Modeling NoC Architectures by Means of Deterministic and Stochastic Petri Nets

H. Blume, T. von Sydow, D. Becker, and T.G. Noll

Chair for Electrical Engineering and Computer Systems,
RWTH Aachen University, Schinkelstraße 2, 52062 Aachen
{blume, sydow, dbecker, tgn}@eecs.rwth-aachen.de

Abstract. The design of appropriate communication architectures for complex Systems-on-Chip (SoC) is a challenging task. One promising alternative to solve these problems are Networks-on-Chip (NoCs). Recently, the application of deterministic and stochastic Petri-Nets (DSPNs) to model on-chip communication has been proven to be an attractive method to evaluate and explore different communication aspects. In this contribution the modeling of basic NoC communication scenarios featuring different processor cores, network topologies and communication schemes is presented. In order to provide a test bed for the verification of modeling results a state-of-the-art FPGA-platform has been utilized. This platform allows to instantiate a soft-core processor network which can be adapted in terms of communication network topologies and communication schemes. It will be shown that DSPN modeling yields good prediction results at low modeling effort. Different DSPN modeling aspects in terms of accuracy and computational effort are discussed.

1 Introduction

With the advent of heterogeneous Systems-on-Chip (SoCs), on-chip communication issues are becoming more and more important. As the complexity of SoCs is increasing a variety of appropriate communication architectures are discussed, ranging from basic bus oriented to highly complex packet oriented Network-on-Chip (NoC) structures [1], [2]. These communication structures have to be evaluated and quantitatively optimized presumably in an early stage of the design process. Various approaches have been developed in order to evaluate SoC communication performance and to compare architectural alternatives. Examples for such communication modeling approaches are:

- simulative approaches, e. g. applying SystemC [3], [4],
- combined simulative-analytic approaches [5],
- formal communication modeling and refinement systems applying dedicated modeling languages like the Action Systems Formalism [6],
- stochastic approaches applying Markov Models [7], Queuing Theory [8] or deterministic and stochastic Petri Nets [9], [10].

Each of these techniques provides its individual advantages and disadvantages. For example, simulative approaches like [3] provide highly accurate results but suffer from

T.D. Hämäläinen et al. (Eds.): SAMOS 2005, LNCS 3553, pp. 374–383, 2005.

long simulation times, making them not appropriate for an early stage of communication modeling and evaluation. Recently, communication modeling approaches which are based on so-called deterministic and stochastic Petri-Nets (DSPNs) have been presented. In [9], it could be shown that applying these DSPN modeling techniques it is possible to efficiently trade modeling effort and modeling accuracy. Applying very simple but exemplary test scenarios like resource conflicts in state-of-the-art DSP architectures and basic bus-based communication test cases a very good modeling accuracy with low modeling effort could be achieved. In order to prove that these techniques are also suitable for more complex communication scenarios the application of these DSPN-based modeling techniques to NoC-communication problems like on-chip multi-processor communication is investigated in this paper. In order to verify the modeling results a generic NoC test bed has been built first. Therefore, an FPGA-based platform has been utilized on which several proprietary so-called soft-core processors (Nios [11]) besides other components like DMA-controllers, on-chip memories or dedicated (custom-made) logic blocks can be instantiated and connected using different communication architectures. The FPGA based generic platform allows to determine NoC performance in terms of latency, throughput respectively bandwidth etc. These results will be compared to the results which were achieved by the DSPN model. Due to limited hardware resources, highly complex communication scenarios with for example different topologies of processor clusters (different hierarchy levels etc.) are not emulated using RISC-like Nios soft-cores. Instead, such processor networks were tested with rudimentary processor cores which implement the functionality of a sender and/or a receiver. The following issues have been addressed within this contribution: Modeling effort, modeling accuracy and required computation time depending on the DSPN solving methods. The paper is organized as follows: Chapter 2 sketches the basics of DSPN modeling. In chapter 3 some details on the FPGA-based test bed which has been utilized in the experiments are described. The modeling of NoC test scenarios and the corresponding results are described in chapter 4. Conclusions are given in chapter 5.

