be generic enough to support large number of potential use cases taking into account that all use cases are not known at design time.

CONCLUSION

The results of our studies became the basement of a new industrial standard - UniPro, which is currently developed within the scope of MIPI standardization alliance. This work is not finished yet, so more technical details are still to come and updates are coming on a daily basis.

Essentially, fast broad deployment of embedded networks is seen by many as a question of survival for the mobile device integrator companies and the whole industry. This observation can be seen as a direct consequence of the following factors:

- Business value chain is changing to a vertical model.
- Time-to-market is ever more critical.
- Device complexity has reached a level at which it has become very expensive to maintain

Added to the actual turmoil in the mobile industry, the place of driver of the next dominant architecture is still to be taken. In the mobile industry, some sees also the arrival of embedded network solutions as important as the first introduction of a standard extension bus in the PC industry. It is clear that a standard interface has made a dramatic positive change in the PC industry. The proposed solution for embedded networks, based on MIPI UniPro, has the potential of reshaping the mobile industry.

MIPI UniPro 1.0 was released in January 2008, which was a very important step as it was the first standard available for adaptation by the industry leading towards a full embedded network solution. As for the future plans, currently we are mostly focused on finalizing the standard for the MIPI UniPro 1.1 embedded networks solution, which is an evolution of version 1.0. At the same time, many people already start speaking about on-chip extension of the UniPro standard. Other hot topics, which are equally relevant for on- and off-chip embedded networks are:

- A full-scale Quality of Service solution with strict and soft guaranties.
- Addressing of security aspect for the new standard, analysis of potential threads and solutions to prevent compromising of the overall security of the network.
- Network reliability and especially optimization of buffer lengths and other related cost factors.
- Efficient power management of the network devices and the network infrastructure as a whole.

These topics have already been explored for future release of the MIPI UniPro specification.

As a final remark, we would like to say that our team has been very fortunate to have a chance to see the start of this idea and corresponding research based on a vision of what the future should be and it was a great honor for us to drive it up to standardization.

REFERENCES

Adriahantenaina, A., Charlery, H., Greiner, A., Morteiz, L., & Zeferino, C. A. (2003). SPIN: A scalable, packet switched, on-chip micro-network. In *Proceedings of Design, Automation and Test in Europe: Designers'. Forum*, *2*, 70–73.

Ahonen, T., Virtanen, S., Kylliäinen, J., Truscan, D., Kasanko, T., Siguenza-Tortosa, D., et al. (2004). A brunch from the coffee table-case study in NoC platform design. In J. Nurmi, H. Tenhunen, J. Isoaho, & A. Jantsch (Eds.), *Interconnect-centric design for advanced SoC and NoC* (pp. 425-453). Boston: Kluwer Academic Publishers.

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

Dall'Osso, M., Biccari, G., Giovannini, L., Bertozzi, D., & Benini, L. (2003). Xpipes: A latency insensitive parameterized network-on-chip architecture for multi-processor SoCs. In *Proceedings of the 21st International Conference on Computer Design* (pp. 536-539).

ESA. (2009). *Spacewire standard working group*. Retrieved April 2, 2009, from http://spacewire.esa.int/content/Home/HomeIntro.php

Evain, S., Diguet, J.-Ph., & Houzet, D. (2006). NoC design flow for TDMA and QoS management in a GALS context. *EURASIP Journal on Embedded Systems, 2006*(63656), 12.

Goossens, K., Dielissen, J., Gangwal, O. P., Pestana, S. G., Radulescu, A., & Rijpkema, E. (2005). A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC Design and verification. In *Proceedings of the conference on Design, Automation and Test in Europe* (Vol. 2, pp. 1182-1187).

Goossens, K., Dielissen, J., & Radulescu, A. (2005). Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design & Test of Computers*, *22*(5), 414–421. doi:10.1109/MDT.2005.99

IEEE. (1990). *ISA IEEE personal computer bus standard P996, draft D2.02*. Retrieved April 2, 2009, from http://standards.ieee.org/

Jantsch, A. (2006). Models of computation for networks on chip. In *Proceedings of Sixth International Conference on Application of Concurrency to System Design* (pp. 165-178).

LaCie. (2009). *FireWire® 800 – technology brief*. Retrieved April 2, 2009, from http://www.lacie.com/download/more/WhitePaper_FireWire_800.pdf

Lappeteläinen, A., Tuopola, J.-M., Palin, A., & Eriksson, T. (2008). *Networked systems, services and information, the ultimate digital convergence*. Paper presented at the 1st International NoTA conference, Helsinki, Finland.

Marescaux, T., & Corporaal, H. (2007). Introducing the SuperGT network-on-chip: SuperGT QoS: More than just GT. In *Proceedings of the 44th annual conference on Design automation* (pp. 116-121).

Millberg, M., Nilsson, E., Thid, R., Kumar, S., & Jantsch, A. (2004). The Nostrum backbone-a communication protocol stack for networks on chip. In *Proceedings of 17th International Conference on VLSI Design* (pp. 693-696).

MIPI. (2009). *Mobile industry processor interface alliance*. Retrieved April 2, 2009, from http://www.mipi.org/

Moadeli, M., Vanderbauwhede, W., Shahrabi, A., & Ould-Khaoua, M. (2008). Modeling differentiated services-based QoS in wormhole-routed NoCs. In *Proceedings of 22nd International Conference on Advanced Information Networking and Applications* (pp. 942-949).

Neuenhahn, M. C., Blume, H., & Noll, T. G. (2008). Quantitative design space exploration of routing-switches for network-on-chip. *Advances in Radio Science: an Open Access Journal of the U.R.S.I., 6*, 145-150.

NoTA. (2009). *Network on terminal architecture, NoTA world open architecture initiative*. Retrieved April 2, 2009, from http://www.notaworld.org/

PCI-SIG. (2008). *PCI specifications documents, peripheral component interconnect (PCI) standard*. Retrieved April 2, 2009, from http://members.datafast.net.au/dft0802/specs.htm and http://www.pcisig.com/specifications/ordering_information

PCI-SIG. (2009). *PCI express - computer expansion card interface*. Retrieved April 2, 2009, from http://www.pcisig.com/members/downloads/specifications/pciexpress/ PCI_Express_Base_Rev_2.0_20Dec06_cb2.pdf

Philips Semiconductors. (2000). *I2C-bus specification version 2.1*. Retrieved April 2, 2009, from http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

SATA-IO. (2009). *Serial ATA-IO: Enabling the future*. Retrieved April 2, 2009, from http://www.serialata.org

Schwerdtfeger, M. (2000). *SPI - Serial Peripheral Interface*. Retrieved April 2, 2009, from http://www.mct.net/faq/spi.html

USB. (2009). *USB implementers forum, Universal Serial Bus (USB) 2.0 specification*. Retrieved April 2, 2009, from http://www.usb.org/developers/docs/

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 1, edited by Seppo Virtanen, pp. 22-36, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 9
Implementation of FFT on General-Purpose Architectures for FPGA

Fabio Garzia
Tampere University of Technology, Finland

Roberto Airoldi
Tampere University of Technology, Finland

Jari Nurmi
Tampere University of Technology, Finland

ABSTRACT

This paper describes two general-purpose architectures targeted to Field Programmable Gate Array (FPGA) implementation. The first architecture is based on the coupling of a coarse-grain reconfigurable array with a general-purpose processor core. The second architecture is a homogeneous multi-processor system-on-chip (MP-SoC). Both architectures have been mapped onto two different Altera FPGA devices, a StratixII and a StratixIV. Although mapping onto the StratixIV results in higher operating frequencies, the capabilities of the device are not fully exploited. The implementation of a FFT on the two platforms shows a considerable speed-up in comparison with a single-processor reference architecture. The speed-up is higher in the reconfigurable solution but the MP-SoC provides an easier programming interface that is completely based on C language. The authors' approach proves that implementing a programmable architecture on FPGA and then programming it using a high-level software language is a viable alternative to designing a dedicated hardware block with a hardware description language (HDL) and mapping it on FPGA.

DOI: 10.4018/978-1-4666-0912-9.ch009

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

INTRODUCTION

During recent years the popularity of FPGAs has grown increasingly because of the reduced cost and development time of FPGA-based solutions. It is well known that the FPGA implementation is more convenient for smaller market volumes and this is generally the case of embedded systems (Sangiovanni-Vincentelli, 1993). In addition, the boundary line that defines when it is better to use an Application-Specific Integrated Circuit (ASIC) is moving toward larger volumes with the advancement of silicon technologies, allowing the exploration of FPGA-based architecture for different application fields.

However, a FPGA-based approach still has some drawbacks: a larger area, less performance, and higher power consumption when compared to an ASIC approach (Kuon & Rose, 2006). Even though this trend is also changing in favor of FPGAs, they are roughly one order of magnitude slower than standard cell ASIC implementations and two orders of magnitude compared with full custom approaches (Chinnery & Keutzer, 2000).

FPGA implementation is usually associated with hardware design. A digital circuit is designed following a RTL flow but its synthesis is based on FPGA resources instead of standard cell library components as in the case of ASIC implementation. The main advantage is that the hardware synthesis is faster and the manufacturing costs follow the trend that we mentioned above. However, it is possible to implement a general-purpose system on a FPGA as well, e.g. a single-processor system. In this case we can also reduce the cost of the initial design, which is implemented programming the processor in software rather than creating a dedicated hardware architecture using a HDL. Processor-based architectures can be programmed using C language or similar programming languages, which are more popular than HDLs. This approach has a drawback: a digital system based on a processor is usually slower than a system based on dedicated hardware. However, in many situations the flexibility of a general-purpose system is an unavoidable requirement.

This paper proposes two solutions based on general-purpose hardware for FPGAs. In these two solutions, we try to overcome the performance degradation due to the choice of FPGA implementation and the adoption of a general-purpose approach. The evaluation of these two solutions is based on the implementation of an N-point FFT algorithm. The FFT is used in various multimedia applications, for example image processing, audio processing, and video compression. In addition, Software Defined Radio (SDR) applications use the FFT for OFDM-based wireless protocols such as IEEE802.11a/g and WiMAX. The development of SDRs pushes towards novel architectural solutions because it needs flexibility and high performance at the same time.

The most common way to map an FFT on a general-purpose system is to use a software implementation on a digital signal processor (DSP). It is not a case that this approach was already under study in the 90s (Meyer & Schwarz, 1990). A DSP provides a general-purpose architecture optimized for signal-processing algorithms.

An alternative that is very popular nowadays is the use of coarse-grain reconfigurable hardware. The programmability of these devices is comparable with CPUs and DSPs, even though they require a better knowledge of the architecture to achieve the best results. These devices are based on arrays of homogeneous processing elements (PE). These PEs can be as simple as an ALU or as complex as a processor. The interconnections can be fixed or based on a network approach. Typically it is possible to define at run-time the functionality of each single PE as well as the interconnections between them.

The FFT has been used as a case study in the proposal of such solutions. One of the most successful devices is Morphosys, Kamalizad, Pan, and Bagherzadeh's (2003) showing that the mapping of the FFT on their Morphosys2 was the most efficient. Very good results were achieved by XPP

(Baumgarte et al., 2003) and ADRES (Mei, Vernalde, Verkest, Man, & Lauwereins, 2003). Also Montium and the Chameleon system, the latter developed mainly for SDR applications, provided an optimized implementation of FFT (Zhang, Kokkeler, & Smit, 2007). Moreover, Montium and XPP were successfully employed to realize non-power-of-two FFTs (Rivaton, Quevremont, Zhang, Wolkotte, & Smit, 2005).

The multi-processor system-on-chips are an alternative to coarse-grain acceleration for the implementation of the FFT. Bahn, Yang, and Bagherzadeh (2008) analyzed the parallelization of the FFT algorithm on 2x2, 4x4, and 8x8 meshes of processors. Kim, Kim, and Sobelman (2005) focused on a star topology Network-on-Chip (NoC) architecture designed for the CDMA algorithm.

Another common way to implement the FFT is designing a dedicated architecture. This has been done lately and in particular for wireless baseband engines. Kuo, Wen, and Wu (2003) proposed a programmable 64-2048 FFT processor to cover different wireless standards. Camarda, Prevotet, and Nouvel (2009) designed a FFT processor targeted to Digital Video Broadcasting (DVB). Ojail, David, Chevobbe, and Demigny (2009) focused on low-power architectures for SDR.

All these engines are designed for ASIC implementation. However, there is hardware FFT engines created for FPGAs which are provided by FPGA vendors for a fee. For instance, Altera (2009a) has its own FFT Megacore Function. This kind of engine is area optimized but their final performance could depend on the design in which they are instantiated.

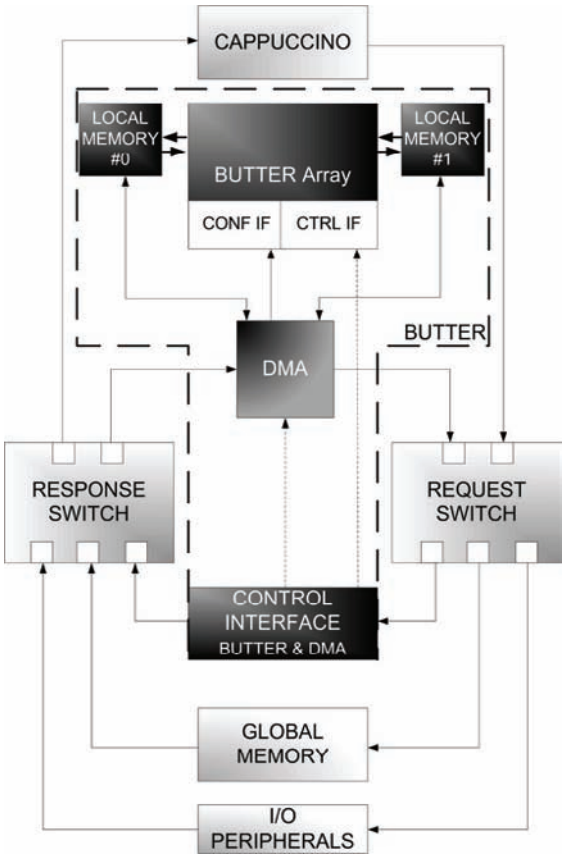
In this work, we propose two general-purpose architectures for FPGA. The first is a coarse-grain reconfigurable architecture. Unlike the solutions described above, we have dealt with the resource availability of a FPGA device although our implementation is not comparable with ASIC based solutions in terms of speed. The second is a multi-processor solution.

The work is organized as follows. In the next sections we analyze in detail the proposed architectures. Then we provide a description of the FFT algorithm and the implementation on our solutions. Finally we show the results and draw some conclusions.

THE RECONFIGURABLE PLATFORM

The reconfigurable platform (Figure 1) is based on the coupling of a general-purpose RISC processor called CAPPUCCINO and a run-time reconfigurable coarse-grain array called BUTTER (Brunelli, Garzia, & Nurmi, 2008b). These two processing devices communicate with memories and additional peripherals using a non-blocking network

Figure 1. Reconfigurable platform



of switches which is composed of a request and a response switch.

CAPPUCCINO Processor

CAPPUCCINO (Figure 2) is a RISC processor with floating-point capabilities. It is the product of merging two separate components: COFFEE RISC core processor and MILK co-processor.

COFFEE (Kylliäinen, Ahonen, & Nurmi, 2007) is a RISC core with a 6-stage pipeline and an embedded integer multiplier. Because of the open-source nature of the COFFEE RISC project, the VHDL model of the processor is available for free as well as its C compiler.

MILK co-processor (Brunelli et al., 2008a) was developed inside the same project with the purpose of providing floating-point acceleration to COFFEE. Subsequently, the floating-point arithmetic blocks composing Milk co-processor were integrated in the pipeline of COFFEE processor creating CAPPUCCINO.

CAPPUCCINO has a separate interface for instruction and data memories. The instruction memory interface is directly connected with a 64Kbytes scratchpad memory. The data memory interface is connected through the switched interconnection network to the data memory and I/O peripherals. The data memory is a 256Kbytes scratchpad memory.

BUTTER Co-Processor

The core of the BUTTER co-processor is an array of 4x8 processing elements (PEs) depicted in Figure 3. Each PE receives two 32-bit inputs and generates two 32-bit outputs. The first output is the result of one of the following operations: addition, subtraction, multiplication, shifting, or a logical operation implemented through a 4-word LUT. The arithmetic operations support integer and floating-point values. In addition, sub-word computations are also possible. This means that a 32-bit operand can be processed as two distinct 16-bit operands or four distinct 8-bit operands.

All of the operations listed above provide the possibility to use a constant value stored in a dedicated register (the "immediate" register) as a second operand.

The PEs are interconnected using local, interleaved, and global wires. The local wires connect one PE with its upper, upper-left, upper-right, and left neighbors. The interleaved connections run between two non-adjacent PEs placed in the same column. The global connections provide one operand to all the PE of the same row (global horizontal) or column (global vertical).

Figure 2. CAPPUCCINO processor with instruction memory

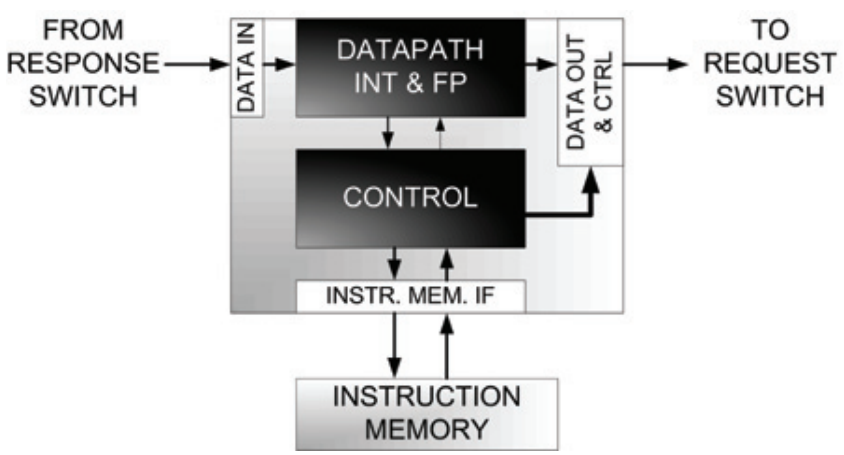
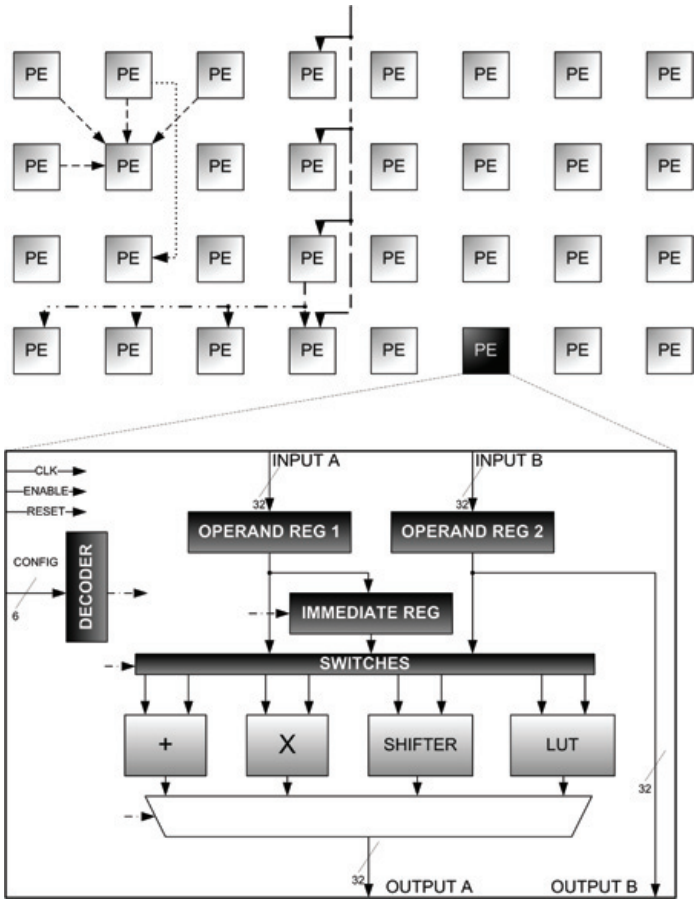


Figure 3. BUTTER Co-processor with the possible interconnections between PEs (top) and the PE architecture (bottom)



The array described above is completely configurable at run-time. The user can decide which operations he wants to perform in each PE and how to interconnect the PEs. This is achieved by sending a 16-bit configuration word to each PE. This configuration word defines which functional unit should be employed and where the input operands come from. A small memory is used inside the PE to hold the configuration words. This memory can host up to four different words and the switching between two contexts already stored in the memory requires just one clock cycle. Loading of a new context is achieved using a pipelined mechanism described in a previous publication (Garzia, Brunelli, & Nurmi, 2008a).

The array is directly connected with two local memories which are able to feed the array with sixteen 32-bit operands and to store sixteen 32-bit results per clock cycle. The local memories are controlled using a dedicated DMA that takes care of the transfers to and from the main memory. Since data to be processed with BUTTER must be first transferred to the local memories, we provide a ping-pong mechanism that avoids the need of further transfers when the input data are the results of BUTTER´s previous processing (Garzia, Brunelli, Rossi, & Nurmi, 2008b).

The two local memories are dual-ported. This characteristic enables simultaneous access of BUTTER and DMA to the memories, allowing

hiding of the transfer time in the processing time of BUTTER.

The Switched Interconnection Network

The communication between the two processing devices (CAPPUCCINO and BUTTER) and the system peripherals (memory and I/O) is based on programmable switches designed to replace a bus infrastructure in order to support parallelized memory-mapped communication (Ahonen & Nurmi, 2006).

Transactions are managed using a TDMA mechanism. Simultaneous requests to the same peripheral, result in the assignment of the current time slot to one of the requesters, chosen according to an arbitration mechanism based on input queue priorities programmable through a control register interface. Transactions are strictly pipelined to keep a low-priority request from blocking a higher-priority one because of an incomplete memory access.

Programming Model

We adopted a model-of-computation in which the processor runs the main functions and controls the accelerator, while the accelerator is only employed for computationally heavy kernels.

The implementation of an application onto the platform requires that the developer first specifies manually which kernels should be mapped onto the accelerator and then performs the actual mapping using a graphical user interface (GUI) designed by us. The GUI produces the configuration files for the array, which can be handled using dedicated C libraries that we provided. In this way the runtime task allocation and control of BUTTER and DMA is always performed using C code running on the RISC processor.

Implementation Details

The platform described above has been implemented on FPGA. Some design features were driven by this implementation choice. For example, the use of dual-port memories is justified by the fact that most of the embedded memory blocks in FPGA chips are dual-ported. The I/O peripherals inserted in our platform can be found in any FPGA development board and support different kind of multimedia applications.

We decided to evaluate our platform on two different FPGA devices from different technologies, one from the Altera StratixII family and one from the Altera StratixIV. The Altera StratixII is based on a 1.2V, 90-nm, all-layer copper SRAM process, while the StratixIV is realized with the TSMC 40-nm technology. In both cases we chose the most capable device on the market. In the first case, the Altera StratixII EP2S180F is characterized by 71,760 programmable logic blocks called Adaptive Logic Modules (ALM). Each ALM contains combinational logic implemented as two adaptive look-up tables (ALUT), several multiplexers, and two registers. In addition, the device provides 384 DSP blocks to implement arithmetic operations and on-chip memory that amounts to 9,383,040 bits (Altera, 2007). In the second case, the Altera StratixIV EP4SGX530 is a device containing 212,480 ALMs, 27,376 kbits of on-chip memory, and 1,204 18-bit embedded DSP elements (Altera, 2009b). The StratixIV is roughly four times bigger than the StratixII and contains three times more memory. What we expect from the comparison is a design that utilizes the same amount of logic resources but runs at a higher frequency, because of the adoption of a new silicon technology. For the synthesis flow we used Altera QuartusII 9.0 SP1 version.

The reconfigurable platform was implemented first in the StratixII device. Details can be found in Table 1. The numbers in the table refer to the ALMs and not to the ALUTs or registers. These numbers give a more precise indication of the

Table 1. Synthesis of the reconfigurable platform on an Altera StratixII device (EP2S180F)

Entity	Resource Utilization				Max. Frequency (MHz)	
	ALM	%	Memory (KB)	DSP Blocks	Slow model	Fast model
Platform	40,046	55.8%	7,162,589	280	-	-
CAPPUCCINO and Instr. Mem.	11,391	15.8%	525,250	24	68	115
BUTTER Array	26,581	37%	0	256	27	48
Butter's Local Mem. and DMA	1,066	1.5%	526,107	0	106	181
System Memory	63	<0.1%	2,097,152	0	"	"
Interconnection Network and Ifs	276	0.4%	0	0	"	"
I/O Peropherals	669	0.9%	4,014,080	0	"	"

percentage of utilization, because they also take into account ALMs partially used, i.e., in which only one of the two ALUTs or one of the two registers are utilized.

The reconfigurable array occupies almost three times the resources of the RISC processor. The difference is mainly due to the complexity of BUTTER´s interconnections. In addition, the floating-point calculation requires a large amount of logical resources and significantly increases the critical path inside the device. This happens because all of the PE operations are performed in one clock cycle in order to ease the application mapping. This means that the clock period of BUTTER is affected by the longest latency operation, i.e., the floating-point multiplication.

We decided to evaluate the use of separate clock domains to reduce the impact of BUTTER's slower frequency on the global implementation. Using different clocks, we noticed a significant difference in the maximum operating frequency between CAPPUCCINO and BUTTER. While the processor can run at 68MHz, the array cannot exceed 27MHz. This suggests that the use of separate clock domains is efficient in optimizing the operating frequency of the system. In order to reduce the complexity, we thought of using a derived clock. If BUTTER runs at 25MHz, CAPPUCCINO and the rest of the system can run at 50MHz without requiring particular synchronization logic.

It is possible to further optimize the platform. From Table 1 it can be seen that the memory subsystem is allowed to run at 106MHz. Therefore, we can modify the DMA in order to perform the transfers at double frequency or in other words, 100MHz. The FFT implementation was carried out using these assumptions.

Table 2 shows the synthesis results of the same platform on the StratixIV. As expected, the resource utilization is the same in terms of ALMs. On the other hand, the device is bigger and it is possible to implement a much larger amount of logic. This suggests that the platform should be modified in order to more completely exploit the device resources.

Another significant difference is in the operating frequency. It is true that for the logic the frequency increases 40% in average, but for the memories the situation is different. The frequency actually decreases. These figures are preliminary and thus should be accepted with caution. In addition, they refer to engineering samples which showed significant issues on the memories. However, we believe that the frequency drops is due to the different memory architecture. The memory blocks of the StratixIV are characterized

Table 2. Synthesis of the reconfigurable platform on an Altera StratixIV device (EP4SGX530)

Entity	Resource Utilization				Max. Frequency (MHz)	
	ALM	%	Memory Bits	18-bit DSP Blocks	*Slow model (0°C)*	*Fast model (0°C)*
Platform	38,267	18%	7,161,600	276	-	-
CAPPUCCINO and Instr. Mem.	11,391	4.8%	524,288	20	97	156
BUTTER Array	24,421	11.5%	0	256	39	64
Butter's Local Mem. and DMA	1,513	0.7%	524,288	0	96	126
System Memory	21	<0.1%	2,097,152	0	"	"
Interconnection Network and IFs	254	0.1%	0	0	"	"
I/O Peripherals	676	0.3%	4,015,872	0	"	"

by a smaller grain size. The biggest block is 16Kx8, while in the StratixII it was 64Kx8. They are also better distributed inside the chip. This suggests that our monolithic approach for the memory architecture is not adequate, making the mapping onto the FPGA device more difficult that results in a lower operating frequency.

The conclusion is that the platform has to be modified to take advantage of the StratixIV architecture. However, the developed FFT implementation is only valid if we adopt the same ratio between operating frequencies that we used for the StratixII. Since the maximum operating frequency of the memory is still around 100MHz, we have to use exactly the same frequencies as in the StratixII, i.e., BUTTER runs at 25MHz, CAPPUCCINO at 50MHz, and DMA and memories at 100MHz.

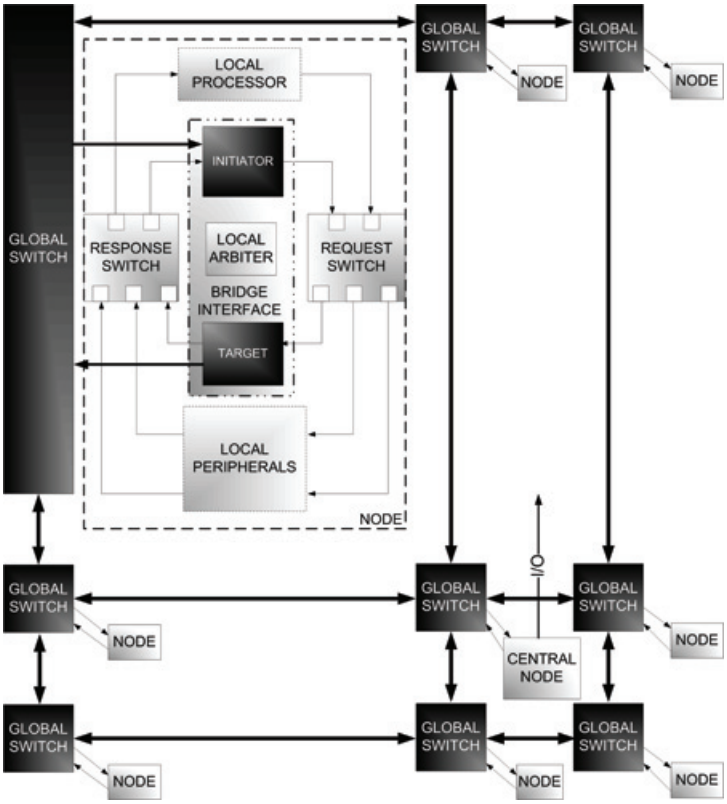
The MP-SoC Platform

The MP-SoC platform is based on the Silicon Café template developed at Tampere University of Technology (Nurmi, 2008). The template can support the design of different kinds of multi-processor architectures composed of either homogeneous or heterogeneous nodes. The number of nodes and the PE inside a single node are not defined by the template. In our instantiation, Silicon Café is composed by 9 homogeneous nodes arranged in a mesh topology, where each node hosts a COFFEE RISC core or a CAPPUCCINO RISC core. The communication infrastructure between the nodes as well as inside any single node is defined by the template and is based on a hierarchical Network-on-Chip (NoC).

The Hierarchical Network-on-Chip

The NoC (Figure 4) can be viewed as the composition of two levels of communication, local and global. At the local level a non-blocking switch infrastructure (the same used in the reconfigurable platform) is responsible for the communication inside the node. At the global level, a mesh of switches supports the communication between nodes. The two communication levels (local and global) are interfaced together through a bridge. It has an initiator side that acts as a master inside a node and a target side that acts as a slave. The initiator is responsible for the routing of the data coming from the global NoC to the peripherals inside the node. The target takes care of the routing of local data to remote peripherals. The routing follows a wormhole approach and it is defined into a look-up table accessible by the target. Each node can route up to 16 different possible paths. Moreover, the delivery of packets can be also done through multicast or broadcast (Airoldi, Garzia, Ahonen, Milojevic, & Nurmi, 2009a).

Figure 4. MP-SoC Platform



Memory Model

The system is based on a distributed memory where each node has its own memories. There are two separate 32-bit scratchpad memories, one for instructions and one for data. The size is fixed considering the available on-chip memory as described in the Implementation section.

Each node is able to perform remote writes onto another node, while remote reads are not supported at the moment. Whenever a node needs to read data from another node it has to write a read request onto the "*mailbox*" of the remote node letting it know the return address. The remote node will then provide the required data at the memory address found in its mailbox as soon as it is computed. In addition, the PEs are cache-less, hence the data coherence is maintained at application level without any hardware support. In fact the data coherence is managed via software through synchronization messages, typically at the end of each subtask. The synchronization messages are responsible to start data exchange between different nodes and to guarantee the synchronization between parallel executions of sequential tasks. In the case where data are not yet available, the PE will stall until the data is available.

Programming Model

To implement an application onto the MP-SoC, the different functionality of the nodes needs to be considered. In particular, the central node is the global controller. It takes care of task partitioning and mapping. It is the only one connected to external peripherals and it takes care of the distribution of the data to the other eight nodes. On the other hand, the eight nodes are used for

the actual processing. They all act together as a parallel accelerator.

All the nodes are comprised of either a COFFEE or a CAPPUCCINO processor; therefore they can be programmed using C. In this early stage of development we do not yet have an automatic partitioning tool. The application developer must decide which parts of the application should be mapped on the processing nodes and how tasks are distributed. It is then sufficient to compile each portion mapped onto a processing node separately and load the binary into the instruction memory of each node.

IMPLEMENTATION DETAILS

The platform described above was implemented in two versions, one that supports only integer operations and one that supports also floating-point arithmetic. The first version is called *Ninesilica* and is characterized by the use of a COFFEE RISC core as the processing core in all the nodes. In the second version called *Ninesilica-FP* we replaced COFFEE with CAPPUCCINO. The difference between the two choices is significant because CAPPUCCINO is twice as big as COFFEE in terms of resource utilization. In fact, the first version fits well in the StratixII device, while the second version does not fit. It would need almost 100,000 ALMs, while only 71,000 are available. The second version has thus been implemented only on the StratixIV.

Besides the processor cores, every node is characterized by memories. The central node uses the biggest amount of memory, a 256Kbytes data memory and a 32Kbytes instruction memory. In the central node the instruction memory is not accessible via the communication switches because we do not expect any remote processor to modify its behavior. On the other hand, the central node is connected with different I/O peripherals which use additional memory. Each processing node also contains 32Kbytes data memory and a 16Kbytes instruction memory. The instruction memory is dual-ported. One port is directly connected to the processor core, whereas the other is interfaced to the switches allowing the remote access from the central node (the housekeeper).

Table 3 collects the synthesis results for Ninesilica. Considering a single node, most of the resources are utilized by the processing core. Only 5% of the node area (0.3% of the FPGA device) corresponds to the local network interface. On the other hand, the global network utilizes 3.9% of the chip. It is clear that the network does not introduce any bottleneck, since the critical path is related to the processor. The system uses the embedded DSP blocks of the FPGA to implement the functional units of the processor. The entire platform occupies 60% of the FPGA device. The maximum operating frequency is equal to 83MHz.

Table 3. Synthesis of the MP-SoC Platform on an Altera StratixII device (EP2S180F)

Entity	**Resource Utilization**				**Max. Frequency (MHz)**	
	ALM	**%**	**Memory Bits**	**DSP Blocks**	**Fast model**	**Slow model**
Ninesilica	42,751	59.6%	5,505,024	144	146	83
Central Node	4,663	6.5%	2,359,396	16	“	“
Processing Node	4,474	6.2%	393,216	16	“	“
Global Network Interface	2,836	3.9%	0	0	“	“
COFFEE	4,290	6.0%	0	16	“	“
Local Network Interface	206	0.3%	0	0	“	“

Table 4 shows the results for the implementation of the same platform on a StratixIV device. For the same memory issues we mentioned before, the memory we used is smaller than in the StratixII implementation. In particular, it was not possible to instantiate a single block of 256Kbytes in the central node. However, it would be possible to instantiate such an amount of memory using separate banks.

In this case we do not notice any drops in the operating frequency of the memory. The reason is that in Ninesilica the memory is better distributed and this helps the mapping onto the StratixIV device.

The logical resource utilization is similar, but our platform now occupies only 20% of the chip. This means that we can introduce more complex architectures, e.g., replacing the processor cores with more complex blocks or increasing the number of nodes. We are currently working on a platform based on four instantiations of Ninesilica. From the table we notice that the frequency is also increased (35% for the slow model). However, it is to be noted that these results are also preliminary, depending on the FPGA vendor.

The second version of the platform, Ninesilica-FP, has only been implemented on the StratixIV. The synthesis results are in Table 5. The platform occupies 45% of the device. The size is doubled, because the processor cores in each node are twice as big as before. The memory utilization is the same. We use 4 additional DSP blocks for each processor, because of the floating-point pipeline. The latter also introduces a longer critical path since the maximum operating frequency drops with respect to Ninesilica implementation.

Table 4. Synthesis of the MP-SoC Integer Platform on an Altera StratixIV device (EP4SGX530)

Entity	**Resource Utilization**				**Max. Frequency (MHz)**	
	ALM	%	Memory Bits	DSP Blocks	Fast model	Slow model
Ninesilica	42,350	19.9%	3,670,016	144	184	114
Central Node	4,552	2.1%	524,288	16	“	“
Processing Node	4,423	2%	393,216	16	“	“
Global Network Interface	2,755	12.9%	0	0	“	“
COFFEE	4,249	1.9%	0	16	“	“
Local Network Interface	200	<0.1%	0	0	“	“

Table 5. Synthesis of the MP-SoC floating-point platform on an Altera StratixIV device

Entity	**Resource Utilization**				**Max. Frequency (MHz)**	
	ALM	**%**	**Memory Bits**	**18-bit DSP Elements**	**Fast model**	**Slow model**
Ninesilica-FP	47,910	45%	3,670,016	180	169	103
Central Node	10,316	4.8%	524,288	20	“	“
Processing Node	10,210	4.8%	393,216	20	“	“
Global Network Interface	2,747	12.9%	0	0	“	“
CAPPUCCINO	10,038	4.7%	0	20	“	“
Local Network Interface	200	<0.1%	0	0	“	“

THE REFERENCE PLATFORMS

In order to compare the performance of the two architectures described above, we introduce two single-core reference platforms, one for the integer case, one for the floating-point. The reference platforms are of the architectural form depicted in Figure 5. For the integer case the processor core used is COFFEE whereas for the floating-point case it is CAPPUCCINO.

Since both of the processors can be programmed in C, it is possible to run any C application on the reference platforms. It is also possible to quantify the speed-up obtained using the reconfigurable platform or the MP-SoC platform. The use of the same reference platform allows a fair comparison between the two platforms under study.

Table 6 summarizes the maximum operating frequencies in the reference platforms according to the processor used.

The FFT Algorithm

The FFT algorithm that we mapped onto the two platforms is based on radix-2 decimation in frequency, introduced by Cooley and Tukey (1965). The algorithm was chosen because of its scalability and intrinsic support of parallelization.

The idea behind the radix-2 decimation is that an N-point FFT can be implemented using a dedicated stage plus two N/2-point FFTs. The dedicated stage is composed of N/2 butterfly operations. Iterating this procedure, we can end up with a 2-point FFT that corresponds to a simple butterfly operation between 2 complex values. This means that an N-point FFT can be decomposed in a succession of n stages, where n is equal to the base 2 logarithm of N and each stage processes N complex values using N/2 butterflies.

The main issue for the mapping of a radix-2 algorithm on our platforms is to decide how each stage of the FFT is mapped, i.e., how we implement a single butterfly and how many butterflies

Figure 5. Reference platform

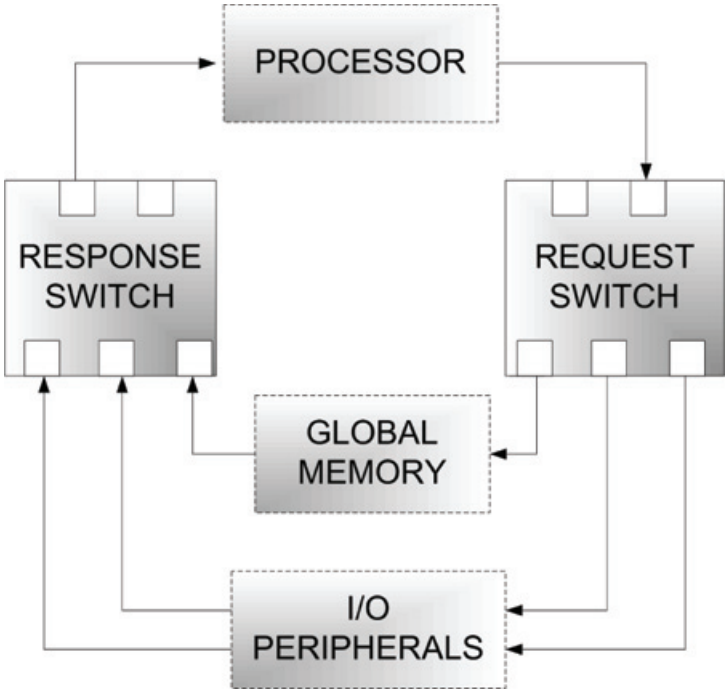


Table 6. Operating frequencies in the reference platforms

Processor Used	Maximum Operating Frequency [MHz]			
	Altera StratixII EP2S180F		Altera StratixIV EP4SGX530	
	Slow Model	Fast Model	Slow Model	Fast Model
COFFEE	83	165	114	184
CAPPUCCINO	66	111	103	169

we can process in parallel. Other issues concern the way how data are exchanged between different stages and how the data reordering is carried out.

Implementation of the FFT on the Coarse-Grain Reconfigurable Platform

In the reconfigurable platform the FFT processing stages are mapped onto the BUTTER coarse-grain reconfigurable array. The CAPPUCCINO processor manages the data transfers using the DMA and enables the processing of the results (Garzia, Airoldi, Brunelli, Giliberto, & Nurmi, 2009).

We can map two butterflies onto one context of the array. Since the array employs pipelined processing, two butterflies can be processed in one clock cycle. Considering a N-point FFT, we need N/2 butterflies per stage which corresponds to N/4 cycles per stage. This number multiplied by the number of stages gives the cycles count referred to the processing. For example, a 64-point FFT requires 6 stages, so the total cycle count is 6*(64/4+3) = 112. However, this number neither takes into account the overhead related with control operations, transfers, and neither reconfiguration, nor the additional cycles required reordering the data between two consecutive stages.

The control overhead is the number of clock cycles needed for the control operations: polling of the status of BUTTER and DMA, setting the control words, enabling the processing of BUTTER, or the DMA transfers. These operations taken together can require as much as hundreds of clock cycles between consecutive processing stages.

The transfer overhead is related with the transfer of data between the two memories and the system memory. In this case one of the reasons for the transfers is the need of moving the twiddle factors to the local memory after each stage.

The reconfiguration overhead is the number of clock cycles needed to switch context. It depends on the number of contexts that can be hosted by the configuration memory. If the contexts required by the application can be stored locally, then the reconfiguration requires only one cycle. In the second case, 32 clock cycles are needed to change the context.

Additional overhead is related to the data reordering between consecutive stages. We mentioned before that the two local memories use a ping-pong mechanism. One might therefore expect that after each stage we need only to switch the processing direction between the memories before performing the following stage. Unfortunately this is not true: the way how the data are stored in the memory after each stage does not always allow a direct reprocessing. A data reordering stage is needed. Since there is no direct mechanism, it is necessary to move the results to the global memory and then back to the local memories.

Table 7 shows the execution time needed for 64-point, 1024-point and 2048-point FFTs in integer arithmetic. Table 8 shows the corresponding values using floating-point arithmetic. For BUTTER there is no difference in the cycle count since in the array integer and floating-point operations are performed in one clock cycle. In the 64-point FFT, 512 clock cycles are needed for the transfers. This happens because we need to perform 1,024

32-bit transfers throughout the FFT. Part of these transfers is due to the coefficients and part to the need of reordering the data between consecutive steps. Since the DMA works at double speed, only 512 cycles are needed. The processing takes 228 cycles. Although BUTTER only needs 112 cycles, the number doubles since the cycles are calculated with the clock of CAPPUCCINO (twice as fast). All of the remaining clock cycles (roughly 600) are overhead due to control mechanisms.

The algorithm is strongly dominated by the control and transfer overhead. However, the impact of the control overhead is constant for different amounts of data, so the speed-up compared with the reference platform increases with the number of FFT points. From Table 7, the speed-up of the 64-point is 7X, while for 1024-point is 15X. After a 1024-point FFT, the transfer overhead is increasing, because the limited size of the local memories cannot host all the samples at once. Therefore we notice a drop in the speed-up from 15X to 13.5X.

For the floating-point case (Table 8), the speed-up of BUTTER is higher, because the implementation on the single processor platform is slower.

Considering the StratixIV implementation (Table 9), we have to cope with the issues of the memory and we use the same operating frequency for BUTTER. The result is that the speed-up in comparison with the reference platform decreases significantly.

Implementation of the FFT on Ninesilica

The implementation of FFTs on Ninesilica takes advantage of the native data parallelism of the FFT algorithm. Indeed, each butterfly within a FFT stage is data independent from the others. As explained in the "Programming Model" section, the MP-SoC utilizes 8 computational nodes to perform algorithms. Hence, in the case of an N-point FFT (N/2 butterflies per stage), each node computes N/16 butterflies. If we consider two following FFT steps, data reordering is required, forming a data exchange phase between the nodes of the

Table 7. Performance evaluation of FFT implementation (integer, StratixII)

System	Frequency [MHz]	64-point			1024-point			2048-point		
		Cycles	Time	Speed-up	Cycles	Time	Speed-up	Cycles	Time	Speed-up
Reference Platform	83	22,214	267µ*s*	1x	492,071	5.9*ms*	1x	1,066,595	12.85*ms*	1x
Reconfigurable System	50	1,896	38 µ*s*	7x	20,148	0.4*ms*	15x	47,564	0.95*ms*	13.5x
Ninesilica	83	3,773	45 µ*s*	6x	74,676	0.9*ms*	6.5x	154,805	1.86*ms*	6.9x

Table 8. Performance evaluation of FFT implementation (floating-point, StratixII)

System	Frequency [MHz]	64-point			1024-point			2048-point		
		Cycles	Time	Speed-up	Cycles	Time	Speed-up	Cycles	Time	Speed-up
Reference Platform	66	25,094	380 µ*s*	1x	553,541	8.4*ms*	1x	1,189,542	18*ms*	1x
Reconfigurable System	50	1,896	38 µ*s*	10x	20,148	0.4*ms*	21x	47,564	0.95*ms*	19x

Table 9. Performance evaluation of FFT implementation (floating-point, StratixIV)

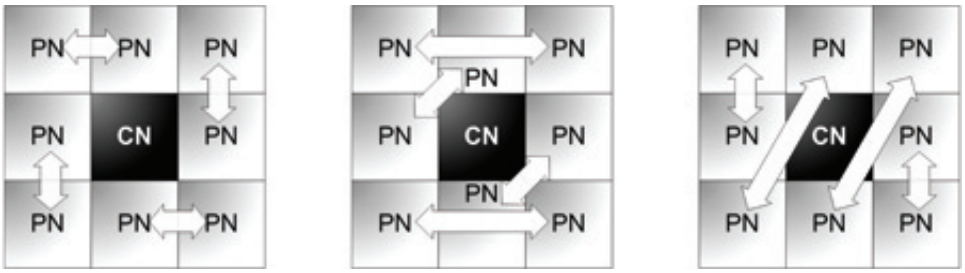
System	Frequency [MHz]	64-point			1024-point			2048-point		
		Cycles	Time	Speed-up	Cycles	Time	Speed-up	Cycles	Time	Speed-up
Reference Platform	103	25,094	243.6 μs	1x	553,541	5.4*ms*	1x	1,189,542	11.5*ms*	1x
Reconfigurable System	50	1,896	38 μs	6.4x	20,148	0.4*ms*	13.5x	47,564	0.9*ms*	13x
Ninesilica	103	3,451	33.5 μs	7.3x	82,456	0.8ms	6.75x	175,265	1.7ms	6.8x

architecture. According to the result obtained in a previous work (Airoldi, Garzia, & Nurmi, 2009b) (Figure 6), this data exchange phase is performed only in the first three FFT stages. Therefore, from the fourth FFT stage ahead there is no need of data exchange between computational nodes. This is due to the way how the samples are distributed among the nodes. In fact, at this stage each node already have got the correct subset of data to independently complete the computation of the following FFT stages and eventually returns the final values to the central node. The data exchange between computational nodes is performed in couples. Each data exchange phase involves two different actors at time (see Figure 3). The scheduling of which butterfly has to be performed on which node was studied to reduce the latency of the data exchange phase and to obtain the minimum number of data exchange phases required. This happens because the maximum number of hops on the network is limited to two.

Table 7 shows the implementation results of different FFT sizes (64-point, 1024-point and 2048-point) for integer data. The MP-SoC performs a 2048-point FFT in 154,805 clock cycles (including the initial data transfer), which compared to the implementation on a single COFFEE RISC core leads to a speed-up of 6.89X. If we strip off the initial transfer, the FFT is then performed in 109,742 clock cycles leading to a speed-up of 9.7X, which is higher than the intuitively expected limit of 8X (number of computational nodes). This is due to the ability of the MP-SoC to reorder data on the fly during data transfers whereas the single core has to reorder data in additional preprocessing stages.

Moreover, analyzing the speed-ups that we obtained for different FFT sizes we notice that the speed-up grows with the FFT sizes. This is due to the low impact of the required data exchange phases and to the low latency of intra-cluster communication. The combination of these factors

Figure 6. Communication between nodes in the different FFT stages (Adapted from Airoldi, Garzia, & Nurmi, 2009b)



helps to raise the speed-up closer to the expected limit. However, even if we consider small FFT sizes such as 64-point the speed-up does not drop under 6X, showing the high efficiency of parallelization in the proposed approach.

Comparison between the Two Proposed Architectures

Looking at Table 7, we can see that the implementation of the 32-bit integer FFT is faster in the reconfigurable architecture in comparison with the MP-SoC even though the first shows a stronger dependence on the size of the FFT. A reconfigurable approach is closer to hardware performance but it needs longer development time. In fact, the mapping on the coarse-grain array requires a higher effort than the MP-SoC, the latter being entirely programmable using C. The resource utilization is basically the same, but the reconfigurable array is more complex and its maximum operating frequency is smaller.

The reconfigurable array on the StratixII can perform a floating-point FFT with the same result. On the other hand, the mapping of Ninesilica-FP is not possible on the same device.

Considering Table 9, we can say that the two systems are not able to fully exploit the capabilities of the StratixIV device. In fact, the speed-ups are the same as in the StratixII case. Only a small percentage of the available resources are used. The maximum frequency rises except for the memories in the reconfigurable platform.

Both of the architectures should be revised for an efficient StratixIV implementation. In the MP-SoC, it is possible to increase the number of nodes simply extending the NoC infrastructure, but the software implementation needs to be parallelized in accordance with the number of processors. Instead, the reconfigurable platform can be revised increasing the size of BUTTER array. Augmenting the number of columns it is possible to achieve a higher parallelism and therefore to speed-up the implementation. In this case, the software part does not need any significant modification, but the interfaces between array and the rest of the system as well as the local memories must be redesigned.

Comparison with the State of the Art

Table 10 presents some figures related to alternative solutions. We considered some architectures implemented either on ASIC or FPGA. The area figures are not reported because the comparison between ASIC and FPGA implementations is not possible. As test case we considered a 1,024-point FFT implemented in integer arithmetic.

The Altera FFT Megafunctions (Altera, 2009a) is a HDL block provided by Altera to implement the FFT on the FPGA. It is customizable at design time for the FFT size required by the application. For this reason this dedicated implementation is the fastest.

Morphosys2 (Kamalizad, Pan, & Bagherzadeh, 2003) is a coarse-grain reconfigurable array similar to BUTTER. It is particularly suitable for the FFT implementation, since the number of cycles is close enough to the dedicated architecture. However, its size does not allow the mapping on FPGA. From the literature (Kamalizad, Pan, & Bagherzadeh, 2003) we can see that its transistor count is 20M. It is almost 10X bigger than BUTTER, whose transistor count is 2,368,000 in an ASIC implementation (Brunelli, Garzia, & Nurmi, 2008b). Since BUTTER utilizes more than 10% of the biggest StratixIV device, we can argue that Morphosys2 would never fit on such device.

The FFT processor developed by Kuo, Wen, and Wu (2003) is a programmable solution intended for wireless applications and implemented on ASIC. It is possible to decide at run-time the size of the FFT to use. The reason why it outperforms our platforms is that is intended only for the processing of the FFT, therefore its architecture is optimized for the task. In addition, the ASIC implementation guarantees higher operating frequencies.

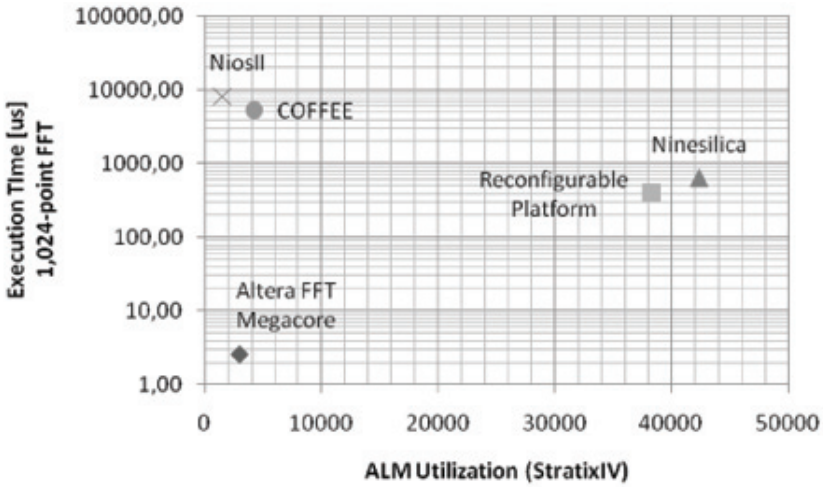
Table 10. Comparison with state-of-the-art architectures

System	Implementation	Frequency [MHz]	1024-point FFT Performance	
			Cycles	Time
Altera FFT Megafunction	FPGA Altera StratixIV	396	1,024	2, 59 μs
Morphosys2	ASIC 130 *nm*	450	2,613	5.8 μs
Programmable 64-2,048 FFT Processor	ASIC 350 *nm*	80	3,200	40 μs
Reconfigurable System	FPGA Altera Stratix IV	50	20,148	0.4*ms*
Ninesilica	FPGA Altera Stratix IV	114	74,676	0.65*ms*
ARM926EJ-S	ASIC 45*nm*	400	460,000	1.15*ms*
NiosII	FPGA Altera Stratix IV	185	1,516,371	8.2*ms*

With regard to general-purpose solutions, the NiosII processor (Altera, 2009c) and the ARM processor are general-purpose RISC cores implemented respectively on FPGA and on ASIC. Our platforms expose a higher degree of parallelism enabling them to outperform both these processor cores. Ninesilica is 12.6X faster than NiosII and 1.8X faster than the ARM. The reconfigurable platform is 20X faster than NiosII and 2.8X faster than the ARM. Notice also that our platforms provide the support for floating-point arithmetic.

Figure 7 depicts the ratio between the execution times and the ALM utilization of different FPGA solutions. The Altera FFT Megacore is the only fixed hardware solution and shows the best performance in this design space. The other solutions are programmable. NiosII and COFFEE are single cores. NiosII is a smaller but slower than COFFEE. They are characterized by low resource utilization, but the FFT implementation is slower than the two proposed platforms. Therefore our platforms are preferable in situations where the flexibility must be coupled with strict timing requirements.

Figure 7. Execution times (FFT 1K points) vs. ALM utilization in the described FPGA solutions



CONCLUSION

This work presents two general-purpose architectures implemented on FPGA. The first architecture is based on a coarse-grain reconfigurable array, the second on a homogeneous MP-SoC. The two architectures have been mapped on both Altera StratixII and Altera StratixIV devices. Although the implementation on StratixIV shows higher operating frequencies, the two platforms should be revised in order to fully exploit the characteristics of the device.

As a case study, we implemented the FFT algorithm on the two architectures. The mapping showed that we can get a considerable speed-up in comparison with a single-processor reference platform. The speed-up is higher in the reconfigurable architecture, but the MP-SoC provides an easier programming interface that is completely based on C language. The successful implementation of two architectures proves that it is possible to program a FPGA device solely with a simple programming language like C.

ACKNOWLEDGMENT

Authors would like to acknowledge the Foundation of Nokia Corporation and GETA, Research School in Electronics, Telecommunications and Automation, for their financial support. Authors would like to thank Claudio Brunelli and Carmelo Giliberto for their invaluable help.

REFERENCES

Ahonen, T., & Nurmi, J. (2006). Programmable switch for shared bus replacement. In *Proceedings of the 2006 International Conference on Ph.D. Research in Microelectronics and Electronics (PRIME '06)* (pp. 241-244). Washington, DC: IEEE. ISBN: 1-4244-0156-9 DOI: 10.1109/RME.2006.1689941

Airoldi, R., Garzia, F., Ahonen, T., Milojevic, D., & Nurmi, J. (2009a). Implementation of W-CDMA cell search on a FPGA based Multi-Processor System-on-Chip with power management. In *Proceedings of the IX International Symposium on Systems, Architectures, MOdeling and Simulation (SAMOS IX)* (LNCS, pp. 88-97). Berlin: Springer Verlag.

Airoldi, R., Garzia, F., & Nurmi, J. (2009b). Implementation of a 64-point FFT on a Multi-Processor System-on-Chip. In *Proceedings of the 5th International Conference on Ph.D. Research in Microelectronics & Electronics (Prime '09)*, Cork, Ireland (pp. 20-23). Washington, DC: IEEE.

Altera. (2007). *Stratix II Device Handbook*.

Altera. (2009a). *FFT Megacore Function*.

Altera. (2009b). *Stratix IV Device Handbook*.

Altera. (2009c). *NiosII Processor Reference Handbook*.

Bahn, J. H., Yang, J., & Bagherzadeh, N. (2008). Parallel FFT algorithms on network-on-chips. In *Proceedings of the Fifth International Conference on Information Technology: New Generations (ITNG '08)* (pp. 1087-1093). Washington, DC: IEEE.

Baumgarte, V., Ehlers, G., May, F., Nückel, A., Vorbach, M., & Weinhardt, M. (2003). PACT XPP-a Self-Reconfigurable Data Processing Architecture. *The Journal of Supercomputing*, *26*(2), 167–184. doi:10.1023/A:1024499601571

Brunelli, C., Campi, F., Mucci, C., Rossi, D., Ahonen, T., & Kylliäinen, J. (2008a). Design space exploration of an open-source, IP-reusable, scalable floating-point engine for embedded applications. *Journal of Systems Architecture*, *54*(12), 1143–1154. doi:10.1016/j.sysarc.2008.05.005

Brunelli, C., Garzia, F., & Nurmi, J. (2008b). A coarse-grain reconfigurable architecture for multimedia applications featuring subword computation capabilities. *Real-Time Image Processing*, *3*(1-2), 21–32. doi:10.1007/s11554-008-0071-3

Camarda, F., Prevotet, J.-C., & Nouvel, F. (2009). Implementation of a reconfigurable fast fourier transform application to digital terrestrial television broadcasting. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'09)* (pp. 353-358).

Chinnery, D. G., & Keutzer, K. (2000). Closing the gap between ASIC and custom: an ASIC perspective. In *Proceedings of the Design Automation Conference (DAC2000)* (pp. 637-642).

Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, *19*(90), 297–301. doi:10.2307/2003354

Garzia, F., Airoldi, R., Brunelli, C., Giliberto, C., & Nurmi, J. (2009). Mapping of the FFT on a reconfigurable architecture targeted to SDR applications. In *Proceedings of International Symposium on System-on-Chip (SOC'09)*, Tampere, Finland (pp. 157-160). Washington, DC: IEEE.

Garzia, F., Brunelli, C., & Nurmi, J. (2008a). A pipelined infrastructure for the distribution of the configuration bitstream in a coarse-grain reconfigurable array. In *Proceedings of the 4th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC '08)*, Montpellier, France (pp. 188-191). ISBN: 978-84-691-3603-4

Garzia, F., Brunelli, C., Rossi, D., & Nurmi, J. (2008b). Implementation of a floatingpoint matrix-vector multiplication on a reconfigurable architecture. In *Proceedings of the 15th Reconfigurable Architecture Workshop (RAW '08)* (pp. 1-6). Washington, DC: IEEE.

Kamalizad, A. H., Pan, C., & Bagherzadeh, N. (2003). Fast Parallel FFT on a Reconfigurable Computation Platform. In *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBACPAD' 03)*. Washington, DC: IEEE.

Kim, D., Kim, M., & Sobelman, G. (2005). Parallel FFT computation with a CDMA-based network-onchip. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS'05)* (Vol. 2, pp. 1138-1141). Washington, DC: IEEE.

Kuo, J.-C., Wen, C.-H., & Wu, A.-Y. (2003). Implementation of a programmable 64-2048-point FFT/IFFT processor for OFDM-based communication systems. In *Proceedings of the 2003 International Symposium on Circuits and Systems (ISCAS'03)* (Vol. 2, pp. 121-124).

Kuon, I., & Rose, J. (2006). Measuring the gap between FPGAs and ASICs. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays (FPGA '06)* (pp. 21-30). New York: ACM.

Kylliäinen, J., Ahonen, T., & Nurmi, J. (2007). General-purpose embedded processor cores- the COFFEE RISC example. In J. Nurmi (Ed.), *Processor Design: System-on-Chip Computing for ASICs and FPGAs* (Chapter 5, pp. 83-100). New York: Springer. ISBN-10: 1402055293 ISBN-13: 978-1-4020-5529-4

Mei, B., Vernalde, S., Verkest, D., Man, H. D., & Lauwereins, R. (2003). ADRES: An architecture with tightly-coupled VLIW processor and coarse grained reconfigurable matrix. In *Field-Programmable Logic and Applications* (Vol. 2778, pp. 61-70). ISBN 978-3-540-40822-2

Meyer, R., & Schwarz, K. (1990). FFT implementation on dsp-chips-theory and practice. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'90)* (Vol. 3, pp. 1503-1506).

Nurmi, J. (2008, July 23-27). Silicon Café: A Heterogeneous Multi-Processor Platform Based on Coffee (RISC Core), invited mini-keynote. In *Proceedings of the 8th International Forum on Application-Specific Multi-Processor SoC (MP-SoC'08)*, Valkenburg, The Netherlands.

Ojail, M., David, R., Chevobbe, S., & Demigny, D. (2009). A reconfigurable FIR/FFT unit for wireless telecommunication systems. In *Proceedings of the International Conference on, Field Programmable Logic and Application (FPL'09)* (pp. 645-648). Washington, DC: IEEE.

Rivaton, A., Quevremont, J., Zhang, Q., Wolkotte, P., & Smit, G. (2005). Implementing Non Power-of-Two FFTs on Coarse-Grain Reconfigurable Architectures. In *Proceedings of the International Symposium on System-on-Chip (SoC'05)* (pp. 82-85). Washington, DC: IEEE.

Sangiovanni-Vincentelli, A. (1993). Some considerations on Field Programmable Gate Arrays and their impact on system design. In *Field-Programmable Gate Arrays: Architecture and Tools for Rapid Prototyping* (LNCS 705/1993, pp. 26-34). New York: Springer. ISSN 0302-9743 ISBN 978-3-540-57091-2

Zhang, Q., Kokkeler, A., & Smit, G. (2007). An efficient FFT for OFDM based cognitive radio on a Reconfigurable Architecture. In *Proceedings of the International Conference on Communications (ICC'07)* (pp. 6522-6526). Washington, DC: IEEE.

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 3, edited by Seppo Virtanen, pp. 24-43, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Section 3
On-Chip Communication

Chapter 10
Performance Analysis of On-Chip Communication Structures under Device Variability

Faiz-ul Hassan
University of Glasgow, UK

Wim Vanderbauwhede
University of Glasgow, UK

Fernando Rodríguez-Salazar
University of Glasgow, UK

ABSTRACT

On-chip communication is becoming an important bottleneck in the design and operation of high performance systems where it has to face additional challenges due to device variability. Communication structures such as tapered buffer drivers, interconnects, repeaters, and data storage elements are vulnerable to variability, which can limit the performance of the on-chip communication networks. In this regard, it becomes important to have a complete understanding of the impact that variability will have on the performance of these circuit elements in order to design high yield and reliable systems. In this paper, the authors have characterized the performance of the communication structures under the impact of random dopant fluctuation (RDF) for the future technology generations of 25, 18, and 13 nm. For accurate characterization of their performance, a Monte Carlo simulation method has been used along with predictive device models for the given technologies. Analytical models have been developed for the link failure probability of a repeater inserted interconnect which uses characterization data of all communication structures to give an accurate prediction of the link failure probability. The model has also been extended to calculate the link failure probability of a wider communication link.

DOI: 10.4018/978-1-4666-0912-9.ch010

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

INTRODUCTION

As transistor gate lengths continue to shrink according to Moore's law (Moore, 1965), designers are now able to incorporate complete systems with complex functionalities in a single System-on-Chip (SoC). As a result, Embedded System design is migrating from board level integration to on-chip level integration, where components are often-times developed by third parties and integrated by the system designer. The increase in complexity of these systems has introduced a number of challenging problems in the design, project management, simulation and verification of these devices. For instance, according to the ITRS (ITRS, 2003), as the technology scales, the devices are increasingly becoming faster relative to the interconnect. Therefore, modern SoC and Embedded designers are faced with the difficult task of orchestrating the computation of a large number of fast local third party hardware blocks, across the whole chip, by using (relatively) progressively slower interconnects. Consequently, on-chip communication is rendering major constraints to the performance of the system.

Network-on-Chip (NoC) has been proposed as a suitable design methodology (Benini & Micheli, 2002; Dally & Towles, 2001) for modern SoCs which offers an interconnect-centric system architecture to manage the problems of on-chip communication. It works as a small on-chip communication network consisting of typical communication layers such as physical, data link, network, transport and application, each one geared for particular functions. The design of the physical layer is of vital importance as it provides a physical communication media from router-to-router and from router-to-functional units (FUs) for the exchange of data between them. The global interconnect between the routers and semi-global interconnect between the routers and functional units put many challenges for designing them for high performance, especially in deep sub-micron region (Lee & Yoo, 2004). These communication links can either be (a) multi-bit parallel, (b) partially parallel where an *n*-bit packet or flit is divided in to a small number of *m*-bits or (c) source-synchronous serial (Kim & Sobelman, 2006).

The increase in clock speed and chip size means that synchronization can no longer be obtained by the use of a global clock, and that other models of synchronization (such as the use of multiple clock domains or by the Globally Asynchronous Locally Synchronous (GALS) approach) are required. Effectively the maximum synchronous area is determined by the clock skew. The individual synchronous blocks can communicate with each other synchronously, asynchronously or in a self timed manner. In this paper, we have chosen to focus on cross clock domain communication structures, since due to their high performance and low design effort, it is likely that they will be used in future designs, in which an overlap of different synchronous blocks on the chip communicate with each other synchronously. However, these structures are not completely immune to skew which can develop in the global clock and wide data communication links.

On-chip communication in these systems generally consists of clock distribution network and data links between the functional units. Their performance depends on the performance of the constituting elements like, line drivers, repeaters, interconnect and data storage elements (Flip-flops or latches). Owing to the technology scaling, the variability in the devices and interconnect has emerged as one of the major challenges faced by the silicon industry (Asenov, 2007; Vishvanathan, Ravikumar, & Menezes, 2005; Borkar, 2005; Lin, Spanos, Milor, & Lin, 1998) and the impact of variability on the performance of the circuits has become exceptionally critical (Nassif, 2000; Keutzer & Orshansky, 2002). The variability at deep sub-micron (DSM) region has resulted in the transition from a deterministic design paradigm to probabilistic methods (ITRS, 2005; Narasimhan & Sridhar, 2007) and so there could be large

discrepancies between the design and the products being manufactured (Agarwal, Sylvester, & Blaauw, 2006). The impact of process variations is increasing with every new technology generation because process tolerances are not scaling exactly in the same proportion with the design dimensions (Bowman, Duvall, & Meindl, 2002). The variability in the devices and interconnect produces delay uncertainty in the communication structures which can introduce timing uncertainty in the clock and data signals. The technology scaling has pushed the clock frequencies in the gigahertz region in the continuous quest for higher circuit performances and as a result, clock period has reduced to below nanoseconds. Therefore, uncertainty in the clock and data signals can produce timing violations at various clocked registers under the very tight design margins left therein. The design margins and consequently the maximum achievable performance and yield depend on the magnitude of the variability.

In the context of on-chip communication designed under the abstraction of NoC, the variability can adversely affect the performance of the clock distribution circuits and the links between the routers and routers-to-functional units. The constituting elements of these structures like line drivers, repeaters, interconnect and flip-flops are vulnerable to variability. The delay variability in these elements is added statistically to produce cumulative delay variability which will have a (hopefully small) probability for which the design constraints are not met and lead to timing violations. In NoC, the faults caused by these timing violations are very important as they can produce collision of packets and results in an increase of latency (Mondal, Ragheb, Wu, Aziz, & Massoud, 2007) in the best case, to link and communication failures on the worst.

In order to evaluate the performance of a particular NoC infrastructure designed in future technology generations, for instance the links, an accurate characterization of the performance of the individual communication structures are required in those technologies. It becomes difficult to make this evaluation for the future technologies for which the characterization data from the manufacturing industry is not available. However, such a study becomes important for designers and academia so that they can formulate efficient design methodologies for the coming technology generations under tight design margins and other technology challenges.

In this work we present a systematic study to investigate the effect that variability will introduce in the communication structures for the future technology generations of 25, 18, and 13 nm physical lengths using predictive device models. Statistical variability is the most dominant form of variability in 45 nm technology (Aikawa et al., 2008) and is expected to further grow in next technology generations (Ishimaru, 2007). Random dopant fluctuation (RDF) in the devices is the most important source of statistical device variability (Asenov, Brown, Davies, & Slavcheva, 2003) and therefore the study is focused on the impact of RDF on the performance of NoC communication structures. It is important to note however that RDF is not the only source of variability of actual designs, and these other effects would need to be considered along with RDF to obtain a whole picture of the problem (for instance, interconnect variability and the effect of intra-die systematic variability has not been considered in this work due to the unavailability of data from the foundry for the given future technologies).

The characterization data obtained through this study can be used to calculate link failure probability for a repeater inserted link. An analytical model has been developed for this purpose that can take the values of the characterization data of different elements like line drivers, repeaters and flip-flops and gives link failure probability as a function of the link operating frequency. The model has been extended to approximate the probability of link failure for a wider link and gives link failure probability as a function of the number of lines in the bus and the operating frequency.

This study, is to the best of our knowledge, the first published work in which the performance of individual communication structures (using line drivers, repeaters and flip-flops) has been studied under device variability due to RDF.

The rest of the paper is organized as follows. First of all, the methodology adopted in this study is presented. In the next section, the effect of technology scaling on the gate delay of the devices has been given. After that, buffers of different sizes have been used to find a relation between buffer size and delay variability. Afterwards, investigations have been carried out to study the impact of device variability on the performance of tapered buffers. A statistical relation has been derived for the delay variability as a function of number of stages in the driver chain. In the next section, the results of the performance of repeaters under device variability have been discussed. Similarly, the results for the timing characteristics of the flip-flops are given in the next section. Analytical models have been developed for the link failure probability of a single interconnect line and a multi-bit communication link and are placed in the next section. Finally in the last section, some conclusions of the study are described.

METHODOLOGY

For this study, the future technology generations of 25, 18 and 13 nm physical gate length bulk-MOSFETs are considered. For these technology generations, predictive device model card libraries have been used for NMOS and PMOS devices and are based on Glasgow's 3D "atomistic" statistical device simulations with random dopant fluctuations (RDF). The performance of 6-T SRAM has been evaluated in (Cheng, Roy, Brown, & Asenov, 2006) using the same device model cards. A detailed description of the methodology adopted for the extraction of these model cards have been given in (Cheng, Roy, & Lema, 2005). The devices in each library are macroscopically similar but are microscopically different due to the difference in the number and position of the dopant atoms in the channel. So all the devices in each library have different characteristics due to statistical variations in the device parameters and the distribution of these variations represents the distribution of variations found in the general population. For the statistical analysis, Monte Carlo simulation method has been used with random selection of the devices from the given model card libraries, while constructing different circuits. The circuits are biased with a supply voltage of 1.1V, 1.0V and 0.9V for the technology generations of 25, 18, and 13 nm, respectively (ITRS, 2005). Different delay measurements taken during this study are corresponding to 50% of the signal levels during the transitions. In this paper, the delay variability has been expressed as $3\sigma/\mu$, where μ is the mean delay and σ is the standard deviation of the delay.

TECHNOLOGY SCALING AND GATE DELAY OF THE DEVICES

In a particular technology generation, the maximum clock speed and the speed at which computation can be performed, is determined by the gate delay. On-chip communication will need to be designed to support these speeds in order to preclude data starvation. Due to statistical variation in the devices, gate delay is no longer a fixed quantity, but a random variable which follows a given distribution. For better estimation of the maximum clock speed, statistically accurate description of the delay is required to be derived with consideration of the effects introduced due to variability. In this section, we study the impact of device variability in the gate delay of an inverter in a given technology, as representative of delay in more complex combinational circuits and gates. This delay has been measured in terms of FO4 delay and is used as a reference or benchmark to which we can compare the results of the communication structures. The metric

FO4 delay or "fan-out-of-four inverter delay" has been used elsewhere (Ho, Mai, & Horowitz, 2001) and is a quite reasonable metric, as four is the typical average gate connectivity in a digital circuit (Liu, Zheng, & Tenhunen, 2004). This is defined as the delay through an inverter driving four copies of it. Since the effect of variability is more pronounced in smaller geometries, therefore FO4 delay has been measured corresponding to the delay of minimum sized inverters, as shown in Figure 1. Here we have used minimum sized inverters of size w_{min}=25, 18 and 13 nm for the given three technology generations of 25, 18 and 13 nm, respectively.

