Available online at www.sciencedirect.com

ScienceDirect

Journal of Systems Architecture 53 (2007) 466–476

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Application of deterministic and stochastic Petri-Nets for performance modeling of NoC architectures

H. Blume [*], T. von Sydow, D. Becker, T.G. Noll

Chair for Electrical Engineering and Computer Systems, RWTH Aachen University, Schinkelstr. 2, 52062 Aachen, Germany

Received 31 March 2006; received in revised form 6 November 2006; accepted 6 November 2006
Available online 21 December 2006

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 testbed 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 communication performance prediction results at low modeling effort. Different DSPN modeling aspects in terms of accuracy and computational effort are discussed.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Deterministic and stochastic Petri-Nets; Performance Modeling; NoC; SoC; On-chip communication

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 and sometimes application specific Network-on-Chip (NoC) structures [5,6,12,20]. 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

- emulation on FPGA-based platforms [19],
- simulative approaches, e.g. applying SystemC [14,17],

[*] Corresponding author. Tel.: +49 2418097591; fax: +49 2418092282.
E-mail addresses: blume@eecs.rwth-aachen.de (H. Blume), sydow@eecs.rwth-aachen.de (T. von Sydow), becker@eecs.rwth-aachen.de (D. Becker), tgn@eecs.rwth-aachen.de (T.G. Noll).

- combined simulative-analytic approaches [15],
- formal communication modeling and refinement systems applying dedicated modeling languages like the Action Systems Formalism [22],
- stochastic approaches applying Markov Models [18], Queuing Theory [13] or deterministic and stochastic Petri-Nets [9,10,8].

Each of these techniques provides its individual advantages and disadvantages. For example, simulative approaches based on SystemC like [14] provide highly accurate results but suffer from long simulation times, making them not appropriate for an early stage of communication modeling and evaluation. Emulation of communication architectures and scenarios on FPGA-based platforms [19] provides on one hand the possibility to acquire fastly results for different aspects. On the other hand the modeling and realization effort of the emulation is very high. The complexity of the modeled scenarios is bounded to the capacity of the used FPGA. 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. Basic but exemplary test scenarios like resource conflicts in state-of-the-art DSP architectures and basic bus-based communication test cases demonstrate a very good modeling accuracy at low modeling effort. In this paper we investigate and prove the usability of these techniques for more complex communication scenarios. Therefore, DSPN-based modeling techniques are adapted to NoC-communication problems like on-chip multi-processor communication.

In order to verify the modeling results a generic NoC testbed has been built first. Therefore, an FPGA-based platform has been utilized on which several proprietary so-called soft-core processors (Nios [1]) 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. 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: Section 2 sketches the basics of DSPN modeling. In Section 3 some details on the FPGA-based testbed which has been utilized in the experiments are described. The modeling of NoC test scenarios and the corresponding results are described in Section 4. Conclusions are given in Section 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. For example 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 currently 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 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. For example in the case of two or more concurrently

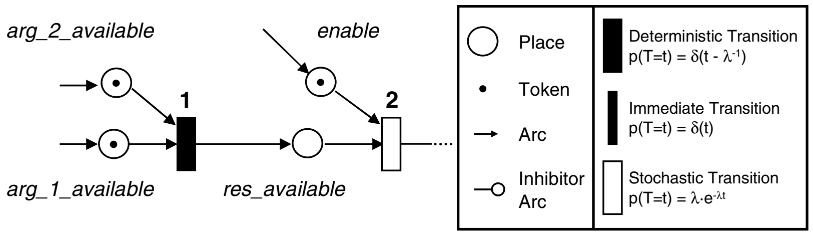
arg_2_available
enable
1
2
arg_1_available
res_available
Place
Token
Arc
Inhibitor Arc
Deterministic Transition p(T=t) = δ(t - λ⁻¹)
Immediate Transition p(T=t) = δ(t)
Stochastic Transition p(T=t) = λ·e^{-λt}

Fig. 1. An example of a section from a deterministic and stochastic Petri-Net model.

enabled deterministic transitions, mathematical analysis is not possible [16]. Fig. 1 depicts a section of a very basic 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 (in this case a stochastic transition) is going to fire with a random delay with an exponential distribution.