2 Short Synopsis of Modeling with DSPNs

A comprehensive overview of the modeling possibilities with deterministic and stochastic Petri nets (DSPNs) and all corresponding options are not in the scope of this paper. Here, only those basics will be shortly sketched which are used in the following sections. DSPNs consist of so-called places, arcs and transitions. Places, depicted as circles in the graphical representation, are states of system components. E.g. a place could be named *compute result* to make clear that this place represents the state of computing a result in the belonging component. Places can be untagged or marked with one or more tokens which illustrate that the corresponding place is actually allocated. The change of a state can be described by so-called transitions. Three types are differentiated: There are immediate transitions, transitions with a probability density function for the delay (e.g. negative exponential) or deterministically delayed transitions. Furthermore, priorities can be assigned to each transition. Transitions and places are connected via arcs. Arcs can be of two types, regular or inhibitor. Inhibitor arcs are identified by a small inversion circle instead of an arrowhead at the destination end of it (see Fig. 1). If more

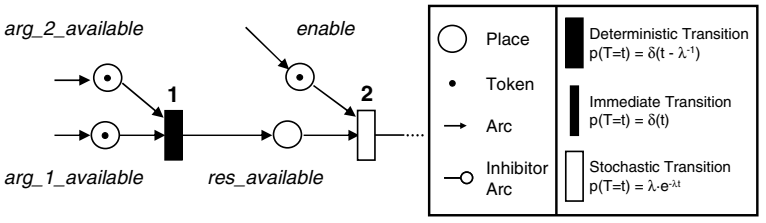

Fig. 1. Exemplary Section out of a deterministic and stochastic Petri-net model

than one input place is connected to a transition via regular arcs, the transition will only be activated when all connected places are marked. If one or more of these arcs is an inhibitor arc the transition will not fire if the corresponding place is marked. Once a Petri net model is implemented, performance measures, such as marking probabilities and the expected number of tokens for places and throughput for deterministic and exponential transitions can be defined and subsequently computed by simulation, mathematical approximation or mathematical analysis.

The different alternatives vary in terms of accuracy and computational effort. E.g. in the case of two or more concurrently enabled deterministic transitions, mathematical analysis is not possible [12]. Fig. 1 depicts a section of a very simple modeling example where the transition 1 will fire only when both connected places (*arg_1_available*, *arg_2_available*) contain at least one token. Then this transition will fire with a deterministic delay time, modeling the processing time for e.g. a computational kernel. The corresponding delay time T1 for this transition is a configuration parameter of this DSPN. After this delay time has been elapsed, one token will be taken from all of the transitions input places (here: *arg_1_available* and *arg_2_available*), and one token will be placed in all connected output places (here only one: *res_available*). Depending on the status of its other input places (here: *enable*) - the next transition 2 (in this case a stochastic transition) is going to fire with a random delay with an exponential distribution. In principle, each communication scenario can be modeled by DSPNs. Some SoC modeling examples can be found in [9] and a comprehensive overview of modeling with DSPNs is given in [12]. A variety of DSPN modeling environments is available today [13]. In the course of the modeling experiments described here, the DSPN modeling environment DSPNexpress [14] has been used. DSPNexpress provides a graphical editor for DSPN models, as well as a solver backend for numerical analysis of DSPNs. Experiments can be performed for a fixed parameter set and for a parameter sweep across a user-defined range of values. The package supports the computation of the transient response e.g. the distribution of tokens after a certain amount of cycles (using Picards Iteration Algorithm) as well as computation of the steady state behavior of the realized DSPN model. The latter can be realized by iteratively using the Generalized Minimal Residual Method, by employing the direct quadrature method or by utilizing the discrete event simulator backend [12]. These methods correspond to the DSPN computation methods mentioned in the beginning of this chapter.