HSPICE simulations were performed (using the Monte Carlo method described in the methodology section) and FO4 delay measurements were taken for the given technology generations. The mean value of the FO4 delay is plotted for the three technologies and results are shown in Figure 2. The standard deviation of the FO4 delay is also represented in the form of error bars. It can be seen that the mean value of the FO4 delay decreases, whereas the delay variability increases, with the decrease of the gate length. In order to probe into the nature of the delay distributions, histograms are plotted from the measurement data and are shown in Figure 3. It becomes evident that the dispersion of the distributions increases with gate length scaling. Moreover, the distributions are asymmetric about the mean delay and the degree of asymmetry increases with the decrease of the gate length. The positively skewed nature of the distributions has a detrimental impact on the performance of the circuits as a significant number of samples beyond the nominal value imply a long tail which will certainly limit the speed of the circuits and also introduces reliability issues.

More importantly, it becomes obvious that the dispersion and the worst case of FO4 delay grow hyper-linearly as the technology scales down. Due

Figure 1. The definition of FO4 delay

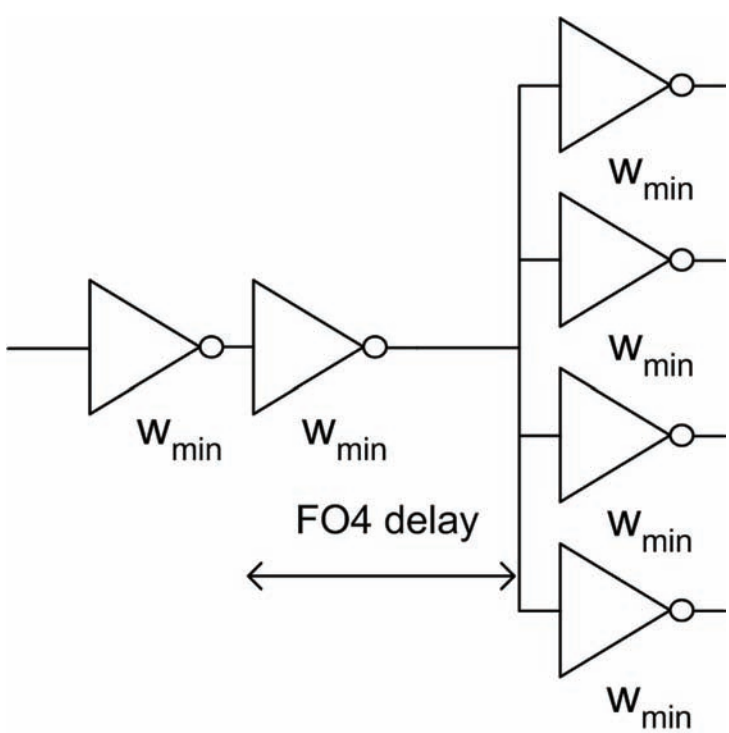


Figure 2. FO4 delay during high-to-low transition for different technology generations. The error bars represent the uncertainty in delay (±1σ)

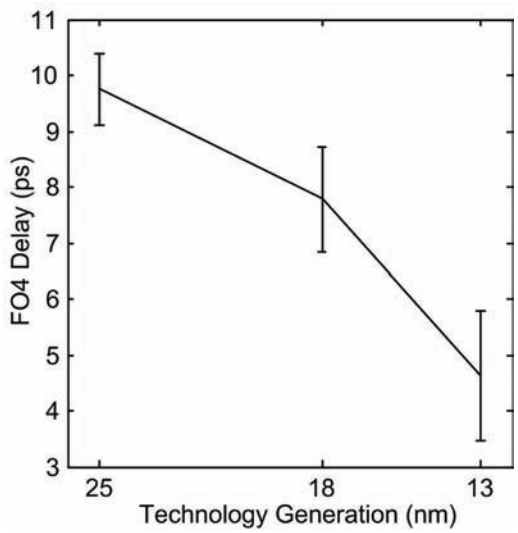
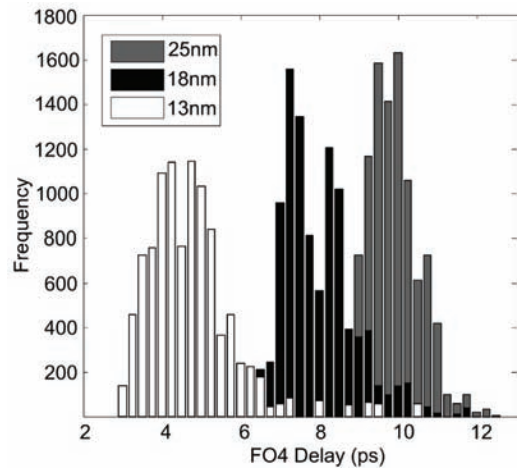


Figure 3. Delay distribution of minimum sized inverters with a fan-out of four for the technology generations of 25, 18, and 13 nm



to this fact, the performance of the circuits will certainly be affected unless some corrective measures are not incorporated in the design of the circuits. The effect becomes more important in the design of synchronous systems under the tight design margins typical of high performance circuits. It is obvious then, that variability in the devices warrants a careful consideration during the design of high performance circuits.

DELAY UNCERTAINTY IN BUFFERS

The efficiency of high performance circuits not only depends on the performance of computational elements but also depends greatly on the communication network responsible for the exchange of data between the computational elements. Delay uncertainty in the clock signal can produce setup and hold time violations at the data registers. Similar violations can also occur in the data signals. A large number of buffers are used in these communication networks that can introduce delay uncertainty in the signals. For designing high performance circuits under tight timing constraints, the delay uncertainty will have to be reduced. Therefore, design methodologies are required to be explored that reduces delay uncertainty.

The delay of a CMOS buffer (inverter) to the first order, as given by Bakoglu (Bakoglu, 1990) is

$$t_d = 0.7 R_{drv} C_L \tag{1}$$

where R_{drv} is the on-resistance and C_L is the capacitive load at the output of the inverter. The inverter resistance R_{drv}, which is approximated by averaging the drain currents at the extreme points of the high-to-low and low-to-high transitions, is given by

$$R_{drv} = \frac{L}{W \mu C_{ox} (V_{DD} - V_T)} \tag{2}$$

where,

L = transistor gate length,
W = transistor gate width,
C_{ox} = gate capacitance per unit area,

μ = mobility of the transistor,
V_{DD} = supply voltage.

A variation in these factors will cause the inverter resistance (and consequently the drain current) to change and eventually will result in the variability of gate delay. In deep sub-micron (DSM) region, it is very difficult to precisely control all transistor parameters during the fabrication process. Therefore in a batch of similar transistors, different parameters can have a complete distribution with some nominal value and a wide spread about this nominal value. For instance, due to random dopant fluctuations (RDF), the threshold voltage V_T of the transistors will have some distribution (wider or narrower). Hence, the on-resistance of the transistors can no longer be treated as a fixed quantity; rather it will follow a distribution, resulting in the distribution of the inverter delay. Let χ_{dev} represents the effect of RDF on V_T, then the on-resistance of the inverter will be given by:

$$R_{drv} = \frac{L}{W\mu C_{ox}(V_{DD} - V_T\chi_{dev})} \tag{3}$$

Therefore,

$$t_d = \frac{0.7LC_L}{W\mu C_{ox}(V_{DD} - V_T\chi_{dev})} \tag{4}$$

In order to evaluate the effect of variations in V_T on the delay of the inverter, we differentiate t_d with respect to χ_{dev}, yielding

$$\frac{\partial t_d}{\partial \chi_{dev}} = \frac{0.7LV_TC_L}{W\mu C_{ox}(V_{DD} - V_T\chi_{dev})^2} \tag{5}$$

This shows that the sensitivity of the inverter delay is inversely proportional to the size (width) of the inverter. In the same way, the delay sensitivity to other transistor parameters can be determined. Therefore, we can deduce that the technique of circuit scaling can be used to minimize the effect of RDF on delay variability.

We now quantify the effect of RDF on the delay performance of individual buffers of different sizes. SPICE models are developed for the buffers of sizes $1w_{min}$, $2w_{min}$, $3w_{min}$, $5w_{min}$, $7w_{min}$, $10w_{min}$, $15w_{min}$, $20w_{min}$, and $25w_{min}$ with a load of $25w_{min}$ buffer connected at the output of each buffer (where w_{min}=size of the minimum sized buffer=25, 18 and 13 nm for the given three technology generations). The results of Monte Carlo simulations are shown in Figure 4, where mean delay and delay variability are plotted for the given buffer sizes. It can be seen that the buffer delay and dispersion in delay is inversely proportional to the buffer size, as expected from Equation (4) and (5). More importantly, the relation is not linear and a small increase in the size of the buffer can give us significant advantage towards the improvement in delay and delay variability, especially at smaller buffer sizes.

It has also been found that there is a difference in the amount of delay variability for low-to-high and high-to-low transitions, as shown by the dashed lines in Figure 5. For instance, it is larger during high-to-low transitions and the effect is more prominent at smaller buffer sizes. This is due to the reason that during high-to-low transition, NMOS transistor is conducting and PMOS is going to turn-off. The size of the PMOS transistor is normally taken as twice the size of the NMOS transistor to make identical delay in both swings. The difference in the sizes of NMOS and PMOS transistors in the inverter is responsible to make the difference in delay variability. Therefore, while considering delay variability, its magnitude in both swings is required to be considered.

If we assume that delay variations in buffers of different sizes are independent of each other and if $V_d(1w_{min}) = 3\sigma(1w_{min})/\mu(1w_{min})$ is the delay variability of a minimum sized inverter in a given technology generation, then the delay vari-

Figure 4. (a) Mean buffer delay, (b) Delay variability, plotted as a function of buffer size for 18 nm technology generation. The curves have been plotted for the average response in low-to-high and high-to-low transitions

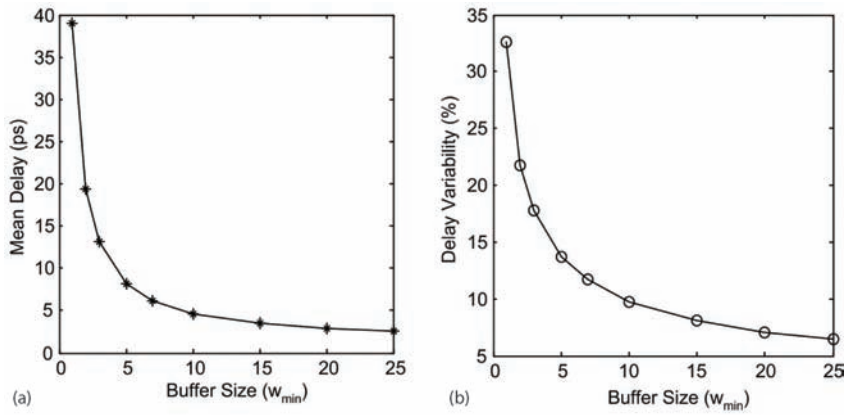
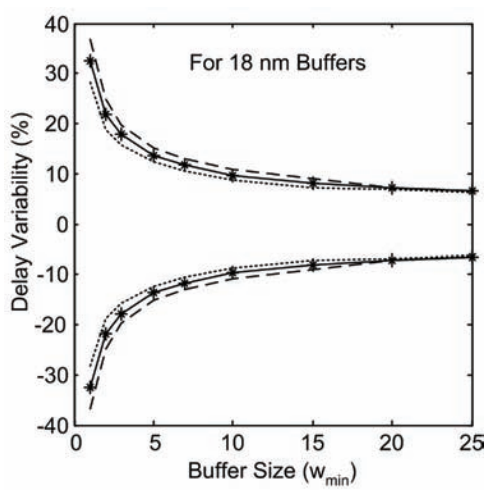


Figure 5. Delay variability plotted against buffer size for 18 nm buffers. The smaller dashed lines represent delay variability for low-to-high transition and bigger dashed lines for high-to-low transition. Similarly, the solid lines are for the average response



ability of an inverter of size X can be approximated as

$$V_d\left(mw_{\min}\right) = \frac{3\sigma(mw_{\min})}{\mu(mw_{\min})} = \frac{3\sigma(1w_{\min}) \,/\, \mu(1w_{\min})}{\sqrt{m}} \tag{6}$$

This relation can be used to make an estimate of the delay variability in a buffer of given size.

TAPERED BUFFER DRIVERS

In CMOS integrated circuits, large capacitances are common in large fan-out circuits and/ or in long range interconnects. Therefore, in order to source and sink relatively large amount of current, a tapered buffer system is used to drive such circuitry, especially where the load is predominantly capacitive. For instance, in clock distribution network, such drivers are used to power up the clock source signal. The device variability can introduce delay uncertainty in these buffers resulting in the introduction of skew in clock distribution networks and in on-chip communication networks, thus limiting the performance and yield.

Such drivers are comprised of a chain of cascaded inverters with increasing buffer sizes as shown in Figure 6. The drivers are sized according to (Cherkauer & Friedman, 1995) with total number of inverters in the system equal to N such that the last inverter in the chain can drive the load connected at its output. For the optimal delay performance of tapered buffers, a logarithmic

tapering factor $\beta = e = 2.72$) (has been proposed (Nemes, 1984), though practically this value is seldom used.

While using such buffers in the circuits, their delay performance under device variability needs to be known. Therefore, in this work we have investigated their delay performance when implemented in the given three technologies. A chain of five inverters (the first stage being of minimum size) has been used for this study and adjacent inverters in the driver chain are sized with a tapering factor β equal to 3. The delay performance of the drivers has been studied during low-to-high and high-to-low input transitions.

The results show that as we proceed along the chain of inverters, the cumulative mean delay increases at each next stage, in a linear manner, as shown by the straight lines in Figure 7. However, the slope of these lines decreases with technology scaling, which means that tapered buffers can be constructed with relatively lesser delay penalty for smaller technologies. As we have already shown that due to device variability, the inverters used in the tapered buffer drivers introduce delay uncertainty at each stage which accumulates statistically and appears at the output of the driver. The amount of this delay variability increases in a non linear fashion with the number of stages and is shown in the form of error bars in Figure 7. This delay variability will have detrimental effect in the designing of high speed circuits. The tapered buffer drivers from all the given technology generations show the same response and maximum delay variability has been observed in 13 nm drivers.

Since inverters of different sizes are used in the driver chain, the share of each stage towards delay variability cannot be the same. The results show that earlier stages of the tapered buffer drivers contribute major portion of the delay variability (as shown in Figure 8), because they are constructed with relatively smaller transistors. Again, it has also been found that the delay uncertainty introduced by each stage is different during low-to-high and high-to-low transitions due to the reason mentioned before. However, this difference reduces as we move along the chain towards larger sizes. This fact is shown in Figure 9 where the gap between the two solid lines gradually decreases with stage number and finally the lines almost coincide after the fifth stage. The difference developed in all the stages travels through the chain and accumulates accordingly, thus making a difference in the delay variability at the output of the nth stage, depending upon the type of the input transition. For instance, the difference in the delay variability for low-to-high and high-to-low input transitions at the output of the 3rd and 5th stages is about 9% and 5%, respectively, for 13 nm drivers. It is also observed that maximum delay variability appears in the cumulative and stage delays for low-to-high input transitions, as shown in Figure 9.

During the designing of the circuits, tapering factor $\beta = e$ is not always the best choice and so tapered buffers with different tapering factor are

Figure 6. Tapered buffer driver system

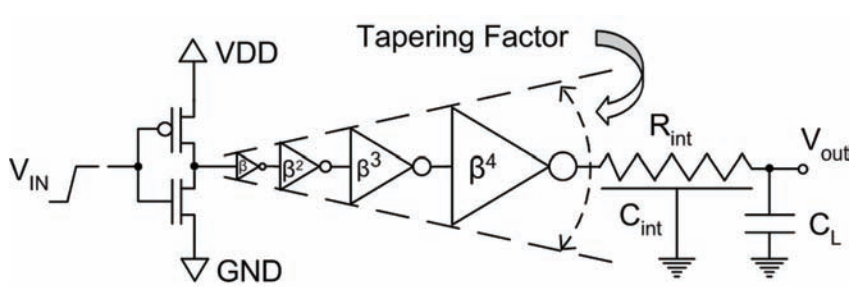


Figure 7. Cumulative mean delay in tapered buffer drivers of the given three technology generations along with the delay uncertainty shown as error bars (corresponding to 1σ)

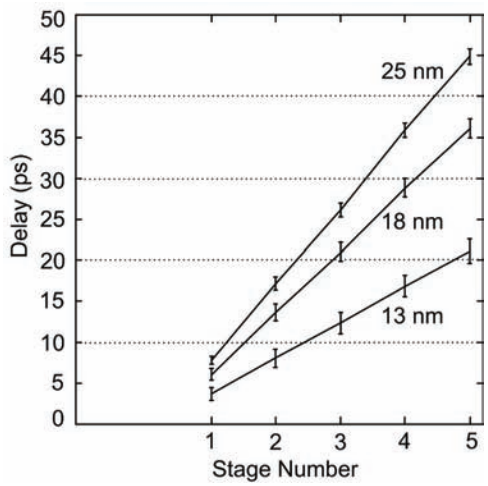


Figure 8. Delay variability introduced by different stages of the tapered buffer driver for low-to-high input transition

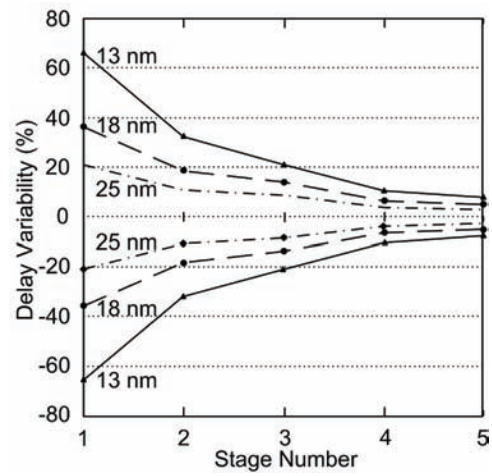
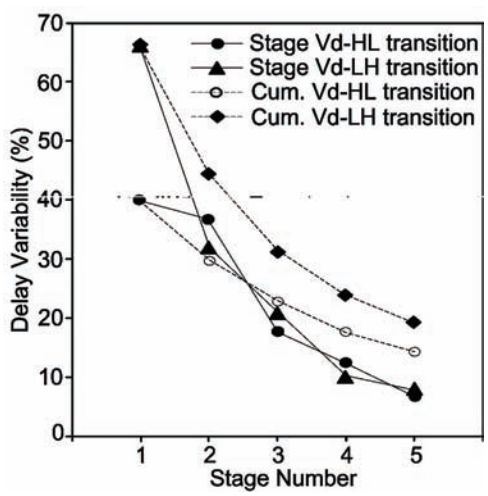


Figure 9. Cumulative and stage delay variability during low-to-high and high-to-low transitions for 13 nm tapered buffer driver

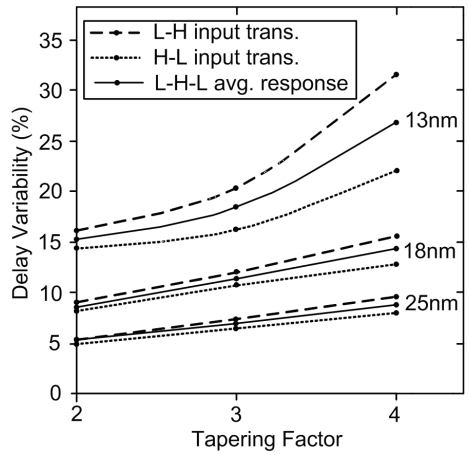


used. Therefore, we have extended the study on tapered buffers to see the effect of β on their delay characteristics. The results are shown in Figure 10, where delay variability has been plotted for the tapered buffers having tapering factor of two, three and four for the given three technology generations. In all these cases, tapered buffers are so constructed that their first stage is a minimum sized inverter w_{min} (=25, 18 and 13 nm for the technology generation of 25, 18 and 13 nm, respectively) with number of stages equal to 6, 4 and 3 corresponding to the tapering factor of 2, 3, and 4, respectively. All these tapered buffers are driving a load of 64X inverter in the corresponding technology.

An interesting observation has been made on the results that delay variability increases with the increase of tapering factor and this effect becomes more prominent for smaller technology generations. As discussed before, the delay variability is different for low-to-high and high-to-low transitions for even properly T-sized devices in the inverters (for identical performance in both swings). However, it has been observed that this difference in performance also increases with the increase of tapering factor and becomes worse for smaller technologies at larger tapering factors.

Larger tapering factors are sometimes attractive for power and area efficient designs. However, in the presence of device variability, the designers will have to make a trade-off between these parameters and the amount of tolerable delay

Figure 10. Delay variability of tapered buffer drivers for different tapering factors during high-to-low and low-to-high input transitions



variability (larger β means lesser power and area requirement as compared to smaller β, but greater delay variability).

If n is the stage number in the tapered buffer driver, then its size will be given by

$$w(n)=\beta^{n-1} \tag{7}$$

With the assumption that delay uncertainty introduced by each stage of the tapered buffer driver is independent of each other (true for RDF), the delay uncertainty introduced by the nth stage can then be approximated by

$$\sigma_d\left(n\right)=\frac{\sigma(1w_{\min})}{\sqrt{w(n)}} \tag{8}$$

For optimally sized chain of buffers (according to β) in the tapered buffer driver, the mean value of the delay at n^{th} stage is

$$t_{d_{mean}}\left(n\right)=n\cdot t_{d_{mean}}\left(1w_{\min}\right) \tag{9}$$

By using Equation (8) and (9), the delay variability at the nth stage of the tapered buffer driver can be approximated in first order as

$$V_d\left(n\right)=\left[\frac{3\sqrt{\sigma_{\mathrm{d}}^2\left(1\right)+\sigma_{\mathrm{d}}^2\left(2\right)+\sigma_{\mathrm{d}}^2(3)\ldots\sigma_{\mathrm{d}}^2(\mathrm{n})}}{t_{d_{mean}}\left(n\right)}\right]$$

$$V_d\left(n\right)=\left[\frac{3\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}}\sigma_{\mathrm{d}}^2(\mathrm{i})}}{t_{d_{mean}}\left(n\right)}\right] \tag{10}$$

REPEATERS

Owing to the technology scaling, the interconnect is becoming slower relative to the devices (ITRS, 2005) due to its continuously shrinking dimensions. Therefore, the use of repeaters is very common in long interconnects for reducing the dependence of the interconnect delay on length from quadratic to linear. Although, the insertion of repeaters in the interconnect lines reduces the overall delay, it introduces delay uncertainty in the lines. In the individual interconnect lines; the effect of the delay uncertainty introduced by the repeaters is that the bandwidth will have to be reduced in order to obtain a particular yield. In clock distribution networks, the delay variation due to these repeaters can produce skew across various branches and will limit its performance. This delay variation is particularly unfavorable in wider communication channels because in synchronous links, the speed of the link is limited by the slowest line in the complete channel. Due to statistical variations in the devices, the cumulative delay at the receiving end of the communication channel will be a complete distribution with some nominal value and spread around this mean. Moreover, delay characteristics of the same communication channel on different chips produced in the same batch will not be the same but randomly

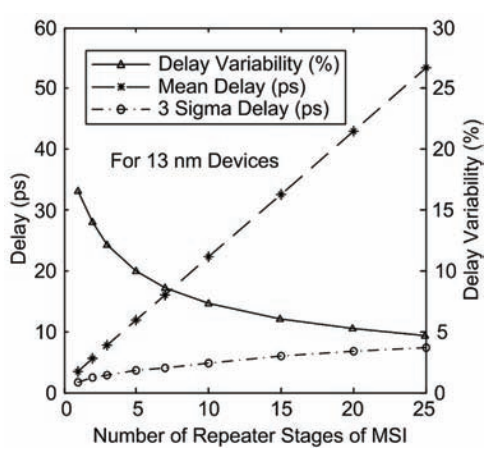
distributed. Therefore, while designing such communication links, the delay characteristics of the repeaters should be known to explore different design options for better performance.

In this study, we have quantified the amount of delay variability in a chain of repeaters of various sizes. Figure 11 shows the results for the repeaters constructed with minimum sized inverters (MSI). It is evident that the mean cumulative delay increases linearly with the increase of the number of repeater stages in the chain. The dispersion (standard deviation) of delay also increases as square root of the number of repeater stages. This is because statistical variations in each repeater stage are independent of each other and can be additive or subtractive towards the cumulative delay. The delay variability on the other hand decreases with the number of repeater stages but at the expense of reduced speed of the repeater line.

If $t_{d_{sec}}$ is the mean delay and $\sigma_{d_{sec}}$ is the standard deviation in the delay for every section of repeated interconnect line, then the cumulative mean delay at the nth stage will be

$$t_{d_{cum}}(n) = nt_{d_{sec}} \tag{11}$$

Figure 11. Delay variability in a chain of minimum sized repeaters of 13 nm plotted against the number of repeater stages



and the standard deviation for cumulative delay at this stage will be

$$\sigma_{d_{cum}}(n) = \sqrt{n}\sigma_{d_{sec}} \tag{12}$$

Therefore, the normalized delay variability at the nth stage will be

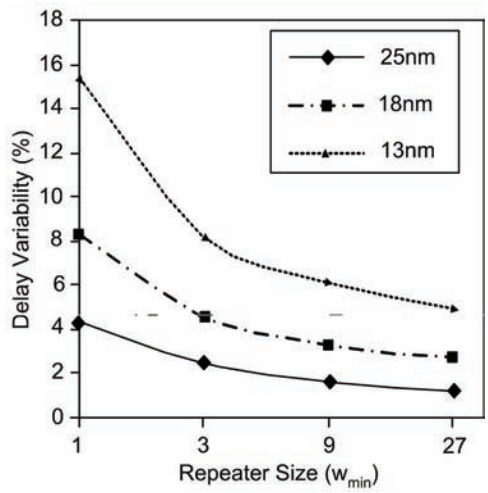
$$V_{d_{cum}} = \frac{3\sigma_{d_{cum}}(n)}{t_{d_{cum}}(n)} = \frac{3\sqrt{n}\sigma_{d_{sec}}}{nt_{d_{sec}}} = \frac{3\sigma_{d_{sec}}}{\sqrt{n}t_{d_{sec}}} \tag{13}$$

As the magnitude of the delay uncertainty introduced by buffers (inverters) depends on their size, a repeater line having large sized repeaters will have less delay variability as compared to the repeater line constructed with small repeaters driving a particular interconnect load. The simulation results shown in Figure 12 endorse this fact. Here cumulative delay uncertainty has been plotted as a function of repeater size for a chain of 20 repeaters. The results demonstrate that a repeater interconnect with large repeaters offers less delay uncertainty as compared to the similar chain constructed with small repeaters. However, a trade-off will have to be made for getting this advantage, as large sized repeaters will consume more power and will take more chip area.

DATA STORAGE REGISTERS (FLIP-FLOPS)

A common technique to enhance the throughput in synchronous digital circuits is the use of Flip-Flops (FFs) and so architectures with deeply pipelined stages are used. Similarly, flip-flops are also used for the storage of different digital signals on the chip, for instance as the last stage of a communication channel. Thus clocked storage elements are essential for a digital circuit. As a result of this tendency, the number of flip-flops

Figure 12. Cumulative delay variability plotted as a function of repeater size in a chain of 20 repeaters



on a chip is growing and therefore FFs represent a significant area of the chip.

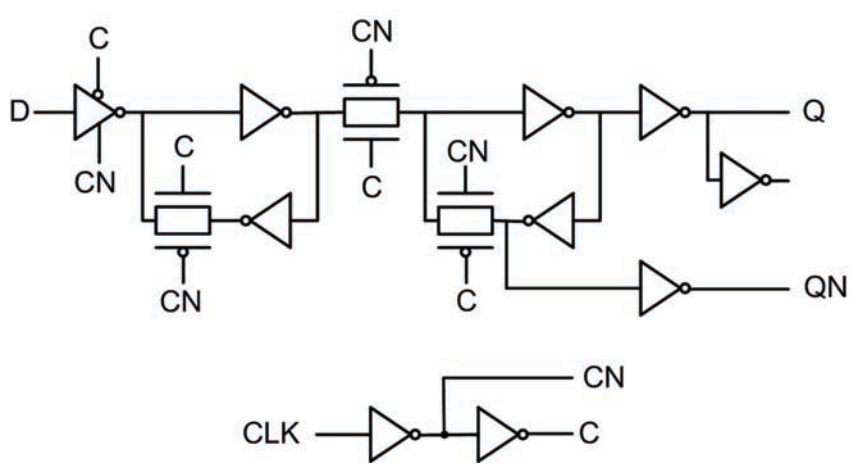
The performance of the circuits incorporating flip-flops as storage elements depends, to a great extent, on the timing characteristics of the flip-flops. However, the statistical variations in the devices constituting these flip-flops affect their timing characteristics. Consequently, the performance of the whole circuit is affected by this variability. Hence it becomes imperative to estimate the timing characteristics of the flip-flops under the impact of statistical variations in order to design high-performance circuits with high yield.

Hence, the effect of device variability on the timing characteristics of a standard CMOS D-flip-flop (DFF) (TSMC, 2001; Lotze, Ortmanns, & Manoli, 2008), as shown in Figure 13, has been studied in this work; in particular the setup and clock-to-Q times have been determined. Since accurate modeling of the flip-flops with the consideration of statistical variation in the devices is difficult, therefore transient analysis of the timing parameters of the DFFs has been performed through HSPICE simulations for accurate results. In digital circuits, a large number of flip-flops are used which are typically constructed with minimum sized devices, in the interest of power and area savings. Therefore, in this work we have also used minimum sized flip-flops for the measurement of their timing parameters.

Timing Measurement Procedure

The procedure adopted for the measurement of different timing parameters of the flip-flops is given below:

Figure 13. A standard D-Flip-Flop used in the simulation



Setup Time

The standard DFF circuit netlists are constructed using random selection of devices from predictive model card libraries. The clock pulse width is made sufficiently large and the data is also kept stable for sufficiently long time after the arrival of the clock signal (to avoid hold time violation). The data is made available at the data input D of the flip-flop quite earlier than the minimum setup time before the arrival of the clock edge. Thus the flip flop safely latches the data at output Q. In the next step, the data-to-clock rising edge time is swept with an increment of 0.2 ps until the flip-flop is unable to hold the data. The minimum data-to-clock rising edge time for which the flip-flop correctly latches the data is the setup time.

CLK-to-Q Time

For the measurement of CLK-to-Q time, the setup time, hold time and clock pulse width were made quite large to avoid any of the timing violations and CLK-to-Q time was measured as the time between the rising edge of the clock and the output Q. These measurements were made for several flip flops constructed with random selection of devices and CLK-to-Q time is measured with an accuracy of 0.2ps.

Results

The results show that the timing parameters of the DFFs are very sensitive to statistical variation in the devices. The mean value of setup time and CLK-to-Q time measured during the simulation are plotted for the given three technology generations and standard deviation in these parameters is also shown in the form of error bars in Figure 14. From the mean value of timing parameters, we can see that the performance of the DFFs improves with technology scaling. However, each of these timings have large amount of variability which increases with the scaling of the technology. For instance, the variability in the setup time of 25 nm DFFs is about 13% and in the 13 nm DFFs reaches to 40%. Similarly, the variability in the CLK-to-Q time is 19% and 56% for the 25 and 13 nm DFFs, respectively. From Figure 15, we can see that setup time, hold time and CLK-to-Q time spans a large timing space (there appears to be no visible correlation between the parameters) and design margins will have to be chosen while keeping in view this space in order to achieve a particular yield.

As the speed of the circuits is increasing with technology scaling, relatively smaller amount of tolerances are now left in the clock period (say). Therefore the timing variability in the future circuits will certainly tend to reduce their performance unless some corrective measures are not taken.

PERFORMANCE OF COMMUNICATION LINKS

The data between any two computational units on a chip is exchanged through interconnects which usually run in a group of wires interspersed with

Figure 14. Timing parameters of DFF measured under device variability due to RDF for the technology generations of 25, 18 and 13 nm

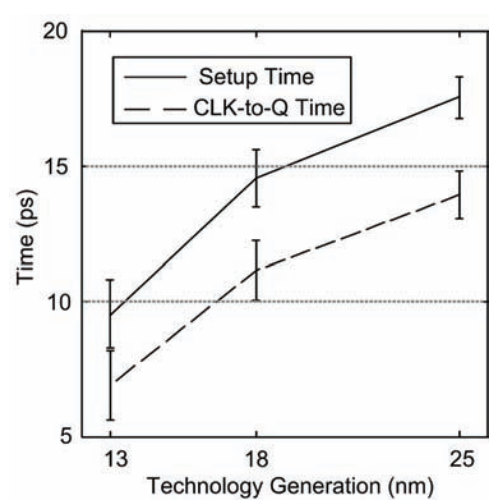
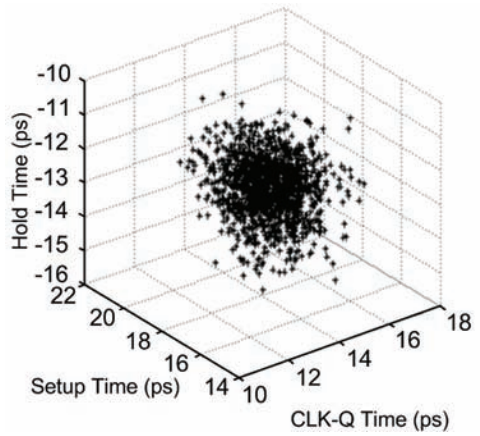


Figure 15. 3D-space occupied by the timing parameters of the DFF



repeaters at uniform distances to improve their delay characteristics. However, statistical variation in the devices introduces delay uncertainty at every communication structure which accumulates in a statistical fashion to produce a significant amount of delay variability at the far end of the interconnect. In a batch of chips fabricated with the same setup and even in the same production run, the delay of a particular interconnect in different chips will have a probability distribution with the mean generally equal to the nominal delay. Due to this reason, there will be occurrences when the delay of an interconnect at the receiver will be larger than the nominal value plus the design margin, causing a link failure. These design margins are neither required to be pessimistic nor optimistic to get maximum achievable performance with adequate reliability level. However, due to delay variability, the throughput of the channel will have to be certainly compromised in order to keep the probability of link failure below a certain acceptable limit. Therefore, variability in the devices constituting the communication structures may contribute significantly towards the performance of clock distribution networks and the data links, which are basically composed of these structures. In order to evaluate the performance of these circuits for future technologies, a detailed modeling of the circuits is required along with the consideration of variability. Alternatively, if the characterization data of the individual communication structures is available with certain accuracy, the performance estimation of these circuits can be made with reasonable accuracy. Based on this approach, a methodology is given below to determine the link failure probability for a single and wider links under the impact of variability.

Interconnect Link Failure Probability

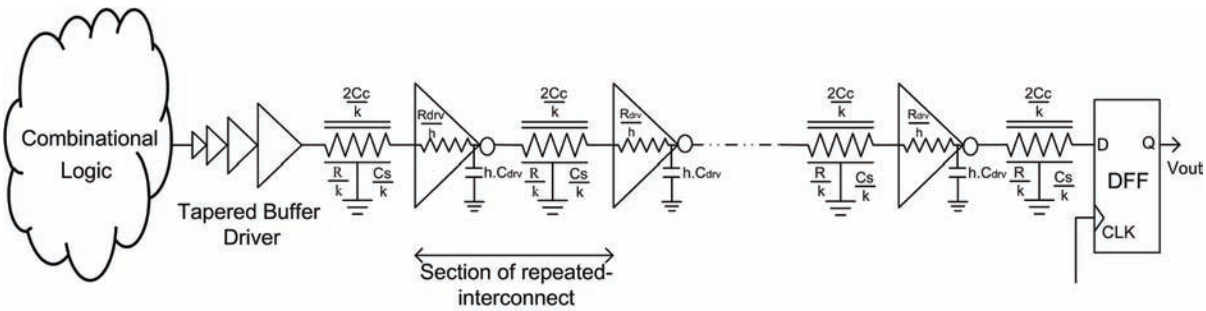
A typical communication link is shown in Figure 16 which consists of tapered buffer drivers, interconnect wires, repeaters and data storage elements. The output of the combinational logic is powered up using tapered buffer driver before transmitting it through the link. The repeaters are used to improve the delay characteristics, especially in predominantly resistive interconnects. Similarly, flip flops or latches are used to hold the data at the receiving end. The bandwidth of the link depends upon the cumulative delay introduced by each of these elements plus the setup time of the flip-flop. The nominal delay of such a data link from input to output can be calculated using the following equation

$$T_{d_{link}} = t_{d_{tap-buff}} + t_{d_{rep-inter}} + t_{CLK-Q_{FF}} \tag{14}$$

In the above expression, $T_{d_{link}}$ is the total delay of the link, $t_{d_{rep_inter}}$ is the repeater-inserted interconnect delay and $t_{CLK-Q_{FF}}$ is the CLK-Q time of the flip-flop.

Let k be the number of repeaters with size of each repeater equal to h times the size of the minimum sized repeater (inverter). In a particular technology, if the output impedance of a minimum sized inverter is R_{drv} and output capacitance is C_{drv}, then the output impedance of a repeater of size h becomes R_{drv}/h and the output capacitance

Figure 16. A typical data communication link



$h*C_{drv}$. In Figure 16, the symbol represents a capacitively coupled interconnect. If we assume that R is the interconnect resistance, C_c is the coupling capacitance with the neighboring interconnects and C_s is the self capacitance of the interconnect, the propagation delay of one section of the repeated interconnect (Pamunuwa & Tenhunen, 2001), which is taken to be the time difference of the input and output waveforms at 50% of the transition points, is given by

$$t_{0.5.sec} = 0.7R_{drv}(C_s + C_{drv} + 2.2 \times 2C_c) + R(0.4C_s + 0.58C_c + 0.7C_{drv})$$

The total delay of the interconnect inserted with repeaters is given by

$$t_{0.5} = k\left[0.7\frac{R_{drv}}{h}\left(\frac{C_s}{k} + hC_{drv} + 2.2\frac{2C_c}{k}\right) + \frac{R}{k}\left(0.4\frac{C_s}{k} + 0.58\frac{C_c}{k} + 0.7hC_{drv}\right)\right]$$

As we have already seen, the repeater inserted interconnect exhibit delay variability due to parameter variations. Similarly, the variability in the performance of other communication structures like the tapered buffer drivers and the data latching registers (FFs) also contribute in the overall delay variability of the interconnect system.

Generally speaking, the delay equation for the complete link under the impact of device and interconnect variability can be written as

$$T_{d_{link}} = \mu_{d_{tap-buff}} + \mu_{d_{rep-inter}} + \mu_{CLK-Q_{FF}} + \sqrt{\sigma^2_{d_{tap-buff}} + \sigma^2_{d_{rep-inter}} + \sigma^2_{CLK-Q_{FF}}} \quad (15)$$

This equation consists of two parts, the mean value and standard deviation of the delay distribution, which can be denoted as

$$\mu_{d_{link}} = \mu_{d_{tap-buff}} + \mu_{d_{rep-inter}} + \mu_{CLK-Q_{FF}} \quad (16)$$

$$\sigma_{d_{link}} = \sqrt{\sigma^2_{d_{tap-buff}} + \sigma^2_{d_{rep-inter}} + \sigma^2_{CLK-Q_{FF}}} \quad (17)$$

Similarly, $\mu_{d_{rep-inter}}$ and $\sigma^2_{d_{rep-inter}}$ are given by

$$\mu_{d_{rep-inter}} = \mu_{sec1} + \mu_{sec2} + \mu_{sec3} + \ldots \mu_{secN} \quad (18)$$

and

$$\sigma^2_{d_{rep-inter}} = \sigma^2_{sec1} + \sigma^2_{sec2} + \sigma^2_{sec3} + \ldots \sigma^2_{N} \quad (19)$$

where μ_{sec} and σ^2_{sec} represents the mean and variance of the delay of each section of the repeater inserted interconnect.

Now if we have complete description of the delay characteristics of the individual communication structures under the impact of device and/ or interconnect variability due to any of their parameters, we can approximate the overall performance of the complete link using Equations (15)-(19). The results can also be used to estimate the probability of the link failure due to variability, as explained below.

Let us assume that the link is operating at a clock frequency f having clock period T_{CLK}. For the flip-flop (having setup time T_{setup}) to correctly latch the data, the delay of the interconnect must satisfy the following constraint:

$$T_{CLK} - T_{setup} \geq t_{d_{tap-buff}} + t_{d_{rep-inter}}$$

Therefore, the probability that correct data is transmitted between the input and output is given by

$$q = \mathrm{P}\left(T_{CLK} - T_{setup} \geq t_{d_{tap-buff}} + t_{d_{rep-inter}}\right) \quad (20)$$

A design margin is also used to cater for the delay variation due to different circuit parameters and let it be Δ_m. Therefore, expression (20) can be written as

$$q = \mathrm{P}(T_{CLK} - \Delta_m - T_{setup} \geq t_{d_{tap-buff}} + t_{d_{rep-inter}}) \quad (21)$$

We define the time delay between the input of the tapered buffer driver and the input D of the receiving flip-flop to be $\mathrm{T}'_{\mathrm{d_{link}}}$. Then the probability that the delay of the link will be greater than $T_{CLK} - \Delta_m$ - T_{setup} is given by

$$P\left[\mathrm{T}'_{\mathrm{d_{link}}} > T_{CLK} - \Delta_m - T_{setup}\right]$$
$$= 1 - P[\mathrm{T}'_{\mathrm{d_{link}}} < T_{CLK} - \Delta_m - T_{setup}]$$

$$= 1 - F\left(\frac{T_{CLK} - \Delta_m - \mu_{setup} - \mu'_{\mathrm{d_{link}}}}{\sqrt{(\sigma'_{\mathrm{d_{link}}})^2 + (\sigma_{setup})^2}}\right) \quad (22)$$

$$= Q\left(\frac{T_{CLK} - \Delta_m - \mu_{setup} - \mu'_{\mathrm{d_{link}}}}{\sqrt{(\sigma'_{\mathrm{d_{link}}})^2 + (\sigma_{setup})^2}}\right) \quad (23)$$

In the above expressions,

$$\mu'_{\mathrm{d_{link}}} = \mu_{d_{tap-buff}} + \mu_{d_{rep-inter}}$$

and $\sigma'_{\mathrm{d_{link}}} = \sqrt{\sigma^2_{d_{tap-buff}} + \sigma^2_{d_{rep-inter}}}$

If we assume that the delay variability in the clock signal is σ_{CLK} with some mean μ_{CLK}, Equation (22) will become

$$P\left[\mathrm{T}'_{\mathrm{d_{link}}} > T_{CLK} - \Delta_m - T_{setup}\right] = Q\left(\frac{\mu_{CLK} - \Delta_m - \mu_{setup} - \mu'_{\mathrm{d_{link}}}}{\sqrt{(\sigma'_{\mathrm{d_{link}}})^2 + (\sigma_{setup})^2 + (\sigma_{\mathrm{CLK}})^2}}\right) \quad (24)$$

Again in expressions (22)-(24), F is the cumulative distribution function of the link delay and Q is the classical error function, respectively. The Borjesson's approximation (Borjesson & Sundberg, 1979), as given below, can be used to evaluate Q.

$$Q(x) \approx \left[\frac{1}{(1-a)x + a\sqrt{x^2 + b}}\right] \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

with $a = 1/\pi$ and $b = 2\pi$.

Table 1. Statistical delay characteristics of different parts of the link

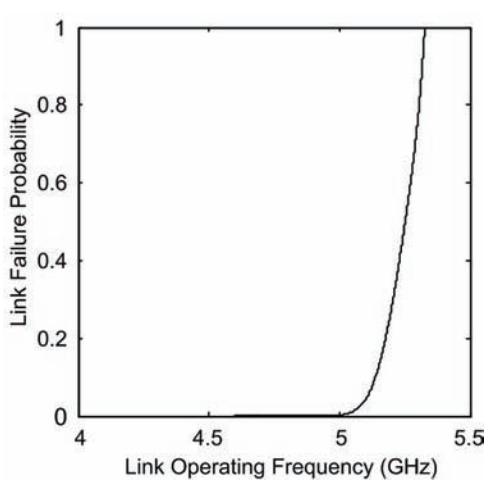
	Tapered Buffer	Repeater Inserted Interconnect	Flip-Flop Setup Time	Flip-Flop CLK-Q Time
Mean, μ	20.95 ps	152.9 ps	14.52 ps	11.12 ps
Std. Deviation, σ	1.11 ps	2.12 ps	1.05 ps	1.09 ps

Case Study

Consider a typical interconnect of length 500 μ*m*, width 0.675 μ*m*, and thickness 0.324 μ*m* in 18 nm technology. The delay characteristics of this interconnect inserted with 10 repeaters of size $5w_{min}$ are given in Table 1 along with the performance characteristics of the tapered buffer driver and flip-flop used in the complete link.

For a design margin Δ_m = 5ps and an uncertainty in the clock period σ_{CLK} = 2ps, the probability of the link failure has been plotted as a function of the operating frequency and is shown in Figure 17. It can be seen that beyond a certain operating frequency, the link failure probability starts increasing from zero. This particular curve corresponds to delay variability due to only RDF in the devices. However in the real case, there are other sources of variability in the devices as well as in the interconnect and therefore overall delay variability in the link will be even larger. Thus for a particular link failure probability, the operating frequency will have to be reduced, otherwise the yield will be reduced for high speed links under tight design margins.

Figure 17. Link failure probability as a function of link operating frequency



Probability of Failure in Wider Links

Due to delay variability in the individual interconnect lines, a wider communication link comprising of multiple interconnects will have a complete distribution of the delay having some mean and standard deviation. This delay distribution is particularly detrimental in high speed links as it introduces skew amongst various lines in a wider data bus. The skew in the delay of a wider link limits the speed of the link as well as increases the probability of the link failure. If P_i is the probability of the individual interconnect failure due to timing violation, then the failure probability of the complete link of width N (no of lines) is given by

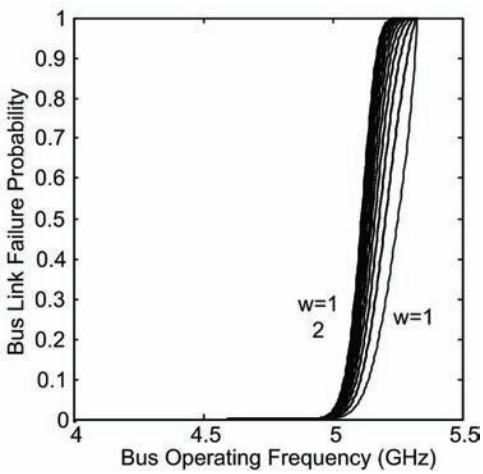
$$P_{bus} = 1 - \prod_{i=1}^{N}(1 - P_i) \tag{25}$$

In the special case where individual interconnect reliabilities are same, $P_i = P$, the link failure probability becomes

$$P_{bus}=1-(1-P)^N \tag{26}$$

The link failure probability for the buses of different widths is plotted as a function of operating frequency of the link and is shown in Figure 18, where the delay characteristics of the individual

Figure 18. Link failure probability for the busses of different widths as a function of operating frequency



interconnects have been taken to be the same as described in the previous section. It is evident that the link failure probability increases with the increase of the number of lines in the bus. Even if all the interconnects do not fail due to timing violation, the delay variability will create significant amount of skew in the bus interconnects at the receiving end which will certainly deteriorate the performance of the buses.

CONCLUSION

Variability is becoming a major constraint in the designing of high performance systems, especially in deep sub-micron technologies. The technology scaling has enabled computation to proceed faster than the communication between different computational blocks on the chip. Although Network-on-Chip has the potential to improve on-chip communication, this study has shown that variability in the communication infrastructures can compromise the ability to meet the designed targets, unless due attention to it is given during the design phase. In particular, we have critically examined the effect of device variability due to RDF on the performance of the basic elements of on-chip communication, such as tapered buffer drivers with different tapering factor, repeaters of different sizes, and data storage registers (FFs). FO4 delay measurements have also been taken, as representative of the logic circuitry and results can be used as a performance benchmark. The study revealed that RDF has significant impact on the performance of communication structures and their performance deteriorates very significantly with technology scaling from 25 to 13 nm. As a design methodology, scaling up of circuits in the critical paths can be employed to minimize the effects of device variability, in particular, since we have shown that this trade-off is not linear and a small increase in the repeater size can give substantial benefit towards the performance. For instance, we have corroborated that large sized repeaters can be used in the interconnect to reduce delay variability, however, the power and area penalties due to this passive technique of circuit scaling should be compared with active countermeasure techniques which can be used to mitigate the delay variability.

Although NoC is more robust against on-chip communication faults than simpler designs, we note that such occurrences have increased hyper-linearly (and will continue to do so) due to device variability. In order to evaluate the performance of a typical NoC link, we have derived analytical models to predict link failure probability (LFP) using the characterization data of the individual on-chip communication elements. The results show that link failure probability increases significantly with the increase of device variability and is a limiting factor in the maximum operating frequency of a synchronous link.

The results also show that link failure probability depends on the width of the communication channel, as skew beyond a certain limit may result in the failure of the link. The LFP increases with the increase of channel width.

REFERENCES

Agarwal, K., Agarwal, M., Sylvester, D., & Blaauw, D. (2006). Statistical interconnect metrics for physical-design optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *25*(7), 1273–1288. doi:10.1109/TCAD.2005.855954

Aikawa, H., et al. (2008). Variability aware modeling and characterization in standard cell in 45 nm CMOS with stress enhancement technique. *VLSI Technology Symposium,* 90-91.

Asenov, A. (2007). Statistical device variability and its impact on yield and performance. In *Proceedings of the 13th IEEE International On-line Testing Symposium.*

Asenov, A., Brown, A. R., Davies, J. H., Kaya, S., & Slavcheva, G. (2003). Simulation of intrinsic parameter fluctuations in decananometre and nanometer scale MOSFETs. *IEEE Transactions on Electron Devices*, *50*(9), 1837–1852. doi:10.1109/TED.2003.815862

Bakoglu, H. B. (1990). *Circuits, Interconnections, and Packaging for VLSI*. Reading, MA: Addison-Wesley.

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

Borjesson, P., & Sundberg, C.-E. (1979). Simple approximations of the error function Q(x) for communications applications. *IEEE transanctions on communications, 27*(3), 639-643.

Borkar, S. (2005). Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, *25*(6), 10–16. doi:10.1109/MM.2005.110

Bowman, K. A., Duvall, S. G., & Meindl, J. D. (2002). Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of Solid-state Circuits*, *37*(2), 183–190. doi:10.1109/4.982424

Cheng, B., Roy, S., Roy, G., Adamu-Lema, F., & Asenov, A. (2005). Impact of intrinsic parameter fluctuations in decanano MOSFETs on yield and functionality of SRAM cells. *Solid-State Electronics*, *49*(5), 740–746. doi:10.1016/j.sse.2004.09.005

Cheng, B. J., Roy, S., Roy, G., Brown, A., & Asenov, A. (2006). Impact of random dopant fluctuation on bulk CMOS 6-T SRAM scaling. In *Proceedings of the Solid-State Device Research Conference (ESSDERC 2006)*.

Cherkauer, B. S., & Friedman, E. G. (1995). Design of tapered buffers with local interconnect capacitance. *IEEE Journal of Solid-state Circuits*, *30*(2), 151–155. doi:10.1109/4.341744

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

Ho, R., Mai, K. W., & Horowitz, M. A. (2001). The future of wires. *Proceedings of the IEEE*, *89*(4), 490–504. doi:10.1109/5.920580

Ishimaru, K. (2007). 45nm/32nm CMOS-Challenge and Perspective. In *Proceedings of the 37th European Solid-State Research Conference (ESSDERC 2007)* (pp. 32-35).

ITRS. (2003). *International Technology Roadmap for Semiconductor*. Semiconductor Industry Association.

ITRS. (2005). *International Technology Roadmap for Semiconductors*. Semiconductor Industry Association.

Keutzer, K., & Orshansky, M. (2002). From blind certainty to informed uncertainty. In *Proceedings of the IEEE International Workshop on Timing Issues in the specification and synthesis of digital systems* (pp. 37-41).

Kim, M., Kim, D., & Sobelman, G. E. (2006). Network-on-Chip link analysis under power and performance constraints. In *Proceedings of ISCAS 2006.*

Lee, K., Lee, S.-J., & Yoo, H.-J. (2004). SILENT: serialized low energy transmission coding for on-chip interconnection networks. In *Proceedings of the IEEE/ACM International conference on Computer Aided Design* (pp. 448-451).

Lin, Z., Spanos, C. J., Milor, L. S., & Lin, Y. T. (1998). Circuit sensitivity to interconnect variation. *IEEE Transactions on Semiconductor Manufacturing*, *11*(4), 557–568. doi:10.1109/66.728552

Liu, J., Zheng, L. R., & Tenhunen, H. (2004). Interconnect intellectual property for Network-on-Chip (NoC). *Journal of Systems Architecture*, *50*(2-3), 65–79. doi:10.1016/j.sysarc.2003.07.003

Lotze, N., Ortmanns, M., & Manoli, Y. (2008). Variability of flip-flop timing at sub-threshold voltages. In *Proceedings of the 13th International Symposium on Low Power Electronics and Design (ISLPED 2008).*

Mondal, M., Ragheb, T., Wu, X., Aziz, A., & Massoud, Y. (2007). Provisioning On-Chip networks under buffered RC interconnect delay variations. In *Proceedings of the ISQED'07.*

Moore, G. E. (1965). Cramming More Components onto Integrated Circuits. *Electronics, 38*(8).

Narasimhan, A., & Sridhar, R. (2007). Impact of variabity on clock skew in H-tree clock networks. In *Proceedings of the 8th International Symposium on Quality Electronic Design.*

Nassif, S. (2000). Delay variability: Sources, Impact and Trends. In. *Proceedings of the International Solid State Circuits Conference*, *2000*, 368–369.

Nemes, M. (1984). Driving large capacitances in MOS LSI systems. *IEEE Journal of Solid-state Circuits*, *19*(1), 159–161. doi:10.1109/JSSC.1984.1052103

Pamunuwa, D., & Tenhunen, H. (2001). Repeater insertion to minimise delay in coupled interconnects. In *Proceedings of the 14th International Conference on VLSI Design.*

TSMC. (2001). *TSMC 0.18μm process 1.8 volt in SAGE-X™ standard cell library*. Artisan components, Inc.

Vishvanathan, V., Ravikumar, C. P., & Menezes, V. (2005). Design technology challenges in the Sub-100 nanometer era. *Periodical of the VLSI society of India-VLSI Vision, 1*(1).

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 4, edited by Seppo Virtanen, pp. 40-62, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 11
Schedulability Analysis for Real Time On-Chip Communication with Wormhole Switching

Zheng Shi
University of York, UK

Alan Burns
University of York, UK

Leandro Soares Indrusiak
University of York, UK

ABSTRACT

In this paper, the authors discuss a real-time on-chip communication service with a priority-based wormhole switching policy. The authors present a novel off-line schedulability analysis approach, worst case network latency analysis. By evaluating diverse inter-relationships and service attributes among the traffic flows, this approach can predict the packet network latency for all practical situations. The simulation results provide evidence that communication latency calculated using the real time analysis approach is safe, closely matching the figures obtained from simulation.

INTRODUCTION

On-chip multiprocessing has reached mainstream in server, desktop and laptop computers, and is being increasingly applied to embedded systems. As 45nm and 32nm process technologies mature, we approach the mark of 2 billion transistors on a single silicon die, allowing for the on-chip integration of dozens of processing cores. This is an increase by one order of magnitude, if compared with processors that are commercially available. As the number of cores per chip increases, on-chip communication plays an increasingly important role on system performance, power dissipation, chip area, design effort and total cost. Since early solutions based on point-to-point interconnects or on-chip busses don't scale and quickly become a communication bottleneck (Furber & Bainbridge, 2005; Henkel, Wolf, & Chakradhar, 2004), Networks-on-Chip (NoCs) have emerged

DOI: 10.4018/978-1-4666-0912-9.ch011

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

as a suitable architectural template for on-chip communication (Benini & De Micheli, 2002; Dally & Towles, 2001). Such template offers a generic communication platform which can be parameterised and reused for a large number of SoC designs.

As an integration platform, the NoC should have the capability to provide different level of services for various applications on the same network (Benini & De Micheli, 2001). Some applications have very stringent communication service requirements; the correctness relies on not only the communication result but also the completion time bound. A data packet received by a destination too late could be useless. These critical communications are called real-time communications. For a packet transmitted over the network, this time bound is denoted by the packet network latency. The worst case acceptable time metric is defined to be the deadline of the packet. A traffic-flow is a packet stream which traverses the same route from the source to the destination and requires the same grade of service along the path. For hard real-time traffic-flows, it is necessary that all the packets generated by the traffic-flow must be delivered before their deadlines even under worst case scenarios. A set of real-time traffic flows over the network are termed schedulable if all the packets belonging to these traffic flows meet their deadlines under any arrival order of the packet set.

As a popular switching control technique, wormhole switching (Ni & McKinley, 1993) has been widely applied for on-chip networks due to its high throughput and smaller buffering requirements (Kavaldjiev & Smit, 2003). However, the analysis of the real-time packet schedulability for wormhole switching networks is still an open problem. Predictable behaviour of the network services is essential to support real time requirements – particularly satisfying deadline bounds. But the situation for on-chip wormhole networks is partially non-deterministic due to the contentions in communication. In on-chip networks, several tasks running on different nodes exchange information periodically. During a transmission period, each transmitted packet shares the resources, such as buffers or physical links, with other packets. When several packets try to access the same resource at the same time, contention occurs and the network can only serve one packet and must suspend the others based on some arbitration policy. Once a packet becomes blocked, it can block other packets, which can in turn block other packets, and so on. The exact analysis of congestion in this situation is hard due to the possibility of a packet becoming blocked at several routers during its journey from source to destination. The contention problem leads to packet delays and even missed deadlines. Therefore, it is necessary to give an arbitration strategy and analysis approach to predict whether all the real-time packets can meet their timing requirements.

In this paper we explore real-time communication in wormhole switching for on-chip networks. Utilizing the priority based preemption arbitration (Balakrishnan & Ozguner, 1998; Hary & Ozguner, 1997; Kim, Kim, Hong, & Lee, 1998; Lu, Jantsch, & Sander, 2005), a novel real-time schedulability analysis approach is presented which successfully emulate wormhole switching as classic single processor scheduling model (Liu & Layland, 1973). By evaluating diverse inter-relationships and service attributes among the traffic-flows, this approach can predict the packet transmission latency for a given traffic-flow based on three quantifiable different interferences: direct interference from higher priority traffic-flows, indirect interference from higher priority traffic-flows and self-blocking produced by the previous packet from same traffic-flow. To further validate our approach, we performed extensive simulations aiming to demonstrate that the analysis is always safe, and to show situations when it is tight or not.

The rest of this paper is organized as follows: The second and third sections cover the major features of NoC architectures based on wormhole

switching and introduce the associated real-time communication model. A preemptive arbitration structure is deployed to implement priority-driven transmission scheduling. The fourth and fifth sections present a novel off-line schedulability analysis approach. The sixth section evaluates the presented analysis by comparing it with experiments based on simulation. Finally, the last section concludes the paper.

PRIORITY BASED WORMHOLE SWITCHING

The design space of a NoC architecture is very large, as many of its features can be parameterised to better meet design goals. Topology, routing algorithm, flow control, arbitration, buffering and switching strategies are among of these features. This paper focuses on a smaller portion of the design space, as it limits itself to NoCs based on wormhole switching. Currently, wormhole switching has been widely used by many NoC implementations such as Æthereal (Goossens, Dielissen, & Radulescu, 2005) and HERMES (Moraes, Calazans, Mello, Möller, & Ost, 2004). However, wormhole switching is particularly vulnerable to network contention and its effects are hard to predict. In order to increase the predictability of such architectures, we further constraint our design space and focus on NoC architectures that use priority-based arbitration and deterministic routing. The following subsections provide further details on the rationale behind our decisions, and provide insights on how such decisions allowed us to make NoC architectures more amenable to real-time analysis.

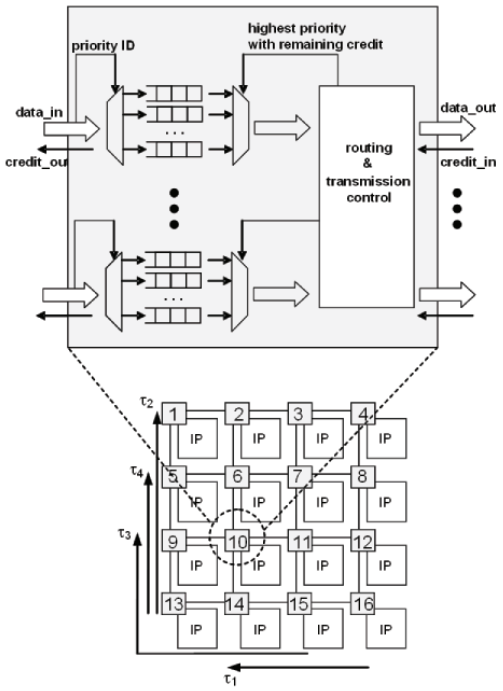
Wormhole Switching

Early multi-hop networks normally used store-and- forward packet switching (Duato, Yalamanchili, & Ni, 2002). In this approach, the entire packet is stored in a buffer when a packet reaches an intermediate node. But this approach inevitably brings larger size buffer and longer packet latency. To address the problems, wormhole switching (Ni & McKinley, 1993) is proposed which utilize the cut-through switching technique that a data packet arriving at an intermediate node is immediately forwarded to the next node without buffering if the next output channel is available. The implementation mechanism is that each packet in a wormhole network is divided into a number of fixed size flits. The header flit contains the routing information and governs the route. As the header advances along the specified path, the remaining flits follow in a pipeline way. If the header flit encounters a link already in use, it is blocked until the link becomes available. In this situation, all the flits of the packet will be buffered by the routers along the path, so only a small number of flits must be buffered by each router.

A typical implementation of a wormhole-based on-chip network is illustrated in Figure 1. A NoC consists of multiple *intellectual property* (IP) blocks and routers that are connected using links as a grid so that they are able to communicate with each other by sending and receiving messages. The IPs which undertake the computational data process could be arbitrary, typical examples are embedded processors, dedicated hardware devices, or reconfigurable fabrics. Different IPs do not directly connect each other; the communication between IPs has to go through a set of routers. The router nodes deliver the data from source to its destination according to the chosen switching protocols and routing policies. The links connect the routers, providing the raw bandwidth by physical channels. All these components work together as a network basis to provide the data service for the applications.

Figure 1 also illustrates a number of traffic flows loaded on this NoC platform. For example, τ_3 starts in router 15 and passes through routers 14 and 13 before terminating in router 9. We shall return to this example later in the paper.

Figure 1. NoC architecture (with detail of the router structure) and traffic flow examples



Priority Preemptive Arbitration

In conventional wormhole routers, data flits held by routers access the output link based on first-come, first-service or round-robin discipline. These schemes are suitable for non-real-time networks since they are fair and produce good average performance. But in real-time communication, the network must ensure each real-time packet meets its deadline bound. Priority arbitration is proposed to resolve this problem (Balakrishnan & Ozguner, 1998; Mutka, 1994). Within this work, we will consider a flit level pre-emption method, which has similar structure as in (Hary & Ozguner, 1997; Kim et al., 1998).

Such scheme can be implemented using a flow control mechanism that supports virtual channels (VCs) (Dally, 1992). Such mechanism multiplexes every link among different communication flows by using VCs that are assigned a priority level. Likewise, every communication flow is assigned a priority level (which is inherited by all packets of that flow), and that level defines which VC it should use. Thus, a packet with priority *i* can only request the VCs associated with priority value *i*. This one-to-one relation can ensure that any VC is requested by no more than one traffic flow.

Figure 1 shows the internal structure of such a NoC node. In each input port, a different FIFO buffer stores flits of packets arriving through different virtual channels (i.e., different priorities). The router assigns an output port for each incoming packet according to their destination node. We use a credit-based approach (Bjerregaard & Sparso, 2004) to guarantee that data is only forwarded from a router to the next when there's enough buffer space to hold it. At any time, a flit of a given packet will be sent through its respective output port if it has the highest priority among the packets being sent out through that port, and if it has credits (that is, buffer space on the respective buffer of the neighbouring node connected to that output port). If the highest priority packet can't send data because it is blocked elsewhere in the network, the next highest priority packet can access the output link. In addition, a higher priority packet can also pre-empt a lower priority packet during its transmission. Therefore, the allowable service time for a traffic flow comprehends all the time intervals at which no higher priority traffic flow competes for the same physical link.

Related Works

Wormhole switching achieves high throughput with less buffer requirement, when compared to packet-switching (Kavaldjiev & Smit, 2003). But the major difficult is how to analyze and validate its correctness which is greatly desired by hard real time communication services.

The first work to explore the packet's timing property in wormhole switching was published by Mutka (1994) in 1994. Utilizing rate monotonic scheduling algorithm (Liu & Layland, 1973). Mutka's scheme ensures that all the flows shared

the same link will meet their deadlines if the link utilization is no more than a specific upper bound. Balakrishnan and Ozguner (1998) proposed a lumped link model analysis technique to address the schedulability. In the lumped link model, all the links the traffic-flow travels are lumped as one shared resource and the interference due to all higher priority flows using one or more of the links along the path of the flow is calculated. However, due to lumping, direct and indirect contentions are treated in the same way; Balakrishnan's result is sufficient but pessimistic. Hary and Ozguner (1997) utilized the same model proposed in (Balakrishnan & Ozguner, 1998) but ignored indirect competition, the result is hence optimistic. In this paper we treat the indirect contentions as interference jitter and get a tight upper bound on network latency. In (Kim et al., 1998), Kim used a blocking dependency graph to express the contentions a flow may meet and derived the packet delivery upper bound. The analysis by Lu et al. (2005) utilized a contention tree to take account of the parallel communication in disjoint traffic-flows and tried to improve the accuracy of analysis. But this parallel consideration is not reasonable when worst case network latency is desired.

REAL TIME COMMUNICATION MODEL AND ASSUMPTIONS

System Model and Attributes

A wormhole switching real-time network Γ comprises n real-time traffic-flows $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. Each traffic-flow τ_i is characterized by attributes $\tau_i = \left(P_i, C_i, T_i, D_i, J_i^R\right)$. We assume that all the traffic-flows which require timely delivery are periodic or sporadic. The lower bound interval on the time between releases of successive packets is called the period T_i for the traffic-flow τ_i. Each traffic-flow τ_i has a distinct priority value P_i. All the packets that belong to τ_i inherits the same priority P_i. The value 1 denotes the highest priority and larger integers denote lower priorities. We assume the traffic-flow is prioritized by any possible global priority assignment policy. The issue of priority assignment is beyond the scope of this paper. Each real-time traffic-flow has deadline D_i constraint which means all the packets belonging to this traffic-flow have the restriction that it should be delivered from a source router to a destination router within a certain delay bound even in the worst case situation. The analysis in section 4 has the same restriction as (Balakrishnan & Ozguner, 1998; Kim et al., 1998; Lu et al., 2005) that each traffic-flow's deadline must be less than or equal to its period, $D_i \leq T_i$ for all $\tau_i \in \Gamma$. In section 5, this restriction will be removed. The attribute J_i^R is the release jitter (Audsley, Burns, Richardson, Tindell, & Wellings, 1993) and denotes the maximum deviation of successive packets released from its period. If a packet from τ_i is generated at time a, then it will be released for transmission by time a + J_i^R and have an absolute deadline of a + D_i. Consequently, the restriction between deadline and period when the release jitter exists is also updated with $D_i \leq T_i - J_i^R$.

The *basic network latency* occurs when no traffic-flow contention exists. The basic network latency is determined by its source/destination routing distance, packet size and link bandwidth. For real-time evaluation purpose, we most determine the *maximum basic network latency C*. Let H denote the hops between source and destination nodes and S indicate the constant processing delay in each router. Let the maximum packet size belonging to τ_i be $L_i^{\max}$ and the flit size for wormhole switching be f_{size}, the maximum basic network latency C_i is given by:

$$C_i = \left\lceil \frac{L_i^{\max}}{f_{size}} \right\rceil \cdot \frac{f_{size}}{B_{link}} + H \cdot S \quad (1)$$

where B_{link} is link bandwidth. The second term in the equation counts the routing delay experienced by the header flit. Once the header flit reaches the destination, the transmission time of this pipelined packet is determined by the packet size and network bandwidth which is accounted by the first term. Here we only consider the longest possible, or maximum, basic network latency for evaluation. If the real-time traffic-flow can meet its deadline with the maximum basic network latency, it will meet the deadline for any other basic network latency scenario.

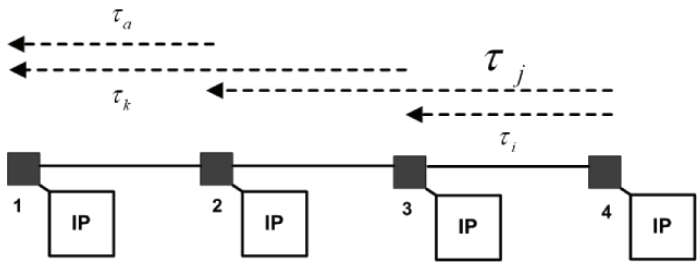
Inter-Relationship Between Traffic-Flows

To capture the relations between traffic-flows and the physical links of the network, we formalize the mesh network topology as a directed graph G: $V \times E$. V is a set, whose elements are called nodes, each node v_i denotes one router in the mesh network. E is a set of ordered pairs of vertices, called edges. An edge $e_{x,y} = \{v_x \rightarrow v_y\}$ is considered to be a real physical link from router v_x to router v_y; v_x is called the source and v_y is called the destination of the link. We define a mapping space from the traffic-flow set to the physical links $\Gamma \rightarrow E$. Given a set of n traffic-flows, we can map them to the target network. The routing β_i of each traffic-flows τ_i is denoted by the ordered pairs of edges, $\beta_i = \{e_{a,b}, e_{b,c}, ..., e_{y,z}\}$. If a traffic-flow τ_i shares at least one link with τ_j, the intersection set for τ_i and τ_j is computed $\beta_i \cap \beta_j$. If $\beta_i \cap \beta_j = \varphi$, τ_i and τ_j are disjoint (share no physical link).

During the journey of packet transmission, the contention interference from higher priority flow will heavily delay the current packet. Based on whether they share the same physical links, we categorize the competition relationship between the traffic-flows into two different types: *direct competing relationship*, and *indirect competing relationship*. The direct competing relationship means a traffic-flow has at least one physical link in common with the other one. When two traffic-flows τ_i and τ_j have a direct competing relationship, $\beta_i \cap \beta_j \neq \varphi$ and meet the condition $P_j > P_i$, τ_j will force a direct interference with the observed traffic-flow τ_i. For flow τ_i, we define a *direct interference set* S_i^D which includes all the traffic-flows meeting the above conditions, $S_i^D = \{\tau_j \mid \beta_j \cap \beta_i \neq \varphi$ and $P_j > P_i$ for all $\tau_j \in \Gamma\}$.

With the indirect competing relationship, by comparison, the two traffic-flows do not share any physical link but there is (are) intervening traffic-flow(s) between the given two traffic-flows. For example, if there are three flows τ_i, τ_j, τ_k meeting the following situation $\beta_k \cap \beta_i = \varphi$, $\beta_k \cap \beta_j \neq \varphi$ and $\beta_i \cap \beta_j \neq \varphi$, τ_j is the intervening flow in this case, then there is an indirect competing relationship between τ_i and τ_k. Notice that the indirect competing has the transitivity. If there more than one intermediate flow exists, the indirect competing relationship still holds. Following the above case, if there is a new flow τ_a which has an indirect competing relationship with τ_j (τ_k is intermediate flow between τ_a and τ_j), and τ_a does not share any physical link with τ_i, then there is an indirect competing relationship between τ_a and τ_i (both τ_k and τ_j are intermediate flows). Figure 2 shows the situation. When two flows τ_i and τ_k have indirect competing relationship and meet the condition $P_k > P_j > P_i$, τ_j is the intervening flow, τ_i may suffer an indirect interference from τ_k even when they do not share any physical link. For flow τ_i, we define an *indirect interference set* S_i^I which includes all the traffic-flows meeting the above conditions, $S_i^I = \{\tau_k \mid \tau_k$ has indirect competing relationship

Figure 2. Transitivity in indirect competing relationship



with τ_i and $P_k > P_j > P_i$ for all $\tau_k \in \Gamma$, where τ_j is any intermediate flow}.

Return to the example in Figure 1. The four prioritized traffic-flows sorted from high to low priority are τ_1, τ_2, τ_3 and τ_4. Flows τ_1, τ_2 have no shared links with any other higher priority traffic-flow so no direct or indirect interference, $S_1^I = S_1^D = S_2^I = S_2^D = \varphi$. Flow τ_3 competes with τ_1 and τ_2 and gets $S_3^I = \varphi$ and $S_3^D = \{\tau_1, \tau_2\}$. Flow τ_4 directly contends with τ_2 and τ_3 and indirectly competes with τ_1, $S_4^D = \{\tau_2, \tau_3\}$ and $S_4^I = \{\tau_1\}$.

With this definition of the communication model, we will derive, in the next section, an upper bound on the network latency of real-time traffic-flows.

NETWORK LATENCY UPPER BOUND ANALYSIS

In priority based wormhole switching, the packet advances when it has the highest priority and remaining buffer along its path. So the whole links used by a traffic-flow are treated as one resource and other traffic-flows using one or more links along the path of the observed flow produce contention interferences. To determine the upper bound of network latency for a real-time traffic-flow, the maximum basic network latency and maximum contention interference need to be calculated. The maximum basic network latency can be calculated by static analysis of the traffic-flow pattern and the network features (Equation (1)). Another important factor dominating the latency upper bound is interference. In this section, we assume that the analysis is based on the single packet instance, which means for a traffic-flow, before the current packet instance is accepted by the destination, the successive packet can not be generated from its source node. In another word, the restriction condition $D < T - J^R$ is always true. So under the fixed priority arbitration discipline and single packet instance assumption, only traffic flows with higher priority than the current one can cause interference. In order to evaluate these possible competing interferences, we should find all the higher priority traffic flows which could affect the observed one and calculate the interference upper bound by analyzing the characteristics of these higher priority traffic-flows. Also, we notice that there is a resource competition relationship between the observed flow and its relevant higher priority flows. This is similar to the processor resource model (Audsley et al., 1993; Tindell, Burns, & Wellings, 1994) in real time scheduling, in which all the tasks contends for a single shared processor resource. In wormhole switching networks, the shared physical communication links are also contended for by the associated traffic-flows. Utilizing the priority preemption, we can analyze this model following the real-time scheduling approach for single-processors.

