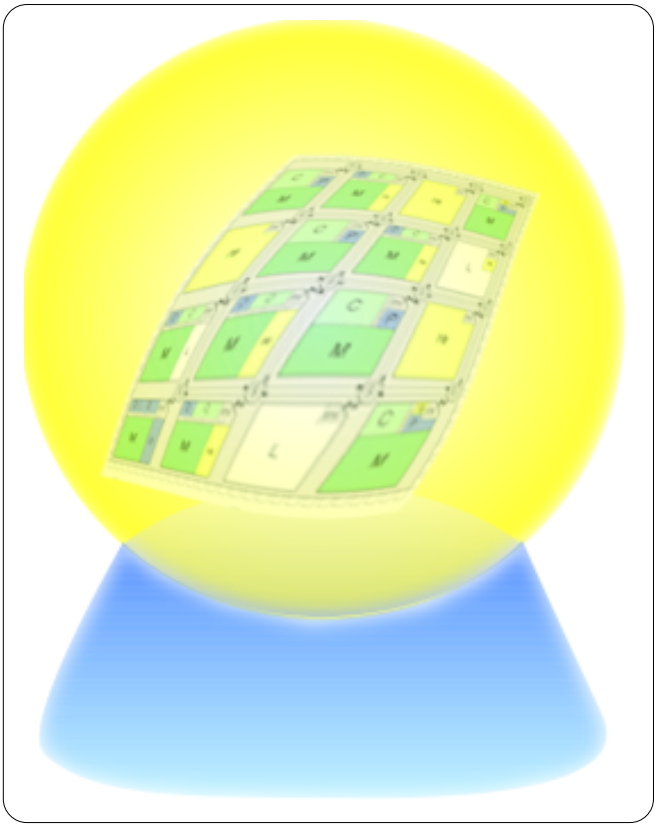
ESPOO 2004 VTT PUBLICATIONS 523



Juha-Pekka Soininen

Architecture design methods for application domain-specific integrated computer systems

VTT PUBLICATIONS 523

Architecture design methods for application domain-specific integrated computer systems

Juha-Pekka Soininen

VTT Electronics

Academic dissertation to be presented
with the assent of the Faculty of Technology, University of Oulu,
for public discussion in Kajaaninsali (L6), Linnanmaa, Oulu,
on May 7th, 2004, at 12 noon.

ISBN 951–38–6363–8 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6364–6 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Leena Ukskoski

Otamedia Oy, Espoo 2004

Soininen, Juha-Pekka. Architecture design methods for application domain-specific integrated computer systems. Espoo 2004. VTT Publications 523. 118 p. + app. 51 p.

Keywords decision support methods, quality estimations, mappability estimation, platform based design

Abstract

The role of the single computer inside application-specific integrated circuits is changing with the increasing capacity of semiconductor technology. The system functionality can be partitioned to a set of communicating application domain-specific computers instead of developing the most efficient general-purpose computers that fulfil all kinds of computing needs. The main design challenges are the complexity and diversity of application-domains and the complexity of platforms which can provide enough capacity for those applications.

The architecture design methods presented in this thesis are targeted at application domain-specific computers that are implemented as integrated circuits. Backbone-platform-system design methodology separates the technology, platform design efficiency and application development problems from each other. It also provides a system design framework for the architecture design methods presented. The methods are based on complexity, mappability, and capacity-based quality estimations that are used as decision support and quality validation tools. Abstract models of both applications and architectures enable rapid estimations and adequate coverage in design space exploration.

The methods have been applied to various case examples. Complexity-based estimation provided a systematic approach to the selection of an architecture template that takes into account the changes in technologies and design efficiency. Mappability-based processor-algorithm quality estimation enabled us to study more than 10,000 processor architectures for WLAN modem transceiver example. Capacity-based quality estimation was used in the performance evaluation of configurable multiprocessor architecture. In all cases the respective simulations using for example instruction-set simulators would have taken much longer and required advanced post-processing of results.

Preface

This research was carried out in VTT Electronics between 1999 and 2002. Most of the research was done in MULTICS, SCIFI and NOCARC projects. I want to express my gratitude to all companies involved in those projects and, especially, to Tekes for funding them.

Very many people have supported, encouraged and inspired me during this work. Docent Martti Forsell and Research Professor Aarne Mämmelä have constantly encouraged me, guided me into scientific research methods and emphasized the importance of research and education. Mr. Kari Tiensyrjä has been in a key role in setting up the research projects and research atmosphere in which this work has been done. This thesis would not have been possible without his help. Mr. Jari Kreku, Mr. Antti Pelkonen, and Mr. Jussi Roivainen have been developing and implementing the ideas in this thesis with me. Their role has been indispensable and I owe them a lot.

Mr. Tapio Rautio, Ms. Anu Purhonen, Mr. Klaus Melakari and Mr. Mika Kasslin from Nokia provided a lot of interesting ideas during the MULTICS project. The large steering group of the SCIFI project was a challenging and productive discussion forum. The comments and feedback from all the steering group members and the project group was extremely valuable. In the NOCARC project the co-operation with Royal Institute of Technology in Sweden, and Nokia and Ericsson resulted in an excellent exchange of ideas. The discussions and co-operation with Professor Axel Jantsch and Professor Shashi Kumar from the Royal Institute of Technology and Mr. Klaus Kronlöf from Nokia was very fruitful.

I also appreciate the work of all the other co-authors of original papers not mentioned so far, i.e. Ms. Sandrine Boumard, Professor Ahmed Hemani, Professor Hannu Heusala, Mr. Michael Millberg, Professor Petri Mähönen, Mr. Yang Qu, Mr. Jussi Riihijärvi, Mr Mika Saaranen, Mr Tommi Salminen, and Professor Johnny Öberg.

VTT has supported me in this work by providing excellent research facilities and working environment. I want to thank the management and all the people at VTT. I especially want to thank the support services at VTT Electronics.

Because of their professional way of managing everything from travelling arrangements to daily routines, I have been able to concentrate on my research work.

I want to thank Professor Hannu Heusala for supervising this thesis and for several encouraging discussions. I also wish to thank Professor Jari Nurmi and Professor Rolf Ernst, the reviewers of the thesis, for their comments and suggestions.

I want to thank HPY:n tutkimussäätiö and Seppo Säynäjäkankaan tiedesäätiö for their very motivating and important economical support.

Matti and Vappu Turunen let me to use their apartment as a peaceful place to complete this thesis. Most of this thesis was written there during November 2002 and October 2003. Thank you. Kiitos.

Finally, I want to thank my wife Tuija and our children Ulla and Otto for their patience and support. Allowing one to concentrate on the work is one part of it; the other part is to force the mind to think about the more important issues, such as simulated combat flights.

Juha-Pekka Soininen

7.2.2004

List of original papers

This thesis includes seven original papers published in scientific international journals or proceedings of international conferences. They are included here with the permissions of the original publishers.

1. Kumar, S., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Öberg, J., Tiensyrjä, K. and Hemani, A. A Network on Chip Architecture and Design Methodology. *Proceedings of 2002 IEEE Computer Society Annual Symposium on VLSI*, Pittsburgh, Pennsylvania, USA, 24–25 April 2002. IEEE Computer Society Press 2002. Pp. 117–124.

2. Soininen, J.-P., Forsell, M., Pelkonen, A., Kreku, J., Jantsch, A. and Kumar, S. Extending platform based design to Network on Chip systems. *Proceedings of 16th International Conference on VLSI Design 2003*, New Delhi, India, 4–8 January 2003. IEEE Computer Society Press 2003. Pp. 401–408.

3. Roivainen, J., Riihijärvi, J., Mähönen, P., Soininen, J.-P. and Saaranen, M. Minimisation of functionality and implementation of embedded low-power WWW-server. *Electronics Letters 2002*. Vol. 38, No. 2, pp. 100–101.

4. Soininen, J.-P., Boumard, S., Salminen, T. and Heusala, H. Application of decision-making method for architecture selection of ADSL modem. *Proceedings of Euromicro Symposium on Digital Systems Design, DSD 2001*, Warsaw, Poland, 4–6 September 2001. IEEE Computer Society Press 2001. Pp. 21–28.

5. Soininen, J.-P., Kreku, J., Qu, Y. and Forsell, M. Mappability Estimation Approach for Processor Architecture Evaluation. *Proceeding of 20th NORCHIP Conference 2002*, Copenhagen, Denmark, 11–12 November 2002. Technoconsult, Cogenhagen NV, Denmark 2002. Pp. 171–176.

6. Soininen, J.-P., Kreku, J., Qu, Y. and Forsell, M. Fast Processor Core Selection for WLAN Modem using Mappability Estimation. *Proceedings of the 10th International Symposium on Hardware/Software Codesign*, CODES 2002, Estes Park, Colorado, USA, 6–8 May 2002. ACM Press 2002. Pp. 61–66.

7. Soininen, J.-P., Pelkonen, A. and Roivainen, J. Configurable Memory Organisation for Communication Applications. *Proceedings of Euromicro Symposium on Digital System Design, DSD2002*, Dortmund, Germany, 4–6 September 2002. IEEE Computer Society Press 2002. Pp. 86–93.

I was the principal author of all except Paper 1 and Paper 3. However, my contributions to these papers have also been essential.

Contents

Appendices of this publication are not included in the PDF version.
Please order the printed version to get the complete publication
(http://www.vtt.fi/inf/pdf/)

List of acronyms

ACM	Association for Computing Machinery
ADSL	Asymmetric Digital Subscriber Line
ALAP	As Late As Possible
ALU	Arithmetic Logic Unit
ASAP	As Soon As Possible
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction-set processor
ATPG	Automated Test Pattern Generation
BiCMOS	Bipolar Complementary Metal-Oxide Semiconductor
BPS	Backbone-Platform-System design methodology
CAD	Computer Aided Design
CFI	CAD Framework Initiative
CFSM	Communicating Finite State Machine
CIF	Caltech Intermediate Format
CISC	Complex Instruction Set Computer
CMOS	Complementary Metal-Oxide Semiconductor
COTS	Commercial Off-The-Shelf

CPI	Cycles Per Instruction
CPU	Central Processing Unit
DASC	Design Automation Standards Committee
DLP	Data Level Parallelism
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
EDA	Electronic Design Automation
EDIF	Electronic Design Interchange Format
FFT	Fast Fourier Transform
FOM	Figure Of Merit
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GDSII	Graphic Design System II
GPP	General-Purpose Processor
HDL	Hardware Description Language
HW	Hardware
I/O	Input/Output

IBM	International Business Machines
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IHIP	Information Highway Interface Platform
ILP	Instruction Level Parallelism
IP	Intellectual Property or Internet Protocol
IPC	Instructions Per Cycle
ISA	Instruction Set Architecture
ISS	Instruction Set Simulator
ITRS	International Technology Roadmap for Semiconductors
MAC	Multiply and Accumulate
MESCAL	Modern Embedded Systems, Compilers, Architectures, and Languages project
MIPS	Million Instructions Per Second
MMX	Multimedia extension
MOPS	Million Operations Per Second
MPEG	Moving Picture Expert Group
NoC	Network on Chip
PC	Personal Computer

PDA	Personal Digital Assistant
PLD	Programmable Logic Device
PLP	Program Level Parallelism
QFD	Quality Function Deployment
RASSP	Rapid prototyping of Application-Specific Signal Processors
RDPA	Reconfigurable Data Path Array
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTL	Register Transfer Level
RTOS	Real Time Operating System
SMS	Short Message Service
SoC	System on Chip
SOI	Silicon On Insulator
SRAM	Static Random Access Memory
SW	Software
TLP	Thread Level Parallelism
TOSCA	Tools for System Co-design Automation
TSMC	Taiwan Semiconductor Manufacturing Company

UML	Unified Modeling Language
WAP	Wireless Access Protocol
VCI	Virtual Component Interface
VCX	Virtual Component eXhange
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VIS	Visual Instruction Set
WLAN	Wireless Local Area Network
VLIW	Very Long Instruction Word
VLSI	Very Large Scale Integration
VSIA	Virtual Socket Interface Alliance

1. Introduction

The development of semiconductor technology has been rapid since the invention of an integrated circuit in 1958 by Kilby and Noyce. Since then, the complexity of a single chip, the number of gates, has approximately doubled every 18 months, according to the prediction/estimation known as Moore's law (Moore 1965). Current state-of-the-art processors, such as Pentium4, have more than 40 million transistors in a single chip (Bentley 2001) and the design of over 50 million gates ASICs has already been started (Koehl et al. 2003). The silicon complexity, the number of gates per chip, is the product of chip size and gate density. It measures only the structural complexity of the chip. The silicon capacity, the product of silicon complexity and clock frequency, also measures the functional complexity of the chip. The internal clock frequencies of state-of-the-art chips have increased with the shrinking feature size, and local clocks approaching gigahertz range frequencies are nowadays possible inside ASICs (Carrig 2000).

The impact of integrated circuit-based systems on our everyday lives has been enormous. In addition to personal computers (PC) and laptops, we daily use tens of computers that are embedded in products and integrated into application-specific integrated circuits (ASIC). There are integrated circuits and embedded computers, or even computer networks, in clocks, stereos, TVs, ovens, refrigerators, mobile phones, cars, etc. Almost every product that uses electricity nowadays also uses integrated circuit technology. In the year 2000 alone, more than 1 billion embedded processors were being sold and the total value of the semiconductor market was about 200 billion Euro (IC Insights 2001).

The silicon content, the value of integrated circuits, of an electronic product has also increased steadily. The "intelligence" of ASICs has increased. System-on-chip, SoC, is an ASIC that contains one or several processor cores and embedded software. The number of SoC designs has been predicted to outperform the number of standard logic designs in 2003 (IC Insights 2001). According to the international technology roadmap for semiconductors (ITRS 2001), the trend of increasing silicon capacity and structural complexity will continue at least until 2016, when the ultimate physical limits of silicon and manufacturing equipment are reached. This means that from the current 0.13 μm technology generation we will move to 90 nm technology in 2004 and 65 nm

technology in 2007. In fact, some news announcements from silicon vendors such as TSMC and IBM indicate development that is even more rapid. The overall maximum complexity of a single chip system in production will increase from 899 million transistors to 1,020 billion transistors and 2,041 billion transistors respectively. In 2016 the complexity will be about 16.3 billion transistors with 22 nm technology.

Computers have been divided into servers, desktop computers and embedded computers (Hennessy & Patterson 2003). Particularly in the high-end embedded computing domain, such as telecommunications and image processing, the optimal processor architectures have been application-specific (Parhi 1999). The applications of embedded computers are typically fixed at system design time and the system designer can optimise his architecture for those specific applications. In general-purpose computing the computer system designer provides computational and storage capacity for general problems. The application domain-specific computers combine the benefits of both approaches for a limited set of application types. The objective is to design platforms for a variety of products or product variants that are optimised for the characteristic requirements of the application types.

The full exploitation of future silicon capacity requires new architecture approaches and new design paradigms. Multiple computers in a single chip (Cesário et al. 2002), message-based communication networks (Guerrier & Greiner 2000), reconfigurable processing arrays (Compton & Hauck 2002), and different embedded memory technologies such as SRAM, DRAM and FLASH are already reality. Traditional computer architectures are not scalable because they will have significant communication and performance overheads. The limitations of ASIC technology, such as power consumption, communication, I/O, testability, etc., will require new solutions. In addition, the required design effort is becoming a major limitation to system complexity.

1.1 Problem definition

The problem with architecture design methodology will be the complexity of the target system. The number of objects in a computer system architecture is increasing rapidly due to the achievements in ASIC technology. The amount of

software and the complexity of software systems are also increasing. The evaluation of design alternatives and the optimisation of system parameters cannot be done using traditional methods because of the required work and resources.

A system-on-chip architect has to integrate the views of other experts in the design team when real systems are developed. Integrated circuit technology sets the boundary that limits the architectural choices. The theory of computing systems and their architectures define alternatives from which the system architect has to choose the basic approach. The application requirements that result from customer needs and expectations give the system architect measures of the goodness of his choices. Finally, the design methodologies, design flows, languages, methods, and tools are the means for organising and executing the system architect's tasks. Architecture design methodology must take these different aspects into account and provide the means for selecting most feasible alternatives.

The most significant CMOS technology-dependent architecture design constraints are caused by the signal delays (Sylverster & Keutzer 1999, Ho et al. 2001) and power dissipation. The speed of signal is a constant that depends on the characteristics of the wire. The synchronous design style used in current designs cannot be used when the size of the chip increases, the gate delay decreases and the clock frequency increases. The power dissipation is a problem because every logic operation and signal transition consumes energy and although voltage scaling, etc., can reduce the energy consumption, the overall complexity of the chip increases far more rapidly. It will not be possible to deal with the required electric currents and dissipate the resulting heat.

System-on-Chip architectures are a special case of computer architectures. They have usually been based on RISC processors, DSP processors, embedded SRAM memories and dedicated hardware accelerators. However, the more complex processors and architectures are now becoming feasible as integrated computer architectures as well. Parallel and distributed architectures implementing asynchronous communication schemes and package-based messaging with on-chip networks have also been proposed recently (Guerrier & Greiner 2000, Dally & Towles 2001, Benini & De Micheli 2002). Reconfigurability as an intellectual

property block and new coarse-grain reconfigurable platform has introduced a new dimension to architecture design space (Hartenstein et al. 1996).

In embedded systems, application-specific architectures can lead to better performance and more power efficient solutions. Customisation can be done at various levels from products and architectures to components. The problem is to find the balance between generality and optimality. The identification of a product platform and a product kernel is essential at the product level. At the architectural level the question is how to fix the interconnections between the computational resources and the communication resources. The design space varies from generic interconnect networks to fixed hardware architectures. Reconfigurable processors are examples of design-time customisation at the component level. Optimisation of the instruction-set architecture of the processors for specific applications has been reported to give significant benefits over general-purpose processors (Wang et al. 2001).

Design productivity is becoming the major obstacle to system-on-chip development. It has increased significantly more slowly than silicon capacity and is, therefore, going to be the limiting factor to the size and complexity of an ASIC. High-level synthesis and higher abstraction levels in design were the first attempts to deal with complexity in the early 1990s. The applicability of the developed approaches turned out to be limited to a very special type of system. Intellectual property block-based design was the second attempt to tackle the problem, but even the extensive reuse of third-party design seems not to be enough. Currently the focus is on platform-based design approaches, where the application and computing platform development are separated from each other, and where the idea is to use existing platforms as a basis onto which new functionalities or resources are added.

The design process can be viewed as a sequence of design decisions. In the beginning the number of alternative implementations is largest and each decision actually reduces the design space, as depicted in Figure 1. At the end of the process there is only one alternative and the implementation of the product instance remains. Simultaneously, the number of design objects increases and the cost of making changes to the design increases. Therefore, the ability to understand the consequences of early technology and architecture-related decisions are essential in reducing the design costs and time.

The architecture design methodology and design flow depends on the type of system. With fixed hardware architectures, the functional modelling of the final application precedes the structural design. With computer and software-based systems, the processor is designed based on the generic idea of the application software characteristics. The system-on-chip technology enables co-optimisation of both the architecture and the functionality software. Because of complexity, abstract presentations of models and applications are needed, and special focus must be put on decision-making criteria.

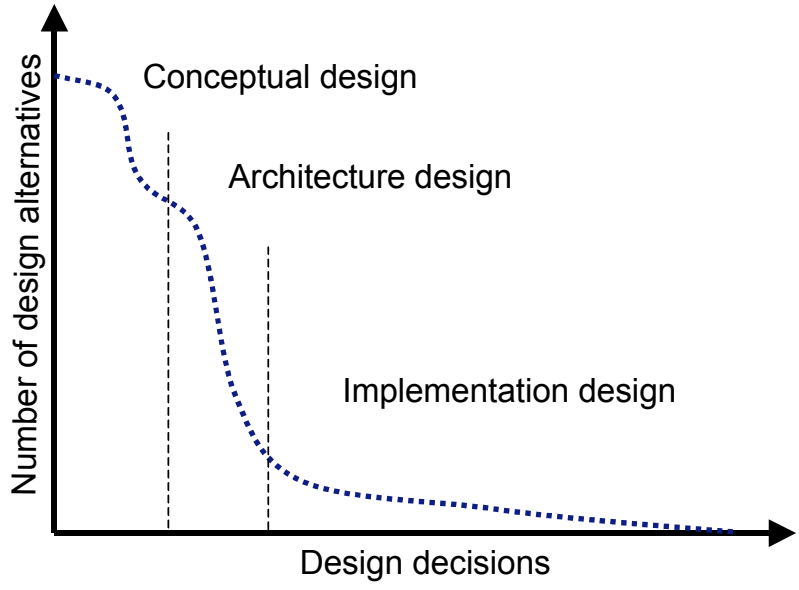


Figure 1. The effect of design decisions on the number of possible implementations of the system.

Three approaches are used for system architecture quality estimation today: analytical modelling, simulation and benchmarking (Heidelberger & Lavenberg 1984, Jain 1991). Analytical modelling can be applied when the problems are relatively simple. Benchmarking requires that the hardware is available, and the results are strongly dependent on benchmark program characteristics. In the case of embedded systems in particular, the final application may have so many unique characteristics that benchmarks are not applicable. Simulation alternatives can be used, but in the case of performance simulation the construction of a simulation model requires much work, and in the case of hardware simulation, the simulation performance is a major bottleneck.

1.2 Research hypothesis

The objective of the research was to develop methods and techniques that help in making system architecture decisions during system-on-chip design. The particular aim has been to provide fast techniques that use abstract models, and to develop simple estimation algorithms. Component selection, allocation of functionality and performance design are the design activities that differ from traditional electronic system design when complex system platforms are the target. *The basic questions that describe the problem are: Why does some computer system architecture give better performance and cost characteristics to some applications than some other architectures? How could we use abstract models for the analysis of the potential of the architecture?* The system architecture contains both hardware architecture and software architecture. The methods presented in this thesis are targeted at the hardware architecture design.

The hypothesis in this research is that by applying quality estimation techniques at the architectural level it is possible to distinguish better alternatives from worse, and thereby limit the design space effectively. The applicable estimation techniques include complexity, mappability and capacity estimations.

The complexity-based quality estimation can be used in identifying the capacity boundaries and characteristics of processing and communication resources. The hypothesis in this thesis is that the evaluation of architecture and technology candidates can be done using a complexity estimate-based figure of merits.

The hypothesis in mappability-based quality estimation is that it can reveal the potential of the system before it is actually implemented. This can be done by studying the characteristics of the system from different viewpoints, e.g. by analysing the logical execution flows of algorithms and by studying the operation of the processing resources. Such techniques that can estimate the final quality of a system can be used for helping to decide which components to use, how to modify the components, and what kind of components to use.

With capacity-based quality estimation the hypothesis is that modelling workload characteristics and mapping to architecture capacity models allows rapid and simple evaluations of the quality of architecture. New transaction-level

modelling and simulation methods should be very suitable for these kinds of feasibility studies.

The novelty of the proposed approach is in the chosen abstraction levels, and in the idea of estimating the potential rather than the characteristics of the system model. The benefit of the chosen approach is that the estimators can be simpler because they do not need detailed system models. Secondly, the ability to design an optimal system is not restricted because of abstract models. In this approach optimistic guesses on system quality are made during the estimation. This allows the evaluation of more alternatives more rapidly, which will be a necessity in the future.

1.3 Research methods

Industrial design methodologies for complex systems-on-chips are the results of evolutionary processes inside the companies. Success has depended on the ability to integrate the required competence areas with the ability to learn from mistakes. The role of academic research and the various EDA tool vendors has been to provide the material in the form of theories, methods and tools. Revolutionary, complete solutions for all aspects of SoC design have not been successful so far.

The research method in this thesis follows the evolutionary nature of industrial SoC design methods. The architecture design methodology presented here is a synthesis of technology trends, architecture trends, product market trends, research trends and my background and experience in this field.

The complexity of target architectures is such that the formal, detailed and complete modelling and measurement type of validation is not economically feasible. Therefore, in this thesis the complete design methodology is not validated as such. Instead, the feasibility of the approach is demonstrated by developing three methods, applying them to the case example systems, and comparing the results with more detailed implementations or simulations. Each of the estimation methods studied in this thesis is a potential topic for further research. In this work the focus is on the overall methodology and the estimation methods are only studied to the point where the potential and feasibility is

proven. The value of the overall methodology can only be assessed using common sense and intuition.

1.4 Organisation of thesis

This thesis consists of an introduction and seven original papers. The selected papers represent the latest results of the research I have carried out. They have been written between 2000 and 2002. However, the ideas in the thesis are the outcome of fifteen years of research starting from the late 1980s, when ASIC design was taking its first steps in using hardware description languages (HDL) instead of graphical block diagrams.