3 FPGA Based NoC Testbed

As a test bed for the evaluation of NoC communication scenarios and architectures an FPGA-based system has been applied. As a reference platform an FPGA development board featuring an APEX20K200EFC484 FPGA, 1 MByte of flash memory, 256 KBytes of SRAM, serial interfaces (e.g. for downloading the program to the on-board flash memory) and several display options (e. g. LC display) has been used. Multi processor networks have been implemented on this platform by instantiating Nios soft-core processors. The Nios embedded processor is a general-purpose load/store RISC CPU, that can be combined with a number of peripherals, custom instructions, and hardware acceleration units to create a custom system-on-a-programmable-chip solution. The processor can be configured to provide either 16 or 32 bit wide registers and data paths to match given application requirements. Both versions use 16 bit wide instruction words. Version 3.2 of the Nios core, as used here, typically features about 1100 logic elements (LEs) in 16 bit mode and up to 1700 LEs in 32 bit mode incl. hardware accelerators like hardware multipliers. More detailed descriptions of the components can be found in [11]. Based on such a Nios core a processor network consisting of a general communication structure that interfaces various peripherals and devices to various Nios cores can be constructed. The so-called Avalon [15] structure is used to connect devices to the Nios cores. It is a dynamic sizing communication structure that allows devices with different data widths to be connected, with a minimal amount of interfacing logic. In order to realize processor networks on this platform the so-called SOPC (system on a programmable chip) Builder [16] has been applied. It is a tool for composing heterogeneous architectures including the communication structure out of library components such as CPUs, memory interfaces, peripherals and user-defined blocks of logic. The SOPC Builder generates a single system module that instantiates a list of user-specified components and interfaces incl. an automatically generated interconnect logic. It allows modifying the design components, adding custom instructions and peripherals to the Nios embedded processor and configuring the connection network.

4 Modeling NoC Test Scenarios

4.1 Nios-Based NoC Test Scenario

The first analyzed system is composed of two Nios soft-cores which compete for access to an external shared memory (SRAM) interface. Each core is also connected to a private memory region containing the program code and to a serial interface which is used to ensure communication with the host PC. The proprietary communication structure used to interconnect all components of a Nios-based system is called Avalon [15] which is based on a flexible crossbar architecture. The block diagram of this fundamental resource sharing experiment is depicted in Fig. 2. Whenever multiple masters can access a slave resource, SOPC Builder [16] automatically inserts the required arbitration logic. In each cycle when contention for a particular slave occurs, access is granted to one of the competing masters according to a Round Robin arbitration scheme. For each slave, a so-called share is assigned to all competing masters. This share represents the fraction of contention cycles in which access is granted to this corresponding master. Masters

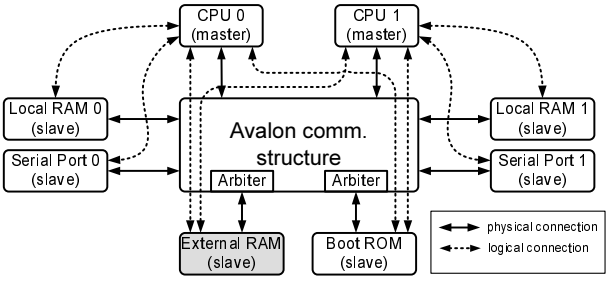

Fig. 2. Block diagram of fundamental resource sharing experiment

incur no arbitration delay for uncontested or acquired cycles. Any masters that were denied access to the slave automatically retry during the next cycle, possibly leading to subsequent contention cycles.