The analysis algorithm is based on the computation of the worst case network latency R for

each traffic-flow. If $R_i \leq D_i$, the analysis ensures that flow τ_i always meets the time boundary. Generally, we need to find the condition which can trigger the traffic-flow's worst case network latency. Inspired by Liu and Layland's seminal paper (Liu & Layland, 1973), two major conditions to achieve the worst case latency are identified:

Condition 1. *All the higher priority traffic flows release packets at their maximum rate with maximal packet size.*

Condition 2. *The packet from the observed traffic flow is released simultaneously with all the direct higher priority packets (referred to as the critical instant).*

Taking these two conditions into account, we will quantify the analysis based on two distinguishing interferences:

- Direct interference from higher priority traffic-flows.
- Indirect interference from higher priority traffic-flows.

This section covers the network latency analysis of a traffic-flow assuming only direct higher priority traffic-flow exists. The worst case latency occurs when two critical conditions are satisfied.

Theorem 1. *The maximum network latency R_i* for traffic flow τ_i can be found by:

$$R_i = \sum_{\forall \tau_j \in S_i^D} \left\lceil \frac{R_i + J_j^R}{T_j} \right\rceil C_j + C_i \tag{2}$$

when only direct higher priority traffic-flows exist.

Proof: For an observed traffic-flow τ_i, the network latency R_i of a packet released from τ_i is addressed as:

$$R_i = C_i + I_i \tag{3}$$

where I_i is the interference summation from the higher priority traffic-flow(s). The maximum basic network latency C_i is constant and known in advance by static analysis. Supposing that we can find an upper bound of the total network latency R_i, the maximum number of instances released from any higher priority flow τ_j is no more than:

$$\left\lceil \frac{R_i + J_j^R}{T_j} \right\rceil$$

assuming the worst case release scenario of τ_j: the first packet release starts J_j^R later than the normal arrival, and the subsequent releases are most packet size with the maximum release rate of $1/T_j$. These higher priority packets will immediately preempt τ_i's packet transmission and delay the latter. Consequently, the interference summation from all the flows in S_i^D is thus:

$$I_i = \sum_{\forall \tau_j \in S_i^D} \left\lceil \frac{R_i + J_j^R}{T_j} \right\rceil C_j \tag{4}$$

Using Equation(4) to substitute I_i in Equation(3), we can produce:

$$R_i = \sum_{\forall \tau_j \in S_i^D} \left\lceil \frac{R_i + J_j^R}{T_j} \right\rceil C_j + C_i$$

We note the variable R_i appears on both sides of the Equation(2). This equation can be solve using an iterative technique (Audsley et al., 1993). Let R_i^{n+1} be the (n+1)th iterative value generated from the equation:

$$R_i^{n+1} = \sum_{\forall \tau_j \in S_i^D} \left\lceil \frac{R_i^n + J_j^R}{T_j} \right\rceil C_j + C_i$$

The iteration starts with $R_i^0 = C_i$ and terminates when $R_i^{n+1} = R_i^n$. This iteration also should halt if $R_i^{n+1} > D_i$, which denotes the deadline miss for this packet. By this iterative technique, the worst case network latency with direct interference can be calculated ($R_i = R_i^{n+1} = R_i^n$).

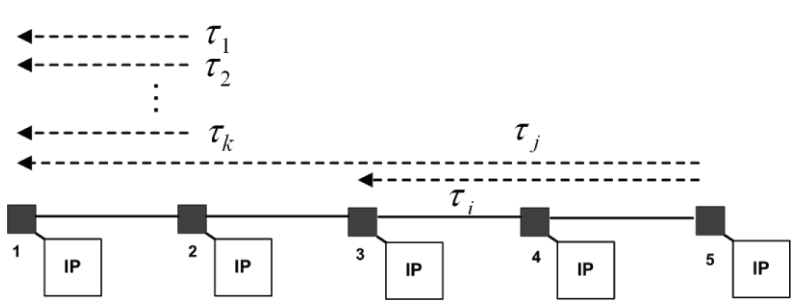
Indirect Interference from Higher Priority Traffic-Flow

The model indicated by Equation (2) only considers the direct interference from higher priority traffic-flow since their activities directly delay the completion time of the observed flow. However, when indirect higher priority traffic-flow exits, some extra interference also needs to be taken into account.

Consider the following scenario: $\tau_1, \tau_2, ..., \tau_k, \tau_j$ and τ_i are loaded on the network with the inter-relations of traffic-flows illustrated in Figure 3. Traffic-flows are sorted with descending priority, τ_1 with highest priority, τ_i with lowest priority. Here we follow the old fashion (reverse alphabetical order) (Audsley et al., 1993; Tindell et al., 1994) that τ_i as the observed flow has the lowest priority and τ_j, τ_k get higher priority than τ_i, $P_k > P_j > P_i$. We examine the competing relation of τ_i and τ_j and get $S_i^D = \{\tau_j\}$ and $S_i^I = \{\tau_1, ..., \tau_k\}$ and $S_j^D = \{\tau_1, ..., \tau_k\}$. Without considering the possible indirect interference from $\tau_1, \tau_2, ..., \tau_k$, the worst case network latency of τ_i occurs when τ_j and τ_i are released at the same time.

When $\tau_1, \tau_2, ..., \tau_k$ as indirect higher priority traffic-flows are taken into account, even though they do not share any physical link with τ_i, we find their services still can impose an extra interference on τ_i. The time-line graph in Figure 4 shows such a situation. The solid up arrow in the graph indicates the release time of a packet from a traffic-flow. The bold circle denotes the time when the complete packet is received at the destination. We assume τ_j is released with the other higher priority packets at the same time 0, the packets from $\tau_1, \tau_2, ..., \tau_k$ will contend with τ_j. This contention delays the start time of τ_j until time t_i, t_i is the start time of τ_j first transmission service. At the time $0+t_i$, τ_i is released, the packet from τ_j immediately preempts τ_i. It is easy to find that τ_j imposes the interference C_j upon the τ_i during the time interval $[t_i, R_j]$. At the time $0+T_j$, τ_j is released again but this time all the higher priority traffic-flows $\tau_1, \tau_2, ..., \tau_k$ only send a very small packet or even do not send any packet. Flow τ_j in this situation does not suffer any interference from them and gets network service immediately. From the view of τ_i, the time interval between two successive releases

Figure 3. A case of indirect interference



from τ_j is only (T_j- t_i). Consider our original assumption, the worst case network latency occurs when all the packets from the higher priority traffic-flows are released periodically with their maximum packet sizes and the minimum packet release interval T. The maximum interference a packet from τ_i suffered from a higher priority traffic-flow is calculated by $\left\lceil \frac{R_i + J_j^R}{T_j} \right\rceil C_j$. But in this case, the minimum interval between subsequent preemptions from τ_j is only (T_j- t_i) which is less than the original minimum interval assumption T_j. Note that this phenomenon can only occur when considering indirect interferences.

Theorem 2. *The upper bound of interference suffered by* τ_i from direct higher priority traffic-flow τ_j is:

$$\left\lceil \frac{R_i + R_j - C_j + J_j^R}{T_j} \right\rceil C_j \tag{5}$$

when the indirect interference is considered.

Proof: Let s_j denote the packet release time from τ_j. In this analysis assumption, without loss of generality, the first packet is released at time 0. Therefore, each packet from τ_j is generated periodically at the time instant 0, T_j, $2T_j$,…, qT_j. Let $s_{j,q} = (q-1)\,T_j$, where q is the sequential number of the packets. But a real application does not always meet this constraint and has application release jitter. More specifically, the release time $s_{j,q}$ satisfies:

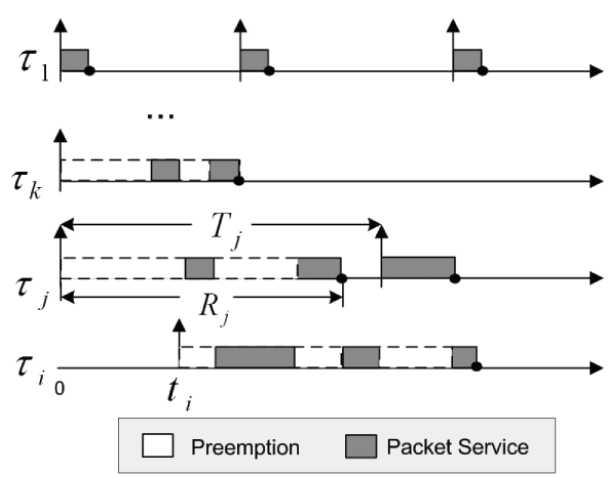
$$(q-1)T_j \le s_{j,q} \le (q-1)T_j + J_j^R$$

In addition, in Figure 4, we observe that the possible interference from higher priority packets also defers its starting transmission time. If the worst case network latency for τ_j is R_j, the upper bound of the start transmission time is R_j - C_j. The real start transmission time for each packet satisfies:

$$(q-1)T_j \le s_{j,q} \le (q-1)T_j + J_j^R + R_j - C_j$$

Now we evaluate the maximum interference suffered by τ_i from τ_j in a given time interval. Here we assume the start transmission time of τ_j is $a = R_j - C_j + J_j^R$, this is the upper bound of the start transmission time since it is released. A

Figure 4. The problem of indirect interference



packet from τ_i is released simultaneously with τ_j and b is the corresponding completion time of this release. The worst case interference occurs when the packets from τ_j are released at their maximum frequency since τ_i is released. Figure 5 illustrates this situation:

The number of preemptions by τ_j is given by the positive integer number g within the interval $[a, b]$, $g \in N$. The last release of τ_j should fall into the interval before the completion of τ_i, therefore g is the largest value that satisfies:

$$a - (R_j - C_j) - J_j^R + (g-1)T_j < b$$

or, equivalently,

$$g < \frac{J_j^R + R_j - C_j + b - a}{T_j} + 1$$

The largest positive integer number satisfying this inequality is given by

$$g = \left\lceil \frac{J_j^R + R_j - C_j + b - a}{T_j} \right\rceil$$

The interval $[a, b]$ marks the worst case network latency of τ_i, $b - a = R_i$. Therefore, the interference upper bound from τ_j is:

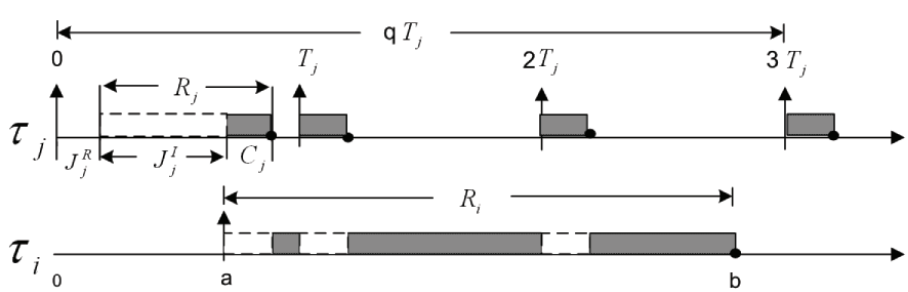
$$\left\lceil \frac{R_i + R_j - C_j + J_j^R}{T_j} \right\rceil C_j$$

The packet from τ_i will experience worse latency than what is predicted by Equation(2) on account of the indirect interference from $\tau_1, \tau_2, \ldots, \tau_k$ which delays τ_j and may cause more hits on τ_i. Therefore, the worst case network latency does not occur when the packet is released simultaneously with all the higher priority packets but at the point when the packet from the observed traffic-flow is released at the same time as all the direct higher priority packets finish waiting and start transmission. So we modify our critical instant (Condition 2) with:

Condition 3. *The critical instant for each flow happens at the time that the packet from the observed traffic-flow is released exactly when all the direct higher priority packets start transmission.*

Actually, different traffic flows in NoC may share different sets of physical links according to their routes and hence has its own direct and indirect higher priority flows. So each flow could has its own critical instant. It is unlikely that the simultaneous release (time 0) will be the critical instant for all the traffic-flows. In Section 5, by a practical application simulation, we will show

Figure 5. Upper bound analysis of indirect interference



enough evidence that the critical instant for all the flows is not necessarily same.

The deviation induced by indirect interference from higher priority flows between consecutive releases is called interference jitter, using symbol J^I to denote it. The interference jitter of a traffic-flow is the maximum deviation between two successive packet start transmission times which can be obtained by computing the difference between its maximum and minimum value. Consider the situation that no higher priority packet is sent in a period, the minimum packet start transmission time becomes zero. Accordingly, the upper bound of interference jitter for a traffic-flow is given by

$$J_j^I < R_j - C_j$$

Note that not all the traffic-flows suffer interference jitter, this only happens when the observed traffic-flow τ_i has indirect interference, namely, J_j^I exists if and only if $S_j^D \cap S_i^I \neq \phi$.

Theorem 3. *The network latency upper bound* R_i for traffic-flow τ_i can be found by:

$$R_i = \sum_{\forall \tau_j \in S_i^D} \left\lceil \frac{R_i + J_j^R + J_j^I}{T_j} \right\rceil C_j + C_i \tag{6}$$

when both direct and indirect higher priority traffic-flows exist.

Proof: I_i is the maximum interference from flows whose priorities are higher than the current flow τ_i. I_i can be presented by a summation of all *I(j)*; where *I(j)* is the maximum interference from τ_j, $\tau_j \in S_i^D$:

$$I_i = \sum_{\tau_j \in S_i^D} I(j)$$

According to Theorem 2, the maximum amount of interference from all flows that have higher priority than τ_i is given by:

$$I_i = \sum_{\tau_j \in S_i^d} \left\lceil \frac{R_i + R_j - C_j + J_j^R}{T_j} \right\rceil C_j$$

As a result, the worst case network latency in case of interference jitter and release jitter are calculated as follows:

$$R_i = \sum_{\tau_j \in S_i^d} \left\lceil \frac{R_i + R_j - C_j + J_j^R}{T_j} \right\rceil C_j + C_i$$

We modify our traffic-flow attributes with the six-tuple ($P_i, C_i, T_i, D_i, J_i^R, J_i^I$) for τ_i. Differing from the conclusion of (Balakrishnan & Ozguner, 1998; Hary & Ozguner, 1997; Lu et al., 2005), here we treat the indirect interference as the interference jitter of direct higher priority traffic-flow and obtain a tighter upper bound for worst case network latency.

EXTENDING THE NETWORK LATENCY UPPER BOUND ANALYSIS

The analysis in the last section relies on finding the worst case amount of interference from higher priority traffic-flows which can pre-empt the observed one. However, this approach is valid only when network latency is less than period. In a situation, with a long transmission journey or heavy network congestion, the network latency for a packet can be quite large. It is possible that while the end flits from a previous packet are being delivered, the start flits of the next packet are already introduced. Since the data packets within the same virtual channel are served in first-in-first-service order, when a packet has to wait for the transmission of previous packets from the same traffic-flow, blocking occurs - we refer to this as self blocking. In this situation, the possible latency interferences suffered by the new arrival packet comes from not only higher priority traffic-flows

but also by the previous packet instance. None of current analysis techniques can work in this case because all these approaches only consider the single packet instance situation, with the explicit constraint $D \leq T - J^R$, and treat this multiple packet instances as the evidence of a missing deadline. However, it can very well happen that the deadline is longer than the period in the case of NoC communication. This constraint of $D \leq T - J^R$ brings inflexibility to network exploration and design. Here we lift this constraint by permitting the packet deadline to be more than its period and extend our analysis model to cover this multiple instances case.

Analysis with Self Blocking

The concept of busy period firstly introduced by Lehoczhy (Lehoczky, 1990) is used in analysis of network latency with multiple instances. A modified version of the priority level-*n* busy period, to apply to on-chip networks, is defined as a contiguous time interval during which a set of link resources are keep busy and serve all the traffic-flows of priority higher than or equal to the priority *n*. Consider the case showed in Figure 1, traffic-flows τ_2, τ_3 and τ_4 with priority 2, 3 and 4 respectively. For the link resources between router 13 and 5 (τ_4's path), the priority level-4 busy period is the contiguous time duration that the corresponding link resource keeps serving all the queueing packets with priority 4 or higher (2 or 3 in this case). Figure 6 illustrates this level-4 busy period.

The current analysis techniques proposed last section has the constraint $D \leq T - J^R$. It implies that if τ_i is schedulable then the current priority level-*i* busy period will end at or before the time instant of the next packet arrival from τ_i. However if we allow $D > T - J^R$, a current packet can be waiting transmission without missing its deadline when a successive packet from τ_i arrives, and thus the busy period can be extended beyond T_i. More than one packet instances from τ_i are served during a priority level-*i* busy period. In this case, the packet network latency would include extra interference from the previous packet arrival. So, the analysis in this section is concerned with analyzing the interference from the analyzed traffic-flow itself as well as interference from other traffic-flows in the network.

Figure 7 shows the multiple packet arrivals during a priority level-*i* busy period B_i. Consider the existence of release jitter, the packet release could start J_i^R later. The busy period illustrated

Figure 6. Priority level-4 busy period

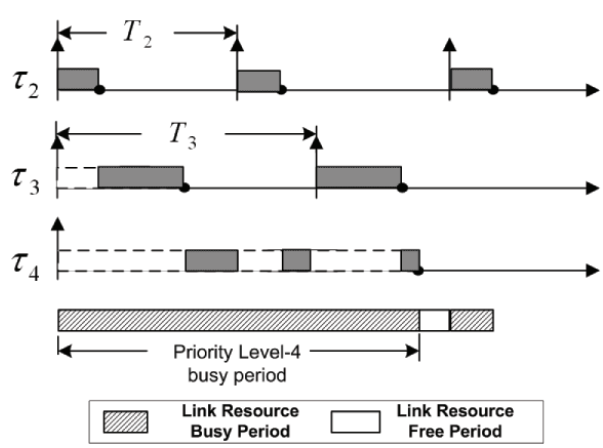
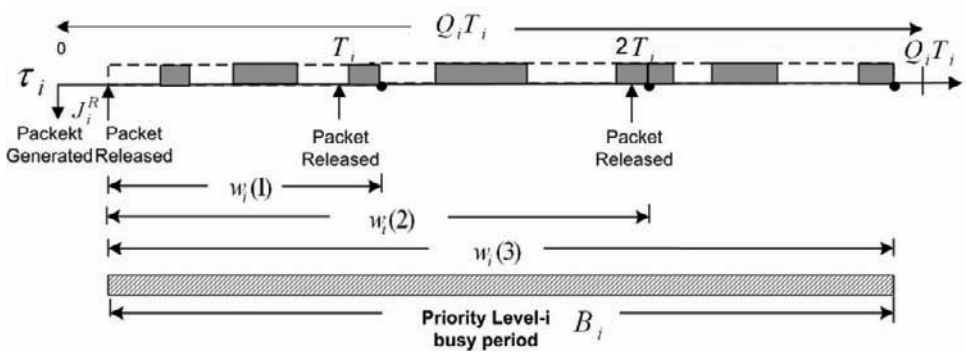


Figure 7. Priority level-i period with multiple packet arrivals



in this figure begins when the first packet from τ_i or higher is released and ends when the network is idle or a lower priority packet less than *i* is able to get a chance for transmission.

Theorem 4. *The width of priority level-i busy period* B_i can be found by:

$$B_i = \sum_{\tau_j \in S_i^D} \left\lceil \frac{R_i + R_j - C_j + J_j^R}{T_j} \right\rceil C_j + \left\lceil \frac{B_i + J_i^R}{T_i} \right\rceil C_i \tag{7}$$

Proof: According the definition of busy period, all the arrival packets of priority *i* or higher before the end of the busy period must be transmitted during the busy period. So the width of the busy period is equal to the service requirement made by traffic-flow τ_i in addition to all the higher priority packet arrivals in the busy period. Supposing that we can find an upper bound of the total busy period B_i, the maximum number of instances Q_i of a flow τ_i that falls into this busy period is calculated by:

$$Q_i = \left\lceil \frac{B_i + J_i^R}{T_i} \right\rceil$$

If the packet release from τ_i is simultaneously with the start of busy period. Consequently, the maximum service requirement from τ_i is thus:

$$\left\lceil \frac{B_i + J_i^R}{T_i} \right\rceil C_i$$

On the other hand, the maximum service time requirements generated by all the higher priority traffic-flows during the busy period should be no more than:

$$\sum_{\tau_j \in S_i^D} \left\lceil \frac{R_i + R_j - C_j + J_j^R}{T_j} \right\rceil C_j$$

As a result, the width of priority level-*i* busy period B_i can be found by:

$$B_i = \sum_{\tau_j \in S_i^D} \left\lceil \frac{R_i + R_j - C_j + J_j^R}{T_j} \right\rceil C_j + \left\lceil \frac{B_i + J_i^R}{T_i} \right\rceil C_i$$

We note that if $B_i \le T - J^R$, then the busy period ends at or before the second packet from τ_i is generated. This means that only one packet instance of τ_i is served during the busy period. Equation(7) degrades to Equation(6). If $B_i > T - J^R$, then more than one packet instance of τ_i is generated during a priority level-*i* busy period. Some

successive generated packets might be blocked by the previous packets. In this situation, to determine the worst case network latency, we need to check the network latency for each generated packet during the busy period. The maximum of these values gives the worst case network latency.

Theorem 4. *Having the traffic-flow with* $B_i > T - J^R$, the maximum network latency for τ_i is given by:

$$R_i = \max_{q=1,\ldots,Q_i} (w_i(q) - (q-1)T_i + J_i^R) \tag{8}$$

Where ***q*** *is the index of packet instance, and* $w_i(q)$ is given by:

$$w_i(q) = qC_i + \sum_{\tau_j \in S_i^D} \left\lceil \frac{w_i(q) + J_j^R + J_j^I}{T_j} \right\rceil C_j \tag{9}$$

Proof: Suppose we have calculated the priority level-*i* busy period and $B_i > T - J^R$, so more than one packet instance falls in the busy period. We need check each release to find the maximum latency. We use the index variable q to denote a packet instance of τ_i. The first packet in the busy period corresponds to $q = 1$ and the final one is $q = Q_i$. We assume a time window $w_i(q)$ which denotes the time interval from the beginning of the busy period until the completion of the qth packet transmission. The time windows for 1st, 2nd and 3rd packets in the busy period are showed in Figure 7 as $w_i(1)$, $w_i(2)$ and $w_i(3)$) respectively. The time window $w_i(q)$ is given by:

$$w_i(q) = qC_i + \sum_{\tau_j \in S_i^D} \left\lceil \frac{w_i(q) + J_j^R + J_j^I}{T_j} \right\rceil C_j$$

The variable qC_i accounts for the transmission service time of the first q packets instances of τ_i during the busy period and

$$\sum_{\tau_j \in S_i^D} \left\lceil \frac{w_i(q) + J_j^R + J_j^I}{T_j} \right\rceil C_j$$

denotes the service requirements from all the higher priority flows which fall in this time window $w_i(q)$. The value of $w_i(q)$ can be found by iteration starting with $w_i^o(q) = C_i + qT_i$ and ending when $w_i^{n+1}(q) = w_i^n(q)$. The generated time of qth packet happens at instant $(q-1)T_i$ relative to the start of busy period so the network latency of the qth instance is given by:

$$R_i(q) = w_i(q) - (q-1)T_i + J_i^R$$

As mentioned above, the worst case network latency can occur at any one of these packet releases during the busy period, and thus the worst case network latency is given by:

$$R_i = \max_{q=1,\ldots,Q_i} (w_i(q) - (q-1)T_i + J_i^R)$$

EXPERIMENTAL RESULTS

To better validate the proposed analysis, we compare its latency results with those obtained from the simulation model of an actual NoC platform under the communication load imposed by a realistic application. The chosen application is the control of an autonomous vehicle, as it includes both large volume data flows as well as control loops with short messages. Furthermore, this is an embedded application which is likely to benefit from on-chip multiprocessor platforms, because of the high computing power required by its many subsystems and because its constraints on size and power consumption are better met by on-chip rather than on-board integration. The autonomous vehicle we consider is designed to recognize an unknown space by populating a database of obstacles, obtained through stereo photogrammetry

and ultrasonic sensors. Each of the two cameras used for the stereo photogrammetric analysis feeds the system with 25 frames/second, each frame with 640x480 resolution and 8 bits colour representation. Each frame is partitioned in 4 quadrants which are loaded from the camera's frame buffer concurrently. Each quadrant of the frame is processed for background estimation and feature extraction, then the features of all quadrants of a frame are merged. Then, stereo photogrammetry analyses the features from both cameras and estimates the distance from the vehicle to the obstacle, calculates the obstacle absolute location (by using its own position obtained from a positioning sensor) and adds it to the database. Ultrasonic sensors also add obstacle information to the database, but since they have much shorter range, they are mainly used to refine the entries obtained with photogrammetry.

Figure 8. Application tasks

Tasks	Description
TPMS	Tyre pressure monitoring system
VIBS	Vibration sensor
SPES	Speed sensor
POSI	Position sensor interface
USOS	Ultasonic sensor
FBU1	Frame buffer - Left camera, upper-left quadrant
FBU2	Frame buffer - Left camera, upper-right quadrant
FBU3	Frame buffer - Left camera, lower-left quadrant
FBU4	Frame buffer - Left camera, lower-right quadrant
FBU5	Frame buffer - Right camera, upper-left quadrant
FBU6	Frame buffer - Right camera, upper-right quadrant
FBU7	Frame buffer - Right camera, lower-left quadrant
FBU8	Frame buffer - Right camera, lower-right quadrant
STAC	Stability control
TPRC	Tyre pressure control
DIRC	Direction control
NAVC	Navigation control
OBDB	Obstacle database
BFE1	Background estimation and feature extraction 1
BFE2	Background estimation and feature extraction 2
BFE3	Background estimation and feature extraction 3
BFE4	Background estimation and feature extraction 4
BFE5	Background estimation and feature extraction 5
BFE6	Background estimation and feature extraction 6
BFE7	Background estimation and feature extraction 7
BFE8	Background estimation and feature extraction 8
FDF1	Feature data fusion 1
FDF2	Feature data fusion 2
STPH	Stereo photogrammetry
THRC	Throttle control
VOD1	Visual odometry 1
VOD2	Visual odometry 2
OBMG	Obstacle database manager

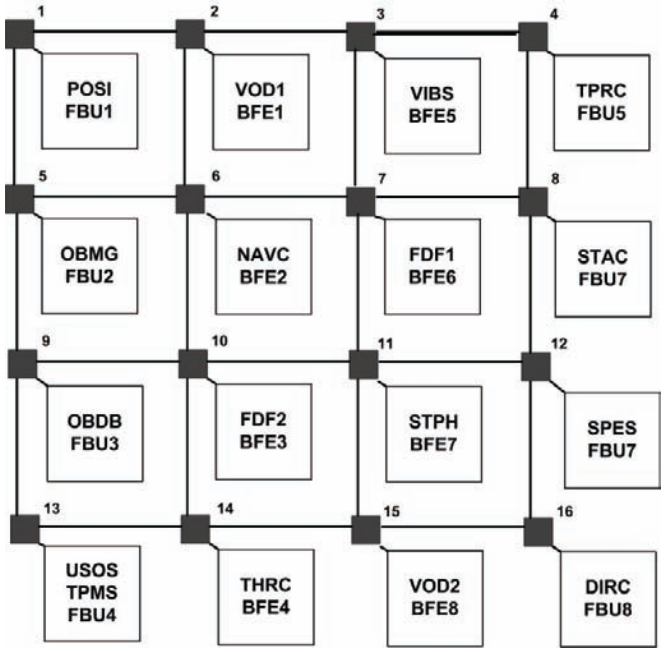
The vehicle uses the obstacle database to guide its navigation process, which controls the speed and the direction of the vehicle. Also included in this example is the stability control for the vehicle, which analyses its vibration, the pressure of its tyres and its speed. Figure 8 shows the tasks that compose the application and Figure 9 shows the traffic flows that those tasks use to communicate (activation period shown in seconds, payload in flits of 32 bits). Figure 10 shows the chosen mapping of those tasks onto 16 processors interconnected by a 4x4 NoC.

We chose a flit-accurate simulation model for the NoC platform implemented on top of Ptolemy II framework. It reused part of the NoC component library presented in (Indrusiak, Ost, Möller, Moraes, & Glesner, 2008), but the components to model virtual channels, priority-based arbitra-

Figure 9. Inter-task traffic-flows

Flow	Source	Dest	Flits	Period (s)	Deadline (s)	Priority
1	POSI	NAVC	1024	0.5	0.5	31
2	NAVC	OBDB	2048	0.5	0.5	32
3	OBDB	NAVC	16384	0.5	0.5	33
4	OBDB	OBMG	16384	0.5	0.5	34
5	NAVC	DIRC	512	0.1	0.1	24
6	SPES	NAVC	512	0.1	0.1	25
7	NAVC	THRC	1024	0.1	0.1	26
8	FBU3	VOD1	38400	0.04	0.04	1
9	FBU8	VOD2	38400	0.04	0.04	2
10	VOD1	NAVC	512	0.04	0.04	3
11	VOD2	NAVC	512	0.04	0.04	4
12	FBU1	BFE1	38400	0.04	0.04	5
13	FBU2	BFE2	38400	0.04	0.04	6
14	FBU3	BFE3	38400	0.04	0.04	7
15	FBU4	BFE4	38400	0.04	0.04	8
16	FBU5	BFE5	38400	0.04	0.04	9
17	FBU6	BFE6	38400	0.04	0.04	10
18	FBU7	BFE7	38400	0.04	0.04	11
19	FBU8	BFE8	38400	0.04	0.04	12
20	BFE1	FDF1	2048	0.04	0.04	13
21	BFE2	FDF1	2048	0.04	0.04	14
22	BFE3	FDF1	2048	0.04	0.04	15
23	BFE4	FDF1	2048	0.04	0.04	16
24	BFE5	FDF2	2048	0.04	0.04	17
25	BFE6	FDF2	2048	0.04	0.04	18
26	BFE7	FDF2	2048	0.04	0.04	19
27	BFE8	FDF2	2048	0.04	0.04	20
28	FDF1	STPH	8192	0.04	0.04	21
29	FDF2	STPH	8192	0.04	0.04	22
30	STPH	OBMG	4096	0.04	0.04	23
31	POSI	OBMG	1024	0.5	0.5	35
32	USOS	OBMG	1024	0.1	0.1	27
33	OBMG	OBDB	4096	1.0	1.0	37
34	TPMS	STAC	2048	0.5	0.5	36
35	VIBS	STAC	512	0.1	0.1	28
36	STAC	TPRC	2048	1.0	1.0	38
37	SPES	STAC	1024	0.1	0.1	29
38	STAC	THRC	1024	0.1	0.1	30

Figure 10. Mapping of the application tasks onto 4x4 NoC



tion and credit-based flow control had to be created. The strategy to evaluate the communication latency for a particular application mapped onto a NoC platform was adapted from the one presented in (Maatta et al., 2008), using actors that are mapped to producers and consumers plugged to the NoC simulation model. The major difference is that (Maatta et al., 2008) allows designers to explicitly define sequencing relationships among the communication flows using UML sequence diagrams, while here the flows are assigned priorities.

Figure 11 presents the communication latency results for the autonomous vehicle, using both the proposed analysis approach and the implemented simulation model. The analysis results are displayed as worst-case latency calculated from our new approach. Simulation results are presented for three different scenarios, each simulated for several hours (representing the execution of the application for 16 seconds, i.e. the processing of 400 frames from each camera). For convenience, the last column in the result figure gives the percentage between the maximum latency observed by simulation against the result calculated by analysis.

In the first scenario, all traffic flows are released simultaneously upon simulation start without any jitter. It was proved by Liu and Layland (Liu & Layland, 1973) that such situation is the worst case scenario for fixed-priority tasks sharing a single resource, as it will impose the maximum possible load to the system. However, as explained before, finding the critical instant when considering multiple resources is far from trivial, so it is unlikely that simultaneous release will be the critical instant for all the traffic-flows. Actually, the simulation results show that some of the traffic flows had worst-case latencies which are equal (or very close to) the values found by our analysis, but flows such as 24, 25, 26, 27 and 31 had latencies that were between 5-20% against those from the analysis, while traffics 5,6 and 38 were between 20-50%.

The second and third simulation scenarios aimed to capture the results of different interfer-

Figure 11. Communication latency figures obtained using analysis and simulation (in microseconds)

Flows	Communication Latency (us)							Margin (%)
	Theory Analysis	Simulation no jitter		Simulation jitter 10%		Simulation jitter 50%		
	Worst Case	Max	Avg	Max	Avg	Max	Avg	
1	814.90	**778.14**	396.81	10.36	10.36	10.36	10.36	95.49
2	56.80	**35.98**	30.96	20.60	20.60	20.60	20.60	63.35
3	1342.38	**941.98**	560.65	305.11	169.84	163.96	163.96	70.17
4	1147.60	**1095.70**	716.72	202.58	165.53	163.92	163.92	95.48
5	25.82	**5.46**	5.39	5.32	5.32	5.32	5.32	21.15
6	1229.74	**399.44**	202.36	369.68	18.63	159.25	7.85	32.48
7	461.52	**394.34**	204.92	10.38	10.38	204.25	13.01	85.44
8	384.16	**384.16**	384.16	384.16	384.16	384.16	384.16	100.00
9	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	100.00
10	5.20	**5.20**	5.20	5.20	5.20	5.20	5.20	100.00
11	394.62	**389.24**	389.24	385.31	13.62	331.48	8.59	98.64
12	768.24	**768.08**	768.08	768.08	413.02	768.08	388.03	99.98
13	394.54	**394.32**	394.32	394.32	384.49	389.20	384.17	99.94
14	768.24	**768.08**	768.08	768.05	395.48	768.05	386.35	99.98
15	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	100.00
16	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	100.00
17	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	100.00
18	384.08	**384.08**	384.08	384.08	384.08	384.08	384.08	100.00
19	768.16	**768.08**	768.08	768.08	408.99	768.08	387.26	99.99
20	409.88	404.56	404.56	**404.60**	36.89	404.60	24.64	98.71
21	425.24	**425.04**	425.04	389.21	30.59	404.56	22.99	99.95
22	445.84	**445.52**	445.52	376.36	34.43	367.42	26.13	99.93
23	466.48	**466.00**	466.00	404.64	29.49	212.49	21.41	99.90
24	409.92	25.72	25.72	**383.68**	30.13	294.41	21.56	93.60
25	425.32	41.08	41.08	**396.14**	30.75	150.35	21.13	93.14
26	445.88	61.52	61.52	**404.56**	32.18	181.89	21.10	90.73
27	471.74	82.00	82.00	**392.69**	30.53	218.87	21.52	83.24
28	486.68	**466.00**	466.00	466.00	104.31	466.00	84.14	95.75
29	568.68	**547.92**	547.92	541.46	97.43	428.72	85.14	96.35
30	61.68	**61.58**	61.58	61.51	41.35	41.12	41.12	99.84
31	620.16	51.45	38.65	**390.27**	10.32	10.32	10.32	62.93
32	435.56	**394.36**	202.36	231.09	12.52	343.26	13.24	90.54
33	445.72	**424.98**	424.98	49.29	41.73	41.04	41.04	95.35
34	472.00	**415.13**	225.55	347.13	34.32	88.98	23.69	87.95
35	25.88	**25.63**	15.44	5.24	5.24	5.24	5.24	99.03

ence patterns, so we introduced a release jitter for each packet of every traffic flow. In the second scenario, the upper bound of the release jitter of a packet was equal to 10% of the period of the respective traffic flow, while in the third scenario the upper bound was equal to 50% of the period. The release jitter of each packet was chosen randomly between zero and the upper bound with uniform distribution. It can be seen that the results for traffic flows 24, 25, 26, 27 and 31 in the second scenario are much closer to the worst-case values found using our analysis.

The fact that the results for some flows are still well below the worst-case values found using analysis is because our simulations couldn't cover every possible release scenario, and thus

every possible interference pattern, due to the sheer size of the configuration space. To confirm this hypothesis, we empirically chose a particular release scenario that would force flow 6 to have a latency which is closer to the worst case value found with our analysis. We then simulated that scenario, which comprehended the release of all flows but flows 8 and 13 upon simulation start. Flows 8 and 13 were configured to have offsets of 380 and 760 microseconds, respectively. In this circumstance, the latency of flow 6 was found to be 1149.20 microseconds, which corresponds to 93% of the worst-case value found using the proposed analysis. In all cases, the results found using simulation were consistently less or equal the worst-case bound found by our analysis, which further attests the safety of the analysis. In some cases, we can say that the analysis is tight - the difference between the figures obtained using both methods is zero, and in many cases the difference is very small. Such simulation shows the case that the maximum latency could be reached or closed if special release scenarios are used.

CONCLUSION

Network-on-chip architectures need to provide different levels of service to meet the needs of each component of the applications that are executed by the SoC platform. The requirements of real-time application components require a scheduling strategy and analysis approach to predict whether all real-time packets can meet their timing bounds. In this paper, we introduce an analysis approach for real-time on-chip communication with wormhole switching and fixed priority scheduling. The worst case network latency upper bound for each traffic flow can be found in all situations. The simulation results provided evidence that communication latency calculated using the real time analysis approach is safe, closely matching the figures obtained from simulation. By using the proposed analysis, system designers can flexibly evaluate the schedulability of traffic flow sets with different QoS requirements in a real-time on-chip communication platform at design time, allowing the impact of communication latency to be properly considered during application development before a system prototype is available.

REFERENCES

Audsley, N., Burns, A., Richardson, M., Tindell, K., & Wellings, A. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, *8*, 284–292.

Balakrishnan, S., & Ozguner, F. (1998). A priority-driven flow control mechanism for real-time traffic in multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems*, *9*(7), 664–678. doi:10.1109/71.707545

Benini, L., & De Micheli, G. (2001). Powering networks on chips. In *Proceedings of the International Symposium on System Synthesis.*

Benini, L., & De Micheli, G. (2002). Networks on chips: A new SoC paradigm. *Computer*, *35*(1), 70–78. doi:10.1109/2.976921

Bjerregaard, T., & Sparso, J. (2004). Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip. In *Proceedings of the 2nd Norchip Conference.*

Dally, W. (1992). Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, *3*(2), 194–205. doi:10.1109/71.127260

Dally, W., & Towles, B. (2001). Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Desing Automation Conference (DAC).*

Duato, J., Yalamanchili, S., & Ni, L. (2002). *Interconnection networks: an engineering approach*. San Francisco, CA: Morgan Kaufmann.

Furber, S., & Bainbridge, J. (2005). Future trends in SoC interconnect. In *Proceedings of the IEEE international symposium on VLSI design, automation and test*.

Goossens, K., Dielissen, J., & Radulescu, A. (2005). Æthereal network on chip: concepts, architectures, and implementations. *IEEE Design & Test of Computers*, *22*(5), 414–421. doi:10.1109/MDT.2005.99

Hary, S., & Ozguner, F. (1997). Feasibility test for real-time communication using wormhole routing. *IEE Proceedings. Computers and Digital Techniques*, *144*(5), 273–278. doi:10.1049/ip-cdt:19971369

Henkel, J., Wolf, W., & Chakradhar, S. (2004). On-chip networks: A scalable, communication-centric embedded system design paradigm. In *Proceedings of the 17th International Conference on VLSI Design*.

Indrusiak, L., Ost, L., Möller, L., Moraes, F., & Glesner, M. (2008). Applying UML Interactions and Actor-oriented Simulation to the Design Space Exploration of Network-on-Chip Interconnects. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI* (pp. 491-494).

Kavaldjiev, N., & Smit, G. (2003). A survey of efficient on-chip communications for SoC. In *Proceedings of the 4th Progress Symposium on Embedded Systems*, 129-140.

Kim, B., Kim, J., Hong, S., & Lee, S. (1998). A real-time communication method for wormhole switching networks. In *Proceedings of the International Conference on Parallel Processing.*

Lehoczky, J. (1990). Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the IEEE Real Time Sytem Symposium.*

Liu, C., & Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for computing Machinery, 20*(1), 46-61.

Lu, Z., Jantsch, A., & Sander, I. (2005). Feasibility analysis of messages for on-chip networks using wormhole routing. In *Proceedings of the Conferenece on Asia South Pacific Design Automation.*

Maatta, S., Indrusiak, L., Ost, L., Moller, L., Nurmi, J., Glesner, M., et al. (2008). Validation of executable application models mapped onto network-on-chip platforms. In *Proceedings of the International Symposium on Industrial Embedded Systems.*

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

Mutka, M. (1994). Using rate monotonic scheduling technology for real-timecommunications in a wormhole network. In *Proceedings of the 2nd workshop on parallel and distributed real time systems.*

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

Tindell, K., Burns, A., & Wellings, A. (1994). An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems Journal*, *6*(2), 133–151. doi:10.1007/BF01088593

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 2, edited by Seppo Virtanen, pp. 1-22, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 12
Modeling Communication in Multi–Processor Systems–on–Chip Using Modular Connectors

Leonidas Tsiopoulos
Åbo Akademi University, Finland

Kaisa Sere
Åbo Akademi University, Finland

Juha Plosila
University of Turku, Finland

ABSTRACT

Formal methods of concurrent programming can be used to develop and verify complex Multi–Processor Systems–On–Chip in order to ensure that these systems satisfy their functional and communication requirements. The authors use the Action Systems formalism and show how asynchronous communication of Multi–Processor Systems–on–Chip can be modeled using generic connectors composed out of simple channel components. The paper proposes a new approach to modeling generic and hierarchical connectors for handling the complexity of on–chip communication and data flow. The authors' goal is to avoid overloaded bus–based architectures and give a distributed framework. A case study presents the authors' modeling methodology.

1. INTRODUCTION

As the technologies improved and the electronic systems became increasingly complex with many independent modules communicating with each other on a single PCB board, a need for a well defined inter–communication scheme arose. Several manufacturers introduced so called bus–based System–on–Chip (SoC) communication frameworks in order to deal with the increased complexity. One of such products is the AMBA 2 AHB interface bus (ARM, 2005). The common feature of these frameworks is that there is a bus to which master modules, such as processor cores and DSPs, must connect in order to access slave modules such as memories. In such bus–based SoC

DOI: 10.4018/978-1-4666-0912-9.ch012

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

architectures the bus is a shared communication resource with the masters competing for the access to the slaves. Because it is a shared resource, it can become overloaded quickly, especially when several master modules are connected to it (Lahiri, Raghunathan, & Dey, 2001).

In order to overcome the single bus overloading, bus matrix–based architectures (Nakajima, Yamamoto, Ozaki, Sezaki, Kanakagi, Furuzono, Sakamoto, Aruga, Sumita, Tsutsumi, Ueda, & Ichinomiya, 2002) appeared consisting of several parallel buses to offer the bandwidth required for complex SoCs. The main drawback of this approach is that every master is connected to every slave in the system, resulting in a very large number of buses. To overcome this disadvantage, approaches for creating a partial bus matrix for a given application have been proposed (Pasricha, Ben–Romdhane, & Dutt, 2007) in order to reduce complexity and power consumption as well as increase efficiency. During the same time, the Network–on–Chip communication paradigm (Hemani, Jantch, Kumar, Postula, Öberg, Millberg, & Lindqvist, 2000) has been introduced offering very high communication bandwidth and network scalability through its regular structure. Depending on the requirements and priorities of a given application (high communication bandwidth, latency, reduced power consumption, etc) a system designer can choose a suitable communication scheme out of the aforementioned ones.

Multi–Processor Systems–on–Chips (MPSoC) are used in many important applications, from home use electronics and embedded systems to large control and data transfer systems. Hence efficient methods are needed in order to specify, model the communication and verify their design. Formal methods of concurrent programming, such as CSP (Hoare, 1985), Petri Nets (Desel & Juhás, 2001) and Action Systems (Back & Kurki – Suonio, 1983) can be used to design these systems. Action Systems, the modeling language of this paper, provides a rigorous framework for MPSoC design (Back, Martin, & Sere, 1995; Plosila, Seceleanu, & Sere, 2004) and it is sufficient for the work presented here. There is adequate tool support from the B Method (Abrial, 1996), by converting Action Systems to B Action Systems (Waldén & Sere, 1998), as well as the notion of *stepwise refinement* (Back & Sere, 1996) is built in the framework which allows an abstract system to be refined to a more concrete one preserving its original functionality. Note that refinement is out of the scope of this paper.

In this paper we present a well structured way to model communication infrastructures in heterogeneous MPSoCs and focus on formal specification and modeling. We propose a development method where we create the specification in a component–based and hierarchical manner. We have a *Globally Asynchronous Locally Synchronous* (GALS) systems point of view (Chapiro, 1984) where each computation and storage unit can be synchronized to a private local clock but their communication is established asynchronously eliminating the need for a global system clock. Within our framework modular connectors are composed out of asynchronous communication channels relying on the request and acknowledgement phases of the asynchronous communication. The needed arbitration is achieved within the composed connectors depending on the channels' functionality, their topological placement and the connector's own logic.

Nowadays MPSoCs are heterogeneous and may consist of many different regions requiring different communication infrastructures, thus generic and correct–by–construction connectors can be used to manage the complexity and fulfill the communication requirements. We believe that the absence of complicated non–modular and non–hierarchical arbiter modules will help in reducing the cost (in time and in money) of developing MPSoCs. The already proved functionality of the subsystems aids component reuse and helps to reduce the complexity of MPSoC arbitration.

The structure of the rest of the paper is as follows. In Section 2 we give an overview of the Ac-

tion Systems formalism following from the work presented in (Tsiopoulos, Sere, & Plosila, 2006). We formally define what a connector is within our approach and we give a theorem showing when a connector is *internally consistent*. In Section 3 we formally specify some basic asynchronous channels and we present how to compose modular connectors. In Section 4 we give an example in order to present how the asynchronous connectors of Section 3 can interconnect MPSoC modules. In Section 5 we present some related work. Finally, in Section 6 some concluding remarks are drawn and we give an overview of our future work.

2. SPECIFICATIONS IN ACTION SYSTEMS

Action Systems (Back & Kurki – Suonio, 1983) is a state–based formal framework for specification and development of reactive systems. It offers a good basis for hardware design (Back, Martin, & Sere, 1995; Plosila, Seceleanu, & Sere, 2004) and it is based on the language of guarded commands (Dijkstra, 1976). In this section we define the basic concepts of Action Systems used here.

A statement S is defined as:

$S =_{\text{def}} skip$ (empty statement)

$| \; x := e$ ((multiple) assignment)

$| \; x := x'.P$ (non–deterministic assignment)

$| \; S_1; \ldots; S_n$ (sequential composition)

| **do**A**od** (iterative composition)

| |[**var**x; S]| (block with local variable).

Here P is a predicate which should be satisfied by the value x', A is an action, x is a variable or a list of variables and e is an expression or a list of expressions. The quantified multiple assignment statement with quantified variables and expressions is denoted as:

$| \; [1 \leq i \leq n: x(i) := e(i)]$ (quantified (multiple) assignment).

An action A is defined by:

$A =_{\text{def}} g \rightarrow S$ (guarded action)

$A \; [] \; B$ (non–deterministic choice)

where g is a boolean expression called a guard, S is a statement, and A, B are actions. The quantified non–deterministic choice of actions is denoted as:

$[[]1 \leq i \leq n: A(i)] = A(1) \; [] \; \ldots \; [] \; A(n)$.

Actions are atomic, which means that only their input–output behavior is of interest, i.e., only the initial and final state with respect to the variables of an action can be observed. An action is said to be enabled in a state when its guard evaluates to true in this state.

Action compositions enable efficient modeling of complex system behavior. In this paper we will use the *non–deterministic choice composition* ([]), *sequential composition* (;) and the *prioritized composition* (//) of actions in our system specifications.

The sequential composition of actions A and B can be defined in terms of the non–deterministic choice and an auxiliary local program counter variable p (initialized to 1) as follows (Plosila, Seceleanu & Sere, 2004):

$(A \; ; B) =_{\text{def}} (p = 1 \wedge gA \rightarrow SA \; ; p := 2 \; [] \; p = 2 \wedge gB \rightarrow SB \; ; p := 1)$,

where the guards of actions A and B are denoted gA and gB respectively, and the statements of actions A and B are denoted SA and SB. The prioritizing composition of actions A and B is defined as (Sekerinski & Sere, 1996):

Figure 1. An action system A

```
sys A (in y_A; out z_A; inout u_A)
|[ var x_A
  invariant I(x_A, y_A, z_A, u_A)
  actions A
  init x_A, y_A, z_A, u_A := x_A0, y_A0, z_A0, u_A0
  exec do A od ]|
```

$A \,//\, B =_{\text{def}} A \,[]\, (\neg gA \rightarrow B)$,

where the enabled lower level priority action B can be executed if the higher level priority action A is not enabled.

Action Systems for MPSoC Modeling

An action system A suitable for MPSoC modeling is a statement of the form given in Figure 1.

Interface ports y_A z_A and u_A are the global variables of this action system. Through these interface ports system A can communicate with other systems on a SoC. The *input* variables that are read (received) but not assigned by system A are defined as in ports, the *output* variables that are assigned only by system A are defined as out ports and the *input–output* variables that are assigned by both A and other systems are defined as inout ports. Variables x_A are local to the system. Invariant I of A (properties of A that must always hold) gives its guaranteed behavior with respect to the global and the local variables. Actions A is a list of uniquely named actions of A with their detailed description and they model the functionality of A. The behavior of an action system is as follows: the variables are first initialized in the init-clause and then the enabled actions in the do-od loop of the **exec**-clause are selected for execution one at a time according to the specified action composition. In this paper, the action composition can be one of the compositions defined in the previous section or a combination of them. Actions operating on different variables can be executed in any order or in parallel.

We say that an action system A is *internally consistent* (i.e., the variables and actions conform to the invariant) if:

- The actions of A are guaranteed to always terminate.
- The initialization, the invariant and the actions are independent of all variables outside $y_A \cup z_A \cup u_A \cup x_A$ of A.
- The initialization and the actions of A establish the invariant.

Based on this we can further use action system A in system compositions as will be seen later.

Connectors in Action Systems

Several action systems can be combined to form action systems. Let A and B be two action systems of the form given in Figure 2.

With variables x_A and x_B disjoint, action systems A and B can be composed together to be subsystems of a third action system. Let us call this resulting hierarchical action system a Connector.

Definition. *A* Connector *is a system with the following form*:

Figure 2. Action systems A and B

sys A (**in** y_A; **out** z_A; **inout** u_A)	**sys** B (**in** y_B; **out** z_B; **inout** u_B)
|[**var** x_A	|[**var** x_B
invariant $I(x_A, y_A, z_A, u_A)$	**invariant** $I(x_B, y_B, z_B, u_B)$
actions A	**actions** B
init $x_A, y_A, z_A, u_A := x_{A0}, y_{A0}, z_{A0}, u_{A0}$	**init** $x_B, y_B, z_B, u_B := x_{B0}, y_{B0}, z_{B0}, u_{B0}$
exec do A **od**]|	**exec do** B **od**]|

sys Connector (**in**y_C; **out**z_C; **inout**u_C)

|[**subsys** A || B

varx_C

invariant$I(x_C, y_C, z_C, u_C)$

actions$C$

init$x_C, y_C, z_C, u_C := x_{C0}, y_{C0}, z_{C0}, u_{C0}$

exec do$C$**od**]|,

where the **subsys**–clause defines the participating subsystems and their parallel composition (Plosila, Seceleanu, & Sere, 2004). Connector is defined to be the action system:

sys ConnectorAB (**in**$y_C \subseteq y_A \cup y_B$; **out**$z_C \subseteq z_A \cup z_B$; **inout**$u_C \subseteq u_A \cup u_B$)

|[**var**$x_C, x_A, x_B, (y_A \cup y_B) - y_C, (z_A \cup z_B) - z_C, (u_A \cup u_B) - u_C$

invariant$I(x_C, y_C, z_C, u_C, (y_A \cup y_B) - y_C), (z_A \cup z_B) - z_C, (u_A \cup u_B) - u_C) \wedge$

$I(x_A, y_A, z_A, u_A) \wedge I(x_B, y_B, z_B, u_B)$

actionsC, A, B

init$x_C, x_A, x_B, \ldots, u_C := x_{C0}, x_{A0}, x_{B0}, \ldots, u_{C0}$

exec do$C$ [] A [] B**od**]|,

where some of the ports of A and B are now ports of ConnectorAB and the rest of the ports of A and B become local variables of ConnectorAB.

The hierarchical action system ConnectorAB combines the variables, the initializations and the actions of systems A and B. Renaming may be applied on the variables and actions if internal name clashes occur. The invariant of ConnectorAB gives its guaranteed behavior with respect to the ports and the local variables. The ports of ConnectorAB will be called its interface whereas the ports $(y_A \cup y_B) - y_C$, $(z_A \cup z_B) - z_C$ and $(u_A \cup u_B) - u_C$ of A and B are local variables of ConnectorAB and used for communication between A and B. Actions C connect subsystems A and B together and they might add extra functionality if needed. ConnectorAB is executed by initializing the local variables and the ports and then interleaving the execution of its actions.

A Connector whose functionality depends on the prioritized composition (Sekerinski & Sere, 1996) of its subsystems A and B is an action system with the following subsystem composition clause:

subsys A // B.

Connector is then defined to be the action system:

sys ConnectorAB (**in**$y_C \subseteq y_A \cup y_B$; **out**$z_C \subseteq z_A \cup z_B$; **inout**$u_C \subseteq u_A \cup u_B$)

|[**var**x_C, x_A, x_B

invariant $I(x_C, y_C, z_C, u_C) \wedge I(x_A, y_A, z_A, u_A) \wedge I(x_B, y_B, z_B, u_B)$

actions C, A, B

init $x_C, x_A, x_B, \ldots, u_C := x_{C0}, x_{A0}, x_{B0}, \ldots, u_{C0}$

exec do C [] $(A \mathbin{//} B)$ **od**]|.

The following theorem specifies when a connector is internally consistent.

Theorem. *Assume that the subsystems* A *and* B *of a connector* C *are internally consistent. Furthermore assume that*

(a) *The local variables and the action identifiers* (*names*) *of the systems are disjoint,*

(b) *Each subsystem respects the invariant of the other subsystem and the invariant of connector* C, *and*

(c) *Connector* C *respects the invariants of the subsystems.*

Then C *is internally consistent.*

Theorem (a) requires that the variables as well as the action names of the systems are disjoint. This is achieved by renaming. Theorem (b) requires that the actions of the subsystems should preserve each other's invariant as well as the connector's invariant. Finally, Theorem (c) requires that the actions of the connector should preserve the invariants of its subsystems. This is a restriction on the assignments to the ports of the subsystems. Relying on this theorem we can build consistent systems out of consistent subsystems in a compositional manner. The proof of the theorem follows from the proof of the compositionality theorem in (Back & von Wright, 2003).

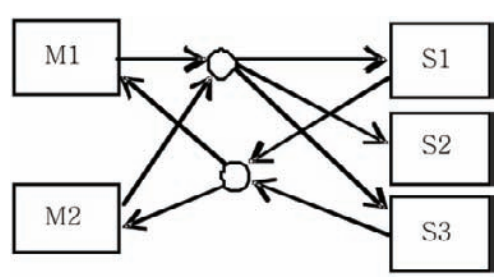
3. CONSTRUCTING CONNECTORS

The specification of MPSoCs in general leads to complex systems, thus a component–based development method is potentially helpful. To achieve this, we start by specifying a few simple action systems as basic components called channels. Then we compose modular connectors out of these channels in order to distribute data packets from the masters to the slaves and vice–versa. The formal design framework with consistency properties enables us to verify the overall functional correctness of a system's architecture, including correctness of data transfer protocols on channels between system modules. An abstract architectural view of our modeling approach with two masters and three slaves is shown in Figure 3. The details hidden by the circles in Figure 3 will be presented by a case study in the rest of Section 3 and in Section 4.

Asynchronous Channels

The channels we specify in this paper are all asynchronous, i.e. each data element enters and exits the channel according to the interface control variables which model the *asynchronous communication handshakes*. The data elements coming into the channels come out again in the same order. We start by giving specifications of *push* and *pull* channels as well as FIFO push channels. *Push* channels are used to transfer data

Figure 3. MPSoC communication with asynchronous connectors



from a master to a slave upon *request*, while a *pull* channel is used to transfer data from a slave to a master upon *acknowledgement* (Peeters, 1996).

Asynchronous Push Channels

A push channel must at least have one in data interface port, named *din*, one out data interface port, named *dout*, and two inout control interface ports named *cin* and *cout*, respectively. The ports *cin* and *cout* model the asynchronous communication handshakes at the input and at the output of the channel, respectively, and for simplicity they are of type *boolean*. The value *true* corresponds to the request value and the value *false* corresponds to the acknowledgment value of the asynchronous communication. Moreover, the ports *din* and *dout* are used for the data propagation and they are of the generic type *data*. Additional interfaces can be defined if needed.

Before we give the properties of a basic push channel, we clarify the correspondence of the control and data ports. Depending on the value of the control ports *cin* and *cout*, we assume the following for the existence of data in the *push* case:

$cin =_{def}$ data at the input

$cout =_{def}$ data at the output

$\neg cin =_{def}$ no data at the input

$\neg cout =_{def}$ no data at the output.

Furthermore, when a master M sends data to a slave S and assuming there is one push channel Ch in between, the following sequence of asynchronous control flow occurs:

$cout_M \rightarrow cin_{Ch} \rightarrow cout_{Ch} \rightarrow cin_S$.

Let PushChannel be the name of a basic asynchronous push channel. The specification of this channel is given in Figure 4. Its guaranteed behavior is as follows:

- New data cannot be received at the input if the previous data has not been taken from the output.

This property is formalized in the invariant and the channel's functionality is modeled with the sequential composition of actions *PushTransferData* and *PushAckTransfer*.

The execution starts when a request is assigned to *cin* indicating that there is data at the input. The specification of a similar channel component but with one additional boolean **inout** global variable, named *block*, is given in Figure 5. Let us call this channel BlockingPushChannel.

The *blocking* feature is important in component compositions where this channel must ensure that no other component (being able to assign *block*) will interleave its actions during the sequential composition of actions *BlockAndTransferData* and *UnblockAndAckTransfer* of BlockingPush-

Figure 4. Specification of PushChannel

```
sys PushChannel(in din : data; out dout : data; inout cin, cout : boolean)
|[ invariant cout ⇒ cin
  actions  PushTransferData : cin → cout, dout := true, din
           PushAckTransfer : ¬cout → cin := false
  init cin, cout := false
  exec do PushTransferData ; PushAckTransfer od ]|
```

Figure 5. Specification of BlockingPushChannel

```
sys BlockingPushChannel(in din : data; out dout : data; inout cin, cout, block : boolean)
|[ invariant cout ⇒ cin
  actions  BlockAndTransferData : cin ∧ ¬block → cout, dout, block := true, din, true
           UnblockAndAckTransfer : ¬cout → cin, block := false
  init cin, cout, block := false
  exec do BlockAndTransferData ; UnblockAndAckTransfer od ]|
```

Channel. Such compositions will be presented in the next sections.

One–Place FIFO Push Channel

A one–place FIFO push channel has the same interface ports as PushChannel specified above. Let us call this FIFO channel Fifo1PushChannel. Its guaranteed property is as follows:

- New data received at the input can be propagated to the output only when the output is available to receive it.

This property is formalized by the guard of action *Fifo1TransferData* of the channel's specification shown in Figure 6.

Fifo1TransferData is enabled when there is data at the input, i.e., *cin* equals *true*, and the output is free to receive it, i.e., *cout* equals *false*. The difference of Fifo1PushChannel compared to the specification of PushChannel is that it is less restrictive to the environment. In other words, there is no invariant in Fifo1PushChannel to be preserved by any system sending data to it. The guard of the action of Fifo1PushChannel ensures that the guaranteed property of this channel will always hold. We believe that all push channel specifications are relevant to the systems we are interested in.

Asynchronous Pull Channels

A basic pull channel consists of the same interface ports as the basic PushChannel but it propagates data upon acknowledgement. Before we give the properties of this channel, we clarify the correspondence of the control and data ports. Depending on the value of the control ports *cin* and *cout*, we assume the following for the existence of data in the *pull* case:

$cin =_{def}$ no data at the input

$cout =_{def}$ no data at the output

$\neg cin =_{def}$ data at the input

$\neg cout =_{def}$ data at the output.

Furthermore, when a master M requests data from a slave S and assuming there is one pull

Figure 6. Specification of Fifo1PushChannel

```
sys Fifo1PushChannel(in din : data; out dout : data; inout cin, cout : boolean)
|[ actions  Fifo1TransferData : cin ∧ ¬cout → dout, cout, cin := din, true, false
  init cin, cout := false
  exec do Fifo1TransferData od  ]|
```

channel Ch in between, the following sequence of flow control occurs:

$cin_M \rightarrow cout_{Ch} \rightarrow cin_{Ch} \rightarrow cout_S \rightarrow \neg cout_S \rightarrow \neg cin_{Ch} \rightarrow \neg cout_{Ch} \rightarrow \neg cin_M$.

Let PullChannel be the name of this basic asynchronous pull channel. The property of this channel is as follows:

- In order for data and an acknowledgement to exist at the output, there must have been propagated from the input.

The specification of PullChannel is given in Figure 7. Action *PullTransferRequested* propagates the request signal from the output to the input of the channel. Action *PullTransferData* propagates the acknowledgement and the data from the input to the output of the channel. This occurs because from the master–slave protocol point of view the slave can send data to a master through acknowledgement. The invariant formalizes the above property.

The specification of a BlockingPullChannel with the additional blocking **inout** global variable is given in Figure 8. The same reasons of existence as of BlockingPushChannel apply to BlockingPullChannel too.

Asynchronous Connectors

Let us now present how to build asynchronous connectors out of the asynchronous channels specified in the previous subsection. In this paper we specify connectors for *prioritized* and *replicated push* data transfers as well as for *prioritized* and *replicated pull* data transfers, as we consider them to be representative for the systems we are interested in. We could specify several additional connectors but we concentrate on the aforementioned models in this paper. Moreover, we consider multiple place FIFOs to be connectors composed out of multiple Fifo1PushChannels. We compose the push connectors out of asynchronous

Figure 7. Specification of PullChannel

```
sys PullChannel(in din : data; out dout : data; inout cin, cout : boolean)
|[ invariant ¬cout ⇒ ¬cin
  actions  PullTransferRequested : cout → cin := true
           PullTransferData : ¬cin → cout, dout := false, din
  init  cin, cout := false
  exec  do PullTransferRequested ; PullTransferData  od  ]|
```

Figure 8. Specification of BlockingPullChannel

```
sys BlockingPullChannel(in din : data; out dout : data; inout cin, cout, block : boolean)
|[ invariant ¬cout ⇒ ¬cin
  actions  BlockPullTransferRequested : cout ∧ ¬block → cin, block := true, true
           UnblockPullTransferData : ¬cin → cout, dout, block := false, din, false
  init  cin, cout, block := false
  exec  do BlockPullTransferRequested ; UnblockPullTransferData  od  ]|
```

push channels. Furthermore, we compose the pull connectors out of asynchronous pull channels.

FIFO Connector of N–Places

A FIFO connector of *n–places* is composed out of *n* Fifo1PushChannels, where $n \geq 2$, connected in series as shown in Figure 9. If $n = 1$ then the resulting system is a Fifo1PushChannel. The arrows with variable names *c_fifo*(1..*n* – 1) (read control FIFO) show the pairwise mapping of the subsystems' interface control (asynchronous handshake) ports. Similar data control variables are defined (but not shown in Figure 9) to map the data ports of the subsystems. These mapping variables are now the local array variables of the connector which substitute the corresponding in, out and inout interface ports of the subsystems as shown in Figure 9. Let us call this connector FifoConnector.

In order to create the formal specification of FifoConnector we need first to instantiate the first one–place FIFO channel, then the nth one–place FIFO channel and finally the intermediate FIFO channels to be composed as subsystems in the subsys-clause. These instantiations are defined as follows:

Fifo1(1): Fifo1PushChannel([*din*/*din*], [*d_fifo*(1)/*dout*], [*cin*/*cin*], [*c_fifo*(1)/*cout*]),

Fifo1(*n*): Fifo1PushChannel([*d_fifo*(*n*–1)/*din*], [*dout*/*dout*], [*c_fifo*(*n*–1)/*cin*], [*cout*/*cout*]),

Fifo1(*i*): Fifo1PushChannel([*d_fifo*(*i*–1)/*din*], [*d_fifo*(*i*)/*dout*],[*c_fifo*(*i*–1)/*cin*],[*c_fifo*(*i*)/*cout*]),

where $i = 2..n - 1$. The interface ports of FifoConnector are given from the above instantiations of Fifo1(1) and Fifo1(*n*). The in data port of Fifo1(1) becomes the in port of the connector, the out data port of Fifo1(*n*) becomes the out data port of the connector and the control ports *cin* of Fifo1(1) and *cout* of Fifo1(*n*) become the inout ports of FifoConnector. Excerpt of the specification of this connector with a queue size parameter *n* is shown in Figure 10. Since there is no extra logic to be added, additional actions are not needed.

Generic Push Prioritizer

A generic connector for *prioritized* push data transfers is composed out of *n* BlockingPushChannels, where $n \geq 2$. If $n = 1$ then the resulting system is a simple BlockingPushChannel. Let us call this connector Push_Prioritizer. The functionality of

Figure 9. A FifoConnector of n–places

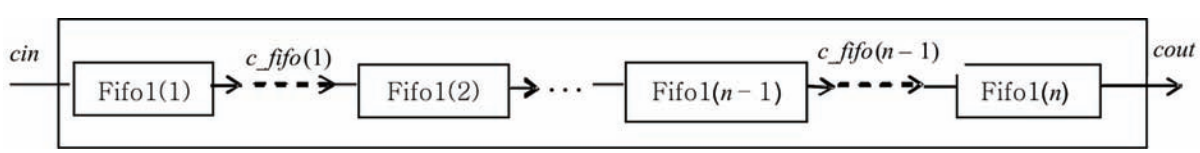


Figure 10. Excerpt of FifoConnector

```
sys FifoConnector (in din : data; out dout : data; inout cin, cout : boolean) [n]
|[ subsys Fifo1(1) || ... || Fifo1(n)
  var c_fifo(1..n–1) : boolean, d_fifo(1..n–1) : data
  init cin, c_fifo(1..n–1), cout := false ]|
```

Push_Prioritizer results from the prioritized composition of its subsystems. Only the output of the channel with the highest priority is observed at each execution cycle. In order to create the formal specification of this connector we need first to rename and instantiate its composed subsystems, as well as their ports, as follows:

PushPrioChannel(i): BlockingPushChannel
([*dinPushPrio*(i)/*din*], [*doutPushPrio*(i)/*dout*],

[*cinPushPrio*(i)/*cin*], [*coutPushPrio*(i)/*cout*],
[*PushBlock*/*block*]),

where $i = 1..n$. The in, out data ports and inout interface control ports of the subsystems constitute the in, out data and inout interface control ports of Push_Prioritizer, respectively. The renamed and shared between the instantiated subsystems blocking interface variable *PushBlock* becomes now local variable of Push_Prioritizer. Push_Prioritizer is shown in Figure 11.

The property to be maintained guaranteed of Push_Prioritizer is as follows:

- Data and request can only exist on one output at each execution cycle.

Excerpt of the specification of Push_Prioritizer is shown in Figure 12. The invariant in the figure formalizes the above property. Notice the decreasing priority level from PushPrioChannel(1) to PushPrioChannel(n).

Generic Push Replicator

A generic connector for *replicated* push data transfers is composed out of $n + 1$ PushChannels, where $n \geq 2$. If $n = 1$ then the connector would function in the same way as two PushChannels connected in series. The connector has n output channels and one extra channel constitutes the input channel. Let us call this connector Push_Replicator. A diagrammatic view of a generic Push_Replicator is shown in Figure 13.

Figure 11. A Push_Prioritizer with n blocking push channels

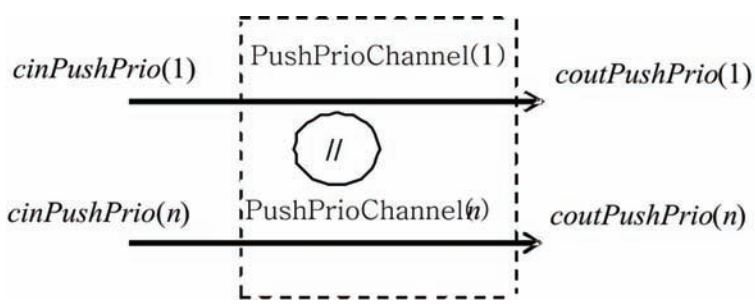
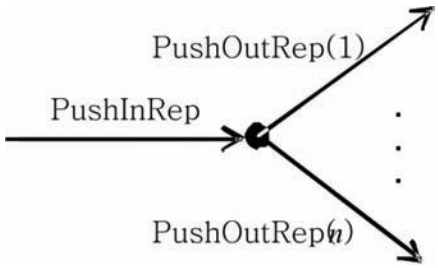


Figure 12. Excerpt of Push_Prioritizer

```
sys Push_Prioritizer (in dinPushPrio(1..n) : data; out doutPushPrio(1..n) : data;
                      inout cinPushPrio(1..n), coutPushPrio(1..n) : boolean) [n]
|[ subsys PushPrioChannel(1) // … // PushPrioChannel(n)
   var PushBlock : boolean
   invariant (∀i.1≤i≤n ∧ coutPushPrio(i) ⇒ (∀j.1≤j≤n ∧ i≠j ⇒ ¬coutPushPrio(j)))
   init cinPushPrio(1..n), coutPushPrio(1..n), PushBlock := false ]|
```

Figure 13. A generic Push_Replicator



The instantiation and renaming of its subsystems and their ports is as follows:

PushInRep: PushChannel ([*dinPushInRep*/*din*], [*doutPushInRep*/*dout*],

[*cinPushInRep*/*cin*], [*coutPushInRep*/*cout*]),

PushOutRep(*i*): PushChannel ([*dinPushOutRep*(*i*)/*din*], [*doutPushOutRep*(*i*)/*dout*],

[*cinPushOutRep*(*i*)/*cin*], [*coutPushOutRep*(*i*)/*cout*]),

where $i = 1..n$. The in data port, *dinPushInRep*, and inout control port, *cinPushInRep*, of channel PushInRep form the inputs of Push_Replicator. Similarly, all the out data ports, *doutPushOutRep*(*i*), and all the inout control ports, *coutPushOutRep*(*i*), of channels PushOutRep(*i*) form the outputs of Push_Replicator. The rest of the subsystems' ports become local variables of Push_Replicator. The property to be always guaranteed of this connector is as follows:

- The data on the input ports of the output channels will be the same as long as no new data will appear on the output port of the input channel.

An action is needed to model the above property as well as to acknowledge the output port of the input channel so that it can receive new request and data. Let us call this action *PushReplicate* which is specified as follows:

PushReplicate: $(\forall i: 1 \le i \le n: \neg cinPushOutRep(i)) \wedge coutPushInRep \rightarrow$

$[1 \le i \le n: cinPushOutRep(i), dinPushOutRep(i) := true, doutPushInRep]$;

$coutPushInRep := false$.

Excerpt of the specification of Push_Replicator is shown in Figure 14. The invariant shown in this figure formalizes the property of Push_Replicator.

Figure 14. Excerpt of Push_Replicator

```
sys Push_Replicator(in dinPushInRep : data; out doutPushOutRep(1..n) : data;
                    inout cinPushInRep, coutPushOutRep(1..n) : boolean) [n]
|[ subsys PushOutRep(1)|| … || PushOutRep(n) || PushInRep
  var coutPushInRep, cinPushOutRep(1..n) : boolean;
      doutPushInRep, dinPushOutRep(1..n) : data
  invariant (∀i.1≤i≤n ∧ cinPushOutRep(i) ∧ ¬coutPushInRep ⇒
                                  dinPushOutRep(i) = doutPushInRep)
  init cinPushInRep, coutPushInRep, cinPushOutRep(1..n),
                                  coutPushOutRep(1..n) := false
  actions PushReplicate
  exec do PushReplicate od ]|
```

Generic Pull Prioritizer

A generic connector for *prioritized* pull data transfers is composed out of n BlockingPullChannels, where $n \geq 2$. If $n = 1$ then the connector would function in the same way as a BlockingPullChannel. Let us call this connector Pull_Prioritizer. The functionality of Pull_Prioritizer is similar to the functionality of Push_Prioritizer, i.e., the input and output of the subsystem with the highest priority are observed at each execution cycle. The difference being that in the pull data transfer case, data is propagated during the acknowledgement phase.

In order to create the formal specification of this connector we need first to rename and instantiate n BlockingPullChannels, as well as their ports, as follows:

PullPrioChannel(i): BlockingPullChannel
([*dinPullPrio*(i)/*din*], [*doutPullPrio*(i)/*dout*],

[*cinPullPrio*(i)/*cin*], [*coutPullPrio*(i)/*cout*],
[*PullBlock*/*block*]),

where $i = 1..n$. Pull_Prioritizer is shown in Figure 15. The in, out data and inout interface control ports of the subsystems constitute the in, out data and inout interface control ports of Pull_Prioritizer, respectively. The renamed and shared between the instantiated subsystems blocking interface variable *PullBlock* becomes now local variable of Pull_Prioritizer.

The property of Pull_Prioritizer to be maintained guaranteed is as follows:

- Request for a data transfer exists only on one input of one subsystem at each execution cycle.

Excerpt of the specification of Pull_Prioritizer is shown in Figure 16. The invariant in the figure formalizes the above property.