The introduction of VHDL in 1987 revolutionised ASIC design and enabled the transition from a very structural gate array ASIC to register-transfer level (RTL) synthesis and cell-based designs. At the same time, the technology allowed the integration of a processor core and dedicated logic into the same chip. System modelling, simulation and SW/HW partitioning were research topics that initiated a number of papers. (Juntunen et al. 1988, Soininen et al. 1989, Sipola et al. 1991, Kauppi & Soininen 1991, Kauppi et al. 1992, Sipola et al. 1992)

In the 1990s it was obvious to the research community that using processors in the ASIC enabled new system optimisation possibilities. Silicon technology enabled the integration of more complex processors, embedded memories and analogue functions into the same chip. System-level simulation and verification became more important (Soininen et al. 1995, Soininen et al. 1997, Soininen et al. 1998a). In the late 1990s the system-on-chip projects started to be too expensive and complicated. In order to increase the usability of the design at system-level, the product platform and platform chip concepts were introduced and studied also in Soininen (1997), Soininen et al. (1998b), and Soininen (1999). The first ideas for a new architecture design method were presented in Soininen et al. (2001). The concepts of intellectual property block or virtual component were needed to enable reuse at block level. These issues were studied in Riihijärvi et al. (2001) and Pelkonen et al. (2001).

Today we are facing a paradigm shift from analogue to the introduction of VHDL. SystemC is a new language, which is claimed to integrate software

development and hardware development in a way that enables system-level design. The system designer is again drawing blocks and mapping functionality into them. In architecture research the new computing element array ASICs integrate computers, processors or ALUs into flexible fabrics that can be reused in various types of products. It remains to be seen whether these innovations can fulfil their promises. Anyway, if we look at the topology of these innovations, they have interesting similarities with gate array ASICs. The circle seems to be closing.

The introduction part of this thesis is organised as follows. Chapter 2 briefly introduces the design methods for IC-based systems; Chapter 3 introduces the main design problems the system architect has to face when developing complex systems; Chapter 4 presents the main ideas of the original papers in a condensed form; Chapter 5 introduces the original papers; and Chapter 6 gives the conclusions.

2. Design methods for IC-based systems

Integrated circuit technology has provided a variety of implementation formats for system designers. Implementation format defines what technology is used, how the switching elements are organised on the surface of chip and how the system functionality will be implemented. The implementation format also affects how systems are designed and limits the system complexity. Examples of the alternatives are a combinational logic implemented with discrete components and a billion-transistor Network-on-Chip consisting of tens of computers.

The integrated circuit technology has mostly been based on silicon. Today the majority of IC-based systems are based on CMOS technology (IC Insights 2001). The CMOS switching elements that implement basic Boolean functions such as AND, OR and NOT in the digital systems are the dominant technology. The current trend is to use Silicon-on-Insulator (SOI) technology to improve the circuit speed and power efficiency. However, the bipolar technology is widely used in analogue devices, BiCMOS technology is used in mixed-signal and microprocessor designs, and Gallium-Arsenide and Silicon-Germanium-based products are used in high-performance products.

If we look at the organisation of switching elements, regularity of organisation and granularity of elements are the essential parameters. Examples of regular formats are gate arrays, programmable logic devices (PLD), memories, field programmable gate arrays (FPGA), reconfigurable data path arrays (RDPA), and vector processors. Regularity has a strong impact on the required design effort, because the reuse of the design is very simple. The problems with regularity are that the usability and performance of resources may be limited because of the structure. The granularity expresses how much functionality is encapsulated into one design object. In fine-grain, medium-grain and coarse-grain formats the examples of basic units are the logic gates, standard cells – such as arithmetic logic units (ALUs) or multipliers – and intellectual property blocks – such as processor cores, network interfaces, etc. – respectively. The granularity affects the number of required design objects and, thereby, the required design or integration effort.

The three main approaches for implementing the system functionality are dedicated systems, reconfigurable systems and programmable systems. The

differentiating factor is the configuration frequency, i.e. how often the structure of the system is changed. In dedicated systems the structure is fixed at the design time, as in ASICs. In programmable systems the data path of the processor core, for example, is configured by every instruction fetched from memory during the decode-phase. The traditional microprocessor-based computer is the classical example. In the reconfigurable systems the structure of the system can be altered by changing the configuration data, as in field programmable gate arrays (FPGA). In dynamically reconfigurable systems the configuration can be done during run-time; in statically reconfigurable systems the configuration is done off-line.

2.1 Basic blocks of integrated computers

The basic building blocks for application domain-specific integrated computers are input, output, data path, memory and control – as with an ordinary computer (Patterson & Hennessy 1998). The implementation format of each basic block can, at least theoretically, be any combination of the previous classes. For example, the data path and control can be implemented using fixed dedicated hardware, as in ASICs (Smith 1997), or reconfigurable logic (Caspi et al. 2001, Compton & Hauck 2002) or microprogrammed units, as in processors (Anceau 1986). The data path can consist of regular elements, as in reconfigurable arrays, or dedicated, pipelined blocks, as in superscalar processors. The granularity of data path elements can vary from single gates in ASIC to complete processors in multiprocessor architectures. However, an integrated computer system typically consists of a processor or processors, memory organisation and communication network. The tasks of the system architect are to know them, to identify the best alternatives and to join them, so that the system fulfilling the quality targets can be quantified and designed.

The diversity of the applications has led to a variety of processor architectures that exploit parallelism differently (Forsell 2002a). The basic taxonomy of processor architecture was presented by Flynn (1966) and used the number of instruction streams and data elements being processed as the criteria. Skillicorn (1988) extended Flynn's taxonomy to multiprocessor architectures and took into account the internal structure of the processor, e.g. pipelining and parallelism. Corporaal (1998) used the instruction issue rate, number of operations per

instruction, number of data elements in each operation and degree of superpipelining as the dimensions of the architecture design space.

The first microprocessors were sequential processors based on Von Neuman architecture (Mazor 1995). Initially, the lack of processor-memory bandwidth and poor compilers resulted in complex instruction sets, and they were called complex instruction set computers, CISC. In the early 1980s the invention of reduced instruction set architecture, RISC, and the development of VLSI technology enabled the implementation of a pipelined architecture, larger register banks and more address space in a single chip processor. In the 1990s the focus was on the exploitation of instruction-level parallelism, ILP, which eventually resulted in the modern general-purpose processors (Slater 1996). Superscalar and very long instruction word architectures, VLIW, are examples of dynamic and static parallelism. Data-level parallelism, DLP, has been exploited in instruction set extensions such as Intel's MMX and Sun's VIS, where the objective has been to improve the performance of multimedia applications. Recently the silicon capacity has enabled the implementation of the first microprocessors that exploit thread-level parallelism, TLP (Koufaty & Marr 2003), and process-level parallelism, PLP, where several processor cores are integrated into a single chip. OMAP processors from Texas Instruments are well-known examples. Research approaches that are even more complex have been proposed by Taylor et al. (2002) and Mai et al. (2000).

Another dimension in processor architectures has been the application orientation that has led to a variety of different types of instruction sets and organisations. Today, in addition to *general-purpose processors,* we also have *microcontrollers*, in which the memory organisation and peripherals are integrated into the same chip. *Digital signal processors* were invented to perform simple stream-based processing, such as filtering. The Harvard and SuperHarvard architectures, advanced addressing, efficient interfaces, and powerful functional units such as multiplier and barrel shifters give superior performance in the limited application space (Madisetti 1995). *Multimedia processors* are targeted at applications where data parallelism can be exploited efficiently, i.e. real-time compression and decompression of audio and video streams and generation of computer graphics. Multimedia processors can be divided into microprocessors with multimedia instruction extensions and highly parallel DSP processors (Kuroda & Nishitani 1998). *Network processors* have an

effective interconnection network between the processing elements that operate in parallel and efficient instructions for packet classification (Wolf & Turner 2001). *Reconfigurable data path arrays,* RDPAs, have coarse-grain functional units, such as ALUs or multipliers, as configurable elements (Hartenstein et al. 1996). Bondalapati & Prasanna (2002) and Compton & Hauck (2002) give extensive overviews of configurable systems, technologies and use.

Customisation of instruction set is another technique for improving the performance of a processor in a specific application domain. *Coprocessors* are hardware accelerators for specific types of applications, such as floating-point arithmetic or multimedia. In the reconfigurable coprocessors and reconfigurable computers the idea is to dynamically load the instruction into an FPGA block (Mangione-Smith et al. 1997). *Configurable processors* are targeted at System-on-Chip products. The idea is that the processor's instructions are optimised for the application during the SoC design. The basic architecture is typically a pipelined RISC architecture, and new instructions are either integrated into the pipeline or implemented as coprocessors or architectural extensions (Wang et al. 2001, Campi et al. 2001). *Application specific instruction processors*, ASIP are designed for a particular application set. The idea is that the complete instruction set, and the selection of the architecture template, is based on the application analyses (Jain et al. 2001, Itoh et al. 2000). *Application specific processors* are processors that are synthesised from the application description using a built-in architecture template of the synthesis system. The idea is to extract computation resources from the application description and to synthesise the control that minimised the resources within given performance constraints (De Man et al. 1990).

Memory can be characterised with size, number of ports, latency and bandwidth. *Latency* is the access delay of a randomly chosen data element and *bandwidth* is the data rate of the elements. Semiconductor memory components are typically 2-dimensional regular structures, and latency is inversely proportional to the size of memory. The bandwidth depends on the memory buses, internal organisation of memory and access logic.

Computer memory is typically composed of hierarchically organised memory components or blocks. The levels of the hierarchy are called *layers*. The faster and more expensive memories are in the upper layers; the slower, bigger and

cheaper memories are in the lower layers. The objective is to keep the most accessed data items at the top of the hierarchy (Patterson & Hennessy 1998). *Cache memories* are fast memories between processors and the main memories that keep the most frequently used data items easily accessible. The idea is to benefit from spatial and temporal locality of memory accesses. Smith (1982) gives a good overview of cache memories. *Virtual memory* is a technique that implements the translation of a program's address space to a physical address space (Killburn et al. 1962). The benefits are that separately compiled programs can share the same memory and that by moving code and data between the main memory and the secondary storage a single program can exceed the physical address space (Jacob & Mudge 1998). *Shared memories* can be accessed by more than one processor in a multiprocessor system.

The main memory classes are volatile memories, e.g. static random access memory, SRAM, dynamic random access memory, DRAM, and non-volatile memories, such as read-only memory, ROM, and FLASH memory. SRAM is a fast memory, but typical implementation takes six transistors per bit. DRAM is a dense memory, only one transistor per bit, but the latency, e.g. access delay, is high. The FLASH memory also suffers from high latencies – the writing takes time, because it must be done for large blocks (Bez et al. 2003). The DRAM and FLASH memories are internally asynchronous and have different latencies for random and sequential accesses. For DRAM in particular, this has led to a variety of solutions for speeding up the overall performance, such as fast page mode accesses, synchronous interfaces, and intelligent control interfaces (Cuppu et al. 1999).

The design of memory organisation for an integrated computer system using *embedded memories* differs from a component-based memory system, because it is possible to change the parameters of the blocks, such as bus widths and memory capacity (Panda et al. 1999). The number of memory blocks does not cause a significant price increase, as is the case with discrete memory components. Only the total area is significant, because the design is relatively simple due to the regular structures of memories. Therefore, it is possible to divide the memory into smaller blocks according to the application requirements and architecture characteristics (Panda et al. 2001). The size of the memory blocks is very simple to change, making it possible to optimise the total area. The communication inside the chip is fast, which means that it is often possible

to simplify the memory organisation by reducing the number of layers. Removing caches or replacing them with *scratchpad* memories, for example, also improves power efficiency and simplifies the memory management units (Banakar et al. 2002). It is also possible to use wide buses and exploit data parallelism in communication as it is done between the cache and main memories.

Communication network is the third main component of a computer system. Communication channels can be divided into dedicated channels, e.g. *signals,* and shared channels, e.g. buses and networks. The *buses* connect subsystems, and *networks* connect computers according to the classical definition. The dedicated channels may be static *point-to-point* connections or dynamic *switched* connections. However, these distinctions are becoming very difficult with advanced buses and interconnect networks in multicomputer ASICs.

The buses can be further divided into CPU-memory and I/O buses, parallel and serial buses, or synchronous and asynchronous buses, according to their purpose or physical implementation. A typical personal computer has a CPU-memory bus that connects the cache and main memory, a high-speed bus for graphics processing, a peripheral bus and an I/O bus for external devices. The *bus transaction* is the basic read or write operation in the bus. The *bus master* is the component that can initiate transactions. The *bus slave* is the passive counterpart. In the case of multiple masters, the *bus arbiter* controls which one gets the bus. Several techniques, such as split transactions, pipelining and packet-switched buses, have been proposed for improving bus efficiency. The basics of buses, the main components and principles of operation are explained in more detail by Hennessy & Patterson (2003).

The AMBA bus is a typical example of a System-on-Chip bus. It consists of a high-speed bus for the CPU-memory interface and a low-speed peripheral bus for other, less demanding blocks (ARM 1999). Virtual Socket Interface Alliance has developed a standard interface for virtual components to on-chip buses, but real buses must be adapted to this interface with glue logic (Lennard et al. 2000). The MicroNetwork approach uses concurrent protocols, scalable data path widths and pipelining for decoupling the IP block design from the communication design (Wingard 2001). Lahtinen et al. (2002) distributes the arbitration among the connected agents. The approach is especially suitable for

continuous media systems. As examples of more complex buses, Leijten et al. (1998) have proposed a communication architecture that consists of FIFOs and time-multiplexed switches, and Lahiri et al. (2001) have introduced a randomised arbitration approach that improves bus performance. Salminen et al. (2002) gives an overview of System-on-Chip buses.

In some single-chip computer systems crossbar-switches and multistage interconnection networks based on packet-based messaging networks have replaced buses as communication channels. Interconnection networks originate from parallel supercomputer systems. Culler & Pal Singh (1999) give an introduction to the concepts, Sivaram et al. (2002) introduce taxonomy to switch architectures and Duato et al. (1997) describe the network topologies and networking principles in more detail. Dally & Towles (2001) have proposed on-chip interconnection networks instead of top-level wiring of ASICs. The idea is partitioning of the chip area into tiles and network and increasing the modularity and the efficiency of the system. The basic topology is 2-dimensional mesh with buffered routers and processing tiles. Guerrier & Greiner (2000) presented a generic packet-switched interconnection template for on-chip communication and assessed its feasibility with regard to cost and performance. The basic topology is fat-tree and the router is based on a partial crossbar with input buffers and shared output buffers. A multitude of Network-on-Chip related architectures have been published since these early papers (Karim et al. 2002, Benini & De Micheli 2002, Sgroi et al. 2001, Saastamoinen et al. 2002, Wielage & Goossens 2002.). Jantsch & Tenhunen (2003) summarise the main results.

Today, the integrated computers are eventually manufactured in ASIC foundries using silicon-processing technology. This applies to System-on-Chip solutions, FPGA-based devices and reconfigurable data path arrays, and, most likely, it will apply to Networks-on-Chip. This leads to two main requirements for system architecture and design methodology. First, it must be possible to implement all the building blocks with the chosen process. This means that we have to be able to use the technologies, such as, for example, DRAM memories and FPGAs, as part of the ASIC manufacturing process. Second, we have to be able to design the complete system. This means that we have to be able combine the information from different areas such as algorithm design, software design, hardware design, so that decision making at the system level is possible.

2.2 Models and languages

The purpose of the design is to convert the user's requirements into a system that fulfils those requirements. In order to perform this task we need to have models of the system at different levels of details, i. e. abstraction levels, and languages that describe the contents of those models. The design process is basically a sequence of steps that takes a model of a design at a higher level and transforms or refines it into a lower level model, so it consists of both models and methods. At every abstraction level we have a model that is based on a language consisting of syntax and semantics. The model of computation defines how the abstract machine can behave in operational semantics. It is important to notice that one language can contain several models of computation, and one model can have parts described in different languages.

Edwards et al. (1997) define the most common models of computation used in embedded systems.

- Discrete event models are based on the idea that tokens or events carry a totally ordered time stamp, as in VHDL simulation. The major problem is non-determinism in case of simultaneous events.

- Communicating Sequential Processes were proposed by Hoare (1985). Harel (1987) introduced hierarchy, concurrency and non-determinism into traditional finite state machines. Finite State Machine-based languages have been used by Balarin et al. (1997).

- The synchronous and reactive models (Berry & Gonthier 1992) are based on the idea that all signal event pairs are totally ordered and globally available, as in cycle-based simulators where all the values of signals are evaluated at every clock cycle.

- Dataflow Process Networks (Lee & Parks 1995) describes the systems as directed graphs where computation occurs in nodes, communication happens via arcs, and events and data are presented as tokens.

Other models of computation include rendezvous-based models that are used in process calculus. Lee & Sangiovanni-Vincentelli (1998) have proposed a

framework for comparing models of computation in the form of a meta-model. The goal is to enable transformation between modelling languages, which is needed in system synthesis from heterogeneous specification. Gajski et al. (1994) presented a model taxonomy for codesign methods, which divided models into state-oriented models, activity-oriented models, structure-oriented models, data-oriented models and heterogeneous models. The RASSP taxonomy targeted mainly at DSP systems used temporal, data value, functional and structural resolutions and programming level as classification criteria (Hein et al. 1997).

The abstraction levels and design domains have developed with the ability to use more abstract design methods and new technologies. The Y-chart by Gajski (1988) divided system models into behavioural, structural and physical domains, which each had five abstraction levels: system level, algorithm level, microarchitecture level, logic level and gate level. Later, Gajski et al. (1996) refined the levels more suitable for Systems-on-Chip. The new levels were processor level, register level, gate level and transistor level. The D-cube classification by Ecker (1995) divides the systems according to views, values and timing. The view can be sequential, concurrent or structural. The values can be abstract, composite or bits. The timing can be based on causality, clocking or propagation delays. The RASSP project collected and classified the most used techniques in the DSP domain; the subsystem's hierarchy and emphasising performance issues were the main enhancements (Hein et al. 1997).

There is a lot of variety in abstraction levels, design domains and design tasks in the design of a complex integrated multiprocessor system. This has led to a variety of languages for different purposes. The languages targeted at the modelling of complete systems are very difficult to classify according to traditional classification criteria. Therefore, only a brief introduction to the most important language classes is given.

Formats are used for presenting design information. CIF (Caltech Intermediate Format) and GDSII Stream are formats used for presenting the final layout of the ASIC. EDIF (Electronic design interchange format) is used for presenting design data such as netlist, so that it can be transferred between design environments and design tools. The CAD framework initiative, CFI, has defined a standard for

design representation, which is based on an information model for electrical connectivity information. (Smith 1997)

Hardware description languages (HDL) can be roughly divided into structural languages that replaced the schematic entries and behavioural languages that are used as higher abstraction level inputs for synthesis tools or simulation tools. VHDL and Verilog are currently the most used languages and include features of both classes (Smith 1997). Using C and, currently, C++ as a basic of HDL has been considered an attractive alternative and has generated a variety of languages, such as HardwareC (Gupta & De Micheli 1993) and SystemC (Grötker et al. 2002). Architecture description languages (Bashford et al. 1994, Fauth et al. 1995, Pees et al. 1999) are a special case of HDLs. They have been targeted at the modelling of computer and processor architectures.

Software languages were developed for easing up the program development. The microprogramming languages define how the processor is configured during the execution of an instruction. The machine code is the sequence of instructions executed by the processor. The programming languages or high level languages, such as ALGOL, Fortran, Basic or C, are transformed into machine code by compilers or interpreters (Aho et al. 1986). In Java language, the whole computer is emulated by a virtual machine program.

Specification and modelling languages are targeted at design capture instead of implementation description. Data flow process networks have been widely used in digital signal processing systems (Lee & Parks 1995). The actual content of the processes has been modelled with host languages like Prism or C. Formal specification languages, such as Synchronized Transitions, Unity and LOTOS, enable mechanical model checking of system properties. System-level design languages (Bahill et al. 1998) are used for creating high-level abstraction of the desired system. Unified Modelling Language, UML is an attempt to combine different approaches under one framework (Fowler & Scott 1997).

2.3 Design flows

Every product consists of two parts – functionality and physical realisation, e.g. resources – and the design flow is a process in which the system requirements

related to these parts are transformed into a product fulfilling them. In the process the designer adds implementation details to a system model until the product is designed. These additions concern either functionality or resources, as depicted in Figure 2. The design flow can then be considered a path on a plane, from a point where no implementation details exist to the point where the product's implementation is fully specified.

The optimal design flow depends on various aspects, including technologies, design tools, business strategies, and market situation. The design flows for mass-market products emphasise optimisation of manufacturing and material costs, which are mostly dependent on resource optimisation. The competitive advantages of high-end products are often related to functional performance. In these cases the management of functional complexity, advanced algorithms and performance benefits via optimisations of functionality are critical. In emerging markets the time-to-market issues are important and the emphasis is, therefore, on the reuse of existing knowledge and parallel development of new features.

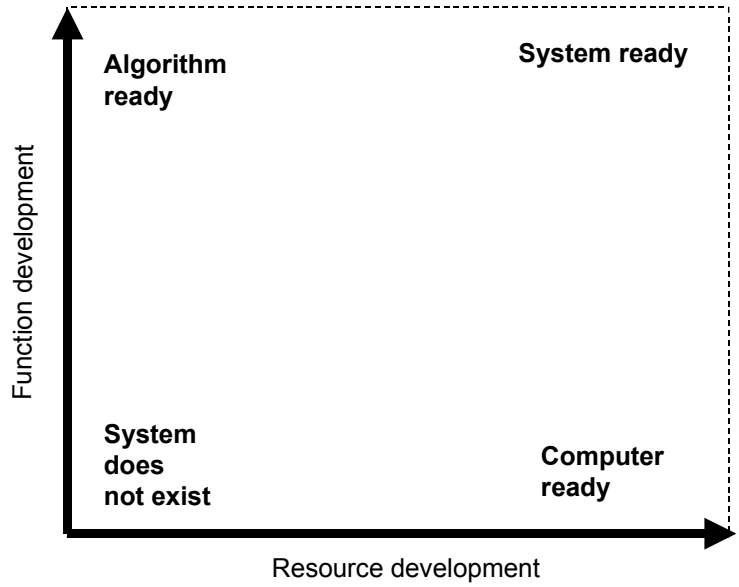


Figure 2. Design plane concept.

The IC-based systems can be roughly divided into three classes: ASICs, computers-based systems and System-on-Chips. The ASICs are mainly dedicated hardware implementations of algorithms, e.g. algorithm-on-chip-type

systems. The algorithm design precedes the resource design in the design flow. The computer-based systems are embedded system-type designs and the generic resources are designed before the actual functionality is implemented using software. In the System-on-Chip design the idea is find the optimal balance between the two earlier approaches. In practice this means the implementation of independent models and system-level optimisation criteria during architecture definition.

2.3.1 Dedicated hardware systems

ASICs are, by definition, applications implemented as integrated circuits. Designing and implementing an ASIC is an expensive and time-consuming process. It typically takes from 6 to 18 months to complete and the mask costs of an advanced manufacturing process alone are several hundred thousand Euros. A characteristic feature of an ASIC is that design mistakes or errors cannot be corrected from the final chip. Redesign and a new manufacturing iteration are required instead, taking at least one month to complete. Therefore, validation of the system functionality and verification of the design is extremely important. (Smith 1997)

Specification of functionality and algorithms

The dedicated hardware system design starts from definition and validation of functionality. In most of the design flows the design of functionality is separated from the design of implementation, e.g. resources and their control. The motivation has been the separation of concerns. Functionality has been considered independently of implementation and, therefore, has been modelled and validated using methods that are specific to the types of systems. The state-based models and algorithms are the two most common classes of functional specifications. In reactive systems the state-based approaches are more common because it is easier to model the input-output responses and internal states of a system. In the case of data manipulation, the algorithmic approaches are more natural.

The quality of the functional description depends on the functional correctness, refinement process and complexity. Functional simulation is the most common

approach for the validation of correctness, but formal approaches have also been proposed. Functional correctness is especially important in the case of dedicated hardware implementation, because of the extremely high cost of redesign and manufacturing. The main problem with simulation-based validation is simulation coverage. The state-space explosion problem is especially significant in user interfaces and reactive products. The number of use cases or use scenarios can easily exceed any simulation capacity. Formal approaches, such as state reachability analysis and property checking, have been applied to life-critical systems because they provide complete coverage and certainty, but the cost of use, e.g. the modelling effort, is high.

The functional specification itself may contain several abstraction levels, from pure behaviour to implementable behaviour, and the design flow must support this refinement process so that the final implementation complexity can be minimised. For example, in digital signal processing systems the design typically starts from mathematical models, e.g. sets of equations. Then they are partitioned into modules that are described using algorithms and real numbers, e.g. floating point arithmetic. These algorithms are validated using functional simulations. However, the implementation is normally much cheaper if the total cost function, which has number of bits in data, number of operations between data, the required storage for data, and the control of operations as the main factors, is minimised. Part of this minimisation must be done in the functional specification. For example, the number of bits can be affected by the selection of number system and by scaling. The effects of these decisions must be modelled in a fixed-point model and validated by simulations.

ASIC design

The ASIC design itself means the development of resources that implement the desired functionality. The ASIC design flow consists of *logical design* comprising design entry, netlist generation, system partitioning, logic simulation – and *physical design* – comprising floorplanning, placement, routing, circuit extraction and post-layout simulations (Smith 1997). The design entry contains the architecture design activities in this model, netlist generation refers to logic synthesis, and system partitioning refers to partitioning the system into different ASICs. The actual tools and design methods that have been used in various

phases changed with the advances in the EDA industry, from hand design to behavioural synthesis and formal verification (MacMillen et al. 2000).