In the modeled scenario the common slave resource for which contention occurs is a shared external memory unit (shaded in gray in Fig. 2) containing data to be processed by the CPUs. Within the scope of this fundamental resource sharing scenario several experiments with different parameter setups have been performed to prove the validity of the DSPN modeling approach. Adjustable parameters include e.g. the priority shares assigned to each processor, the ratio of write and read accesses, the mean delay between memory accesses etc. These parameters have been used to model typical communication requirements of basic operators like digital filters or block read and write operations running on these processor cores. In addition, an experiment simulating a more generic, stochastic load pattern, with exponentially distributed times between two attempts of a processor to access the memory has been performed. Here, each memory access is randomly chosen to be either a read or a write operation according to user-defined probabilities. The distinction between load and store operations is important here because the memory interface can only sustain one write access every two cycles, whereas no such limitation exists for read accesses. The various load profiles were implemented in C, compiled on the host PC and the resulting object code has been transferred to the Nios cores via the serial interface for execution. In the case of the generic load scenario, the random values for the stochastic load patterns were generated in a MATLAB routine. The determined parameters have been used to generate C code sequences corresponding to this load profile. The time between two attempts of a processor to access the memory has been realized by inserting explicit NOPs into the code via inline assembly instructions. Performance measurements for all scenarios have been achieved by using a custom cycle-counter instruction added to the instruction set of the Nios cores. In a first step, a *simple* DSPN model has been implemented (see Fig. 3) in less than two hours. Distinction between read and write accesses was explicitly neglected to achieve a minimum modeling complexity. The DSPN consists of four sub-structures; two parts represent the load generated by the Nios cores (**CPU #1** and **#2**), a simple **cycle process subnet** providing a clock signal and the most complex part being the **arbitration subnet**. Altogether this simple model includes 19 places and 20 transitions. The immediate transitions T1, T2 and T3 and the belonging places P1,

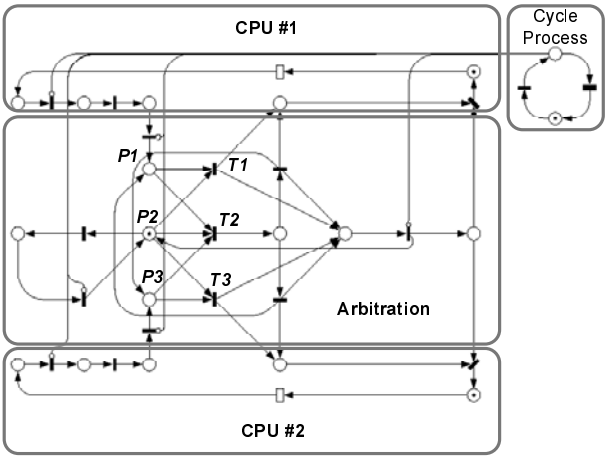

Fig. 3. DSPN for rudimentary Nios example (cycle generator)

P2 and P3 (see Fig. 3) are an essential part of the Round Robin arbitration mechanism implemented in this DSPN. The marked transition P2 denotes that the memory is ready and memory access is possible. P1 and P3 belong to the CPU load processes and indicate that the corresponding CPU (#1, #2) tries to access the memory. If P1 and P2 or P3 and P2 are tagged the transition T1 or accordingly transition T3 will fire and remove the tokens from the connected places (P1, P2 or P2, P3). CPU #1 or CPU #2 has been assigned the memory access in this cycle. A collision occurs if P1, P2 and P3 are tagged with a token. Both CPUs try to access the memory in the same cycle (P1 and P3 marked). Furthermore, the memory is ready to be accessed (P2 marked). A higher priority has been assigned to transition T2 during the design process. This means that if the conditions for all places are equal the transition with the highest priority will fire first. Therefore, T2 will fire and remove the tokens from the places. Thus, the transitions T1, T2 and T3 and the places P1, P2 and P3 handle the occurrence of a collision.