Generic Pull Replicator

A generic connector for *replicated* pull data requests is composed out of $n + 1$ PullChannels, where $n \geq 2$. If $n = 1$ then the connector would

Figure 15. A generic Pull_Prioritizer

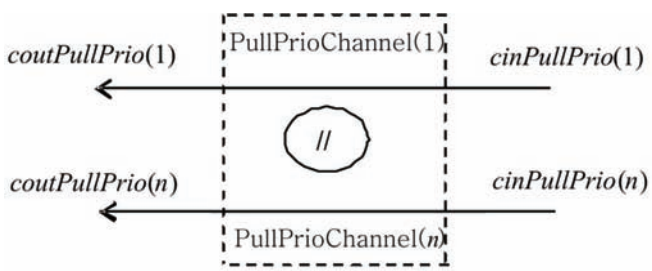


Figure 16. Excerpt of the specification of Pull_Prioritizer

sys Pull_Prioritizer (**in** *dinPullPrio*(1..*n*) : *data*; **out** *doutPullPrio*(1..*n*) : *data*;
inout *cinPullPrio*(1..*n*), *coutPullPrio*(1..*n*) : *boolean*) [*n*]
|[**subsys** PullPrioChannel(1)// … // PullPrioChannel(*n*)
var *PullBlock* : *boolean*
invariant $(\forall i.1 \leq i \leq n \wedge cinPullPrio(i) \Rightarrow (coutPullPrio(i) \wedge (\forall j.1 \leq j \leq n \wedge i \neq j \Rightarrow \neg cinPullPrio(j))))$
init *cinPullPrio*(1..*n*), *coutPullPrio*(1..*n*), *PullBlock* := *false*]|

function in the same way as two PullChannels connected in series. The connector has *n* input channels and one output channel. Let us call this connector Pull_Replicator. The difference in functionality of Pull_Replicator compared to Push_Replicator is that data is propagated on the acknowledgement phase from the input channels to the output channel after requests for data are replicated to the input channels from the output channel.

In order to create the formal specification of Pull_Replicator we need first to rename and instantiate *n* PullChannels to be the input channels as well as one channel to be the output channel as follows:

PullOutRep: PullChannel ([*dinPullOutRep*/*din*], [*doutPullOutRep*/*dout*],

[*cinPullOutRep*/*cin*], [*coutPullOutRep*/*cout*]),

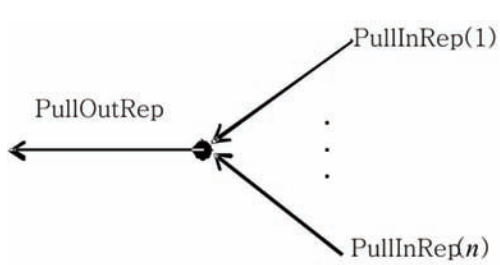
PullInRep(*i*): PullChannel ([*dinPullInRep*(*i*)/*din*], [*doutPullInRep*(*i*)/*dout*],

[*cinPullInRep*(*i*)/*cin*], [*coutPullInRep*(*i*)/*cout*]),

where $i = 1..n$. The out and inout ports of channel PullOutRep form the output ports of Pull_Replicator and the in and inout ports of channels PullInRep(1..*n*) form the input ports of Pull_Replicator. A diagrammatic view of a generic Pull_Replicator is shown in Figure 17.

The properties to be always guaranteed of this connector are as follows:

Figure 17. A generic Pull_Replicator



- The request on the outputs of all the input channels must have been propagated from the output channel.

Two actions are needed to model the above properties. Action *RequestReplicate* replicates a request from the input of the output channel to all the outputs of the input channels:

RequestReplicate: *cinPullOutRep* ∧ (∀*i*: 1≤*i*≤*n*: ¬coutPullInRep(*i*)) →

[1≤*i*≤*n*: *coutPullInRep*(*i*):= *true*].

Action *PropagateData* propagates data from the input channel which received it first, to the output channel:

PropagateData: [[]1≤*i*≤*n*: ¬*coutPullInRep*(*i*) ∧ *cinPullOutRep* →

cinPullOutRep, *dinPullOutRep*:= *false*, *doutPullInRep*(*i*)].

A new request for data will be propagated from the output channel to the input channels after all the input channels have completed their asynchronous communication phases. Excerpt of the specification of Pull_Replicator is shown in Figure 18. The invariant shown in this figure formalizes the properties of Pull_Replicator.

4. FORMAL SPECIFICATION OF THE COMMUNICATION IN AN MPSOC

We can now use the connectors specified in the previous section to model the propagation of data in an abstract MPSoC scheme of two masters and three slaves. Prior to the final interconnection of the systems developed so far to the masters and slaves of this example, let us specify abstractly a Master and a Slave. These specifications are given in Figure 19 and Figure 20, respectively. We

Figure 18. Excerpt of the specification of Pull_Replicator

```
sys Pull_Replicator (in dinPullInRep(1..n) : data; out doutPullOutRep : data;
                     inout cinPullInRep(1..n), coutPullOutRep : boolean) [n]
|[ subsys PullInRep(1) || ... || PullInRep(n) || PullOutRep
  var coutPullInRep(1..n), cinPullOutRep : boolean;
      doutPullInRep(1..n), dinPullOutRep : data
  invariant (∀i. 1≤i≤n ∧ coutPullInRep(i) ⇒ cinPullOutRep)
  init cinPullInRep(1..n), coutPullOutRep := false
  actions RequestReplicate, PropagateData
  exec do RequestReplicate ; PropagateData od ]|
```

specify only *interface actions* dealing with data transfers. We do not include local computation actions but it is assumed that they exist. Generally masters model processing units on an MPSoC and slaves model memories. Furthermore, the output ports of the masters will be responsible for sending data to the slaves via push connectors and the input ports of the masters will be responsible for receiving data from the slaves via pull connectors. Similarly for the slaves, their input ports will be receiving data from the masters and their output ports will be sending data to the masters.

Master sends data to Slave with action *MSendData* by assigning data and a request to ports *dout*

Figure 19. Specification of master

```
sys Master (in din : data; out dout : data; inout cin, cout : boolean)
|[ var buf : set of data
  actions MSendData : ¬cout → cout := true; dout := d.d : data
          MRequestData : ¬cin → cin := true
          MReceiveData : ¬cin → buf := buf ∪ {din}
  init cin, cout, buf := false, false, ∅
  exec do MSendData [] (MRequestData ; MReceiveData) od ]|
```

Figure 20. Specification of slave

```
sys Slave (in din : data; out dout : data; inout cin, cout : boolean)
|[ var buf : set of data
   actions SReceiveData : cin → cin := false; buf := buf ∪ {din}
           SSendData : cout → cout := false; dout := d.d : data
   init cin, cout, buf := false, false, ∅
   exec do SReceiveData [] SSendData od ]|
```

and *cout*, respectively. These values propagate through push connectors and arrive to Slave. Then, Slave receives the data with action *SReceive-Data* which stores the data of in port *din* to a local buffer and acknowledges the communication by setting its inout port *cin* to *false*. Master may request a data transfer from Slave by assigning the request value to its inout port *cin* which is then propagated to Slave via pull connectors. Slave responds to the request with action *SSend-Data* by acknowledging and placing data to its interface ports *cout* and *dout*, respectively. The acknowledgement and the data by Slave arrive to Master via pull connectors. A new communication cycle may then begin.

Let us now instantiate connectors specified in the previous section to model the asynchronous communication of two master and three slave instances. Let us further assume that one master has higher priority than the other master (as it is usually the case in MPSoCs) for push data transfers to the slaves as well as for pull data requests. A diagrammatic view of this system instantiation is given in Figure 21. The arrows show the pair-wise relationships of the subsystems' ports as well as the flow of data. The names of the arrows are the new local variables of the final system which substitute the corresponding ports of the subsystems in order to do the actual final mapping. The names of the subsystems come from the instantiation which is given below.

The instantiation of the subsystems as well as the substitution of their interface ports with the mapping variables is given next. We start with the instantiation of two masters named M1 and M2:

M1: Master ([*d_pull*(5)/*din*], [*d_push*(1)/*dout*], [*c_pull*(5)/*cin*], [*c_push*(1)/*cout*]),
M2: Master ([*d_pull*(6)/*din*], [*d_push*(2)/*dout*], [*c_pull*(6)/*cin*], [*c_push*(2)/*cout*]).

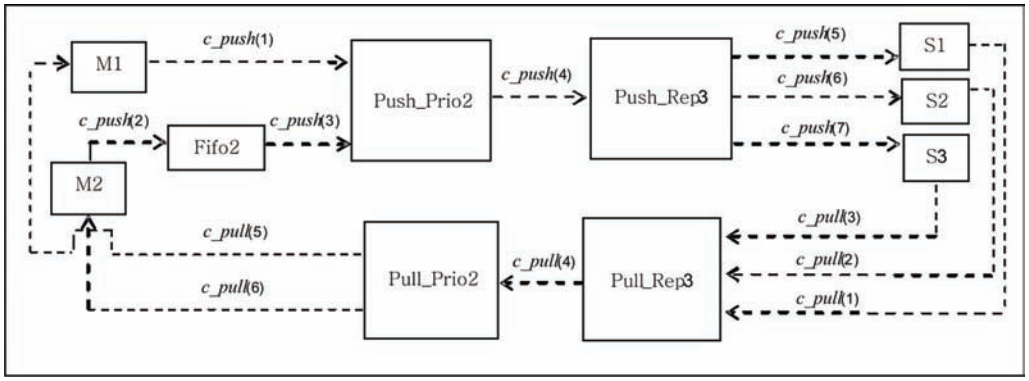
The instantiation of three slaves named S1, S2 and S3 is as follows:

S1: Slave ([*d_push*(5)/*din*], [*d_pull*(1)/*dout*], [*c_push*(5)/*cin*], [*c_pull*(1)/*cout*]),
S2: Slave ([*d_push*(6)/*din*], [*d_pull*(2)/*dout*], [*c_push*(6)/*cin*], [*c_pull*(2)/*cout*]),
S3: Slave ([*d_push*(7)/*din*], [*d_pull*(3)/*dout*], [*c_push*(7)/*cin*], [*c_pull*(3)/*cout*]).

We then instantiate the connectors used for the interconnection of the masters and the slaves. First we instantiate a two–place FIFO connector named Fifo2:

Fifo2: FifoConnector ([*d_push*(2)/*din*], [*d_push*(3)/*dout*], [*c_push*(2)/*cin*], [*c_push*(3)/*cout*])[2].

Figure 21. An abstract MPSoC of two masters and three slaves and the data flow between them



A Push_Prioritizer with two BlockingPushChannels is named Push_Prio2 and instantiated as follows:

Push_Prio2: Push_Prioritizer ([*d_push*(1)/*dinPushPrio*(1)], [*d_push*(3)/*dinPushPrio*(2)],

[*d_push*(4)/*doutPushPrio*(1..2)], [*c_push*(1)/*cinPushPrio*(1)],

[*c_push*(3)/*cinPushPrio*(2)], [*c_push*(4)/*coutPushPrio*(1..2)])[2].

A Push_Replicator with three output PushChannels is named Push_Rep3 and instantiated as follows:

Push_Rep3: Push_Replicator([*d_push*(4)/*dinPushInRep*], [*d_push*(5)/*doutPushOutRep*(1)],

[*d_push*(6)/*doutPushOutRep*(2), [*d_push*(7)/*doutPushOutRep*(3)],

[*c_push*(4)/*cinPushInRep*], [*c_push*(5)/*coutPushOutRep*(1)],

[*c_push*(6)/*coutPushOutRep*(2)], [*c_push*(7)/*coutPushOutRep*(3)]) [3].

A Pull_Replicator with three input PullChannels is named Pull_Rep3 and instantiated as follows:

Pull_Rep3: Pull_Replicator ([*d_pull*(3)/*dinPullInRep*(1)], [*d_pull*(2)/*dinPullInRep*(2)],

[*d_pull*(1)/*dinPullInRep*(3)], [*d_pull*(4)/*doutPullOutRep*],

[*c_pull*(3)/*cinPullInRep*(1)], [*c_pull*(2)/*cinPullInRep*(2)],

[*c_pull*(1)/*cinPullInRep*(3)], [*c_pull*(4)/*coutPullOutRep*]) [3].

Finally, a Pull_Prioritizer with two BlockingPullChannels is named Pull_Prio2 and instantiated as follows:

Pull_Prio2: Pull_Prioritizer ([*d_pull*(4)/*dinPullPrio*(1..2)], [*d_pull*(5)/*doutPullPrio*(1)],

[*d_pull*(6)/*doutPullPrio*(2)], [*c_pull*(4)/*cinPullPrio*(1..2)],

[*c_pull*(5)/*coutPullPrio*(1)], [*c_pull*(6)/*coutPullPrio*(2)]) [2].

Notice the mapping of Push_Prio2 to Push_Rep3 and the mapping of Pull_Prio2 to Pull_Rep3 respectively. One variable to do the mapping to the relevant input and output of these connectors is enough because only one output or input of the prioritizing connectors is active at each execution cycle. Moreover, these connectors model the higher priority of master M1. Fifo2 helps store data sent by the lower priority master M2 until this data can be received by Push_Prio2. Excerpt of this final system specification called System is given in Figure 22.

The guaranteed global communication properties between the subsystems of System are as follows:

- The data received by all the slaves is equal to the data sent by either one of the masters, if the same master has not initialized a new *push* data transfer.
- The data received by either master is equal to the data sent by the slave which acknowledged first a *pull* data transfer.

The invariant of Figure 22 formalizes the above properties. There is not extra logic to be added, hence actions are not needed. The initialization preserves the invariants of the subsystems as well as the invariant of System ensuring its internal consistency. Moreover, the internal consistency

Figure 22. Excerpt of the specification of final system

```
sys System( )
|[ subsys M1 || M2 || S1 || S2 || S3 || Fifo2 || Push_Prio2 || Push_Rep3 ||
        Pull_Rep3 || Pull_Prio2
  var c_push(1..7), c_pull(1..5) : boolean; d_push(1..7), d_pull(1..5) : data
  invariant ((∀i. 5≤i≤7 ∧ c_push(i) ∧ ¬c_push(4) ∧ ¬c_push(1) ⇒
                                        (d_push(i) = d_push(1))) ∨
            (∀i. 5≤i≤7 ∧ c_push(i) ∧ ¬c_push(4) ∧ ¬c_push(3) ∧ ¬c_push(2) ⇒
                                        (d_push(i) = d_push(2)))) ∧
            ((∨i. 1≤i≤3 ∧ ¬c_pull(5) ∧ ¬c_pull(4) ∧ ¬c_pull(i) ⇒ (d_pull(5) = d_pull(i))) ∨
             (∨i. 1≤i≤3 ∧ ¬c_pull(6) ∧ ¬c_pull(4) ∧ ¬c_pull(i) ⇒ (d_pull(6) = d_pull(i))))
  init  c_push(1), …, c_push(7), c_pull(1), …, c_pull(5) := false;
        d_push(1), …, d_push(7), d_pull(1), …, d_pull(5) := d.d : data ]|
```

of System is ensured by the internal consistency of its subsystems.

The proofs of this example, according to the theorem of Section 2, were processed manually but it would not be difficult to create B Action Systems (Waldén & Sere, 1998) specifications and use the commercial tool AtelierB (2009) of the B Method (Abrial, 1996) to discharge automatically all the proof obligations. This route was taken for the application of our compositional method to the modeling of an asynchronous Network–on–Chip routing scheme (Tsiopoulos & Waldén, 2006), but for the work presented in this paper we preferred the greater flexibility of Action Systems.

Depending on the application extra logic may be added to the connectors and the system modules such as System above through the actions–clause. Moreover, different algorithms on the data transfers may be applied depending on system requirements. We have shown how already specified components can be instantiated and composed together in a formal manner to achieve the desired system functionality.

5. RELATED WORK

A framework which influenced our approach is *Reo* (Arbab & Mavaddat, 2002). It is a recently introduced channel–based coordination model, wherein complex coordinators, called connectors, are compositionally built out of simpler ones. The connectors coordinate components and both the connectors and the components are distributed and mobile. However, they only rely on the functionality of the subsystems and their topological interconnection, while our framework allows us to add and verify extra functionality (through the actions, exec and invariant–clauses) in the composed system for controlling the subsystems.

In (Razavi & Sirjani, 2006) the authors presented a component–based approach to specify and verify system level designs. Hardware components are specified in SystemC and they are translated to constraint automata. The interconnections of the hardware components are created using Reo channels and connectors which in turn are translated to constraint automata, too. The whole system is then modeled as constraint automata which could be finally formally verified with a tool. This tool is currently under construction. We propose a different development approach wherein we start first with the formal specification and verification of systems and their interconnection to avoid errors at early development phases, being able to translate later our verified formal models to hardware implementation languages (Metsälä, Westerlund, & Plosila, 2009). Recently

we presented a mapping approach of hierarchical formal models in B Action Systems to hardware implementation languages through Model Based Testing (Tsiopoulos & Satpathy, 2009).

Peeters (1996) specified networks of asynchronous handshake components called handshake circuits. Various push and pull channels were used to connect these components together. Furthermore, Peeters' models were focused on implementation issues of the programming language *Haste* (Handshake Solutions, 2009) for asynchronous circuit design and synthesis following a top–down approach, whereas we concentrate on the specification of modular and distributed connectors for modeling the underlying network for the control flow and data propagation between components on an MPSoC following a component–based approach.

There is a considerable amount of work on formal MPSoC development which is dedicated on the verification and debugging of existing informal communication protocol specifications from different providers (Roychoudhury, Mitra, & Karri, 2003) without presenting a new communication framework as we present in this paper. Generally *model checking* has been used to verify existing MPSoC buses.

Petri Nets have extensively been used for formal modeling and verification of communication frameworks through simulation. The authors in (Pelz & Tutsch, 2007) presented compositional high–level Petri Nets models, in a form of multistage interconnection networks, for multicast traffic in NoC architectures. They simulated the models creating multicast traffic patterns to have a basis for performance evaluation and comparison of various architectures. Similarly we can create several different network architectures by instantiation of modular connectors and simulate traffic patterns at the formal specification level within ProB (Leuschel & Butler, 2005) which is a model checking and animating tool for the B Method and consequently for B Action Systems. A first description of the approach can be found in (Tsiopoulos, 2009).

Several novel architectures have been presented to overcome the shortcomings of bus–based interconnection schemes as well as improve regular NoC interconnects. For example, the authors of (Grecu, Pratim Pande, Ivanov, & Saleh, 2004) presented a *butterfly fat–tree* topology for scalable communication–centric SoC interconnect architectures. The authors of (Tavakkol, Moraveji, & Sarbazi–Azad, 2008) presented a *mesh connected crossbar* architecture to enhance the communication bandwidth between processing elements by adding diagonal links in regular NoC mesh topologies sacrificing simplicity. We can model and verify such topologies within our framework achieving scalability through the modular connectors and their instantiation. Moreover, other related software architectures which bring at the same level of importance components and connectors can be found elsewhere (Shaw & Garlan, 1996).

6. CONCLUSION

In this paper we presented a new compositional approach for the specification of generic connectors for formal MPSoC modeling. More specifically, the communication infrastructure consists of a distribution of different channels composed into connectors and further distribution of connectors. Thus, arbitration and data transfers can be taken care of by a combination of the following: (a) the composed functionalities of the instantiated channels, (b) the connector's own actions and (c) the topological placement of these channels when forming connectors. We believe that this approach facilitates the modeling of complex MPSoCs. Moreover, reusability of already defined and formally proven modular components is of most importance allowing rapid change of communication requirements. Only five different abstract channel specifications were presented, but examples of their overall functionality when

composed into connectors were shown in the case study.

We aimed at presenting what could be a realistic view of the connections between master and slaves modules in an MPSoC and showed how asynchronous coordination can be modeled within our approach. An obvious direction for future work is of course tool support. For this we use the tools for Action Systems (Plosila, Sere, & Waldén, 2005), but envisage a graphical UI for our purposes. Moreover, our interest is on how this formalism would help the design of complicated regular nanoscale structures. A first result on this application has been presented in (Tsiopoulos & Waldén, 2006).

Our future work will be on modeling different channels and different generic connectors that can form a library of channels and connectors for MPSoC design. Implementations can then be tested against such formal models to aid system verification and help to avoid system errors before the final hardware synthesis takes place. A first result on this approach can be found in (Tsiopoulos & Satpathy, 2009). For this we are going to map our specifications to hardware description languages such as *Haste* (Handshake Solutions, 2009). Furthermore, performance evaluation and analysis as well as efficient application mapping through simulation of data traffic patterns between processors and memory units on an MPSoC at the formal specification level is of high interest. For this we are going to use the tool ProB. A first description of the approach can be found in (Tsiopoulos, 2009).

REFERENCES

Abrial, J.-R. (1996). *The B–Book: Assigning Programs to Meanings*. Cambridge, UK: Cambridge University Press. doi:10.1017/CBO9780511624162

Arbab, F., & Mavaddat, F. (2002). Coordination through channel composition. In *Proceedings of Coordination Languages and Models* (LNCS 2315, pp. 22-39). New York: Springer Verlag.

ARM. (2005). *AMBA*. Retrieved from http://www.arm.com/products/solutions/amba2overview.html

Atelier, B. (2009). *Home*. Retrieved from http://www.atelierb.societe.com/

Back, R. J., & Kurki-Suonio, R. (1983). Decentralization of Process Nets with Centralized Control. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing* (pp. 131-142).

Back, R. J., Martin, A. J., & Sere, K. (1995). An action system specification of the Caltech asynchronous microprocessor. In *Proceedings of the 3rd International Conference: Mathematics on Program Construction (MPC'95)*, Kloster Irsee, Germany (pp. 159-179).

Back, R. J., & Sere, K. (1996). Superposition Refinement of Reactive Systems. *Formal Aspects of Computing*, *8*(3), 324–346. doi:10.1007/BF01214918

Back, R. J., & von Wright, J. (2003). Compositional Action System Refinement. *Formal Aspects of Computing*, *15*(2-3), 103–117. doi:10.1007/s00165-003-0005-6

Chapiro, D. M. (1984). *Globally–asynchronous locally–synchronous systems*. Unpublished PhD thesis, Stanford University, Stanford, CA.

Desel, J., & Juhás, G. (2001). What is a Petri Net? Informal Answers for the Informed Reader. In H. Ehrig et al. (Eds.), *Unifying Petri Nets* (LNCS 2128, pp. 1-25).

Dijkstra, E. W. (1976). *A Discipline of Programming*. Upper Saddle River, NJ: Prentice Hall.

Grecu, C., Pratim Pande, P., Ivanov, A., & Saleh, R. (2004). A Scalable Communication–Centric SoC Interconnect Architecture. In *Proceedings of the 5th International Symposium on Quality Electronic Design* (pp. 343-348).

Handshake Solutions. (2009). *Haste.* Retrieved from http://www.handshakesolutions.com/

Hemani, A., Jantch, A., Kumar, S., Postula, A., Öberg, J., Millberg, M., & Lindqvist, D. (2000). Network on a Chip: An architecture for billion transistor era. In *Proceedings of the IEEE NorChip Conference*.

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Upper Saddle River, NJ: Prentice Hall.

Lahiri, K., Raghunathan, A., & Dey, S. (2001). Evaluation of the traffic–performance characteristics of system–on–chip communication architectures. In *Proceedings of 14th International Conference on VLSI Design* (pp. 29-35).

Leuschel, M., & Butler, M. (2005). ProB: A Model Checker for B. In *Proceedings of FME 2003* (LNCS 2805, pp. 855-874). New York: Springer Verlag.

Metsälä, T., Westerlund, T., & Plosila, J. (2009). *Introducing Action Systems Class Hierarchy to SystemC Modeling* (Tech. Rep. No. 946). Turku, Finland: Turku Centre for Computer Science (TUCS).

Nakajima, M., Yamamoto, T., Ozaki, S., Sezaki, T., Kanakagi, T., Furuzono, T., et al. (2002). A 400MHz 32b embedded microprocessor core AM34–1 with 4.0GB/s cross-bar switch for SoC. *IEEE International Solid–State Circuits Conference, 1*, 342-472.

Pasricha, S., Ben–Romdhane, M., & Dutt, N. D. (2007). BMSYN: Bus matrix communication architecture synthesis for MPSoC. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *26*(8), 1454–1464. doi:10.1109/TCAD.2007.891376

Peeters, A. (1996). *Single–Rail Handshake Circuits*. Unpublished PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands. Retrieved from http://www.win.tue.nl/async-bib/pdf/Peeters96.pdf

Pelz, E., & Tutsch, D. (2007). Formal Models for Multicast Traffic in Network on Chip Architectures with Compositional High–Level Petri Nets. In J. Kleijn & A. Yakovlev (Eds.), *Proceedings of ICATPN 2007* (LNCS 4546, pp. 381-401).

Plosila, J., Seceleanu, T., & Sere, K. (2004). Formal Communication Modeling and Refinement. In Nurmi, J., Tenhunen, H., Isoaho, J., & Jantsch, A. (Eds.), *Interconnect-Centric Design for Advanced SoC and NoC. Alphen aan den Rijn*. The Netherlands: Kluwer.

Plosila, J., Sere, K., & Waldén, M. (2005). Asynchronous System Synthesis. *Science of Computer Programming*, *55*, 259–288. doi:10.1016/j.scico.2004.05.018

Razavi, N., & Sirjani, M. (2006). Using Reo for Formal Specification and Verification of System Designs. In *Proceedings of Fourth IEEE/ACM International Conference on Formal Methods and Models for Co–Design*.

Roychoudhury, A., Mitra, T., & Karri, S. R. (2003). Using Formal Techniques to Debug the AMBA System–on–Chip Bus Protocol. In *Proceedings of the Conference on Design, Automation and Test in Europe* (Vol. 1, pp. 10828-10834).

Sekerinski, E., & Sere, K. (1996). A Theory of Prioritizing Composition. *Oxford Journals -. The Computer Journal*, *39*(8), 701–712. doi:10.1093/comjnl/39.8.701

Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, NJ: Prentice Hall.

Tavakkol, A., Moraveji, R., & Sarbazi–Azad, H. (2008). Mesh Connected Crossbars: A Novel NoC Topology with Scalable Communication Bandwidth. In *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications* (pp. 319-326).

Tsiopoulos, L. (2009). Towards Dependable Placement of NoC Resources. In *Proceedings of NODES '09 workshop* (pp. 1-9).

Tsiopoulos, L., & Satpathy, M. (2009). Model Based Testing of a Network–on–Chip Component. [ENTCS]. *Electronic Notes in Theoretical Computer Science*, *253*(2), 101–116. doi:10.1016/j.entcs.2009.09.054

Tsiopoulos, L., Sere, K., & Plosila, J. (2006). *Modeling SoC Systems with Flow Control* (Tech. Rep. No. 746). Turku, Finland: Turku Centre for Computer Science (TUCS).

Tsiopoulos, L., & Waldén, M. (2006). Formal Development of NoC Systems in B. *Nordic Journal of Computing*, *13*(1-2), 127–145.

Waldén, M., & Sere, K. (1998). Reasoning about Action Systems Using the B Method. *Formal Methods in System Design*, *13*(1), 5–35. doi:10.1023/A:1008688421367

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 2, edited by Seppo Virtanen, pp. 23-44, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 13
Generating Process Network Communication Infrastructure for Custom Multi-Core Platforms

Peter Sørensen
Technical University of Denmark, Denmark

Jan Madsen
Technical University of Denmark, Denmark

ABSTRACT

We present an approach for generating implementations of abstraction layers implementing the communication infrastructure of applications modeled as process networks. Our approach is unique in that it does not rely on assumptions about the capabilities and topology of the underlying platform. Instead, a generic implementation is adapted to the particular platform based on information retrieved from analyzing the platform. At the heart of the approach is a novel method for analyzing the capabilities of custom execution platforms composed of components. The versatility and usefulness of the approach and analysis method is demonstrated through a case study.

1. INTRODUCTION

Modern embedded applications are increasingly relying on heterogeneous multi-processor platforms to provide the computational power and flexibility needed to meet the ever increasing demands on performance, features and reliability. By designing a platform towards executing a specific application it is in many cases possible to achieve a more efficient system in terms of performance, power consumption and resource usage than what could have been achieved using a more general-purpose platform. Designing such platforms is expensive, time consuming and in many cases infeasible because of budget limita-

DOI: 10.4018/978-1-4666-0912-9.ch013

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

tions and/or tight time-to-market constraints. In recent years, there has been an increasing interest in system-level design tools based on component-based design methodologies from academia (Bergamaschi et al., 2000; Nikolov, Stefanov, & Deprettere, 2008) as well as industry (Altera, 2009; Xilinx, 2009). These tools allow the designer to assemble complex platforms from libraries of existing hardware and software components. The task of designing a platform consists of deciding on which components to use, how to configure them and, finally, how to connect them. Together, the resulting component *allocation* (choice of components), component *configuration* and platform *topology* (organization of the components) determines how well suited the platform is for executing the application.

Compared to single processor systems, heterogeneous multi-processor systems are difficult to program. In the absence of a platform-wide programming solution, shielding the application developer from the details of the platform, the developer must program each processor individually. This is problematic for several reasons: First, the application becomes dependent on the particular platform making re-use hard and changes in the platform, application and/or mapping difficult. Second, a partitioning and mapping of the application must be decided upon before any code is written which leaves little room for errors and may lead to resource waste due to over-engineering. Finally, in addition to implementing the application, the application developer must also worry about low-level processor-to-processor interfacing that in turn will require intimate knowledge of the platform in particular and low-level programming in general.

Through the use of a parallel programming model, for example, TTL (van der Wolf, de Kock, Henrikson, Kruijtzer, & Essink, 2004), YAPI (de Kock et al., 2000), and OpenMP (www.openmp.org), it is possible to abstract away different aspects of the platform, such as communication and synchronization, and thus effectively de-couple the application from any actual platform (Jerraya, Bouchhima, & Frédéric, 2006; Paul, 2003; Martin, 2006). A programming model can be provided as an integral part of a programming language or as an add-on in the form of an application programming interface (API). In most cases, a given platform does not directly implement a given programming model and thus an abstraction layer, realizing the abstractions of the programming model, is needed. An application layer can be application-specific so that it only provides support for the functionality that is actually used by a particular application. The opposite of an application-specific layer is a general purpose layer where functionality is provided to support a broader range of applications.

Compared to the cost of designing a full custom ASIC platform the cost of developing an associated abstraction layer is negligible. However, in the context of system-level design tools, where a multi-processor platform can be designed (and instantiated on an FPGA) in a matter of hours, the cost of developing an abstraction layer is taking up a relativity larger portion of the design budget and thus becomes an important factor to consider in any attempt at further optimizing the design process for customized FPGA-based embedded systems.

Problem Description

For embedded systems, programming models based on the *Kahn Process Network* (KPN) (Kahn, 1974) model of computation (MoC) have been studied intensively. An application modeled as a KPN consists of processes communicating and synchronizing using unidirectional FIFO channels. In theory, the capacity of the channels is unbounded and writing to them is a non-blocking operation. In this work, we will consider a more practical version with bounded channel capacity and blocking write that we will refer to simply as a *process network*. The process network MoC is especially well suited for capturing streaming (e.g. image processing) applications and has previ-

ously been shown to be a suitable representation for efficiently mapping applications onto multi-processor platforms (Nieuwland, et al., 2004; Todor, Zissulescu, Turjan, Kienhuis, & Deprettere, 2004; Piementel, Hertzberger, Lieverse, van der Wolf, & Deprettere, 2001).

In this article, we will focus on the problem of automatically generating an implementation of the communication infrastructure (the channels) of a process network targeting custom execution platforms where the processes of the process network have been mapped to programmable processors and the channels to memories. This has been studied previously for restricted classes of platforms (Nikolov, Stefanov, & Deprettere, 2008). More specifically, the topologies of the target platforms are often required to be fixed such that inter-process communication is always possible and easy to realize by, for example, assuming that all processors are connected to the same global interconnect. In the case of custom platforms, nothing can be assumed about the component allocation, configuration and platform topology. This means that the scheme used to implement the channels of the process network cannot be based on assumptions about the platform and, instead, must be based on information retrieved from analyzing the particular platform. Also, it is not given that a particular mapping of an application onto a platform is in fact feasible (can be realized) either because of limitations in the platform, the implementation scheme or a combination of the two.

As an example of some of the problems involved in working with arbitrary platforms Figure 1 shows two alternative mappings of a simple process network consisting of two processes (p_0, p_1) communicating through a single channel (c_0) to a platform consisting of two processors (pe_0, pe_1) and two memories (mem_0, mem_1). The topology of the platform is such that pe_0 can access both memories where as pe_1 can only access memory mem_1. The mapping shown in **a)** is considered feasible because both processes can access mem_1. The mapping shown in **b)**, however, is infeasible because p_1 cannot access memory mem_0 since pe_1 is not connected to bus_0 as a master.

Figure 2 shows the context in which we consider the problem. A process network application consists of a number of processes interacting using an API, acting as an abstraction layer between the platform and the application, exposing the process network programming model. A platform consists of a number of programmable processors and memories connected to each other using other re-usable components. The API is tied to the platform by means of a *platform abstraction layer* (PAL) that provides a realization of the API for the particular platform.

Rather than focusing on attempting to generate a platform abstraction layer for process networks in general we are interested in application-specific implementations where the generated

Figure 1. Two different mappings of a simple process network: **a)** *a feasible mapping, both processes can access c_0 and* **b)** *an infeasible mapping, process p_1 cannot access memory mem_0 where c_0 is mapped.*

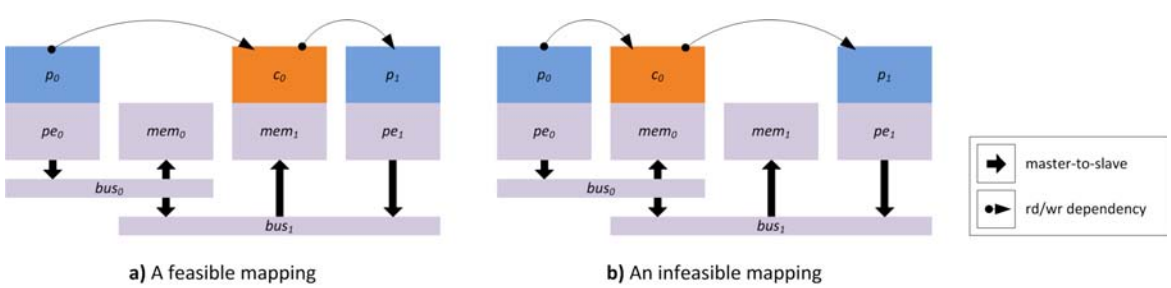
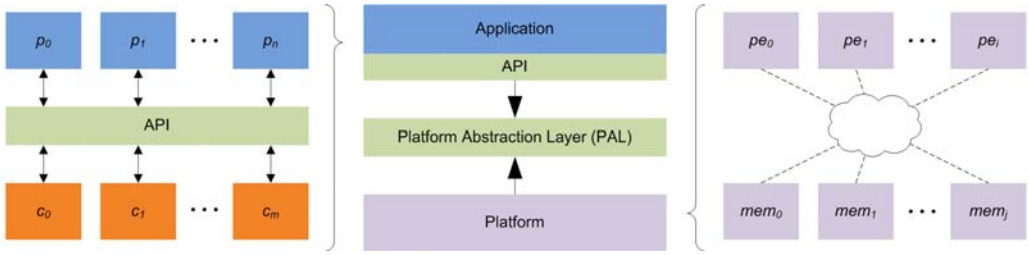


Figure 2. Application, platform abstraction layer and platform.



layer supports exactly the features used by the process network in question. An application-specific implementation is more efficient because it only has to support a fixed number of processes and channels and does not have to provide the means for any two processors of the platform to communicate – which may not necessarily be possible.

Contributions

In this article, we present an approach for automatically generating an application specific platform abstraction layer. The abstraction layer implements the process network communication infrastructure of an application that has been mapped onto a custom platform built from re-usable components.

We have developed an analysis method which can be used to 1) discover the possibilities of a platform in terms of (low-level) services available at various points of interest and 2) gather information useful for synthesizing access to such services. This information is used for deriving concrete implementations of the channels on the basis of the before mentioned implementation scheme. The method requires that the components of the platform have been characterized using a formalism called the *service relation model*. We consider the service relation model formalism and the associated analysis method the main contribution of this work.

Finally, we have developed a procedure for determining if a particular user-provided mapping of a process network onto a platform is feasible or not and, in the event that it is not, to attempt to extend the process network and mapping into a feasible solution by adding additional processes and channels to the process network. This procedure also uses our analysis method as a back-end. We consider this procedure an important contribution of this work.

Scope of Work

In this work, we only consider static process networks with bounded FIFO channels where both reading and writing operation have blocking semantics. Furthermore, we assume that each channel has exactly one writer and one reader. The scheme we use for implementing channels requires all channels to be mapped to memories. This means that our tool cannot exploit any existing hardware or software channels in the platform. Also, our approach is limited to platforms where only programmable processors, supported by C compilers, are used as processing components. Finally, we assume that a service relation model characterization of the target platform is available. Deriving such a description for a platform is currently a manual undertaking.

Article Overview

This article is organized as follows: Section 2 briefly surveys related work. In section 3 we give an overview of the design flow in which we envision a tool based on our approach to be used. In section 4, the channel implementation scheme is

presented. The service relation model formalism and the associated analysis and synthesis methods are described in sections 5 and 6. In section 7, we present a procedure for testing the feasibility of a mapping and correcting infeasible mappings by extending the process network. Section 8 contains a case study and, finally, section 9 the concluding remarks.

2. RELATED WORK

In (van der Wolf, de Kock, Henrikson, Kruijtzer, & Essink, 2004) the authors present a programming model called *Task-Transaction Level* (TTL). TTL is based on a process network MoC where tasks (processes) communicate with each other using bounded FIFO buffers (channels). In their approach, the API implementing the TTL interface is manually implemented on a per-platform basis. The channel implementation scheme used by our tool is inspired by the scheme used in the TTL implementation described in (van der Wolf, de Kock, Henrikson, Kruijtzer, & Essink, 2004). TTL includes a number of different communication and synchronization primitives in addition to the (simple) blocking read and write supported by our tool.

The *StepNP* customizable platform and its associated *MultiFlex* multi-processor programming environment, (Paulin, Pilkington, Langevin, Bensoudane, & Nicolescu, 2004), is an example of a combination of a platform and a tool that generates abstraction layers. The tool is capable of generating abstraction layers suitable for a symmetric multi-processing (SMP) model using shared memory and a distributed system object (DSOC) message passing programming model. This approach is limited by the underlying assumption that all processors in the system are connected to the same central interconnect. Also, it relies on platform-provided services in the form of hardware accelerators for message passing and task scheduling.

The tool *Embedded System-level Platform synthesis and Application Mapping* (*ESPAM*) spans the entire design process from application and platform specification to FPGA implementation (Nikolov, Stefanov, & Deprettere, 2008). The tool's very elaborate front-end allows applications to be specified in a restricted subset of the C programming language. *ESPAM* can extract possible parallelism, in the form of a KPN, from the sequential C program. Given the KPN the *ESPAM* tool is capable of determining an upper bound on the buffer size for each channel in the network such that deadlocks will be avoided without the use of detection and recovery mechanisms. The resulting process network can be mapped to multi-processor platforms also created using *ESPAM*'s platform specification and generation tools. The *ESPAM* tool allows the designer to assemble multi-processor platforms using a fairly small set of predefined components. Compared to our approach, the *ESPAM* tool offers only three possible platform topologies: crossbar, shared bus, and point-to-point network. In the case of the point-to-point network, the topology of the network must match that of the application. Also, *ESPAM* requires special communication controllers and memories to be integrated into the platform at key locations to implement the channels. *ESPAM* uses the *Xilinx Embedded Development Kit* as a backend.

Gajski et al. proposes a methodology and algorithm for the automatic refinement of abstract communication channels into actual bus-level implementations (Abdi, Shin, & Gajski, 2003). This work is focused on generating wrappers for a single communication component. In contrast, we focus on the problem of generating channel implementations spanning an entire platform.

As part of their ongoing research into system-level design methodologies, Jerraya et al. proposes a design flow where an application is decomposed into a set of virtual components connected through a virtual interconnect (Cesario et al., 2002). A high-level parallel programming model is used to

de-couple these components from each other and to abstract away platform details. Much of their effort has been spent trying to automate the generation of wrappers used to connect components with an actual interconnect. Interestingly, they use a model called the *Service Dependency Graph* to synthesize wrappers between components (Kriaa, Bouchhima, Youssef, Petrot, Fouillart, & Jerraya, 2006; Sarmento et al., 2005). Despite the name, the service dependency graph model has little in common with the service relation model presented in this article. The Service Dependency Graph is used to capture service requirements and dependencies amongst the sub-components of an interface connecting an implementation of a virtual component to an actual interconnect. The Service Relation Model, on the other hand, is used to capture the flow and availability of services in an entire system composed of components.

The literature contains many examples of other component models that can be used to analyze various properties of systems composed of components, for example, Acme (Garlan, Monroe, & Wile, 2000), Koala (van Ommering, van der Linden, Kramer, & Magee, 2000). The service relation model differs from these models in its ability to cope with hardware, middleware and application-specific software components uniformly and, most notably, in its intended use.

The two leading FPGA vendors, *Xilinx* (2009) and *Altera* (2009), both market platform design tools incorporating the component-based design approach. They do enable the designer to construct rather complex platforms by providing rich libraries of both hardware and software components. These tools do not provide any platform-wide programming solution. We envision tools based on the approach described in this article to be used in combination with such a tool.

3. DESIGN FLOW

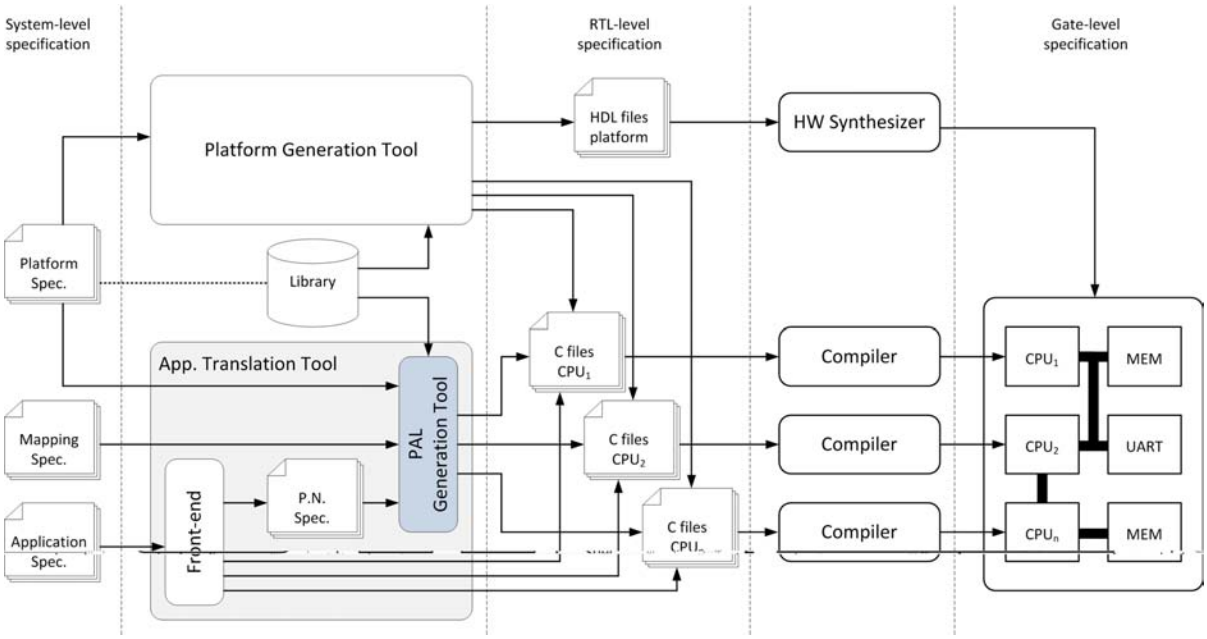
Figure 3 shows a design flow supported by our approach. The design is shown at three different levels of abstraction: the system-level, the register transfer level and the gate level. At the system-level the design consists of an application specification, a platform specification and a mapping between the two. The platform is a composition of components taken from a library of existing hardware and software components. The platform specification specifies which components are used, how they are configured and connected. The application is captured in a high-level language supporting the process network MoC (e.g. TTL) and, conceptually, consists of a number of concurrent processes communicating and synchronizing using unidirectional FIFO channels. Finally, the mapping consists of associations between the processes and channels in the application and processors and memories in the platform.

The platform is refined into a register transfer level description using a platform generation tool (*Xilinx PlatformStudio* for example). The result of running this tool on the platform specification is a set of hardware description language (HDL) files implementing the hardware and a set of C files for each programmable processor implementing the platform software. The HDL files are later processed by a hardware synthesis tool to obtain a gate-level implementation of the hardware.

Using an application translation tool and the mapping, the platform specification and the platform independent application specification is refined into a platform dependent implementation consisting of a set of C files for each programmable processor in the mapping specification. For each processor in the design, the C files generated by the application translation and platform generation tools for the processor can be combined and compiled to obtain the final program images.

A tool based on the approach presented in this article is intended to be used as part of the

Figure 3. A possible design flow supported by our approach (PAL generation tool).



application translation process embedded within the application translation tool of Figure 3. This tool will require three inputs: a description of the platform given in the service relation model formalism (described in Section 5), a description of the process network embedded in the application and, finally, a mapping of application processes onto processors in the platform and channels onto memories. Given these inputs, the tool produces an implementation of the applications (global) communication infrastructure.

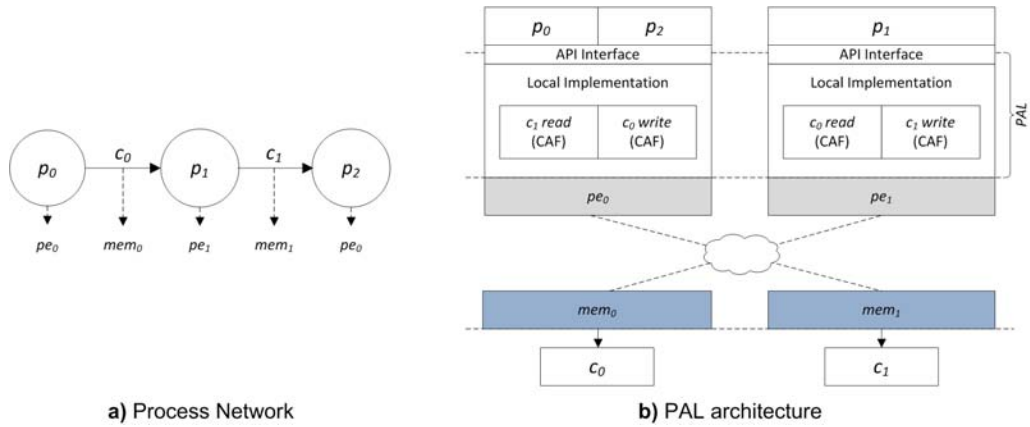
4. LAYER ARCHITECTURE AND CHANNEL IMPLEMENTATION

In this section, we present the architecture of the generated platform abstraction layers and the implementation scheme used by our approach for implementing channels.

The generated process abstraction layer (PAL) is organized as a set of *local implementations* as shown in Figure 4. Each processor in the platform is associated with a local implementation that services all application processes mapped to that processor though a simple API.

Conceptually, the communication infrastructure of a process network application consists of a set of channels accessed by means of read and write primitives with blocking semantics. A channel can be implemented in many different ways. In this work, we will consider channels realized as circular buffers in memory accessed from programmable processors. A buffer is organized as a series of equally sized tokens each of which is capable of containing a single message. In the generated implementation, a channel consists of a *channel buffer*, two *channel control* variables (read and write indices) and two *channel access functions* (CAF). The channel buffer and the channel control are pure data entities that are placed in memory in accordance with the user-specified mapping. The two channel access functions, a *read function* and a *write function*, provide implementations of the channel access protocol. The read function is part of the local implementation associated with the processor where the application process writing to the channel is mapped.

Figure 4. Architecture of the generated platform abstraction layers.



Similarly, the write function is part of the local implementation associated with the processor where the application process reading from the channel is mapped.

Channel Access Protocol

The read and write channel access functions implementing the access protocol follow the same pattern as shown in Figure 5: First, the functions check if the channel is ready (not empty or not full) by reading and comparing the control variables. If the channel is not ready the functions will loop waiting for the status of the channel to change. Once ready, the functions will proceed with the actual copying of data to or from the channel buffer. When completed, the functions update the control variables.

The actual reading and writing of memory is hidden by means of a set of functions (e.g. *get_wr_idx()* and *set_wr_idx()* for reading and writing the write index member of the channel control). In order to achieve a concrete implementation of a channel we need to provide implementations of these functions. Deriving implementations of these functions is not necessarily a trivial matter because their content is dependent on the platform and the mapping. A method for doing this is presented in Section 6.

Figure 5. Read and write functions synchronized using polling. The actual access to memory is hidden by means of the functions get_wr_idx(), set_wr_idx(), get_rd_idx(), set_rd_idx(), get_buffer_offset(). The actual copying of data is done by means of two functions copy_to_token() and copy_from_token(). The number of tokens and size of each token is given by means of the two definitions TOK_CNT and TOK_SIZE.

```
0 void write_c1(void* data) {
1   /* wait for store to be not full */
2   while(((get_wr_idx() + 1) & TOK_CNT) == get_rd_idx());
3
4   copy_to_token(
5     get_buffer_offset(), get_wr_idx(), data, TOK_SIZE
6   );
7
8   /* update write index */
9   set_wr_idx((get_wr_idx() + 1) % TOK_CNT);
10 }
```

a) Write access function

```
0 void read_c1(void* data) {
1   /* wait for store to be not empty */
2   while(get_wr_idx() == get_rd_idx());
3
4   copy_from_token(
5     get_buffer_offset(), get_rd_idx(), data, TOK_SIZE
6   );
7
8   /* update read index */
9   set_rd_idx((get_rd_idx() + 1) % TOK_CNT);
10 }
```

b) Read access function

The channel access protocol is inspired by a similar protocol used in a TTL implementation for a Philips DSP platform, (van der Wolf, de Kock, Henrikson, Kruijtzer, & Essink, 2004). A key feature of the protocol is that it ensures safe concurrent access to the channel buffer without the use of synchronization primitives such as locks or semaphores (Nieuwland, et al., 2004). If a protocol requiring such synchronization primitives had been used then these must necessarily be provided by the platform (in sufficient quantities) that would dramatically limit the range of platforms supported by our tool. In order to be able to distinguish between a full and an empty buffer the protocol requires that the buffer is extended with an additional token which is never used.

5. SYSTEM/PLATFORM ANALYSIS

In this section, we introduce the service relation model. The service relation model is a formalism used to model flow and availability of services in systems composed of (re-usable) components. Our approach uses a description of the target platform given in this formalism for analyzing the capabilities of the platform and to generate the platform abstraction layer.

In the service relation model, a system is viewed as a network of components providing and forwarding services. We distinguish between two kinds of components: component templates and component instances. The service relation model consists of two related formalisms: one used for characterizing component templates and one used for describing systems composed of component instances. In this article, we will focus exclusively on explaining the latter formalism.

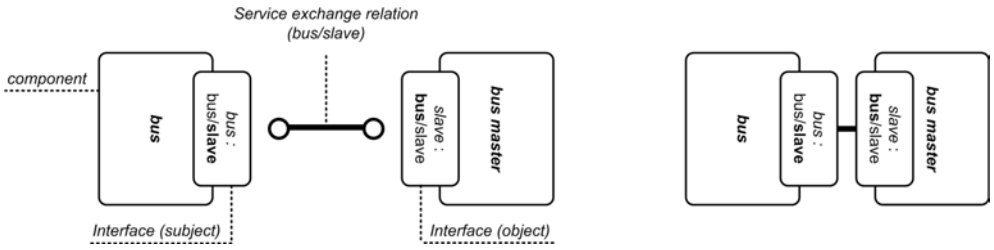
The Service Relation Model (SRM) Formalism

Most models supporting component-based design of execution platforms employs the basic concepts of components, interfaces (ports) and connections. In these models, a component represents a physical entity such as a processor, bus or a block of memory. Likewise, an interface represents a physical connection point where a component may be connected to another component with a matching interface. The *service relation model* extends this line of thought to also include conceptual components and conceptual connections (called *relations*). In other words, the components of the service relation model need not necessarily have a physical manifestation and their relationships (connections) can be conceptual as well as physical.

The components of the service relation model are defined by the service or services they offer to other components. A service can be any kind of work that a component may carry out on behalf of another component. As an example consider a bus: the ability of a bus master attachment to issue read and write requests to slave attachments can be seen as a service provided by the bus to the master attachment. A component representing a function in the C programming language might be considered to provide other (software) components with the service that it describes.

Components, in the service relation model, are related to each other using a special class of binary relations called *service exchange relations*. A service exchange relation is characterized by facilitating some kind of exchange of services between two components. The relation between a bus master attachment and a bus is an example of such a relation since the bus master attachment may request service from the bus. Similarly, a software component may have a service exchange relationship with its processor because it, conceptually, is serviced by the processor. On the other hand, a relation stating that two components are identical is not a service exchange relation because no exchange of service can be attributed to it. A service exchange relation consists of three parts: the relation itself and two roles (the subject and the object). A component in the service relation

Figure 6. Interfaces and service exchange relations. The name of the interface and the type of service exchange relation supported is printed inside the box representing the interface. The role associated with the interface is shown in **bold**.



model exposes its services through a set of interfaces each of which is associated with exactly one role of a predefined service exchange relation. Two components may be connected to each other, using a service exchange relation, only if one of the components defines an interface with the subject role of the relation and the other define an interface with the object role of the relation (see Figure 6).

Besides the service exchange relation the model supports two additional relations: *service import relations* and *service export relations*. In contrast to service exchange relations, the service import and export relations are used inside components to relate the services and interfaces. An export relation states that a given service is exported to a given interface. Recall that we said that the services of a bus are provided to its master attachments. In the service relation model, this is modeled by an export relation relating the bus services with the interfaces representing the bus role of a bus-master/bus service exchange relation. A service exported to an interface is said to be *available* at that interface. Services available at an interface are also considered to be available at the interface of another component if the two interfaces are (properly) connected by means of a service exchange relation.

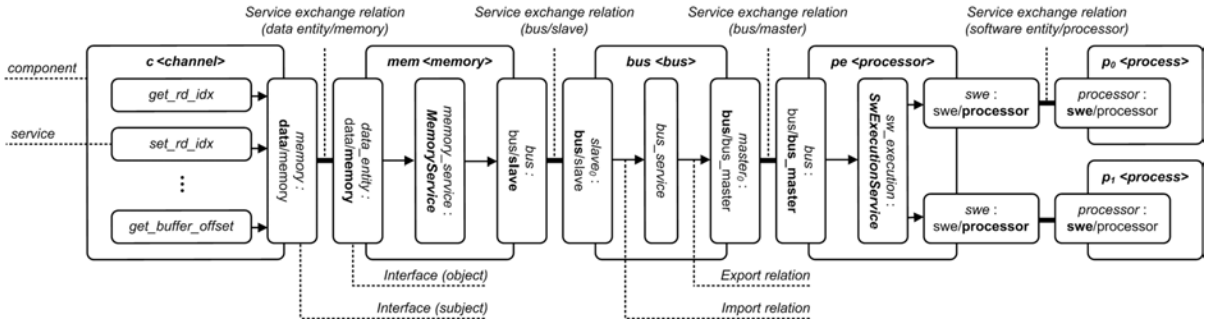
The import relation facilitates *service aggregation* which essentially is an "accessible through" relationship between any services available at an interface and a service defined in the component. A service representing the ability to execute software on a processor can be said to aggregate services of its attached data bus since the services of the bus may be invoked using the memory load and store instructions of the (RISC) processor. A driver is another example of service aggregation. The service offered by a driver aggregates the service offered by the underlying device. A service aggregated by another service is considered available at that service and at any interfaces and services where the aggregating service is considered available.

Figure 7 shows a graphical representation of a SRM description of a simple system consisting of a processor, a bus, a memory, a channel and two processes. The components are connected by means of four different service exchange relations. This description will be used as an example in the following sections.

Services Hierarchies

The services of a model are organized in a service class hierarchy where each service is a member of one or more service classes. A service class is itself not a service but a grouping of services that share some similarities. All services are implicitly a member of the base service class *"service"*. The purpose of this hierarchy is to attach meaning to the individual services. For example, knowing which classes a service belongs to tells us something about that service and/or its parent component.

Figure 7. A simple system described in the service relation model.



The Structure of Memories, Processors and Software Components

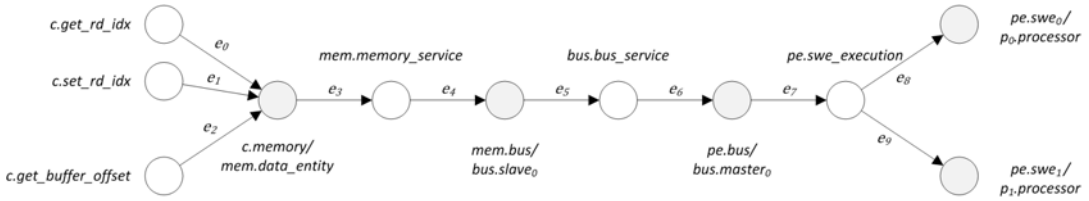
Our approach makes some assumptions about how memories, (programmable) processors and software components are modeled. First we assume that all processors provide a service labeled *sw_execution* declared to be a member of the service class *"SwExecutionService"* representing the processors ability to execute software. Also, we assume that all software components are related to their respective processor using a *software-entity/processor* service exchange relation (abbreviated *swe/processor*) through which the processor exports its *sw_execution* service to the software component. Finally, we assume that all memory components provide a service labeled *"memory_service"* declared to be a member of the service class *"MemoryService"*. These assumptions enable us to easily identify memory, processor and software components and to determine which software components are related to which processors.

Service Availability Analysis

The objective of service availability analysis is to determine the set of services available at each service and interface in a service relation model of a design. For this purpose a more explicit representation of the service flow, called a service flow graph, is extracted from the model. Figure 8 shows the service flow graph of the model depicted in Figure 7.

A service flow graph is an acyclic graph containing three different sets of nodes representing different entities of the original service relation model: A set of service nodes representing the services, a set of unconnected interface nodes representing any unconnected interfaces in the original model and, finally, a set of connected interface nodes representing pairs of interfaces connected using a service exchange relation. The

Figure 8. Service flow graph of the SRM description depicted in Figure 7. White nodes represent services and gray nodes connected interfaces.



nodes are connected to each other using edges representing the service import and export relations of the original service relation model. Extracting the graph is fairly straightforward and will not be discussed here.

Given a service flow graph, a worklist algorithm (see for instance (Nielson, Nielson, & Hankin, 2005)) can be used to compute the sets of services available at each node. The set of services available at a node tells us which services can be invoked by the parent component of the service or interfaces represented by the node.

A service flow graph is not allowed to have cycles. The rationale for disallowing cycles is that they do not show up in practice and that it allows for a simple and more efficient analysis backend. A cycle in the service flow graph would mean that at least one service would be available through itself and we have not yet been able to come up with an example where this is the case.

6. SERVICE INVOCATION SYNTHESIS

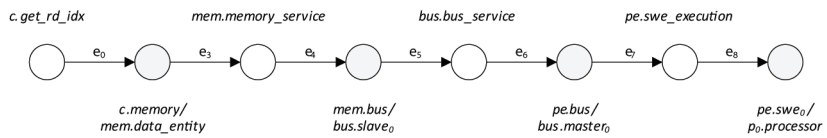
A trace of relations, services and interfaces connecting a service x with another service or interface y where x is available at y in the service flow graph is called a service aggregation chain. A service aggregation chain represents one way to invoke a service offered by a foreign component through a possibly very complex structure of other components. A service aggregation chain can easily be extracted from a service flow graph using backtracking. There might exist multiple different service aggregation chains each representing different ways to invoke a given service at a particular node in the service flow graph.

When a service is available at a location it means that it may somehow be accessed from that location. The service aggregation chain linking a location with an available service tells us which services, interfaces and import/export relations will be involved in accessing the service from that location. This information is not sufficient to actually realize the invocation of the service because we cannot uniquely identify the service amongst other services also aggregated at the nodes of the chain.

The missing information is added to the model by means of the concepts of parameters and invocation functions. A parameter is simply a label/value pair associated with an entity of the service relation model. Parameters can be used to store additional information about the system such as the offset of a slave peripheral in the memory space of a bus component. An invocation function is a function associated with an import or export relation and is used to extract information about how a specific service available at the source node of the relation is to be invoked from the target node. The role of an invocation function is as follows: given information about how to invoke a service s from a node n_1, the invocation function associated with a relation having n_1 as source and node n_2 as target will provide the information needed to invoke s from n_2. In order to compute the desired output, the function can use the input information and any parameters associated with the service and interface represented by the two nodes. The type of input expected and output produced by an invocation function is dictated by the service and interface related by the associated relation. More specifically, the type of output produced and the type of input expected by two functions associated with a pair of relations connected to the same node in a service aggregation chain must be the same. The process of determining how to invoke a service, given a service aggregation chain, consists of applying the invocation functions associated with the relations of the chain to each other in a backwards manner as shown in Figure 9.

As an example, we again consider the simple system of Figure 7. We adopt the view that a channel has its own local address space which is mapped to a segment of the address space of the memory to which the channel is mapped. In this local memory space the control variables are

Figure 9. Service aggregation chain for accessing the service c.get_rd_idx from the p_0.processor interface.



$$I = f_{e_8}(f_{e_7}(f_{e_6}(f_{e_5}(f_{e_4}(f_{e_3}(f_{e_0}()))))))$$

located at address 0x00 and 0x04 and the channel buffer at address 0x08. Also, we assume that the offsets of the channels in the address space of the memories are available as parameters labeled *base_addr* associated with the *data_entity* interfaces of the memories and, similarly, that the offset of the memories in the address space of the bus are available as parameters also labeled *base_addr* associated with the slave interfaces of the bus. The invocation function definitions below enable us to generate C expressions for invoking the services provided by the channels from the process components in Box 1.

Here, the information required to invoke a memory mapped service is encoded using a triple <op, addr, val> where op is one of the two operations *RD* (read) or *WR* (write), *val* a symbol representing the value to be written in case *op* = *WR* and, finally, *addr* is the address to be accessed.

Assuming that the memory is mapped to an address segment of the bus starting at address 0x20000000 and that the channel is located at address 0x200 within the address space of the memory we get the following result for synthesizing access to the *set_rd_idx* service with *val* = *idx*.

```
*((int*)0x20000204) = idx;
```

It is important to note that an invocation function is tied to a particular import or export relation in an SRM description and can be used with any service aggregation chains that include that relation.

Box 1.

$$f_{e_0}(\ ) = \langle RD, 0X00, 0X04 \rangle$$
$$f_{e_1}(val) = \langle WR, val, 0X04 \rangle$$
$$f_{e_2}(\ ) = \langle WR, 0X00, 0X08 \rangle$$
$$f_{e_3}(\langle op, val, addr \rangle) = \langle op, val, addr + src.base_addr \rangle$$
$$f_{e_4}(\langle op, val, addr \rangle) = \langle op, val, addr \rangle$$
$$f_{e_5}(\langle op, val, addr \rangle) = \langle op, val, addr + src.base_addr \rangle$$
$$f_{e_6}(\langle op, val, addr \rangle) = \langle op, val, addr \rangle$$
$$f_{e_7}(\langle op, val, addr \rangle) = \langle op, val, addr \rangle$$
$$f_{e_8}(\langle op, val, addr \rangle) = f_{e_9}(\langle op, val, addr \rangle) = \begin{cases} *((\text{int}*)addr) = val; & if\ op = WR \\ *((\text{int}*)addr); & if\ op = RD \end{cases}$$

7. MAPPING FEASIBILITY

A mapping of a process network onto a platform is considered infeasible if at least one channel cannot be implemented. In order for a channel to be implemented, using the scheme described in the section 4, it must be possible for the processors to which the processes communicating via the channel are mapped to access the memory where the channel is mapped. In the following, we will use the following definitions:

Definition 1 (Process Network): *A process network is a triple pn = <P,C,D> where P = $\{p_0,\ldots,p_n\}$ is a set of processes, C = $\{c_0,\ldots,c_m\}$ is a set of channels, and D = $\{d_0,\ldots,d_q\}$ is a set of dependencies. A dependency is triple d = <p,c,op> where p ∈ P, c ∈ C and op ∈ {wr, rd}.*

Definition 2 (Mapping): *A mapping of an application onto a platform consists is a pair m = $<m_p,m_c>$ where m_p: P → PE is a map mapping processes to processors and m_c: C → MEM is a map mapping channels to memories.*

Definition 3 (Dependency Satisfiability): *A dependency d = <p,c,op> is said to be satisfied by a mapping m = $<m_p,m_c>$ if the memory $m_c[c]$ is accessible from the processor $m_p[p]$.*

Definition 4 (Mapping Feasibility): *A mapping m = $<m_p,m_c>$ of a process network pn = <P,C,D> onto a platform is feasible if all dependencies d ∈ D are satisfied by m for the platform otherwise the mapping is infeasible.*

Using a SRM description of the platform, it is fairly straightforward to determine if a dependency *d = <p,c,op>* is satisfied or not by testing if the *MemoryService* provided by the memory component $m_c[c]$ is available through the *sw_execution* service of the $m_p[p]$ processor component:

(Dependency satisfiability check) $availableAt(m_p[p].swe_execution) \cap \{m_c[c].memory_service\} = \emptyset$ where, *availableAt: interface ∪ service → 𝒫(service)* is a function returning the set of services available at an interface or service in the SRM model.

Given an infeasible mapping of a process network onto a platform we may attempt to extend the process network with appropriately mapped processes and channels so that otherwise impossible memory accesses can be done indirectly via other processors. Figure 10 shows an example of this.

As long as the added processes only forward data the transformed process network will behave similar to the original because the transformation is equivalent to simply extending the channel capacity and/or introducing delay in the channel.

Figure 10. **a)** *a process network consisting of two processes and a single channel mapped onto the platform depicted in Figure 1. This particular mapping is infeasible because it is not possible for p_0 mapped to pe_1 to access mem_0 where the channel is mapped* **b)** *The process network and mapping of a) extended with a new process p_{new} mapped to pe_0 and channel c_{new} mapped to mem_1 enabling the copying of data from p_0 to c_0 in mem_0.*

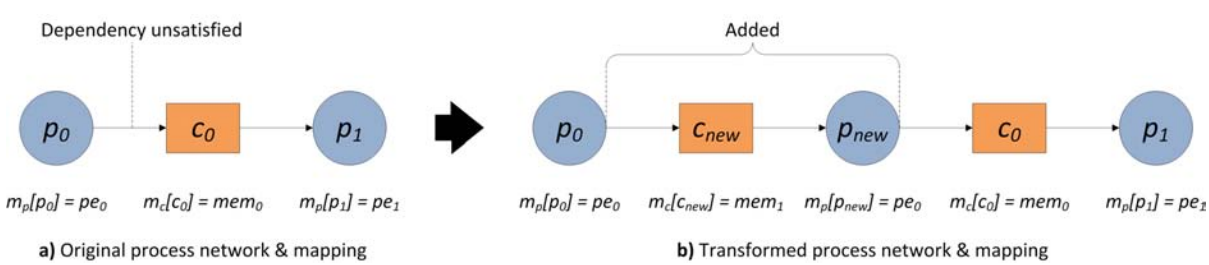


Figure 11 shows an algorithm for transforming a possibly infeasible mapping and process network into a feasible mapping and process network. The algorithm tests the satisfiability of each dependency in the process network. If a dependency $d = <p,c,op>$ is determined to be unsatisfiable, then the algorithm proceeds to check if the access can be done indirectly via another processor. For this to be possible the platform must contain a processor pe_x that can access the memory $m_c[c]$ and a memory mem_x accessible from both $m_p[p]$ and pe_x. In the algorithm, each processor pe_x in the platform is tested individually. First, the set of memories shared with $m_p[p]$ is computed (line 7). Next, the algorithm determines if the memory $m_c[c]$ is accessible from pe_x (line 8). If the two processors have at least one shared memory and if $m_c[c]$ is accessible from pe_x then the algorithm inserts a new process p_2 mapped to pe_x and channel c_2 mapped to one of the shared memories into the process network (line 9-19). If the algorithm fails to locate suitable processor and memory then it terminates with failure.

Notice that the algorithm only attempts to correct unsatisfiable dependencies by implementing indirection via a single processor and memory. A more general solution would consider multiple levels of indirection through multiple processors and memories.

8. CASE STUDY

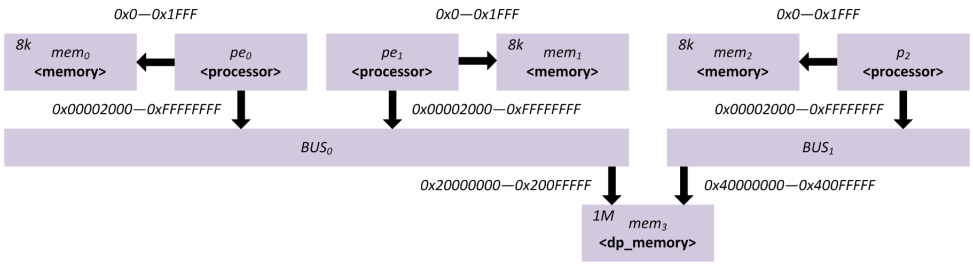
In this section, we present a case study where an application is mapped to a custom platform.

Figure 12 shows a schematic of the example platform. The platform contains three processor instances (pe_0, pe_1, pe_2) each of which is connected to a local memory (mem_0, mem_1, mem_2). The infrastructure of the platform consists of two busses (bus_0, bus_1). The processors pe_0 and pe_1 are connected to bus_0 as master attachments and pe_2 is the only master of bus_1. A dual-port memory

Figure 11. The process network transformation algorithm. The algorithm uses several auxiliary functions: First()returns the first element of a set, GetParentComponent() returns the parent component of a service and instanceOf() returns all instances of a given service in a SRM model.

```
1   procedure transform_pn(pn : PN, m : M, platform : SRM) begin
2       foreach dependency d <p,c,op> in pn.D do
3           if m_c[c].memory_service □ availableAt(m_p[p].swe_execution) then
4               /* dependency d is not satisfied */
5               resolved ← false;
6               foreach processor pe in platform do
7                   S_shared ← availableAt(m_p[p].swe_execution) ∩ availableAt(pe.swe_execution) ∩ instanceOf(MemoryService);
8                   if m_c[c].memory_service □ availableAt(pe.swe_execution) ∧ S_shared ≠ Ø then
9                       /* processor pe can forward */
10                      p_2 ← CreateForwardingProcess(p);
11                      c_2 ← CreateForwardingChannel(c);
12                      pn.P ← pn.P ∪{ p_2 };
13                      pn.C ← pn.C ∪ { c_2 };
14                      m_p[p_2] ← pe;
15                      m_c[c_2] ← GetParentComponent(First(S_shared));
16                      op_other ← nil;
17                      if op= wr then op_other ← rd; else op_other ← wr; end if;
18                      pn.D ← (pn.D \ d) ∪ {<p,c_2,op>,<p_2,c_2,op_other>,<p_2,c,op >};
19                      resolved ← true;
20                      break;
21                  end if;
22              end foreach;
23              if □resolved then
24                  return failure;
25              end if;
26          end if;
27      end foreach;
28      return success;
29  end procedure;
```

Figure 12: Schematic of the example platform. Memory controllers and other peripherals are not shown. The address range occupied by slave components in the address spaces of the masters are shown next to the arrows representing master-to-slave relationships.



(mem_3) is connected to both busses. The figure also shows the range of addresses occupied by slave components in the address spaces of the master components. Note that the dual port memory mem_3 is mapped to different segments in the address spaces of the two busses.

The example platform presented here is a simplified version of a real platform built using the Xilinx Embedded Development Kit for testing purposes. Components representing timers required by the operating systems, debugging utilities and various interface controllers have been left out in order to save space. These components have trivial SRM characterizations and could easily be added. The platform specification used by this tool is sufficiently detailed to allow for automated extraction of an SRM description (including necessary parameter values to allow service invocation generation).

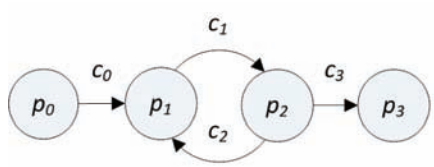
The application consists of four processes (p_0, p_1, p_2, p_3) and four channels (c_0, c_1, c_2, c_3). Figure 13 shows the process network and the mapping onto the platform of Figure 12.

In generating an abstraction layer implementing the communication infrastructure of the application for the example platform there are two keys problems:

a) **Mapping Feasibility.** The mapping is infeasible because process p_2 mapped to processor pe_2 cannot access memory mem_0 where channel c_3 is mapped.
b) **Service Invocation Synthesis**. The dual port memory mem_3 is mapped to different address segments in the address space of the two busses. This means that the effective addresses of data within mem_3 may differ when accessed from different processors.

As a first step in generating the layer, the algorithm of section 7 is used to check the feasibility of process network and mapping for the example

Figure 13. Process network of the application and user-supplied mapping and channel capacity information.