Current ASIC design flows are based on synthesis. The input to synthesis is either a behavioural or RTL-level HDL description of the system. These approaches are called behavioural synthesis and logic synthesis respectively. The result of behavioural synthesis is RTL-level description, which must then be synthesised using logic synthesis. *Behavioural synthesis,* also called high-level synthesis, comprises the compilation, transformation, scheduling, allocation, partitioning and output generation phases (Walker & Camposano 1991, Gajski 1988). Compilation generates an internal presentation from the source language. Transformations are compiler optimisations or hardware-specific optimisations that try to optimise the behaviour of the design. The scheduling assigns operations to a point in time that is a control step in case of synchronous design. The allocation can be divided into register allocation, function allocation and interconnection binding. The idea is to define what resources are used for the execution of operations, and how to control the execution. It is mostly resolved using graph-theoretical methods. In order to process the design more efficiently it may be feasible to partition it into smaller, independent pieces. This can be done in various phases of the synthesis process. Output generation generates the RTL description for logic synthesis. *Logic synthesis* consists of translations and optimisations, but the output of synthesis is mostly determined by the coding style (Kurup & Abbasi 1997, Keating & Bricaud 1999). The basic flow is that the HDL is first translated to a network of generic logic cells that are technology independent. Second, the generic network is optimised using logic optimisation techniques; the optimisation is controlled by design constraints given by the user. Finally, the generic cells are mapped to cells from the target library that is technology-dependent. The optimisation goal in synthesis tools has been to meet timing constraints while minimising the area that is the main cost factor in ASICs. Advanced scheduling and allocation techniques, more complex library components, static timing analysis and timing-driven optimisations at the generic netlist level have been used for achieving these objectives. The current challenge in synthesis is taking the interconnect delay into account. The timing estimation has been based on gate delays and average wiring delays, but with shrinking feature sizes, interconnect delays are so dominating that average models are not accurate enough. *Physical synthesis* approaches try to solve these problems.

Figure 3 depicts the activities needed in ASIC design flow. The physical design and verification are the most time-consuming activities. Physical design, which is also called layout synthesis, is largely done using automated tools. However, there have recently been attempts to integrate the physical design and synthesis tools in order to manage the timing closure problems better (Chan et al. 2003). The verification consists of two parts: verification of design, which is done using simulation and formal methods, and verification of implementation, which means the design of tests – e.g. automated test pattern generation (ATPG) – and design for testability – e.g. insertion of scan structures and test ports. Current hardware simulators create native language executables, use event-based and cycle-based simulations, and support external executable models and hardware acceleration.

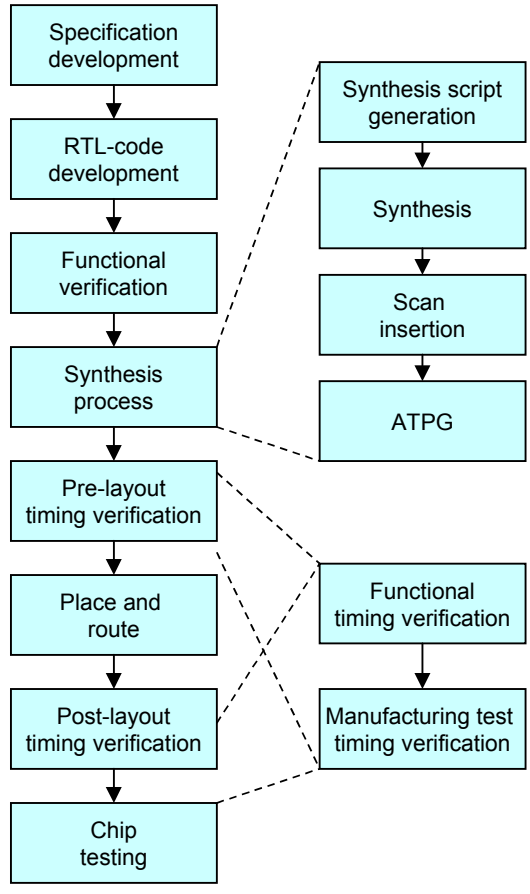


Figure 3. ASIC design flow.

In dedicated hardware design flow the functionality is designed before the resources. Only minor changes to functionality are possible when a typical industrial ASIC design is started. The fixed-point model design in digital signal processing is one example where this kind of optimisation is possible. If we use the design space representation of Figure 2, the path goes from bottom-left corner to top-left and from there to top-right.

2.3.2 Computer-based system design

The computer-based systems consist of computer hardware, software environment and applications as depicted in Figure 4. The design flow of computer-based systems has two distinct phases. Before the actual functionality of the system can be designed, we have to have an existing computer and software development tools and an environment for software. Typically, the computers are not designed with a single application in mind. Instead, the objective is to provide computation and storage capacity for a set of applications. The application set can be anything between arbitrary functions to a set of few predefined functions only, but in general, the computer design aims at providing generic resources for various types of systems. Naturally, the benefit is to be able to reuse the hardware design in several products and thereby improve the design productivity.

The same objectives have also affected software development. First, the high-level programming languages and compilers divided the software development into coding and compilation that were processor-independent and dependent respectively. Second, the operating systems encapsulated hardware-related functionalities into system services that could be called from the applications. The result was that application development became computer-independent. Recently, the Java virtual machine has moved the abstraction level even higher; the application can be developed without knowing anything about the actual implementation of the hardware or software platform.

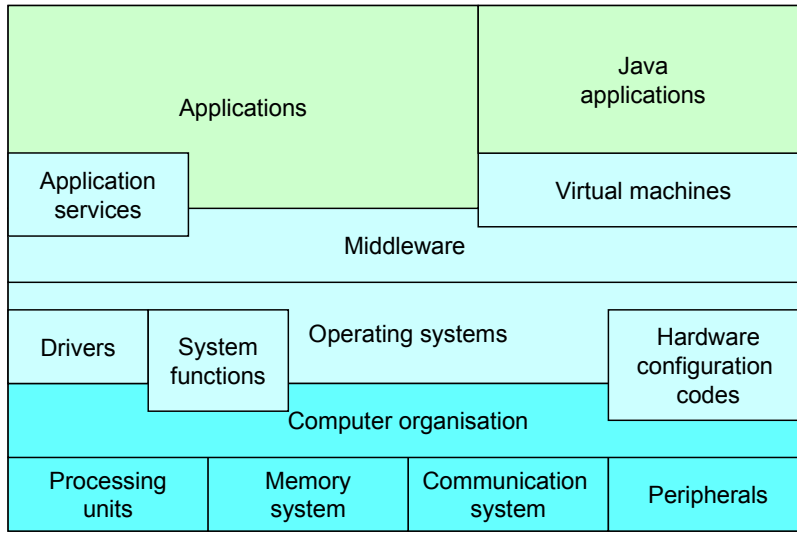


Figure 4. Basic structure of a computer-based system.

The cost of using existing hardware and software platforms is the implementation overheads. The economical and physical constraints are difficult to meet. Therefore, the choice of reuse level is an important design decision.

Designing computer resources

In the computer design the problems are to understand which design attributes are important and to maximise the performance and power efficiency under the design constraints. This process is covered in detail by Hennessy & Patterson (2003). The main activities are instruction set design, functional organisation – e.g. memory system design, processor design, bus architecture, I/O system design – logic design and implementation.

The actual design flow and tools naturally depend on the degree of freedom we have. However, the generic principle is that first the characteristic features of the application domain are analysed and the model of the workload is generated. Second, the system model is implemented and the workload is mapped to it. Third, the quality characteristics of the system are evaluated; in the case of the computers, the performance has been considered most important. Fourth, the system model is refined until a satisfactory quality is reached. The process is then repeated with a more detailed system model until the implementation exists.

If it is possible to change the internal structure of the processor, there are lot of design alternatives. In addition to the instruction set architecture – e.g. basic architecture style, memory addressing, operands, operations and control flow issues – pipelining and parallelism must also be studied in order to maximise the performance and resource utilisation. The most important objectives are "to make the common case fast", to benefit from overlapping of instructions. In addition to the hardware, compatibility and software development must also be considered. The processor design typically starts with analytical modelling, analysis and performance simulation. Several architecture languages and simulation environments have also been developed (Bashford et al. 1994, Fauth et al. 1995, Pees et al. 1999).

If we are limited to modifying the processor instruction set or memory organisation, a better understanding of the application domain becomes more important. In the case of processors, the profiling and identification of the critical functions are techniques that can help in the optimisation of the instruction set. These are especially important in the case of application-specific instruction-set processors and reconfigurable processors (Wang et al. 2001). Instruction-set simulation is the validation technique used at this level. The analysis of the memory footprints and locality of data, and the modelling of memory bandwidth with analytical methods or trace-based simulations are tools for cache and main memory optimisation (Panda et al. 2001).

The role of performance analysis using simulation or monitoring with benchmark programs is important when the objective is to balance the system performance and identify possible bottlenecks, as in designing computers using existing components (Bose & Conte 1998). At this level the system model is constructed using the models of existing building blocks, and, instead of designing components, the focus is on the selection of components.

In all phases of computer design the evaluation of performance depends on the workload model. Therefore, the quality of the workload model is essential and workload modelling is a widely studied topic (John et al. 1998). The main problem is that the applications run by the end users are unknown and their characteristics are always more or less guesses. It has also been shown that typical workloads do not exist and the characteristics of different applications are contradictory.

Designing functionality as software

In a computer-based system the system functionality is implemented as a software system. Two fundamental abstractions in software systems are processes and operating system (Wolf 2001). Process is a unique execution of a program and operating system provides mechanisms for sharing the computer resources for the processes and for communication between processes. Typically, the software systems are so complex that models and concepts that are significantly more abstract are needed (McDermid 1991). Three basic problems have to be solved during the design. First is the *formulation of the problem*, which is done during the requirement analysis and definition of the software architecture. Second is the *design of implementation*, which means selection of algorithms, operating systems and programming languages, and the actual coding. The third is *mapping of functionality* onto the hardware, e.g. development of hardware-dependent code, compiling and testing.

The software architecture is defined as a structure or structures of the computing system, which comprises software components, the externally visible properties of those components, and the relationships among them (Bass et al. 1998). The software architecture model by Kruchten (1995) includes several structures, which address different sets of concerns. It is divided into the logical, process, physical and development views. The functional specification gives a logical view of the software architecture. It is a static model of the system. A physical view is needed for mapping functionality into the hardware architecture. In the process view the objects are mapped to processes and threads, which are lightweight processes. The physical view and the process view together create a runtime view of the system. Finally, in the development view the software is divided into modules that represent the actual code blocks. The methods for the evaluation of the software architectures aim at identifying potential risks and validating the quality attributes (Dobrica & Niemelä 2002). The most common approaches are scenario-based questioning, reasoning and measuring.

The implementation of functionality starts with a more detailed design. We have to define a more detailed structure of the software, and to identify solutions to design problems. The operating systems and other middleware services define the basic operational principles of the system. The design patterns define how different general-purpose problems should be solved (Gamma et al. 1995, Tichy

1997, Buschmann et al. 1996). The selection or design of algorithms, data types and data structures specifies in detail what must be implemented (Aho et al. 1983). The final coding must be done using some programming language. There are a number of alternatives and the selection and feasibility of the language depends on the design goals. Different languages emphasise different issues, such as portability, compatibility, security, data processing, parallelism, etc. Kale (1998) gives an overview of the available languages and their use.

The mapping of application code to a processor is done automatically by a compiler, but the applications also use system services that are hardware-dependent programs provided by operating systems and hardware drivers. The most important tasks of RTOS are process management – e.g. process control and scheduling – interprocess communication, memory management and I/O management.

A compiler is a program that generates machine executable code from a source code representation (Aho et al. 1986). The compiler maps the source code functionality into data addresses and instructions that are eventually decoded in a processor to signals controlling the data flow. The compiler is thus partly responsible for producing efficient code that uses the hardware resources efficiently. Therefore, different code-optimisation functions are critical parts of the compilers (Muchnick 1997). The performance of the final program depends on the compilation quality and most of the time-critical functions, such as interrupt handlers, or functions that require special processors, such as DSP processors, are hand-coded in assembly language.

2.3.3 System-on-Chip design

The *embedded system design* was the predecessor of codesign. Early embedded systems were based on COTS components, but very soon complete computers called microcontrollers were integrated on single chips. The main design problems considered performance and memory limitations, and, because of fixed architecture, there was no need to consider software and hardware together before the integration phase. The system-level modelling (Zave 1982, Ward 1986, Harel 1987, Rumbaugh 1991) and experiments with hardware/software system simulators (Smith et al. 1985) initiated the codesign ideas.

In the late 1980s it became possible to develop and implement computer organisations and architectures using ASIC design methods. This introduced new possibilities for system optimisation, because hardware/software partitioning is more flexible and not constrained by costs, as it was earlier. The first System-on-Chip design flows were based on an embedded system design, as depicted in Figure 5. The design started with a functional specification that was partitioned to software and hardware that were designed separately. The partitioning was based on a fixed or predesigned architecture template, and the results were verified using cosimulation.

The first *software/hardware-partitioning or cosynthesis* approaches focused on improving the characteristics of algorithm implementations. The COSYMA approach developed by Ernst et al. (1993) had a C-code as a starting point and moved code segments to the hardware in order to meet performance requirements. The VULCAN approach by Gupta and De Micheli (1993) had an opposite approach; the functionality was modelled with HardwareC and functions were moved to the software in order to reduce cost.

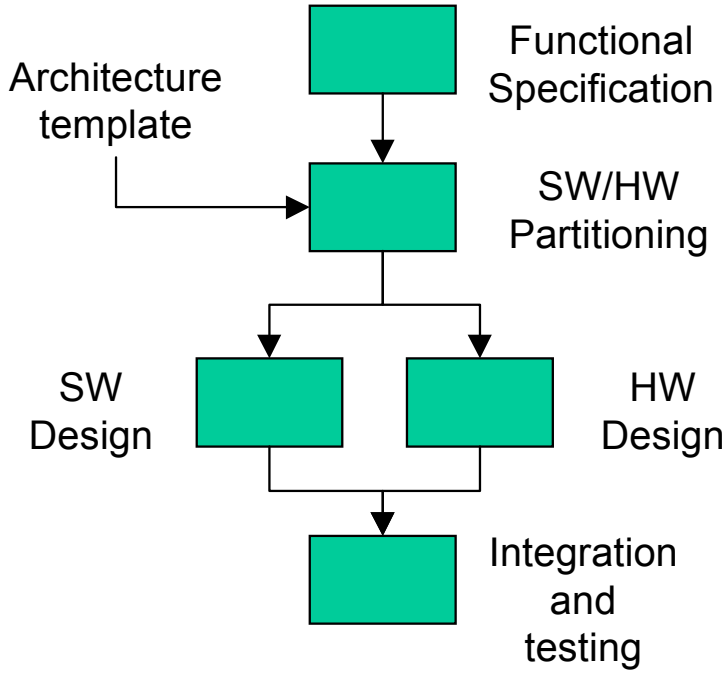


Figure 5. Specification-oriented SoC design flow.

The codesign of instruction-set architecture was an alternative to coprocessors, which first lead to the codesign of application-specific instruction-set processors (ASIPs) and later to reconfigurable processors (Heinrich et al. 1997). The objective was to improve the performance of the processor architecture for a given algorithm. The optimisation was done from application programs to operating systems and instruction-set definition.

The need for considering complete systems rather than algorithms required implementation-independent functional specifications, system-level optimisation and functional verification, which resulted in the *SW/HW codesign approaches* (De Micheli 1994). The SW/HW codesign was defined by the IEEE DASC study group as a design methodology supporting the concurrent development of hardware and software (cospecification, codevelopment and coverification) in order to achieve shared functionality and performance goals for a combined system. The more detailed specification-oriented codesign flow showing the main phases and activities is presented in Figure 6. Simulations at different abstraction levels are needed for the validation of performance. Estimations and analyses are need for the making of the design decisions.

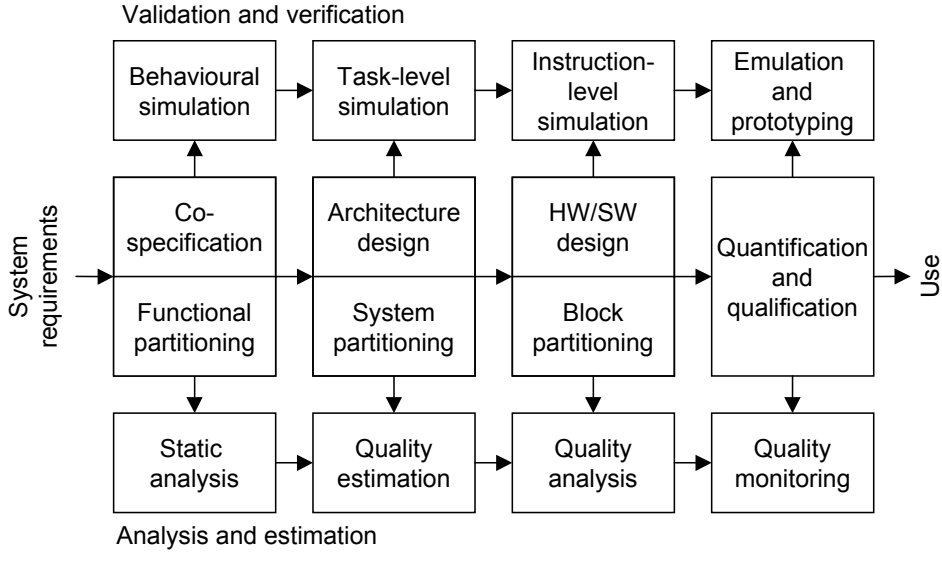


Figure 6. SW/HW codesign process (Soininen et al. 1998a).

Different aspects of SW/HW codesign have been reviewed by De Micheli (1994), De Micheli and Gupta (1997), Ernst (1998) and Wolf (2003). The functional specification can be either *homogeneous*, where a single system-level language is used (Kumar et al. 1996, Ismail et al. 1994) or *heterogeneous*, where the specification consists of several application-specific languages (Buck et al. 1994, Madsen et al. 1997). *Granularity* of language defines the sizes of the objects that are mapped to the technologies. Approaches using fine-grained (Barros & Rosenstiel 1992), medium-grained (Ernst et al. 1993) and coarse-grained (D'Ambrosio & Hu 1994) objects have been presented. The architectures have been based on *coprocessors* (Ernst et al. 1993) or *communication synthesis* (Daveau et al. 1997). Approaches that aim at optimising memory systems (Li & Wolf 1999, Panda et al. 1999) or communication (Ismail et al. 1994, Madsen et al. 1997) have also been proposed. System partitioning objectives have been described using *cost functions*. Typical criteria have been performance, speed up, utilisation, and other constraints, such as cost or power consumption (Fornaciari et al. 1998). In addition, more generic quality criteria have been proposed by Jóźwiak (2001). *Partitioning algorithms* are typically either constructive or iterative. The problems with design space exploration are the number of design alternatives, the accuracy of implementation estimates and the exploration time limitations. The partitioning generates software, hardware and interfaces that must be designed or generated, and their operations must be scheduled and synchronised. *Scheduling* is a widely studied topic and both instruction-level and process-level approaches are needed in the codesign system. Different partitioning and scheduling approaches are presented in De Micheli & Gupta (1997). The approaches that try to automate the whole codesign process are called the *system synthesis* approaches (Chou et al. 1995, Madsen et al. 1997, Gajski et al. 1998, Eles et al. 1998).

The increasing complexity of the designs demanded efficient reuse and more reusable hardware implementations. In the mapping-based approaches the idea is to evaluate different mappings of functions to architectural elements. These *function/architecture codesign* approaches allowed the use of existing architectures, or separately modelled architectures, and the role of quality evaluation increased (Balarin et al. 1997). The intellectual property (IP) blocks and virtual components were proposed as a solution to design productivity problems in the late 1990s. The virtual socket interface alliance (VSIA), which

defined the concept of virtual components and virtual component exchange (VCX), has developed the required business practices. The IP-block concept contains both the functionality and the implementation. In *IP-based design* the main problems are related to the evaluation of IP-block quality and integration issues (Keating & Bricaud 1999). Gajski et al. (1999) have proposed an IP-centric embedded system design methodology, where the major challenges are the interface synthesis among the various IP blocks and system verification. This SpecC approach integrates IP-blocks into a specification-oriented design flow (Gajski et al. 2000). The SystemC approach relies on mapping and performance simulations (Grötker et al. 2002).

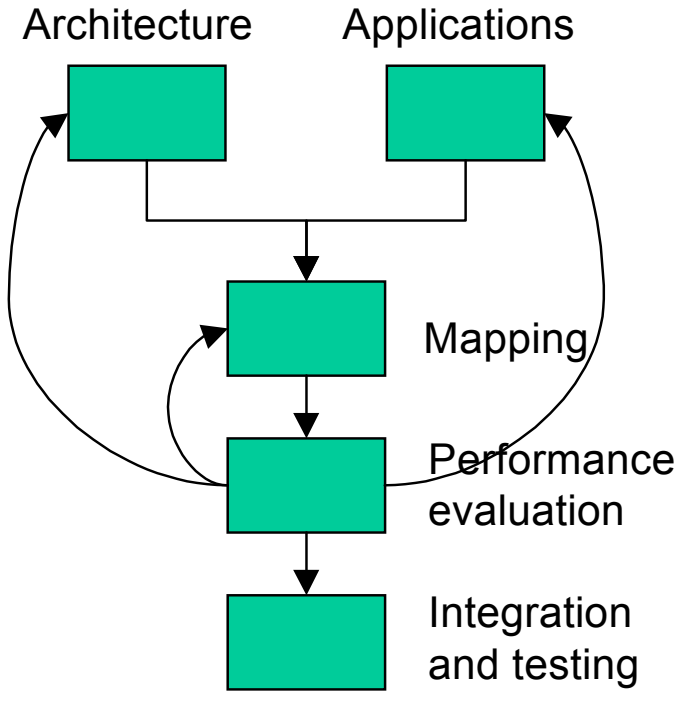


Figure 7. Mapping-centric SoC design flow.

Platform-based design integrated and extended the earlier methods by reusing system architectures and topologies in addition to components. Chang et al. (1999) divided the SoC design into block authoring and SoC integration, and introduced three levels of platforms. The platform was defined as an architectural environment created to facilitate the reuse. The first level was the IC integration platform, which consisted of high-value general-purpose IP cores,

such as processor cores, memories and buses. The second level was the system integration platform, which was process-specific and contained application domain-specific IP blocks and support for embedded software architecture. The third level was the manufacturing integration platform, which was a fixed hardware chip except for FGPA and processor programming. Ferrari and Sangiovanni-Vincentelli (1999) define platforms as precisely defined layers of abstraction through which only relevant information is allowed to pass. The platform design consists of application domain-oriented platform definition and selection of platform instances The designs implemented on top of the platforms are isolated from irrelevant system details (Horowitz et al. 2003). Keutzer et al. (2000) introduce a software platform on top of the hardware platform and tackle the complexity problems by separating functions from architectures and communications from computations. The software platform is an application programming interface and an abstraction of a multiplicity of computing resources and device drivers. In the MESCAL approach the focus is on the simple programming of the platforms (August et al. 2002). The Metropolis environment (Balarin et al. 2003) is based on a metamodel that supports functional capture and analysis, architecture description and mapping, and an application-programming interface, which enables tool integration and the use of different design languages.

The verification of a partitioned system requires *cosimulation* tools, which can simulate both software and hardware (Staunstrup & Wolf 1997). System verification has two main approaches. Homogeneous models are simulated using one or more simulators that communicate with each other. In the heterogeneous approaches different parts of the system are modelled using different languages and abstraction-levels and then simulated or executed using language-specific tools. The problem in coverification is the differences in time scales. In hardware the basic time-step is in the nanosecond range; in a typical software operating system the time-step is in milliseconds. One has to either abstract the software execution, e.g. to model the processor hardware at a higher abstraction level than the rest of the hardware, or abstract the hardware. Commercial cosimulation environments are mainly used for verification of interface functionality. Hardware emulator-based approaches provide more performance, but they also require almost-final designs.

2.4 System-level design methodologies

The system design methodology and design flow consists of a set of activities that have to be performed. A good methodology helps to solve the most essential problems. Its purpose is to partition the design problem into manageable tasks and define the tools and practices for those tasks.

The actual design problems depend on the type of system; thus the methods and tools for executing these activities are system-specific. For example, the exploitation of the capacity of the physical channel is very important in telecommunication systems and is determined by the quality of the algorithms. Therefore, the development of methods focuses on the algorithm modelling, specification and performance evaluation. In software development the interfacing with a variety of hardware platforms and operating systems, and the management of the complexity puts high demands on system modelling and analysis. Safety is essential in life-critical systems and the quality of the verification of implementation is important.

Another dimension is the principle on which the systems are developed. Martin (2002) presents six scenarios for system-level design. The classification is based on IC technology and the EDA industry's view. The basic differences in the approaches are in the architectural template and the usability of the technologies and tools. The most commonly used approach is to organise the work and workflows as the basis for the methodology and design process modelling.

The *waterfall model* divides the software development process into the requirement analysis, architecture design, coding, testing and manufacturing phases that are executed sequentially in a top-down fashion (Royce 1970). In the *V-model* the role of testing and verification is emphasised. Each phase of the waterfall model is accompanied by the respective validation or verification phase (McDermid 1991).