Though the modeling results applying this simple DSPN model are quite accurate (relative error less than 10%, see Fig. 4), it is possible to increase the accuracy even more by extending the modeling effort for the arbitration subnet. For example it is possible to design a DSPN model of the arbitration subnet which properly reflects the differences between read and write cycles. Thus, the arbitration of write and read accesses has been modeled in different processes resulting in different DSPN subnets. This results in a second and more complex DSPN model. The implementation of this *complex* model has taken about three times the effort in terms of implementation time (approximately five hours) than the simpler model described before. Now, the DSPN model consists of 48 transitions and 45 places. Compared to the simple model the maximum error has been further reduced (see Fig. 4). The complex model also properly captures border cases caused e. g. by block read and write operations. The throughput measured for a code sequence containing 200 memory access instructions has been compared to the results of the simple and complex DSPN model. Fig. 4 shows the relative error for

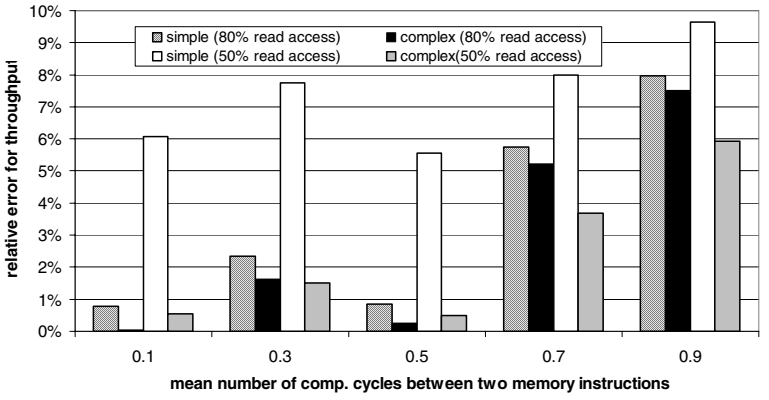

Fig. 4. Comparison of simple and complex DSPN model for different load scenarios

the throughput which is achieved by varying the mean number of computation cycles between two attempts of a processor to access the memory. Furthermore, the results are depicted for two cases, where 50% and 80% of the memory accesses are read accesses, respectively. For example in case of 50% read accesses and a mean of 0.9 computation cycles between two memory accesses, the maximum relative error is reduced from about 10% achieved with the simple model to about 6% with the complex model. As mentioned before, the evaluation of DSPNs can be performed by different methods. The effort in terms of computation time has been compared for a couple of experiments. Generally, the time consumed when applying the simulation method is about two orders of magnitude longer than the time consumed by the analysis methods (direct and iterative). For the example of the complex model the computation time of the DSPN analysis method only amounts to 0.3 sec. and the DSPN simulation time (10^7 memory accesses) amounts to 30 sec. on a Linux-based PC (2.4 GHz, 1 GByte of RAM).

4.2 Generic Processor Core Network

As an example for a more complex network, the DSPN model for a hierarchical network featuring three processor cores, three memory sections and two hierarchically connected switches has been developed. All components are connected by full-duplex links capable of transmitting one data word each clock cycle (see Fig. 5). Because of the limited amount of available logic elements, this network cannot be implemented using Nios cores. Furthermore, SOPC Builder only provides limited configuration options with regard to the Avalon structure in the form of priority shares, and specifically cannot accommodate hierarchical interconnects. Therefore, a flexible generic NoC testbed has been implemented to facilitate implementation of complex networks and arbitrary interconnect structures. To keep the logic element count at a tractable level, the network components are reduced to provide only the functionality that directly affects network traffic. For example, simplified load generators acting as pseudo-CPUs are used instead of Nios cores. Implementation of different routing and arbitration schemes is possible by realizing corresponding interconnection components, e.g. switches. In the example