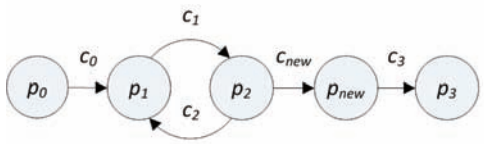


a) Process network

Mapping					
Processes			Channels		
p_0	→	pe_0	c_0	→	mem_3
p_1	→	pe_1	c_1	→	mem_3
p_2	→	pe_2	c_2	→	mem_3
p_3	→	pe_0	c_3	→	mem_0

b) Mapping

Figure 14. Extended process network and mapping.



a) Extended process network

Mapping					
Processes			Channels		
p_0	→	pe_0	c_0	→	mem_3
p_1	→	pe_1	c_1	→	mem_3
p_2	→	pe_2	c_2	→	mem_3
p_3	→	pe_0	c_3	→	mem_0
p_{new}	→	pe_0	c_{new}	→	mem_3

b) Extended mapping

platform. The algorithm is capable of detecting the problem with process p_2 not being able to access memory mem_0 where c_3 is mapped. By analyzing the platform, the algorithm is able to determine that processor p_0 and mem_3 can be used to implement the memory access. A new process P_{new} mapped to pe_0 and channel c_{new} mapped to mem_3 are added. Figure 14 shows the extended process network.

Before generating the code, the size of channels in memory must be computed and their exact locations in the memories determined. The size of a channel in bytes is easily computed as the number of tokens times the size of each token plus eight bytes for the channel administration. Here we assume that the channels can be laid out 0x100 bytes apart in the beginning of the memories.

The final step of the procedure is to generate the code implementing the abstraction layer. For this purpose we construct a SRM description of the complete system. Figure 15 shows such a description for the example platform and application. The actual code is generated by combining the implementation independent templates for the channel access functions (see Figure 5) with C expressions generated using service invocation synthesis as described in section 6 on the basis of the complete SRM description. Figure 16 shows the resulting code for the channel c_2.

This case study demonstrates that our approach is capable of generating an abstraction layer implementing the communication infrastructure of a process network for a custom platform with a non-standard hardware infrastructure. The generated code for accessing a channel is quite efficient and may be compiled into a few dozen instructions for a typical RISC 32-bit processor.

CONCLUSION

In this article, we have presented our approach for generating an abstraction layer implementing the communication infrastructure of an application modeled as a process network. The approach is based on an abstract implementation scheme that imposes very few restrictions on the underlying platform. More specifically, the scheme expects application processes to be mapped to processors and channels to memories and assumes that the memories to which the channels are mapped are somehow accessible from the processors executing the processes accessing the channels. At the heart of our approach is the service relation model and its associated analysis method for analyzing the flow and availability of services in systems composed of components. We have shown how information retrieved from analyzing a platform described using the model can be used for deriving concrete channel implementations based on the abstract implementation scheme. To further expand the design space supported by our approach, we have presented a procedure for implementing process-to-channel accesses that are not directly supported by the platform indirectly by extending the process network with new appropriately mapped processes and channels. This procedure

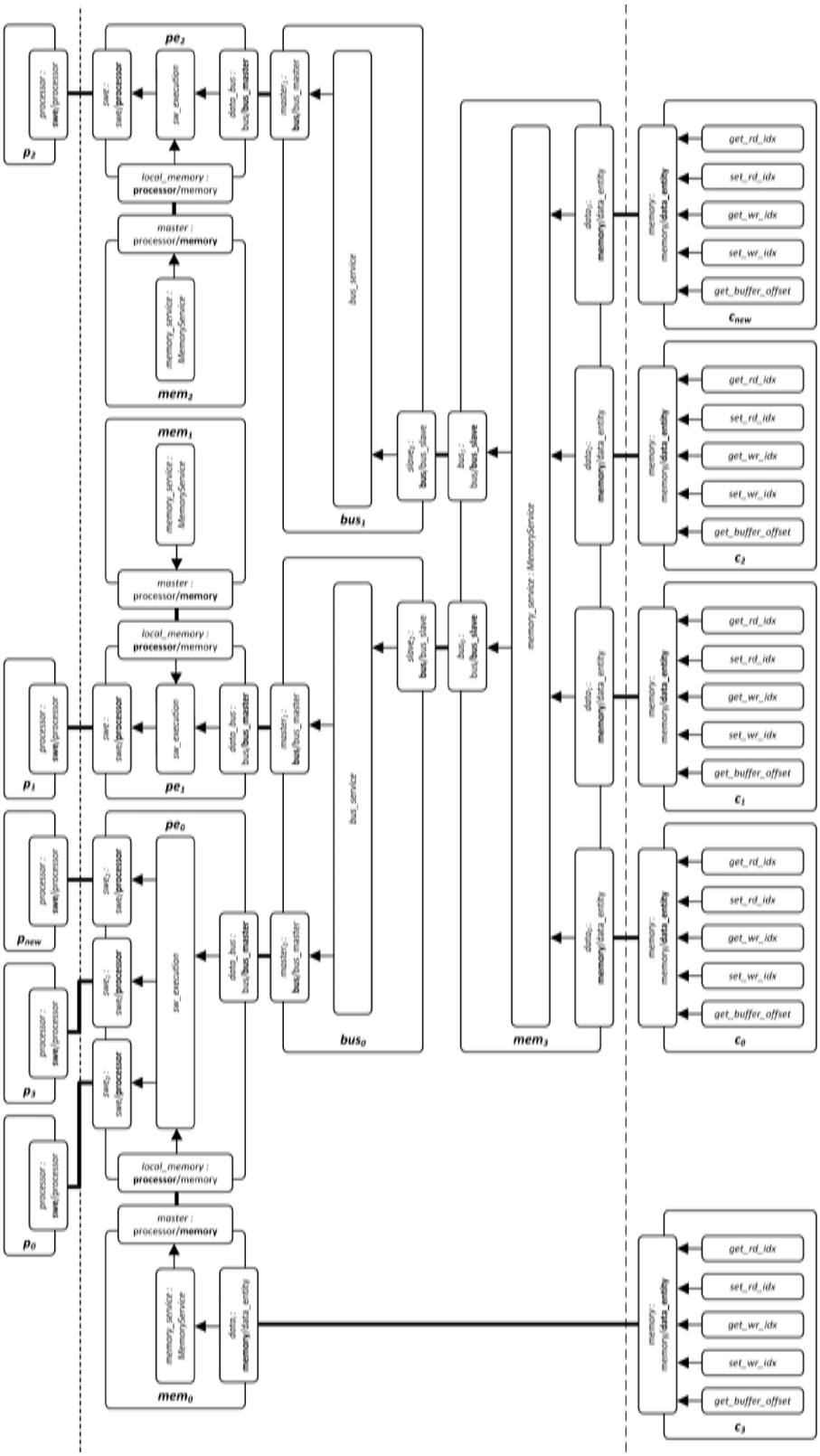


Figure 15. SRM description of the complete system (application, platform and mapping).

Figure 16. Generated read and write functions for the channel c2.

```
 0 #define TOK_CNT  (2 + 1)
 1 #define TOK_SIZE 4
 2
 3 int  get_rd_idx()        { return *((*int)0x20000300); }
 4 int  get_wr_idx()        { return *((*int)0x20000304); }
 5 void set_wr_idx(int v)   { *((*int)0x20000304) = v; }
 6 int  get_buffer_offset() { return *((*int)0x20000308); }
 7
 8 void write_c2(void* data) {
 9   /* wait for store to be not full */
10   while(((get_wr_idx() + 1) & TOK_CNT) == get_rd_idx());
11
12   copy_to_token(
13     get_buffer_offset(), get_wr_idx(), data, TOK_SIZE
14   );
15
16   /* update write index */
17   set_wr_idx((get_wr_idx() + 1) % TOK_CNT);
18 }
```

a) Write access function

```
 0 #define TOK_CNT  (2 + 1)
 1 #define TOK_SIZE 4
 2
 3 int  get_rd_idx()        { return *((*int)0x40000300); }
 4 void set_rd_idx(int v)   { *((*int)0x40000300) = v; }
 5 int  get_wr_idx()        { return *((*int)0x40000304); }
 6 int  get_buffer_offset() { return *((*int)0x40000308); }
 7
 8 void read_c2(void* data) {
 9   /* wait for store to be not empty */
10   while(get_wr_idx() == get_rd_idx());
11
12   copy_from_token(
13     get_buffer_offset(), get_rd_idx(), data, TOK_SIZE
14   );
15
16   /* update read index */
17   set_rd_idx((get_rd_idx() + 1) % TOK_CNT);
18 }
```

b) Read access function

also uses information retrieved from analyzing a service relation model description of the platform.

In this article, we have exclusively used the service relation model for reasoning about memory mapped services. The model and the associated analysis and invocation synthesis methods are general enough to also be used with other kinds of infrastructure such as interrupt sub-systems, peer-to-peer connections and software frameworks. The problem with these is that the services they provide may only be available in limited quantity which means that a given service may only be part of a limited number of service invocations. For example, an interrupt signal can only be used to propagate a single event. Deciding on how to use such services involves formulating and solving a satisfiability problem. We are currently looking into extending our approach to also support interrupt sub-systems and peer-to-peer connections for synchronization and data transfer.

ACKNOWLEDGMENT

This work was in part supported by Grant No. 003-2006-2 "DaNES", under the Danish National Advanced Technology Foundation, and "ArtistDesign", the European FP7 Network-of-Excellence on Embedded Systems Design.

REFERENCES

Abdi, S., Shin, D., & Gajski, D. (2003). Automatic communication refinement for system level design. In *DAC '03: Proceedings of the 40th Conference on Design Automation,* Anaheim, CA (pp. 300-305). New York: ACM Publishing.

Altera. (2009). *Quartus® II Development Software handbook: SOPC Builder*. Retrieved May 2009, from http://www.altera.com/literature/lit-sop.jsp

Bergamaschi, R., Lee, W. R., Richardson, D., Bhattacharya, S., Muhlada, M., Wagner, R., et al. (2000). Coral - automating the design of systems-on-chip using cores. In *Proceedings of the IEEE 2000 Custom Integrated Circuits Conference,* Orlando, FL (pp. 109-112). Washington, DC: IEEE Computer Society.

Cesario, W. O., Lyonnard, D., Nicolescu, G., Paviot, Y., Sungjoo, Y., Jerraya, A. A., et al. (2002). Multiprocessor SoC platforms: A component-based design approach. *Design & Test of Computers,* 52-63.

de Kock, E. A., Smits, W. J., van der Wolf, P., Brunel, J.-Y., Kruijtzer, W. M., Lieverse, P., et al. (2000). YAPI: Application modeling for signal processing systems. In *DAC '00: Proceedings of the 37th Conference on Design Automation,* Los Angeles (pp. 402-405). New York: ACM Publishing.

Garlan, D., Monroe, R. T., & Wile, D. (2000). Acme: Architectural description of component-based systems. In *Foundations of component-based systems* (pp. 47-67). New York: Cambridge University Press.

Jerraya, A. A., Bouchhima, A., & Frédéric, P. (2006). Programming models and HW-SW interfaces abstraction for multi-processor SoC. In *DAC '06: Proceedings of the 43rd Annual Conference on Design Automation,* San Francisco (pp. 280-285). New York: ACM Publishing.

Kahn, G. (1974). The semantics of a simple language for parallel programming. In *IFIP Congress 74.* Amsterdmam, The Netherlands: North-Holland Publishing.

Kriaa, L., Bouchhima, A., Youssef, W., Petrot, F., Fouillart, A.-M., & Jerraya, A. (2006). Service based component design approach for flexible hardware/software interface modeling. In *Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping, 2006,* Chania, Crete (pp. 156-162). Washington, DC: IEEE Computer Society.

Martin, G. (2006). Overview of the MPSoC design challenge. In *Proceedings of the 43rd Annual Conference on Design Automation,* San Francisco (pp. 274-279). New York: ACM Publishing.

Nielson, F., Nielson, R. H., & Hankin, C. (2005). Principles of program analysis. In F. Nielson, R. H. Nielson, & C. Hankin (Eds.), *Principles of program analysis* (pp. 365-392). Berlin, Germany: Springer Verlag.

Nieuwland, A., Kang, J., Gangwal, O. P., Sethuraman, R., Busá, N., & Goossens, K. (2004). C-HEAP: A heterogeneous multi-processor architecture template and scalable and flexible protocol for the design of embedded signal processing systems. *Design Automation for Embedded Systems*, 233–270.

Nikolov, H., Stefanov, T., & Deprettere, E. (2008). Systematic and automated multiprocessor system design, programming and implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 542–555. doi:10.1109/TCAD.2007.911337

Paul, J. M. (2003). Programmers' views of SoCs. In *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis,* Newport Beach, CA (pp. 156-181). New York: ACM Publishing.

Paulin, P. G., Pilkington, C., Langevin, M., Bensoudane, E., & Nicolescu, G. (2004). Parallel programming models for a multi-processor SoC platform applied to high-speed traffic management. In *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis,* Stockholm, Sweden (pp. 48-53). New York: ACM Publishing.

Piementel, A. D., Hertzberger, L. O., Lieverse, P., van der Wolf, P., & Deprettere, F. E. (2001). Exploring embedded-systems architectures with Artemis. *IEEE Computer,* 57-63.

Sarmento, A., Kiraa, L., Arnaud, G., Youssef, M.-W., Bouchhima, A., Rousseau, F., et al. (2005). Service dependency graph, an efficient model for hardware/software interfaces modeling and generation for SoC design. In *Proceedings of the Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '05,* Jersey City, NJ (pp. 261-266). Washington, DC: IEEE Computer Society.

Todor, S., Zissulescu, C., Turjan, A., Kienhuis, B., & Deprettere, E. (2004). System design using Khan process networks: The Compaan/Laura approach. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2004* (pp. 340-345).

van der Wolf, P., de Kock, E., Henrikson, T., Kruijtzer, W., & Essink, G. (2004). Design and programming of embedded multiprocessors: An interface-centric approach. In *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis,* Stockholm, Sweden (pp. 206-217). New York: ACM Publishing.

van Ommering, R., van der Linden, F., Kramer, J., & Magee, J. (2000). The Koala component model for consumer electronics software. *IEEE Computer*, *33*, 78–85.

Xilinx. (2009). *Xilinx Platform Studio and Embedded Development kit*. Retrieved May 2009, from http://www.xilinx.com/tools/embedded.htm

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 1, edited by Seppo Virtanen, pp. 37-63, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Section 4
Formalisms and Methodologies for Embedded Communication Systems

Chapter 14
Joint Validation of Application Models and Multi-Abstraction Network-on-Chip Platforms

Sanna Määttä
Tampere University of Technology, Finland

Luciano Ost
Catholic University of Rio Grande do Sul, Brazil

Leandro Möller
Technische Universität Darmstadt, Germany

Manfred Glesner
Technische Universität Darmstadt, Germany

Leandro Soares Indrusiak
University of York, UK

Jari Nurmi
Tampere University of Technology, Finland

Fernando Moraes
Catholic University of Rio Grande do Sul, Brazil

ABSTRACT

Application models are often disregarded during the design of multiprocessor Systems-on-Chip (MPSoC). This is due to the difficulties of capturing the application constraints and applying them to the design space exploration of the platform. In this article we propose an application modelling formalism that supports joint validation of application and platform models. To support designers on the trade-off analysis between accuracy, observability, and validation speed, we show that this approach can handle the successive refinement of platform models at multiple abstraction levels. A case study of the joint validation of a single application successively mapped onto three different platform models demonstrates the applicability of the presented approach.

INTRODUCTION

The design space of multiprocessor systems based on Network-on-Chip (NoC) interconnects is very large. Considering the interconnect structure alone, designers must choose or parameterise a large number of components according to application-specific constraints, such as buffering and flow control mechanism, network topology, word width, packet structure, or routing algorithm.

In this article, we propose a model-based methodology to evaluate the performance of a particular setup of a NoC interconnect under the constraints of a particular application. Our goal

DOI: 10.4018/978-1-4666-0912-9.ch014

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

is to support the creation of a unified application model, which we can jointly validate with different platform models, each representing an alternative configuration of the NoC interconnect.

The validation of application models and platform models for NoC-based systems can be performed in several ways, ranging from formal and semi-formal approaches based on graph representations of both application and platform (Hu & Marculescu, 2005) to ad-hoc approaches that actually emulate the execution of the application on a prototype of the NoC using multiple FPGA boards (Saint-Jean et al., 2005). The next section presents further details of such approaches. Our approach combines semi-formal techniques with executable and simulatable models for joint validation of the application and platform. Both models are based on the principles of actor-orientation (Eker et al., 2003), which the Application Modelling Section covers. Actor-oriented models include the explicit description of the concurrent behaviour of the model's components. To improve the expressiveness of actor-oriented models, we extend them by using Unified Modelling Language (UML) (Object Management Group, 2005) sequence diagrams for explicit ordering of inter-component communication. Applying such technique to application modelling is also addressed in the Application Modelling Section, whose particular emphasis is on the employment of hierarchical composition of systems with heterogeneous models of time and concurrency (one of the major benefits of actor-orientation). The Platform Modelling Section covers the modelling of hardware platforms based on NoCs. Within the scope of this article, a hardware platform includes the interconnect structures, an abstract representation of processing and storage elements, and a number of observability features (referred as scopes). We present a few strategies for the modelling of such platforms, each of them having different characteristics regarding simulation speed, accuracy, observability, and modelling effort. The Application Mapping onto Successively Refined Platform Models Section presents a methodology for the exploration of those different platforms in order to meet the application requirements. Furthermore, we present a case study of the validation of the proposed methodology. Then we analyse qualitatively each modelling strategy and comparatively the performance results obtained from different configurations of the NoC interconnect. Finally, we conclude the article and present some future work.

RELATED WORK

This article is based on the premise that it is necessary to consider the impact of the application on its underlying platform early at the design process in order to meet all performance, area, power consumption, and time-to-market constraints. Many research initiatives are also built on that premise, especially in multiprocessor System-on-Chip (MPSoC) design. Some of them are detailed below.

Kempf et al. (2006) present a framework targeted to MPSoC software development, verification, and evaluation. Their framework does not require the platform model to be complete before the software development can start. Software can be developed in four different levels of abstraction that vary in accuracy and simulation speed. The framework uses SystemC for simulation and XML to describe task mappings and timings. Furthermore, the framework provides an efficient design space exploration environment for instance by providing designers with various communication architectures. Ristau et al. (2008) discuss design space exploration early at the design process as well as the exploration of different mapping strategies. However, their application and platform models are simplified, disregarding the inter-process communication costs. Lei and Kumar (2003) describe the application as parameterisable task graphs, which are mapped onto NoC architecture. Their work aims at supporting mapping based on genetic algorithms

and minimising overall execution time of a task graph. Furthermore, they have also implemented a tool that uses a two-step genetic algorithm, achieving optimised solutions for regular NoC topologies in affordable time. In contrast to the work mentioned above, Saint-Jean et al. (2007) have an ad-hoc approach that emulates the execution of the application on a prototype of the NoC using multiple FPGA boards.

UML supports object-oriented system design, which, among other things, makes it popular among software designers. In fact, its popularity is increasing also among hardware and embedded system communities (Martin, 2002). Riccobene et al. (2005) present a System-on-Chip (SoC) design methodology using code generation from UML diagrams to an executable SystemC model. Furthermore, Arpinen et al. (2006) use UML state chart diagrams to support automatic code generation in order to map UML applications onto the platform. In this article, we rely on a similar UML-based modelling approach, but we do not require code generation in order to validate the application. Instead, we provide the model itself with execution semantics so that we can validate the model directly.

APPLICATION MODELLING

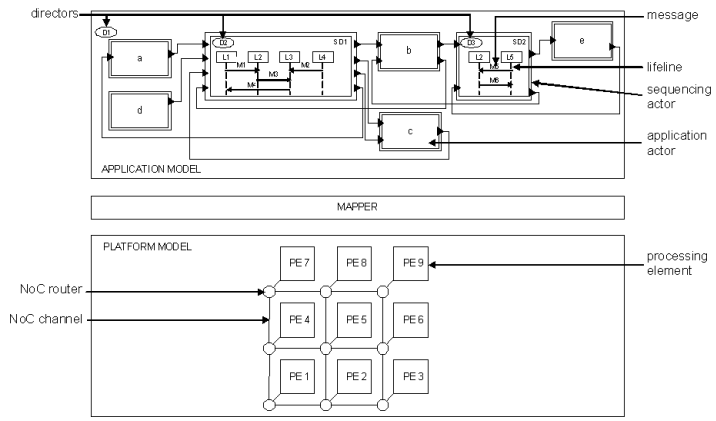
Due to the complexity and heterogeneity of state-of-the art embedded systems, we need to use various levels of abstractions and multiple Models of Computation (MoC) (Eker et al., 2003; Martin, 2002). Heterogeneous modelling using SystemC (Grötker et al., 2002) would require extensions to the standard SystemC simulation kernel, so that models containing Synchronous Data Flow (SDF), Communicating Sequential Processes (CSP), or Finite State Machine (FSM) MoCs can be simulated with the alternative kernel and Discrete Event (DE) models with the standard kernel (Patel & Shukla, 2004). In order to avoid ad-hoc solutions for truly heterogeneous system modelling and simulation, we can find a suitable alternative on the Ptolemy II framework (Brooks et al., 2007), which natively supports the composition of multiple MoCs on a single system model.

Actor-orientation, which is the theoretical foundation behind the Ptolemy II framework, was initially proposed as a mathematical model of concurrent computation (Hewitt et al., 1973) and later as a formal model of concurrency (Agha, 1986). The main components of actor-orientation are actors, which are concurrent elements and use channels and message passing for communication. The Ptolemy II framework offers visual modelling of actor-oriented designs and a simulation kernel, which supports the simulation of hierarchical designs using different MoCs for different subsystems. Furthermore, Ptolemy II framework allows the implementation of new MoCs. Using the ideas of actor-orientation and the flexibility of the Ptolemy II framework, we can create an executable application model using UML sequence diagrams for describing the communication patterns of the application and encapsulating the diagrams inside an actor (we call it a sequencing actor) (Indrusiak et al., 2007; Indrusiak & Glesner, 2007). The upper part of Figure 1 shows an application model, particularly illustrating the encapsulation of UML sequence diagrams inside sequencing actors.

Taking into account the foundations mentioned above, we can formally state that an application model is a directed clustered graph G = G(A, C), where A is a set of actors and C is a set of communication links. $A_i \subset A$ is an actor. A_i contains input ports $p_{i1}, p_{i2,\ldots,} p_{in} \in P_{i,} P_i \subseteq P$ and output ports $p_{o1}, p_{o2,\ldots,} p_{on} \in P_{o,} P_o \subseteq P$. Actors A_i and A_j exchange data tokens over the communication link $C_{i,j} \in C$, which connects one of the output ports of A_i with one of the input ports of A_j.

Actors can be either atomic (a simple capsule to a sequential computation a $\in A_i$) or composite. Composite actors $W_i \subset W, W \subset A$, which are responsible for the hierarchical composition in G, are composed of other actors so that $A_{j,} A_{k,\ldots,} A_z \subset W_i$.

Figure 1. Joint validation of application and platform



Opaque composite actors $K_i \subset W$ are composite actors that are able to control their own execution semantics (and that of their non-opaque components). The execution semantics of an opaque composite actor is explicitly defined by a special component called a director d ∈ D, so that ∀ $K_i \subseteq W$, $K_i \supset$ d. A sequencing actor $Q_i \subset Q$ is a special kind of opaque composite actor, and its sole functionality is to guarantee that the incoming data tokens are written to its output ports according to a predefined sequence. This sequence is defined by a UML sequence diagram, which is denoted as a tuple SD = ⟨ L, E, m, n, < ⟩. L is a finite set of lifelines and E a finite set of events. Events can either be send or receive events, s ∈ S, r ∈ R, $S \subset E$, $R \subset E$ respectively. A bijection m: $S \rightarrow R$ matches each send event s to its corresponding receive event r. A surjection n: $E \rightarrow L$ assigns each event e to one and only one lifeline $\ell \in L$, though a lifeline ℓ can (and it is likely to) be assigned more events. Finally, $< \subseteq E \times E$ is an acyclic relation between events consisting of a total order of all e whenever n(e) = ℓ ∈ L, and s < r whenever m(s)=r.

A sequence diagram SD and its respective sequencing actor Q_i are related as follows. A bijection f: $P_i \rightarrow S$ maps each input port p_i of Q_i onto a send event s of SD. Likewise, another bijection b: $P_o \rightarrow R$ maps each output port p_o of Q_i onto a receive event r of SD. The arrival of a data token at an input port p_i can cause the triggering of the send event s = f(p_i) *iff* all preceding events of that lifeline had been triggered previously: ∀e, n(e) = n(s) = ℓ, e < s. Finally, the triggering of the receive event r = b(p_o) can cause the release of the data token through the output port p_o.

Constraint 1: If an actor A_i is connected to a sequencing actor Q_i, all its output ports $p_{o1}, p_{o2,\ldots,}$ $p_{on} \in A_i$ must be connected to input ports $p_{i1}, p_{i2,\ldots,}$ $p_{in} \in Q_i$ so that n(f(p_{i1})) = n(f(p_{i2})) = … = n(f(p_{in})), that is, their respective send events must be assigned to the same lifeline. Likewise, all its input ports $p_{i1}, p_{i2,\ldots,} p_{in} \in A_i$ must be connected to output ports $p_{o1}, p_{o2,\ldots,} p_{on} \in Q_i$ so that n(b(p_{o1})) = n(b(p_{o2})) = … = n(b(p_{on})). Furthermore, n(b(p_{o1})) = n(f(p_{i1})), so that one actor A_i connected to a sequencing actor Q_i can be related to a single lifeline of the sequence diagram governing the behaviour of Q_i. Nothing prevents, however, that a lifeline could be related to multiple actors A_i ... A_n.

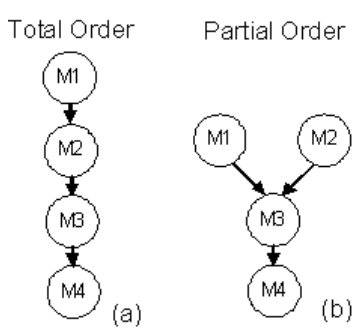
Figure 1 depicts that each lifeline (L1-L5) of a sequence diagram represents one application actor (denoted by letters a-e) connected to an input or an output port of the sequencing actor. Each message (M1-M6) between lifelines represents communication between two actors of the application model, and the precedence relation (denoted by the vertical dimension on the diagram) defines

the order in which messages should be delivered between the application actors. In fact, the director inside the sequencing actor enforces this order to be maintained.

Indrusiak and Glesner (2007) present two different directors for controlling the execution of sequence diagrams, total order and partial order, based on SDTODirector and SDPODirector respectively. The total order director maintains the order of the messages of a sequence diagram whereas the partial order director maintains the order separately on each lifeline. Both of the directors create a precedence graph of all messages of the sequence diagram. Figure 2 depicts the difference between total and partial order precedence graphs. Total order sequence diagram can be illustrated using a directed graph, where the precedence of messages M1-M4 (see the leftmost sequence diagram in Figure 1) is the same as their order in the sequence diagram. Partial order is also a directed graph, but now it is possible to see that the order of messages M1 and M2 does not matter, because the order of messages is enforced separately on each lifeline and M1 and M2 do not have events on the same lifeline.

The directors SDTODirector and SDPODirector enable the execution of application models within the Ptolemy II framework. In the Application Mapping onto Successively Refined Platform Models Section, we describe the necessary extensions to Ptolemy II so that application models can be mapped onto platform models, and each message of the application is back-annotated with the communication latency of the platform.

Figure 2. The difference between total order and partial order precedence graphs



PLATFORM MODELLING

From a formal point of view, a platform can be seen as a directed graph H = G(T,V), where T is a set of processors and V are the communication channels through which the processors exchange data packets. However, such models must also contemplate functional details such as internal latencies, congestion effects, and routing and arbitration delays, so that designers can properly explore the NoC design space and optimise the interconnect architecture to a particular traffic scenario (Pande et al., 2005). The use of abstract platform models is a trade-off between the *level of detail* and *model confidence*. The *level of detail* refers to the structural and behavioural abstraction of the NoC's components. The structural abstraction means the granularity of a data storage (that is, a storage for a flit or a packet) and the number of included components and their interconnects. The behavioural abstraction includes how and when the components update their internal state and concurrently interact with other components. The *model confidence* means how useful the model is for a particular purpose for instance in terms of accuracy compared to a reference model.

In this article, the reference model is the HERMES infrastructure (Moraes et al., 2004), which is briefly described in the next subsection. Moreover, we present a high-level model based on UML interactions, which derive from the RTL model of HERMES infrastructure called RENATO. Our next model, JOSELITO, is a simplified NoC model that uses the Payload Abstraction Technique (PAT) and allows performance evaluation by combining simulation and analytical methods. Furthermore, BOÇA is another simplified model abstracting all the internal state of a router except its buffers. Finally, we give an overview of the actors developed for observing

and debugging the execution of an application running on top of the NoC models, called scopes.

HERMES Reference Model

HERMES is a Register Transfer (RT) level infrastructure, which implements a low area overhead packet switching NoC. HERMES supports both 2D mesh and torus topologies and allows designers to select routing algorithm, control flow mechanism, flit size, and buffer depth. Its routers have centralised switching control logic and five bi-directional ports. One port is used to establish the communication between a router and its local processing element, whereas the others are connected to the neighbour routers. HERMES uses round-robin arbitration for granting access to incoming packets, which are stored in a FIFO buffer. The priority of an input port is a function of the last input port having a routing request granted. If the incoming packet request is granted by the arbiter, the XY routing algorithm is executed to connect the input port to the correct output port. If the algorithm returns a busy output port, the header flit and all subsequent flits of this packet are blocked. After all flits in a packet are transmitted, the port is released (Moraes et al., 2004).

RENATO

RENATO is an actor-oriented model based on two UML sequence diagrams, which describe the interactions of the HERMES router components. These UML sequence diagrams, shown in Figure 3, are the arbitration request by a particular input buffer (interaction on the left side of the figure) and the transmission of a flit from one input buffer to a neighbour router through an output channel (interaction on the right side of the figure) (Indrusiak et al., 2008).

The interaction on the left side of Figure 3 illustrates the establishment of a connection between an input buffer to an output port. Initially, the input buffer sends the packet header to the routing controller. The routing controller asks the arbiter to choose one of the possible incoming requests that can arrive from any of the five input buffers. After getting a positive response, the arbiter sends the header flit to the router and the router executes a particular algorithm (for instance XY algorithm) to determine which output port the packet should be sent to. Once the routing is done, the controller verifies if the chosen output port is free. If the output port is free, the input buffer establishes the connection to the output port, otherwise the connection is refused and the input buffer must start the whole input-output connection requesting process again.

The interaction on the right side of Figure 3 models the transmission of a flit between two neighbouring routers. The controller receives the flit from the local input buffer and checks which output port is allocated to it. After sending the flit, the controller waits for an acknowledgement. A positive acknowledgement removes the successfully sent flit from the source input buffer, a negative acknowledgement causes a re-sending of the flit.

Neither of the sequence diagrams in Figure 3 considers timing among the interactions. In order to improve the accuracy of RENATO, timing information extracted from the simulation of the HERMES VHDL model was added to the respective transaction model. When validating the model, performance results of RENATO were compared to the results of HERMES (cycle-accurate simulation) using different traffic scenarios and NoC configurations (Indrusiak et al., 2008). For long-lasting traffic, the error is only about 10%, which is a very good figure considering that the actor-oriented model is based on the interactions only and works without a synchronising clock signal.

JOSELITO

JOSELITO is a more abstract platform model than RENATO. However, it uses the same UML interactions defined in Figure 3. The main difference

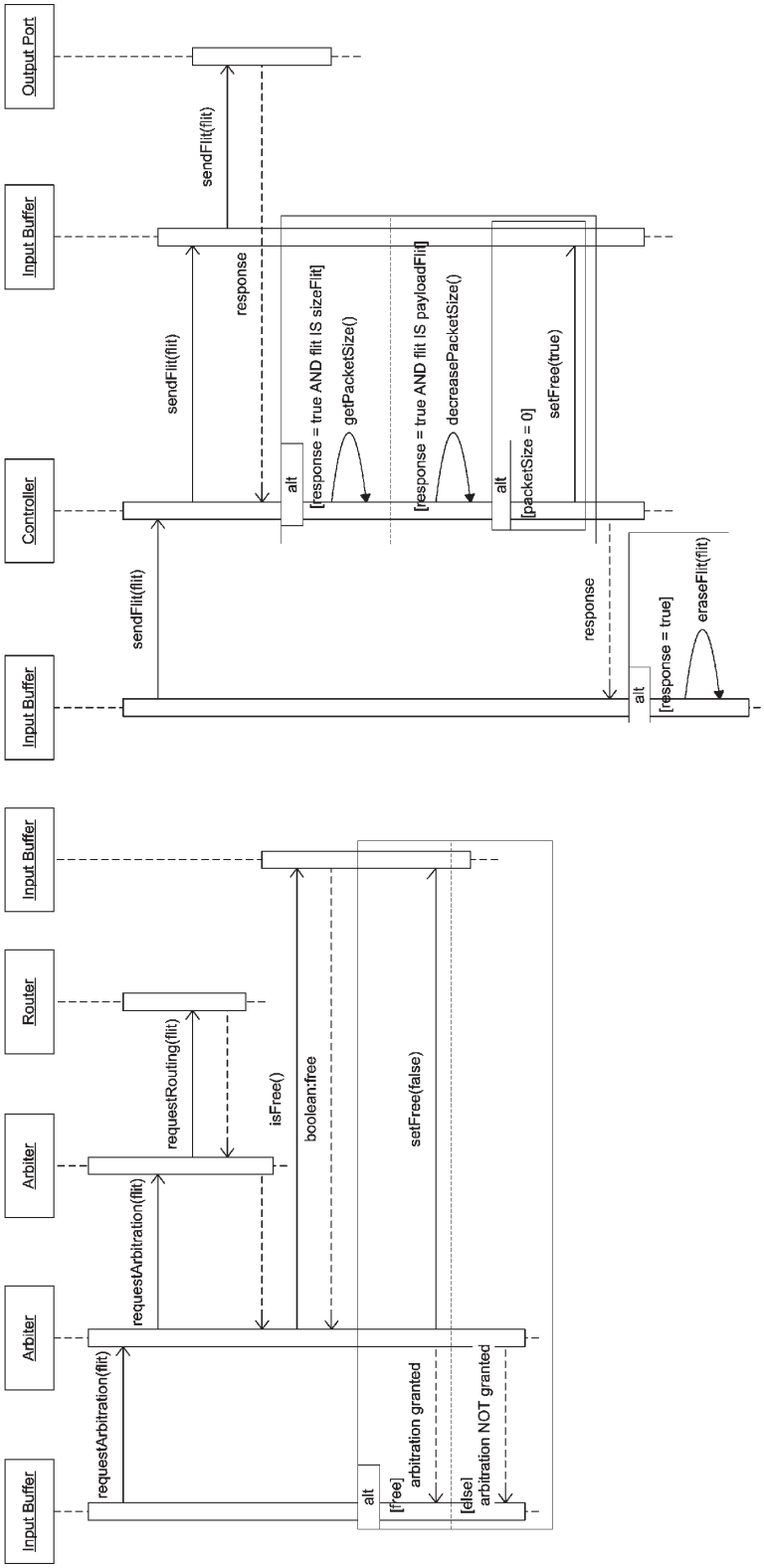


Figure 3. UML sequence diagrams depicting interactions between components of a HERMES NoC

between JOSELITO and RENATO is the decrease in JOSELITO's simulation time, caused by the reduction of the number of communication events due to the flit by flit packet forwarding. JOSELITO uses the Payload Abstraction Technique (PAT), which comprises that: *(i)* the packet is defined as a *header* and a *trailer, (ii)* the buffer is a FIFO structure modelled as a finite state machine, *(iii)* packet headers are released from a given router once there is available buffer space at next hop on its route, and *(iv)* a simple analytical method is used to calculate the packet trailer release time (*ptrt*) (Ost et al., 2008).

Even if the transmission of the packet payload is abstracted away, the simulation model can still represent both unblocked and blocked packet transmission scenarios. If no resource conflicts occur (unblocked scenario), the latency and throughput of the NoC can be measured with no loss of accuracy. In a blocked scenario, when a header packet arrives in an input buffer, two blocking situations can occur: either the desired output port is reserved to another input port or the target neighbour input buffer is not able to receive a header or a trailer of the packet. In such cases, a connection of an intermediate router (for instance, a router between the source and the target processing elements) can be closed without considering the impact of the blockage to other packets.

In this context, the *ptrt* is used to ensure a correct functionality during packet transfers, allowing the designer to obtain high accuracy latency and throughput results with shorter simulation time in comparison with RENATO. The accuracy of those results also depends on the parameters obtained from the RTL simulation. One possible alternative for the analytical calculation of the *ptrt*, presented by Ost et al. (2008) and adopted within the current work, is shown in Equation (1).

$$ptrt = hft + pcksize * ctf \quad (1)$$

where

ptrt is the packet trailer release time,
hft is the header forwarding time,
pcksize is the packet size (number of flits),
and c*tf* is the number of clock cycles to transmit one flit

The results presented by Ost et al. (2008) show that the worst case latency of JOSELITO is 5.26% lower than the latency of HERMES. Furthermore, JOSELITO was in average 2.3 times faster than RENATO in 88% of the executed cases. It is important to mention that the analytical method is not restricted to the three parameters presented in Equation (1). Therefore, different parameters (such as buffer capacity) can be included in order to improve the *ptrt* estimation.

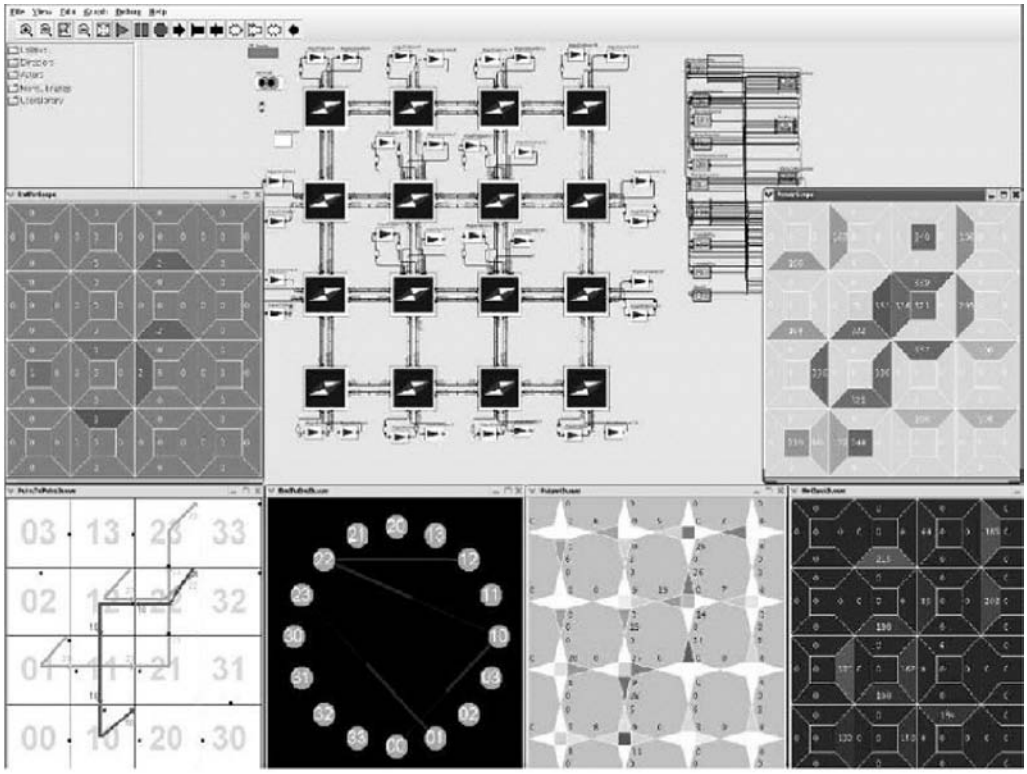
BOÇA

The third platform model considered in this article is the fastest and most abstract one, but it obviously pays for that by having the lowest accuracy. BOÇA considers the multi-hop nature of NoC communications and the buffering of flits at the input ports of each router, but it abstracts completely the arbitration logic and internal state of the routers. Therefore, BOÇA's way to handle the interference among traffic flows does not reflect a real implementation.

Debugging Support

In order to increase the observability and debugging capabilities of the platform models, we have created actors to monitor traffic, collect data from the network, and display it graphically. All those extensions are combined to a small framework called *NoCScope*. The *NoCScope* is a composition of seven scopes*: BufferScope*, *InputScope*, *OutputScope*, *PowerScope*, *HotspotScope*, *EndToEndScope*, and *PointToPointScope.* Figure 4 presents a screen capture of the model running with six of the scopes (*PointToPointScope*, *EndToEndScope*, *OutputScope*, and *HotSpotScope* in the bottom

Figure 4. Application model, platform model, and the scopes



of the picture and *Bufferscope* and *Powerscope* above them). The *Bufferscope* is currently used to analyse buffer occupation, from which we can determine the router memory requirements. The *InputScope* and the *OutputScope* capture the activity of each input and output channel of the routers. With the *PowerScope* we obtain the power consumption of the routers by analysing the transition activity on the channel, as Palma et al. (2006) have done. The *HotspotScope* emphasizes and quantifies the output ports of the NoC routers that are trying to send packets to the next router, but are being blocked due to congestion. The *EndToEndScope* gives us an overview of which modules are communicating with each other and the *PointToPointScope* presents the complete path that the packets use inside the NoC. In all cases, the displays are updated as the simulation runs. Therefore, we can have full observability and controllability of the dynamic behaviour of the NoC model.

APPLICATION MAPPING ONTO SUCCESSIVELY REFINED PLATFORM MODELS

The major goal of the work this article presents is to guide system designers to choose and parameterise a platform model that can cope with the requirements imposed by a particular application. In previous sections, we have described the proposed approaches for modelling the application and platform. In this section, we present a methodology that allows the joint validation of such models, refining the idea that Määttä et al. (2008) present, but considering multiple platform models that range from more abstract and fast ones to more

accurate ones. We propose a methodology based on successive refinement of platform models, so that designers can explore the trade-off between the platform model accuracy, observability, and execution speed. Concretely this means that a designer should start by validating the application on top of a fast but not very accurate model of the platform, and refine such model step-wise so that the accuracy is increased at the expense of the model's execution speed. This enables speeding up the design space exploration.

Conceptually, the link between the application and platform models is established so that a particular application task should be executed by a particular processor of the platform model. This process is known as application-platform mapping and has received substantial attention by the research community within the past few years (Hu & Marculescu, 2005; Marcon et al., 2005; Murali et al., 2005). Thus, it is necessary to define the mapping in order to validate the performance of an application executed on a particular platform.

Following the formal definitions the Application Modelling and Platform Modelling Sections have presented, we define mapping as a function $u: L \rightarrow T$, which defines that a lifeline $\ell \in L$ is assigned to a processor $t \in T$. By assigning a lifeline ℓ to a processor t, the mapping process denotes that all the sequential computations $a_1, a_2, \ldots, a_n$ that are encapsulated by the actors related to the lifeline ℓ are executed by the processor t. Again, we mean here that an actor is related to a lifeline *iff* its ports are connected to the ports that are mapped to the events contained by that lifeline, as described in Constraint 1.

In order to have an implementation of such a mapping function, we introduce an additional element to our approach, the mapper, which has access to information from both the application and platform models, and reacts to events that the directors of sequencing actors and the abstract processing elements generate. The mapper assigns each of the lifelines (lifelines L1 to L5 in Figure 1) to one abstract processing element at the platform model (denoted as PE in Figure 1).

For each token a sequencing actor receives, a pre-defined message within its sequence diagram will be triggered (for example, the message M1 sent by lifeline L1 to lifeline L2). When this happens, the corresponding director D2 interrupts the delivery and notifies the mapper about the message. Since the mapper is responsible of assigning each lifeline to an abstract processing element, it knows that for instance lifeline L1 is mapped to PE 2, whereas L2 is mapped to PE 7. Once the mapper receives the information about the triggered message, it will command the processing element associated to the sender of the message (PE 2) to generate the corresponding traffic into the interconnect structure (in the case of a NoC platform, it must create a packet with destination, size, and payload, and write this packet on the local input port of the corresponding router). Then the mapper waits until the processing element associated to the receiver of the message (PE 7) notifies the complete reception of the packet. Upon notification, the mapper calls back the director D2 (which has notified the triggering of the message) and informs it that the message can now be delivered. After that, the director can forward the message to the output port of the sequencing actor, and the message reaches its destination with the exact latency that it would take if the application is executed on top of the implementation platform.

As the mapper implementation has a critical influence on the communication costs, our approach considers the choice of the mapper as a design decision. Therefore, our implemented approach is extensible, so that we can create a library of different mapping algorithms and heuristics. However, this is not within the scope of this article. The current mapper implementation supports only static mapping and has no constraints (each application actor can be mapped to any processing element). These are not restrictions of the modelling approach, and future work will explore the possibility of different static and dynamic mapping

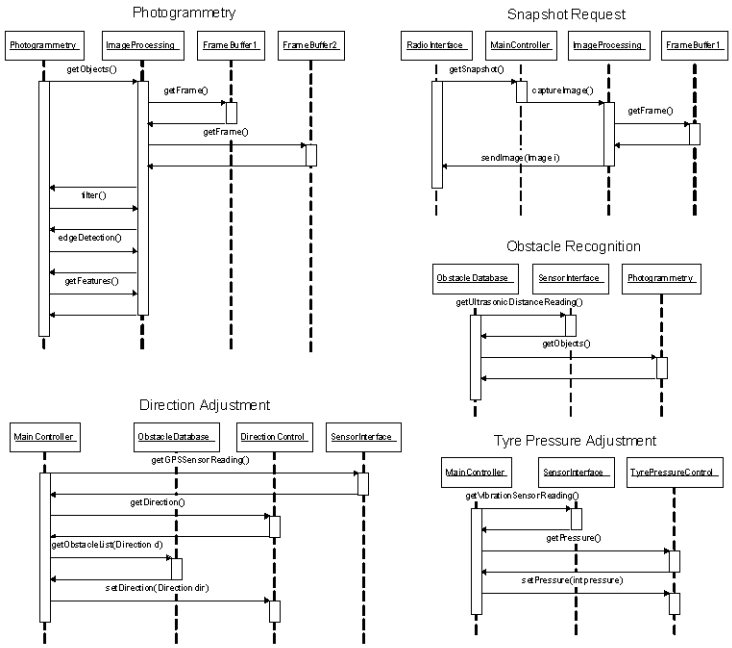
heuristics, as well as the sharing of processing elements by multiple application actors.

CASE STUDY

Figure 5 illustrates the sequence diagrams of the application model we have used in this case study. The application is an autonomous vehicle that is modelled in five different sequence diagrams: photogrammetry, obstacle recognition, direction adjustment, tyre pressure adjustment, and snapshot request. The vehicle has two cameras, which capture images from the direction the vehicle is moving. The photogrammetry logic of the system pre-processes the images and extracts three-dimensional features by exploring the stereoscopy in both images. The obstacle recognition extracts the coordinates of the possible obstacles that might force the vehicle to adjust its direction. The obstacle coordinates are then fed to the obstacle database, which is also adjusted with the information from the ultrasonic sensor (which can measure more precisely the distance of obstacles that are closer). The direction adjustment logic can determine the vehicle's current position using GPS and then adjust the direction according to the information of the obstacle database in order to avoid collisions with obstacles. The vehicle contains also sensors able to measure vibration. If the vehicle vibrates too much, it affects the quality of the camera images. Therefore, the tyre pressure adjustment is able to adjust the vehicle's tyre pressure so that it is suitable for the surfaces it moves on. Finally, a radio interface enables interaction with external entities. In this example, the radio interface is only used to command the capture of images.

We have simulated the application on three different platform models, RENATO, JOSELITO, and BOÇA, all of them implementing a 4x4 mesh topology. The packets that the application actors send to each other have a maximum size of 48 flits, each flit of 16 bits. Empirically this packet size seems to be a good trade-off between the overhead the multiple packet headers generate

Figure 5. UML sequence diagrams of an autonomous vehicle application



versus the long term occupation of platform resources, such as channels and buffers. Application messages that are larger than 48 * 16 bits are divided in multiple packets.

We have used two different random mappings for each platform and have measured the network latency for each message of each sequence diagram. We have measured the network latency from the point the processing element sends a packet containing the message to the point the packet arrives at the processing element of the target node. That is, how long a packet spends in the network, either routed between switches or in an input buffer of a switch waiting for routing. Therefore, the latency of messages depends on the network congestion, data size of the corresponding packet, and the mapping (that is, if the sending and receiving actors of the packet are mapped to nodes that are close to each other or not).

We have fixed the operation frequency of the platforms to 50 MHz and simulated the system for 18 seconds of wall-clock time. Application-specific constraints define the execution frequency of each sequence diagram. The photogrammetry, obstacle recognition, and direction adjustment are executed once every two seconds, while the tyre pressure adjustment and snapshot request are executed once a second.

Table 1 presents the worst case latency results for two different mappings of the application over RENATO, JOSELITO, and BOÇA. Since RENATO is the closest model to a RTL implementation, it is used as a reference within this analysis. As expected, BOÇA presents a significant error for the worst case latency (around 45% in comparison with RENATO). This is due to the fact that it is not back-annotated with timing delays from a real implementation and its modelling of network arbitration is very simplistic. However, when simulating the application model in the PtolemyII environment, simulating the application over BOÇA was 402 times faster for mapping 1 and 492 times faster for mapping 2 in comparison with RENATO.

As soon as we have executed our application on BOÇA, we have used JOSELITO to extract more information about which mapping would work better when considering a mesh topology network. Even though JOSELITO has a high worst case latency error (around 30%), JOSELITO was proved to be useful, because every time JOSELITO has lower latency, RENATO has lower latency as well. The simulation of the application on JOSELITO was 2.8 times faster for mapping 1 and three times faster for mapping 2 in comparison with RENATO.

One additional observation we can make from Table 1 is related to the long worst case latency for photogrammetry and snapshot request sequence diagrams. These long latencies are due to the large size of some of their messages. Only those two sequence diagrams are involved in transfer-

Table 1. Worst case latency results of two different mappings of the same application mapped onto the platform models (latency figures in milliseconds, error percentage compared to the reference model).

	Mapping 1			Mapping 2		
	RENATO	**JOSELITO**	**BOÇA**	**RENATO**	**JOSELITO**	**BOÇA**
Direction Adjustment	0,39	0,27 (30%)	0,24 (37%)	0,38	0,26 (30%)	0,25 (32%)
Obstacle Recognition	0,12	0,09 (29%)	0,07 (41%)	0,10	0,07 (29%)	0,07 (29%)
Photogrammetry	1,41	0,99 (30%)	0,54 (62%)	1,37	0,95 (31%)	0,52 (61%)
Snapshot Request	1,35	0,94 (30%)	0,90 (33%)	1,31	0,90 (31%)	0,89 (32%)
Tyre Pressure Adjustment	0,14	0,09 (32%)	0,01 (55%)	0,11	0,08 (30%)	0,05 (50%)
Total	3,41	2,38 (30%)	1,76 (46%)	3,27	2,26 (31%)	1,78 (45%)

ring image frames from one processing element to another.

An important issue when modelling systems in different abstraction levels is the relative order of the platform models when exploring the design space of different features, such as mapping. Table I shows the relative ranking of RENATO, JOSELITO, and BOÇA for the two mappings. The same order is observed in both mappings. These results point out the fidelity among different platform models, enabling design space exploration at higher abstraction levels.

CONCLUSION AND FUTURE WORK

This article has presented an application modelling formalism that supports the joint validation of application and platform models. This approach is based on executable models, and uses a back-annotation strategy to increase the accuracy of the execution of a given application by considering the timing behaviour obtained from the platform on which the application runs.

In order to support a top-down design flow, the proposed formalism supports the successive refinement of platform models at multiple abstraction levels. Fast and abstract platform models should be used early on the design flow in order to perform rough evaluations and to rule out poorly performing platforms. Accurate models should be used later for fine-tuning platform parameters and choosing the best mapping.

We have presented three NoC models, each of them having different strengths regarding simulation speed, accuracy, observability, and modelling effort. We have simulated two different mappings of the same application on all three platform models, and therefore found out that the trends of latency are consistent across abstraction levels. The formalised definition of application, platform, and mapping proposed in this article allows a plug-and-play approach what comes to jointly validate an application mapped onto a platform. The swapping of platforms is effortless, and a given application can be simulated over different platforms by simply choosing each platform template (and mapping heuristic) from a library.

In the future, we will explore the effect of different static and dynamic mapping heuristics on the communication latency of an application. We will also consider the effect of mapping several application actors onto the same processing element in order to reduce the network latency (at the expense of increased computation times due to the multitasking overhead). Furthermore, we intend to support UML 2.0 sequence diagrams including combined fragments, so that designers can specify applications with communication behaviour that includes parallel, optional, and iterative triggering of messages.

REFERENCES

Agha, G. A. (1986). *ACTORS: A model of concurrent computation in distributed systems*. Cambridge, MA: MIT Press.

Arpinen, T., Kukkala, P., Salminen, E., Hännikäinen, M., & Hämäläinen, T. D. (2006). Configurable multiprocessor platform with RTOS for distributed execution of UML 2.0 designed applications. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 1324-1329).

Brooks, C., Lee, E. A., Liu, X., Neuendorffer, S., Zhao, Y., & Zheng, H. (2007). *Heterogeneous concurrent modeling and design in Java: Volume 1: Introduction to Ptolemy II*. Berkeley, CA: EECS Department, University of California.

Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., & Ludvig, J. (2003). Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, *91*(1), 127–144. doi:10.1109/JPROC.2002.805829

Grötker, T., Liao, S., Martin, G., & Swan, S. (2002). *System design with SystemC*. Norwell, MA: Kluwer Academic Publishers.

Hewitt, C., Bishop, P., & Steiger, R. (1973). A universal modular actor formalism for artificial intelligence. In. *Proceedings of the IJCAI*, *III*, 235–245.

Hu, J., & Marculescu, R. (2005). Energy- and performance-aware mapping for regular NoC architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *24*(4), 551–562. doi:10.1109/TCAD.2005.844106

Indrusiak, L. S., & Glesner, M. (2007). Specification of alternative execution semantics of UML sequence diagrams within actor-oriented models. In *Proceedings of the Symposium on Integrated Circuits and System Design* (pp. 330-335).

Indrusiak, L. S., Ost, L., Möller, L., Moraes, F., & Glesner, M. (2008). Applying UML interactions and actor-oriented simulation to the design space exploration of network-on-chip interconnects. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI* (pp. 491-494).

Indrusiak, L. S., Thuy, A., & Glesner, M. (2007). Executable system-level specification models containing UML-based behavioral patterns. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 301-306).

Kempf, T., Karuri, K., Wallentowitz, S., Ascheid, G., Leupers, R., & Meyr, H. (2006). A SW performance estimation framework for early system-level-design using fine-grained instrumentation. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 468-473).

Lei, T., & Kumar, S. (2003). A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proceedings of the Euromicro Symposium on Digital Systems Design* (pp. 180-189).

Määttä, S., Indrusiak, L. S., Ost, L., Möller, L., Nurmi, J., Glesner, M., et al. (2008). Validation of executable application models mapped onto network-on-chip platforms. In *Proceedings of the IEEE Symposium on Industrial Embedded Systems* (pp. 118-125).

Marcon, C., Borin, A., Susin, A., Carro, L., & Wagner, F. (2005). Time and energy efficient mapping of embedded applications onto NoCs. In *Proceedings of the Asia South Pacific Design Automation Conference* (pp. 33-38).

Martin, G. (2002). UML for embedded systems specification and design: Motivation and overview. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 773-775).

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

Murali, S., Benini, L., & De Micheli, G. (2005). Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees. In *Proceedings of the Asia South Pacific Design Automation Conference* (pp. 27-32).

Object Management Group. (2005). *UML profile for schedulability, performance, and time specification, version 1.1*. Retrieved April 27, 2009, from http://www.omg.org/docs/formal/05-01-02.pdf

Ost, L., Möller, L., Indrusiak, L. S., Moraes, F., Määttä, S., Nurmi, J., et al. (2008). A simplified executable model to evaluate latency and throughput of networks-on-chip. In *Proceedings of the Symposium on Integrated Circuits and Systems Design* (pp. 170-175).

Palma, J. C. S., Indrusiak, L. S., Moraes, F. G., Ortiz, A. G., Glesner, M., & Reis, R. (2006). Adaptive coding in networks-on-chip: Transition activity reduction versus power overhead of the Codec Circuitry. In Proceedings of *PATMOS* (pp. 603-613).

Pande, P. P., Grecu, C., Jones, M., Ivanov, A., & Saleh, R. (2005). Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Computer*, *54*(8), 1025–1040. doi:10.1109/TC.2005.134

Patel, H. D., & Shukla, S. K. (2004). *SystemC kernel extension for heterogeneous system modeling – a framework for multi-MoC modeling & simulation*. Norwell, MA: Kluwer Academic Publishers.

Riccobene, E., Scandurra, P., Rosti, A., & Bocchio, S. (2005). A SoC design methodology involving a UML 2.0 profile for SystemC. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 704-709).

Ristau, B., Limberg, T., & Fettweis, G. (2008). A mapping framework for guided design space Exploration of heterogeneous MP-SoCs. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 780-783).

Saint-Jean, N., Benoit, P., Sassatelli, G., Torres, L., & Robert, M. (2007). Application case studies on HS-scale, a MP-SOC for embbeded systems. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS)* (pp. 88-95).

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 1, edited by Seppo Virtanen, pp. 86-101, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 15
Hierarchical Agent Monitored Parallel On-Chip System:
A Novel Design Paradigm and its Formal Specification

Liang Guang
University of Turku, Finland

Juha Plosila
University of Turku, Finland

Jouni Isoaho
University of Turku, Finland

Hannu Tenhunen
University of Turku, Finland

ABSTRACT

In this paper, the authors present a formal specification of a novel design paradigm, hierarchical agent monitored SoCs (HAMSOC). The paradigm motivates dynamic monitoring in a hierarchical and distributed manner, with adaptive agents embedded for local and global operations. Formal methods are of essential importance to the development of such a novel and complex platform. As the initial effort, functional specification is indispensable to the non-ambiguous system modeling before potential property verification. The formal specification defines the manner by which the system can be constructed with hierarchical components and the representation of run-time information in modeling entities and every type of the monitoring operations. The syntax follows the standard set theory with additional glossary and notations introduced to facilitate practical SoC design process. A case study of hierarchical monitoring for power management in NoC (Network-on-chip), written with the formal specification, is demonstrated.

DOI: 10.4018/978-1-4666-0912-9.ch015

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

INTRODUCTION

Parallel many-core SoC has become a mainstream general-purpose computing platform enabled by the constant shrinking of the transistor size. Tilera processor (Bell et al., 2008) has 64 cores fabricated in 90nm technology, and TeraFLOPS (Vangal et al., 2008) has 80 cores in 65nm technology. The parallelization is expected to continue in the foreseeable future with 1000-core multiprocessors emerging soon (Asanovic et al., 2006).

Run-time monitoring has been acknowledged as an effective approach to achieve desirable features in embedded systems, including reliability, dependability and power efficiency. The approach is motivated by the "always-optimal" design paradigm (Rabaey, 2007), where the system is fine-tuned at the run-time towards optimal performance. Dynamic monitoring is effective in dealing with the increasing influence of PVT (process, voltage and thermal) variations in deep-submicron VLSI systems (Borkar et al., 2003). The variations and corresponding unpredictable behaviors affect the timing analysis, increase the fault rate, and require design considerations for each component and at each architectural level (Borkar et al., 2003; Karnik, De, & Borkar 2004). Run-time execution of a massively parallel platform can deviate significantly from static analysis. As a result, methods for dynamic monitoring of various parameters, such as power and temperature in SoC platforms (Isci, Buyuktosunoglu, Cher, Bose, & Martonosi, 2006; Brooks & Martonosi, 2001), have been proposed.

Embedding multiple agents into distributed system is an effective method to provide dynamic monitoring services (Chen, 1998; Parunak, 1998; Naji, Wells, & Etzkorn, 2004). Generally speaking, agents are autonomic and intelligent components in charge of system control responsibilities, while their exact definitions differ in various contexts. Agent-based intelligence integration is considered particularly useful in an unpredictable and highly dynamic system, such as a mobile computing platform (Chen, 1998). The behaviors of either an individual agent or a community of multiple agents have been studied (Parunak, 1998). The integration of agent intelligence can also be realized in programming (Shoham, 1993), where the behaviors of agents are specified by programmers into the source code.

Agent-based design of monitoring services can be innovatively applied to parallel embedded or on-chip systems (Naji et al., 2004), when the parallelization of on-chip components results in the complexity comparable to that of large-scale platforms. A multiple-agent-based system allows for decentralized and hierarchical control policies. Centralized controllers are common in single-core or few-core systems, while being inefficient in highly parallel platforms. Local monitoring is delayed by global communication, and the central monitor is burdened with an overwhelming amount of responsibilities taking care of all resources. Decentralized control promotes adaptive distributed computing so that each part of the system can be quickly configured. For instance, distributed thermal hotspot prevention techniques in multi-core architectures are proposed in Sartori and Kumar (2009). Hierarchical monitoring, as a further development to decentralized control (Wang & Cao, 2003), gives higher priorities to upper level monitoring operations, so that the global optimization is prioritized over local performance. In ElastIC processor (Sylvester, Blaauw, & Karl, 2006) for instance, every processing element is equipped with self-adjustment capability with a centralized diagnostic and adaptivity processing (DAP) unit in responsible of global tracking and configuring functions. In terms of the implementation methods, agents can be realized in both software and hardware manners. For instance, reconfigurable hardware can be utilized to implement dynamic agent functions (Naji, Wells, & Aborizka, 2002; Naji et al., 2004).

Formal methods are effective enablers to the development of a new design paradigm. Formal specification can be utilized to express designers'

intentions at every design level. In particular, on system-level, formal methods facilitate clear and accelerated abstraction of the system architecture. An abstracted design specification is to be elaborated iteratively until the final implementation, and each level of representation can be modeled by specific formal expressions (Edwards, Lavagno, Lee, & Sangiovanni-Vincentelli, 1997; Christopher, James, Michael, & Walter, 2006). Formal specification is especially useful to the development of a new paradigm or system, whose structures or features are not common knowledge to the designers. For example, an agent-based architecture was introduced in the design of a space flight center (Christopher et al., 2006), where formal specification was written to model the new platform as a community of ten agent entities with each agent modeled as a process in CSP language. The specification not only improves the understanding of the system for all levels of designers, but also enables future verification of properties.

We propose a novel paradigm, HAMSOC (Hierarchical Agent Monitored SoC), to realize scalable and efficient design of monitoring functionality in parallel SoCs. It offers resilience and adaptivity to each structural level of a parallel system of any size, so that dependable performance, power efficiency and required features can be achieved. Monitoring functionality is provided by hierarchically structured agents, whose individual behavior and mutual cooperation lead to monitored operations at each level of the system. As the initial development of the proposed paradigm, system-level design is focused at present, which specifies the high-level abstraction of the platform.

Formal specification of the HAMSOC platform is presented in this article. The specification models the system as being composed of hierarchical components with their corresponding agents, with monitoring operations flowing between agents and components. Each entity, either agent or component, is represented by its state, which captures its run-time status as a set of dynamic and static parameters. The monitoring operations are issued by certain levels of agents, with defined triggering conditions. Compared to the existing models of agent-based systems, the proposed specification is novel in its close relation to practical SoC/VLSI design. For instance, agents and components are abstracted as concrete entities, instead of as processes (common in software engineering). In addition, monitoring functions are specified with parameters and operations practical in SoC platforms. As a result, such specification can be directly applied to real design process. Simple syntax with notations based on the standard set theory is adopted.

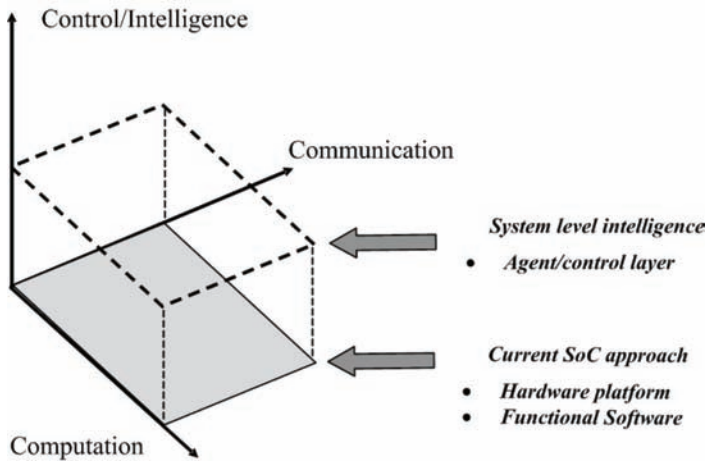
HIERARCHICAL AGENT MONITORED SoCS

Hierarchical agent monitoring is a novel paradigm for scalable and efficient design of monitoring services on parallel systems. This section presents the motivation and concise overview of the paradigm.

Monitoring-Centric Design

The full design space of embedded systems contains a wide range of aspects, including exploration of programming models, data processing mapping, interconnection, and memory architectures as well as the realization of control flow. An essential aspect of a new system design paradigm is the orthogonalization of concerns (Keutzer, Newton, Rabaey, & Sangiovanni-Vincentelli, 2000). Focused research on specific design concerns does not exclude the possible interactions with other aspects, but rather motivates the concentration of design efforts to improve the efficiency of system development. Current design space exploration mainly consists of two dimensions: computation and communication (Keutzer et al., 2000). We propose the addition of the third dimension, the control flow (here used interchangeably with monitoring

Figure 1. 3-Dimensional design space



flow), which addresses all necessary control and monitoring operations for data processing and communication (Figure 1). The separation of the monitoring layer is beneficial, considering that monitoring operations have their own behaviors and constraints, which make their abstraction as necessary as the separation of functional design and communication-based design. Moreover, the separated design space focusing on monitoring operations improves the portability of certain techniques to different platforms. Several previous works have suggested similar approaches on specific platforms, for instance Ciordas, Hansson, Goossens, and Basten (2006).

Hierarchical Agent Monitored SoCs

Hierarchical agent monitors are proposed to construct the monitoring layer in the HAMSOC system. From the top to the bottom level, we define application, platform, cluster and cell agents (Figure 2).

The hierarchy addresses the organizational structure of a parallel SoC platform, which is usually modularized as a pool of components with interconnections, as exemplified in Bell et al. (2008) and Vangal et al. (2008). Given the example of a many-core platform with NoC (network-on-chip) communication infrastructure (Figure 3), we consider a basic component unit (one processing core together with the local switch, network interface and all outgoing links) as a cell, which is monitored by one cell agent. A group of cells is dynamically configured as a cluster, monitored by a cluster agent. The whole platform, composed of run-time configured clusters, is monitored by one platform agent. The interaction between the platform and the application is managed by a software-based application agent.

Figure 2. Agent hierarchy

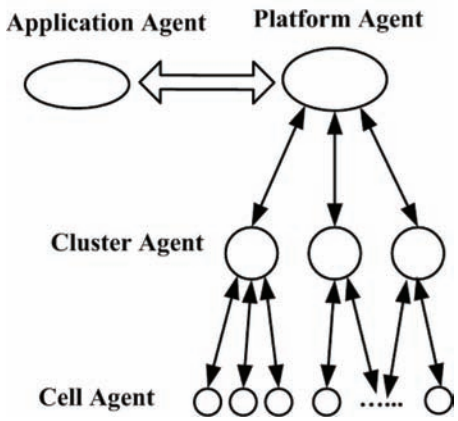
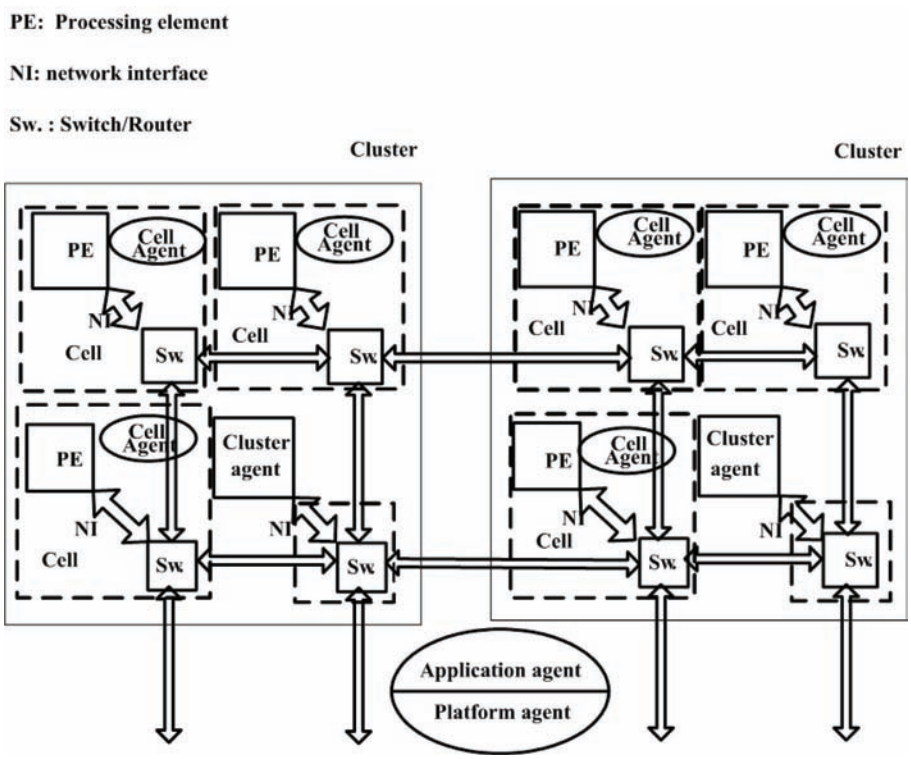


Figure 3. A segment of hierarchical-agent-monitored SoC with on-chip network

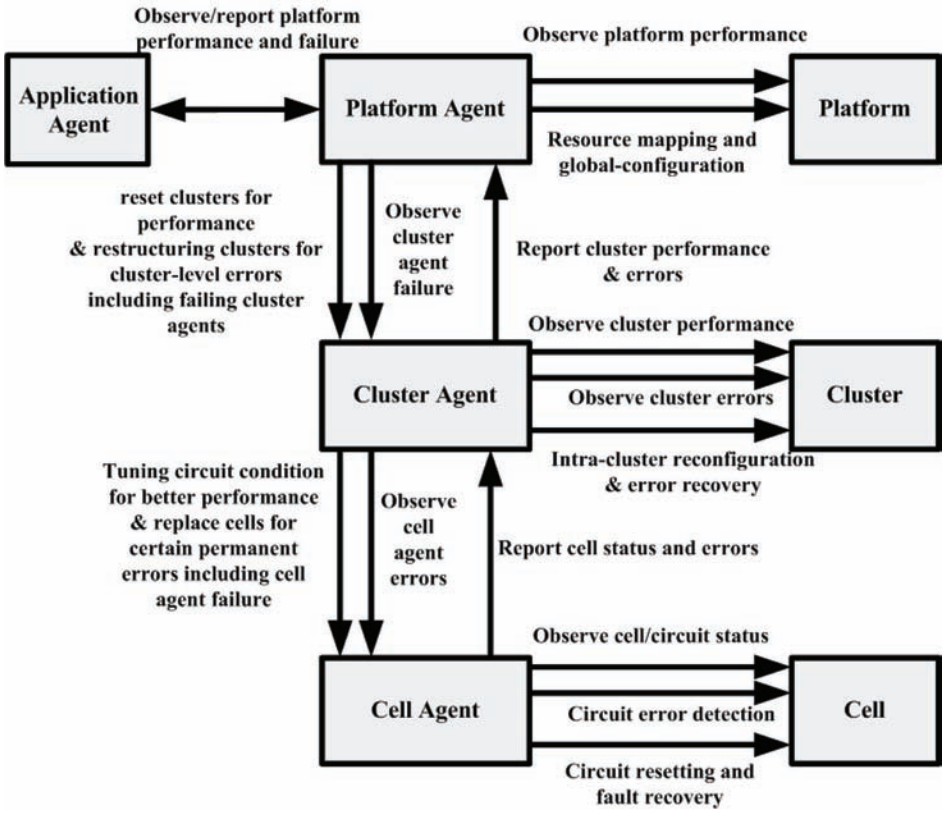


Hierarchical Monitoring Operations

The agents first of all enforce the hierarchical organization of on-chip components. The cell is the atomic monitoring unit in the system. The monitoring relation of the cell agent to the corresponding cell, which is typically a fine-grained functional module, is fixed until the cell fails. This is a design tradeoff to have a proper reconfiguration granularity. However, the affiliation of cells to clusters is dynamic. In other words, a cell can be reconfigured to a different cluster during execution. Any cluster is always initially configured by the platform agent, dependent on the computing requirements of the application. Following this organization policy, the platform is dynamically composed of clusters and non-used cells.

The agents perform hierarchical monitoring functions. The application agent provides application specification including functional and non-functional requirements, such as computing time and power constraints. The platform agent analyzes the application specification, and accordingly configures the global setting of the platform, such as network topology, resource allocation and application mapping. Cluster agents are the regional monitors in individual clusters. They set the inter-cluster configuration, for instance the supply voltage and frequency of the cluster, usually based on the application mapped on to the cluster. Cell agents are the local monitors, which implement various mechanisms for tuning and monitoring of fine-grained parameters, for instance communication speed in individual channels or timing errors on a particular critical path. In addition, in order to provide hierarchical performance monitoring and fault-tolerance, each level of agents need to report the system performance of their corresponding level and any detection of fault to their higher-level agents. Since agents themselves are possible victims of faults and errors, the error states of agents are also traced

Figure 4. Typical hierarchical monitoring operations on SoCs



by their higher-level agents. Typical hierarchical monitoring operations on parallel SoC platforms are illustrated in Figure 4, while elaborated discussions can be found in Yin et al. (2008).