The *spiral model* is based on the idea that the phases of the waterfall model are repeated for different versions of the system (Boehm 1988). These sequences of versions include conceptual models, specifications, prototypes and product versions. This successive refinement approach is also called the *incremental*

development approach, because the idea is to bring in more details during the development.

The *meet-in-the-middle* principle combines the top-down and bottom-up approaches (De Man et al. 1990). The idea is to define the primitive components and objects and to develop them and the system structure simultaneously, so that the final system is constructed from these primitives in the middle of the design process. ASIC synthesis tools with libraries and IP-based design are derivatives of this principle. In the *hierarchical design* methods the system is partitioned to subsystems that are designed separately.

Concurrent engineering attempts to take all the aspects of system design into account from the very beginning of the development process. It aims at developing subsystems simultaneously and increasing the communication between different design teams. Cross-functional teams and information sharing are the main enablers (Linton et al. 1992).

In *product family development* the idea is to simultaneously develop a complete set of product variants (Wheelwright & Clark 1992). These approaches are based on the ideas of mass customisation, development of a platform product that is easy to modify, or development of a core product that can be extended by product features.

2.5 Quality validation

In product development the design methodology is responsible for product success in the sense that it must guarantee that the company performs the activities that are needed in quality validation. In the requirement analysis phase the system requirements should be transformed into quality criteria that can be used in the validation of design tasks. System quality is a broad topic, as depicted in *Figure 8*. When considering complex computing systems, the three most important quality dimensions are performance, cost and variability, but their importance is naturally dependent on the product and its services.

Figure 8. System quality criteria.

The quality characteristics in *Figure 8* are very generic and cannot be directly used in product development. The task of the system designer is the transformation of user needs to physical requirements in different abstraction levels and the partitioning of those needs and requirements to subsystems and technologies.

Different abstraction levels of product models of embedded computer-based systems have different quality criteria. If we think about the main abstraction levels then the quality criteria such as performance, power efficiency, area, etc. are mostly dependent on technologies that are chosen for the implementation. The computational capacity, utilisation of system resources, modularity, etc. are typically important when considering architecture models. In communication systems, for example, the functional performance depends on chosen algorithms and their characteristics. The customer satisfaction or product's success in market are measures of quality, when complete products are discussed. It is important to notice, however, that the quality criteria of different levels are not independent from each other. The customer satisfaction, for example, results from the decisions that have been done related to algorithm, architecture, implementation style, and technology selections.

The main problem is how to transfer the customer satisfaction-type of criteria into the technical requirements. Techniques such as the Taguchi method and quality function deployment (Ross 1988, Day 1993) try to solve this problem by introducing loss functions and correlation matrixes for quality characteristics and technical requirements.

2.5.1 Performance evaluation

Performance, P is defined as $P = C_s \cdot WL$, where C_s is the computing system and WL is the workload. So, the performance is always dependent on the computing system and the workload, but the choice of performance criteria is free. Typical performance metrics of the computer system are the execution time and power consumption (Hennessy & Patterson 2003). Only the execution time of the workload is considered in this chapter, but other criteria such as latencies or capability to meet deadlines could be similarly analysed. The computing system can be a complete system, a computer system, or a processor. With the complete system, the workload is the final code and use case and the performance is basically the systems capability to process events or data, which results from the real execution time taken by the application functions. With computer systems, the workload tries to emulate the real applications. The performance is then the average processing capacity, which is typically expressed as millions of instructions or operations per second, MIPS or MOPS. With processors, the focus is on the architectural performance, which is typically measured as clock cycles per instruction, CPI, or its inverse, instructions per cycle, IPC (Bose & Conte 1998, Flynn et al. 1999).

Performance evaluation methods can be classified into three main classes: analytical methods, simulation methods and measurement methods (Jain 1991).

Analytical methods are based on a mathematical model of the system and workload. Markov chains, queuing models, Petri-nets, etc., are typical examples (Krishna 1996). Analytical performance modelling and simulation are only applicable to the earlier stages of design because of the complexity problems (Heidelberger & Lavenberg 1984). Capacity planning – especially determination of computing, caching and buffering capacity – I/O design, scheduling decisions and helping in resource organisation design are typical tasks. Analyses are

typically based on solving the equations, but, in many cases, simulation is also used as a supporting tool.

In the simulation the execution of a workload is simulated using a computer program. In hardware simulation the behaviour of the hardware elements is simulated. In functional simulation only the functionality is simulated. Typically, instruction-set simulators (ISS) are used as simulation engines. In performance simulation only the effects of the workload are simulated. The simulation can be a stand-alone computer program or it can be based on a simulator kernel, which can be further supported with a hardware accelerator.

The monitoring or measurement-based approaches need working prototypes or final hardware. The working prototype need not run at the final speed, so it can be based on FPGAs or emulators. The basic technique is that the prototype or product is instrumented so that performance data-related information can be collected during the execution of the workload. It is possible to apply the same workload generation approaches to the measurement-based approaches as in simulation.

In the simulation and execution-based approaches the workload can be based on several alternatives. In the execution-based simulations the workload is an executable program (Rowson & Sangiovanni-Vincentelli 1997) or execution trace of the program (Lahiri et al. 2000). The executable program can be a real application or a benchmark program, which can be simple instruction sequences, instruction mixes, program kernels of widely known programs, or synthetic programs that try to mimic different types of instruction sequences (Guthaus et al. 2001, Lee et al. 1997, Skadron et al. 2003). Krishnaswamy & Scherson (2000) unified coarse-grain and fine-grain benchmarks into an evaluation technique that uses a performance vector, where each component presents a speed of a dedicated subsystem. The performance vector is computed from the benchmark execution results, where a representative set of benchmarks is required. The approach speeds up the evaluation time when compared with simulations, but it requires an existing processor or processor model in which the benchmarks are run. The main benefit is that details of the processor architecture behaviour can be analysed using larger coarse-grain benchmark programs.

If we compare the accuracy of the results, it is clear that execution and simulation provide the best results. In the functional simulation the instruction-set simulators are typically cycle accurate, but there can be a lot of variety in the accuracy of the functional models of HW blocks. The accuracy of the performance simulation totally depends on the accuracy of the workload and architecture models. It is a trade-off between modelling costs, simulation speed and accuracy. Analytical approaches are usually based on very abstract architecture models, but the cost is the accuracy of the results.

The cost of performance evaluation depends on the maturity of the functions and resources. Performance evaluations using analytical approaches and performance simulations that utilise existing simulator kernels are relatively simple to conduct. With functional simulations and hardware simulations, the design cost starts to dominate very rapidly. If the design is based on existing architectures, where simulation models are available, these effects are naturally smaller, but they do still exist. Monitoring-based approaches are only economically feasible when a prototype exists. Even the instrumentation costs may become significant, especially if we need to collect information on the internal behaviour of the hardware system.

2.5.2 Estimation methods

Estimations are needed when making decisions about a system in the early phases of design. Complexity estimations are needed, especially when feasibility is considered. Performance estimations are mainly used when choosing the implementation techniques. However, the estimation metrics are similar in both cases. The purpose is to estimate the cost in terms of area, code size, complexity, power or design effort, or time in terms of physical time or number of operations.

Hardware estimation

The methods that can be applied in architecture evaluation vary depending on the type of target architecture. Complexity estimation of algorithms is used when designing dedicated hardware implementations. Ward et al. (1984) present the basic concept of figure of merit and a system cost function. This extends

operational complexity-based methods into an estimation of hardware complexity that takes both control functions and address generation into account.

Estimation of hardware performance and cost is a critical part of the hardware synthesis system. There are two main approaches. The first is to assign cost values for system functions. The second approach is to use synthesis tools without going into detailed time-consuming optimisations. For example, Jain et al. (1992) present an estimator of area-delay curve that is based on the data flow graph properties and cost characteristics of library components. An estimator of hardware resources or performance has been presented by Sharma and Jain (1993). Mitra et al. (1993) have presented an estimator for FSM complexity. The estimator analyses the structure of FSM and evaluates the state encoding optimisation and minimisation possibilities. Henkel and Ernst (1998) use path-based estimation in a cosynthesis system, which takes both controller and data path parts into account. Salchak & Chawla (1997) take the design complexity issues into account by analysing the structure and relationships of VHDL code.

Software estimation

The idea in software estimation is to analyse the running time or execution time of the program. The running time depends on the input data set, the quality of code generated by the compiler, the characteristics of the computer, and the time complexity of the algorithm.

At the algorithm level both the compiler and computer are unknown. Therefore, time complexity $O(n)$ can only be presented as a function of the size of the input data set (Aho et al. 1983).

At the specification-level the execution-time information is added to the specification language objects, and the overall performance is estimated from the specification structure. In the POLIS approach (Suzuki & Sangiovanni-Vincentelli 1996) typical execution times of nodes in s-graphs or CFSM specifications are obtained by running simple benchmark programs in a target processor that has similar instruction mixes as the C-code generated from the specifications.

At the source-code level the idea is to use the structure of the source code as a basis for the estimation. Puschner & Koza (1989) introduced new language constructs for the specification of the execution bounds of program segments. The worst-case execution time is then analysed from the extended source code. Engblom et al. (1998) map program execution information to the source code level from the compilation process. The approach takes the effects of compiler optimisations into account but still enables source code analysis techniques. Brandolese et al. (2001) combine source code analysis, timing information of elementary operations and profiling data with statistical measures of source code characteristics.

At the compiled code level it is possible to take the effects of the compilation process into account, and, especially, the target-independent optimisations. Malik et al. (1997) divide the analysis problems into path analysis and resource utilisation analysis techniques. Their approach is based on using graph techniques and the statically analysed execution bounds of basic blocks. Gong et al. (1994) have used three-address code representation and a generic estimation model. The program path is based on the branching probabilities. The estimator computes such software metrics as execution time, program memory size and data memory size based on the generic instructions and technology files of the target processors. The technology files contain the execution times of the generic instructions. The approach was extended for application-specific instruction set processors having VLIW architectures in Gong et al. (1995). Wolf & Ernst (2000) extend the basic blocks into program segments and take the execution context into account. The execution cost of the segment is determined using instruction execution costs or simulation with known input data and a cycle-true processor model. Lăzărescu et al. (2000) use an assembly-level C-code containing the timing information that is generated from the optimised assembly-code in their cosimulation approach. The idea is that the target compiler optimisations can be taken into account in the host-based simulation. Giusto et al. (2001) present a statistical approach for the generation of the instruction's timing information. In the TOSCA approach the software execution time is estimated by calculating the average process execution times (Allara et al. 1998). First, minimum and maximum values are generated using the compiler and unbounded and highly restricted processor architectures. Then the average values are calculated using characterisation of the target processor-compiler pair with benchmark programs.

System estimation and mappability

The traditional technique for the analysis of a system consisting of a computer and software is cosimulation. The cosimulation environments can be characterised according to the abstraction levels of the hardware models, interface models and software models. The software can be executed using real processors, emulated processors, RTL-simulation models of the processors, instruction-set simulators of the processors (Hughes et al. 2002, Austin et al. 2002), or more abstract functional simulation models (Chandra & Moona 2000). Similar abstraction levels are available for the other hardware parts. In the processor-hardware interface bit-accurate models, bus-accurate functional models and abstract communication channels have been used (Benini et al. 2003). The software itself has been compiled to the target processor or the host processor. In order to speed-up the simulation performance, parallel computers or hardware accelerators have been used.

When cosimulation is used for system-level estimation, the problem is the trade-off between modelling effort, accuracy and performance. Complex software systems cannot be simulated using a clock-accurate processor or bus models. There are three basic approaches for solving the performance problems. The first is to increase the abstraction level of the hardware simulation by, for example, using concurrent processes that communicate using channels (Buck et al. 1994, Grötker et al. 2002). The second is to use host-based execution of the software instead of trying to execute machine code instructions (Živojnović & Meyr 1996). The third is to replace the final code with a more abstract workload model that only generates events for the hardware architecture instead of implementing the complete functionality of the software (Lahiri et al. 2000). The abstracted code can be based on the software estimations presented earlier or statistical workload generators, as in network simulations.

Mappability estimation means the evaluation of the potential quality of the processor architecture-algorithm combination. This differs from the cosimulation-based approaches in that the idea is to estimate the potential quality by comparing the application and architecture characteristics. In cosimulation the idea is to study the quality of the combination using abstracted models and simulation. In the mappability estimation the quality of mapping process, e.g. how the optimal implementation can be designed, is included in the estimation.

Mappability estimation is an emerging topic, because it is most feasible when both the architecture and application can be affected, which is possible in SoC designs. Retargetable compilation (Bhartacharyya et al. 2000) and estimation (Ghazal et al. 2000) have been used in estimation of DSP architecture performance. In the approach by Ghazal et al. (2000) the parameterised architecture model allows study of the potential of the architecture. The idea is that aggressive compiler optimisation techniques are applied to application code using the different architecture parameters. The attempt is not to produce executable code but to study the potential consequences of the optimisations and the potential mapping quality.

Carro et al. (2000) calculate application profiles (control, memory and data) using target-independent three-address code and virtual machines for microcontroller, RISC and DSP types of processors. The real processors are modelled using the relative costs of different types of instructions that are analysed using the virtual machine of a processor that is based on the processor characteristics, e.g. the main parameters. The applicability of a processor is analysed by calculating the application performance distances of the processor parameters and the application profile values.

Sciuto et al. (2002) uses affinity values for determining most suitable processing element class for each system functionality. The possible processing element classes are GPP, DSP and ASIC-like device architectures. The affinity value provide quantification of a matching between structural and functional features of possible implementation. The affinity value is based on data involved in the execution of functionality and on its structural properties.

Gupta et al. (2000) have developed a two-stage processor evaluation method for embedded system design. The first stage is a processor selector that extracts the application parameters and compares them with the processor description. The parameters, such as average block size, number of MAC operations, ratio of address and data computation instructions, ratio of I/O and total instructions, average arc-length in data flow graphs and unconstrained ASAP schedule analysis results, are related to operational concurrency. The second stage is a code-size and performance estimator, which creates a cycle estimate based on simplified list scheduling of the algorithm using the target architecture parameters.

3. Architecture design challenges in future SoC-based systems

New approaches for architecture design are needed because the complexities of user needs, algorithms and technology capability are increasing more rapidly than design productivity and verification capability (ITRS 2001), as illustrated in Figure 9. The architecture design method mostly affects design productivity, but its role is to narrow all the gaps in Figure 9. The architecture can enable better exploitation of the silicon capacity and may simplify the verification burden. However, it also affects how the user requirements and algorithms can be implemented within the economic constraints. Essentially, from the technology and implementation perspective, the problem is how to divide and separate the system into manageable units. From the product requirement perspective, the problem is how to join or link the variety of needs and alternatives so that the right choices and selections can be made.

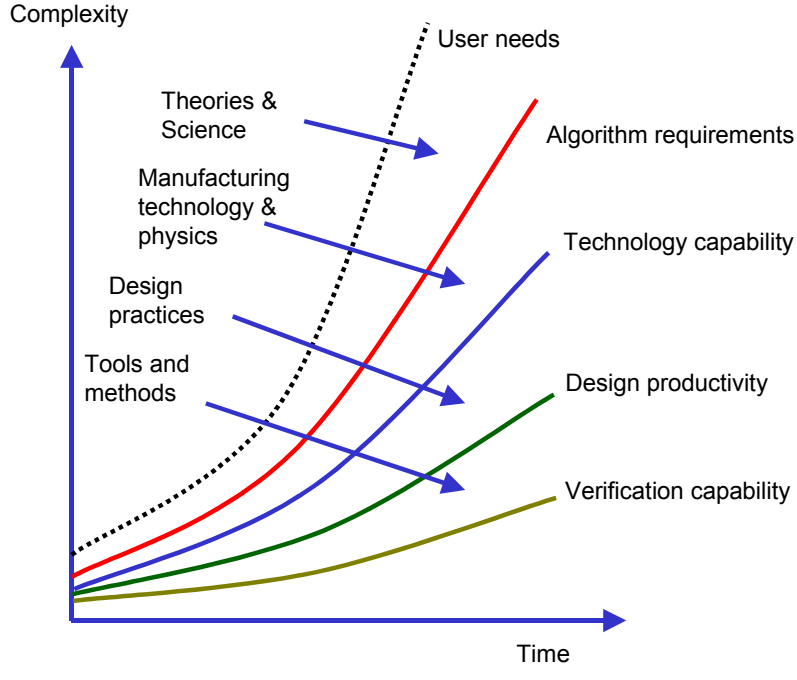


Figure 9. The development and relationships of relative complexities of user needs, computational requirements of algorithms, implementation capability, design capability and verification capability as a function of time.

3.1 Technology capacity

The number of transistors in a single chip is going to increase for the next ten years. With 65 nm technology the number of transistors will exceed 2.4 billion. The maximum internal clock frequency will increase to the gigahertz range (ITRS 2001). This will cause serious problems for current system architectures and design styles because they do not scale up to such dimensions and complexities.

If we consider the available silicon surface and the characteristics of the gates and wires from the digital design view, it is clear that the area that is reachable during a single clock is far less than the chip area. Therefore, we have to partition our chip area to several clock domains that communicate with each other asynchronously. Similar partitioning is required for testability, energy distribution and communication reasons. We need to reserve a part of the chip area for this kind of infrastructure and to partition the rest for application purposes, where each part has its own synchronisation, as illustrated in Figure 10.

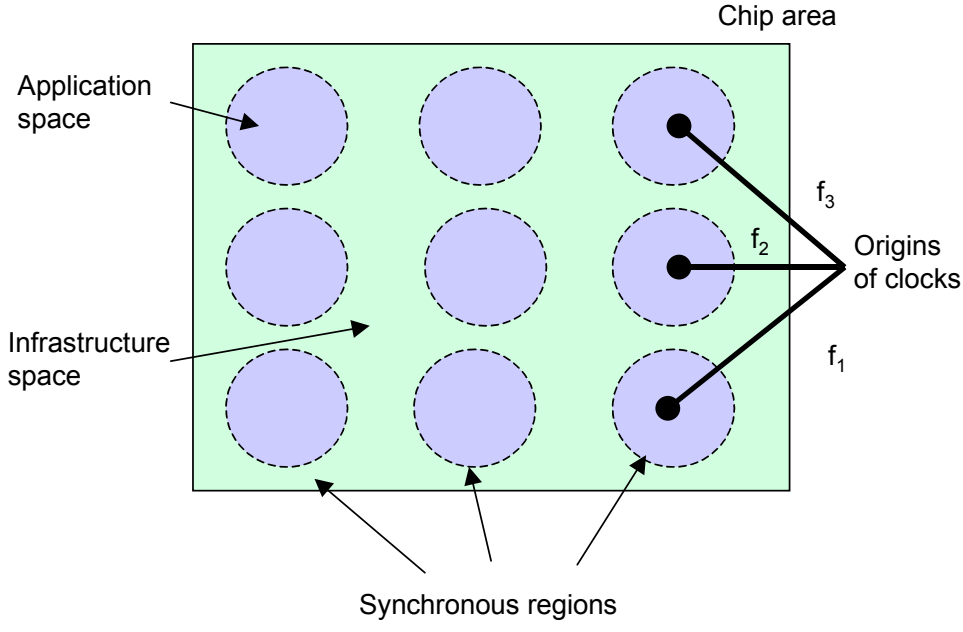


Figure 10. Partitioning of chip area to applications and infrastructure.

The silicon surface is a high-capacity computation platform that can be exploited in different ways. A sequential processor cannot exploit such a capacity effectively because of synchronisation problems. Operation-level parallelism leads to an extremely complex, expensive and time-consuming design of dedicated hardware. Instruction-level parallelism (ILP) leads to extremely complex control of execution, and, besides, it is difficult to extract enough ILP from large applications (Jouppi & Wall 1989). Thread-level parallelism could be a viable alternative, but it requires significantly more threads than in current processors (Forsell 2002b). Task and process-level parallelism leads to parallel computer architectures, which have problems related to shared memories and communication, but these architectures could exploit the area more effectively. Besides, the programming model would be simple. Application-level parallelism means integration of embedded systems into a single chip and a design approach that resembles the distributed system design. Such architecture could exploit all the possible areas, but the feasibility requires that the ratio of local and global computations is high.

Both integration and effective use of different implementation technologies must be supported by architecture design methodology. Technology embedding means additional costs in the ASIC manufacturing process, but the use of different technologies can also improve the final product characteristics. Advanced memory technologies and reconfigurable arrays or areas are good examples. The linkage between the costs and benefits of the technologies and applications must be created in the design process.

3.2 Product requirements and economics

Complex SoC projects stretch the capabilities of companies. Managing functional diversity and complexity are product requirement-related issues that must be addressed by system architecture and design methodology. For example, personal communication devices will have both local and mobile communication capabilities, various types of multimedia features, intelligent user interfaces, etc., as depicted in Figure 11. Good implementation will require various types of technologies and computing architectures, which means that the implementation of total functionality may consist of a set of optimised subsystems that are integrated by common functional “glue”.

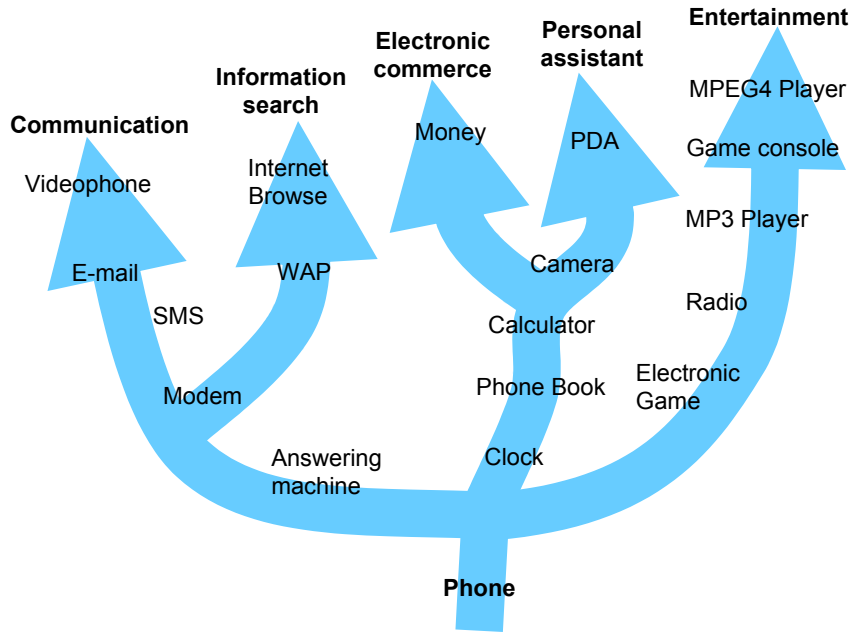


Figure 11. Diversity of services in mobile terminals.

Commercial products must meet both the market window and target cost constraints. Design productivity and design reusability are the critical parameters in this respect. The cost/object can be used as measure of effectiveness. If we analyse the cost/transistor in different types of designs, it is easy to see that dedicated logic is more costly than IP-block-based design or memory design, which benefits from regular and repeatable blocks. If we use cost/function, the software approaches are better than the reconfigurable or hardware approaches. This means that we must favour flexibility over fixed solutions and regularity over ad hoc solutions.

The cost of design will be such that design reuse must be exploited at all levels and reusable units must very large. In addition to IP-block reuse, we have to reuse the computation architectures, network architectures, software platforms, existing chips and design methods. This means that we have to have very clearly defined interfaces for connecting resources to the infrastructure and for using the resources with the hardware or software. The reuse of architecture requires

means to modify and configure the architecture model so that application area-specific platforms can be designed. The reuse of a complete chip needs standardised application mapping procedures that hide the architecture and hardware details from the application designers.

3.3 Management of diversity and complexity

The product development comprises several areas of expertise and, in the decision-making process, we have to be aware of the effects on all aspects of the design. When we start to develop our ideas into products, the number of details increase rapidly and the design methods and practices become diverse, as in Figure 12. Working simultaneously in multiple technologies requires that we partition our system functionality to the technologies. Partitioning requires that we have portable models for validation and estimation that can be transferred between technologies and subsystems. We must also derive quality criteria for each abstraction level from the quality criteria of the product. Then we have to make a trade-off between different technologies and optimise the complete system instead of subsystems. This means that we need analysis and estimation methods that can be used in early phases of the design and that take into account the couplings between the quality characteristics, the partitioning, the functions and the technologies. Naturally, we have to validate our decisions at various levels of abstraction. This needs an ability to combine different models. Finally, we have to be able to use technology-dependent methods and tools. This requires encapsulation and interfacing capabilities.