In principle, any 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 [16]. A variety of DSPN modeling environments is available today [21]. For the modeling experiments described in this contribution, we have applied the DSPN modeling environment DSPNexpress [11]. 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 Picard's 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 [16]. These methods correspond to the DSPN computation methods mentioned in the beginning of this section.

3. FPGA-based NoC testbed

As a testbed for the evaluation of NoC communication scenarios and architectures two different FPGA development boards one featuring an APEX FPGA and the other one including a more complex Stratix II FPGA have been utilized. The application of the latter development board including a Stratix II device has become necessary to realize more complex NoC models. For basic NoC models the utilization of the APEX based development board has been sufficient. The development boards include flash memory, additional SRAM, serial interfaces (e.g. for downloading the program to the on-board flash memory) and several display options (e.g. LC display). Multi processor networks have been implemented on these development boards by instantiating Nios (APEX) and Nios II softcore processors (Stratix II), respectively, on the corresponding FPGAs. The synthesizable Nios embedded processors are general-purpose load/store RISC CPUs that can be combined with a number of peripherals, custom instructions, and hardware acceleration units to create custom system-on-a-programmable-chip solutions. The processors can be configured to provide either 16 or 32 bit wide registers and data paths to match given application requirements. Both data width versions use 16 bit wide instruction words. Version 3.2 of the Nios core typically features about 1100 logic elements (LEs) in 16 bit mode and up to 1700 LEs in 32 bit mode including hardware accelerators like hardware multipliers. The enhanced Nios II version, applicable only on Stratix FPGA devices, requires, depending on the processor configuration, from 700 up to 1800 LEs. More detailed descriptions of the components can be found in [1]. 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 [3] communication structure is used to connect devices to the Nios cores. Avalon is a dynamic sizing communication structure based on a switch fabric that allows devices with different data widths to be connected, with a minimal amount of interfacing logic. The corresponding interfaces of the Avalon communication structure are based on a proprietary specification provided by Altera [4].

In order to realize processor networks on this platform the so-called SOPC (system on a programmable chip) Builder [2] has been applied. SOPC 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

In the following, different test scenarios are described. The first two scenarios have been evaluated with the FPGA-based testbeds mentioned in Section 3 and modeled by means of DSPNs. The first scenario is a fundamental resource sharing conflict. This scenario has been evaluated on the APEX-based FPGA development board. Two Nios soft-cores are competing for a common used resource. The second scenario is a more complex network communication example. Three Nios II soft-core processors are included within this scenario. It was necessary for this example to use the more complex Stratix II based FPGA development board for the evaluation of this scenario. Additionally, a basic mesh-based NoC-topology is analyzed using a DSPN model. Here, the influence of NoC parameters like the burst length or the interval between two bursts on the resulting performance (expressed e.g. by the average establishing time of NoC-connections) is exemplarily discussed. The scenarios are explained in detail in the following subsections.

4.1. Fundamental resource sharing 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 [3] 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 [2] 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 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 incur no arbitration delay for uncontested or acquired cycles. Any mas-

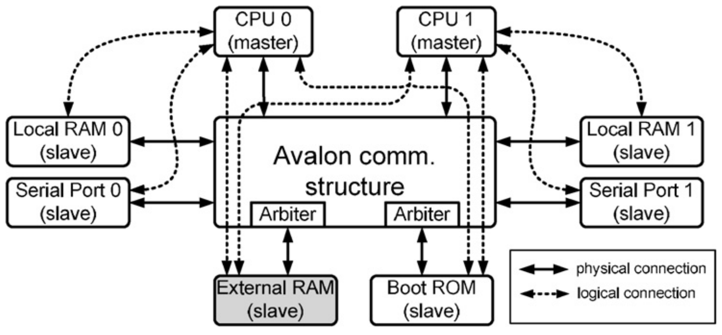

Fig. 2. Block diagram of fundamental resource sharing experiment.

ters 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:

- the priority shares assigned to each processor,
- the ratio of write and read accesses,
- the mean delay between memory accesses.