Hierarchical agent monitoring framework is intrinsically scalable from the perspective of design methods. The partitioning of monitoring functions onto hierarchical levels is intended for accommodating any number of resources. The hierarchy ensures that higher-level monitoring operations always take priority over low-level ones, which provides global-level optimization. Moreover, high level agents do not monitor low level components directly, which provides a proper encapsulation of low-level details. It should also be noted that the concept of clusters is scalable as clusters can be firstly grouped into super-clusters in case of a very large number of such clusters. In addition, the monitoring hierarchy can be efficiently expanded to multi-application-multiple-platform systems, in which case additional interactions are needed to handle the arbitration between different application agents and platform agents. The description in this paper is based on a single-application-single-platform system with one level of clusters.

GLOSSARY AND NOTATIONS

The formal specification is consistent with the syntax of the standard set theory, with some additional notations (Table 1) introduced to specify particular concepts in practical design process of the HAMSOC system. The proper understanding of these symbols requires the explanation of related terms and concepts listed below:

Table 1. Notations

Notation	Explanation
Names of Agents and Components	
$\boldsymbol{Agent_i}$	Index name for any agent
AAG	Application agent
PLAG	Platform agent
CLAG	Cluster agent
CEAG	Cell agent
$\boldsymbol{Comp_i}$	Index name for any component
Platform	The platform (monitored by the platform agent)
Cluster	A cluster
Cell	A cell
FU	The functional unit within a cell and monitored by a cell agent.
$\boldsymbol{Comp/Agent_i^{i=1..N}}$	listing all concerned components or agents, written in their index names.
Representation of State and Monitoring Operations	
{ }	A set, used to specify the representation of an entity or a monitoring operation.
$=_{def}$	The symbol to introduce the representation of component/agent state, or the format of a field.
:	The symbol to extract a field or subfield (right side) from a component or agent state (left side). For example, ***PLAG*: *ErrorState*** (The error state of the platform agent) A subfield can be extracted from the representation directly without showing the upper level field first, in order to reduce the expression length (given the constraints that the subfields and fields do not have the same phrase).
/	The symbol either connects optional fields in one position within a set, or alternative values of a field.
=	Static assignment of values (right side) to a field or subfield (left side)
:=	Dynamic assignment of values (right side) to a field or subfield (left side)
$\equiv \neq >$	Comparison operators (equivalent, non-equivalent, larger), used to evaluate the value of a parameter. The value types allowed for a particular operator are defined by designers. Similar operators can be defined, for instance $\leq$ (equivalent or less than).
$\oplus$	An operator to express the structure of a component. Sub-components affiliated to the component and the agent monitoring the component can be connected with the symbol (in no particular order) to represent the dynamic organization of the component.
N/A	To denote a field which is in the generic representation format but not used in that of a particular entity or monitoring operation.

- **Agent**: Agent in general refers to the monitor of a structural group. In the defined hierarchy, there is one cell agent in each cell, and one cluster agent in each cluster, and one platform agent for the whole platform. The only exception is application agent, which is not directly monitoring any resources, but capturing the dynamic requirements of the application. Agent is always defined relative to the structural group it monitors. For instance, a cell agent is considered as an agent if we are designing the cell, but considered as part of a component (the cluster) from the perspective of the cluster agent.
- **Component**: Component in general refers to the resource that an agent monitors. In the defined hierarchy, the component monitored by a cell agent is a cell. The component monitored by a cluster agent is a

cluster, and the component monitored by a platform agent is the platform. Component is defined relative to a particular agent.

- **Entity**: A general name referring to either an agent or component.
- **Index Name**: A short-hand notation of an entity with a numerical subscript as the index, for instance $Cluster_1$. The index name is used to distinguish the entity, and does not contain any information of the entity. If a component has the same subscript (in its index name) as that of the agent on the same level, it implies the agent is directly monitoring the component.
- **State, Component State, Agent State, Representation of State**: State in general refers to the dynamic condition of a component or agent. It is usually captured by the values of any parameters (physical, electrical or logical) of the entity. Theoretically, not all parameters are observable or accessible at the run-time, but only those which can be observed or accessed are taken account in the design process. The low-level design determines whether a parameter is observable and accessible. For instance, the integration of error detection circuitry makes the error state of the corresponding component observable. We define the representation of the state of an entity as a set, which lists all considered parameters. The format of the set is formalized so that it is understood in the same way throughout the design process. To simplify the writing, in the following explanation, the state of an entity always refers to the representation of the state unless stated otherwise.
- **Direct Monitoring and Remote Monitoring**: Direct and remote monitoring are two kinds of monitoring relations between an agent and another entity. Section "Direct and Remote Monitoring" will explain them in detail.
- **Monitoring Operation**: Monitoring operation is a dynamic operation, initiated by an agent, triggered by defined conditions to perform observation or reconfiguration actions. Monitoring operations are classified into several types. Each type specifies a category of operations on or across particular hierarchical levels. The representation of a monitoring operation is a set, which lists all information needed to specify the operation in a formalized format.
- **Field**: One element in the state of an entity, or in the representation of a monitoring operation.
- **Subfield**: If one field is a set itself, an element within the set is a subfield.
- **Phrase**: The textual name of a field or a subfield. For instance, the current of a circuit can be named as "Current", in contrast to the value of the current "3A".
- **Value**: The actual value of a field or a subfield. The value can be of any type, for instance numerical, Boolean or enumeration, as defined by the designers.
- **Static**: If the value of a parameter is assigned at design time and remains constant during execution, it is a static parameter. The assignment of a value to a static parameter is a static operation (finished before execution).
- **Dynamic**: If the value of a field or a subfield is assigned during execution, or can be changed during execution, the field or subfield represents a dynamic parameter. Its assignment operation is a dynamic operation.
- **Observable**: If the value of a parameter can be observed during execution, it is an observable parameter. An observable parameter is usually dynamic, except for certain static parameters, whose values require run-time checking in case of errors.
- **Reconfigurable**: If the value of a parameter can be changed during execution, it is a

reconfigurable parameter. A reconfigurable parameter is always dynamic.

States of Components and Agents

The states of components and agents are formally represented as sets, listing all parameters considered in the design. The representations of component and agent state follow a similar format, but they contain different fields based on specific design considerations.

Component State

The representation of a component state is formatted as follows:

$$\boldsymbol{Comp_i} =_{def} \{\boldsymbol{Type}, \boldsymbol{Affi}, \boldsymbol{St}ruc, P_{STA,} P_{OBS,} P_{REC}\}$$

Each field in the set is explained below:

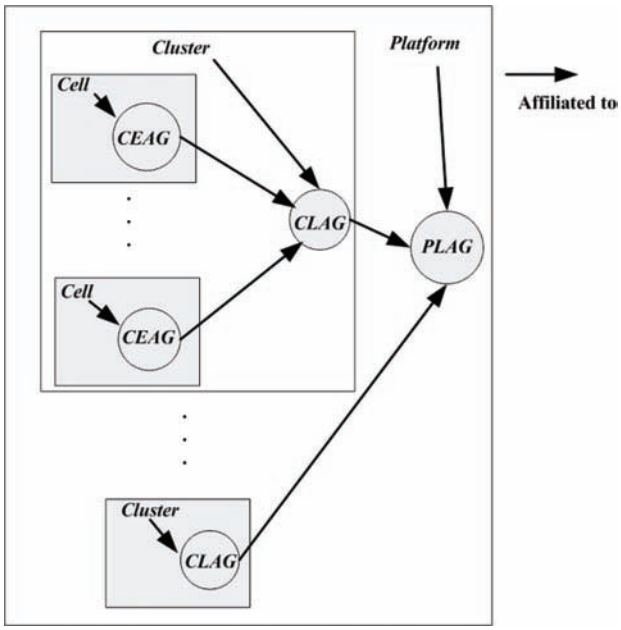
- ***Type***: The type of component, enumeration value. Possible types, as defined here, are *Cell*, *Cluster* and *Platform*. The functional unit within a cell does not have a separate state representation.
- ***Affi***: The agent that an entity is affiliated to. This field specifies the location of the entity in the hierarchy of the platform. A component is affiliated to its directly monitoring agent. An agent is affiliated to a higher level agent which is directly monitoring the higher level component. Such iterative affiliation relations build up the monitoring hierarchy until the level of the platform agent (Figure. 5). The platform agent and application agent do not have further affiliations.

As an example, a cell is affiliated to its cell agent, which is affiliated to the higher level cluster agent. The affiliation relations can be formalized as:

$$Cell_i : Affi = CEAG_i$$

$$\boldsymbol{CEAG_i} : \boldsymbol{Affi} := \mathrm{CLAG_j}$$

Figure 5. Illustration of affiliation relation in the HAMSOC platform



The affiliation of a cell to its cell agent is static, so the notation "=" is used, and the index names of the cell and its cell agent have the same subscript. The affiliation of a cell agent to the cluster agent can be dynamically configured, thus notation " := " is used. If a cell is not affiliated to any cluster, its affiliation will be denoted as ***Non_Assigned***.

- ***Struc***: The structure of a component. Each component may consist of several sub-components and agents. All components higher than cells have dynamic structures, implying that the affiliation of sub-components to the component is dynamically configured.

$\oplus$ is the notation (Table 1) to express the connection of subcomponents or agents to build up the structure of a component. As an example:

$$\boldsymbol{Cell_i : Struc = CEAG_i} \oplus \{ \boldsymbol{FU} \mid \boldsymbol{FU}: \boldsymbol{Affi} \equiv \mathrm{CEAG}_i \}$$

The structure of a cell contains the cell agent, and all functional units affiliated, statically, to the cell agent.

$$\boldsymbol{Cluster_i : Struc} := \mathrm{CLAG}_i \oplus \{ \boldsymbol{Cell} \mid (\boldsymbol{Cell}: \boldsymbol{Affi}): \boldsymbol{Affi} \equiv \mathrm{CLAG}_i \}$$

The structure of a cluster contains the cluster agent, and all cells whose cell agents are affiliated to the cluster agent. (***Cell*** : Affi **)** denotes the cell agent monitoring the cell.

$$\boldsymbol{Platform}: \boldsymbol{Struc} := \mathrm{PLAG} \oplus \{ \boldsymbol{Cluster} \mid (\boldsymbol{Cluster}: \boldsymbol{Affi}): \boldsymbol{Affi} \equiv \mathrm{PLAG} \}$$

$$\oplus \{ \mathit{Cell} \mid (\mathit{Cell}: \mathit{Affi}): \mathit{Affi} \equiv \mathrm{Non_Assigned} \}$$

The structure of a platform contains the platform agent and all clusters whose cluster agents are affiliated to the platform agent, as well as all cells which are not assigned to any clusters at the moment.

$\boldsymbol{P_{STA}}$, $\boldsymbol{P_{OBS}}$, $\boldsymbol{P_{REC}}$: $\boldsymbol{P_{STA}}$ is the list of all static parameters of the component. $\boldsymbol{P_{OBS}}$ is the list of all observable parameters. $\boldsymbol{P_{REC}}$ is the list of all reconfigurable parameters of the component. These lists specify the classification of all available parameters as intended on the system-level abstraction, which need to be validated by the low-level feasibility. As a simple example, if a NoC platform is designed with these features: the voltage of all network links are fixed at 1.5V, the current on the links can be detected by a current detector, and the frequency of the links can be reconfigured, the following specification can be formalized:

$$\boldsymbol{Link}_i^{i=1..N} : \boldsymbol{P_{STA}} =_{def} \{ \boldsymbol{Voltage} = 1.5\mathrm{V} \}$$

$$\boldsymbol{Link}_i^{i=1..N} : \boldsymbol{P_{OBS}} =_{def} \{ \boldsymbol{Current} \}$$

$$\boldsymbol{Link}_i^{i=1..N} : \boldsymbol{P_{REC}} =_{def} \{ \boldsymbol{Freq} \}$$

Agent State

The agent state adopts a similar representation to the component state:

$$\boldsymbol{Agent_i} =_{def} \{ \boldsymbol{Type, Affi, P_{OBS}, MoniCap} \}$$

Each field of the set is explained below:

- ***Type***: The type of agent, enumeration value. Possible types defined here are ***CEAG***, ***CLAG***, ***PLAG*** and ***AAG***.
- ***Affi***: The affiliation of an agent is the agent monitoring the higher level component (Fig. 5).
- $\boldsymbol{P_{OBS}}$: The list of all observable parameters of the agent, such as *ErrorState* (error state). Current specification does not consider the configuration within agents

themselves, thus no $\boldsymbol{P_{STA}}$ (static parameters) or $\boldsymbol{P_{REC}}$ (reconfigurable parameters) are specified.

***MoniCap**:* The list of all types of monitoring operations (Section "Monitoring Operations") allowed for an agent. We here define seven types of monitoring operations (Figure 6), and specify the *MoniCap* of each agent level as follows:

AAG: *MoniCap* = { *Communicate* }

PLAG: *MoniCap* = { *Communicate*, *Inquire*, *Trace*, *Order*, *Configure* }

$CLAG_i$: *MoniCap* = { *Inform*, *Inquire, Trace*, *Order*, *Configure* }

$CEAG_i$: *MoniCap* = { *Inform*, *Report*, *Trace*, *Configure*}

The representation of agent does not contain a field expressing its structure, since the internal structure of any agent is not considered in the current design abstraction.

Direct and Remote Monitoring

To differentiate the hierarchical levels between agents and entities they monitor, we introduce two terms: direct monitoring and remote monitoring.

Direct monitoring refers to the situation that a component or an agent is directly affiliated to another agent. For instance, a cluster agent is directly monitoring its cluster and the cell agents of all cells which are configured into the cluster. In formalized expressions:

$$(\boldsymbol{Agent_i} / \boldsymbol{Comp_i} : \boldsymbol{Affi}) \equiv \mathrm{Agent_j}$$

$\boldsymbol{Agent_i}$ and $\boldsymbol{Comp_i}$ are directly monitored by $\boldsymbol{Agent_j}$. We can also say that $\boldsymbol{Agent_j}$ is directly monitoring $\boldsymbol{Agent_i}$ and $\boldsymbol{Comp_i}$.

Remote monitoring refers to the situation that an agent resides on a higher level than the agent directly monitoring an entity, but it can still monitor the entity through iterative affiliation relation downward the hierarchy. In formalized expressions:

$$(\boldsymbol{Agent_i} / \boldsymbol{Comp_i} : \boldsymbol{Affi})_{K\,(K>1)} \equiv \mathrm{Agent_j}$$

$\boldsymbol{Agent_j}$ is monitoring $\boldsymbol{Agent_i}$ and $\boldsymbol{Comp_i}$, in K times iteration.

$(\boldsymbol{Agent_i} / \boldsymbol{Comp_i} : \boldsymbol{Affi})_K$ is defined as follows:

$$(\boldsymbol{Agent_i} / \boldsymbol{Comp_i} : \boldsymbol{Affi})_{K\,(K>1)} \equiv (\boldsymbol{Agent_i} / \boldsymbol{C}omp_i : Affi)_{K-1} : Affi$$

$$(\boldsymbol{Agent_i} / \boldsymbol{Comp_i} : \boldsymbol{Affi})_1 \equiv \mathrm{Agent_i} / Comp_i : Affi$$

For instance, the platform agent is remotely monitoring a configured cell agent with K=2.

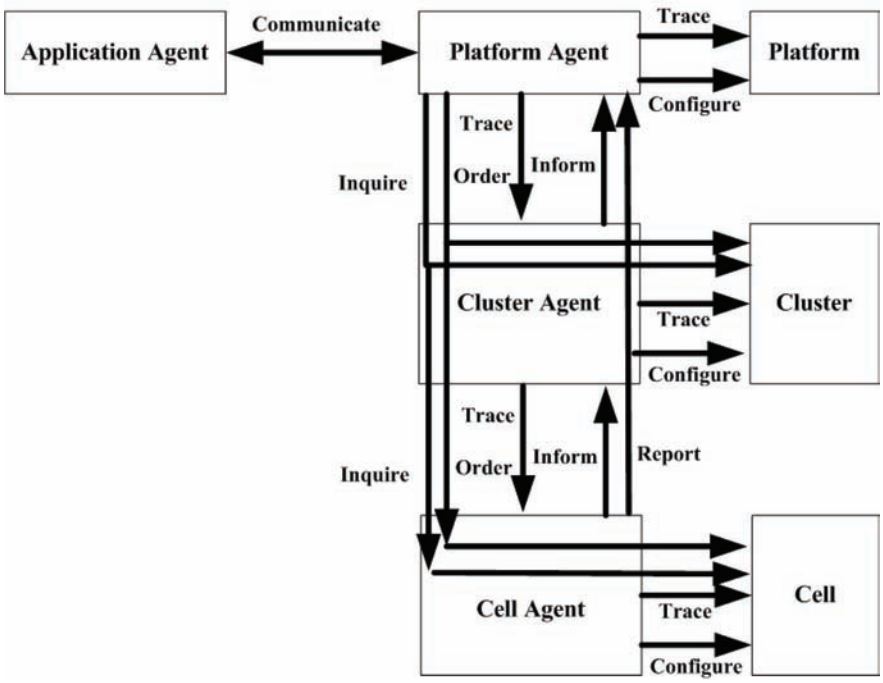
Monitoring Operations

Monitoring operations are formally specified as sets with all required information listed. To support modularized design and implementation of similar operations, they are classified into seven types, which are able to express all generic monitoring functions considered in the hierarchy so far.

Types of Monitoring Operations

A particular type of monitoring operation is specified to represent one group of operations, which share a similar purpose and hierarchical flow. Seven types of monitoring operations are defined (Figure 6): *Communicate*, *Trace*, *Configure*, *Inquire*, *Order*, *Report* and *Inform*. Such a classification of types considers both the similarity of functionality and hierarchical positions. The monitoring operations are used to fulfill two generic functions: observing run-time states of component or agent, and reconfiguring parameters of components. *Communicate*, *Trace*, *Inquire*, *Report* and *Inform* are used for the former

Figure 6. Types of monitoring operations defined on each level of agents



purpose, while *Configure* and *Order* are for the latter purpose. In terms of hierarchical positions, *Communicate* operation is defined between application and platform agents, as they are not in hierarchically different levels but work in parallel. Other types of monitoring operations all flow across different hierarchical levels, as illustrated in Figure. 6.

Monitoring operations of the same type share similar design and implementation considerations, including communication protocol design. As an example, *Trace* is defined to specify operations issued by the initiating agent to check the state of its directly monitored agent or component. *Trace* operation goes directly from the initiator to the target (without any other agents in between), and implicitly requires a reply message containing the value of the traced parameter. However, in *Inquire* operation, which is used to specify operations issued by the initiating agent to check the state of its remotely monitored agent or component, the communication goes across at least another level of agent.

Format

The representation of monitoring operations follows the generic format as below:

$\{ Type, InAgent, Condition, Message, ReAgent,$

$(Comp_i : Struc / P_{OBS} / P_{REC}) / (Agent_i : P_{OBS}),$
$NewValue \}$

Each field of the set is explained below:

- ***Type***: The type of monitoring operation, enumeration value. Possible types defined here are *Communicate*, *Trace*, *Configure*, *Inquire*, *Order*, *Report* and *Inform*.
- ***InAgent***: The initiating agent of the monitoring operation, written in its index name.

- *Condition: Condition* specifies when the monitoring operation is to be issued, and can be expressed with static or dynamic timing.

Static timing condition is to capture particular and periodic timestamps pre-known at design time. Notations of particular times are defined by the designers, for instance ***SystemStartup*** can be used to denote the startup time of the system. Notation of periodic times is formalized as:

$(StartTime, (N, CLK_{Agent_i}))$

It defines the start time (*StartTime*), and the interval $(\boldsymbol{N}, \boldsymbol{CLK_{Agent_i}})$. *StartTime* denotes a particular timestamp. The interval is denoted as a number (***N***) of cycles of an agent ($\boldsymbol{Agent_i}$).

Dynamic timing is defined as an evaluation of the run-time values of fields or subfields in an agent or component state. As an exception, for static parameters, the evaluation can be based on their static values. The evaluation can use any comparison operator defined, for instance $\equiv \neq >$ as in Table 1. As an example:

$\boldsymbol{Cluster_i}$ **:** ***Voltage*** > 5V

The condition evaluates if the voltage of the cluster is larger than 5 volts. Whenever the evaluation is true, the operation is triggered.

- ***Message:*** The information sent in *Communicate* operation. *Communicate* operation is not a command, and allows flexible information exchange between the application and platform agent, for instance

Application**:** *MaximalPower* (the maximal allowable power of running the application)

- ***ReAgent:*** The receiving agent of the *Communicate* or *Report* operation.
- $(\boldsymbol{Comp_i}$**:** ***Struc*** / $\boldsymbol{P_{OBS}}$ / $\boldsymbol{P_{REC}})$ / $(\boldsymbol{Agent_i}$ **:** $\boldsymbol{P_{OBS}})$: This field specifies the parameter to be traced or reconfigured. It can be the structure (***Struc***), an observable parameter (one subfield in $\boldsymbol{P_{OBS}}$) or a reconfigurable parameter (one subfield of $\boldsymbol{P_{REC}}$) of a component. It can also be an observable parameter of an agent.
- ***NewValue:*** The new value that is to be assigned to the component in a reconfiguration (*Order* or *Configure* operation).

Specification of Each Type of Monitoring Operations

Based on the generic format of the monitoring operations, each type of operations can be specified as below:

Communicate**:** *Communicate* is used to exchange information between application and platform agent, written in the following format:

{ *Communicate*, *InAg*, *Condition*, *Message*, *ReAg*, *N/A*, *N/A* }

Trace: ***Trace*** is issued by one agent to check the run-time value of an observable parameter of an agent or component directly monitored by the initiating agent, written in the following format:

{ *Trace*, $Agent_i$, *Condition*, *N/A*, *N/A*,

$(\boldsymbol{Comp_j} \mid \boldsymbol{Comp_j}$**:** ***Affi*** $\equiv \mathrm{Agent_i})$ / $(\boldsymbol{Agent_j} \mid \boldsymbol{Agent_j}$ **:** ***Affi*** $\equiv \mathrm{Agent_i})$**:** $\boldsymbol{P_{OBS}}$, ***N/A*** }

Configure: ***Configure*** is issued by an agent to modify a reconfigurable parameter or the structure of a component that is directly monitored by the agent, written in the following format:

{ *Configure*, $Agent_i$, *Condition*, *N/A*, *N/A*,

$(\boldsymbol{Comp_j} \mid \boldsymbol{Comp_j}$**:** ***Affi*** $\equiv \mathrm{Agent_i})$**:** ***Struc*** / $\boldsymbol{P_{REC}}$, ***NewValue*** }

Inquire: ***Inquire*** is issued by one agent to check the run-time value of an observable parameter of an agent or component remotely monitored by the initiating agent, written in the following format:

{ *Inquire*, $Agent_i$, *Condition*, *N/A*, *N/A*,

$(\boldsymbol{Comp_j} \mid (\boldsymbol{Comp_j}\mathbf{:}\ \boldsymbol{Affi})_{K(K>1)} \equiv \mathbf{Agent_i}\ ) / (\mathbf{Agent_j} \mid (Agent_j : Affi)_{K(K>1)} \equiv \mathrm{Agent_i})\mathbf{:}\ \mathbf{P_{OBS}}, \mathbf{N/A}\ \}$

Order: ***Order*** is issued by an agent to modify a reconfigurable parameter or the structure of a component remotely monitored by the agent, with the following format:

{ *Order*, $Agent_i$, *Condition*, *N/A*, *N/A*,

$(\boldsymbol{Comp_j} \mid (\boldsymbol{Comp_j}\mathbf{:}\ \boldsymbol{Affi})_{K(K>1)} \equiv \mathbf{Agent_i})$: Struc / P_{REC}, *NewValue* }

Inform: ***Inform*** is issued by one agent to initiatively report the run-time value of an observable parameter of an entity directly monitored by the initiating agent, to its directly monitoring agent. It is written in the following format:

{ *Inform*, $Agent_i$, *Condition*, *N/A*, *N/A*,

$(\boldsymbol{Comp_j} \mid \boldsymbol{Comp_j}\mathbf{:}\ \boldsymbol{Affi} \equiv \mathrm{Agent_i}) / (\boldsymbol{Agent_j} \mid \boldsymbol{Agent_j}\mathbf{:}\ \boldsymbol{Affi} \equiv \mathrm{Agent_i})\mathbf{:}\ \boldsymbol{P_{OBS}}, \boldsymbol{N/A}\ \}$

Report: ***Report*** is issued by one agent to report the run-time value of an observable parameter of an entity directly monitored by the initiating agent, to one of its remotely monitoring agents. It is written in the following format:

$\{\ \boldsymbol{Report}, \boldsymbol{Agent_i}, \boldsymbol{Condition}, \boldsymbol{N/A}, \boldsymbol{Agent_j} \mid (\boldsymbol{Agent_i}\mathbf{:}\ \boldsymbol{Affi})_{K(K>1)} \equiv \mathrm{Agent_j},$

$(\boldsymbol{Comp_t} \mid \boldsymbol{Comp_t}\mathbf{:}\ \boldsymbol{Affi} \equiv \mathrm{Agent_i}) / (\boldsymbol{Agent_t} \mid \boldsymbol{Agent_t}\mathbf{:}\ \boldsymbol{Affi} \equiv \mathrm{Agent_i})\mathbf{:}\ \boldsymbol{P_{OBS}}, \boldsymbol{N/A}\ \}$

CASE STUDY

Hierarchical Power Management for On-Chip Communication

Here the formal specification is applied to model hierarchical power monitoring services for on-chip communication in NoC platforms. Inter-router communication is one major power contributor in multi/many-core SoCs (Soteriou & Peh, 2007), and it will be more substantial with further parallelization into thousand-core SoC era. Dynamic traffic loads of on-chip communication have temporal and spatial variation. Run-time tradeoff of transmission performance and power is expected to provide maximal power/energy efficiency with minimal performance penalty (Guang, Nigussie, Koskinen, & Tenhunen, 2009; Shang, Peh, & Jha, 2003).

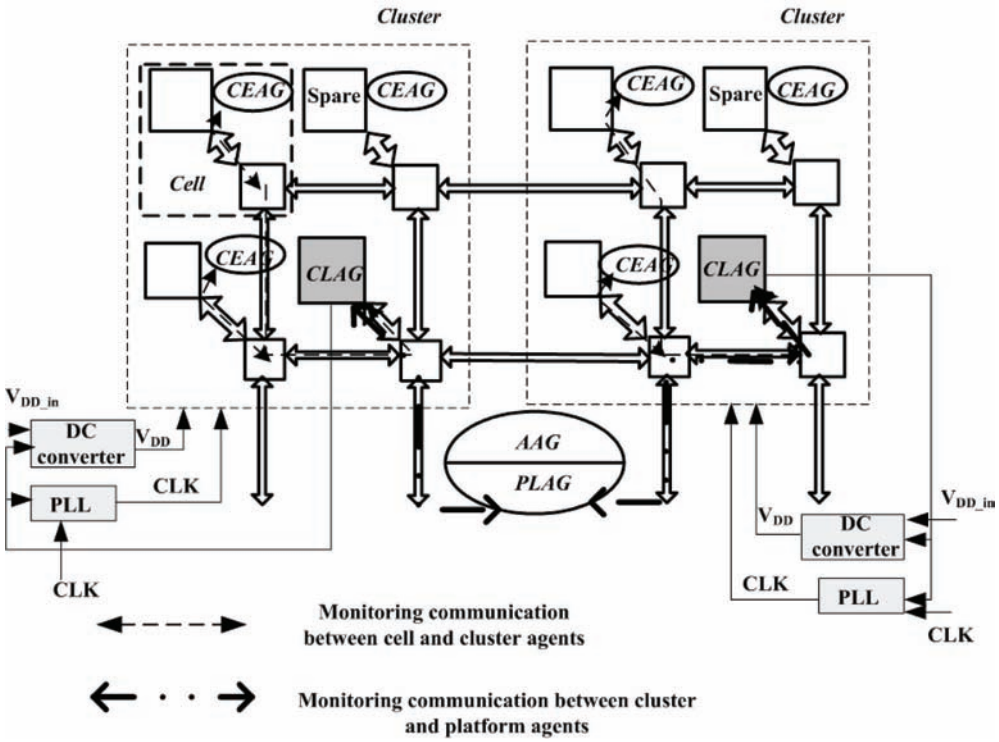
Scenario Illustration

A modularized many-core platform (Figure. 7), with $K * N$ cores, is assumed. The platform assigns M processing cores ($M \leq N$) to one cluster with K clusters in total, and the rest will be used as spares in case of failures. Each cluster is equipped with one DC converter and one PLL (phase-locked loop), to provide dynamically-variable supply voltage and working frequency respectively. They are both configured by the cluster agent. Each cell agent traces the buffer load in the local router, and reports the load and any connection error to the cluster agent.

The agent hierarchy applies cluster-based DVFS (Dynamic Voltage and Frequency Scaling) (Guang et al., 2009). Each cluster island can be provided with multiple voltage and frequency levels. When the local traffic load, quantified by the percentage of buffers occupied, is higher than a defined threshold

$\boldsymbol{L_{upper}}$, the cluster is to be equipped with a higher level frequency and voltage supply. If the local traffic load is lower than another threshold $\boldsymbol{L_{lower}}$,

Figure 7. Segment of the NoC platform with hierarchical power monitoring operations



a lower level frequency and voltage supply will be provided. In this manner, the network will run at the minimal power consumption with no local traffic congestion in any cluster. Such dynamic power tradeoff method has been studied and analyzed, on different granularities in (Guang & Jantsch, 2006; Guang et al., 2009; Yin, Guang, Nigussie, Liljeberg, & Tenhunen, 2009).

The average buffer load (in a history window) in each cell is measured by the cell agent, and reported to the cluster agent. Each cell agent also detects and reports failures in local network connection to the cluster agent. The cluster agent dynamically adjusts the level of voltage and frequency of the cluster, based on the average of buffer load reported by all affiliated cell agents. In case of cell failures, it replaces the failed cell with a spare one. The cluster agent also traces the power consumption of the cluster (for instance by special circuitry on the DC converter (Chen, 2008)), and reports it to the platform agent. The platform agent determines the initial clusterization, and dynamically sends total platform power to the application agent.

Component and Agent States

The states of each component and agent are formalized as follows:

Platform $=_{def}$ { ***Platform***, ***N/A***, ***Struc***, { }, { ***ErrorState***, ***NetPower***}, { } }

Platform: ***Struc*** := PLAG ⊕ { ***Cluster*** | (***Cluster***: ***Affi***)$_2$ ≡PLAG}

⊕ { ***Cell*** | (***Cell***: ***Affi***)$_2$ ≡Non_Assigned}

PLAG $=_{def}$ { ***PLAG***, ***N/A***, { }, {***Communicate***, ***Trace***, ***Configure***, ***Inquire***, ***Order***}}

The top level platform has two observable parameters, error state (***ErrorState***) and network power (***NetPower***), with no static and reconfigurable parameters ({} represents an empty set). The structure of the platform contains the platform agent, all configured clusters and all spare cells.

$$\boldsymbol{Cluster}_i^{i=1..K} =_{def} \{ \text{Cluster}, \text{CLAG}_i, \boldsymbol{Struc}, \{\}, \{\boldsymbol{ClusterLoad}, \boldsymbol{ClusterPower}\}, \{\boldsymbol{Voltage}, \boldsymbol{Frequency}\}\}$$

$$\boldsymbol{Cluster}_i^{i=1..K} : \boldsymbol{Struc} := \text{CLAG}_i \oplus \{ \boldsymbol{Cell} \mid \boldsymbol{Cell}: \boldsymbol{Affi} \equiv \text{CLAG}_i\}$$

$$\boldsymbol{CLAG}_i^{i=1..K} =_{def} \{\boldsymbol{CLAG}, \boldsymbol{PLAG}, \{\boldsymbol{ErrorState}\}, \{ \mathit{Trace}, \mathit{Configure}, \mathit{Inquire}, \mathit{Order}, \mathit{Inform}\}\}$$

Each cluster is affiliated to its cluster agent, and its power consumption and average buffer load are observable parameters. Voltage and frequency of the cluster can be dynamically adjusted.

$$\boldsymbol{Cell}_i^{i=1..K*N} =_{def} \{ \boldsymbol{Cell}, \boldsymbol{CEAG}_i, \boldsymbol{Struc}, \{\}, \{\boldsymbol{Error}\mathit{State}, \mathit{BufferLoad}\}, \{\}\}$$

$$\boldsymbol{Cell}_i^{i=1..K*N} : \boldsymbol{Struc} = \text{CEAG}_i \oplus \{ \boldsymbol{FU} \mid \boldsymbol{FU}: \boldsymbol{Affi} \equiv \text{CEAG}_i\}$$

$$\boldsymbol{CEAG}_i^{i=1..K*N} =_{def} \{\boldsymbol{CEAG}, \boldsymbol{Non_Assi}\mathit{gned} / \mathit{CLAG}_j, \{\mathit{ErrorState}\}, \{\mathit{Trace}, \mathit{Configure}, \mathit{Report}, \mathit{Inform}\}\}$$

Each cell is affiliated to a cell agent, with its error state and bufferload observable. A cell agent is either dynamically affiliated to a cluster agent or not configured to any cluster at the moment (***Non_Assigned***).

Monitoring Operations

With the presented specification, the required operations for the hierarchical power monitoring can be formalized as below:

$$\{\boldsymbol{Order}, \boldsymbol{PLAG}, \boldsymbol{SystemStartup}, \boldsymbol{N/A}, \boldsymbol{N/A}, \boldsymbol{CEAG}_i^{i=1..M*K} : \mathit{Affi}, \mathit{CLAG}_{mod(i,K)}\}$$

$$\{\boldsymbol{Order}, \boldsymbol{PLAG}, \boldsymbol{SystemStartup}, \boldsymbol{N/A}, \boldsymbol{N/A}, \boldsymbol{CEAG}_i^{i=M*K+1..N*K} : \boldsymbol{Affi}, \boldsymbol{Non_Assigned}\}$$

$$\{\boldsymbol{Configure}, \boldsymbol{PLAG}, \boldsymbol{SystemStartup}, \boldsymbol{N/A}, \boldsymbol{N/A}, \boldsymbol{CLAG}_i^{i=1..K} : \mathit{Affi}, \mathit{PLAG}\}$$

$$\{\mathit{Communicate}, \mathit{PLAG}, (\mathit{SystemStartup}, (10^4, \mathit{CLK}_{PLAG})), \mathit{Platform}: \mathit{NetPower}, \mathit{AAG}, \mathit{N/A}, \mathit{N/A}\}$$

The platform agent configures M*K cells into K clusters, with a modular mapping, at the startup of the system (***SystemStartup***). ***Mod (i, j)*** is the modular function. At the run-time, the platform periodically sends the network power to the application agent by ***Communicate*** operation.

$$\{\boldsymbol{Configure}, \boldsymbol{CLAG}_i^{i=1..K}, \mathit{Cluster}_i: \mathit{Cluster\text{-}Load} > \text{L}_{upper}, \boldsymbol{N/A}, \boldsymbol{N/A}, \mathit{Cluster}_i : \mathit{Voltage}, \mathit{HigherVoltage}\ (\mathit{Cluster}_i : \mathit{Voltage})\}$$

$$\{\boldsymbol{Configure}, \boldsymbol{CLAG}_i^{i=1..K}, \mathit{Cluster}_i: \mathit{Cluster\text{-}Load} > \text{L}_{upper}, \boldsymbol{N/A}, \boldsymbol{N/A}, \mathit{Cluster}_i : \mathit{Freq}, \mathit{HigherFreq}\ (\mathit{Cluster}_i : \mathit{Freq})\}$$

$$\boldsymbol{Cluster_i}: \boldsymbol{ClusterLoad} := \mathit{Avg}\ (\{ (\mathit{Cell} \mid (\mathit{Cell}: \mathit{Affi})_2 \equiv \mathit{CLAG}_i) : \mathit{BufferLoad}\})$$

Each cluster agent adjusts the voltage and frequency level of the local cluster dynamically, based on the average of buffer loads informed

by all cell agents affiliated to the cluster. ***Avg*** **()** is the function returning the average of all input parameters. Here only the specification for the upward scaling is shown, leaving that of downward scaling omitted. ***HigherVoltage*** **()**, ***HigherFreq*** **()** return the higher level voltage and frequency respectively.

$\{\boldsymbol{Configure}, \boldsymbol{CLAG}_i^{i=1..K}, (\boldsymbol{Cell}_j \mid (\boldsymbol{Cell}_j\boldsymbol{:}\ \boldsymbol{Affi})_2 \equiv CLAG_i)$: $ErrorState \equiv$Failure,

$\boldsymbol{N/A}, \boldsymbol{N/A}, \boldsymbol{CEAG}_t\boldsymbol{:}\ \boldsymbol{Affi} \mid \boldsymbol{CEAG}_t\boldsymbol{:}\ \boldsymbol{Affi} \equiv$Non_Assigned, $\mathrm{CLAG}_i\}$

In case a cell suffers from (permanent) failure, one of the spare cells will be assigned by the cluster agent to replace the failed one.

$\{\ \boldsymbol{Inform}, \boldsymbol{CLAG}_i^{i=1..K}, (\boldsymbol{SystemStartup}, (\boldsymbol{10^4}, CLK_{CLAG_i}))$, N/A, N/A,

$Cluster_i$: $ClusterPower$, $N/A\ \}$

Each cluster agent periodically informs the platform agent about the power consumption of the cluster.

$\{\ \boldsymbol{Inform}, \boldsymbol{CEAG}_i^{i=1..M*K}, (\boldsymbol{SystemStartup}, (\boldsymbol{10^2}, \boldsymbol{C}LK_{CEAG_i}))$, N/A, N/A,

$Cell_i$: $BufferLoad$, $N/A\ \}$

$\{\ \boldsymbol{Inform}, \boldsymbol{CEAG}_i^{i=1..M*K}, (\boldsymbol{SystemStartup}, (\boldsymbol{10^2}, \boldsymbol{CLK}_{\boldsymbol{1e2}, CEAG_i}))$, N/A, N/A,

$Cell_i$: $ErrorState$, $N/A\ \}$

Each cell agent periodically informs the buffer load and error state of the local cell to the cluster agent.

CONCLUSION

Hierarchical agent monitoring architecture is a promising approach for developing hierarchical online monitoring services for massively parallel SoCs. The adoption of formal methods to specify and validate such a system is an essential step towards the actual implementation with guaranteed performance.

System-level specification of the HAMSOC platform was presented. The specification models the system as being composed of hierarchical entities, categorized into components and monitoring agents at each level. The run-time status of each entity is formalized as a set, listing all parameters that are of concern to the designers. Monitoring operations are defined between agents and targeting entities, based on statically or dynamically specified conditions. Straightforward syntax based on the classic set theory is adopted, with extra notations to express concepts considered in practical SoC/VLSI development. The formal specification non-ambiguously captures the design intentions on the system-level abstractions, and can be conveniently applied in actual design process.

Starting from this novel yet initial work, we will define specific properties on HAMSOC systems, which are significantly different from, and usually more complicated than, those existing or considered in current SoC systems. For instance, the state space of the whole system is highly dynamic, as the result of parallel state transitions of components and agents triggered by constantly flowing monitoring operations. Allowable state transitions and reachable states are important issues in the verification of the system's security and dependability. Formal methods will be introduced to express and construct the state space, so that related property verification can be performed.

REFERENCES

Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., et al. (2006). *The Landscape of Parallel Computing Research: A View from Berkeley* (Tech. Rep. No.UCB/EECS-2006-183). Berkeley, CA: University of California, Berkeley.

Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., et al. (2008). TILE64 Processor: A 64-Core SoC with Mesh Interconnect. In *Proceedings of the Digest of Technical Papers of IEEE International Solid-State Circuits Conference*, San Francisco, CA (pp. 88-598). Washington, DC: IEEE.

Borkar, S., Karnik, T., Narendra, S., Tschanz, J., Keshavarzi, A., & De, V. (2003). Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the Design Automation Conference* (pp. 338-342).

Brooks, D., & Martonosi, M. (2001). Dynamic thermal management for high-performance microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture* (pp.171-182). New York: ACM.

Chen, C.-S. (2008). *On-chip current and power measurement techniques for integrated circuits with regulated power*. Doctoral dissertation, University of Rhode Island.

Chen, L. (1998). AgentOS: the agent-based distributed operating system for mobile networks. *Crossroads*, *5*(2), 12–14. doi:10.1145/333151.333156

Ciordas, C., Hansson, A., Goossens, K., & Basten, T. (2006). A Monitoring-Aware Network-on-Chip Design Flow. In *Proceedings of the 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools DSD,* Los Alamitosa, CA (pp. 97-106). Washington, DC: IEEE Computer Society.

Edwards, S., Lavagno, L., Lee, E. A., & Sangiovanni-Vincentelli, A. (1997). Design of embedded systems: formal models, validation, and synthesis. *Proceedings of the IEEE*, *85*(3), 366–390. doi:10.1109/5.558710

Guang, L., & Jantsch, A. (2006). Adaptive Power Management for the On-Chip Communication Network. In *Proceedings of the 9th EUROMICRO DSD Conference,* Los Alamitos, CA (pp. 649-656). Washington, DC: IEEE Computer Society.

Guang, L., Nigussie, E., Koskinen, L., & Tenhunen, H. (2009). Autonomous DVFS on Supply Islands for Energy-Constrained NoC Communication. In *Proceedings of ARCS 09* (LNCS 5545, pp. 183-194). Berlin: Springer Verlag.

Isci, C., Buyuktosunoglu, A., Cher, C.-Y., Bose, P., & Martonosi, M. (2006), An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proceedings Of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (pp.347-358). Washington, DC: IEEE Computer Society.

Karnik, T., De, V., & Borkar, S. (2004). Statistical design for variation tolerance: key to continued Moore's law. In *Proceedings of the International Conference on Integrated Circuit Design and Technology (ICICDT '04)* (pp. 175-176).

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36*(1), 41–50. doi:10.1109/MC.2003.1160055

Keutzer, K., Newton, A. R., Rabaey, J. M., & Sangiovanni-Vincentelli, A. (2000). System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on CAD*, *19*(12), 1523–1543.

Naji, H. R., Wells, B. E., & Aborizka, M. (2002). Hardware agents. *In proceedings of the 11th International Conference on Intelligent Systems on Emerging Technologies*, Boston, MA.

Naji, H. R., Wells, B. E., & Etzkorn, L. (2004). Creating an adaptive embedded system by applying multi-agent techniques to reconfigurable hardware. *Future Generation Computer Systems*, *20*(6), 1055–1081. doi:10.1016/j.future.2004.02.002

Parunak, H. V. D. (1998). *Practical and industrial applications of agent-based systems (Tech. Rep.)*. Industrial Technology Institute.

Rabaey, J. M. (2007). *Scaling the Power Wall: Revisiting the Low-Power Design Rules*. Paper presented at the Keynote speech at SoC 07 Symposium, Tampere, Finland.

Rouff, C. A., Rash, J. L., Hinchey, M. G., & Truszkowski, W. F. (2006). Formal Methods at NASA Goddard Space Flight Center. In Rouff, C. A., Hinchey, M. G., Rash, J. L., Truszkowski, W. F., & Gordon-Spears, D. (Eds.), *Agent Technology from a Formal Perspective* (pp. 287–309). London: Springer. doi:10.1007/1-84628-271-3_10

Sartori, J., & Kumar, R. (2009). Distributed peak power management for many-core architectures. In *Proceedings Of the Design* (pp. 1556–1559). Automation & Test in Europe.

Shang, L., Peh, L.-S., & Jha, N. K. (2003), Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proceedings of the 9th international Symposium on High-Performance Computer Architecture*. Washington, DC: IEEE Computer Society.

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, *60*(1), 51–92. doi:10.1016/0004-3702(93)90034-9

Soteriou, V., & Peh, L.-S. (2007). Exploring the Design Space of Self-Regulating Power-Aware On/Off Interconnection Networks. *IEEE Transactions on Parallel and Distributed Systems*, *18*(3), 393–408. doi:10.1109/TPDS.2007.43

Sylvester, D., Blaauw, D., & Karl, E. (2006). ElastIC: An Adaptive Self-Healing Architecture for Unpredictable Silicon. *IEEE Design & Test of Computers*, *23*(6), 484–490. doi:10.1109/MDT.2006.145

Vangal, S. R., Howard, J., Ruhl, G., Dighe, S., Wilson, H., & Tschanz, J. (2008). An 80-Tile Sub-100-W TeraFLOPS Processor in 65nm CMOS. *IEEE Journal of Solid-state Circuits*, *43*(1), 29–41. doi:10.1109/JSSC.2007.910957

Wang, D., & Cao, J. (2003). On optimal hierarchical configuration of distributed systems on mesh and hypercube. In *Proceedings of the International Parallel and Distributed Processing Symposium*. Washington, DC: IEEE Computer Society.

Yin, A. W., Guang, L., Liljeberg, P., Rantala, P., Nigussie, E., Isoaho, J., et al. (2008). Hierarchical Agent Architecture for Scalable NoC Design with Online Monitoring Services. In *Proceedins of the First International Workshop on Network on Chip Architectures* (pp. 58-63).

Yin, A. W., Guang, L., Nigussie, E., Liljeberg, P., & Tenhunen, H. (2009). Architectural Exploration of Per-Core DVFS for Energy-Constrained On-Chip Networks. In *Proceedings of the Euromicro DSD (Digital System Design) conference 2009,* Los Alamitos, CA (pp. 141-146). Washington, DC: IEEE Computer Society.

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 2, edited by Seppo Virtanen, pp. 86-105, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Chapter 16
Service-Oriented Development of Fault Tolerant Communicating Systems: Refinement Approach

Linas Laibinis
Åbo Akademi University, Finland

Elena Troubitsyna
Åbo Akademi University, Finland

Sari Leppänen
Nokia Research Center, Finland

ABSTRACT

Telecommunication systems must have a high degree of availability, that is, a high probability of correct and timely provision of requested services. To achieve this, correctness of software for such systems should be ensured. Application of formal methods helps increase confidence in building correct software. However, to be used in practice, formal methods should be well integrated into existing development process. In this paper, the authors propose a formal model-driven approach to development of communicating systems. The authors formalize and extend the Lyra approach—a top-down service-oriented method for development of communicating systems. Lyra is based on transformation and decomposition of models expressed in UML2. The authors formalize Lyra in the B Method by proposing a set of formal specification and refinement patterns reflecting the essential models and transformations of the Lyra phases. Moreover, this paper extends Lyra to integrate reasoning about fault tolerance in the entire development flow.

DOI: 10.4018/978-1-4666-0912-9.ch016

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

1 INTRODUCTION

Modern telecommunication systems are usually distributed software-intensive systems providing a large variety of services to their users. Development of software for such systems is inherently complex and error prone. However, software failures might lead to unavailability or incorrect provision of system services, which in turn could incur significant financial losses. Hence it is important to guarantee correctness of software for telecommunication systems.

Formal methods have been traditionally used for reasoning about software correctness. Nevertheless, they are yet insufficiently well integrated into current development practice. Unlike formal methods, Unified Modeling Language (UML) developed by Rumbaugh, Jakobson, and Booch (1998) has a lower degree of rigor for reasoning about software correctness but is widely accepted in industry. UML is a general purpose modelling language and, to be used effectively, should be tailored to a specific application domain.

Nokia Research Center has developed the design method Lyra described by Leppânen, Turunen, and Oliver (2004) – a UML2-based service-oriented method specific to the domain of communicating systems and communication protocols. The design flow of Lyra is based on the concepts of decomposition and preservation of the externally observable behaviour. The system behaviour is modularised and organized into hierarchical layers according to the external communication and related interfaces. It allows the designers to derive the distributed network architecture from the functional system requirements via a number of model transformations.

From the beginning Lyra has been developed in such a way that it would be possible to bring formal methods (such as program refinement, model checking, model-based testing etc.) into more extensive industrial use. A formalization of the Lyra development would allow us to ensure correctness of system design via automatic and formally verified construction. The achievement of such a formalization would be considered as significant added value for industry.

In this paper we propose a set of formal specification and refinement patterns reflecting the essential models and transformations of Lyra. Our approach is based on stepwise refinement of a formal system model in the B Method proposed by Abrial (1996) – a formal framework with automatic tool support. While developing a system by refinement, we start from an abstract specification and gradually incorporate implementation details into it until executable code is obtained. While formalizing Lyra, we single out a generic concept of a communicating service component and propose patterns for specifying and refining it. In the refinement process the service component is decomposed into a set of service components of smaller granularity specified according to the proposed pattern. Moreover, we demonstrate that the process of distributing service components between different network elements can also be captured by the notion of refinement.

To achieve system fault tolerance, we extend Lyra to integrate modelling of fault tolerance mechanisms into the entire development flow. We demonstrate how to formally specify error recovery by rollbacks as well as reason about error recovery termination.

The proposed formal specification and development patterns establish a background for automatic generation of formal specifications from UML2 models and expressing model transformations as refinement steps. Via automation of the UML2-based Lyra design flow we aim at smooth incorporation of formal methods into existing development practice.

2 LYRA: SERVICE-BASED DEVELOPMENT OF COMMUNICATING SYSTEMS

Overview of Lyra

Lyra described by Leppänen, Turunen, and Oliver (2004) is a model-driven and component-based design method for the development of communicating systems and communication protocols. It has been developed in the Nokia Research Center by integrating the best practices and design patterns established in the area of communicating systems. The method covers all industrial specification and design phases from prestandardisation to final implementation. It has been successfully applied in large-scale UML2-based industrial software development, e.g., for specification of architecture for several network components, standardisation of 3GPP protocols, implementation of several network protocols etc.

Lyra has four main phases: *Service Specification*, *Service Decomposition*, *Service Distribution* and *Service Implementation*. The *Service Specification* phase focuses on defining services provided by the system and their users. The goal of this phase is to define the externally observable behaviour of the system level services via deriving logical user interfaces. In the *Service Decomposition* phase the abstract model produced at the previous stage is decomposed in a stepwise and top-down fashion into a set of service components and logical interfaces between them. The result of this phase is the logical architecture of the service implementations. In the *Service Distribution* phase, the logical architecture of services is distributed over a given platform architecture. Finally, in the *Service Implementation* phase, the structural elements are adjusted and integrated into the target environment, low-level implementation details are added and platform-specific code is generated. Next we discuss Lyra in more detail with an example.

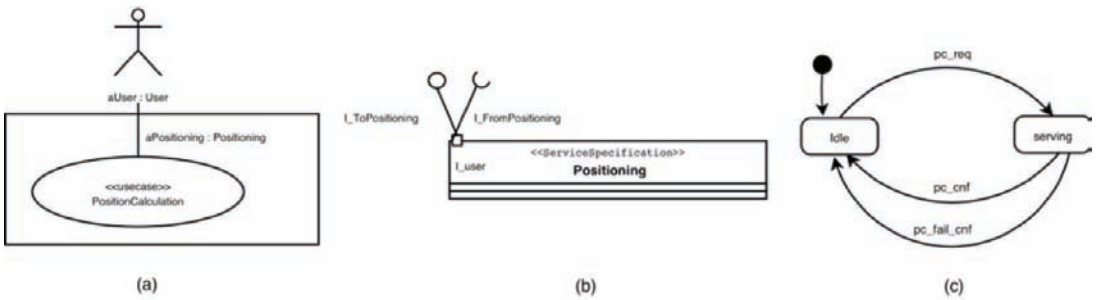
Lyra by Example

We model a positioning system which provides positioning services to calculate the physical location of a given item of user equipment (UE) in a mobile network as described in 3GPP Technical specification 25.305 and 3GPP Technical specification 25.453. We consider Position Calculation Application Part (PCAP), which manages communication between two standard network elements. We assume that the PCAP functional requirements are correctly defined and, hence, focus on the architectural decomposition and distribution decisions.

The *Service Specification* phase starts from creating the domain model of the system. It describes the system with included system level services and different types of external users. Each association connecting an external user and a system level service corresponds to a logical interface. For the system and the system level services we define active classes, while for each type of an external user we define the corresponding external class. The relationships between the system level services and their users become candidates for *PSAPs – Provided Service Access Points* of the system level services. The logical interfaces are attached to the classes with ports. The domain model for the Positioning system and its service *PositionCalculation* is shown in Figure 1(a) and PSAP of the Positioning system – *I_User PSAP* is shown in Figure 1(b). The UML2 interfaces *I_ToPositioning* and *I_FromPositioning* define the signals and signal parameters of I user PSAP. We formally describe the communication between a system level service and its user(s) in the *PSAPCommunication* state machine as illustrated in Figure 1(c). The positioning request *pc_req* received from the user is always replied: with the signal *pc_cnf* in case of success, and with the signal *pc_fail_cnf* otherwise.

To implement its own services, the system usually uses external entities. For instance, to provide the *PositionCalculation* service, the po-

Figure 1. (a) Domain model (b) PSAP of positioning (c) state diagram



sitioning system should first request Radio Network Database (*DB*) for an approximate position of User Equipment (*UE*). The information obtained from *DB* is used to contact *UE* and request it to emit a radio signal. At the same time, one or more Reference Location Measurement Unit devices (*ReferenceLMU*) are contacted to provide additional measurements of radio signals. The radio measurements obtained from *UE* and *ReferenceLMU* are used to calculate the exact position of *UE*. The calculation is done by the Algorithm service provider (*Algorithm*), which produces the final estimation of the *UE* location. Let us observe that the services provided by the external entities partition execution of the *PositionCalculation* service into the corresponding stages. In the next phase of the Lyra development – *Service Decomposition* – we focus on specifying the service execution according to the identified stages.

In the *Service Decomposition* phase, we introduce the providers of external services into the domain model constructed previously, as shown in Figure 2. The model includes the external service providers *DB, UE, ReferenceLMU* and *Algorithm*, which are then defined as the external classes. For each association between a system level service and the corresponding external class we define a logical interface. The logical interfaces are attached to the corresponding classes via ports called *USAPs – Used Service Access Points*, as presented in Figure 3.

Let us observe that the system behaviour is modularised according to the related service access points – PSAPs and USAPs. Moreover, the functional architecture is defined in terms of service components, which encapsulate the functionality related to a single execution stage or another logical piece of functionality.

In Figure 4 we present the architecture diagram of the Positioning system. Here *ServiceDirector* plays two roles: it controls the service execution flow and handles the communication on the PSAP. The behaviour of *ServiceDirector* is presented in Figure 5. The top-most state machine specifies the communication on PSAP, while the state subma-

Figure 2. Domain model

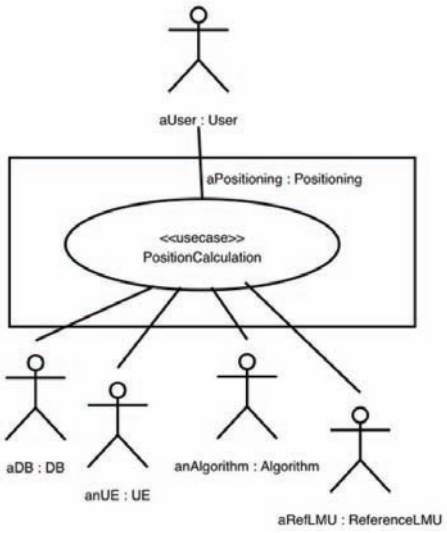
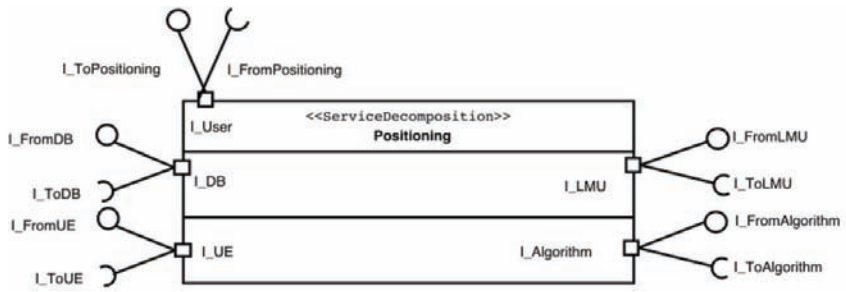


Figure 3.PSAP and USAPs of positioning



chine *Serving* specifies a valid execution flow of the position calculation. The sub-states of *Serving* encapsulate the stage-specific behaviour and can be represented as the corresponding submachines. In their turns, these machines (omitted here) include the specifications of specific PSAP-USAP communications.

The modular system model produced at the *Service Decomposition* phase allows us to analyse various distribution models. In the next phase – *Service Distribution* – the service components are distributed over a given network architecture. The signaling network protocols are used for communication between the service components allocated on distant network elements.

In Figure 6 we illustrate the physical structure of the distributed positioning system. Here *Positioning_RNC* and *Positioning_SAS* represent the predefined network elements called *RNC* and *SAS* correspondingly. The standard interface *Iupc* is used in the communication between them. We map the functional architecture obtained at the previous stage to the given network architecture by distributing the service components between the network elements. The functional architecture of the *Positioning_SAS* network element is illustrated

Figure 4. PositionCalculation functional architecture

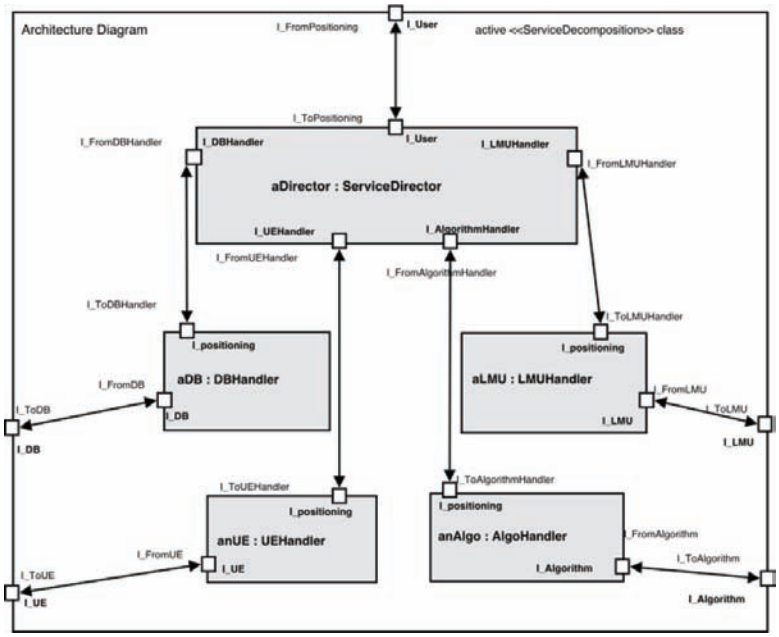


Figure 5. Service director: PSAP communication and execution control

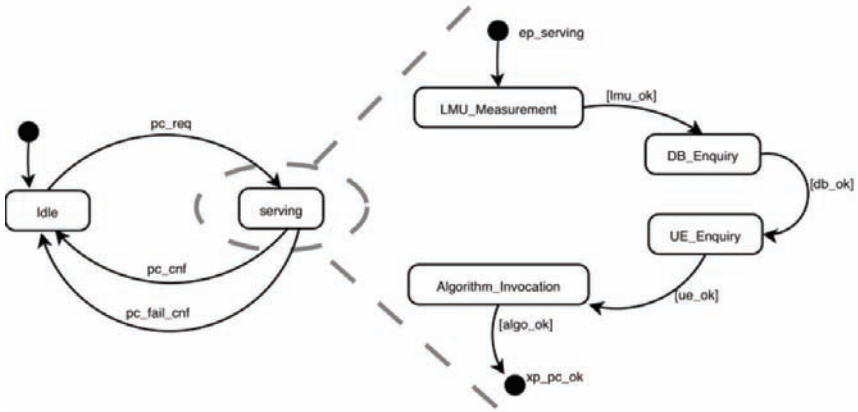
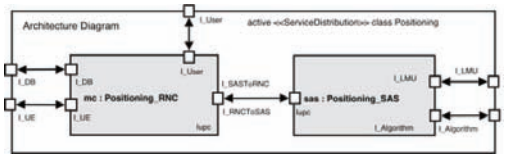


Figure 6. Service architecture



in Figure 7. The functionality of *ServiceDirector* specified at the *Service Decomposition* phase is now decomposed and distributed over the given network. *ServiceDirector_SAS* handles the interface towards the *RNC* network element and controls the execution flow of the positioning calculation process in the *SAS* network element.

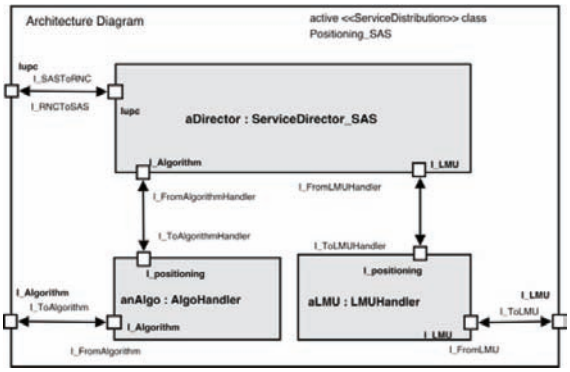
Finally, at the *Service Implementation* phase we specify how the virtual communication between entities in different network nodes is realized using the underlying transport services. We also implement data encoding and decoding, routing of messages and dynamic process management. This stage is described in detail by Leppânen, Turunen, and Oliver (2004) as well as in 3GPP Technical specification 25.453 and 3GPP Technical specification 25.305.

In the next section we give a brief introduction into our formal framework, the B Method, which we will use to formalize the development flow described above.

3 DEVELOPING SYSTEMS BY REFINEMENT IN THE B METHOD

The B Method proposed by Abrial (1996) is an approach for the industrial development of highly dependable software. The method has been successfully used in the development of several complex real-life applications as described in Handbook for Correct System Construction. Recently the B method has been extended by the Event B framework proposed by Abrial (1996). The framework enables modelling of event-based systems. Event B is particularly suitable for developing distributed, parallel and reactive systems. In fact, this extension has incorporated the action system formalism proposed by Back

Figure 7. Architecture of Positioning_SAS



and Sere (1990) in the B Method. In the rest of the paper, we refer to the B Method together with its extension Event B simply as B.

The tool support available for B provides us with the assistance for the entire development process. For instance, Atelier B as described in AtelierB: User and Reference Manuals, one of the tools supporting the B Method, has facilities for automatic verification and code generation as well as documentation, project management and prototyping. It has a plug-in for integrating modelling in Event B. Atelier B provides us with a high degree of automation in verifying correctness that improves scalability of B, speeds up development and, also, requires less mathematical training from the users.

Modelling in B

The B Method adopts the top-down approach to system development. The development starts from creating a formal system specification. A formal specification is a mathematical model of the required behaviour of a system, or a part of a system. In B, a specification is represented by a collection of modules, called Abstract Machines. The *Abstract Machine Notation* (AMN), is used in constructing and verifying them. An abstract machine encapsulates a local state (local variables) of the machine and provides operations on the state. A simple abstract machine has the following general form:

```
MACHINE AM
SETS TYPES
VARIABLES v
INVARIANT I
INITIALISATION INIT
EVENTS
    E1 = …
    …
    EN = …
END
```

The machine is uniquely identified by its name *AM*. The state variables of the machine, *v*, are declared in the VARIABLES clause and initialised in *INIT* as defined in the INITIALISATION clause. The variables in B are strongly typed by constraining predicates of the machine invariant *I* given in the INVARIANT clause. The invariant is usually defined as a conjunction of the constraining predicates and the predicates defining the properties of the system that should be preserved during system execution. All types in B are represented by non-empty sets. Local types can be introduced by enumerating the elements of the type, e.g., *TYPE = {element1, element2,...}*, or by defining them as subsets of already existing types or sets.

The operations $E_1,..., E_N$ of the machine are defined in the EVENTS clause. The operations are atomic meaning that, once an operation is chosen, its execution will run until completion without interference. There are two standard ways to describe an operation in B: either by the preconditioned operation PRE *cond* THEN *body* END or the guarded operation SELECT *cond* THEN *body* END. Here *cond* is a state predicate, and *body* is a B statement. If *cond* is satisfied, the behaviour of both the precondition operation and the guarded operation corresponds to the execution of their bodies. However, these operations behave differently when an attempt to execute them from a state where *cond* is false is undertaken. In this case the precondition operation leads to a crash (i.e., unpredictable or even non-terminating behaviour) of the system, while the execution of the guarded operation is blocked. The preconditioned operations are used to describe the operations that will be turned (implemented) into procedures called by the user. On the other hand, the guarded operations are useful when we have to specify system behaviour in terms of its reactions on the occurrence of certain events. The operations of event-based systems are often called events.

The B statements that we will use to describe the bodies of events have the following syntax:

S == *x:= e* |**IF***cond***THEN***S1***ELSE***S2***END**| *S1 ; S2* | *x:: T* |

ANY*z***WHERE***Q***THEN** S **END** | *S1* || *S2* |...

The first three constructs – an assignment, a conditional statement and a sequential composition have the standard meaning. A sequential composition is disallowed in abstract specifications but permitted in refinements. The remaining constructs allow us to model nondeterministic or parallel behaviour in a specification. Usually they are not implementable so they have to be refined (replaced) with executable constructs at some point of program development. We use two kinds of nondeterministic statements – the nondeterministic assignment and the nondeterministic block. The nondeterministic assignment *x:: T* assigns the variable *x* an arbitrary value from the given set (type) *T*. The nondeterministic block **ANY** *z* WHERE *Q* THEN *S* END introduces the new local variable *z* which is initialised (possibly nondeterministically) according to the predicate *Q* and then used in the statement *S*. Finally, *S1* || *S2* models parallel (simultaneous) execution of *S1* and *S2* provided *S1* and *S2* do not have a conflict on state variables. The special case of the parallel execution is a multiple assignment, which is denoted as *x, y:= e1,e2*.

The B statements are formally defined using the weakest precondition semantics defined by Dijkstra (1976). Intuitively, for a given statement *S* and a post-condition *P*, the weakest precondition *wp(S,P)* describes the set of all such initial states from which execution of *S* is guaranteed to establish *P*. The weakest precondition semantics is a foundation for establishing correctness of specifications and verifying refinements between them. To show correctness (consistency) of an event-based system, we should demonstrate that its invariant is true in the initial state (i.e., after the initialisation is executed) and that every event preserves the invariant:

wp(INIT,I) = true, and

$g_i \wedge I \Rightarrow wp(E_i, I)$

Refinement of Event-Based Systems

The basic idea underlying stepwise development in B is to design the system implementation gradually, by a number of correctness preserving steps called refinements. The refinement process starts from creating an abstract specification and finishes with generating executable code. The intermediate stages yield the specifications containing a mixture of abstract mathematical constructs and executable programming artifacts. In general, refinement process can be seen as a way to reduce non-determinism of the abstract specification and replace abstract mathematical data structures by data structures implementable on a computer. Hence refinement allows us to introduce implementation decisions gradually.

Formally, we say that the statement *S* is refined by the statement *S'*, written $S \subseteq S'$, if, whenever *S* establishes a certain post-condition, so does *S'*:

$S \subseteq S'$ if and only if for all post-conditions *p*: $wp(S, p) \Rightarrow wp(S',p)$

In the AMN, the results of intermediate development stages – refinement machines – have essentially the same structure as the more abstract specifications. In addition, a refinement machine explicitly states which specification it refines.

Assume that the refinement machine *AM'* given below is a result of refinement of the abstract machine *AM*:

```
REFINEMENT AM'
REFINES AM
VARIABLES v'
INVARIANT I'
```

```
INITIALISATION INIT'
EVENTS
    E1 = …
    …
    EN = …
END
```

In *AM'* we replace the abstract data structures of *AM* with concrete ones. The invariant of *AM'* , *I'*, defines now not only the invariant properties of the refined specification but also the connection between the newly introduced variables and the abstract variables that they replace. For a refinement step to be valid, every possible execution of the refined machine must correspond (via *I'*) to some execution of the abstract machine. To demonstrate this, we should prove that *INIT'* is a valid refinement of *INIT*, each event of *AM'* is a valid refinement of its counterpart in *AM* and that the refined specification does not introduce additional deadlocks, i.e.,

wp(INIT' , ®wp(INIT, ®I')) = true,

$I \wedge I' \wedge g_i' \Rightarrow g_i \wedge wp(S', ®wp(S, ®I'))$, and

$I \wedge \cup g_j \Rightarrow g_i'$

Often refinement process introduces new variables and the corresponding computations (new events) on them, while leaving the previous variables and computations essentially unchanged. Such refinement is referred to as *superposition* refinement defined by Back and Sere (1996). Let us consider the abstract machine AM_S and the refinement machine AM_SR:

```
MACHINE AM_S
VARIABLES a
INVARIANT I
INITIALISATION INIT
EVENTS
      E = SELECT g
              THEN S END
END
REFINEMENT AM_SR
REFINES AM_S
VARIABLES a,b
INVARIANT I'
VARIANT V
INITIALISATION INIT'
EVENTS
      E = SELECT g
              THEN S END
     E1 = SELECT g1
              THEN S1 END
     E2 = SELECT g2
              THEN S2 END
END
```

Observe that the refinement machine contains the new events E_1 and E_2 as well as the new clause VARIANT. The new events define computations on the newly introduced variables *b* and, hence, can be seen as the events refining the statement *skip* on the abstract variables. Every new event should decrease the value of the variant. This allows us to guarantee that new events cannot take the control forever, since the variant expression cannot be decreased infinitely. For each newly introduced event, we should demonstrate that the variant expression is a natural number and execution of the event decreases the variant, i.e.,

$V \in NAT$, and

$I' \wedge g_i \Rightarrow wp((n := V ; S_i), n<V)$

In B, there are also mechanisms for structuring the system architecture by modularisation. The abstract machines can be composed by means of several mechanisms providing different forms of encapsulation. For instance, if the machine *C* INCLUDES the machine *D* then all variables and operations of *D* are incorporated in *C*. However, to guarantee internal consistency (and hence independent verification and reuse) of *D*, the

machine *C* can change the variables of *D* only via the operations of *D*.

Next we illustrate modelling and refinement in B by presenting a formal development of fault-tolerant communicating systems according to the Lyra methodology.

4 TOWARDS FORMALIZING AND EXTENDING LYRA

Modelling a Service Component in B

In Section 2 we have defined a service component as a coherent piece of functionality that provides its services to a service consumer via PSAP(s). We used this term to refer to the providers of external services introduced at the *Service Decomposition* phase. However, the notion of a service component can be generalized to represent the service providers at different levels of abstraction. Indeed, even the entire Positioning system can be seen as a service component providing the *Position Calculation* service. On the other hand, peer proxies introduced at the lowest level of abstraction can also be seen as the service components providing the physical data transfer services. Therefore, the notion of a service component is central to the entire Lyra development process.

A service component has two essential parts: functional and communicational. The functional part is a ”mission” of a service component, i.e., the service(s) that it is capable of providing. The communicational part is an interface via which a service component receives requests to execute the service(s) and sends the results of service execution.

Execution of a service usually involves certain computations. We call the B representation of this part of a service component *Abstract CAlculating Machine* (*ACAM*). The communicational part is correspondingly called *Abstract Communicating Machine* (*ACM*), while the entire B model of a service component is called *Abstract Communicating Component* (*ACC*). The abstract machine *ACC* (Figure 8) presents the proposed pattern for specifying a service component in B.

While specifying a service component, we adopt a systemic approach, i.e., model the service component together with the relevant part of its environment, the service consumer. Namely, when modelling the communicational (*ACM*) part of *ACC*, we also specify how the service consumer places requests to execute a service in the operation *input* and reads the results of service execution in the operation *output*. The input parameters *param* and *time* of the operation input model the parameters of a request and the maximal time allowed for executing the service. For instance, in the Positioning System example described in Section 2, an arrival of the position calculation request – the signal *pc_req* – can be represented as an instantiation of the operation *input*. Moreover, the request might have parameters – the precision of position calculation defined by the service consumer and the maximal execution time defined by the system, e.g., according to the current network load. The parameters of the request are stored in the internal data buffer *in_data*, so they can be used by *ACAM* while performing the required computations.

In our initial specification we abstract away from the details of computations required to execute a service, i.e., *ACAM* is modelled as a statement non-deterministically generating the results of service execution. These results are stored in the internal output buffer *out_data*. The service consumer obtains the results of service provision as the output parameter *res* of the operation *output*. Already in the abstract specification we model possibility of failure – *out_data* might contain values representing the results of not only successful service executions but also failed ones. In our example, in case of successful execution, the signal *pc_cnf* together with the calculated position are sent to the service consumer. Otherwise, the signal *pc_fail_cnf* is generated.

Figure 8. The abstract machine ACC

```
MACHINE ACC
SETS DATA
CONSTANTS NIL, Abort_data
PROPERTIES
  NIL ∈ DATA ∧ Abort_data ∈ DATA ∧ ¬ (Abort_data = NIL)
VARIABLES in_data, out_data
INVARIANT
  in_data ∈ DATA ∧ out_data ∈ DATA
INITIALISATION
  in_data, out_data := NIL, NIL
EVENTS
input(param,time) =
  PRE param ∈ DATA ∧ time ∈ NAT1 ∧ ¬ (param=NIL) ∧ in_data=NIL
  THEN
    in_data := param
  END;
calculate =
  SELECT ¬ (in_data=NIL) ∧ out_data = NIL
  THEN
   out_data :∈ DATA - {NIL}
  END;
res ← output =
  PRE ¬ (out_data = NIL)
  THEN
    res := out_data ||
    in_data,out_data := NIL, NIL
   END
END
```

While executing the operation *output*, the input and output buffers are emptied and the service component becomes ready to accept a new service request. Here we reserve the abstract constant *NIL* to model the absence of data.

In Lyra, a service component is usually represented as an active class with the PSAP(s) attached to it via the port(s). The state diagram depicts the signalling scenario on PSAP including the signals from and to the external class modelling the service consumer. Essentially these diagrams suffice to specify a service component according to the *ACC* pattern. Namely, the UML2 description of PSAP is translated into the communicational (*ACM*) part of the machine *ACC*. The functional (*ACAM*) part of *ACC* should be instantiated by the data types specific to the modelled service component. This translation formalizes the *Service Specification* phase of Lyra.