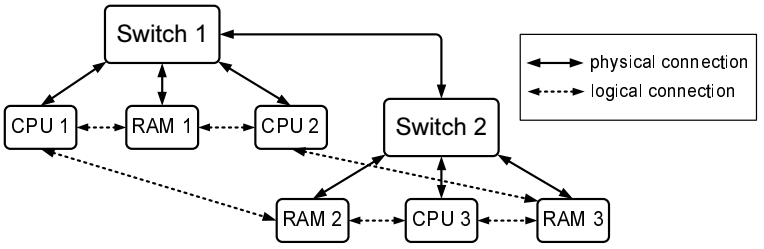

Fig. 5. Exemplary NoC architecture

discussed here, write accesses consist of a single request going from the CPU to the target memory section, whereas read accesses consist of two transactions: An initial request by the CPU, followed by a response from the memory section. The entities for which arbitration has to take place are the output ports of the switches. Access is granted according to a Round Robin scheme with equal priorities. Upon contention, all masters that have been denied access wait until the port is available again. A single-cycle access delay is incurred for the winning master or in the absence of contention. Initiating masters successively acquire all output ports on the path to the addressed slave, and release them once the whole transaction is completed. For the modeled experiment, the processors generate a memory access pattern that is stochastic in space and time: The addressed memory section for each access is determined based on a Bernoulli trial, and the time between consecutive memory accesses for each CPU is exponentially distributed. CPU 1 and 2 generate both local and remote memory accesses, whereas CPU 3 only accesses local memory sections. Fig. 5 sketches the network topology and the logical connections between components.

The DSPN developed to model this setup consists of 64 places and 61 transitions, and provides several parameters defining the network behavior that have been used to set up various experiments. Fig. 6 shows results from two examples: In the experiment whose results are depicted in Fig. 6 a), the ratio between local and remote memory accesses of CPU 1 was varied, and the resulting throughput for all three CPUs was measured. The results show that the mean throughput for CPU 1 is reduced with increasing percentage of remote accesses. This is obvious from the fact that remote transactions require acquisition of two switch ports, whereas local transactions require only one. CPUs 2 and 3, on the other hand, undergo a slight decrease in throughput. This is because CPU 2 and 3 become more likely to collide with CPU 1 while trying to access the output port from switch 1 to switch 2 and memory section 3, respectively. In Fig. 6 b) the mean computational delay between consecutive memory accesses at each CPU was varied. As a performance measure, the ratio between actual measured throughput and the upper bound for the throughput (i. e. assuming instantaneous memory accesses) is depicted. CPU 1 and CPU 2 were assigned fixed remote memory access probabilities (P_{rma}) of 50% and 20%, respectively. The figure shows that longer delays between memory accesses lead to throughput values closer to the upper bound. This can be explained by the fact that collisions become less likely, and that the delay caused by a collision becomes smaller relative to the computational delay. It can also be seen that

the increase in throughput occurs faster for CPUs that have a higher percentage of local memory accesses. As in the previous experiment, this is caused by the additional requests for switch output ports required for remote accesses. Modeling of this hierarchical network took about eight hours. Evaluation of the DSPNs took approximately half a second on the Linux PC mentioned in the previous section, whereas simulation required about 30 sec. on the same machine.

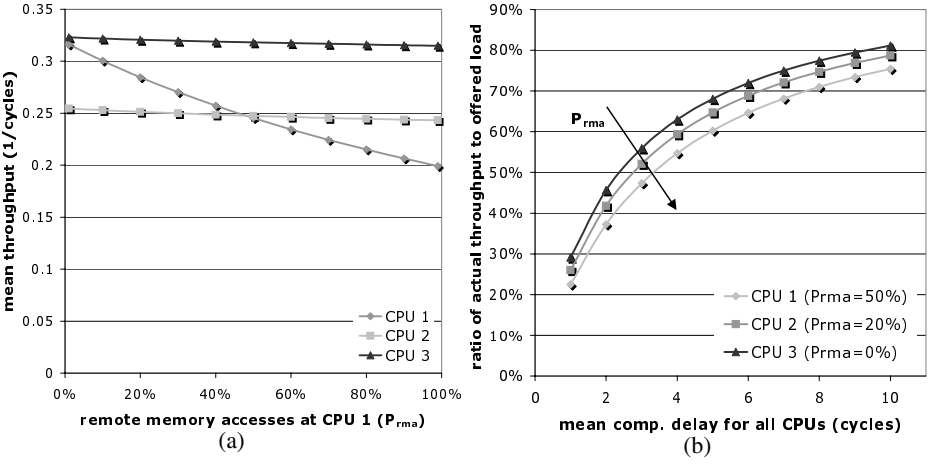

Fig. 6. Results for the modeling of the hierarchical NoC: a) mean throughput vs. percentage of remote memory accesses b) ratio of actual throughput to offered load vs. mean computational delay