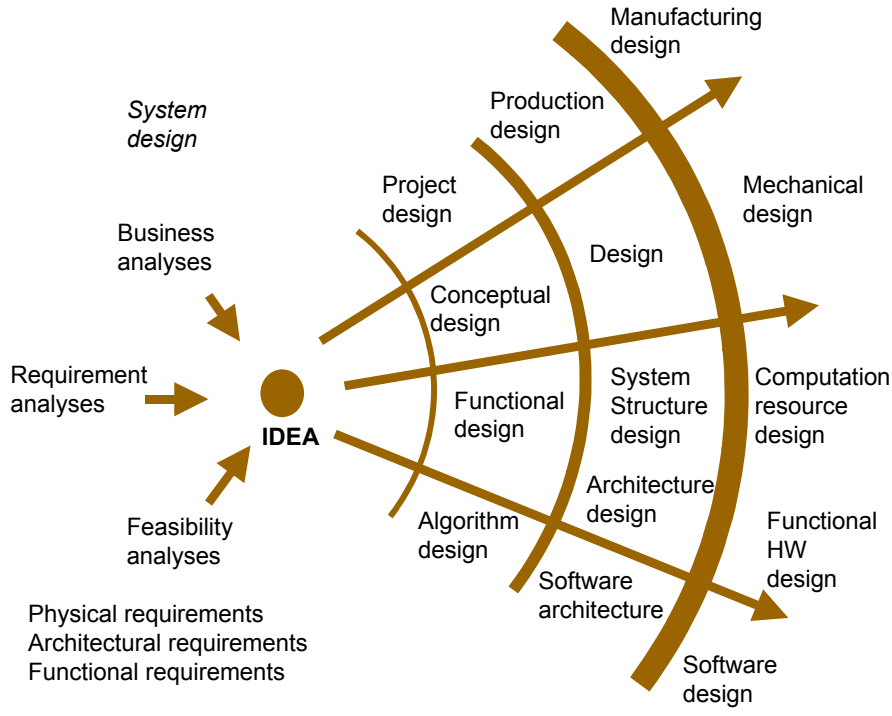


Figure 12. Diversity in system design.

In addition to the management of work and implementation-dependent issues, business and economic issues also have a significant effect on how we should develop our products. Economical constraints such as time to profit require a reduction in the design effort and design time, which can be achieved with the platform-based approach with reuse, both at the implementation level and the design level. The problem is how to guarantee adequate performance and variability. The market success depends on the product's ability to satisfy the users. User needs are diverse and the trend is to customise the products and services. If it is done with software, we sacrifice the performance features. If we do it in hardware, it is not cost effective. Naturally, the problem is more complex and we should consider the competitive strategy and market situation too.

In design methodology we must find out how to balance the system characteristics. The most important are complexity vs. feasibility, flexibility vs. suitability for purpose, and generic vs. dedicated solutions. These relate to the problem of when and how we should apply reuse instead of design.

Complexity/feasibility relates to the principles on which we design our systems. It relates to both hardware architectures and software architectures. Flexibility issues relate to how we identify and encapsulate those parts of the system that must be optimised without losing the programmability and configurability of our system. The generic/dedicated trade-off is mainly a design effort issue. The problem is the separation of common or generic problems from those that need special methods. The generic problems should be solved so that they become a part of the reusable system infrastructure.

4. Architecture design

The architecture of an application domain-specific integrated computer is a combination of platform, computer, network and hardware architectures, as shown in Figure13. The implementation technology requires using hardware architecture design principles. The target is computer architecture, which is optimised for some specific application domain. Because of the complexities of the implementation and system, it also has to be a platform-type of architecture and it has to contain, or has to be connected to, an on-chip network.

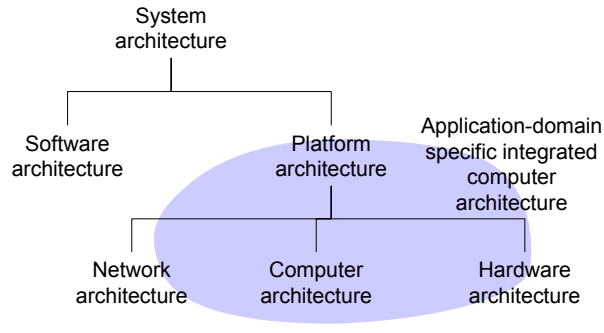


Figure 13. Classification of architectures.

The architecture design is a process that produces the architecture description of the system. The process and the methods used depend on the system requirements and constraints. The approach and methods presented in this thesis; the complexity, mappability and capacity estimations as such can be applied to the design of any application domain-specific integrated computer. However, the real potential can be achieved when the interfaces to complete system development are understood, e.g. how the computer architecture relates to the system architecture, platform architecture and networks, and how an IP block-based hardware design can be exploited most efficiently.

In this thesis the methodology framework for architecture design methods is Backbone-Platform-System (BPS) design methodology that is targeted for the development of Network-on-Chip (NoC) systems, where the total complexity is expected to be above one billion transistors. The NoC architecture that is part of the BPS methodology provides an environment where application domain-specific integrated computers will be needed. It also motivates the reuse of existing

designs, e.g. to use the principles of IP block-based design. Supporting reuse has been the main objective when developing the presented estimation methods.

4.1 Backbone–Platform–System design methodology

The Backbone–Platform–System (BPS) design methodology has been developed based on the "*integrated distributed embedded system*" concept. Management of the diversity in functionality and technologies requires using the principles of distributed system development, which partitions the problems into more manageable units. Management of the complexity requires using integration, reuse and encapsulation, which are common in ASIC and embedded system development. Paper I, Paper II and Soininen & Heusala (2003) give more details about BPS methodology. The purpose in this chapter is to introduce the main principles.

The following four objectives have been considered most important in the development of BPS methodology.

1. Layering is needed for separating different concerns during the system development (Keutzer et al. 2000). In BPS methodology the layers are called the backbone, the platform and the system. The backbone layer is an extension of earlier platform-based design research.

2. Separating infrastructure from applications is necessary for managing the overall complexity. This is an extension of the earlier research, where the development has gone from software/hardware partitioning to function/architecture codesign and to separation of communication and computation (Rowson & Sangiovanni-Vincentelli 1997). The infrastructure includes the principles of basic system construction in addition to support functions.

3. Supporting reuse at all levels is a necessity because of design productivity reasons (Chang et al. 1999). The organisation of work must be based on heavy reuse of existing designs, reuse of software systems and reuse of platforms. The encapsulation of IP and the use of IP as a part of the system are, therefore, critical. Paper III deals with IP block design. Papers IV and V deal with IP block selection. Paper VI considers IP-based system design from the validation perspective.

4. Supporting the diversity in methods and technologies. Heterogeneity has been considered important in earlier codesign approaches, especially in cosimulation (Eker et al. 2003). In BPS methodology the heterogeneity is taken into account both in the system architecture and the supporting methodology integration.

4.1.1 Separation of layers

The purpose of the layered development is to manage the complexity by encapsulating the technology-specific and typically generic issues into a product backbone, and to encapsulate application area-specific issues into a product platform. The product instance-specific issues can then be designed reusing the backbone and platform designs. The layering enables the separation of infrastructure design from resource design and application design. The main problem is that the division between backbone, platform and system does not follow the technology boundaries or even traditional design flow phases; every layer presents a complete system.

The *backbone layer* consists of the infrastructure and communication services. The infrastructure part consists of physical components and wiring for clocking, energy distribution and testing. The communication services consist of physical components, such as channels, switches and network interfaces, and basic services, such as protocols for physical and data link layers. The backbone provides an *integration framework* for platform and resource designers and a *communication model* for application developers.

The *platform layer* describes the computation platform for the target application area and provides a *programming model* for application designers. At the platform layer the hardware of the chip and system services that will the provided for the application developers are fixed. The platform serves as a manufacturing integration platform for system developers.

The *system layer* describes the programmed chip in the final product. The resource allocation, optimisation of network usage and verification of performance and correctness are the main problems that are basically similar to those distributed and parallel system designers have to face.

4.1.2 Separation of infrastructure from applications

The proposed NoC architecture is 2-D mesh, which consists of *resources* and *infrastructure,* as presented in Figure 14. The infrastructure in Figure 14 consists of network elements, e.g. *switches*, *channels* and *resource-network interfaces*, but it also contains other infrastructure services, such as power delivery, testing, etc. The resources are embedded systems integrated into this architecture. Application domain-specific integrated computers are typical examples, but any type of resource that can be implemented in a synchronous area with the silicon technology is possible. From the system perspective, the resources are independent embedded systems that have standard interfaces to the infrastructure services.

The *regions* can be used for embedding different technologies, such as large memory systems, parallel computers or reconfigurable arrays, into our NoC. The region is an area that is isolated from the communication using specific switches, e.g. *wrappers*. The size of the region can be larger than the size of the resource.

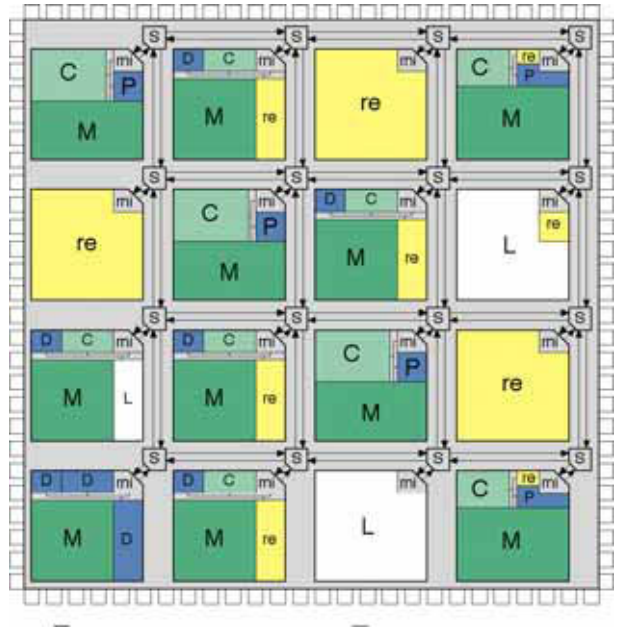


Figure 14. Example of a network-on-chip architecture. S = switch, rni = resource-network interface, P = processor core, C = cache, M = memory, D = DSP core, re = reconfigurable logic, L = dedicated hardware.

4.1.3 Design flow

The overview of the design flow for NoC-based systems is presented in Figure 15. The design flow is divided according to system layers into the backbone design, platform design and application mapping. The blue area in Figure 15 shows the area of the design activities, when developing a NoC system, i.e. a NoC architecture-based product from scratch. The axes in Figure 15 show the completeness of the functions and the resources of the complete system during the design flow.

A generic design activity is divided into analysis, estimation, decision and validation phases. The *Analysis* phase is needed for understanding the input for design decision. *Estimation* is needed for predicting the outcome of possible decisions without putting too much effort into it. *Decision* implements the new system model, which typically includes the refinements and transformations. V*alidation* measures the effects of decisions using the new system model as input. In many cases the validation and estimation methods can be very similar and the differentiating factor may simply be the maturity or abstraction level of the design data.

The *backbone design* combines the technology-dependent implementation and very generic architecture considerations. The analysis phase considers the silicon characteristics and general principles of system construction. The estimation focuses on production yield, power management and infrastructure/application area efficiency. The infrastructure services and network topology, switches, channels, resource-network interfaces and protocols are designed in the decision phase. The quality of these services is measured during the validation phase as simplicity of resource integration, cost and performance.

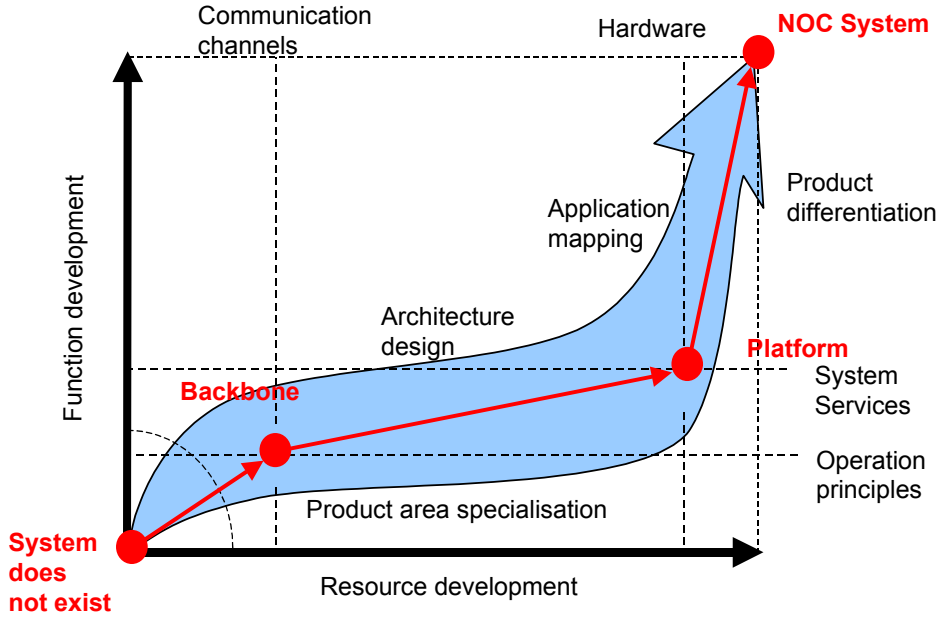


Figure 15. Design flow of NoC-based system.

The *platform design* can be divided into *infrastructure instantiation* and *resource design,* which is the design of the application domain-specific integrated computer covered in Chapter 4.2. The two major decisions in the infrastructure instantiation are the dimensions of the network, and types and sizes of the technology regions. The analysis phase focuses on understanding the characteristics of a typical application workload. The total complexity of computation, communication and storage is needed for scaling, and the characteristics of the applications affect the selection of the technology regions. These network performance and efficiency estimations are needed for the evaluation of the relative costs of the different implementation formats. All the regions and resources are designed and implemented in the decision phase. The validation phase consists of design validation, which is done using performance evaluation methods, and design verification, which is done using ASIC and SoC verification methods. It is clear that the current methods are not capable of handling the complexity of a real NoC platform. Therefore, the critical feature in the selection of methods, tools, and design practices is how to divide these huge problems into manageable smaller problems.

The *application mapping*, the final programming and the configuration of an NoC system consists of four activities. First, we have to model the whole system. Second, we have to partition the functionality into network resources. Third, we have to implement the behaviour. Finally, we have to download the complete system description into the NoC platform and verify it. The analysis focuses on the modelling of the behaviour of the system and its workload for the platform elements. The estimates of system quality are needed for mapping functions to resources. Feasibility of platform for a chosen set of functions, performances of functions with the chosen mappings and utilisation of platform resources are figures of merits that must be considered before committing to implementation design that is the decision phase. Validation and verification of structure and functionality of a complete NoC-based system are problems, where built-in self-test structures and hierarchical testing approaches with separation of communication, data storage and computation are the most likely alternatives. However, these problems are such that the correct answers will only be seen in the future (Vermeulen et al. 2003).

4.2 Design of application domain-specific computer

Application domain-specific computers only make sense if they provide better-quality characteristics for target applications than general-purpose computers. Therefore, the goodness of the combination of application and computer is the main target. The quality characteristics of the computer system can be improved by organisation, operation and implementation. The organisation and operation are factors that depend on the application characteristics.

In the BPS methodology the computing resources have very few preconditions. The area is constrained to a resource area that is the size of the synchronous clocking domain, the resource-network interface has to be implemented according to the backbone specification, and it must be possible to implement the resources with the platform technology. The design of the computer and the processor from scratch is a significant effort. In this approach the objective is to use existing building blocks, such as processor cores, memories and coprocessors. The benefit is the reduced design time, and the cost is likely to be the reduced goodness of the combination.

The main trade-off is between flexibility and performance. The architecture design must be based on application characteristics, which leads to the problem of how to characterise the application so that the essential features are covered without constraining the design space too much. If the application is too dominating, the result will be like a dedicated system that cannot be used in other applications. If the application model is too general, performance or quality targets may be difficult to reach or the result will be oversized for most of the applications.

The design of application-specific computer architecture must produce answers to the following two questions. First, we have to define the kind of objects or basic building blocks of integrated computers, e.g. processing units, memories, and communication channels, we have in our architecture This means that we have to define the types of objects, the number of objects, and the characteristics and parameters of the objects, e.g. sizes of memory blocks, etc. Second, we have to define how these objects are connected to each other. The problem is that the number, types and characteristics of objects and their interconnections depend on application mapping, which is unknown at the design time.

The proposed architecture design method takes a stepwise refinement approach to the problem, as shown in Figure 16. The problem is divided into the technology selection and scaling problem, which forms the *definition of architecture concept*, and the problem of the *selection of the implementations of the objects*. The definition of architecture concept is described in Paper IV, and the selection and validation problems in Papers V–VII. The architectural object selection further divides into the selection and refinement phases, and into processing, memory and communication architecture selections, but only the processing architecture selection is covered in this thesis.

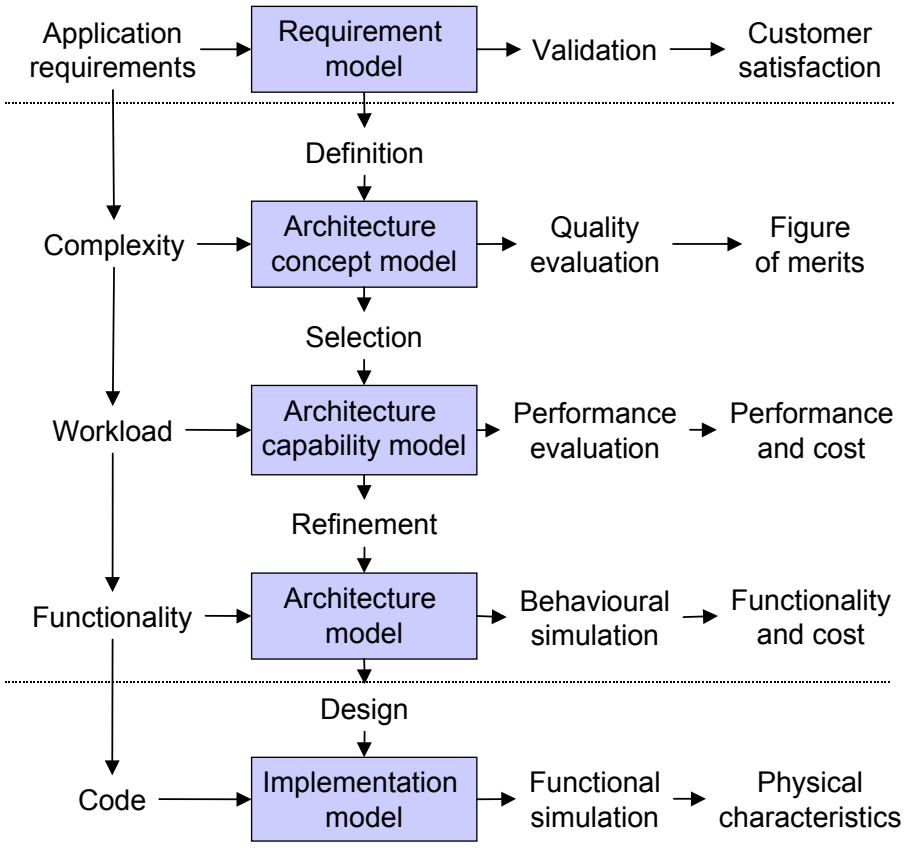


Figure 16. The architecture design flow for application domain-specific integrated computers.

Computer architectures have a large number of quality factors that must be fulfilled. All the quality factors should be taken into account in all design phases The idea in this flow is to enable the evaluation of the architecture quality with the application model that is on the same abstraction level as the architecture model. This also means that the evaluation criteria at each level must be such that it addresses the problems that are tackled by the design decisions in that phase. The proposed architecture design approach has three abstraction levels: conceptual model, capability model and architecture model. We must take care of primary quality objectives, such as minimisation of costs, adequate performance and capacity, or correctness of design, in all the phases, but during those phases we also have other objectives that are to be completely solved. Table 1 summarises some of the most important quality design issues.

Table 1. Quality design issues at different levels of abstraction in the proposed architecture design approach.

	Design objectives	Design decisions	Quality validation problem	Design method
Architecture Concept Model	Minimisation of long-term costs	Technology selections	Accuracy of forecasts	Cost analysis using complexity estimates
Architecture Capability Model	Efficiency and suitability for purpose	Architecture object selections	Number of alternatives	Performance simulations with workload models
Architecture Model	Optimisation of features and correctness of design	Architecture refinement and verification	Complexity of design	Functional simulation with application or benchmark programs

The problem with quality validation is to decide how much uncertainty we can tolerate. Long-term decisions must be based on technology and market forecasts and roadmaps, which are very sensitive to unexpected changes in technology and environment. If we think only about the possibilities of building the computing capacity of a computer, the number of alternative solutions is enormous. A detailed analysis of all of them is impossible in practice. The complexity remains a problem even after the main architecture decisions are made. The optimisation of the computer architecture – so that the final mapping of the architecture and applications during the system design results in adequate quality – must be done using abstracted models. The number of primitive design objects and the complexity of implementation prevent the use of prototypes or detailed simulations.

The evaluation of the architecture quality must take the application requirements into account throughout the design flow. It is also necessary to support both proactive evaluation, which is targeted at decision support, and verification evaluation, which is targeted at the validation of the decisions. In addition, we have to be able to validate both common quality factors and phase-specific quality factors. In the proposed approach this means that the application models used in the evaluation of the architecture quality during the different phases are

complexity models, workload models or functional models. The origin of the model can be a generic benchmark, an application domain-specific representative program or a final application. It is also possible to create the application models using bottom-up or top-down approaches, e.g. to refine models starting from the requirements or to abstract existing code. Simultaneously, the focus of quality evaluation changes from long-tem capacity and cost issues to suitability of processing units, to performance and efficient use of architecture and resources, and finally to the functionality, optimality and cost of the system.

4.2.1 Definition of concept model of architecture

Development of an architecture concept is a long-term strategic task, because one of the most important aims of the platform-based design is to develop platforms that have long lives and high volumes via usability in various product variants and generations. Therefore, there are lot of risks and uncertainty in this task. The quality criteria for technology selections and definition of basic architecture concept are related to the development and manufacturing costs during the whole life cycle of the architecture. These costs are related to technology development and tool development, and to the need for changing, maintaining and modifying the product.

The architecture concept model describes the technologies, and the main architectural objects and their interconnections in the computer. The architecture concept model is a network, in which the nodes are architectural objects and arcs are interconnections. The purpose of the model is to enable rapid evaluation of architecture quality, and to act as a baseline for the selection and design of architectural objects. The architectural objects in the model do not describe the actual, physical blocks. Instead, they are the capacity requirements for some specific types of blocks. Therefore, the basic types of blocks are processing unit, communication channel and memory. The granularity into which the architectural objects are decomposed in the concept model depends on the requirements of the system and on the classes of the reusable objects – e.g. IP blocks, such as processor cores. Therefore, it is also possible to further divide these units into, for example, DSP, GPP, and dedicated HW processing units, or

different bus or memory types. It may be necessary to further split these objects into several physical IP blocks during the design.

The analysis in the concept model definition includes analyses of the characteristics of the target applications and the IP block and technology offerings. The estimation means the complexity and cost estimation of the architecture concepts with possible functional partitions and technology, tool, and design cost development scenarios. The decision means the generation architecture concepts and selection of the most feasible architecture concept. Paper IV covers these issues. Only a limited validation of the architecture concept model is possible. The partitioning of functionality into architecture objects can be validated using performance analyses. The complexity estimates can be validated by more detailed design or more detailed analyses, but that is only applicable to a small set of alternatives. Complete validation of the technology and design cost forecasts is not possible.

4.2.2 Definition of implementations of architectural objects

The second phase in the architecture design is to define how to implement the architectural objects in the architecture concept model. The two alternatives for implementing architecture objects are to reuse existing IP blocks or to design the implementation as a new IP block (Paper III). The principles in both cases are similar. The possible implementations must be first characterised, then evaluated as a part of complete system, and finally instantiated into an architecture model. The difference is that with existing IP blocks we have limited capability to modify the design and the number of design alternatives is typically much smaller. The quality of design is another issue to consider. With IP blocks it is very difficult or impossible to correct design flaws, but, on the other hand, it is difficult or impossible to introduce new design errors too. Cost and availability are also factors to be considered. IP block may be a very cost-effective solution, especially when the complexity of design or manufacturing increases, as is the case with processors and memories. Availability can affect both directions. Novel ideas or standardised solutions may be protected by patents, for example, which means that using them requires the procurement of IP rights and even the IP blocks. Alternatively, the IP is not available.

Object selection

The first task in the definition of the implementation is to select the types of units that will be used in our architecture. The first problem in the selection process is to find suitable candidates for architectural objects and to select the best candidate. The second problem is to verify that the chosen set of objects fulfils the performance and capacity targets of our architecture. In order to do this, we need to have an architecture capability model that can be used as an evaluation platform for the application workload models.

The architectural objects in the architecture capability model are described as a service capability they can give the applications. Service capability can be, for example, processing capacity described as instructions per cycle or data memory transfer capacity described as clock cycles of sequential or random memory accesses. The interfaces in the architecture capacity model are described as abstract channels that communicate via transactions. The main difference is that in the architecture concept model the architecture capacity means requirements, and is expressed in the form of general capacity; in the architecture capacity model the capacity is characterised as the capacity of the chosen unit, and the final performance in the performance evaluation depends on the characteristics of the application workload.

The analysis in the architecture capability model definition phase contains the analysis of the possible architecture objects and application functions. We have to understand the characteristics of the target applications. What kind of computation must we perform in the architecture? What is the structure of the computation? Is it possible to exploit data, instruction, thread or process-level parallelism? We also have to analyse, the kinds of services that are offered by the possible architectural objects. The decision is the selection of an architectural object. This can be an IP block or a set of requirements for implementation design. The goodness of algorithm-architecture pair is essential in the selection of the processing units. Chapter 4.3.2 and Papers V and VI give more details about the mappability estimation that supports those decisions. Estimation of system performance and, especially, of its capability to execute the planned tasks is also needed. The validation of the architecture capability model is done using performance simulations, where application workloads are mapped to capability models, as in Paper VII.