Let us observe that the machine *ACC* can be seen as a specification pattern, which can be instantiated by supplying the details specific to a service component under construction. For instance, the *ACM* part of *ACC* models data transfer to and from a service component very abstractly. We have shown how it can be instantiated for the Positioning system example. While developing a more complex service component, this part can be instantiated with more elaborated data

structures and the corresponding protocols for transferring them.

Next we discuss how to extend Lyra with the explicit representation of the fault tolerance mechanisms and then show the use of the *ACC* pattern in the entire Lyra development process.

Introducing Fault Tolerance in the Lyra Development Flow

Currently the Lyra methodology addresses fault tolerance implicitly, i.e., by representating not only successful but also failed service provision in the Lyra UML models. However, it leaves aside modelling of mechanisms for detecting and recovering from errors – the fault tolerance mechanisms. We argue that, by integrating explicit representation of the means for fault tolerance into the entire development process, we establish a basis for constructing systems that are better resistant to errors, i.e., achieve better system dependability. Next we will discuss how to extend Lyra to integrate modelling of fault tolerance.

In the first development stage of Lyra we set a scene for reasoning about fault tolerance by modelling not only successful service provision but also service failure. In the next development stage – *Service Decomposition* – we elaborate on representation of the causes of service failures and the means for fault tolerance.

In the *Service Decomposition* phase we decompose the service provided by a service component into a number of stages (subservices). The service component can execute certain subservices itself as well as request other service components to do it. According to Lyra, the flow of the service execution is managed by a special service component called *Service Director*. It implements the behaviour of PSAP of a service component as specified earlier. Moreover, it co-ordinates the execution flow by enquiring the required subservices from the external service components.

In general, execution of any stage of a service can fail. In its turn, this might lead to failure of the entire service provision. Therefore, while specifying *Service Director*, we should ensure that it does not only orchestrates the fault-free execution flow but also handles erroneous situations. Indeed, as a result of requesting a particular subservice, *Service Director* can obtain a normal response containing the requested data or a notification about an error. As a reaction to the occurred error, *Service Director* might

- Retry the execution of the failed subservice,
- Repeat the execution of several previous subservices (i.e., roll back in the service execution flow) and then retry the failed subservice,
- Abort the execution of the entire service.

The reaction of *Service Director* depends on the criticality of an occurred error: the more critical is the error, the larger part of the execution flow (Figure 9) has to be involved in the error recovery. Moreover, the most critical (i.e., unrecoverable) errors lead to aborting the entire service. In Figure 6(a) we illustrate a fault free execution of the service S composed of subservices $S_1,...,S_N$. Different error recovery mechanisms used in the presence of errors are shown in Figure 10, 11, and 12.

Let us observe that each service should be provided within a certain finite period of time –

Figure 9. Service execution flow: Fault-free

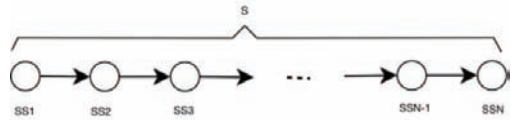


Figure 10. Error recovery by retrying

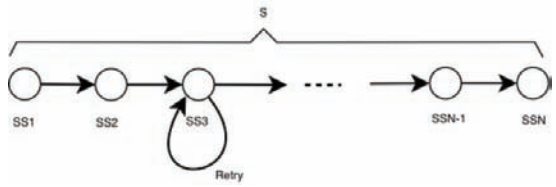


Figure 11. Error recovery by rollbacks

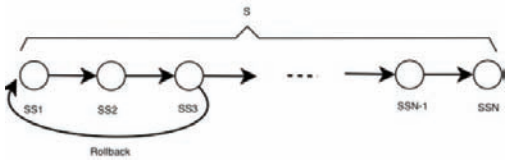
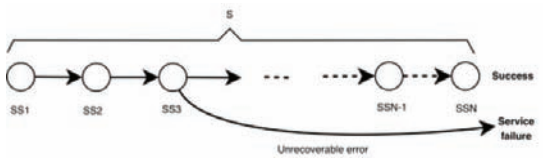


Figure 12. Aborting service execution



the *maximal service response time Max_SRT*. In our model this time is passed as a parameter of the service request. Since each attempt of subservice execution takes some time, the service execution might be aborted even if only recoverable errors have occurred but the overall service execution time has already exceeded *Max_SRT*. Therefore, by introducing *Max_SRT* in our model, we also guarantee termination of error recovery, i.e., disallow infinite retries and rollbacks, as shown in Figure 13.

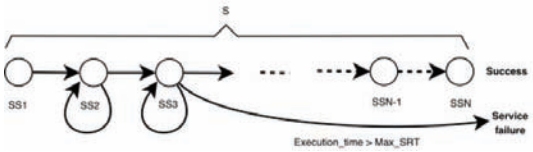
Next we demonstrate how to represent the extended Lyra development as refinement in B.

5 SERVICE-ORIENTED DEVELOPMENT BY REFINEMENT IN B

Formalizing Service Decomposition

In the first stage of our formalized development we used the UML2 models produced at the *Service Specification* phase to specify a service component according to the *ACC* pattern. The next step focuses on modelling the service execution flow with the incorporated fault tolerance mechanisms. Namely, we introduce a representation of *Service Director* into the abstract specification of a service

Figure 13. Aborting due to timeout

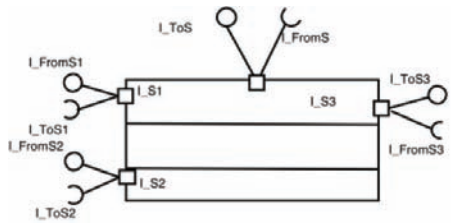


component. This is done by refining the machine *ACC* to capture the design decisions made at the *Service Decomposition* and *Service Distribution* phases. Hence, to derive the specification of *Service Director*, we use UML2 diagrams modelling both the functional and the platform-distributed architectures. In general, we should consider two cases:

1. *Service Director* is ”centralized”, i.e., it resides on a single network element,
2. *Service Director* is “distributed”, i.e., different parts of the execution flow are orchestrated by distinct service directors residing on different network elements.

Assume for simplicity that the set of subservices required in the execution of the service *S* consists of three elements: S_1, S_2 and S_3. At the *Service Decomposition* phase, in both cases the model of the service component providing the service *S* looks as shown in Figure 14. The service distribution architecture diagram for the first case is given in Figure 15. In the second case, let us assume that the execution flow of the service component is orchestrated by two service directors: the *Service Director1*, which handles the communication on PSAP and communicates with the service component providing S_1, and *Service Director2*, which orchestrates the execution of the subservices S_2 and S_3. The service directors communicate with each other while passing the control over the corresponding parts of the execution flow. The architecture diagram depicting the overall arrangement for the second case is shown in Figure 20.

Figure 14. Service component with USAPs



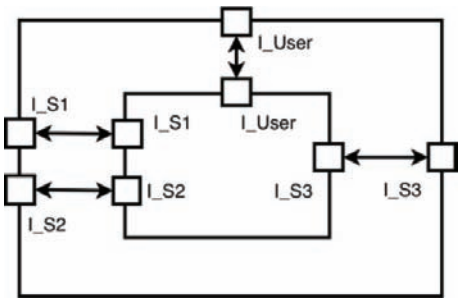
We model the decomposed service as a sequence over the abstract set *TASKS*. Each element of *TASKS* represents the individual subservice. Moreover, we introduce the abstract function *Next* to models the service execution flow. In case of the centralized *Service Director*, the subservices are executed one after another, i.e., the abstract representation of *Next* will be instantiated as follows:

$$\mathrm{Next}(S_i) = S_{i+1}$$

for *i: 1..max_sv*, where max_sv is the maximal number of subservices required to execute the service.

In the second case, the function *Next* describes the execution flow from the point of view of the main service director, i.e., it treats the groups of services managed by other service directors as atomic steps in the execution flow. In our example, the service S_1 is managed by *Service Director1*, while S_2 and S_3 are managed by *Service Director2*. In this case the function *Next* treats the execution of S_2 and S_3 as one execution step the performance of which is delegated to *Service Director2*. Hence, in this example *Next* will be instantiated as follows:

Figure 15. Architecture diagram (case 1)



$Next(S_1) = S_2$, and $Next(S_2) = S_4$

The result of refinement of the machine *ACC* – the machine *ACC_DEC* – is given below (Figure 16, 17, and 18).

The currently executed subservice is modelled by the variable *curr_task*. The results of the current subservice execution are stored in the variable *curr_data*. The results of all subservices already executed are accumulated in the variable *results*. The variable fi*nished* indicates the end of service execution. The variable is set to TRUE when the whole sequence of subservices has been executed or some unrecoverable error has occurred.

To model progress of time, we introduce the variable *time_left*. When a service request is received in the operation *input*, *time_left* is set to the maximal service response time *Max_SRT*, which is received as the second parameter of *input*. The variable old time left is used to force interleaving between progress of the execution flow and the passage of time. The operation timer decreases the value of *time_left*, disables itself and enables the operation *handle*, which specifies the service co-ordinating behaviour of *Service Director*.

Figure 16. The machine ACC_DEC

```
REFINEMENT ACC_DEC
REFINES ACC
SETS
  DATA; TASK; RESPONSE = {OK, REPEAT, ROLLBACK, ABORT}
```

Figure 17. The machine ACC_DEC

```
CONSTANTS Service, Eval, Next, max_sv
PROPERTIES
  Service ∈ seq1(TASK) ∧ size(Service) = max_sv ∧
  Eval ∈ TASK × DATA → RESPONSE ∧
   ∀ dd. (dd ∈ DATA ⇒ ¬ (Eval(Service(1),dd) = ROLLBACK)) ∧
  Next ∈ 1 .. max_sv ⤖ 2 .. max_sv+1 ∧
  1 ∈ dom(Next) ∧ (max_sv+1) ∈ ran(Next) ∧
 ∀ii. (ii ∈ dom(Next) ∧ ¬ (Next(ii)=max_sv+1) ⇒ Next(ii) ∈ dom(Next))∧
 ∀ii. (ii ∈ dom(Next) ⇒ ii<Next(ii)) ∧ ...
VARIABLES
  in_data, out_data, time_left, old_time_left,
  curr_task, resp, finished, results, curr_data
INVARIANT
  resp ∈ RESPONSE ∧
  results ∈ 1 .. max_sv ⇸ DATA-{NIL} ∧
  curr_data ∈ DATA ∧
  curr_task ∈ 1 .. max_sv+1 ∧
  (finished = FALSE ⇒ time_left>0) ∧
  time_left ≤ old_time_left ∧
  dom(results) ⊆ dom(Next) ∧
  (finished = TRUE ⇒ (resp=ABORT) ∨ (curr_task = max_sv+1)) ∧
  (finished = FALSE ⇒ curr_task ∈ 1 .. max_sv) ∧
  (finished = FALSE ⇒ curr_task ∈ dom(Next)) ∧
  (curr_task = max_sv+1 ⇒ ¬ (resp=ABORT)) ∧
  (finished = TRUE ∧ curr_task = max_sv+1 ⇒
  Next ⁻¹ (curr_task) ∈ dom(results)) ∧ ...
INITIALISATION
  in_data, out_data := NIL, NIL ||
  time_left, old_time_left := max_time, max_time ||
  curr_task, resp := 1, OK ||
  finished, results := FALSE, ∅ ||
  curr_data := NIL
EVENTS
input(param,time) =
  PRE param ∈ DATA ∧ time ∈ NAT1 ∧ ¬ (param=NIL) ∧ in_data=NIL
  THEN
    in_data, time_left, old_time_left := param, time, time
  END;
```

In the operation *handle*, we model not only requesting a certain subservice and obtaining its response, but also handling notifications about errors. We introduce the abstract function *Eval*, which evaluates the obtained response from a requested subservice. The result of evaluation is assigned to the variable *resp*. If the subservice was successfully executed then the variable *resp* gets the value *OK*. In this case the next element from the sequence of subservices is chosen for execution according to the function *Next*. If a benign failure has occurred and error recovery merely requires to retry the execution of the failed subservice then the variable *resp* is assigned the value *REPEAT*. This

Figure 18. The machine ACC_DEC

```
handle =
  SELECT ¬ (in_data=NIL) ∧ finished = FALSE ∧ (time_left < old_time_left)
  THEN
    old_time_left := time_left; curr_data :∈ DATA-{NIL};
    resp := Eval(Service(curr_task),curr_data);
    CASE resp OF
      EITHER OK THEN
        results(curr_task) := curr_data;
        curr_task := Next(curr_task);
        IF curr_task = max_sv+1 THEN finished := TRUE END
      OR ROLLBACK THEN
        curr_task := Next ⁻¹ (curr_task);
        results := {curr_task} ⩤ results
      OR REPEAT THEN skip
      OR ABORT THEN finished := TRUE
      END
    END
  END;
timer =
  SELECT ¬ (in_data=NIL) ∧ finished = FALSE ∧ (time_left = old_time_left)
  THEN
    CHOICE
      time_left :∈ {xx | xx ∈ NAT1 ∧ xx<time_left}
    OR
      time_left, resp := 0, ABORT ;
      finished := TRUE
    END
  END;
calculate =
  SELECT ¬ (in_data=NIL) ∧ out_data = NIL ∧ finished = TRUE
  THEN
    IF resp = ABORT THEN out_data := Abort_data
    ELSE
      out_data := results(Next ⁻¹ (curr_task))
    END
  END;
res ← output =
  PRE ¬ (out_data = NIL)
  THEN
    res := out_data ; in_data,out_data := NIL, NIL
  END
END
```

situation is illustrated in Figure 10. However, if a more critical error has occurred, i.e., the variable resp gets the value *ROLLBACK*, the execution of several subservices preceding the failed service should be repeated as well. This case is depicted in Figure 11. The inverse of the function *Next* defines which subservices should be re-executed, i.e., to which subservice the execution flow should

rollback. In this case, we also delete the results of executing these subservices from *results*. Finally, if an unrecoverable error has occurred, i.e., the value of *resp* becomes *ABORT*, then the execution of the service is terminated (i.e., the variable fi*nished* is assigned *TRUE*) as shown in Figure 12.

Let us note, that the variable resp also obtains the value *ABORT* once the timeout has occurred. This is modelled in the operation *timer*. The system might be in a state where the value of *time_left* had already became zero, while the execution of the service has not yet been finished, as depicted in Figure 13.

In the refined machine *ACC_DEC* the guard of the event calculate is strengthened to ensure that the final result of the service is computed only after the execution of all subservices is finished (or aborted), i.e., when fi*nished = TRUE*.

The performed refinement has affected the *ACAM* part of the *ACC* pattern. The newly introduced events allowed us to define the details of execution of the decomposed service. In the VARIANT clause of *ACC_DEC* we not only ensure that the newly introduced events do not take control forever but also that execution of the service terminates.

Let us observe that our approach to introducing fault tolerance can be seen as an abstract implementation of the rollback error recovery frequently used in distributed systems as described by Elnozahy, Alvisi, Wang, and Johnson (2002). Indeed, the operation *handle* defines the rollback procedure by co-ordinating error recovery according to the check-points defined by the function *Next*. The stable data storage is modelled by the variable *results*. The operation *handle* ensures consistency of the system state by the appropriate updates of results.

While defining the execution flow over subservices in *ACC_DEC*, we abstracted away from modelling the details of the communication between *Service Director* and the external service providers – the USAP communication. Moreover, we omitted the explicit representation of the external service providers as such and modelled only the results of subservices provision. In our next refinement steps we decompose the obtained specification to introduce the detailed representation of the external service providers and the USAP communication.

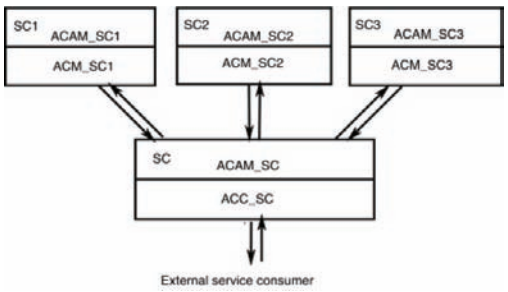
Formal Modelling of Service Distribution

Let us first consider the case of a ”centralized” service director shown in Figure 15. It is easy to observe that the service component *SC* providing the service *S* plays a role of the service consumer for the service components *SC1,..., SCN* providing the subservices S_1,..., S_N. We specify the service components *SC1,..., SCN* as the separate machines *ACC_SC1,... ACC_SCN* according to the proposed pattern *ACC*. The process of translating the UML2 models of *SC1,..., SCN* into B is similar to specifying *SC* at the *Service Specification* phase. The communicational parts of the included machines *ACC_SC1,..., ACC_SCN* specify the PSAPs of *SC1,..., SCN*. To ensure the match between the corresponding USAPs of SC and PSAPs of the external service components, we derive USAPs of SC from PSAPs of SC1,..., SCN.

To define the mechanism for communicating with *SC1,..., SCN*, we refine the operation *handle* to specify the communication on USAPs. Namely, we replace the nondeterministic assignments modelling specific stages of the service execution by the corresponding signaling scenarios: at the proper point of the execution flow, a desired service is requested by writing into the input channel of the corresponding included machine, and later the produced results are read from the output channel of this machine. Graphically this arrangement is depicted in Figure 19.

Modelling the case of a distributed service director is more complex. Let us assume that the execution flow of the service component *SC* is orchestrated by two service directors: the *ServiceDirector1*, which handles the communication

Figure 19. Specification architecture (case 1)

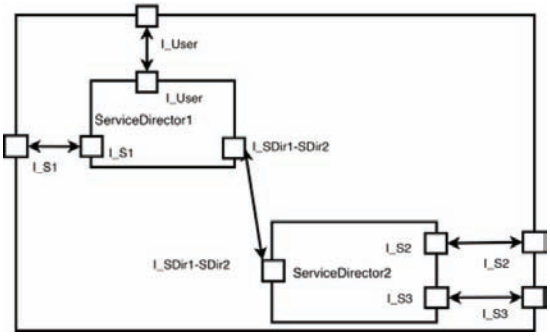


on PSAP of *SC* and communicates with *SC1*, and ServiceDirector2, which orchestrates the execution of the *SC2* and *SC3* services. The architecture diagram depicting the overall arrangement is shown in Figure 20.

The service execution proceeds according to the following scenario: via PSAP of *SC Service-Director1* receives the request to provide the service *S*. Upon this, via USAP of *SC*, it requests the component *SC1* to provide the service S_1. When the result of S_1 is obtained, *ServiceDirector1* requests *ServiceDirector2* to execute the rest of the service and return the result back. In its turn, *ServiceDirector2* at first requests *SC2* to provide the service S_2 and then *SC3* to provide service S_3. Upon receiving the result from S_3, it forwards it to *ServiceDirector1*. Finally, *Service-Director1* returns to the service consumer the result of the entire service *S* via PSAP of *SC*.

This complex behaviour can be captured in a number of refinement steps. At first, we observe

Figure 20. Architecture diagram (case 2)



that *ServiceDirector2*, coordinating execution of S_2 and S_3, can be modelled as a "large" service component *SC2-SC3*, which provides the services S_2 and S_3. Let us note that the execution flow in *SC2-SC3* is orchestrated by the "centralized" service director *ServiceDirector2*. We use this observation in our next refinement step. Namely, we refine the B machine modelling *SC* defined according to the *ACC_DEC* pattern by including into it the machines modelling the service components *SC1* and *SC2-SC3* and introducing the required communicating mechanisms. The result of this refinement step – the machine *SDirector1* – is given below (the parts of *SDirector1*, which coincide with the corresponding parts of *ACC_DEC* are replaced with dots). (See Figure 21, 22, and 23)

The machine *SDirector1* includes the machines *Comp1* and *SDirector2* specifying the service components *SC1* and *SC2-SC3* correspondingly. They are defined according to *ACC* and *ACC_DEC* patterns respectively. Since these machines can be obtained by a simple instantiation of these patterns, we omit their representation here.

The *Service Director* of *SC* communicates with the service component *SC1* and the *Service Director* of *SC2-SC3* by placing the corresponding requests in their input channels and reading

Figure 21. Specification of SDirector1

REFINEMENT *SDirector1*
REFINES *ACC_DEC*
INCLUDES *Comp1*, *SDirector2*
CONSTANTS
boolnum , ...
PROPERTIES
$boolnum \in \mathbf{BOOL} \rightarrow 0 \,..\, 1 \wedge$
$boolnum(\mathbf{FALSE}) = 0 \wedge boolnum(\mathbf{TRUE}) = 1 \wedge$
$max_sv = 3 \wedge Next = \{1 \mapsto 2, 2 \mapsto 4\} \wedge$
$(Service = ([C1_Service] \frown SD2_Service))$
VARIABLES
in_data, *out_data*, *time_left*, *old_time_left*,
curr_task, *resp*, *finished*, *results*, *curr_data*, *start_flag*
INVARIANT
$start_flag \in \mathbf{BOOL}$

Figure 22. Specification of SDirector1

```
VARIANT
  boolnum(start_flag)
INITIALISATION
  ... || start_flag := TRUE
ASSERTIONS
  Next ∈ 1 .. max_sv ⤖ 2 .. max_sv+1 ∧
  1 ∈ dom(Next) ∧ (max_sv+1) ∈ ran(Next) ∧
  ∀ ii. (ii ∈ dom(Next) ⇒ ii<Next(ii)) ∧
  ∀ ii. (ii ∈ dom(Next) ∧ ¬ (Next(ii)=max_sv+1) ⇒ Next(ii) ∈ dom(Next))
EVENTS
input(param,time) = ... END;
handle =
  SELECT ¬ (in_data=NIL) ∧ finished = FALSE ∧
        (time_left < old_time_left) ∧
        ((curr_task=1 ∧ C1_out_data ≠ NIL) ∨
         (curr_task=2 ∧ SD2_out_data ≠ NIL))
  THEN
    old_time_left := time_left;
    CASE curr_task OF
      EITHER 1 THEN curr_data ← C1_output
      OR 2 THEN curr_data ← SD2_output
      END
    END;
    resp := Eval(Service(curr_task),curr_data);
    CASE resp OF
      EITHER OK THEN
        results(curr_task) := curr_data;
        curr_task := Next(curr_task);
        IF curr_task = max_sv+1 THEN finished := TRUE END
      OR ROLLBACK THEN
        curr_task := Next ⁻¹ (curr_task);
        results := {curr_task} ⩤ results
      OR REPEAT THEN skip
      OR ABORT THEN finished := TRUE
      END
    END;
    start_flag := TRUE
  END;
starter =
  SELECT ¬ (in_data=NIL) ∧ finished = FALSE ∧
        (time_left = old_time_left) ∧
         start_flag = TRUE ∧
```

Figure 23. Specification of SDirector1

```
        ((curr_task=1 ∧ C1_in_data = NIL) ∨
         (curr_task=2 ∧ SD2_in_data = NIL))
  THEN
    CASE curr_task OF
      EITHER 1 THEN C1_input(in_data,time_left)
      OR 2 THEN SD2_input(results(Next ⁻¹ (curr_task)),time_left)
      END
    END;
    start_flag := FALSE
  END;
timer = ... END;
calculate = ... END;
res ← output = ... END;
END
```

the responses from their output channels. The order of requests is defined by the function *Next*. The function is instantiated in the PROPERTIES close to represent the particular architecture given in Figure 20. Requesting the services from *CS1* and *SC2-SC3* is modelled in the operation *starter*, reading the output channels of *SC1* and *SC2-SC3* in the operation *handle.* Note, that the operation *handle* have been refined to explicitly model obtaining a response from the requested component.

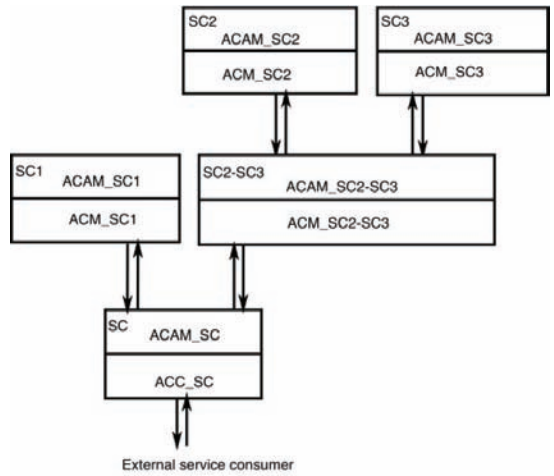
In our consequent refinement step we focus on decomposition of *SC2-SC3*. We single out separate service components *SC2* and *SC3* as before and refine *ServiceDirector2* to model communication with them. The final architecture of formal specification is shown in Figure 24. We omit the presentation of the detailed formal specifications – they are again obtained by recursive application of the proposed specification and refinement patterns.

At the further refinement steps we focus on particular service components and refine them (in the way described above) until the desired level of granularity is obtained. Once all external service components are in place, we can further decompose their specifications by separating their *ACM* and *ACAM* parts. Such a decomposition will allow us to concentrate on the communicational parts of the components and further refine them by introducing details of the required concrete communication protocols.

Discussion

The proposed approach to formalizing Lyra in B allows us to verify correctness of the Lyra decomposition and distribution phases. This is done by introducing generic patterns for communicating service components and then associating the

Figure 24. Specification architecture (case 2)



Lyra development steps with the corresponding B refinements on these patterns. In development of real systems we merely have to establish by proof that the corresponding components in a specific functional or network architecture are valid instantiations of these patterns. All together this constitutes a basis for automating industrial design flow of communicating systems.

The proposed development approach has a high degree of automation – all refinement steps were automatically proved by AtelierB – a tool supporting B. The proposed approach can be implemented not only on a platform supporting the classical B Method but also on the RODIN platform (described by the corresponding manual) as well.

6. CONCLUSION

In this paper we proposed a formal approach to development of communicating distributed systems. Our approach formalizes and extends Lyra – the UML2based design methodology adopted in Nokia. The formalization is done within the B Method and its extension EventB – a formal framework supporting system development by stepwise refinement. We derived the B specification and refinement patterns reflecting models and model transformations used in the development flow of Lyra. The proposed approach establishes a basis for automatic translation of UML2-based development of communicating systems into the specification and refinement process in B. Such automation would enable smooth integration of formal methods into existing development practice. Since UML is widely accepted in industry, we believe that our approach has a potential for wide industrial uptake.

Lyra adopts the service-oriented style for development of communicating systems. We presented the guidelines for deriving B specifications from corresponding UML2 models at each development stage of Lyra and verified the development by the corresponding B refinements. The major model transformations aim at service decomposition and distribution over the given platform. The proposed formal model of communication between the distributed service components is generic and can be instantiated by virtually any concrete communication protocol. Moreover, we demonstrated how to extend Lyra to integrate reasoning about fault tolerance in the entire development flow.

The initial formalization of Lyra has been undertaken using model checking techniques as described by Leppânen, Turunen, and Oliver (2004). However, since telecommunicating systems tend to be large and data intensive, this formalization was prone to the state explosion problem. Our approach helps to overcome this limitation.

Development of distributed communicating systems has been a topic of ongoing research over several decades. Our review of related work is confined to the consideration of the recent research conducted within B.

The pioneering work on formal development of distributed systems in Event B was done by Abrial, Cansell, and Mery (2003). They demonstrated how to prove termination of a complex distributed protocol in Event B. In our work we use the principles defined by them to formalize the service-oriented development of complex communicating systems.

Yadav and Butler (2005) used Event B to design fault tolerant transactions for replicated distributed database systems. They demonstrated how to formally verify by refinement that the design of a replicated database confirms to the one copy database abstraction. Similarly, in our work we use refinement to verify that the externally observable behaviour of distributed implementation of a service is equivalent to its centralized abstraction. However, our primary goal was not only formal verification of service development but also integration of modelling and refinement in B into the existing UML2-based development flow.

Treharne, Schneider, and Bramble (2003) investigated verification of safety and liveness properties of communicating components by combining the B Method and the process algebra CSP. However, they do not consider service decomposition and distribution aspects of the communicating system development.

Bostràm and Walden (2005) proposed a formal methodology (based on the B Method) for developing distributed grid systems. In their approach the B language is extended with grid-specific features and the system development is governed by B refinement. In our approach the system development is guided by the existing development practice, so that the refinement process is hidden behind the facade of UML2.

There is active research going on translating UML to B. Among these, the most notable is research conducted by Snook and Butler (2004) on designing the method and the U2B tool to support the automatic translation. In our future work we are planning to integrate our efforts with the U2B developers to achieve the automatic translation of Lyra into B. While doing this, we will focus specifically on translating models and model transformations used in Lyra to automate formalization of the entire UML-based development process in the domain of the communicating distributed systems. Moreover, we are planning to further enhance the proposed approach to address issues of concurrency.

REFERENCES

Abrial, J.-R. (1996). *The B-Book*. Cambridge, UK: Cambridge University Press. doi:10.1017/CBO9780511624162

Abrial, J.-R. (1996). Extending B without Changing it (for Developing Distributed Systems). In *Proceedings of the 1st Conference on the B Method* (pp.169-191). New York: Springer Verlag.

Abrial, J.-R., Cansell, D., & Mery, D. (2003). A mechanically proved and Incremental development of IEEE 1394 Tree Identity Protocol. *Formal Aspects of Computing*, *14*, 215–227. doi:10.1007/s001650300002

Atelier, B. (n.d.). *User and Reference Manuals*. Retrieved from http://www.atelierb.societe.com/index uk.html

Back, R., & Sere, K. (1994). From Action Systems to Modular Systems. In *Proceedings of the Formal Methods Europe (FME '94)* (LNCS 873, pp. 1-25). New York: Springer Verlag.

Back, R., & Sere, K. (1996). Superposition refinement of reactive systems. *Formal Aspects of Computing*, *8*(3), 1–23. doi:10.1007/BF01214918

Bostràm, P., & Walden, M. (2005). An Extension of Event B for Developing Grid Systems. *In Proceedings of ZB 2005: Formal Specification and Development in Z and B* (LNCS 3455, pp.142-161). New York: Springer Verlag.

Dijkstra, E. W. (1976). *A Discipline of Programming*. Upper Saddle River, NJ: Prentice Hall International.

Elnozahy, E. N., Alvisi, L., Wang, Y., & Johnson, D. B. (2002). A Survey of Rollback-Recovery Protocols in Message Passing Systems. *ACM Computing Surveys*, *34*(3). doi:10.1145/568522.568525

3GPP. (n.d.). *Technical specification 25.305: Stage 2 functional specification of UE positioning in UTRAN*. Retrieved from http://www.3gpp.org/ftp/Specs/htmlinfo/25305.htm

3GPP. (n.d.). *Technical specification 25.453: UTRAN Iupc interface positioning calculation application part (pcap) signalling*. Retrieved from http://www.3gpp.org/ ftp/Specs/html-info/25453.htm

Handbook for Correct System Construction. (n.d.). Retrieved from http://www.esil.univmrs.fr/ spc/matisse/Handbook/

Leppânen, S., Turunen, M., & Oliver, I. (2004). *Application Driven Methodology for Development of Communicating Systems*. Paper presented at the Forum on Specification and Design Languages, Lille, France.

Rumbaugh, J., Jakobson, I., & Booch, G. (1998). *The Unified Modelling Language Reference Manual*. Reading, MA: Addison-Wesley.

Snook, C., & Butler, M. (2004). *U2B -A tool for translating UML-B models into B. UML-B Specification for Proven Embedded System Design*. New York: Springer Verlag.

The RODIN platform. (n.d.). Retrieved from http://rodin-b-sharp.sourceforge.net/

Treharne, H., Schneider, S., & Bramble, M. (2003). *Composing Specifications Using Communication. (2003). Proceedings of ZB 2003: Formal Specification and Development in Z and B* (LNCS 2651). New York: Springer Verlag.

Yadav, D., & Butler, M. (2005). Application of Event B to Global Causal Ordering for Fault Tolerant Transactions. In *Proceedings of Workshop on Rigorous Engineering of Fault Tolerant Systems (REFT'2005)* (pp. 93-102).

This work was previously published in International Journal of Embedded and Real-Time Communication Systems, Volume 1, Issue 2, edited by Seppo Virtanen, pp. 61-85, copyright 2010 by IGI Publishing (an imprint of IGI Global).

Compilation of References

3 GPP. (2008, September). *Tech. Specification 36.211 V8.4.0, E-UTRA; Physical Channels and Modulation*.

3 GPP. (2008b, September). *Tech. Specification 36.212 V8.4.0, E-UTRA; Multiplexing and Channel Coding*.

3 GPP. (2009a). *3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3GPP System Architecture Evolution (SAE): Security Architecture (Release 9)*. Retrieved October 31, 2009, from http://www.3gpp.org/ftp/Specs/html-info/33401.htm

3 GPP. (2009b). *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Medium Access Control (MAC) Protocol Specification (Release 9)*. Retrieved October 31, 2009, from http://www.3gpp.org/ftp/Specs/html-info/36321.htm

3 GPP. (2009c). *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Radio Link Control (RLC) Protocol Specification (Release 8)*. Retrieved October 31, 2009, from http://www.3gpp.org/ftp/Specs/html-info/36322.htm

3 GPP. (2009d). *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Packet Data Convergence Protocol (PDCP) Specification (Release 8)*. Retrieved October 31, 2009, from http://www.3gpp.org/ftp/Specs/html-info/36323.htm

3 GPP. (n.d.). *Technical specification 25.305: Stage 2 functional specification of UE positioning in UTRAN*. Retrieved from http://www.3gpp.org/ftp/Specs/htm-linfo/25305.htm

3 GPP. (n.d.). *Technical specification 25.453: UTRAN Iupc interface positioning calculation application part (pcap) signalling*. Retrieved from http://www.3gpp.org/ftp/Specs/html-info/25453.htm

Abdi, S., Shin, D., & Gajski, D. (2003). Automatic communication refinement for system level design. In *DAC '03: Proceedings of the 40th Conference on Design Automation,* Anaheim, CA (pp. 300-305). New York: ACM Publishing.

Abrial, J. R., Börger, E., & Langmaack, H. (1996). *Formal Methods for Industrial Applications - Specifying and Programming the Steam Boiler Control* (LNCS 1165). New York: Springer Verlag.

Abrial, J.-R. (1996). Extending B without Changing it (for Developing Distributed Systems). In *Proceedings of the 1st Conference on the B Method* (pp.169-191). New York: Springer Verlag.

Abrial, J. R. (1996). *The B-book: Assigning Programs to Meanings*. Cambridge, UK: Cambridge University Press. doi:10.1017/CBO9780511624162

Abrial, J.-R., Cansell, D., & Mery, D. (2003). A mechanically proved and Incremental development of IEEE 1394 Tree Identity Protocol. *Formal Aspects of Computing*, *14*, 215–227. doi:10.1007/s001650300002

Adriahantenaina, A., Charlery, H., Greiner, A., Morteiz, L., & Zeferino, C. A. (2003). SPIN: A scalable, packet switched, on-chip micro-network. In *Proceedings of Design, Automation and Test in Europe: Designers'. Forum*, *2*, 70–73.

Agarwal, K., Agarwal, M., Sylvester, D., & Blaauw, D. (2006). Statistical interconnect metrics for physical-design optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *25*(7), 1273–1288. doi:10.1109/TCAD.2005.855954

Agha, G. A. (1986). *ACTORS: A model of concurrent computation in distributed systems*. Cambridge, MA: MIT Press.

Ahonen, T., & Nurmi, J. (2006). Programmable switch for shared bus replacement. In *Proceedings of the 2006 International Conference on Ph.D. Research in Microelectronics and Electronics (PRIME '06)* (pp. 241-244). Washington, DC: IEEE. ISBN: 1-4244-0156-9 DOI: 10.1109/RME.2006.1689941

Ahonen, T., Virtanen, S., Kylliäinen, J., Truscan, D., Kasanko, T., Siguenza-Tortosa, D., et al. (2004). A brunch from the coffee table-case study in NoC platform design. In J. Nurmi, H. Tenhunen, J. Isoaho, & A. Jantsch (Eds.), *Interconnect-centric design for advanced SoC and NoC* (pp. 425-453). Boston: Kluwer Academic Publishers.

Aikawa, H., et al. (2008). Variability aware modeling and characterization in standard cell in 45 nm CMOS with stress enhancement technique. *VLSI Technology Symposium,* 90-91.

Airoldi, R., Garzia, F., & Nurmi, J. (2009b). Implementation of a 64-point FFT on a Multi-Processor System-on-Chip. In *Proceedings of the 5th International Conference on Ph.D. Research in Microelectronics & Electronics (Prime '09)*, Cork, Ireland (pp. 20-23). Washington, DC: IEEE.

Airoldi, R., Garzia, F., Ahonen, T., Milojevic, D., & Nurmi, J. (2009a). Implementation of W-CDMA cell search on a FPGA based Multi-Processor System-on-Chip with power management. In *Proceedings of the IX International Symposium on Systems, Architectures, MOdeling and Simulation (SAMOS IX)* (LNCS, pp. 88-97). Berlin: Springer Verlag.

Akavia, A. Goldwasser & Vaikuntanathan, V. (2009). Simultaneous hardcore bits and cryptography against memory attacks. In O. Reingold (Ed.), *Theory of cryptography* (LNCS 5444, pp. 474-495).

Altera. (2007). *Stratix II Device Handbook*.

Altera. (2009). *Quartus® II Development Software handbook: SOPC Builder*. Retrieved May 2009, from http://www.altera.com/literature/lit-sop.jsp

Altera. (2009a). *FFT Megacore Function*.

Altera. (2009b). *Stratix IV Device Handbook*.

Altera. (2009c). *NiosII Processor Reference Handbook*.

Alvarez, M., Salami, E., Ramirez, A., & Valero, M. (2007). HD-VideoBench: A benchmark for evaluating high definition digital video applications. In *Workload Characterization* (pp. 120-125). Washington, DC: IEEE Computer Society. DOI:10.1109/IISWC.2007.4362188

Anwar, M. I., Virtanen, S., & Isoaho, J. (2008). A software defined approach to common baseband processing. *Journal of Systems Architecture*, *54*(8), 769–786. doi:10.1016/j.sysarc.2008.01.007

Arbab, F., & Mavaddat, F. (2002). Coordination through channel composition. In *Proceedings of Coordination Languages and Models* (LNCS 2315, pp. 22-39). New York: Springer Verlag.

ARM. (2004). *AMBA AXI Protocol Specification*. Retrieved October 31, 2009, from http://infocenter.arm.com

ARM. (2005). *AMBA*. Retrieved from http://www.arm.com/products/solutions/amba2overview.html

ARM. (2008). *ARM1176JZF-S Processor Technical Reference Manual*. Retrieved October 31, 2009, from http://infocenter.arm.com

Arpinen, T., Kukkala, P., Salminen, E., Hännikäinen, M., & Hämäläinen, T. D. (2006). Configurable multiprocessor platform with RTOS for distributed execution of UML 2.0 designed applications. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 1324-1329).

Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., et al. (2006). *The Landscape of Parallel Computing Research: A View from Berkeley* (Tech. Rep. No.UCB/EECS-2006-183). Berkeley, CA: University of California, Berkeley.

Asenov, A. (2007). Statistical device variability and its impact on yield and performance. In *Proceedings of the 13th IEEE International On-line Testing Symposium.*

Asenov, A., Brown, A. R., Davies, J. H., Kaya, S., & Slavcheva, G. (2003). Simulation of intrinsic parameter fluctuations in decananometre and nanometer scale MOSFETs. *IEEE Transactions on Electron Devices*, *50*(9), 1837–1852. doi:10.1109/TED.2003.815862

Asghar, R., & Liu, D. (2008, May). Dual standard reconfigurable hardware interleaver for turbo decoding. In *Proceedings of ISWPC*, Santorini, Greece (pp. 768-772).

Asghar, R., Wu, D., Eilert, J., & Liu, D. (in press). Memory Conflict Analysis and Implementation of a Reconfigurable Interleaver Architecture Supporting Unified Parallel Turbo Decoding. *Journal of Signal Processing Systems*. doi:.doi:10.1007/s11265-009-0394-8

Atelier, B. (2009). *Home*. Retrieved from http://www.atelierb.societe.com/

Atelier, B. (n.d.). *User and Reference Manuals*. Retrieved from http://www.atelierb.societe.com/index uk.html

Audsley, N., Burns, A., Richardson, M., Tindell, K., & Wellings, A. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, *8*, 284–292.

Azevedo, A., Zatt, B., Agostini, L., & Bampi, S. (2007). MoCHA: A bi-predictive motion compensation hardware for H.264/AVC decoder targeting HDTV. In *Circuits and Systems* (pp. 1617-1620). DOI: 10.1109/ISCAS.2007.378828

Back, R. J., & Kurki-Suonio, R. (1983). Decentralization of Process Nets with Centralized Control. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing* (pp. 131-142).

Back, R. J., Martin, A. J., & Sere, K. (1995). An action system specification of the Caltech asynchronous microprocessor. In *Proceedings of the 3rd International Conference: Mathematics on Program Construction (MPC'95)*, Kloster Irsee, Germany (pp. 159-179).

Back, R., & Sere, K. (1994). From Action Systems to Modular Systems. In *Proceedings of the Formal Methods Europe (FME '94)* (LNCS 873, pp. 1-25). New York: Springer Verlag.

Back, R., & von Wright, J. (1998). *Refinement calculus: A systematic introduction*. New York: Springer-Verlag.

Back, R. J., Petre, L., & Porres, I. P. (2001). Continuous action systems as a model for hybrid systems. *Nordic Journal of Computing*, *8*(1), 2–21.

Back, R. J., & Sere, K. (1996). Superposition refinement of reactive systems. *Formal Aspects of Computing*, *8*(3), 324–346. doi:10.1007/BF01214918

Back, R. J., & von Wright, J. (1998). *Refinement Calculus: A Systematic Introduction*. New York: Springer-Verlag.

Back, R. J., & von Wright, J. (2003). Compositional Action System Refinement. *Formal Aspects of Computing*, *15*(2-3), 103–117. doi:10.1007/s00165-003-0005-6

Back, R., & Sere, K. (1996). Superposition refinement of reactive systems. *Formal Aspects of Computing*, *8*(3), 1–23. doi:10.1007/BF01214918

Bahn, J. H., Yang, J., & Bagherzadeh, N. (2008). Parallel FFT algorithms on network-on-chips. In *Proceedings of the Fifth International Conference on Information Technology: New Generations (ITNG '08)* (pp. 1087-1093). Washington, DC: IEEE.

Bakoglu, H. B. (1990). *Circuits, interconnections, and packaging for VLSI*. Reading, MA: Addison-Wesley.

Balakrishnan, S., & Ozguner, F. (1998). A priority-driven flow control mechanism for real-time traffic in multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems*, *9*(7), 664–678. doi:10.1109/71.707545

Balart, J., Gonzalez, M., Martorell, X., Ayguade, E., Sura, Z., & Chen, T. (2007). A novel asynchronous software cache implementation for the Cell-BE processor. In Proceedings of *Languages and Compilers for Parallel Computing*. *Urbana (Caracas, Venezuela)*, *IL*, 125–140. doi:.doi:10.1007/978-3-540-85261-2_9

Banakar, R., Steinke, S., Lee, B., Balakrishnan, M., & Marwedel, P. (2002). Scratchpad memory: A design alternative for cache on-chip memory in embedded systems. In *Proceedings of Hardware/Software Codesign*, Estes Park (pp. 73-78). New York: ACM. DOI:10.1145/774789.774805

Barbero, L. G., & Thompson, J. S. (2006, June). Rapid prototyping of a fixedthroughput sphere decoder for MIMO systems. In *Proceedings of IEEE ICC*.

Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., & Whelan, C. (2006). The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, *94*(2), 370–382. doi:10.1109/JPROC.2005.862424

Barreto, P. S. L. M., & Naehrig, M. (2006). Pairing-friendly elliptic curves of prime order. In

Basu, S., & Moona, R. (2003). High level synthesis from Sim-nML processor models. In *Proceedings on the 16th International Conference on VLSI Design* (pp. 255260). Washington, DC: IEEE.

Baum, D. S., Salo, J., Milojevic, M., Kyoti, P., & Hansen, J. (2005, May). *MATLAB implementation of the interim channel model for beyond-3G systems (SCME)*.

Baumgarte, V., Ehlers, G., May, F., Nückel, A., Vorbach, M., & Weinhardt, M. (2003). PACT XPP-a Self-Reconfigurable Data Processing Architecture. *The Journal of Supercomputing*, *26*(2), 167–184. doi:10.1023/A:1024499601571

Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., et al. (2008). TILE64 Processor: A 64-Core SoC with Mesh Interconnect. In *Proceedings of the Digest of Technical Papers of IEEE International Solid-State Circuits Conference*, San Francisco, CA (pp. 88-598). Washington, DC: IEEE.

Benini, L., & De Micheli, G. (2001). Powering networks on chips. In *Proceedings of the International Symposium on System Synthesis.*

Benini, L., & De Micheli, G. (2002). Networks on chips: A new SoC paradigm. *Computer*, *35*(1), 70–78. doi:10.1109/2.976921

Benkeser, C., Burg, A., Cupaiuolo, T., & Huang, Q. (2009). Design and Optimization of an HSDPA Turbo Decoder ASIC. *IEEE Journal of Solid-state Circuits*, *44*(1), 98–106. doi:10.1109/JSSC.2008.2007166

Bergamaschi, R., Lee, W. R., Richardson, D., Bhattacharya, S., Muhlada, M., Wagner, R., et al. (2000). Coral - automating the design of systems-on-chip using cores. In *Proceedings of the IEEE 2000 Custom Integrated Circuits Conference,* Orlando, FL (pp. 109-112). Washington, DC: IEEE Computer Society.

Berkmann, J., Carbonelli, C., Dietrich, F., Drewes, C., & Xu, W. (2008). On 3G LTE Terminal Implementation − Standard, Algorithms, Complexities and Challenges. In *Proceedings of IEEE IWCMC*.

Bjerregaard, T., & Sparso, J. (2004). Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip. In *Proceedings of the 2nd Norchip Conference.*

Björkqvist, J., & Virtanen, S. (2006). Convergence of hardware and software towards platforms for radio technologies. *IEEE Communications Magazine*, *44*(11), 52–57. doi:10.1109/MCOM.2006.248165

Boneh, D. (2007). A brief look at pairings based cryptography. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science – FOCS 2007* (pp. 19-26). Washington, DC: IEEE.

Bonsangue, M. M., Kok, J. N., & Sere, K. (1998). An Approach to Object-Orientation in Action Systems. In *Proceedings of Mathematics of Program Construction* (LNCS 1422, pp. 68-95). New York: Springer Verlag.

Borjesson, P., & Sundberg, C.-E. (1979). Simple approximations of the error function Q(x) for communications applications. *IEEE transanctions on communications, 27*(3), 639-643.

Borkar, S., Karnik, T., Narendra, S., Tschanz, J., Keshavarzi, A., & De, V. (2003). Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the Design Automation Conference* (pp. 338-342).

Borkar, S. (2005). Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, *25*(6), 10–16. doi:10.1109/MM.2005.110

Bostràm, P., & Walden, M. (2005). An Extension of Event B for Developing Grid Systems. *In Proceedings of ZB 2005: Formal Specification and Development in Z and B* (LNCS 3455, pp.142-161). New York: Springer Verlag.

Bougard, B., et al. (2008). A Coarse-Grained Array based Baseband Processor for 100Mbps+ Software Defined Radio. In *Proceedings of the Design, Automation and Test in Europe*.

Bowman, K. A., Duvall, S. G., & Meindl, J. D. (2002). Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of Solid-state Circuits*, *37*(2), 183–190. doi:10.1109/4.982424

Brandenburg, M., Schollhorn, A., Heinen, S., Eckmüller, J., & Eckart, T. (2007). From Algorithm to First 3.5G Call in Record Time - A Novel System Design Approach Based on Virtual Prototyping and its Consequences for Interdisciplinary System Design Teams.

Braun, G., Wieferink, A., Schliebusch, O., Leupers, R., Meyr, H., & Nohl, A. (2003). Processor/memory co-exploration on multiple abstraction levels. In *Proceedings of the Design Automation and Test in Europe Conference and Exhibition* (pp. 966-971).

Brooks, C., Lee, E. A., Liu, X., Neuendorffer, S., Zhao, Y., & Zheng, H. (2007). *Heterogeneous concurrent modeling and design in Java: Volume 1: Introduction to Ptolemy II*. Berkeley, CA: EECS Department, University of California.

Brooks, D., & Martonosi, M. (2001). Dynamic thermal management for high-performance microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture* (pp.171-182). New York: ACM.

Brunelli, C., Campi, F., Mucci, C., Rossi, D., Ahonen, T., & Kylliäinen, J. (2008a). Design space exploration of an open-source, IP-reusable, scalable floating-point engine for embedded applications. *Journal of Systems Architecture*, *54*(12), 1143–1154. doi:10.1016/j.sysarc.2008.05.005

Brunelli, C., Garzia, F., & Nurmi, J. (2008b). A coarse-grain reconfigurable architecture for multimedia applications featuring subword computation capabilities. *Real-Time Image Processing*, *3*(1-2), 21–32. doi:10.1007/s11554-008-0071-3

Bucci, M., Giancane, L., Luzzi, R., Marino, M., Scotti, G., & Trifiletti, A. (2008). Enhancing power analysis attacks against cryptographic devices. *IET Circuits*. *Devices and Systems*, *2*(3), 298–305. doi:10.1049/iet-cds:20070166

Burg, A. (2005). VLSI implementation of MIMO detection using the sphere decoding algorithm. *IEEE Journal of Solid-state Circuits*, *44*(7), 1566–1577. doi:10.1109/JSSC.2005.847505

Camarda, F., Prevotet, J.-C., & Nouvel, F. (2009). Implementation of a reconfigurable fast fourier transform application to digital terrestrial television broadcasting. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'09)* (pp. 353-358).

Camposano, R., & Wilberg, J. (1996). Embedded system design. *Design Automation for Embedded Systems*, *1*(1-2), 5–50. doi:10.1007/BF00134682

Cesario, W. O., Lyonnard, D., Nicolescu, G., Paviot, Y., Sungjoo, Y., Jerraya, A. A., et al. (2002). Multiprocessor SoC platforms: A component-based design approach. *Design & Test of Computers,* 52-63.

Chapiro, D. M. (1984). *Globally–asynchronous locally–synchronous systems*. Unpublished PhD thesis, Stanford University, Stanford, CA.

Chaves, R., Kuzmanov, G., Vassiliadis, S., & Sousa, L. (2006) Reconfigurable memory based AES co-processor. In *Proceedings of the International Parallel and Distributed Processing Symposium* (p. 8).

Chen, C.-S. (2008). *On-chip current and power measurement techniques for integrated circuits with regulated power*. Doctoral dissertation, University of Rhode Island.

Chen, T., Zhang, T., Sura, Z., & Tallada, M. G. (2008). Prefetching irregular references for software cache on Cell. In *Code Generation and Optimization* (pp. 155-164). New York: ACM. DOI: 10.1145/1356058.1356079

Cheng, B. J., Roy, S., Roy, G., Brown, A., & Asenov, A. (2006). Impact of random dopant fluctuation on bulk CMOS 6-T SRAM scaling. In *Proceedings of the Solid-State Device Research Conference (ESSDERC 2006)*.

Cheng, B., Roy, S., Roy, G., Adamu-Lema, F., & Asenov, A. (2005). Impact of intrinsic parameter fluctuations in decanano MOSFETs on yield and functionality of SRAM cells. *Solid-State Electronics*, *49*(5), 740–746. doi:10.1016/j.sse.2004.09.005

Chen, L. (1998). AgentOS: the agent-based distributed operating system for mobile networks. *Crossroads*, *5*(2), 12–14. doi:10.1145/333151.333156

Cherkauer, B. S., & Friedman, E. G. (1995). Design of tapered buffers with local interconnect capacitance. *IEEE Journal of Solid-state Circuits*, *30*(2), 151–155. doi:10.1109/4.341744

Chinnery, D. G., & Keutzer, K. (2000). Closing the gap between ASIC and custom: an ASIC perspective. In *Proceedings of the Design Automation Conference (DAC2000)* (pp. 637-642).

Ciordas, C., Hansson, A., Goossens, K., & Basten, T. (2006). A Monitoring-Aware Network-on-Chip Design Flow. In *Proceedings of the 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools DSD,* Los Alamitosa, CA (pp. 97-106). Washington, DC: IEEE Computer Society.

Cockx, J. (2000). Efficient Modeling of Preemption in a Virtual Prototype. In *Proceedings of the 11th International Workshop on Rapid System Prototyping (RSP 2000)* (pp. 14-19). Washington, DC: IEEE Computer Society.

Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, *19*(90), 297–301. doi:10.2307/2003354

CoWare Inc. (2009). *CoWare Processor Designer*. Retrieved November 14, 2009, from http://www.coware.com/products/processordesigner.php

Dahlman, E., Parkvall, S., Sköld, J., & Beming, P. (2008). *3G Evolution: HSPA and LTE for Mobile Broadband*. Oxford, UK: Academic Press.

Dall'Osso, M., Biccari, G., Giovannini, L., Bertozzi, D., & Benini, L. (2003). Xpipes: A latency insensitive parameterized network-on-chip architecture for multi-processor SoCs. In *Proceedings of the 21st International Conference on Computer Design* (pp. 536-539).

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

Dally, W. (1992). Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, *3*(2), 194–205. doi:10.1109/71.127260

de Kock, E. A., Smits, W. J., van der Wolf, P., Brunel, J.-Y., Kruijtzer, W. M., Lieverse, P., et al. (2000). YAPI: Application modeling for signal processing systems. In *DAC '00: Proceedings of the 37th Conference on Design Automation,* Los Angeles (pp. 402-405). New York: ACM Publishing.

Desel, J., & Juhás, G. (2001). What is a Petri Net? Informal Answers for the Informed Reader. In H. Ehrig et al. (Eds.), *Unifying Petri Nets* (LNCS 2128, pp. 1-25).

Dijkstra, E. W. (1976). *A Discipline of Programming*. Upper Saddle River, NJ: Prentice Hall International.

Dingninou, A., Raouafi, F., & Berrou, C. (1999). Organisation de la memoire dans un turbo decodeur utilisant l'algorithm SUB-MAP. In *Proceedings of Gretsi* (pp. 71-74).

Distefano, A., & Me, G. (2008). An overall assesment of mobile internal acquisition tool. *Digital Investigation: The International Journal of Digital Forensics & Incident Response*, *5*, S121–S127.

Duato, J., Yalamanchili, S., & Ni, L. (2002). *Interconnection networks: an engineering approach*. San Francisco, CA: Morgan Kaufmann.

Eckart, T., & Schnieringer, M. (2006). Development and Verification of Embedded Firmware using Virtual System Prototypes. In *Proceedings of the International Symposium on System-on-Chip (SoC 2006)* (pp. 1-1). Washington, DC: IEEE.

Edler, J., & Hill, M. D. (2010). *Dinero IV trace-driven uniprocessor cache simulator*. Retrieved January 28, 2010, from http://pages.cs.wisc.edu/~markhill/DineroIV/

Edwards, S., Lavagno, L., Lee, E. A., & Sangiovanni-Vincentelli, A. (1997). Design of embedded systems: formal models, validation, and synthesis. *Proceedings of the IEEE*, *85*(3), 366–390. doi:10.1109/5.558710

Eilert, J., Wu, D., & Liu, D. (2008). Implementation of a Programmable Linear MMSE Detector for MIMO-OFDM. In *Proceedings of IEEE ICASSP*.

Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., & Ludvig, J. (2003). Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, *91*(1), 127–144. doi:10.1109/JPROC.2002.805829

Elnozahy, E. N., Alvisi, L., Wang, Y., & Johnson, D. B. (2002). A Survey of Rollback-Recovery Protocols in Message Passing Systems. *ACM Computing Surveys*, *34*(3). doi:10.1145/568522.568525

ESA. (2009). *Spacewire standard working group*. Retrieved April 2, 2009, from http://spacewire.esa.int/content/Home/HomeIntro.php

Evain, S., Diguet, J.-Ph., & Houzet, D. (2006). NoC design flow for TDMA and QoS management in a GALS context. *EURASIP Journal on Embedded Systems, 2006*(63656), 12.

Example Library API Reference. (2010). Retrieved January 28, 2010, from https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/3B6ED257EE6235D900257353006E0F6A/$file/SDK_Example_Library_API_v3.0.pdf

Fan, X., Ellis, C., & Lebeck, A. (2001). Memmory controller policies for DRAM power management. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*.

Fauth, A., Van Praet, J., & Freericks, M. (1995). Describing instruction set processors using nML. In *Proceedings of the European Design and Test Conference* (pp. 503507). Washington, DC: IEEE.

Fitzgerald, J., Larsen, P., Muherjee, P., Plat, N., & Verhoef, M. (Eds.). (2005). *Validated designs for object-oriented systems*. London: Springer-Verlag.

Flachs, B., Asano, S., Dhong, S. H., Hofstee, H. P., Gervais, G., & Kim, R. (2006). The microarchitecture of the synergistic processor for a cell processor. *IEEE Journal of Solid State Circuits*, *41*(1), 63–70. doi:10.1109/JSSC.2005.859332

FreeRTOS. (2003). *The FreeRTOS™ Project*. Retrieved October 31, 2009, from http://www.freertos.org

Furber, S., & Bainbridge, J. (2005). Future trends in SoC interconnect. In *Proceedings of the IEEE international symposium on VLSI design, automation and test*.

Gajski, D. D., Samar, A., Gerstlauer, A., & Schirner, G. (2009). *Embedded System Design: Modeling, Synthesis and Verification*. New York: Springer.

Gajski, D. D., & Vahid, F. (1995). Specification and design of embedded hardware-software systems. *IEEE Design & Test of Computers*, *12*(1), 53–67. doi:10.1109/54.350695

Garlan, D., Monroe, R. T., & Wile, D. (2000). Acme: Architectural description of component-based systems. In *Foundations of component-based systems* (pp. 47-67). New York: Cambridge University Press.

Garzia, F., Airoldi, R., Brunelli, C., Giliberto, C., & Nurmi, J. (2009). Mapping of the FFT on a reconfigurable architecture targeted to SDR applications. In *Proceedings of International Symposium on System-on-Chip (SOC'09)*, Tampere, Finland (pp. 157-160). Washington, DC: IEEE.

Garzia, F., Brunelli, C., & Nurmi, J. (2008a). A pipelined infrastructure for the distribution of the configuration bitstream in a coarse-grain reconfigurable array. In *Proceedings of the 4th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC '08)*, Montpellier, France (pp. 188-191). ISBN: 978-84-691-3603-4

Garzia, F., Brunelli, C., Rossi, D., & Nurmi, J. (2008b). Implementation of a floatingpoint matrix-vector multiplication on a reconfigurable architecture. In *Proceedings of the 15th Reconfigurable Architecture Workshop (RAW '08)* (pp. 1-6). Washington, DC: IEEE.

Gehrmann, C., & Ståhl, P. (2006). Mobile Platform Security. *ERICSSON REVIEW. The Telecommunications Technology Journal*, *84*(2), 59–70.

Girault, C., & Valk, R. (Eds.). (2003). *Petri Nets for systems engineering: A guide to modeling, verification, and applications*. Berlin Heidelberg, Germany: Springer-Verlag.

Goldsmith, A. (2005). *Wireless Communications*. Cambridge, UK: Cambridge University Press.

Goldwasser, S., Kalai, Y. T., & Rothblum, G. N. (2008). One-time programs. In D. Wagner (Ed.), *Advances in Cryptology - CRYPTO 2008.* (LNCS 5157, pp. 39-56).

Golub, G. H., & Van Loan, C. F. (1996). *Matrix Computations* (3rd ed.). Baltimore: Johns Hopkins University Press.

Gonzalez, M., Vujic, N., Martorell, X., Ayguade, E., Eichenberger, A. E., Chen, T., et al. (2008). Hybrid access-specific software cache techniques for the Cell BE architecture. In *Parallel Architectures and Compilation Techniques* (pp. 292-302). New York: ACM. DOI: 10.1145/1454115.1454156

Goossens, G., Lanneer, D., Geurts, W., & Van Praet, J. (2006). Design of ASIPs in multi-processor SoCs using the Chess/Checkers retargetable tool suite. In *Proceedings of the International Symposium on System-onChip* (pp. 14). Washington, DC: IEEE.

Goossens, K., Dielissen, J., Gangwal, O. P., Pestana, S. G., Radulescu, A., & Rijpkema, E. (2005). A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC Design and verification. In *Proceedings of the conference on Design, Automation and Test in Europe* (Vol. 2, pp. 1182-1187).

Goossens, K., Dielissen, J., & Radulescu, A. (2005). Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design & Test of Computers*, *22*(5), 414–421. doi:10.1109/MDT.2005.99

Gössel, M., Rebel, B., & Creutzburg, R. (1994). *Memory architecture and parallel access*. New York: Elsevier Science.

Grecu, C., Pratim Pande, P., Ivanov, A., & Saleh, R. (2004). A Scalable Communication–Centric SoC Interconnect Architecture. In *Proceedings of the 5th International Symposium on Quality Electronic Design* (pp. 343-348).

Grötker, T., Liao, S., Martin, G., & Swan, S. (2002). *System design with SystemC*. Norwell, MA: Kluwer Academic Publishers.

Gschwind, M., Hofstee, H., Flachs, B., Hopkins, M., Watanabe, Y., & Yamazaki, T. (2006). Synergistic processing in Cell's multicore architecture. *IEEE Micro*, *26*(2), 10–24..doi:10.1109/MM.2006.41

Guang, L., & Jantsch, A. (2006). Adaptive Power Management for the On-Chip Communication Network. In *Proceedings of the 9th EUROMICRO DSD Conference,* Los Alamitos, CA (pp. 649-656). Washington, DC: IEEE Computer Society.

Guang, L., Nigussie, E., Koskinen, L., & Tenhunen, H. (2009). Autonomous DVFS on Supply Islands for Energy-Constrained NoC Communication. In *Proceedings of ARCS 09* (LNCS 5545, pp. 183-194). Berlin: Springer Verlag.

Gupta, R., Le Guernic, P., Shukla, S. K., & Talpin, J.-P. (Eds.). (2004). *Formal methods and models for system design: A system level perspective.* Boston: Kluwer Academic Publishers.

Haataja, K., & Hyppönen, K. (2008). Man-in-the-middle attacks on Bluetooth: A comparative analysis, a novel attack, and countermeasures. In *Proceedings of the 3rd International Symposium on Communications, Control and Signal Processing* (pp. 1096-1102).

Hadjiyiannis, G., Hanono, S., & Devadas, S. (1997). ISDL: An instruction set description language for retargetability. In *Proceedings of the 34th Design Automation Conference* (pp. 299302). Washington, DC: IEEE.

Hadjiyiannis, G., & Devadas, S. (2003). Techniques for accurate performance evaluation in architecture exploration. *IEEE Transactions on Very Large Scale Integration (VLSI). Systems*, *11*(4), 601615.

Halambi, A., Grun, P., Ganesh, V., Khare, A., Dutt, N., & Nicolau, A. (1999). EXPRESSION: a language for architecture exploration through compiler/simulator retargetability. In D. Borrione & R. Ernst (Eds.), *Proceedings of the Design Automation and Test in Europe Conference and Exhibition* (pp. 485490). Washington, DC: IEEE.

Hämäläinen, H., Heikkinen, J., Hannikainen, M., & Hämäläinen, T. D. (2005). Design of transport triggered architecture processors for wireless encryption. In *Proceedings of the 8th Euromicro Conference on Digital System Design* (pp. 144-152).

Han, L., Han, J., Zeng, X., Lu, R., & Zhao, J. (2008). A programmable security processor for cryptography algorithms. In *Proceedings of 9th the International Conference on Solid-State and Integrated Circuit Technology* (pp. 2144-2147).

Handbook for Correct System Construction. (n.d.). Retrieved from http://www.esil.univmrs.fr/ spc/matisse/ Handbook/

Handshake Solutions. (2009). *Haste.* Retrieved from http://www.handshakesolutions.com/

Hary, S., & Ozguner, F. (1997). Feasibility test for real-time communication using wormhole routing. *IEE Proceedings. Computers and Digital Techniques*, *144*(5), 273–278. doi:10.1049/ip-cdt:19971369

Hausner, J., & Drewes, C. (2008). Cellular Phones: Future Implementation Trends. *Journal of RF-Engineering and Telecommunications*, *62*(7-8), 185–190.

Hemani, A., Jantch, A., Kumar, S., Postula, A., Öberg, J., Millberg, M., & Lindqvist, D. (2000). Network on a Chip: An architecture for billion transistor era. In *Proceedings of the IEEE NorChip Conference*.

Henkel, J., Wolf, W., & Chakradhar, S. (2004). On-chip networks: A scalable, communication-centric embedded system design paradigm. In *Proceedings of the 17th International Conference on VLSI Design*.

Hennessy, J. H., & Patterson, D. A. (2003). *Computer architecture - a quantitative approach* (3rd ed.). Morgan Kaufmann.

Hessel, S., Bruns, F., Bilgic, A., Lackorzynski, A., Härtig, H., & Hausner, J. (2008). Acceleration of the L4/Fiasco Microkernel Using Scratchpad Memory. In *Proceedings of the First Workshop on Virtualization in Mobile Computing (MobiVirt '08)* (pp. 6-10). New York: ACM Press.

Hessel, S., Szczesny, D., Traboulsi, S., Bilgic, A., & Hausner, J. (2009). On the Design of a Suitable Hardware Platform for Protocol Stack Processing in LTE Terminals. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering (CSE '09)* (pp. 1-8). Washington, DC: IEEE Computer Society.

Hewitt, C., Bishop, P., & Steiger, R. (1973). A universal modular actor formalism for artificial intelligence. In. *Proceedings of the IJCAI*, *III*, 235–245.

HiTech Global LLC. (2009). *DDR SDRAM Controller IP Core, Product Specification*. Retrieved November 14, 2009, from http://www.hitechglobal.com/ipcores/ddrsdram.htm

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Upper Saddle River, NJ: Prentice Hall.

Hodjat, A., Hwang, D. D., Lai, B., Tiri, K., & Verbauwhede, I. (2005). A 3.84 gbits/s AES crypto processor with modes of operation in a 0.18 µm CMOS technology. In *Proceedings of the 15th ACM Great Lakes Symposium on VLSI* (pp. 60-63).

Hoffmann, A., Meyr, H., & Leupers, R. (2002). *Architecture Exploration for Embedded Processors with LISA*. Boston: Kluwer Academic Publishers.

Ho, R., Mai, K. W., & Horowitz, M. A. (2001). The future of wires. *Proceedings of the IEEE*, *89*(4), 490–504. doi:10.1109/5.920580

Hu, J., & Marculescu, R. (2005). Energy- and performance-aware mapping for regular NoC architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *24*(4), 551–562. doi:10.1109/TCAD.2005.844106

IBM. (2006). *CoreConnect.* Retrieved from http://www.ibm.com

IEEE. (1990). *ISA IEEE personal computer bus standard P996, draft D2.02*. Retrieved April 2, 2009, from http://standards.ieee.org/

In Proceedings of the Design. *Automation & Test in Europe Conference (DATE '07)* (pp. 1-3). Washington, DC: IEEE.

Indrusiak, L. S., & Glesner, M. (2007). Specification of alternative execution semantics of UML sequence diagrams within actor-oriented models. In *Proceedings of the Symposium on Integrated Circuits and System Design* (pp. 330-335).

Indrusiak, L. S., Ost, L., Möller, L., Moraes, F., & Glesner, M. (2008). Applying UML interactions and actor-oriented simulation to the design space exploration of network-on-chip interconnects. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI* (pp. 491-494).

Indrusiak, L. S., Thuy, A., & Glesner, M. (2007). Executable system-level specification models containing UML-based behavioral patterns. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 301-306).

Indrusiak, L., Ost, L., Möller, L., Moraes, F., & Glesner, M. (2008). Applying UML Interactions and Actor-oriented Simulation to the Design Space Exploration of Network-on-Chip Interconnects. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI* (pp. 491-494).

Isci, C., Buyuktosunoglu, A., Cher, C.-Y., Bose, P., & Martonosi, M. (2006), An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proceedings Of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (pp.347-358). Washington, DC: IEEE Computer Society.

Ishimaru, K. (2007). 45nm/32nm CMOS-Challenge and Perspective. In *Proceedings of the 37th European Solid-State Research Conference (ESSDERC 2007)* (pp. 32-35).

ITRS. (2005). *International Technology Roadmap for Semiconductors*. Semiconductor Industry Association.

James, P. (2007). Can SDV technology be utilized in a smartphone to prevent forensic analysis? In *Proceedings of the 5th Australian Digital Forensics Conference* (pp. 164-178).

Jantsch, A. (2006). Models of computation for networks on chip. In *Proceedings of Sixth International Conference on Application of Concurrency to System Design* (pp. 165-178).

JEDEC. (2008). *JEDEC standard JESD79-3C: DDR3 SDRAM*. Arlington: JEDEC Solid State Technology Association.

Jerraya, A. A., Bouchhima, A., & Frédéric, P. (2006). Programming models and HW-SW interfaces abstraction for multi-processor SoC. In *DAC '06: Proceedings of the 43rd Annual Conference on Design Automation,* San Francisco (pp. 280-285). New York: ACM Publishing.

Kahle, J., Day, M., Hofstee, H., Johns, C., Maeurer, T., & Shippy, D. (2005). Introduction to the Cell multiprocessor. [Riverton, NJ: IBM Corp.]. *IBM Journal of Research and Development*, *49*(4), 589–604. doi:10.1147/rd.494.0589

Kahn, G. (1974). The semantics of a simple language for parallel programming. In *IFIP Congress 74*. Amsterdmam, The Netherlands: North-Holland Publishing.

Kamalizad, A. H., Pan, C., & Bagherzadeh, N. (2003). Fast Parallel FFT on a Reconfigurable Computation Platform. In *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBACPAD' 03).* Washington, DC: IEEE.

Kammler, D., Zhang, D., Schwabe, P., Scharwaechter, H., Langenberg, M., & Auras, D. (2009). Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves. In Clavier, C., & Gaj, K. (Eds.), *Cryptographic Hardware and Embedded Systems - CHES 2009* (pp. 254–271). Heidelberg, Germany: Springer. doi:10.1007/978-3-642-04138-9_19

Karnik, T., De, V., & Borkar, S. (2004). Statistical design for variation tolerance: key to continued Moore's law. In *Proceedings of the International Conference on Integrated Circuit Design and Technology (ICICDT '04)* (pp. 175-176).

Kavaldjiev, N., & Smit, G. (2003). A survey of efficient on-chip communications for SoC. In *Proceedings of the 4th Progress Symposium on Embedded Systems*, 129-140.

Kempf, T., Karuri, K., Wallentowitz, S., Ascheid, G., Leupers, R., & Meyr, H. (2006). A SW performance estimation framework for early system-level-design using fine-grained instrumentation. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 468-473).

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36*(1), 41–50. doi:10.1109/MC.2003.1160055

Keutzer, K., & Orshansky, M. (2002). From blind certainty to informed uncertainty. In *Proceedings of the IEEE International Workshop on Timing Issues in the specification and synthesis of digital systems* (pp. 37-41).

Keutzer, K., Newton, A. R., Rabaey, J. M., & Sangiovanni-Vincentelli, A. (2000). System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on CAD*, *19*(12), 1523–1543.

Khailany, B. K., Williams, T., Lin, J., Long, E. P., Rygh, M., & Tovey, D. W. (2008). A programmable 512 GOPS stream processor for signal, image, and video processing. *IEEE Journal of Solid-State Circuits*, *43*(1), 202–213. doi:10.1109/JSSC.2007.909331

Khronos. (2008). *OpenCL specification V1.0*. Beaverton, OR: Author.

Kim, B., Kim, J., Hong, S., & Lee, S. (1998). A real-time communication method for wormhole switching networks. In *Proceedings of the International Conference on Parallel Processing*.

Kim, D., Kim, M., & Sobelman, G. (2005). Parallel FFT computation with a CDMA-based network-onchip. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS'05)* (Vol. 2, pp. 1138-1141). Washington, DC: IEEE.

Kim, M., Kim, D., & Sobelman, G. E. (2006). Network-on-Chip link analysis under power and performance constraints. In *Proceedings of ISCAS 2006.*

Kocher, P., Lee, R., McGraw, G., Raghunathan, A., & Ravi, S. (2004). Security as a new dimension in embedded system design. In *Proceedings of the 41st Annual ACM IEEE Conference on Design Automation* (pp. 753-760).

Kordon, F., & Lemoine, M. (Eds.). (2004). *Formal methods for embedded distributed systems: How to master the complexity*. Boston: Kluwer Academic Publishers.

Kriaa, L., Bouchhima, A., Youssef, W., Petrot, F., Fouillart, A.-M., & Jerraya, A. (2006). Service based component design approach for flexible hardware/software interface modeling. In *Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping, 2006,* Chania, Crete (pp. 156-162). Washington, DC: IEEE Computer Society.

Kropf, T. (1999). *Introduction to formal hardware verification*. Berlin Heidelberg, Germany: Springer-Verlag.

Kuo, J.-C., Wen, C.-H., & Wu, A.-Y. (2003). Implementation of a programmable 64-2048-point FFT/IFFT processor for OFDM-based communication systems. In *Proceedings of the 2003 International Symposium on Circuits and Systems (ISCAS'03)* (Vol. 2, pp. 121-124).

Kuon, I., & Rose, J. (2006). Measuring the gap between FPGAs and ASICs. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays (FPGA '06)* (pp. 21-30). New York: ACM.