This example shows how important design parameters of an NoC can be explored in an early stage of the NoC design flow by application of DSPN modeling techniques. For subsystems which require higher accuracy, successive refinement of the corresponding DSPN subnets can be applied to gain additional accuracy.

5 Conclusion

The DSPN modeling of basic NoC architectures has been presented in this paper. Applying Nios soft core processors which are connected via an Avalon communication structure it could be shown that the modeling results are very close to the values measured on an FPGA test bed. For this example it could also be shown that modeling effort can be efficiently traded for modeling accuracy. Quantitative results for modeling effort and computational complexity have been presented. Furthermore, a more complex hierarchical NoC has been modeled and the influence of parameters like the distribution of read and write accesses has been studied. This example shows that DSPNs can be leveraged for early stage modeling of communication processes.

References

1. Jantsch, A., Tenhunen, H.: Networks on Chip. Kluwer Academic Publishers (2003)
2. Nurmi, J., Tenhunen, H., Isoaho, J., Jantsch, A.: Interconnect Centric Design for Advanced SoC and NoC. Kluwer Academic Publishers (2004)
3. Kogel, T., Doerper, M., Wieferink, A., Leupers, R., Ascheid, G., Meyr, H., Goossens, S.: A modular simulation framework for architectural exploration of on-chip interconnection networks. In: CODES+ISSS. (2003) 7–12
4. Madsen, J., Mahadevan, S., Virk, K.: Network-centric system-level model for multiprocessor soc simulation. In Nurmi, J., ed.: Interconnect Centric Design for Advanced SoC and NoC. Kluwer Academic Publishers (2004)
5. Lahiri, K., Raghunathan, A., Dey, S.: System-level performance analysis for designing on-chip communication architectures. IEEE Trans. on CAD of Integrated Circuits and Systems **20** (2001) 768–783
6. Plosila, J., Seceleanu, T., Sere, K.: Network-centric system-level model for multiprocessor soc simulation. In Nurmi, J., ed.: Interconnect Centric Design for Advanced SoC and NoC. Kluwer Academic Publishers (2004)
7. Mickle, M.H.: Transient and steady-state performance modeling of parallel processors. Applied Mathematical Modelling **22** (1998) 533–543
8. Kleinrock, L.: Queueing Systems. Volume 1. John Wiley and Sons (1975)
9. Blume, H., von Sydow, T., Noll, T.G.: Performance analysis of soc communication by application of deterministic and stochastic petri nets. In: SAMOS. (2004) 484–493
10. Ciardo, G., Charkasova, L., Kotov, V., Rokicki, T.: Modeling a scalable high-speed interconnect with stochastic petri nets. In: Proc. of the Sixth International Workshop on Petri Nets and Performance Models PNPM'95. (1995) 83–94
11. Altera: Nios Embedded Processor Software Development Reference Manual. (2001)
12. Lindemann, C.: Performance Modeling with Deterministic and Stochastic Petri Nets. JOHN WILEY AND SONS (1998)
13. Petri net tools data base: http://www.daimi.au.dk/PetriNets. (2004)
14. DSPNexpress: http://www.dspnexpress.de. (2003)
15. Altera: Avalon: Bus specification manual, http://www.altera.com/literature/manual/mnl_avalon_bus.pdf. (2003)
16. Altera: SOPC Builder, http://www.altera.com/products/software/products/sopc/sop-index.html. (2004)