Object refinement

The second task after selection of the architecture objects is that their behavioural models must be selected or designed, and connected to each other so that functional verification of the architecture is possible. The result of this phase is the final architecture model, which describes the functionality and interfaces of the architectural objects.

The analysis phase contains the study of the available behavioural simulators of architectural objects or behavioural specifications of functions, and the study of the application software or benchmarks. The decision is the selection, design and instantiation of the simulation models into a system simulation environment. The estimation and validation should focus on the functionality and cost of the architecture. Functional simulation using instruction-set simulators and memory models, and hardware and software estimations are usable tools.

4.3 Decision support methods

The presented NoC design flow requires new methods for supporting the architecture-related decisions. New platform-based design paradigms are replacing the specification and synthesis-based approaches. The selection and mapping are design activities that need a new type of design space exploration support.

Proactive decision support methods, e.g. estimations, are needed for efficient reduction of the design space. Complexity and capacity estimations are needed for sizing and scaling so that we know the quantities of the objects. Mappability estimates are needed for the selection of physical and functional objects so that we know the suitability of the objects. Performance and workload estimations are needed for allocation and validation so that we know the effectiveness of our design. Cost and quality estimation are needed for a feasibility analysis. We have to know as early as possible whether the design work is worth the effort or not.

Validation methods are needed after the decision for the analysis of the consequences. Validation methods can be more detailed because only a small number of alternatives are designed in more detail. However, the validation

methods must also be very effective. Validation must cover all the quality criteria, but if we consider the performance, for example, the quality validation procedure may consist of a network simulation, transaction-level architecture simulations and instruction-level processor simulations.

4.3.1 Complexity-based quality estimation

The selection of a new architecture concept requires analysis and estimation methods that use very abstract information about target applications or available technologies. The reason is that new architecture concepts are only designed when major changes occur in either area or when a company tries to penetrate new markets.

Paper IV presents a method for estimation of architecture quality using complexity-based metrics. The purpose of the method is to enable the comparison of the different approaches to system implementation. It uses complexity estimates of applications, possible architecture concepts, and estimation of design effort and technology development scenarios as a basis for the quality evaluation process. The idea is to encapsulate the quality measures into a single measure called figure of quality, which can be estimated from early mappings of system functionality to initial architectures.

The figure of quality, *FOQ* is obtained using following formula

$$FOQ = C_S(t) + DC_{HW}(t) + DC_{SW}(t), \quad (1)$$

where $C_S(t)$ is the structural complexity of the system. This takes the complexities of both the architecture and the applications into account and is, in fact, a measure of the implementation cost. $DC_{HW}(t)$ is the hardware design cost and $DC_{SW}(t)$ is the software design cost. All the parameters are dependent on time, t, since the technology development and the methodology development affect different cost factors differently. All the parameters also have weight values, because in a different type of system the importance of the implementation and design costs may differ.

The estimation combines the functionality of the system, e.g. mathematical equations and algorithms, and the complexities of the physical objects of the architecture. The process is the following.

1. Analyse the operational complexities, $C_F(f)$, of the application functions, $F = \{f_1, \ldots, f_N\}$, that are the baseline for the system development. Operational complexities can be analysed from pseudo-code representation and contain information on computation, communication and storage.

2. Define the set of architecture concepts, $A = \{A_1, \ldots, A_K\}$, based on the characteristics of the application area and architecture templates. Architecture A_i consists of a set of architecture objects, $A_i = \{a_1, \ldots, a_M\}$, that can be, for example, RISC core, DSP core, coprocessor, SRAM block, etc.

3. Generate system models, that is map functions to architectural objects, $M : F \rightarrow A$, so that each architecture object has a set of functions associated with it in such a way that each function is associated with one and only one architecture object. During the mapping the operational complexity is divided into the architecture object types. Computation complexity is allocated to the processing objects, storage complexity to memory-type objects and communication complexity to communication channels.

4. Estimate the implementation costs – that is, the structural complexities of the objects of the architecture – based on the operational complexity estimates,

$$\forall a_j \in A_i : C_S(a_j) = G_m\left(\sum_{f_k \in a_j} C_F(f_k)\right), \tag{2}$$

where G_m is architecture object-based estimation function. Then create the total structural complexity estimate,

$$C_S(A_i) = \sum_{a_j \in A_i} C_S(a_j). \tag{3}$$

The structural complexity measures of implementation are gates per function, gates per resource, gates per data transfer, instructions per function

and memory per function. Estimation of the implementation cost of the architecture objects based on the operational complexities of the algorithms requires either advanced synthesis or hardware estimation tools, or cumulative knowledge of the implementations in the form of IP libraries or designer experience. Some examples of possible estimation functions, *G,* are given in Paper IV.

5. Estimate the hardware design costs, $DC_{HW,}$ using the structural complexity estimates of the architecture objects, and the software design costs, $DC_{SW,}$ using the operational complexity estimates of those functions that are mapped to the programmable architecture objects. In both cases estimation functions exploit the cumulative knowledge of an organisation.

6. Analyse the technology and market development trends and create functions for weights of quality factors that describe their relative importance as a function of time based on product and market scenarios and technology improvement functions for the implementation and design costs.

7. Analyse the figures of the quality of the architecture concepts.

The method is presented in more detail in Paper IV, which also contains an experiment using an ADSL modem as a product example. The method attempts to narrow the gap between detailed technical analyses that originate from the engineering domain and abstract risk analyses that originate from the management domain. The main strengths of the proposed approach are the simplifications and possibility of adding organisation-specific cumulative knowledge into the estimation process. At the same time, they are also the main weaknesses of the approach. The models that are used are very simple and it is possible that they do not capture all the necessary details. Some of the parameters that are used, especially in the estimation functions, are difficult to define and fine tune. It may be tempting to simply experiment with them and then end up with the wrong conclusions. The proposed method, however, provides a systematic approach for analysing the expected quality of the architecture concept. By automating the analysis process and by adding more advanced models to the analyses of the technology and market changes, it could be a valuable tool for supporting strategic decisions.

4.3.2 Mappability-based quality estimation

Mappability-based quality estimation is developed for evaluating the goodness of the processor architecture and algorithm pair. The processor architecture in this context only means the execution architecture, so it excludes cache memories and memory organisations. It also only deals with single processors and parallelism is only dealt with on the data and instruction levels.

By mappability of an architecture-algorithm pair we mean the degree of matching or correlation between the resources provided by the processor architecture and the requirements described by the algorithm. The perfect match means that the architecture has exactly the right number of resources and an optimal pipeline organisation for the algorithm to be executed. Here the optimality is defined so that increasing the number of any resources would not improve the performance but only decrease their utilisation. In an ideal case the architecture would be executing all the possible computations using all the resources of the architecture at every time step.

Performance and utilisation of resources are typically used as evaluation measures of processor-algorithm pairs. The problem with both of them is that they only express part of the problem. Performance alone does not tell anything about how effectively the resources and architecture are used and how many of the resources are waiting idle. Utilisation alone does not reveal anything about the potential of an algorithm. With simple architectures it is easy to have high utilisation figures without knowing about the possibility of having much higher performance by adding resources to the architecture. Metrics such as performance/cost and performance/energy try to address this problem, but their problem is that they also measure the efficiency of the resource implementation in addition to the architecture.

The mappability estimation presented in this thesis and in Papers V and VI is targeted at supporting decisions related to the processor architecture and algorithm selections during the early phases of the architecture design. This objective prevents the use of simulation or execution as an evaluation method. The basic idea of the estimation method is very simple and presented in Figure 17. The computation of an algorithm can be presented as a control flow graph and data-dependency graphs of its nodes. The control flow graph represents the

dynamic characteristics of the algorithm and the data dependency graphs represent the data operations in the basic blocks of the algorithms. The sets of data dependency graphs, the operations and their dependencies, describe the possible execution boundaries. The execution architecture of a processor core consists of execution resources of operations, local data storage capacity – e.g. registers – and input and output capacity for operations and data. The execution resources are organised in time and space – e.g. pipeline and parallelism - which limits their usability. Their ability to execute operations sets additional constraints. The mappability of a processor and algorithm is a measure of how the algorithm characteristics and processor capability correlate or match with each other, e.g. how effectively the architecture resources can be exploited with a given computational structure.

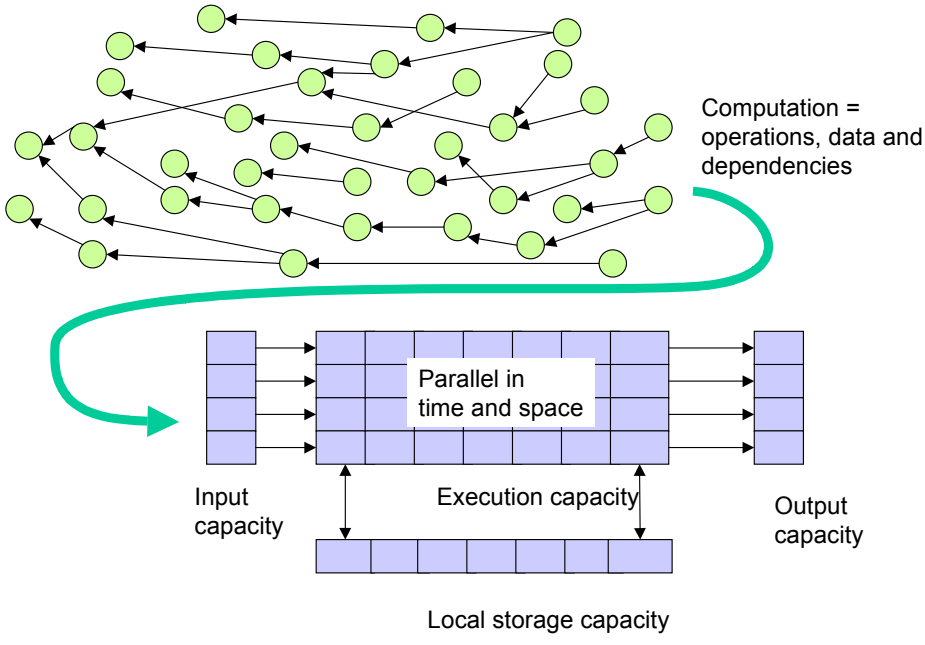


Figure 17. The basic idea in mappability estimation.

In the developed approach we consider the mappability of the processor core (*c)* and algorithm (*a)* from six different viewpoints: instruction set suitability (*iss*), external data availability (*eda*), internal data availability (*ida*), control flow continuity (*cfc*), data flow continuity (*dfc*) and execution unit availability (*eua*).

Furthermore, we characterise each basic block node of algorithm separately and combine the information using basic block execution times as weight values in averaging.

For each viewpoint, $i \in \{iss, eda, ida, cfc, dfc, eua\}$, we have a mappability estimate, $M_i(c,a)$, that is a weighted average of the node estimates, $m_{i,j}(c,a_j)$. For each node we have a node and viewpoint-specific algorithm characterisation, $e_i(a_j)$, and viewpoint-specific core characterisation, $e_i(c)$. The mappability of each viewpoint, $m_{i,j}(c,a_j)$, is analysed separately using simple ratios,

$$m_{i,j}(c,a_j) = \begin{cases} \dfrac{e_i(c)}{e_i(a_j)}, & e_i(c) \le e_i(a_j) \\ \dfrac{e_i(a_j)}{e_i(c)}, & e_i(a_j) < e_i(c) \end{cases}. \tag{4}$$

The total mappability of a viewpoint is then extended to cover the whole algorithm using the weighted average of the mappability values of the nodes,

$$M_i(c,a) = \frac{\sum_{j=1..|V|} \left(w_j \cdot m_{i,j}(c,a_j) \right)}{\sum_{j=1..|V|} w_j}, \tag{5}$$

where j is the basic block node in control flow graph V of the algorithm, and w_j is the weight – e.g. number of execution times of blocks. Total mappability is the weighted average of the viewpoints,

$$M(c,a) = \sum_i w_i \cdot M_i(c,a), \tag{6}$$

where w_i can be used for defining the relative importance of the viewpoints. The essential issue in the approach is the characterisation of the algorithm and core. The details are given in Paper V, but Table 2 summarises it.

Table 2. Characterisation of algorithm and processor core features.

	Algorithm characterisation	Processor core characterisation
Instruction set suitability	Effectiveness and exhaustiveness of instructions in a core with respect to operations in an algorithm	Cost of execution instructions in terms of time and energy
External data availability	Number of instructions to be fetched from memory and number of data operations	Bus capacity of a core
Internal data availability	Number of data to be stored in registers, e.g. number of data dependencies in schedule steps	Number and availability of registers
Control flow continuity	Number of branches	The effect and probability of branch penalty, e.g. superpipelining degree and branch prediction efficiency
Data flow continuity	Possibility to data hazards, e.g. number of data dependencies and mobility of operations in a schedule	Possibility to have data hazards in a pipeline, e.g. superpipelining degree
Execution unit availability	Possibility to execute operations in parallel, e.g. average number of operations in single scheduling steps in ASAP and ALAP schedules	Number of parallel execution paths in core and their capability to execute different types of instructions

The principles of the estimation and early results are given in Paper V, and the results of its use in processor selection are given in Paper VI. The main benefit of our estimation approach is that it is based on comparing the characteristics of the algorithms and processor cores instead of mimicking the execution. Therefore, we can use very simple models of both processors and algorithms. This makes the approach applicable to a variety of design problems, such as processor selection, instruction-set design, processor configuration, execution architecture selection and algorithm selection.

The main limitation of our approach is that the effects of memory organisation, cache and communication are not yet considered because they would require more detailed views of partitioning, compilation and application programs. That would increase the complexity of the estimation significantly. The approach is also limited to very fine-grain parallelism. The absolute performance of a

processor is not yet taken into account in our approach. The designer has to explicitly check that the processor is capable of doing the computation within the specified time frame.

4.3.3 Capacity-based quality estimation

The capacity-based quality estimation method has been developed for evaluating the mapping of the applications to the architecture objects. The method can be used either when the architectural objects are being defined or when an application is being mapped to the final architecture. The figure of quality, *FOQ*, in this context is the function of the performance and utilisation of resources. The performance relates to the execution times of the applications and the communication latencies. The utilisation of resources relates to the ability of the architecture and its objects to perform the tasks allocated to them. The exact format of *FOQ* function depends on the type of system, but it describes the hardware platforms ability to execute different type of applications that are needed in the different uses of the platform.

The basic idea in this estimation method is similar to the function/architecture codesign or mapping-based codesign presented in the POLIS and METROPOLIS approaches: we model the application and the architecture, then we map the application to the architectural units and do the performance evaluations. The main differences in the approaches are due to the chosen abstraction levels of both applications and architectures, which further affects the principles on which the performance models operate.

In our approach the objective is to study multiprocessor systems targeted at some application domain, in this case to mobile computation. Both the hardware architecture and software system may have a lot of variety in their objects. The maturity of the models is low at the time the system must be evaluated. The complexity of the whole system is such that detailed modelling-based approaches and refinement-based approaches are not feasible. Therefore, we have chosen to use abstract models and transaction-based simulations. The quality estimation is based on SystemC 2.0 simulations.

In principle, the quality estimation model (*QEM*) consists of the SystemC 2.0 simulator kernel (*SSK*), application workload models (*AWM*) and architecture capacity models (*ACM*). So, $QEM = (SSK, AWM, ACM)$.

The application workload models consist of functions that request capacity from the architectural objects: $AWM = (CR_i, AC, AM)$, where CR_i is the sets of capacity requests of types $i \in \{comp, comm, stor\}$, AC represents the control behaviour of the workload and AM is the set of application performance monitors. The three capacity request types are computation request, communication request and storage request. The capacity requests are in the form of quantitative units, which are dependent on the request type; the computation requests are modelled with the number of instructions needed by the application or the part of the application; and the communication requests are modelled with the number of bits to be transferred via the communication channel. In this approach the explicit addresses of the communication must be known. The storage requests are modelled with the number of bits to be stored in a memory. The application control, *AC,* can be used for generating hierarchical *AWM*s. The only limitation for *AC* is that it may not interact with *SSK* or *ACM*. The application performance monitors, *AM,* can be used for collecting application or function execution times from the simulation. Such analysis is especially interesting in the early stages of application development, when the performance of the architecture with a new application and existing workload needs to be validated.

The architecture capability model, *ACM*, consists of a set of resource capacity models, RCM_i and their interconnections. The resource types are the same as the request types in *AWM*. The resource capacity model further divides into a set of capacity parameters, resource control model and performance monitors, $RCM = (RCP, RCC, RPM)$.

The resource capacity parameters define the performance behaviour and the upper limits of the resources. The parameters for computation resource are instructions per cycle and total number of cycles, e.g. $RCP_{comp} = \{IPC, TC\}$. The total number of cycles depends on the length of the simulation and the resource's relative clock rate. In the case of steady state analysis it can be left un-specified. The $RCP_{comm} = BPT$, where *BPT* is bits per transaction, and

$RCP_{stor} = \{CPA_j, MS\}$, where CPA_j is latency of memory access of type *j*, and *MS* is the size of memory.

The resource control model, *RCC,* is a function that controls which capacity request is served in the case of simultaneous requests, and how the capacity is consumed. For example, the *RCC* for a bus is a bus arbiter and a timekeeper, which decides how many cycles are spent during the transaction. It is also possible to model the resource sharing as a part of the application workload model. The choice depends on the purpose of the simulation and whether, for example, the operating system is considered part of the architecture or not. In the future it may well be that *RCC* functions are integrated as part of the simulator kernel, *SSK* (Kunkel 2003).

The performance monitors, *RPM,* collect information on the behaviour and states of the architecture objects during the simulation. For the computation resources, the states are idle, waiting and running. For communication resources, the states are active and idle. For storage resources, the states are writing, reading, storing and empty. The *RPM* can also be architecture service-specific. For example, the RPM can collect statistics on the bus or memory operations of all the applications. The collected information serves as a basis for the utilisation and average performance analyses.

The approach can be used in a variety of ways, depending on the analysis objectives and the maturity of the models. The granularity in which the capacity requests are presented can vary from a single instruction to a basic block, a function or a complete application. Similarly, the resource control model can give instructions or bit transactions for applications or more coarse-grain abstract computational capacity.

One example of the use of this approach is given in Paper VII. The system to be studied was an execution platform for future Internet appliances, IHIP. The basic architecture consisted of four DSP cores, FFT and Viterbi coprocessors, and configurable on-chip embedded SRAM memory organisation. In addition, the chip design had an RISC core, DMA, AMBA bus and 16Mbit of on-chip DRAM. The MPEG-2 decoder and HiperLan/2 transceiver applications were studied because they have very different requirements for the computation platform.

The idea in the quality estimation and simulation was to compare the configurable memory and shared memory approaches, and to validate the system. Therefore, the focus in the modelling and simulation was on the configurable memory organisation. The models of the processor and processing capacity request models were very abstract. Both the WLAN transceiver and the MPEG-2 decoder are applications where most of the processing is steady state data stream processing. Therefore, we decided to ignore the application start-up and closing effects.

The results and the details of the experiment are given in Paper VII. It was demonstrated that capacity-based quality estimation could provide information for the system designer that is useful and extremely difficult to get otherwise. The functional simulation model of the IHIP architecture using instruction-set simulators, VHDL simulation models, Denali memory models, and SystemC 2.0 models was also constructed. Although the IHIP simulation model was based on reusing models designed earlier, the performance simulation model was much easier to build. It also turned out that we were not able to run the same case examples in the functional simulation due to software maturity problems (WLAN) and economic constraints (MPEG-2).

The experiment in Paper VII only covers part of the possibilities presented in this thesis. The important issues for future research are how to validate the application workload models and how to partition the system to the application model and the resource model. The workload model is an abstraction of real application code, existing code or planned code. The correctness, accuracy and detail of the application characterisation are the major challenges. One possible approach is to integrate the ideas of the mappability-based quality estimation approach into the workload modelling.

The platforms seem to be expanding in the platform-based design. When the products mature, more and more features are transferred from the application side to the platform side. Initially, the platforms were processor-based computer architectures. Nowadays, they contain the operating systems and middleware functions. Recently, the application domain-specific dedicated functions have been integrated into the platform side. The question is how could we try to follow this trend in the quality estimation as well. One possible approach could be to build interfaces at a high level of granularity, which could lead to an application-platform service interface instead of the application function-resource capacity interface presented in this thesis.

5. Introduction to papers

Paper I (A Network-on-Chip Architecture and Design Methodology, ISVLSI 2002) describes the principles of the NoC architecture and design methodology. My main contribution to this paper is Chapter 4, about design methodology, which is based on my earlier work (Soininen 1997, Soininen et al. 1998b, Soininen 1999). The chapter presents the basic ideas about dividing the design into backbone, platform and system layers, and outlines the requirements for new tools and design methods. I have also contributed to the development of the NoC architecture and, especially, to its region concept. My contribution to considering the NoC resources as embedded systems that are distributed into the silicon surface was also significant.

Paper II (Extending platform-based design to Network-on-Chip systems, VLSI Design 2003) presents the Backbone-Platform-System design methodology in more detail. It introduces the design plane concepts and integrates different design methods and activities into the methodology. It also considers the decision-support methods and their role in the design methodology. The methodology presented in Paper II is the basis for Chapter 4.1 in this thesis. It is also the target design framework for the decision-support methods presented in Chapter 4.3. Paper II was also the basis for Chapter 2 in Jantsch and Tenhunen (2003). I am the first author of the paper and it is mostly based on the my research.

Paper III (Minimisation of functionality and implementation of embedded low-power WWW-server, Electronic Letters 2002) presents an example of an IP block. This short paper focuses on presenting the innovation and benefits of putting WWW-servers' functionality in hardware, and ignores the fact that designing an IP block is two to three times more demanding than designing a hardware block. The designed IP block served as a case example in a research project that focused on IP-based SoC design. The reusable IP blocks are the cornerstones of the architecture design methodology presented in this thesis, which is the reason for including this paper here. My contribution to this paper was inventing the idea of a hardware WWW-server and the initial design of its operation principles and architecture.

Paper IV (Application of decision-making method for the architecture selection of an ADSL modem, DSD2001) presents a complexity-based architecture quality estimation method. The method is applied to a technology and architecture concept selection for an ADSL modem. Modem architectures based on dedicated hardware, simple RISC controllers, DSP cores and general-purpose processors were compared. The comparison was based on a complexity estimate of ADSL functionality, initial mappings of functions to architectures, and technology development and design productivity trends within next ten years. The best comparable architectures found were similar to those found in commercial modems. However, the accuracy of the long-term development remains to be seen. The quality estimation method was developed by me.

Paper V (Mappability Estimation Approach for Processor Architecture Evaluation, NorChip 2002) presents the main concepts and approach for the mappability-based quality estimation for an architecture-algorithm pair. The algorithm-architecture correlation functions are presented and the approach is used in simple test cases. The results were validated using instruction-set simulations and measurements with real processors. I was the first author in this paper. The main contribution is in developing the principles of correlation and correlation functions.

Paper VI (Fast Processor Core Selection for a WLAN Modem using Mappability Estimation, CODES 2002) presents how mappability estimation can be used for defining most suitable processing architectures for application functions. The case example was a HiperLan/2 transceiver, which is very similar to IEEE802.11a. The transceiver was decomposed into functions, of which 13 were studied. Over ten thousand processor architecture models were generated using different combinations of the main architecture parameters. Finally, mappability estimates were calculated for all possible mappings and the best architectures for the functions were selected. The conclusion drawn in the paper was that mappability estimation could be used for the identification of the required architecture characteristics for a set of algorithms, or for the selection of processor cores in the design of a multiprocessor system-on-chip. The method was found suitable for fast reduction of the design space because only simple algorithm and architecture models were needed. I was the first author of this paper. The main contribution is in developing the mappability estimation method, and using it for architecture definition.

Paper VII (Configurable Memory Organisation for Communication Applications, DSD2002) presents how the workload modelling and architecture-level simulations are used in the validation of the configurable memory organisation. The architecture simulation ideas are based on my earlier work with functional modelling and simulation (Kauppi et al. 1992), cosimulation (Soininen et al. 1998a), and system-level simulation (Soininen et al. 1995). The configurable memory organisation is an example of an architectural solution that increases the reusability and flexibility of the system hardware and makes it a platform for a variety of products. I was the first author of the paper. The main contribution is in defining the performance simulation methodology and the configurable memory organisation.

6. Conclusions

The development of integrated circuit technology will change the architectures of computing and processing systems. Already today, even the hand-held products have execution engines that consist of several processors. In the near future, the number of processors and processing units will increase to tens and even to hundreds if the energy consumption is not a severe product constraint. Such a computational capacity enables products that are more intelligent in terms of user interaction, information content and resource consumption. The same technology development will be used for making the individual processors more efficient, more flexible and more adaptive, which will enable us to develop more usable and useful products when the minimisation of physical dimensions is the target.