Kylliäinen, J., Ahonen, T., & Nurmi, J. (2007). General-purpose embedded processor cores- the COFFEE RISC example. In J. Nurmi (Ed.), *Processor Design: System-on-Chip Computing for ASICs and FPGAs* (Chapter 5, pp. 83-100). New York: Springer. ISBN-10: 1402055293 ISBN-13: 978-1-4020-5529-4

LaCie. (2009). *FireWire® 800 – technology brief*. Retrieved April 2, 2009, from http://www.lacie.com/download/more/WhitePaper_FireWire_800.pdf

Lafond, S. (2009). *Simulation of Embedded Systems for Energy Consumption Estimation* (PhD Thesis). TUCS Dissertation Series, (113).

Lafond, S., Nybom, K., Björqvist, J., & Lilius, J. (2008). Receiver Coding Gain in DVB-H Terminals using Application Layer FEC Codes. In *Proceedings of the 3rd International Conference on Digital Telecommuincations (ICDT 2008)* (pp. 110-116). Washington, DC: IEEE.

Lafond, S., & Lilius, J. (2007). Energy consumption analysis for two embedded Java virtual machines. *Journal of Systems Architecture*, *53*(5-6), 328–337. doi:10.1016/j.sysarc.2006.10.003

Lahiri, K., Raghunathan, A., & Dey, S. (2001). Evaluation of the traffic–performance characteristics of system–on–chip communication architectures. In *Proceedings of 14th International Conference on VLSI Design* (pp. 29-35).

Lam, W. K. (2005). *Hardware design verification: Simulation and formal method-based approaches*. Upper Saddle River, NJ: Pearson Education.

Landman, P. (1996). High-level power estimation. In *Proceedings of International Symposium on Low-Power Electronics and Design* (pp. 29-35).

Lappeteläinen, A., Tuopola, J.-M., Palin, A., & Eriksson, T. (2008). *Networked systems, services and information, the ultimate digital convergence*. Paper presented at the 1st International NoTA conference, Helsinki, Finland.

Laprie, J., Avizienis, A., & Kopetz, H. (Eds.), *Dependability: Basic Concepts and Terminology*. New York: Springer Verlag.

Lecomte, T. (2008). Safe and Reliable Metro Platform Screen Doors Control/Command Systems. In *Proceedings of the 14th International Conference on Formal Methods (FM 2008)* (LNCS 5014, pp. 430-434). New York: Springer-Verlag.

Lee, J., Seo, S., Kim, C., Kim, J., Chun, P., Sura, Z., et al. (2008). COMIC: A coherent shared memory interface for Cell-BE. In *Parallel Architectures and Compilation Techniques* (pp. 303-314). New York: ACM. DOI: 10.1145/1454115.1454157

Lee, K., Lee, S.-J., & Yoo, H.-J. (2004). SILENT: serialized low energy transmission coding for on-chip interconnection networks. In *Proceedings of the IEEE/ACM International conference on Computer Aided Design* (pp. 448-451).

Lehoczky, J. (1990). Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the IEEE Real Time Sytem Symposium.*

Lei, T., & Kumar, S. (2003). A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proceedings of the Euromicro Symposium on Digital Systems Design* (pp. 180-189).

Leppânen, S., Turunen, M., & Oliver, I. (2004). *Application Driven Methodology for Development of Communicating Systems*. Paper presented at the Forum on Specification and Design Languages, Lille, France.

Leupers, R., & Marwedel, P. (1998). Retargetable code generation based on structural processor description. *Design Automation for Embedded Systems*, *3*(1), 75108. doi:10.1023/A:1008807631619

Leuschel, M., & Butler, M. (2005). ProB: A Model Checker for B. In *Proceedings of FME 2003* (LNCS 2805, pp. 855-874). New York: Springer Verlag.

Lin, Z., Spanos, C. J., Milor, L. S., & Lin, Y. T. (1998). Circuit sensitivity to interconnect variation. *IEEE Transactions on Semiconductor Manufacturing*, *11*(4), 557–568. doi:10.1109/66.728552

Liu, C., & Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for computing Machinery, 20*(1), 46-61.

Liu, D. (2008*). Embedded DSP processor design - application specific instruction set processors*. New York: Elsevier.

Liu, J., Zheng, L. R., & Tenhunen, H. (2004). Interconnect intellectual property for Network-on-Chip (NoC). *Journal of Systems Architecture*, *50*(2-3), 65–79. doi:10.1016/j.sysarc.2003.07.003

Liu, Y., Chen, P., Xie, G., Liu, Z., & Li, Z. (2008). The design of low-power asynchronous DES coprocessor for sensor network encryption. In. *Proceedings of the International Symposium on Computer Science and Computational Technology*, *2*, 190–193. doi:10.1109/ISCSCT.2008.228

Li, Y. G., Seshadri, N., & Ariyavisitakul, S. (1999). Channel Estimation for OFDM Systems with Transmitter Diversity in Mobile Wireless Channels. *IEEE Journal on Selected Areas in Communications*, *17*(3).

Lotze, N., Ortmanns, M., & Manoli, Y. (2008). Variability of flip-flop timing at sub-threshold voltages. In *Proceedings of the 13th International Symposium on Low Power Electronics and Design (ISLPED 2008).*

Lu, Z., Jantsch, A., & Sander, I. (2005). Feasibility analysis of messages for on-chip networks using wormhole routing. In *Proceedings of the Conferenece on Asia South Pacific Design Automation.*

Lueftner, T., Berthold, J., Pacha, C., Georgakos, G., Sauzon, G., & Hoemke, O. (2007). A 90-nm CMOS Low-Power GSM/EDGE Multimedia-Enhanced Baseband Processor with 380-MHz ARM926 Core and Mixed-Signal Extensions. *IEEE Journal of Solid-state Circuits*, *42*(1), 134–144. doi:10.1109/JSSC.2006.886528

Lundgren, B., & Ödlund, A. (2007). *Expose of patterns in parallel memory access*. Unpublished master's thesis, Linköping University, Sweden.

Määttä, S., Indrusiak, L. S., Ost, L., Möller, L., Nurmi, J., Glesner, M., et al. (2008). Validation of executable application models mapped onto network-on-chip platforms. In *Proceedings of the IEEE Symposium on Industrial Embedded Systems* (pp. 118-125).

Manolakis, K., Ibing, A., & Jungnickel, V. (2008). Performance Evaluation of a 3GPP LTE Terminal Receiver. In *Proceedings of the 14th European Wireless Conference (EW '08)* (pp. 1-5). Berlin: VDE Verlag.

Marcon, C., Borin, A., Susin, A., Carro, L., & Wagner, F. (2005). Time and energy efficient mapping of embedded applications onto NoCs. In *Proceedings of the Asia South Pacific Design Automation Conference* (pp. 33-38).

Marescaux, T., & Corporaal, H. (2007). Introducing the SuperGT network-on-chip: SuperGT QoS: More than just GT. In *Proceedings of the 44th annual conference on Design automation* (pp. 116-121).

Martin, G. (2002). UML for embedded systems specification and design: Motivation and overview. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 773-775).

Martin, G. (2006). Overview of the MPSoC design challenge. In *Proceedings of the 43rd Annual Conference on Design Automation,* San Francisco (pp. 274-279). New York: ACM Publishing.

Martín-Sacristán, D., Monserrat, J. F., Cabrejas-Penuelas, J., Calabuig, D., Garrigas, S., & Cardona, N. (2009). On the Way towards Fourth-Generation Mobile: 3GPP LTE and LTE-Advanced. *EURASIP Journal on Wireless Communications and Networking,* 1-10.

Mehlfuhrer, C., Caban, S., & Rupp, M. (2008). *Experimental Evaluation of Adaptive Modulation and Coding in MIMO WiMAX with Limited Feedback*. EURASIP Journal on Advances in Signal Processing.

Mei, B., Vernalde, S., Verkest, D., Man, H. D., & Lauwereins, R. (2003). ADRES: An architecture with tightly-coupled VLIW processor and coarse grained reconfigurable matrix. In *Field-Programmable Logic and Applications* (Vol. 2778, pp. 61-70). ISBN 978-3-540-40822-2

Metsälä, T., Westerlund, T., & Plosila, J. (2009). *Introducing Action Systems Class Hierarchy to SystemC Modeling* (Tech. Rep. No. 946). Turku, Finland: Turku Centre for Computer Science (TUCS).

Meyer, R., & Schwarz, K. (1990). FFT implementation on dsp-chips-theory and practice. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP '90)* (Vol. 3, pp. 1503-1506).

Michael, M. (2005). *Energy awareness for mobile devices*. Paper presented at the Research Seminar on Energy Awareness, University of Helsinki, Helsinki, Finland.

Millberg, M., Nilsson, E., Thid, R., Kumar, S., & Jantsch, A. (2004). The Nostrum backbone-a communication protocol stack for networks on chip. In *Proceedings of 17th International Conference on VLSI Design* (pp. 693-696).

MIPI. (2009). *Mobile industry processor interface alliance*. Retrieved April 2, 2009, from http://www.mipi.org/

Mishra, P., Kejariwal, A., & Dutt, N. (2003). Rapid exploration of pipelined processors through automatic generation of synthesizable RTL models. In *Proceedings of the 14th IEEE International Workshop on Rapid Systems Prototyping* (pp. 226232). Washington, DC: IEEE.

Mishra, P., & Dutt, N. (Eds.). (2008). *Processor Description Languages*. Burlington, VT: Morgan Kaufmann Publishers.

Mishra, P., Mamidipaka, M., & Dutt, N. (2004). Processor-memory coexploration using an architecture description language. *ACM Transactions on Embedded Computing Systems*, *3*(1), 140162. doi:10.1145/972627.972634

Moadeli, M., Vanderbauwhede, W., Shahrabi, A., & Ould-Khaoua, M. (2008). Modeling differentiated services-based QoS in wormhole-routed NoCs. In *Proceedings of 22nd International Conference on Advanced Information Networking and Applications* (pp. 942-949).

Mondal, M., Ragheb, T., Wu, X., Aziz, A., & Massoud, Y. (2007). Provisioning On-Chip networks under buffered RC interconnect delay variations. In *Proceedings of the ISQED '07.*

Moore, G. E. (1965). Cramming More Components onto Integrated Circuits. *Electronics, 38*(8).

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

Murali, S., Benini, L., & De Micheli, G. (2005). Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees. In *Proceedings of the Asia South Pacific Design Automation Conference* (pp. 27-32).

Mutka, M. (1994). Using rate monotonic scheduling technology for real-timecommunications in a wormhole network. In *Proceedings of the 2nd workshop on parallel and distributed real time systems.*

Naji, H. R., Wells, B. E., & Aborizka, M. (2002). Hardware agents. *In proceedings of the 11th International Conference on Intelligent Systems on Emerging Technologies*, Boston, MA.

Naji, H. R., Wells, B. E., & Etzkorn, L. (2004). Creating an adaptive embedded system by applying multi-agent techniques to reconfigurable hardware. *Future Generation Computer Systems*, *20*(6), 1055–1081. doi:10.1016/j.future.2004.02.002

Najm, F. (1998). A survey of power estimationT Techniques in vlsi circuits. *IEEE Transactions on VLSI Systems*, *2*, 446–455. doi:10.1109/92.335013

Nakajima, M., Yamamoto, T., Ozaki, S., Sezaki, T., Kanakagi, T., Furuzono, T., et al. (2002). A 400MHz 32b embedded microprocessor core AM34–1 with 4.0GB/s cross-bar switch for SoC. *IEEE International Solid–State Circuits Conference, 1*, 342-472.

Narasimhan, A., & Sridhar, R. (2007). Impact of variabity on clock skew in H-tree clock networks. In *Proceedings of the 8th International Symposium on Quality Electronic Design.*

Nassif, S. (2000). Delay variability: Sources, Impact and Trends. In. *Proceedings of the International Solid State Circuits Conference*, *2000*, 368–369.

Nemes, M. (1984). Driving large capacitances in MOS LSI systems. *IEEE Journal of Solid-state Circuits*, *19*(1), 159–161. doi:10.1109/JSSC.1984.1052103

Neuenhahn, M. C., Blume, H., & Noll, T. G. (2008). Quantitative design space exploration of routing-switches for network-on-chip. *Advances in Radio Science: an Open Access Journal of the U.R.S.I., 6*, 145-150.

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

Nielson, F., Nielson, R. H., & Hankin, C. (2005). Principles of program analysis. In F. Nielson, R. H. Nielson, & C. Hankin (Eds.), *Principles of program analysis* (pp. 365-392). Berlin, Germany: Springer Verlag.

Nieuwland, A., Kang, J., Gangwal, O. P., Sethuraman, R., Busá, N., & Goossens, K. (2004). C-HEAP: A heterogeneous multi-processor architecture template and scalable and flexible protocol for the design of embedded signal processing systems. *Design Automation for Embedded Systems*, 233–270.

Nikolov, H., Stefanov, T., & Deprettere, E. (2008). Systematic and automated multiprocessor system design, programming and implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 542–555. doi:10.1109/TCAD.2007.911337

Nilsson, A., Tell, E., & Liu, D. (2008). An 11mm^2 70mW fully programmable baseband processor for mobile WiMAX and DVB-T/H in 0.12μm CMOS. In *Proceedings of the IEEE International Solid State Circuits Conference* (pp. 266-268).

Nilsson, A., Tell, E., & Liu, D. (2009). An 11 mm^2 70 mW Fully-Programmable Baseband Processor for Mobile WiMAX and DVBT/H in 0.12μm CMOS. *IEEE Journal of Solid-state Circuits*, *44*(1), 90–97. doi:10.1109/JSSC.2008.2007167

NIST. (2001). *FIPS Publication 197: Advanced Encryption Standard (AES), National Institute of Standards and Technology (NIST)*. Retrieved October 31, 2009, from http://csrc.nist.gov/publications/PubsFIPS.html

Northwest Logic. (2009). *DDR SDRAM Controller Core, Product Specification*. Retrieved November 14, 2009, from http://www.nwlogic.com/docs/DDR_SDRAM_Controller_Core.pdf

NoTA. (2009). *Network on terminal architecture, NoTA world open architecture initiative*. Retrieved April 2, 2009, from http://www.notaworld.org/

Nurmi, J. (2008, July 23-27). Silicon Café: A Heterogeneous Multi-Processor Platform Based on Coffee (RISC Core), invited mini-keynote. In *Proceedings of the 8th International Forum on Application-Specific Multi-Processor SoC (MPSoC'08)*, Valkenburg, The Netherlands.

Nurmi, J. (Ed.). (2007). *Processor design - System-on-chip computing for ASICs and FPGAs*. Dordrecht, The Netherlands: Springer.

Nurmi, J., Tenhunen, H., Isoaho, J., & Jantsch, A. (Eds.). (2004). *Interconnect-centric design for advanced SoC and NoC*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

NVIDIA. (2009). *NVIDIA CUDA reference manual version 2.2*. Santa Clara, CA: Author.

Object Management Group. (2005). *UML profile for schedulability, performance, and time specification, version 1.1.* Retrieved April 27, 2009, from http://www.omg.org/docs/formal/05-01-02.pdf

OCP. (2009). *OCP-IP*. Retrieved from http://www.ocpip.org

Ojail, M., David, R., Chevobbe, S., & Demigny, D. (2009). A reconfigurable FIR/FFT unit for wireless telecommunication systems. In *Proceedings of the International Conference on, Field Programmable Logic and Application (FPL'09)* (pp. 645-648). Washington, DC: IEEE.

OpenCores. (2009). *OpenCores DDR SDRAM Controller Core Project*. Retrieved November 14, 2009, from http://www.opencores.org/project,ddr_sdr

Ost, L., Möller, L., Indrusiak, L. S., Moraes, F., Määttä, S., Nurmi, J., et al. (2008). A simplified executable model to evaluate latency and throughput of networks-on-chip. In *Proceedings of the Symposium on Integrated Circuits and Systems Design* (pp. 170-175).

Palma, J. C. S., Indrusiak, L. S., Moraes, F. G., Ortiz, A. G., Glesner, M., & Reis, R. (2006). Adaptive coding in networks-on-chip: Transition activity reduction versus power overhead of the Codec Circuitry. In Proceedings of *PATMOS* (pp. 603-613).

Pamunuwa, D., & Tenhunen, H. (2001). Repeater insertion to minimise delay in coupled interconnects. In *Proceedings of the 14th International Conference on VLSI Design.*

Pande, P. P., Grecu, C., Jones, M., Ivanov, A., & Saleh, R. (2005). Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Computer*, *54*(8), 1025–1040. doi:10.1109/TC.2005.134

Parkvall, S., Dahlman, E., Furuskar, A., Jading, Y., Olsson, M., Wanstedt, S., & Zangi, K. (2008). LTE-Advanced - Evolving LTE towards IMT-Advanced. In *Proceedings of the 68th IEEE International Vehicular Technology Conference (VTC 2008)* (pp. 1-5). Washington, DC: IEEE.

Parunak, H. V. D. (1998). *Practical and industrial applications of agent-based systems (Tech. Rep.)*. Industrial Technology Institute.

Pasricha, S., Ben–Romdhane, M., & Dutt, N. D. (2007). BMSYN: Bus matrix communication architecture synthesis for MPSoC. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *26*(8), 1454–1464. doi:10.1109/TCAD.2007.891376

Patel, H. D., & Shukla, S. K. (2004). *SystemC kernel extension for heterogeneous system modeling – a framework for multi-MoC modeling & simulation*. Norwell, MA: Kluwer Academic Publishers.

Patel, K., & Parameswaran, S. (2008). LOCS: A low overhead profiler-driven design flow for security of MPSoCs. In *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (pp. 79-84).

Paul, J. M. (2003). Programmers' views of SoCs. In *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis,* Newport Beach, CA (pp. 156-181). New York: ACM Publishing.

Paulin, P. G., Pilkington, C., Langevin, M., Bensoudane, E., & Nicolescu, G. (2004). Parallel programming models for a multi-processor SoC platform applied to high-speed traffic management. In *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis,* Stockholm, Sweden (pp. 48-53). New York: ACM Publishing.

PCI-SIG. (2008). *PCI specifications documents, peripheral component interconnect (PCI) standard*. Retrieved April 2, 2009, from http://members.datafast.net.au/dft0802/specs.htm and http://www.pcisig.com/specifications/ordering_information

PCI-SIG. (2009). *PCI express - computer expansion card interface*. Retrieved April 2, 2009, from http://www.pci-sig.com/members/downloads/specifications/pciexpress/PCI_Express_Base_Rev_2.0_20Dec06_cb2.pdf

Peeters, A. (1996). *Single–Rail Handshake Circuits*. Unpublished PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands. Retrieved from http://www.win.tue.nl/async-bib/pdf/Peeters96.pdf

Pelz, E., & Tutsch, D. (2007). Formal Models for Multicast Traffic in Network on Chip Architectures with Compositional High–Level Petri Nets. In J. Kleijn & A. Yakovlev (Eds.), *Proceedings of ICATPN 2007* (LNCS 4546, pp. 381-401).

Perry, D. L., & Foster, H. D. (2005). *Applied formal verification*. New York: McGraw-Hill.

Petre, L. (2008). Energy-Aware Middleware. In D. Bustard & R. Sterritt (Eds.), *Proceedings of the 15th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'08)* (pp. 326-334). Washington, DC: IEEE Computer Society Press.

Petre, L., Sere, K., & Waldén, M. (1999). A topological approach to distributed computing. In G. Stefanescu (Ed.), *Proceedings of the Workshop on Distributed Systems (WDS 99)* (ENTCS 28, pp. 97-118). Maryland Heights, MO: Elsevier Science.

Petre, L., Sere, K., & Waldén, M. (2006) A language for modeling network availability. In *Proceedings of the 8th International Conference on Formal Engineering Methods (ICFEM'2006)* (LNCS 4260, pp. 639-659). New York: Springer Verlag.

Philips Semiconductors. (2000). *I2C-bus specification version 2.1*. Retrieved April 2, 2009, from http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

Piementel, A. D., Hertzberger, L. O., Lieverse, P., van der Wolf, P., & Deprettere, F. E. (2001). Exploring embedded-systems architectures with Artemis. *IEEE Computer,* 57-63.

Plosila, J., Seceleanu, T., & Sere, K. (2004). Formal Communication Modeling and Refinement. In Nurmi, J., Tenhunen, H., Isoaho, J., & Jantsch, A. (Eds.), *Interconnect-Centric Design for Advanced SoC and NoC. Alphen aan den Rijn*. The Netherlands: Kluwer.

Plosila, J., Sere, K., & Waldén, M. (2005). Asynchronous System Synthesis. *Science of Computer Programming*, *55*, 259–288. doi:10.1016/j.scico.2004.05.018

Potlapally, N., Ravi, S., Raghunathan, A., & Jha, N. K. (2003). Analyzing the energy consumption of cryptographic protocols. In *Proceedings of the 2003 international symposium on Low power electronics and design (ISLPED'03)* (pp 30-35). New York: ACM.

Power Architecture Version 2.02. (2010). Retrieved January 28, 2010, from http://www-106.ibm.com/developerworks/eserver/library/es-archguide-v2.html

Preneel, B., & Tavares, S. (Eds.). (2005). *Selected Areas in Cryptography - SAC* (pp. 319–331). Heidelberg, Germany: Springer.

Rabaey, J. M. (2007). *Scaling the Power Wall: Revisiting the Low-Power Design Rules*. Paper presented at the Keynote speech at SoC 07 Symposium, Tampere, Finland.

Rajesh, V., & Moona, R. (1999). Processor modeling for hardware software codesign. In *Proceedings of the 12th International Conference on VLSI Design* (pp. 132137). Washington, DC: IEEE.

Rakhmatov, D., & Vrudhula, S. (2003). Energy management for battery-powered embedded systems. *Transactions on Embedded Computing Systems*, *2*(3), 277–324. doi:10.1145/860176.860179

Rantala, P., Isoaho, J., & Tenhunen, H. (2007). Novel agent-based management for fault-tolerance in network-on-chip. In *Proceedings of 10th EUROMICRO Conference on Digital System Design* (pp. 551-555).

Razavi, N., & Sirjani, M. (2006). Using Reo for Formal Specification and Verification of System Designs. In *Proceedings of Fourth IEEE/ACM International Conference on Formal Methods and Models for Co–Design*.

Riccobene, E., Scandurra, P., Rosti, A., & Bocchio, S. (2005). A SoC design methodology involving a UML 2.0 profile for SystemC. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 704-709).

Ristau, B., Limberg, T., & Fettweis, G. (2008). A mapping framework for guided design space Exploration of heterogeneous MP-SoCs. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* (pp. 780-783).

Rivaton, A., Quevremont, J., Zhang, Q., Wolkotte, P., & Smit, G. (2005). Implementing Non Power-of-Two FFTs on Coarse-Grain Reconfigurable Architectures. In *Proceedings of the International Symposium on System-on-Chip (SoC'05)* (pp. 82-85). Washington, DC: IEEE.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120–126. doi:10.1145/359340.359342

Roan, R. L. (2006). Optimizing Architectures for Performance and Area Using Virtual System Prototypes. *Embedded Technology Journal*. Retrieved October 31, 2009, from http://www.embeddedtechjournal.com/articles_2006/20061107_vast.htm

ROHC. (2003). *A Free Implementation of ROHC – Robust Header Compression*. Retrieved October 31, 2009, from http://rohc.sourceforge.net

Roman, C.-G., Julien, C., & Payton, J. (2004). A formal treatment of context-awareness. In M. Wermelinger & T. Margaria-Steffen (Eds.), *Proceedings of the 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004)* (LNCS 2984, pp. 12-36). New York: Springer Verlag.

Rouff, C. A., Rash, J. L., Hinchey, M. G., & Truszkowski, W. F. (2006). Formal Methods at NASA Goddard Space Flight Center. In Rouff, C. A., Hinchey, M. G., Rash, J. L., Truszkowski, W. F., & Gordon-Spears, D. (Eds.), *Agent Technology from a Formal Perspective* (pp. 287–309). London: Springer. doi:10.1007/1-84628-271-3_10

Royal Academy of Engineering. (2009). *Engineering Values in IT*. London: The Royal Academy of Engineering, the Institution of Engineering and Technology and the British Computer Society. Retrieved August 1, 2009 from http://www.raeng.org.uk/news/publications/list/reports/Engineering_values_in_IT.pdf

Roychoudhury, A., Mitra, T., & Karri, S. R. (2003). Using Formal Techniques to Debug the AMBA System–on–Chip Bus Protocol. In *Proceedings of the Conference on Design, Automation and Test in Europe* (Vol. 1, pp. 10828-10834).

Rumbaugh, J., Jakobson, I., & Booch, G. (1998). *The Unified Modelling Language Reference Manual*. Reading, MA: Addison-Wesley.

Russel, S., & Norvig, P. (2003). *Artificial intelligence - a modern approach* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall.

Russell, J. T., & Jacome, M. F. (1998). Software power estimation and optimization for high performance, 32-bit embedded processors. In *Proceddings of the International Conference pn Computer Design (ICCD'98)* (pp. 328-333).

Saint-Jean, N., Benoit, P., Sassatelli, G., Torres, L., & Robert, M. (2007). Application case studies on HS-scale, a MP-SOC for embbeded systems. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS)* (pp. 88-95).

Sangiovanni-Vincentelli, A. (1993). Some considerations on Field Programmable Gate Arrays and their impact on system design. In *Field-Programmable Gate Arrays: Architecture and Tools for Rapid Prototyping* (LNCS 705/1993, pp. 26-34). New York: Springer. ISSN 0302-9743 ISBN 978-3-540-57091-2

Saponara, S., Fanucci, L., Marsi, S., & Ramponi, G. (2007). Algorithmic and architectural design for real-time and power-efficient Retinex image/video processing. *Journal of Real-Time Image Processing*, *1*(4), 267283. doi:10.1007/s11554-007-0027-z

Sarmento, A., Kiraa, L., Arnaud, G., Youssef, M.-W., Bouchhima, A., Rousseau, F., et al. (2005). Service dependency graph, an efficient model for hardware/software interfaces modeling and generation for SoC design. In *Proceedings of the Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '05,* Jersey City, NJ (pp. 261-266). Washington, DC: IEEE Computer Society.

Sartori, J., & Kumar, R. (2009). Distributed peak power management for many-core architectures. In *Proceedings Of the Design* (pp. 1556–1559). Automation & Test in Europe.

SATA-IO. (2009). *Serial ATA-IO: Enabling the future*. Retrieved April 2, 2009, from http://www.serialata.org

Schliebusch, O., Meyr, H., & Leupers, R. (2007). *Optimized ASIP Synthesis from Architecture Description Language Models*. New York: Springer.

Schneider, S. (2000). *Concurrent and real-time systems: The CSP approach*. Chichester, UK: John Wiley & Sons.

Schwerdtfeger, M. (2000). *SPI - Serial Peripheral Interface*. Retrieved April 2, 2009, from http://www.mct.net/faq/spi.html

Sekerinski, E., & Sere, K. (1999) (Eds). *Program development by refinement: Case studies using the B method*. London: Springer-Verlag.

Sekerinski, E., & Sere, K. (1996). A Theory of Prioritizing Composition. *Oxford Journals -. The Computer Journal*, *39*(8), 701–712. doi:10.1093/comjnl/39.8.701

Senthil, G., Gudla, S., & Baruah, P. K. (2008). Exploring software cache on the Cell BE processor. In *High Performance Computing* (p. 5).

Seo, S., Lee, J., & Sura, Z. (2009). Design and implementation of software-managed caches for multicores with local memory. In *High Performance Computer Architecture* (pp. 55-66). DOI:10.1109/HPCA.2009.4798237

Seth, A., Keskar, R. B., & Venugopal, R. (2001). Algorithms for energy optimization using processor instructions. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*.

Shahbahrami, A., Borodin, D., & Juurlink, B. (2008). *Comparison between color and texture features for image retrieval*. Circuits, Systems and Signal Processing.

Shang, L., Peh, L.-S., & Jha, N. K. (2003), Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proceedings of the 9th international Symposium on High-Performance Computer Architecture*.Washington, DC: IEEE Computer Society.

Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, NJ: Prentice Hall.

Sher, M., & Magedanz, T. (2007). 3G-WLAN convergence: Vulnerability, attacks, possibilities and security model. In *Proceedings of the 2nd International Conference on Availability, Reliability and Security* (pp. 198-205).

Shin, M. C., & Park, I. C. (2007). SIMD processor-based Turbo decoder supporting multiple third-generation wireless standards. *IEEE Transaction on VLSI*, *15*, 801–810. doi:10.1109/TVLSI.2007.899237

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, *60*(1), 51–92. doi:10.1016/0004-3702(93)90034-9

Silven, O., & Jyrkkä, K. (2007). Observations on Power-Efficiency Trends in Mobile Communication Devices. *EURASIP Journal on Embedded Systems,* 8-17.

Skoglund, B. (2007). *Code profiling as a design tool for application specific instruction sets*. Unpublished master's thesis, Linköping University, Sweden.

Smyth, N., McLoone, M., & McCanny, J. V. (2006). WLAN security processor. *IEEE Transactions on Circuits and Systems*, *53*(7), 1506–1520. doi:10.1109/TCSI.2006.877888

Snook, C., & Butler, M. (2004). *U2B - A tool for translating UML-B models into B. UML-B Specification for Proven Embedded System Design*. New York: Springer Verlag.

Sonics. (2006). *Micronetwork architecture*. Retrieved from http://www.sonicsinc.com

Soteriou, V., & Peh, L.-S. (2007). Exploring the Design Space of Self-Regulating Power-Aware On/Off Interconnection Networks. *IEEE Transactions on Parallel and Distributed Systems*, *18*(3), 393–408. doi:10.1109/TPDS.2007.43

Speth, M., Fechtel, S. A., Fock, G., & Meyr, H. (2001). Optimum receiver design for OFDM-based broadband transmission - part II: a case study. *IEEE Transactions on Communications*, *49*(4), 571–578. doi:10.1109/26.917759

Stallings, W. (1996). *Computer organization and architecture - designing for performance*. Upper Saddle River, NJ: Prentice-Hall.

Stankovic, J. A., Burns, A., Jeffay, K., Jones, M., Koob, M., & Lee, I. (1996). Strategic directions in real-time and embedded systems. *ACM Computing Surveys*, *28*(4), 751–763. doi:10.1145/242223.242291

Stubblefield, A., Ioannidis, J., & Rubin, A. D. (2002). Using the Mantin, Shamir and Fluhrer Attack to break WEP. In *Proceedings of the 2002 Network and Distributed Systems Security Symposium* (pp. 17-22).

Sylvester, D., Blaauw, D., & Karl, E. (2006). ElastIC: An Adaptive Self-Healing Architecture for Unpredictable Silicon. *IEEE Design & Test of Computers*, *23*(6), 484–490. doi:10.1109/MDT.2006.145

Szczesny, D., Hessel, S., Bruns, F., & Bilgic, A. (2009). On-the-fly Hardware Acceleration for Protocol Stack Processing in Next Generation Mobile Devices. In *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '09)* (pp. 155-162). New York: ACM.

Tanenbaum, A. (1999). *Structured computer organization* (4th ed.). Upper Saddle River, NJ: Prentice-Hall.

Tanenbaum, A. S. (2003). *Computer Networks* (4th ed.). Upper Saddle River, NJ: Prentice Hall/Pearson.

Target Compiler Technologies. (2009). *IP Designer*. Retrieved November 14, 2009, from http://www.retarget.com/products/ipdesigner.php

Tartanoglu, F., Issarny, V., Levy, N., & Romanovsky, A. (2003). Formalizing dependability mechanisms in B. In *Proceedings of the ICSE'2003 Workshop on Architecting Dependable Systems.*

Tavakkol, A., Moraveji, R., & Sarbazi–Azad, H. (2008). Mesh Connected Crossbars: A Novel NoC Topology with Scalable Communication Bandwidth. In *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications* (pp. 319-326).

The RODIN platform. (n.d.). Retrieved from http://rodin-b-sharp.sourceforge.net/

Tindell, K., Burns, A., & Wellings, A. (1994). An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems Journal*, *6*(2), 133–151. doi:10.1007/BF01088593

Tiri, K. (2007). Side-channel attack pitfalls. In *Proceedings of the 44th Annual Conference on Design Automation* (pp. 15-20).

Tiri, K., & Verbauwhede, I. (2006). A digital design flow for secure integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *25*(7), 1197–1208. doi:10.1109/TCAD.2005.855939

Todor, S., Zissulescu, C., Turjan, A., Kienhuis, B., & Deprettere, E. (2004). System design using Khan process networks: The Compaan/Laura approach. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2004* (pp. 340-345).

Treharne, H., Schneider, S., & Bramble, M. (2003). *Composing Specifications Using Communication. (2003). Proceedings of ZB 2003: Formal Specification and Development in Z and B* (LNCS 2651). New York: Springer Verlag.

Tretter, S. A. (2003). *Communication system design using DSP algorithms: With laboratory experiments for the TMS320C6701 and TMS320C6711*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Tsiopoulos, L. (2009). Towards Dependable Placement of NoC Resources. In *Proceedings of NODES '09 workshop* (pp. 1-9).

Tsiopoulos, L., Sere, K., & Plosila, J. (2006). *Modeling SoC Systems with Flow Control* (Tech. Rep. No. 746). Turku, Finland: Turku Centre for Computer Science (TUCS).

Tsiopoulos, L., & Satpathy, M. (2009). Model Based Testing of a Network–on–Chip Component. [ENTCS]. *Electronic Notes in Theoretical Computer Science*, *253*(2), 101–116. doi:10.1016/j.entcs.2009.09.054

Tsiopoulos, L., & Waldén, M. (2006). Formal Development of NoC Systems in B. *Nordic Journal of Computing*, *13*(1-2), 127–145.

TSMC. (2001). *TSMC 0.18μm process 1.8 volt in SAGE-X™ standard cell library*. Artisan components, Inc.

Tuominen, J., Westerlund, T., & Plosila, J. (2008). Power aware system refinement. *Electronic Notes in Theoretical Computer Science*, *201*, 223–253. doi:10.1016/j.entcs.2008.02.022

Tuuna, S., Nigussie, E., Isoaho, J., & Tenhunen, H. (2008). Analysis of delay variation in encoded on-chip bus signaling under process variation. In *Proceedings of the 21st IEEE International Conference on VLSI Design* (pp. 228-234).

Tuuna, S., Zheng, L.-R., Isoaho, J., & Tenhunen, H. (2008). Modeling of on-chip bus switching current and its impact on noise in power supply grid. *IEEE Transactions on Very Large Scale Integration (VLSI). Systems*, *16*(6), 766–770.

Unsal, O. S., Koren, I., & Krishna, C. M. (2004). Towards energy-aware software-based fault tolerance in real time systems. In *Proceedings of the 2002 international symposium on Low power electronics and design* (pp. 124-129).

USB. (2009). *USB implementers forum, Universal Serial Bus (USB) 2.0 specification*. Retrieved April 2, 2009, from http://www.usb.org/developers/docs/

van Berkel, C. H. (2009). Multi-Core for Mobile Phones. In *Proceedings of the Design, Automation & Test in Europe Conference (DATE '09)* (pp. 1260-1265). Washington, DC: IEEE.

van der Wolf, P., de Kock, E., Henrikson, T., Kruijtzer, W., & Essink, G. (2004). Design and programming of embedded multiprocessors: An interface-centric approach. In *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis,* Stockholm, Sweden (pp. 206-217). New York: ACM Publishing.

van Ommering, R., van der Linden, F., Kramer, J., & Magee, J. (2000). The Koala component model for consumer electronics software. *IEEE Computer*, *33*, 78–85.

Vangal, S. R., Howard, J., Ruhl, G., Dighe, S., Wilson, H., & Tschanz, J. (2008). An 80-Tile Sub-100-W TeraFLOPS Processor in 65nm CMOS. *IEEE Journal of Solid-state Circuits*, *43*(1), 29–41. doi:10.1109/JSSC.2007.910957

Vercauteren, F. (2008). Optimal pairings. *Cryptology ePrint Archive, Report 2008/096.* Retrieved November 20, 2009, from http://eprint.iacr.org/2008/096

Vishvanathan, V., Ravikumar, C. P., & Menezes, V. (2005). Design technology challenges in the Sub-100 nanometer era. *Periodical of the VLSI society of India-VLSI Vision, 1*(1).

Waldén, M., & Sere, K. (1998). Reasoning about Action Systems Using the B Method. *Formal Methods in System Design*, *13*(1), 5–35. doi:10.1023/A:1008688421367

Wang, D., & Cao, J. (2003). On optimal hierarchical configuration of distributed systems on mesh and hypercube. In *Proceedings of the International Parallel and Distributed Processing Symposium.* Washington, DC: IEEE Computer Society.

Woh, M., et al. (2008). From SODA to scotch: The evolution of a wireless baseband processor. In *Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture*.

Wolf, W. H. (1994). Hardware-software co-design of embedded systems. *Proceedings of the IEEE, 82(7*), 967-989.

Wu, D., Eilert, J., & Liu, D. (2009). Evaluation of MIMO Symbol Detectors for 3GPP LTE Terminals. In *Proceedings of EUSIPCO 2009*.

Wu, D., Larsson, E. G., & Liu, D. (2009, April). Implementation Aspects of Fixed-Complexity Soft-Output MIMO Detection. In *Proceedings of IEEE VTCSpring*.

Xilinx. (2009). *Xilinx Platform Studio and Embedded Development kit*. Retrieved May 2009, from http://www.xilinx.com/tools/embedded.htm

Yadav, D., & Butler, M. (2005). Application of Event B to Global Causal Ordering for Fault Tolerant Transactions. In *Proceedings of Workshop on Rigorous Engineering of Fault Tolerant Systems (REFT'2005)* (pp. 93-102).

Yan, L., & Sere, K. (2004). A formalism for context-aware mobile computing. In *Proceedings of the Third International Symposium in Parallel and Distributed Computing/ International Workshop on Algorithms, Models and Tools for Parallel Computing and Heterogenous Networks (ISPDC/HeteroPar'04)* (pp. 14-21). Washington, DC: IEEE Computer Society Press.

Yap, L. F., & Jones, A. (2007). Profiling through a digital mobile device. In *Proceedings of the 5th Australian Digital Forensics Conference* (pp. 52-58).

Yin, A. W., Guang, L., Liljeberg, P., Rantala, P., Nigussie, E., Isoaho, J., et al. (2008). Hierarchical Agent Architecture for Scalable NoC Design with Online Monitoring Services. In *Proceedins of the First International Workshop on Network on Chip Architectures* (pp. 58-63).

Yin, A. W., Guang, L., Nigussie, E., Liljeberg, P., & Tenhunen, H. (2009). Architectural Exploration of Per-Core DVFS for Energy-Constrained On-Chip Networks. In *Proceedings of the Euromicro DSD (Digital System Design) conference 2009,* Los Alamitos, CA (pp. 141-146). Washington, DC: IEEE Computer Society.

Yoon, S., & Bar-Ness, Y. (2002). A Parallel MAP Algorithm for Low Latency Turbo Decoding. *IEEE Communications Letters*, *6*, 288–290. doi:10.1109/LCOMM.2002.801310

Zatt, B., Azevedo, A., Agostini, L., Susin, A., & Bampi, S. (2007). Memory hierarchy targeting bi-predictive motion compensation for H.264/AVC decoder. In *VLSI* (pp. 445-446). Washington, DC: IEEE Computer Society. DOI:10.1109/ISVLSI.2007.64

Zhang, Q., Kokkeler, A., & Smit, G. (2007). An efficient FFT for OFDM based cognitive radio on a Reconfigurable Architecture. In *Proceedings of the International Conference on Communications (ICC'07)* (pp. 6522-6526). Washington, DC: IEEE.

Zheng, X., Liu, Z., & Peng, B. (2008). Design and implementation of an ultra low power RSA coprocessor. In *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing* (pp. 1-5).

Zheng, L.-R., & Tenhunen, H. (2001). Fast modeling of core switching noise on distributed LRC power grid in ULSI circuits. *IEEE Transactions on Advanced Packaging*, *24*(3), 245–254. doi:10.1109/6040.938290

Zhu, D., Melhem, R., & Mosse, D. (2004). The effect of energy management on reliability in real-time embedded systems. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer Aided Design (ICCAD'04)* (pp. 35-40). Washington, DC: IEEE Computer Society.

Zimmermann, E., Fettweis, G., Milliner, D. L., & Barry, J. R. (2008, January). Parallel Smart Candidate Adding Algorithm for Soft-output MIMO Detection. In *Proceedings of the International Conference on Source and Channel Coding*, Ulm, Germany.

About the Contributors

Seppo Virtanen received his BSc in Applied Physics, MSc in Electronics and Information Technology (1998), and DSc (Tech.) in Communication Systems (2004) from the University of Turku (Finland). Since 2009 he has been Adjunct Professor of Embedded Communication Systems in University of Turku. He is Editor-in-Chief of *International Journal of Embedded and Real-Time Communication Systems* (IJERTCS) and Senior Member of the IEEE. His current research interests include the design and methodological aspects of reliable and secure systems.

* * *

Alan Burns is a Professor of Real-Time Systems in the Department of Computer Science, University of York, U.K. He graduated in Mathematics from the University of Sheffield in 1974, undertook his PhD at the University of York before taking up a tenured post at the University of Bradford. He joined the University of York again in 1990 and was promoted to a personal chair in 1994. Together with Professor Andy Wellings he founded and heads the Real-Time Systems Research Group at the university; a group that has currently has over 35 members. He has served as Chair of the IEEE Technical Committee on Real-Time Systems and has published widely (over 350 papers and articles, and 10 books). In 2009 he was elected a Fellow of the Royal Academy of Engineering, UK.

Anders Nilsson is principal system architect and co-founder of Coresonic Ltd. He received his M.S. in applied physics and electrical engineering and his Ph.D. in Computer Engineering both from Linköping University, Sweden. His research interests include high speed mobile wireless links, radio technology and baseband processor design; he has three US patent applications, and is the co-author of books 'Radio Design in Nanometer Technologies' and 'Handbook of WiMAX'.

Anupam Chattopadhyay completed his Bachelor's in Electronics and Telecommunications Engineering in Jadavpur University, Kolkata, India in 2000. Following a brief industrial stint, Dr.-Ing. Chattopadhyay undertook Master's studies on Embedded System Design at ALaRI, USI, Switzerland. From 2002 to 2008, he pursued doctoral studies at ISS, RWTH Aachen University, Germany. He obtained his doctorate degree with summa-cum-laude from RWTH Aachen University in 2008. He spent nearly 2 years after his PhD at CoWare India R&D. His major research interests are embedded processor design automation and partially reconfigurable processors. In this field, his works are currently in commercial use and academically well-received in major conferences.

Bastian Bauwens received the Diploma degree in electrical engineering from the Institute for Integrated Signal Processing Systems, RWTH Aachen University, Aachen, Germany. His research interests include embedded processor design automation. Now he is with McKinsey & Company.

Dake Liu is IEEE senior member, professor and the director of computer engineering at Linköping University. He received his PhD in 1995 from Linköping University, Sweden. His research interests include application-specific instruction set processors (ASIP) and on-chip multi-core integrations, SDR platforms for future mobile terminals and radio base stations, and low-power electronics. He was a cofounder, and CTO of Freehand DSP Ltd. He is the cofounder and CTO of Coresonic Ltd. He was a senior ASIC designer in Ericsson Sweden 1995 to 1998. He is currently professor of embedded systems in Nanjing University, China and the professor of HIT China. He worked for Beijing Jiaotong University, China during 1982-1990.

David Kammler received the Diploma degree in electrical engineering from the Institute for Integrated Signal Processing Systems, RWTH Aachen University, Aachen, Germany, in 2003, where he is currently pursuing the Ph.D. degree in electrical engineering. His research interests include embedded processor design automation and memory architectures for application-specific processors.

Di Wu received his M.Sc. and Ph.D. degree in electrical engineering from Linköping University, Sweden, in 2005 and 2009, respectively. During the Ph.D. study, his research interests included software-defined radio, specifically multi-antenna systems and their flexible VLSI implementations. Currently he is working on modem technologies at ST-Ericsson, Sweden.

Elena Troubitsyna is an Academy Research Fellow at the Academy of Finland. She got her PhD in Computer Science in 2000 on design methods for dependable systems. Her research interests include application of formal methods to development of dependable fault tolerant systems. She also conducts research on combining formal methods with informal techniques of safety analysis and semi-formal design techniques such as UML. She has worked on applying formal methods to development of an industrial fault-tolerant system within EU IST projects MATISSE, RODIN, and DEPLOY.

Eric Tell received his M.Sc. in applied physics electrical engineering in 2001 and Ph.D. in computer engineering in 2005, both from Linköping University, Sweden. He is a co-founder of Coresonic and currently holds the position of principal designer at Coresonic. His professional interests are ASIP architectures, algorithms, and design methodologies for flexible baseband processing.

Erik Alfredsson received his M.Sc. in applied physics electrical engineering in 2005 from Linköping University, Sweden. His is currently working at Coresonic AB as a research and development engineer.

Ernst Martin Witte received his Diploma degree in electrical engineering from the Institute for Integrated Signal Processing Systems, RWTH Aachen University, Aachen, Germany, in 2004, where he is currently pursuing the Ph.D. degree. His research focuses on exploration of hardware architectures for wireless receivers with special respect to the fields of multi-antenna signal processing systems and software defined radio.

Fabio Garzia received the Master of Science degree in Microelectronics in 2005 from the University of Bologna, Italy, and the Doctor of Science degree in Technology in 2009 from Tampere University of Technology, Finland. From March to December 2005 he has been working at the "Advanced Research Center on Electronic Systems E. DeCastro, (ARCES)", Bologna. In 2006 he joined the research group of Prof. Jari Nurmi at the Tampere University of Technology, where he carried out his PhD studies and where he is currently holding a position of Senior Researcher. His main research interests are design of System on chip, reconfigurable architectures and platforms for 3D graphics, multimedia and SDR applications.

Fernando Gehm Moraes received the electrical engineering and MSc degrees from the Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1987 and 1990, respectively. In 1994 he received the PhD degree from the Laboratoire d´Informatique, Robotique et Microélectronique de Montpellier (LIRMM), France. He is currently at PUCRS, where he has been an associate professor from 1996 to 2002, and professor since 2002. He has authored and co-authored twelve peer refereed journal articles in the field of VLSI design, comprising the development of networks on chip and telecommunication circuits. He has also authored and co-authored more than 120 conference papers on these topics. He has co-advised 3 MSc, advised 13 MsC, co-advised 3 PhD and advised 1 PhD works. Now he is responsible for the computer science graduate program at PUCRS. His research interests include microelectronics, FPGAs, reconfigurable architectures, and NoCs (networks on chip).

Gerd Ascheid received his Diploma and Ph.D. degrees in electrical engineering (communications eng.) from RWTH Aachen University, Aachen, Germany, in 1977 and 1984. In 1988 he started as a co-founder CADIS GmbH. The company has successfully brought the system simulation tool COSSAP to the market. In 1994 CADIS GmbH was acquired by SYNOPSYS, a California-based EDA market leader where his last position was Senior Director (executive management), wireless and broadband communications service line, Synopsys Professional Services. Since April 2003 he is the Head of the Institute for Integrated Signal Processing Systems of the RWTH Aachen University. He has published several scientific papers in reputed journals and conferences. He is also the Chairman of the cluster of excellence in "Ultra-high speed Mobile Information and Communication (UMIC)" at RWTH Aachen University.

Hannu Tenhunen received the Diplomas from Helsinki University of Technology, Finland, in 1982 and the PhD degree in microelectronics from Cornell University, NY, in 1986. In 1985, he joined Signal Processing Laboratory, Tampere University of Technology, Finland, as an Associate Professor and later served as professor and department director. He was also a coordinator of the National Microelectronics Program of Finland during 1987–1991. Since 1992, he has been with Royal Institute of Technology (KTH), Sweden, where he is a Professor of electronic system design. He served as dean for School of Information and Communication Technology at KTH during 2001-2005. Currently he is director of Turku Centre of Computer Science, Turku, Finland and at University of Turku. His current research interests are VLSI architectures and systems, especially Network-on-Chip and autonomous systems. He has authored or coauthored over 515 presentations and publications, and has over 16 patents pending or granted.

Heinrich Meyr received the M.Sc. and Ph.D. degrees from ETH Zurich, Zurich, Switzerland. He spent over 12 years in various research and management positions in industry before accepting a professorship in electrical engineering with the Institute for Integrated Signal Processing Systems, RWTH Aachen University, Aachen, Germany, in 1977. He has published numerous papers and holds many patents. He is also a cofounder of CADIS GmbH (acquired 1993 by Synopsys, Mountain View, California), a company which commercialized the tool suite COSSAP, and of LISATek, an EDA tool provider for embedded processor design, acquired by CoWare Inc. Dr. Meyr was elected as McKay Distinguished Lecturer at the Electrical Engineering Department of the University of California at Berkeley and was also a recipient of the prestigious Vodafone innovation prize in 2000.

Jan Madsen received the MSc degree in electrical engineering and the PhD degree in computer science from the Technical University of Denmark (DTU), in 1986 and 1992, respectively. Since 2002 he has been full professor of computer-based systems with the Department of Informatics and Mathematical Modelling, DTU. He is the head of the section on Embedded Systems Engineering. His research interests are in modelling, analysis and design of embedded systems, particularly system-level tools for performance analysis and verification, hardware/software codesign, and wireless sensor networks. He has published over 80 articles in international journals and conferences, 7 book chapters, and edited 1 book. He has been the program chair and vice chair of the Design Automation, and Test in Europe Conference (DATE), and program chair and general chair for the Hardware/Software Codesign Conference (CODES). He has been a member of the technical program committee and organizing committee of several technical conferences.

Jari Nurmi is a professor of Computer Systems at Tampere University of Technology (TUT). He has held various research, education and management positions at TUT and in the industry since 1987. He got a PhD degree from TUT in 1994. His current research interests include System-on-Chip integration, on-chip communication, embedded and application-specific processor architectures, and circuit implementations of digital communication, positioning and DSP systems. He is leading a group of about 20 researchers. Dr. Nurmi is the general chairman of the annual International Symposium on System-on-Chip (SoC) and its predecessor SoC Seminar in Tampere since 1999, and a board member of SoC, FPL, and NORCHIP conference series. He was also the general chair of FPL 2005 and SiPS 2009 conferences. He was the head of the national TELESOC graduate school 2001-2005. He is the author or co-author of over 200 international papers, editor of Springer book "Processor Design: System-on-Chip Computing for ASICs and FPGAs," co-editor of Kluwer book "Interconnect-centric Design for Advanced SoC and NoC", associate editor of the "International Journal of Embedded and Real-Time Communication Systems", and has supervised over 100 MSc and Licentiate theses and 12 Doctoral theses. He is a senior member in IEEE Circuits and Systems Society, Computer Society, Signal Processing Society, Solid-State Circuits Society and Communications Society. In 2004, he was one of the recipients of Nokia Educational Award, and the recipient of Tampere Congress Award 2005. He was awarded one of the Academy of Finland Research Fellow grants for 2007-2008.

Jari Nurmi is professor of computer systems at Tampere University of Technology (TUT) since 1999. He has had research, education, and management positions at TUT and industry 1987-1998. He received his PhD degree from TUT in 1994. His current research interests include system-on-chip, network-on-chip, embedded (multi-)processor architectures, and implementations of communication, positioning

and DSP systems. He is leading a group of about 25 researchers. Dr. Nurmi is the general chairman of the international symposium on System-on-Chip (SoC) 1999-2026, board member of SoC, FPL, and NORCHIP. He was the general chair of FPL'05 and SiPS'09 conferences. He was heading TELESOC graduate school 2001-2005. He has (co-)authored over 180 international papers, edited *Processor Design: System-on-Chip Computing for ASICs and FPGAs* (Springer), and co-edited *Interconnect-centric Design for Advanced SoC and NoC* (Kluwer). He is also an associate editor of *IJERTCS*. He has supervised over 100 MSc and 10 Doctoral theses, is a senior member in IEEE, has received Nokia Educational Award 2004, and Tampere Congress Award 2005. He was also Academy of Finland Research Fellow 2007-2008.

Jian Wang is a PhD student in the Department of Electrical Engineering at Linköping University, Sweden. His research interests include memory subsystem design, addressing complexity of SIMDization, vectorization, and on-chip multi-core. He was a VLSI design engineer in Fujitsu Microelectronics (Shanghai) from 2006 to 2008. He got his master degree in Electrical Engineering from Linköping University in 2006, and bachelor degree in Telecommunication Engineering from Beijing University of Posts and Telecommunications in 2004.

Joar Sohl is a PhD student in the Department of Electrical Engineering at Linköping University, Sweden. His research interests include low latency conflict free parallel memory access for SIMDization and on-chip multicore processors. He received his MSc in applied physics and electrical engineering in 2006 from Linköping University. He worked as a consultant for Cybercom from 2006 to 2008, developing simulation software for Ericsson.

Johan Eilert received his M.Sc. in Computer science from Linköping University, Sweden 2003. He jointed in the Division of Computer Engineering of the Department of Electrical Engineering in Linköping in 2003 as a Ph.D. student. His research interests include ASIP architectures and SDR, and custom floating-point datapath.

Jouni Isoaho received his MSc (Tech) in electrical engineering, and his Lic.Tech. and Dr.Tech. in signal processing from Tampere University of Technology, Finland, in 1989, 1992 and 1995, respectively. He was an associate professor in the Royal Institute of Technology, Stockholm, Sweden, until he was invited to the communication systems professorship at the University of Turku, Finland, in 1999. There he heads the communication systems laboratory. His research interests include future communication system concepts, applications and implementation techniques. His special interests relating to embedded systems are dynamically reconfigurable agent based systems for future communication and multimedia applications including information security and dependability aspects.

Juha Plosila received the PhD degree in electronics and information technology from University of Turku, Turku, Finland, in 1999. He is an adjunct professor in digital system design at University of Turku and holds currently the five year position of academy research fellow at the Academy of Finland. He directs, at University of Turku, a research group focusing on modeling, design, and verification of single-chip systems at different abstraction levels. His research interests include formal methods for NoC and embedded system design, reliable on-chip communication methods, on-chip thermal monitoring and control, dynamic reconfiguration, and asynchronous circuit techniques for nanoscale single-chip systems.

Kaisa Sere is a Professor of Computer Science and Engineering at Åbo Akademi University since 1997. She got her PhD in 1990 on the formal design of parallel algorithms from Åbo Akademi University. Between 1993-97 she was Associate Professor in Computer Science at University of Kuopio. She is the founder and leader of the Distributed Systems Laboratory that contains about 25 researchers. Her current research interests are within the design of dependable distributed systems, especially refinement-based approaches to the construction of systems ranging from pure software to hardware and digital circuits. Her research has been supported by the Academy of Finland as well as by the EU framework programmes 5 to 7 with several grants. She has organised several summer schools, conferences and workshops within her research areas. She is the vice chair of the Council of Natural and Engineering Sciences at the Academy of Finland (1.1.2004-31.12.2009).

Leandro Möller is currently a PhD student at the Technical University of Darmstadt, Germany. He is a member of the Microelectronic Systems group in this same University. He received his master and bachelor degree in computer science, in 2005 and 2003, respectively, both from the Catholic University of Rio Grande do Sul, Brazil. Since 2001 he is also a member of the Hardware Design Support Group (GAPH) at Catholic University of Rio Grande do Sul. Further he received two awards, the first place in the Xilinx Student Contest given during the SBCCI conference in 2005 and the Europractice Design Contest Award in the conceptual design category in 2006 given during the DATE conference. In the summer of 2006, he was an intern at Xilinx Research Labs, San Jose, US. His research interests are partial and dynamical reconfiguration of FPGAs, Networks-on-Chip and High Level Modeling of Systems-on-Chip.

Leandro Soares Indrusiak graduated in Electrical Engineering from the Federal University of Santa Maria (UFSM) in 1995 and obtained a MSc in Computer Science from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, in 1998. He held a tenured assistant professorship at the Informatics department of the Catholic University of Rio Grande do Sul (PUCRS) in Uruguaiana from 1998 to 2000. From 2001 to 2008 he worked as a researcher and lead a system-on-chip research group at the Technische Universität Darmstadt, Darmstadt, Germany. His PhD was jointly awarded by UFRGS and TU Darmstadt in 2003. Since 2008, he has held a lectureship in real-time embedded systems at University of York's Computer Science Department. He is also the department's internationalisation advisor. His current research interests include on-chip multiprocessor systems design and validation, application-platform mapping, distributed embedded systems, UML and Java for embedded systems, adaptive and reconfigurable computing.

Leonidas Tsiopoulos received his B.Eng. in Electronic Engineering and Technology Management from the University of Huddersfield, England, in 2001 and his M.Sc. in Software Engineering from Åbo Akademi University, Finland, in 2003. Currently he is a PhD student at Åbo Akademi University. The focus in his research has been on using the component-based design approach to modeling hierarchical, modular and novel communication components for MPSoC/NoC designs. Moreover, implementations within specific hardware programming languages and their model-based testing against verified formal models is of high interest, as well as exploitation of new techniques based on formal methods for efficient and dependable placement of resources on single-chip.

Liang Guang received his B.Eng. from Nanjing University of Aeronautics and Astronautics, China in 2004, and his M.Sc. with a major in electrical engineering from the Royal Institute of Technology, Sweden in 2005. Afterwards, he worked on the topic of parallel computer architecture in the University of Amsterdam, the Netherlands for two years. Now he is pursuing his PhD in the University of Turku, Finland. He has a wide interest in different topics of system-on-chip and network-on-chip design. Currently his research focuses on system-level architectural design, modeling and analysis of adaptive, dependable and power efficient massively parallel on-chip computing platform.

Linas Laibinis is a Senior Researcher at the Department of Information Technologies of Åbo Akademi University, Finland. He got his PhD in Computer Science in 2000 on mechanised formal reasoning about computer programs. His research interests include interactive environments for proof and program construction, as well as application of formal methods to modelling and development of fault tolerant and distributed software systems.

Luciano Ost is currently a PhD student at PUCRS and a member of the Hardware Design Support Group (GAPH) in this same University. Further, he worked one year as invited research at the Microelectronic Systems Institute of the Technische Universitaet Darmstadt. His main research interests are digital systems design, NoCs (networks on chip), and high level modeling of MPSoCs (multi-processor systems on chip). Dr. h. c. mult.

Luigia Petre is a university lecturer at Åbo Akademi University, Department of Information Technologies, Turku, Finland. She got her PhD in Computer Science in 2005 on modeling techniques in formal methods. Her research interests include energy modeling, network availability, metamodeling, UTP, integration of formal methods, time and space dependent computing. She has been one of the organizers and proceedings editors for the Integrated Formal Methods 2002 as well as a programme committee member of this conference in 2002, 2004, 2005, and 2007. Currently, she is coordinating NODES - a Nordic Dependability Network, concerned with deploying a dependability curriculum for the Nordic countries. She has (co-)supervised six MSc theses. She has reviewed about 40 scientific papers for more than 20 journals and conferences. She has created and taught 5 courses and has about 30 publications.

Manfred Glesner was born in Saarlouis/Germany and got his university education at the Saarland University in Saarbrücken/Germany. He studied applied physics and electrical engineering and received his diploma in 1969. In 1975 he received the PhD. degree from the same university for his research work on optimisation techniques in computer aided circuit design. In 1981 he became assistant professor in the Faculty for Electrical Engineering and Information Technology of Darmstadt University of Technology. Since 1989 he owns the chair for Microelectronic Systems at Darmstadt University. For his contributions to the field of microelectronics he has received three honorary causa doctoral degrees, one from Tallinn Technical University (Estonia), one from Bucharest Polytechnical University (Romania), and one from Mongolian Technical University in Ulaan Bator (Mongolia). In 2008, he received the decoration Palmes académiques in the grade of Chevalier.

Michel Gillet is an architect at Nokia Devices in Helsinki, Finland. Gillet has a MSc in computer science from University of Liege, Belgium, 1999, with the title "Ethernet ISDN gateway for VoIP". From 1999 till 2008, he was in the Nokia Research Center with the following research activities: high-speed

serial links, signal integrity, electromagnetic simulations, sensor integration and embedded networks. In 2008, he joined Nokia Devices to drive the productization of embedded network technologies. Since its creation, he is the main Nokia representative in the MIPI UniPro standardization. His research interests include on-chip, off-chip and die-to-die embedded networks, system architecture, system modeling and distributed systems.

Peter Verner Bojsen Sørensen received his MS degree in computer engineering from the Technical University of Denmark in 2006. He is currently working toward the PhD degree in computer engineering at the Department of Informatics and Mathematical Modelling, Technical University of Denmark. From 2001 to 2005 he was a research and development engineer with XO Care A/S, Copenhagen. He has been with the Department of Informatics and Mathematical Modelling since 2006. His research interests include hardware/software codesign and system-level design of embedded systems with emphasis on component-based design approaches.

Rainer Leupers received the M.Sc. and Ph.D. degrees in computer science with honors from the University of Dortmund, Dortmund, Germany, in 1992 and 1997. During 1999-2001 he was also a team leader at ICD, where he headed industrial service projects. He joined RWTH Aachen University, Aachen, Germany, as a Professor in 2002, where he currently heads the Institute for Software for Systems on Silicon. He has published several scientific papers in reputed journals and conferences, and served in the program committees of DAC, DATE, and ICCAD. He has been a cofounder of LISATek, an EDA tool provider for embedded processor design, acquired by CoWare Inc. His research interests include multi-processor SoC design and simulation tools.

Rizwan Asghar received his M.Sc. in Physics from Quaid-i-Azam University, Islamabad, Pakistan, and M.S. in Computer Engineering from Center for Advanced Studies in Engineering, Islamabad, affiliated with University of Engineering and Technology (UET), Texila, Pakistan. He is currently pursuing his Ph.D. degree at the Department of Electrical Engineering, Linköping University, Sweden. His research activity is mainly focused on flexible and re-configurable FEC sub-systems for baseband processors.

Roberto Airoldi received his Bachelor of Science degree in Electronics in 2005 from University of Ferrara (Ferrara, Italy) and his Master of Science degree in Electronics in 2009 from University of Bologna (Bologna, Italy). In 2008 he joined the research group of Prof. Jari Nurmi at Tampere University of Technology as research assistant and since 2009 he is carrying out there his PhD studies in Computer Systems. His main research interests are System-on-Chip, multi-processor architectures, wireless applications and software defined radio.

Sanna Määttä received her MSc degree in information technology from Tampere University of Technology, Finland, in 2004. Currently she is a PhD student at the Department of computer systems at Tampere University of Technology. She works there since 2003, first as a research assistant and after graduation at 2004 as a researcher. In 2007-2008 she has worked one year as a guest researcher at the Institute of Microelectronic Systems at the Technical University of Darmstadt, Germany. Currently she is working in a SYSMODEL project and her main research interests are high level modelling and validation of applications, networks-on-chip, and multiprocessor systems.

Sari Leppänen is a Senior Manager in Operative Architecture Management at Nokia Corporation, Finland. From 1997 until 2007 she has been a Principal Scientist at Nokia Research Center. She got her PhD in Computer Science in 2008 on system and software development of telecommunicating systems.

Sergey Balandin holds MSc in Computer Science from St.-Petersburg Electro Technical University "LETI" (Russia, 1999), the main research subject: "Efficient pattern recognition in highly dynamic environment", secondary subject "Controlling of the robot-centric system"; MSc in Telecommunications from Lappeenranta University of Technology (Finland, 2001), main subject "Efficient IP routing in backbone networks", secondary subject "Network optimization"; and PhD in Telecommunications and Control Theory from Nokia PhD school and St.-Petersburg Electro Technical University "LETI" (2003). Since 1999, Sergey Balandin works in Nokia Research Center, current position - Principle Scientist. His main research interests are: smart spaces, embedded networks, routing and resource management, load balancing in fixed and wireless networks, peer-to-peer mobile and sensor networks. In 2006-2008 Sergey Balandin was Nokia representative in MIPI UniPro standardization. Currently he is Nokia's University cooperation representative in Russian and CIS and the general chair of Finnish-Russian University Cooperation in Telecommunications (FRUCT) program.

Zheng Shi is currently a PhD candidate in the Department of Computer Science at the University of York in the UK. He received the BSc degree (1998 ~ 2002) in the Department of Computer Science, Zheng Zhou University, Zheng Zhou, China and MSc degree (2002 ~ 2005) in Department of Computer Science and Engineering, Beijing Institute of Technology, Beijing, China. His current research interests are in the embedded and real-time system area, including real-time communication network, real time model abstract and schedulability analysis, network-on-chip and quality of service guarantee.

Index

256-bit Barreto-Naehrig curves 96
3rd Generation Partnership Project (3GPP) 122, 139

A

Abstract Machine Notation (AMN) 303
action systems 45, 49, 221
Adaptive Logic Modules (ALM) 161
Adaptive Look-Up Tables (ALUT) 161
Address Generation Units (AGU) 111
Advanced Encryption Standard (AES) 19, 124, 139
Analog Front-End (AFE) 105
application modeling 263, 265, 275
Application Programming Interface (API) 61, 66, 242
Application-Specific Instruction-set Processors (ASIP) 79
Application-Specific Integrated Circuit (ASIC) 103, 157
Architectural Modeling Programming Interface (AMPI) 125
Architecture Description Language (ADL)
    mixed ADLs 81
    behavioral ADLs 81
    structural ADLs 81
Architecture Description Languages (ADL) 79
ARM1176 embedded processor 124
asynchronous communication 220
asynchronous communication handshake 224
asynchronous connectors 227
asynchronous pull channel 225-226
asynchronous push channel 225

B

Behavioral Address Coding Template (BACT) 37
Bit Error Rate (BER) 147, 151
B method 298, 302
Broadcast Channel (BCCH) 129
butterfly fat–tree topology 237

C

Category four User Equipment (CAT4 UE) 102
cell processor 60-61, 77
Channel Access Functions (CAF) 247-248, 257
channel access protocol 247
channel buffer 247-249, 253
channel control variables 247
Channel Preprocessor Unit (ChPU) 112
Channel Quality Indication (CQI) 109
Code Division Multiple Access (CDMA) 103
Common Control Channel (CCCH) 129
Communicating Sequential Processes (CSP) 239, 265
communication service 198
communication structures 178, 180, 191
competing relationship
    direct 203
    indirect 203
Complex-valued Multiply-and-ACcumulate (CMAC) 112
component allocation 242
component-based design 246
components and agents 286
computation unit 45
    resource 46
conflict free memory access 39
cryptographic pairings 95-96, 98-99
cryptosystems 5

D

data security 5
Dedicated Control Channel (DCCH) 130
Dedicated Traffic Channel (DTCH) 126
Deep Sub-Micron (DSM) 178, 183, 195
design-space exploration 80, 93, 96
design verification 15
Device Interconnect Protocol (DIP) 148

device interoperability 4
Dhrystone benchmark application 128, 132
Diagnostic and Adaptivity Processing (DAP) 279
Differential Power Analysis (DPA) 17
Digital BaseBand (DBB) 105, 109, 123
Digital Front-End (DFE) 105, 109
Digital Signal Processor (DSP) 157
Digital Video Broadcasting (DVB) 106, 158
Direct Memory Access (DMA) 26, 59-60, 124
direct monitoring 288
Distributed System Object (DSOC) 245
Double Data Rate (DDR) 82
Downlink Shared Channel (DLSCH) 127

E

electronic systems
    gigascale 2
    nanoscale 2
Element Interconnect Bus (EIB) 61
embedded networks 143, 149, 151, 153
embedded security 5
Embedded System-level Platform synthesis and Application Mapping (ESPAM) 245
embedded systems 1-3, 6, 14, 16
encryption keys 6
energy-awareness 44-45, 50

F

fault tolerance 308
Field Programmable Gate Array (FPGA) 156
Finite State Machine (FSM) 265, 270
First-In First-Out (FIFO) 65, 113
Forward Error Correction (FEC) 103
Frequency Division Duplexing (FDD) 103

G

Globally Asynchronous Locally Synchronous (GALS) 178, 220
Graphical User Interface (GUI) 161
Gray Level Co-occurrence Matrix (GLCM) 60

H

H.264 Motion Compensation 59, 67
Hardware Description Language (HDL) 80, 156, 246
hardware/software co-design 123
HERMES 268
Hierarchical Agent Monitored SoCs (HAMSOC) 278, 280-281
Hybrid Automatic Repeat ReQuest (H-ARQ) 103
Hyper Frame Number (HFN) 130

I

input variables 222
Integrated Signal processing Systems (ISS) 81

K

Kahn Process Network (KPN) 242

L

Large Scale Integration (LSI) 6, 21, 99
level of detail 267
Link Failure Probability (LFP) 179, 194-195
Log-Likelihood Ratio (LLR) 114
Long-Term Evolution (LTE) 102-103, 122
Lyra method 297-299
    service decomposition phase 300
    service distribution phase 301
    service implementation phase 302
    service specification phase 299

M

macroblock (MB) decoding 68
macroblock (MB) partition 68
Medium Access Control (MAC) 126, 139
Memory-Access Timing Model 84
Memory-Access Unit (MAU) 94-95
Memory Flow Controller (MFC) 62
memory interface 79-82, 92, 98
memory-interface controller 86-89, 92
Memory-Interface Description (MID) 80, 86, 88
    behavioral elements 87
    structural elements 86
Memory-Interface Intermediate Representation (MIIR) 89
memory mapping 90, 92-93
memory-resource specific external interface 85-86
memory subsystem 26
    design 28
memory types
    asynchronous 82
    dynamic 83
    pipelined 83
    static 83
    synchronous 82
mesh connected crossbar architecture 237
MIDdleware based on Action Systems (MIDAS) 45
middleware level 44

Minimum Mean-Square-Error (MMSE) 107
Minimum Sized Inverters (MSI) 181-182, 188
model confidence 267
Model of Computation (MoC) 242
Models of Computation (MoC) 154, 265
monitoring operations 288
    specifications 290
    types 288
Motion Compensation (MC) 59-60, 67, 77-78
Motion Vector Prediction (MVP) 60, 68
Motion Vectors (MV) 60, 68
Multidimensional Software Cache (MDSC) 59-61, 64, 76
Multiple-Input Multiple-Output (MIMO) 103
Multi-Processor System-on-Chip (MPSoC) 6, 79, 156, 173, 220, 263

N

network latency
    basic 202
    latency analysis 198
    upper bound analysis 204, 209
network manager 44
Network-on-Chip (NoC) 11, 16, 20, 154, 158, 163, 178, 195, 197-198, 217-218, 263, 276-278, 281, 295
Next Iteration Initialization (NII) 113
nonoregime 1
NoTA 148

O

Open Computing Language (OpenCL) 24
Orthogonal Frequency Division Multiple Access (OFDMA) 103
output variables 222

P

Packet Data Convergence Protocol (PDCP) 126, 139
Paging Control Channel (PCCH) 129
Parallel Memory Coding Template (PMCT) 40
parallel programming 24, 37
Payload Abstraction Technique (PAT) 267, 270
photogrammetry logic 273
Physical Downlink Shared-Channel (PDSCH) 105
PHY solution 147
PHY technology 147
Platform Abstraction Layer (PAL) 243-244, 249
platform modeling 267
    BOÇA 270
    JOSELITO 268
    RENATO 268
platform topology 242
power monitoring 291, 293
Power Processor Element (PPE) 61
Pre-coding Matrix Indication (PMI) 109
Primary Synchronization Signal (PSS) 105
Printed Wiring Board (PWB) 144
processor architecture 79, 82
Process, Voltage and Thermal (PVT) 279
prolog and epilog 25
Protocol Data Unit (PDU) 128
protocol processing 123-124

Q

Quadratic Permutation Polynomial (QPP) 115
Quality-of-Service (QoS) 11, 147, 149, 153, 276

R

radio baseband 110
Radio Link Control (RLC) 105, 126, 139
Random Dopant Fluctuation (RDF) 177, 179, 196
Real-Time Operating System (RTOS) 123, 128, 131
Register Transfer Level (RTL) 10, 55, 79-80, 246
remote monitoring 288
Retinex algorithms 96
Robust Header Compression (ROHC) 126, 140

S

schedulability analysis 198, 200
scratchpad memory 60, 68
    software cache 60
Secondary Synchronization Signal (SSS) 105
self-aware systems 4
service availability analysis 251
Service Data Units (SDU) 128
service dependency graph 246
service exchange relations 249
service export relations 250
service flow graph 251
service impoort relations 250
Service Relation Model (SRM) 244, 249
Single-Carrier Frequency Division Multiple Access (SC-FDMA) 104
Single Instruction Multiple Data (SIMD) 62
Single Instruction Multiple Task (SIMT) 35, 110
Single Processor System-on-Chips (SPSoC) 8
Soft-Input Soft-Output (SISO) 113

Software Defined Radio (SDR) 16, 102-103, 110, 120, 157
stepwise refinement 220
StratixII and StratixIV 156
sustainable development 45
Symmetric Multi-Processing (SMP) 245
Synchronous Data Flow (SDF) 265
Synergistic Processing Elements (SPE) 61
Synergistic Processing Unit (SPU) 62
System-on-Chip (SoC) 8, 14, 16, 19, 79, 123, 138, 156, 173-175, 178, 264-265
System-on-Package (SoP) 8

T

tapered buffers 180, 185
Task-Transaction Level (TTL) 245
Time Division Duplexing (TDD) 103
Transmission Time Interval (TTI) 128, 132, 134

U

UML sequece diagram 265
unified internal interface 85
Unified Modeling Language (UML) 264, 298, 319
UniPro proposal 147
Universal Asynchronous Receiver Transmitter (UART) 124
Universal Serial Bus (USB) 16, 155

V

vector memory 24, 28-30, 40
Vector Register File (VRF) 24, 26, 28-29, 31-33, 35, 38-39
very large scale integration (VLSI) 6, 21, 99
virtual hardware platform 123
Virtual System Prototype (VSP) 123-124, 131, 136

W

Wireless Local Area Networks (WLAN) 5
wormhole switching 199
    communication 199
    priority based 200, 204
    schedulability 199