The cost of increasing complexity is the required design effort. There are clear indications that the design complexity of an ASIC that uses the total technology capacity is reaching the limits of design capacity. The EDA industry, the main system developers and the academic research community have tried to respond to the design productivity challenge by trying to develop more efficient design tools and by trying to promote the reuse of design objects. The most recent outcome of these efforts is the platform-based design paradigm, where the idea is that the design process consists of a sequence of platforms; a new platform is built on top of an earlier platform by mapping or integrating new functionality onto it. The main benefit is that the platform can serve as a reusable design object that can encapsulate any type of design data.

The architecture design methods and quality estimation approaches presented in this thesis aim at solving the complexity-related problems in the development of application-specific computer architectures. The platform-based design consists of two phases: platform definition and application mapping. In the case of future multiprocessor and NoC architectures, both phases are extremely complex and need advanced methods that are based on abstract models and uncertain information.

The Backbone-Platform-System methodology divides the system development into three phases. The backbone layer focuses on the physical design problems. The separation of the infrastructure from the application computation, which

also means the separation of the communication from the computation already at the physical level, helps in isolating the physical design problems from the computation architecture and application development problems. The platform layer focuses on the computation architecture problems. The design of application-specific computers requires that we know how to analyse the application domain, how to select the best technologies and units for our architecture, and how to validate our decisions. The system layer focuses on the application development, especially for application mapping problems.

This thesis presents three quality estimation approaches for helping platform design and application mapping decisions. The complexity-based quality estimation can be used for defining architecture concepts and technologies. The developed figure of quality metric enables the comparison of different types of alternatives at very early stages of the product development process. The mappability-based quality estimation can be used for evaluating the goodness of processor core-algorithm pairs. The developed mappability metric combines the execution time and resource utilisation views of the algorithm execution in a core. The method uses abstract parameter-based models of architecture and a graph-based model of algorithm, and enables fast reduction of design space. The approach can also be used when modelling the application workload for performance simulations. The capacity-based quality estimation can be used for the evaluation of architecture's performance with a specified workload. The performance in this context is either the performance experienced by the application or the performance of the architecture. The estimation is based on a performance simulation using abstract workload models and architecture capability models.

The result of this research is that the proposed quality estimation methods can be used as part of a decision-support toolkit when developing application-specific computer architectures. It is clear though that the current versions of the methods are not complete and further work is required. It must be also stated that the validation of the system-level estimation method is expensive and time consuming. Therefore, the validation in this thesis is partly based on comparisons with existing and comparable designs rather than accurate measurements. The estimation approaches also have limitations, which affect the results, but, on the other hand, estimations always have some risk of error and when we make estimations at early stages of design the risks are higher.

Estimator is a calculator, and the responsibility for avoiding the garbage-in, garbage-out syndrome belongs to the user.

Research topics for the future

Quality-driven development processes will be needed. The essential issue in all product development projects is the user's experience of the product. Therefore, the quality attributes must be migrated into the product development so that the design objectives will serve the total quality of the product. In the case of application-specific computer architectures, the quality comes from the mappability of the architecture and algorithms.

Estimation accuracy is the critical measure of the usability of an estimator. In the case of the estimators proposed in this thesis, the main factor is the quality of the input data. The essential issues to be studied are the analysis of the algorithm characteristics and the modelling of the architecture capabilities. The understanding of the operation of a software system and the mapping process will become more important with systems that are more complex.

Decision-support methods and tools will play an essential role when solving the design complexity problems. The product development is a sequence of decisions. The right decisions lead to a product and the wrong decisions lead to iterations. The increasing number of alternatives forces us to make bigger decisions earlier, and this calls for advanced support. Modelling, estimations, analyses and simulations will be needed.

References

Aho, A., Hopcroft, J., Ullman, J. 1983. Data Structures and Algorithms. Reading, MA, USA: Addison–Wesley Publishing Company. 427 p. ISBN 0-201-00023-7

Aho, A., Sethi, R., Ullman, J. 1986. Compilers: Principles, Techniques and Tools. Reading, MA, USA: Addison–Wesley Publishing Company. 796 p. ISBN 0-201-10194-7

Allara, A., Brandolese, C., Fornaciari, W., Salice, F., Sciuto, D. 1998. System-level performance estimation strategy for sw and hw. Proceedings of International Conference on Computer Design: VLSI in Computers and Processors, ICCD '98. Austin, TX, USA, October 5–7, 1998. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 48–53.

Anceau, F. 1986. The architecture of microprocessors. Wokingham, UK: Addison-Wesley Publishing Company. 252 p. ISBN 0-201-14401-8

ARM. 1999. AMBA Specification (Rev 2.0). ARM Limited. Available at: http://www.arm.com

August, D. I., Keutzer, K., Malik, S., Newton A. R. 2002. A disciplined approach to the development of platform architectures. Microelectronics Journal 33, pp. 881–890.

Austin, T., Larson E., Ernst, D. 2002. SimpleScalar: An Infrastructure for Computer System Modeling. Computer, Vol. 35, No. 2, pp. 59–67.

Bahill, A. T., Alford, M., Bharathan, K., Clymer, J. R., Dean, D. L., Duke, J., Hill, G., LaBudde, E. V., Taipale, E. J., Wymore, A. W. 1998. The Design-Methods Comparison Project. IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, Vol. 28, No.1, pp. 80–103.

Balarin, F., Chiodo, M., Giusto, P., Hsieh, H., Jurecska, A., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A., Sentovich, E., Suzuki, K., Tabbara, B. 1997. Hardware-Software Co-Design of Embedded Systems – The POLIS Approach. Boston, MA, USA: Kluwer Academic Publishers. 297 p. ISBN 0-7923-9936-6

Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A. 2003. Metropolis: An Integrated Electronic System Design Environment. Computer, Vol. 36, No. 4, pp. 45–52.

Banakar, R., Steinke, S., Lee, B-S., Balakrishnan, M., Marwedel, P. 2002. Scratchpad Memory: A Design Alternative for Cache On-Chip Memory in Embedded Systems. Proceedings of 10th International Symposium on Hardware/Software Codesign, CODES 2002. Estes Park, Colorado, USA, 6–8 May 2002. New York, NY, USA: ACM Press. Pp. 73–78.

Barros, E., Rosenstiel, W. 1992. A method for hardware software partitioning. Proceedigns of Computer Systems and Software Engineering. Hague, Netherlands, 4–8 May 1992. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 580–585.

Bashford, S., Bieker, U., Harking, B., Leupers, R., Marwedel, P., Neumann, A., Voggenauer, D. 1994. The MIMOLA Language – Version 4.1. Technical Report, Computer Science Department. University of Dortmund. 135 p.

Bass, L., Clements, P., Kazman, R. 1998. Software Architecture in Practice. Reading, MA, USA: Addison–Wesley, Inc. 452 p. ISBN 0-201-19930-0

Benini, L., Bertozzi, D., Bruni, D., Drago, N., Fummi, F., Poncino, M. 2003. SystemC Cosimulation and Emulation of Multiprocessor SoC Designs. Computer, Vol. 36, No. 4, pp. 53–59.

Benini, L., De Micheli, G. 2002. Networks on chips: a new SoC paradigm. Computer, Vol. 35, No. 1, pp. 70–78.

Bentley, B. 2001. Validating the Intel Pentium 4 Microprocessor. Proceedings of 38th Design Automation Conference. Las Vegas, NV, USA, 18–22 June 2001. New York, NY, USA: ACM Press. Pp. 244–248.

Berry, G., Gonthier, G. 1992. The Synchronous Programming Language ESTEREL: Design, Semantics, Implementation. Science of Computer Programming, Vol. 19, No. 2, pp. 83–152.

Bez, R., Camerlenghi, E., Modelli, A., Visconti, A. 2003. Introduction to flash memory. Proceedings of the IEEE, Vol. 91, No. 4, pp. 489–502.

Bhartacharyya, S., Leupers, R., Marwedel, P. 2000. Software synthesis and code generation for signal processing systems. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 47, Issue 9, pp. 849–875.

Boehm, B. 1988. A Spiral Model of Software Development and Enhancement. Computer, Vol. 21, No. 5, pp. 61–72.

Bondalapati, K., Prasanna, V. 2002. Reconfigurable Computing Systems. Proceedings of the IEEE, Vol. 90, No. 7, pp. 1201–1217.

Bose, P., Conte, T. 1998. Performance Analysis and Its Impact on Design. Computer, Vol. 31, No. 5, pp. 41–49.

Brandolese, C., Fornaciari, W., Salice, F., Sciuto, D. 2001. Source-Level Execution Time Estimation of C Programs. Proceedings of 9th International Symposium on Hardware/Software Codesign, CODES 2001. Copenhagen, Denmark, 25–27 April 2001: New York, NY, USA: ACM Press. Pp. 98–103.

Buck, J., Ha, S., Lee, E., Messerschmitt, D. 1994. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. International Journal of Computer Simulation, Vol. 4, April 1994, pp. 152–182.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. 1996. A System of Patterns – Pattern-oriented software architecture. Chichester, England: John Wiley & Sons. 457 p. ISBN 0-471-95869-7

Campi, F., Canegallo, R., Guerrieri, R. 2001. IP-Reusable 32-Bit VLIW Risc Core. Proceedings of 27th European Solid-State Circuits Conference. Villach, Austria, 18–20 September 2001. Paris, France: Frontier Group. Pp. 456–459.

Carrig, K. 2000. Chip Clocking Effect on Performance for IBM's SA-27E ASIC Technology. MicroNews, Vol. 6, No. 3, pp. 12–16.

Carro, L., Kreutz, M., Wagner, F., Oyamada, M. 2000. System Synthesis for Multiprocessor Embedded Applications. Proceedings of Design Automation and Test in Europe. Paris, France, 27–30 March 2000. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 697–702.

Caspi, E., DeHon, A., Wawrzynek, J. 2001. A Streaming Multi-Threaded Model. Presented at Third Workshop on Media and Stream Processors. Austin, TX, USA, 2 December 2001. 8 p.

Cesário, W., Lyonnard, D., Nicolescu, G., Paviot, Y., Yoo, S., Jerraya, A., Gauthier, L., Diaz-Nava, M. 2002. Multiprocessor SoC platforms: a component-based design approach. IEEE Design & Test of Computers, Vol. 19, No. 6, pp. 52–63.

Chan, Y-H., Kudva, P., Lacey, L., Northrop, G., Rosser, T. 2003. Physical synthesis methodology for high performance microprocessors. Proceedings of 40th Design Automation Conference. Anaheim, CA, USA, 2–6 June 2003. New York, NY, USA: ACM Press. Pp. 696–701.

Chandra, S., Moona, R. 2000. Retargetable Functional Simulator Using High Level Processor Models. Proceedings of 13th International Conference on VLSI Design, 2000. Calcutta, India, 3–7 January 2000. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 424–429.

Chang H., Cooke, L., Hunt, M., McNelly, A., Martin, G., Todd, L. 1999 Surviving the SOC revolution: a guide to platform-based design. Boston, MA, USA: Kluwer Academic Publishers. 235 p. ISBN 0-7923-8679-5

Chou, P., Ortega, R., Borriello, G. 1995. The Chinook hardware/software co-synthesis system. Proceedings of the 8th International Symposium on System Synthesis. Cannes, France, 1–5 September 1995. New York, NY, USA: ACM Press. Pp. 22–27.

Compton, K., Hauck, S. 2002. Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys, Vol. 32, No. 2, pp. 171–210.

Corporaal, H. 1998. Microprocessor architectures from VLIW to TTA. Chichester, UK: John Wiley & Sons. 407 p. ISBN 0-471-97157-X

Culler, D., Pal Singh, J. 1999. Parallel Computer Architecture – A Hardware Software Approach. San Francisco, CA, USA: Morgan Kaufmann Publishers. 1025 p. ISBN 1-55860-343-3

Cuppu, V., Jacob, B., Davis, B., Mudge, T. 1999. A performance comparison of contemporary DRAM architectures. Proceedings of the 26th International Symposium on Computer Architecture. Atlanta, Georgia, USA, 2–4 May 1999. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 222–233

Dally, W., Towles, B. 2001. Route Packets, Not Wires: On-Chip Interconnection Networks. Proceedings of 38th Design Automation Conference. Las Vegas, NV, USA, 18–22 June 2001. New York, NY, USA: ACM Press. Pp. 684–689.

D'Ambrosio, J., Hu, X. 1994. Configuration-level hardware/software partitioning of real-time embedded systems. Proceedings of 3rd International Conference on Hardware/Software Codesign. Grenoble, France, 22–24 September 1994. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 34–41.

Daveau, J.-M., Marchioro, G., Ismail, T., Jerraya, A. 1997. Protocol selection and interface generation for HW-SW codesign. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 5, No. 1, pp. 136–144.

Day, R. 1993. Quality Function Deployment – linking a company with its customers. Milwaukee, Wisconsin, USA: ASQC Quality Press. 245 p. ISBN 0-87389-202-X

De Man, H., Catthoor, F., Goossens, G., Vanhoof, J., Van Meerbergen, J., Note, S., Huisken, J. 1990. Architecture-driven Synthesis Techniques for VLSI Implementation of DSP Algorithms. Proceedings of the IEEE, Vol. 78, No. 2, pp. 319–334.

De Micheli, G. 1994. Computer-aided hardware-software codesign. IEEE Micro, Vol. 14, No. 4, pp. 10–16.

De Micheli, G., Gupta, R.K. 1997. Hardware/software co-design. Proceedings of the IEEE, Vol. 85, Issue 3, pp. 349–365.

Dobrica, L., Niemelä, E. 2002. A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering, Vol. 28, No. 7, pp. 638–653.

Duato, J., Yalamanchili, S., Ni, L. 1997. Interconnection Networks: An Engineering Approach. Los Alamitos, CA, USA: IEEE Computer Society Press. 515 p. ISBN 0-8186-7800-3

Ecker, W. 1995. Classification of Design Steps and Their Verification. Proceedings of European Design Automation Conference with EuroVHDL. Brighton, UK, 18–22 September 1995. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 536–541.

Edwards, S., Lavagno, L., Lee, E., Sangiovanni-Vincentelli, A. 1997. Design of Embedded Systems: Formal Models, Validation and Synthesis. Proceedings of IEEE, Vol. 85, No. 3, pp. 366–390.

Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y. 2003. Taming heterogeneity – the Ptolemy approach. Proceedings of the IEEE, Vol. 91, No. 1, pp. 127–144.

Eles, P., Kuchcinski, K., Peng, P. 1998. System Synthesis with VHDL. Dortrecht, Netherlands: Kluwer Academic Publishers. 370 p. ISBN 0-7923-8082-7

Engblom, J., Ermedahl, A., Altenbernd, P. 1998. Facilitating Worst-Case Execution Times Analysis for Optimized Code. Proceedings of 10th Euromicro Workshop on Real-Time Systems. Berlin, Germany, 17–19 June 1998. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 146–153.

Ernst, R. 1998. Codesign of embedded systems: status and trends. IEEE Design & Test of Computers, Vol. 15, No. 2, pp. 45–54.

Ernst, R., Henkel, J., Benner, Th. 1993. Hardware-Software cosynthesis for microcontrollers. IEEE Design and Test of Computers, Vol. 10, No. 4, pp. 64–75.

Fauth, A., Praet, J., Freericks, M. 1995. Describing instruction set processors using nML. Proceedings of The European Design and Test Conference. Paris,

France, 6–9 March 1995. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 503–507.

Ferrari, A., Sangiovanni-Vincentelli, A. 1999. System Design: Traditional Concepts and New Paradigms. Proceedings of International Conference on Computer Design. Austin, TX, USA, 10–13 October 1999. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 2–12.

Flynn, M. 1966. Very High-Speed Computing Systems. Proceedings of the IEEE, Vol. 54, No. 12, pp. 1901–1909.

Flynn, M., Hung, P., Rudd, K. 1999. Deep-Submicron Microprocessor Design Issues. IEEE Micro, Vol. 19, No. 4, pp. 11–22.

Fornaciari, W., Gubian, P., Sciuto, D., Silvano, C. 1998. Power estimation of embedded systems: a hardware/software codesign approach. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 6, No. 2, pp. 266–275.

Forsell, M. 2002a. Architectural differences of efficient sequential and parallel computers. Journal of Systems Architecture, Vol. 47, No. 13, pp. 1017–1041.

Forsell, M. 2002b. A scalable high-performance computing solution for networks on chips. IEEE Micro, Vol. 22, No. 5, pp. 46–55.

Fowler, M., Scott, K. 1997. UML distilled: applying the standard object modeling language. Reading, MA, USA: Addison–Wesley Publishing Company. 179 p. ISBN 0-201-32563-3

Gajski, D. 1988. Silicon Compilation. Reading, MA, USA: Addison–Wesley Publishing Company. 450 p. ISBN 0-201-09915-2

Gajski, D., Dömer, R., Zhu, J. 1999. IP-Centric Methodology and Design with the SpecC Language. In: Jerraya, A., Mermet, J. (ed.). System Level Synthesis. Proceedings of the NATO Advanced Study Institute on System Level Synthesis for Electronic Design, Il Ciocco, Lucca, Italy, August 1998. Dortrecht, Netherlands: Kluwer Academic Publishers. Chapter 10. Nato Science Series, Vol. 357. ISBN 0-7923-5749-3

Gajski, D., Narayan, S., Ramachandran, L., Vahid, F., Fung, P. 1996. Aiming at 100h Design Cycle. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 4, No. 1, pp. 70–82.

Gajski, D., Vahid, F., Narayan, S., Gong, J. 1994. Specification and Design of Embedded Systems. Englewood Cliffs, NJ, USA: Prentice Hall. 450 p. ISBN 0-13-150731-1

Gajski, D., Vahid, F., Narayan, S., Gong, J. 1998. SpecSyn: an environment supporting the specify-explore-refine paradigm for hardware/software system design. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 6, No. 1, pp. 84–100.

Gajski, D., Zhu, J., Dömer, R. Gerstlauer, A., Zhao, S. 2000. SpecC: Specification Language and Methodology. Boston, MA, USA: Kluwer Academic Publishers. 313 p. ISBN 0-7923-7822-9

Gamma, E., Helm, R., Johnson, R., Vlissides, J. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA, USA: Addison–Wesley Publishing Company. 395 p. ISBN 0-201-63361-2

Ghazal, N., Newton, R., Rabaye, J. 2000. Retargetable Estimation Scheme for DSP Architectures. Proceedings of the Asia and South Pacific Design Automation Conference. Yokohama, Japan, 25–28 January 2000. Piscataway, NJ, USA: IEEE. Pp. 485–489.

Giusto, P., Martin, G., Harcourt, E. 2001. Reliable Estimation of Execution Time of Embedded Software. Proceedings of Design, Automation and Test in Europe. Munich, Germany, 13–16 March 2001. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 580–588.

Gong, J., Gajski, D., Narayan, S. 1994. Software Estimation From Executable Specifications. Journal of Computer and Software Engineering, 2(3), pp. 239–258.

Gong, J., Gajski, D., Nicolau, A. 1995. Performance evaluation for application-specific architectures. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 3, No. 4, pp. 483–490.

Grötker, T., Liao, S., Martin, G., Swan, S. 2002. System Design with SystemC. Boston, MA, USA: Kluwer Academic Publishers. 217 p. ISBN 1-4020-7072-1

Guerrier, P., Greiner, A. 2000. A Generic Architecture for On-Chip Packet-Switched Interconnections. Proceedings of Design Automation and Test in Europe. Paris, France, 27–30 March, 2000. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 250–256.

Gupta, R.K., De Micheli, G. 1993. Hardware-software cosynthesis for digital systems. Design & Test of Computers, IEEE, Vol. 10, Issue 3, pp. 29–41.

Gupta, T., Sharma, P., Balakrishnan, M., Malik, S. 2000. Processor Evaluation in an Embedded Systems Design Environment. Proceedings of 13th International Conference on VLSI Design. Calcutta, India, 3–7 January 2000. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 98–103.

Guthaus, M., Ringenberg, J., Ernst, D., Austin, T., Mudge, T., Brown, R. 2001. MiBench: A free, commercially representative embedded benchmark suite, Proceedings of IEEE 4th Annual Workshop on Workload Characterization. Austin, TX, USA, 2 December 2001. Piscataway, NJ, USA: IEEE. Pp. 3–14.

Harel, D. 1987. Statecharts: a visual formalism for complex systems. Science of Computer Programming, Vol. 8, No. 3, pp. 231–274.

Hartenstein, R., Becker, J., Hertz, M., Nageldinger, U. 1996. A General Approach in System Design Integrating Reconfigurable Accelerators. Proceedings of Innovative Systems in Silicon Conference. Austin, TX, USA, 9–11 October, 1996. Piscataway, NJ, USA: IEEE. Pp. 16–25.

Heidelberger, P., Lavenberg, S. 1984. Computer Performance Evaluation Methodology. IEEE Transaction on Computers, Vol. C-33, No. 12, pp. 1195–1220.

Hein, C., Gadient, A., Kalutkiewicz, P., Carpenter, T., Harr, R., Madisetti, V. 1997. RASSP VHDL Modelling Terminology and Taxonomy, Revision 2.3, June 23, 1997. RASSP Taxonomy Working Group.

Heinrich, M., Ofelt, D., Horowitz, M., Hennessy, J. 1997. Hardware/software co-design of the Stanford FLASH multiprocessor. Proceedings of the IEEE, Vol. 85, No.3, pp. 455–466.

Henkel, J., Ernst, R. 1998. High-Level Estimation Techniques for Usage in Hardware/Software Co-Design. Proceeding of Asia and South Pacific Design Automation Conference. Yokohama, Japan, 10–13 February 1998. Piscataway, NJ, USA: IEEE. Pp. 353–360.

Hennessy, J., Patterson, D. 2003. Computer Architecture – A Quantitative Approach. 3. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers. 883 p. ISBN 1-55860-596-7

Ho, R., Mai, K. W., Horowitz, M. 2001. The Future of Wires. Proceedings of the IEEE, Vol. 89, No. 4, pp. 490–504.

Hoare, C. A. R. 1985. Communicating Sequential Processes. Englewood Cliffs, NJ, USA: Prentice Hall. 256 p. ISBN 0-13-153271

Horowitz, B. Liebman, J. Ma, C. Koo, T., Sangiovanni-Vincentelli, A., Sastry, S. 2003. Platform-Based Embedded Software Design and System Integration for Autonomous Vehicles. Proceedings of the IEEE, Vol. 91, No. 1, pp. 198–211.

Hughes, C., Pai, V., Ranganathan, P., Adve, S. 2002. Rsim: simulating shared-memory multiprocessors with ILP processors. Computer, Vol. 35, No. 2, pp. 40–49.

IC Insights, Inc. 2001. The McClean Report – 2001 Edition. 13901 North 73rd Street, Suite 205, Scottsdale, Arizona, USA: IC Insights, Inc. (CD-ROM)

Ismail, T., Abid, M., Jerraya, A. 1994. COSMOS: a codesign approach for communicating systems. Proceedings of the 3rd International Workshop on Hardware/Software Codesign. Grenoble, France, 22–24 September 1994. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 17–24.

Itoh, M., Higaki, S., Sato, J., Shiomi, A., Takeuchi, Y., Kitajima, A., Imai, M. 2000. PEAS-III: An ASIP Design Environment. Proceeding of International

Conference on Computer Design. Austin, TX, USA, 17–20 September 2000. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 430–436.

ITRS. 2001. International Technology Roadmap for Semiconductors – Executive Summary. 2001 Edition. Semiconductor Industry Association (SIA), the European Electronic Component Association (EECA), the Japan Electronics & Information Technology Industries Association (JEITA), the Korean Semiconductor Industry Association (KSIA), and Taiwan Semiconductor Industry Association (TSIA). 57 p. Available at http://public.itrs.net.

Jacob, B., Mudge, T. 1998. Virtual Memory: Issues of Implementation. Computer, Vol. 31, No. 6, pp. 33–43.

Jain, M. K., Balkrishnan, M., Kumar, A. 2001. ASIP Design Methodologies: Survey and Issues. Proceedings of 14th International Conference on VLSI Design. Bangalore, India, 3–7 January 2001. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 76–81.

Jain, R. 1991. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling. New York, NY, USA: John Wiley & Sons, Inc. 685 p. ISBN 0-471-50336-3

Jain, R., Parker, A., Park, N. 1992. Predicting System-Level Area and Delay for Pipelined and Nonpipelined Designs. IEEE Transactions on Computer-Aided Design, Vol. 11, No. 8, pp. 955–965.

Jantsch, A., Tenhunen, H. (ed.) 2003. Networks on Chip. Boston, MA, USA: Kluwer Academic Publishers. 300 p. ISBN 1-4020-7392-5

John, L., Vasudevan, P., Sabarinathan, J. 1998. Workload characterization: motivation, goals and methodology. Workload Characterization: Methodology and Case Studies – based on the First Workshop on Workload Characterization. Dallas, TX, USA, 29 November, 1998. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 3–14.

Jouppi, N., Wall, D. 1989. Available instruction-level parallelism for superscalar and superpipelined machines. Proceedings of the 3rd international conference on

Architectural support for programming languages and operating systems. Boston, MA, USA, April 3–6, 1989. New York, NY, USA: ACM. Pp. 272–282.

Jóźwiak, L. 2001. Quality-driven design in the system-on-a-chip era: Why and how? Journal of Systems Architecture, Vol. 47, No. 3–4, pp. 201–224.

Juntunen, T., Kivelä, J., Reinikka, A., Sipola, M., Soininen, J.-P., Tiensyrjä, K., Tikkanen, T. 1988 Real-time structured analysis in system level design of embedded ASICs. Microprocessing & Microprogramming, Euromicro Journal, 24, pp. 449–454.

Kale, L. V. 1998. Programming Languages for CSE: The State of the Art. IEEE Computational Science & Engineering, Vol. 5, No. 2, pp. 18–26.

Karim, F., Nguyen, A., Dey, S. 2002. An Interconnect Architecture for Networking Systems on Chips. IEEE Micro, Vol. 22, No. 5, pp. 36–45.

Kauppi, M., Sarkkinen, T., Soininen, J.-P., Tiensyrjä, K. 1992. VELVET – a tool for interactive SA/VHDL design and verification of digital systems. 10th NORCHIP '92 Seminar. Helsinki, Finland, 3–4 November 1992. Helsinki, Finland: Nordisk Industriefond. Pp. 30–35.

Kauppi, M., Soininen, J.-P. 1991. Functional specification and verification of digital systems by using VHDL combined with graphical structured analysis (SA). Proceedings of the 2nd European Conference on VHDL Methods, Stockholm, Sweden, 8–11 September 1991. Stockholm, Sweden: Swedish Institute of Microelectronics. Pp. 204–211.

Keating, M., Bricaud, P. 1999. Reuse Methodology Manual for System-On-A-Chip Designs. 2nd edition. Boston, MA, USA: Kluwer Academic Publishers. 286 p. ISBN 0-7923-8558-6

Keutzer, K., Malik, S., Newton, A., Rabaye, J., Sangiovanni-Vincentelli, A. 2000. System Level Design: Orthogonolization of Concerns and Platform-Based Design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 19, No. 12, pp. 1523–1543.

Killburn, T., Edwards, D., Lanigan, M., Sumner, F. 1962. One-Level Storage System. IRE Transactions, EC-11(2), pp. 223–235.

Koehl, J., Lackey, D., Doerre, G. 2003. IBM's 50 million gate ASICs. Proceedings of Asia and South Pacific Design Automation Conference. Kitakyushu, Japan, 21–24 January 2003. Piscataway, NJ, USA: IEEE. Pp. 628–634.

Koufaty, D., Marr, D. 2003. Hyperthreading technology in the netburst microarchitecture. IEEE Micro, Vol. 23, No. 2, pp. 56–65.

Krishna, C. (ed.). 1996. Performance Modeling for Computer Architects. Los Alamitos, CA, USA: IEEE Computer Society Press. 391 p. ISBN 0-8186-7094-0

Krishnaswamy, U., Scherson, I. 2000. A Framework for Computer Performance Evaluation Using Benchmark Sets. IEEE Transactions on Computers, Vol. 49, No. 2, pp. 1325–1338.

Kruchten, P. 1995. The 4+1 View Model of Architecture. IEEE Software, Vol. 12, No. 6, pp. 42–50.

Kumar, S., Aylor, J., Johnson, B., Wulf, W. 1996. The codesign of embedded systems: a unified hardware/software representation. Boston, MA, USA: Kluwer Academic Publishers. 274 p. ISBN 0-7923-9636-7

Kunkel, J. 2003. Embedded computing – Toward IP-based system-level soc design. Computer, Vol. 36, No. 5, pp. 88–89.

Kuroda, I., Nishitani, T. 1998. Multimedia Processors. Proceedings of the IEEE, Vol. 86, No. 6, pp. 1203–1221.

Kurup, P., Abbasi, T. 1997. Logic Synthesis using Synopsys. 2. ed. Boston, MA, USA: Kluwer Academic Publishers. 322 p. ISBN 0-7923-9786-X

Lahiri, K., Raghunathan, A., Dey, S. 2000. Performance Analysis of Systems with Multi-Channel Communication Architectures. Proceedings of 13th International Conference on VLSI Design. Calcutta, India, 3–7 January 2000. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 530–537.

Lahiri, K., Raghunathan, A., Lakshminarayana, G. 2001. LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs. Proceedings of 38th Design Automation Conference. Las Vegas, NV, USA, June 18–22, 2001. New York, NY, USA: ACM Press. Pp. 15–20.

Lahtinen, V., Kuusilinna, K., Kangas, T., Hämäläinen, T. 2002. Interconnection scheme for continuous-media system-on-a-chip. Microprocessors and Microsystems. Vol. 26, No. 3, pp. 126–138.

Lăzărescu, M., Bammi, J., Harcourt, E., Lavagno, L., Lajolo, M. 2000. Compilation-based Software Performance Estimation for System Level Design. Proceedings of High Level Design Validation and Test Workshop. Berkeley, CA USA, 8–10 November 2000. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 167–172.

Lee, C. Potkonjak, M., Mangione-Smith, W. 1997. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. Proceedings of 13th Annual IEEE/ACM International Symposium on Microarchitecture. Research Triangle Park, NC, USA, 1–3 December 1997. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 330–335.

Lee, E., Sangiovanni-Vincentelli, A. 1998. A Framework for Comparing Models of Computation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 12, pp. 1217–1229.

Lee, E., Parks, T. 1995. Dataflow Process Networks. Proceedings of the IEEE, Vol. 83, No. 5, pp. 773–799.

Leijten, J., van Meerbergen, J., Timmer, A., Jess, J. 1998. Stream communication between real-time tasks in a high-performance multiprocessor. Proceedings of Design, Automation and Test in Europe. Paris, France 23–26 February 1998. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 125–131.

Lennard, C., Schaumont, P., de Jong, G., Haverinen, A., Hardee, P. 2000. Standards for system-level design: practical reality or solution in search of a question? Proceedings of Design Automation and Test in Europe. Paris, France,

27–30 March, 2000. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 576–583.

Li, Y., Wolf, W. 1999. Hardware/software co-synthesis with memory hierarchies. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 10, pp. 1405–1417.

Linton, L., Hall, D., Hutchinson, K., Hoffman, D., Evanczuk, S., Sullivan, P. 1992. First principles of concurrent engineering: A competitive strategy for product development. In: Report of the CALS/Concurrent Engineering Electronic System Task Group.

MacMillen, D., Camposano, R., Hill, D., Williams, T.W. 2000. An industrial view of electronic design automation. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions, Vol. 19, Issue 12, pp. 1428–1448.

Madisetti, V. 1995. VLSI Digital Signal Processors. Boston, MA, USA: Butterworth–Heinemann. 525 p. ISBN 0-7506-9406-8

Madsen, J., Grode, J., Knudsen, P., Petersen, M., Haxthausen, A. 1997. LYCOS: the Lungby Co-Synthesis System. Design Automation for Embedded Systems, Vol. 2, No.2, pp. 195–236.

Mai, K., Paaske, T., Jayasena, N., Ho, R., Dally, W., Horowitz, M. 2000. Smart Memories: a modular reconfigurable architecture. Proceedings of the 27th International Symposium on Computer Architecture. Vancouver, Canada, 10–14 June 2000. New York, NY, USA: ACM Inc. Pp. 161–171.

Malik, S., Martonosi, M., Li, Y. 1997. Static Timing Analysis of Embedded Software. Proceedings of 34th Design Automation Conference. Anaheim, CA, USA, 9–13 June 1997. New York, NY, USA: ACM Inc. Pp. 147–152.

Mangione-Smith, W., Hutchings, B., Andrews, D., DeHon, A., Ebeling, C., Hartenstein, R., Mencer, O., Morris, J., Palem, K., Prasanna, V., Spaanenburg, H. 1997. Seeking Solutions in Configurable Computing. Computer, Vol. 30, No. 12, pp. 38–43.

Martin, G. 2002. The Future of High-Level Modelling and System Level Design: Some Possible Methodology Scenarios. 9th IEEE/DATC Electronic Design Processes Workshop. Monterrey, CA, USA, 21–23 April 2002. Available at http://www.eda.org/edps/edp02/ 5 p.

Mazor, S. 1995. The History of The Microcomputer – Invention and Evolution. Proceeding of the IEEE, Vol. 83, No. 12, pp. 1601–1608.

McDermid, J. 1991. Software engineer's reference book. Oxford, UK: Butterworth. 1500 p. ISBN 0-750-61040-9

Mitra, B., Panda, P., Chaudhuri, P. 1993. Estimating the Complexity of Synthesized Designs from FSM Specifications. IEEE Design and Test of Computers, Vol. 10, No. 1, pp. 30–35.

Moore, G. 1965. Cramming more components onto integrated circuits. Electronics, April 1965, pp. 114–117.

Muchnick, S. 1997. Advanced Compiler Design and Implementation. San Francisco, CA, USA: Morgan Kaufmann Publishers. 856 p. ISBN 1-55860-320-4

Panda, P., Dutt, N., Nicolau, A. 1999. Local Memory Exploration and Optimization in Embedded Systems. IEEE Transactions on Computer-Aided Design of Circuits and Systems, Vol. 18, No. 1, pp. 3–13.

Panda, P., Dutt, N., Nicolau, A., Catthoor, F., Vandecapelle, A., Brockmeyer, E., Kulkarni, C., De Greef, E. 2001. Data Memory Organization and Optimizations in Application Specific Systems. IEEE Design and Test of Computers, Vol. 18, No. 3, pp. 56–68.

Parhi, K. 1999. VLSI Digital Signal Processing Systems – Design and Implementation. New York, NY, USA: John Wiley & Sons, Inc. 784 p. ISBN 0-471-24186-5

Patterson, D., Hennessy, J. 1998. Computer Organization and Design – The Hardware/Software Interface. 2. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers. 759 p. ISBN 1-55860-428-6

Pees, S., Hoffmann, A., Živojnović, V., Meyr, H. 1999. LISA – Machine Description Language for Cycle-Accurate Models of Programmable DSP Architectures. Proceedings of 36th Design Automation Conference. New Orleans, LA, USA, 21–26 June 1999. New York, NY, USA: ACM Press. Pp. 933–938.

Pelkonen, A., Soininen, J.-P., Rautio, T. 2001. Configurable DSP Engine for Internet Age, Proceedings of 19th IEEE Norchip Conference. Kista, Sweden, 12–13 November 2001. Copenhagen, Denmark: Technoconsult. Pp. 47–52.

Puschner, P., Koza, Ch. 1989. Calculating the Maximum Execution Time of Real-Time Programs. Journal of Real-Time Systems 1(2), pp. 160–176.

Riihijärvi, J., Mähönen, P., Saaranen, M., Roivainen, J., Soininen, J.-P. 2001 Providing Network Connectivity for Small Appliances: A Functionally Minimized Embedded Web Server. IEEE Communications Magazine, Vol. 39, No. 10, pp. 74–79.

Ross, P. 1988. Taguchi Techniques for Quality Engineering. New York, NY, USA: McGraw–Hill. 279 p. ISBN 0-07-053866-2

Rowson, J. Sangiovanni-Vincentelli, A. 1997. Interface-based design. Proceedings of 34th Design Automation Conference. Anaheim, CA, USA, 9–13 June 1997. New York, NY, USA: ACM Inc. Pp. 178–183.

Royce, W. 1970. Managing the Development of Large Software Systems. IEEE WESCON. Pp. 1–9.

Rumbaugh, J. 1991. Object-oriented modelling and design. Englewood Cliffs, NJ. USA: Prentice Hall. 500 p. ISBN 0-13-629841-9

Saastamoinen, I., Sigüenza-Tortosa, D., Nurmi, J. 2002. Interconnect IP Node for Future System-on-Chip Designs. Proceedings of the 1st IEEE International Workshop on Electronic Design, Test and Applications, 2002. Christchurch, New Zealand, 29–31 January 2002. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 116–120.

Salchak, P., Chawla, P. 1997. Supporting Hardware Trade Analysis and Cost Estimation Using Design Complexity. Proceedings of VHDL International Users' Forum. Arligton, VA, USA, 19–22 October, 1997. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 126–133.

Salminen, E. Lahtinen, V., Kuusilinna, K., Hämäläinen, T. 2002. Overview of bus-based system-on-chip interconnections Proceedings of International Symposium on Circuits and Systems, Vol. 2. Phoenix, AZ, USA 26–29 May 2002. Piscataway, NJ, USA: IEEE. Pp. 372–375.

Sciuto, D., Salice, F., Pomante, L., Fornaciari, W. 2002. Metrics for Design Space Exploration of Heterogeneous Multiprocessor Embedded System. Proceedings of 10th International Symposium on Hardware/Software Codesign. Estes Park, Colorado, USA, 6–8 May 2002. New York, NY, USA: ACM Press. Pp. 55–60.

Sgroi, M., Sheets, M., Mihal, A., Keutzer, K., Malik, S., Rabaey, J., Sangiovanni-Vincentelli, A. 2001. Addressing the system-on-a-chip interconnect woes through communication-based design. Proceedings of 38th Design Automation Conference. Las Vegas, NV, USA, 18–22 June 2001. New York, NY, USA: ACM Press. Pp. 667–672.

Sharma, A., Jain, R. 1993. Estimating architectural resources and performance for high-level synthesis applications. Very Large Scale Integration (VLSI) Systems, IEEE Transactions, Vol. 1, Issue 2, pp. 175–190.

Sipola, M., Soininen, J.-P., Kivelä, J. 1991. Systems real time analysis with VHDL generated from graphical SA-VHDL. Proceedings of the 2nd European Conference on VHDL Methods. Stockholm, Sweden, 8–11 September 1991. Stockholm, Sweden: Swedish Institute of Microelectronics. Pp. 32–38.

Sipola, M., Soininen, J.-P., Kivelä, J. 1992. Systems Real-Time Analysis with VHDL Generated from Graphical SA-VHDL. In: Mermet, J. (ed.). VHDL for Simulation, Synthesis and Formal Proofs of Hardware. Dortrecht, Netherlands: Kluwer Academic Publishers. 307 p. ISBN 0-7923-9253-1

Sivaram, R., Stunkel, C., Panda, D. 2002. HIPIQS: A High-Perfomance Switch Architecture Using Input Queuing. IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 3, pp. 275–289.

Skadron, K., Martonosi, M., August, D., Hill, M., Lilja, D., Pai, V. 2003. Challenges in Computer Architecture Evaluation. Computer, Vol. 36, No. 8, pp. 30–36.

Skillicorn, D. 1988. A Taxonomy for Computer Architectures. Computer, Vol. 21, No. 11, pp. 46–57.

Slater, M. 1996. The Microprocessor Today. IEEE Micro, Vol. 16, No. 6, pp. 32–44.

Smith, A. 1982. Cache Memories. ACM Computing Surveys, Vol. 14, pp. 473–530.

Smith, C., Frank, G., Cuadrado, J. 1985. An architecture design and assessment system for software/hardware codesign. Proceedings of the 22nd ACM/IEEE conference on Design automation. Las Vegas, NV, USA, June 1985. New York, NY, USA: ACM Press. Pp. 417–424.

Smith, M. 1997. Application-Specific Integrated Circuits. Boston, MA, USA: Addison–Wesley. 1026 p. ISBN 0-201-50022-1

Soininen, J.-P. 1997. Asiakastarvelähtöinen elektroniikkatuoteperheen suunnittelu (Customer oriented design of electronic product family). Licenciate Thesis, University of Oulu. 102 p.

Soininen, J.-P. 1999. Customer oriented product families. ITpress Series on Applied Computer Science and Technology. CONSYSE '97, International Workshop on Conjoint Systems Engineering. Buchenrieder, K & Sedlmeier, A. (eds). Bruchsal, Germany: ITpress. Pp. 19–31.

Soininen, J.-P., Heusala, H. 2003. A Design Methodology for NoC-based Systems. In: Jatsch, A., Tenhunen, H. (ed.). Networks on Chip. Boston, MA, USA: Kluwer Academic Publishers. Pp. 19–38. ISBN 1-4020-7392-5

Soininen, J.-P., Huttunen, T., Saarikettu, J., Melakari, K., Ong, A., Tiensyrjä, K. 1998a. InCo – An interactive codesign framework for real-time embedded systems. Espoo, Finland: VTT. VTT Publications 344. 206 p.

Soininen, J.-P., Huttunen, T., Tiensyrjä, K., Heusala, H. 1995. Cosimulation of real-time control systems. Proceedings of European Design Automation Conference. Brighton, UK, 18–22 September 1995. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 170–175.

Soininen, J.-P., Melakari, K., Tiensyrjä, K. 1998b. Software/hardware codesign for platform chips. Proceedings of Embedded Systems Conference 1998. San Jose, CA, USA, 1–5 November 1998. San Francisco, CA, USA: Miller Freeman. 16 p. (CD-ROM)

Soininen, J.-P., Purhonen, A., Rautio, T., Kasslin, M. 2001. Mobile multi-mode terminal: making trade-offs between software and fixed digital radio, In: Del Re, E. (ed.), Software Radio, Technologies and Services. London, UK: Springer-Verlag. Pp. 237–249. ISBN 1-85233-346-4

Soininen, J.-P., Saarikettu, J., Veijalainen, V., Huttunen, T. 1997. TripleS – a formal validation environment for functional specifications. In: Delgado Kloos, C., Cerny, E. (eds.). Hardware description languages and their application: Specification, modelling, verification and synthesis of microelectronic systems. IFIP TC10WG10. 5th International Conference on Computer Hardware Description Languages and their Applications. Toledo, Spain, 20–25 April 1997. London, UK: Chapman & Hall. Pp. 83–85. ISBN 0-412-78810-1

Soininen J.-P., Sipola M., Tiensyrjä K. 1989. SW/HW – Partitioning of real-time embedded systems. Microprocessing & Microprogramming, Euromicro Journal, Vol. 27, No. 1–5, pp. 239–244.

Staunstrup, J., Wolf, W. 1997. Hardware/software co-design: principles and practice. Boston, MA, USA: Kluwer Academic Publishers. 395 p. ISBN 0-7923-8013-4

Suzuki, K., Sangiovanni-Vincentelli, A. 1996. Efficient Software Performance Estimation Methods for Hardware/Software Codesign. Proceedings of 33rd

Design Automation Conference. Las Vegas, NV, USA, 3–7 June 1996. New York, NY, USA: ACM Press. Pp. 605–610.

Sylverster, D., Keutzer, K. 1999. Rethinking Deep-Submicron Circuit Design. Computer, Vol. 32, No. 11, pp. 25–33.

Taylor, M., Kim, J., Miller, J., Wentzlaff, D., Ghodrat, F., Greenwald, B., Hoffman, H., Johnson, P., Lee, J.-W., Lee, W., Ma, A., Saraf, A., Seneski, M., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S., Agarwal, A. 2002. The Raw microprocessor: a computational fabric for software circuits and general-purpose programs. IEEE Micro, Vol. 22, No. 2, pp. 25–35.

Tichy, W. 1997. A Catalogue of General-Purpose Software Design Patterns. Proceedings of Technology of Object-Oriented Languages and Systems, TOOLS 23. Santa Barbara, CA, USA, 28 July – 1 August, 1997. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 330–339.

Walker, R., Camposano, R. 1991. A Survey of High-Level Synthesis Systems. Boston, MA, USA: Kluwer Academic Publishers. 182 p. ISBN 0-7923-9158-6

Wang, A., Killian, E., Maydan, D., Rowen, C. 2001. Hardware/Software Instruction Set Configurability for System-on-Chip Processors. Proceedings of 38th Design Automation Conference. Las Vegas, NV, USA, 18–22 June 2001. New York, NY, USA: ACM Press. Pp. 184–188.

Ward, J., Barton, B., Roberts, J., Stanier, B. 1984. Figures of Merit for VLSI implementations of digital signal processing algorithms. IEE Proceedings, Vol. 131, Part F, No. 1, pp. 64–70.

Ward, P. 1986. Transformation schema: An extension of the data flow diagram to represent control and timing. IEEE Transactions on Software Engineering, Vol. SE-12, No.2, pp. 198–210.

Wheelwright, S., Clark, K. 1992. Revolutionizing product development: quantum leaps in speed, efficiency and quality. New York, NY, USA: Free Press. 364 p. ISBN 0-02-905515-6

Vermeulen, B., Dielissen, J., Goossens, K., Ciordas, C. 2003. Bringing Communication Networks on a Chip: Test and Verification Implications. IEEE Communications Magazine, Vol. 41, No. 9, pp. 74–81.

Wielage, P., Goossens, K. 2002. Networks on Silicon: Blessing or Nightmare? Proceedings of Euromicro Symposium on Digital System Design. Dortmund, Germany, 4–6 September 2002. Los Alamitos, CA, USA: IEEE Computer Society Press. Pp. 196–200.

Wingard, D. 2001. MicroNetwork-Based Integration for SOCs. Proceedings of 38th Design Automation Conference. Las Vegas, NV, USA, 18–22 June 2001. New York, NY, USA: ACM Press. Pp. 673–677.

Wolf, F., Ernst, R. 2000. Intervals in Software Execution Cost Analysis. Proceedings of 13th International Symposium on System Synthesis. Madrid, Spain, 20–22 September 2000. Los Almitos, CA, USA: IEEE Computer Society Press. Pp. 130–135.

Wolf, T., Turner, J. S. 2001. Design Issues for High-Performance Active Routers. IEEE Journal on Selected Areas in Communications, Vol. 19, No. 3, pp. 404–409.

Wolf, W. 2001. Computers as components: principles of embedded computing system design. San Francisco, CA, USA: Morgan Kauffman Publishers. 662 p. ISBN 1-55860-541-X

Wolf, W. 2003. A Decade of Hardware/Software Codesign. Computer, Vol. 36, No. 4, pp. 38–43.

Zave, P. 1982. An operational approach to requirements specification for embedded systems. IEEE Transactions on Software Engineering, Vol. SE-8, No. 3, pp. 250–269.

Živojnović, V., Meyr, H. 1996. Compiled SW/HW Cosimulation. Proceedings of 33rd Design Automation Conference. Las Vegas, NV, USA, 3–7 June, 1996. New York, NY, USA: ACM Press. Pp. 690–695.

Appendices of this publication are not included in the PDF version. Please order the printed version to get the complete publication (http://www.vtt.fi/inf/pdf/)

Published by

Series title, number and report code of publication

VTT Publications 523
VTT–PUBS–523

Author(s)
Soininen, Juha-Pekka

Title

Architecture design methods for application domain-specific integrated computer systems

Abstract

The role of the single computer inside application-specific integrated circuits is changing with the increasing capacity of semiconductor technology. The system functionality can be partitioned to a set of communicating application domain-specific computers instead of developing the most efficient general-purpose computers that fulfil all kinds of computing needs. The main design challenges are the complexity and diversity of application-domains and the complexity of platforms which can provide enough capacity for those applications.

The architecture design methods presented in this thesis are targeted at application domain-specific computers that are implemented as integrated circuits. Backbone-platform-system design methodology separates the technology, platform design efficiency and application development problems from each other. It also provides a system design framework for the architecture design methods presented. The methods are based on complexity, mappability, and capacity-based quality estimations that are used as decision support and quality validation tools. Abstract models of both applications and architectures enable rapid estimations and adequate coverage in design space exploration.

The methods have been applied to various case examples. Complexity-based estimation provided a systematic approach to the selection of an architecture template that takes into account the changes in technologies and design efficiency. Mappability-based processor-algorithm quality estimation enabled us to study more than 10,000 processor architectures for WLAN modem transceiver example. Capacity-based quality estimation was used in the performance evaluation of configurable multiprocessor architecture. In all cases the respective simulations using for example instruction-set simulators would have taken much longer and required advanced post-processing of results.

Keywords
decision support methods, quality estimations, mappability estimation, platform based design

Activity unit
VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland

ISBN			Project number
951–38–6363–8 (soft back ed.) 951–38–6364–6 (URL:http://www.vtt.fi/inf/pdf/)			
Date April 2004	Language English	Pages 118 p. + app. 51 p.	Price D
Name of project MULTICS, SCIFI, NOCARC		Commissioned by The National Technology Agency (Tekes)	
Series title and ISSN VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)		Sold by VTT Information Service P.O.Box 2000, FIN–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374	

It has become technologically possible to implement several computer systems into a single chip presuming that the design complexity and power efficiency challenges can be conquered. With these billion-transistor ASICs, we have to adopt new on-chip network-based system architectures and to extend the level of design reuse into the use of predesigned architectural platforms. The new architecture paradigm and increasing complexity of applications introduce new challenges for architecture design methods for embedded and mobile systems.

The presented backbone-platform-system design methodology helps in encapsulating circuit design, platform architecure design and application development phases, which makes the management of complexity easier. It also provides a system design framework for the new architecture design methods that are used for decision support and quality validation. The complexity-based estimation is used for the system scaling. The mappability-based estimation is used for the selection of computing resources. The capacity-based estimation is used for supporting application mapping and architecture validation. The main objective is to provide means for rapid evaluation of design alternatives at early phases of design.

Tätä julkaisua myy
VTT TIETOPALVELU
PL 2000
02044 VTT
Puh. (09) 456 4404
Faksi (09) 456 4374

Denna publikation säljs av
VTT INFORMATIONSTJÄNST
PB 2000
02044 VTT
Tel. (09) 456 4404
Fax (09) 456 4374

This publication is available from
VTT INFORMATION SERVICE
P.O.Box 2000
FIN–02044 VTT, Finland
Phone internat. +358 9 456 4404
Fax +358 9 456 4374

ISBN 951–38–6363–8 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6364–6 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)