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. The insertion of NOPs does not lead to an accuracy loss related to pipeline stalls, cache effects or other unintended effects. The discussed examples are inherently constructed in order that these effects do not occur. In a first step, a *basic* DSPN model has been implemented (see Fig. 3) in less than two

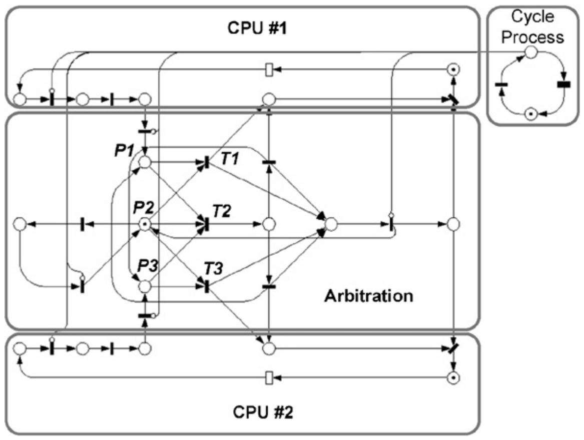

Fig. 3. Basic DSPN for rudimentary Nios example.

hours. Implementation times of the DSPN models are related to the effort a trained student (non-expert) has to spend to realize the corresponding models. The training time for a student to become acquainted with DSPN modeling lasts a couple of days. Distinction between read and write accesses was explicitly neglected to achieve a minimum modeling complexity. The DSPN consists of four substructures:

- two parts represent the load generated by the Nios cores (**CPU #1** and **#2**)
- a basic **cycle process subnet** providing a clock signal
- the more complex **arbitration subnet**

Altogether, this basic model includes 19 places and 20 transitions. The immediate transitions T1, T2 and T3 and the belonging places P1, 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.

The modeling results discussed in the following have been acquired by application of the iterative evaluation method. Though the modeling results applying this basic DSPN model are quite accurate (relative error less than 10% compared to the physically measured values, see Fig. 5), 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 enhanced DSPN model depicted in Fig. 4. The implementation of this *enhanced* model has taken about three times the effort in terms of implementation time (approximately five hours) than the basic model described before. The DSPN model consists now of 48 transitions and 45 places. Compared to the basic model the maximum error has been further reduced (see Fig. 5). The enhanced 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 basic and enhanced DSPN model. Fig. 6 shows the relative error for the throughput which is achieved by varying the mean number of computation cycles between two attempts of a processor to access the memory. On average the relative error of calculated memory throughput is reduced by 4–6% with the transition from the basic to the enhanced model. Using the enhanced DSPN model the maximum estimation error is below 6%.

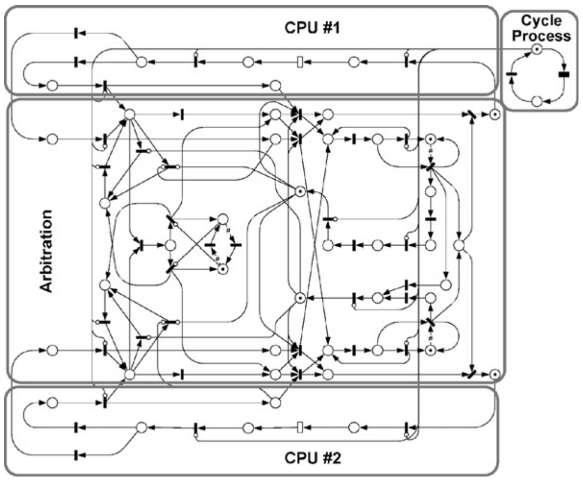

Fig. 4. Enhanced DSPN for rudimentary Nios example.

As mentioned before, the evaluation of DSPNs can be performed by different methods (see Fig. 7). 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

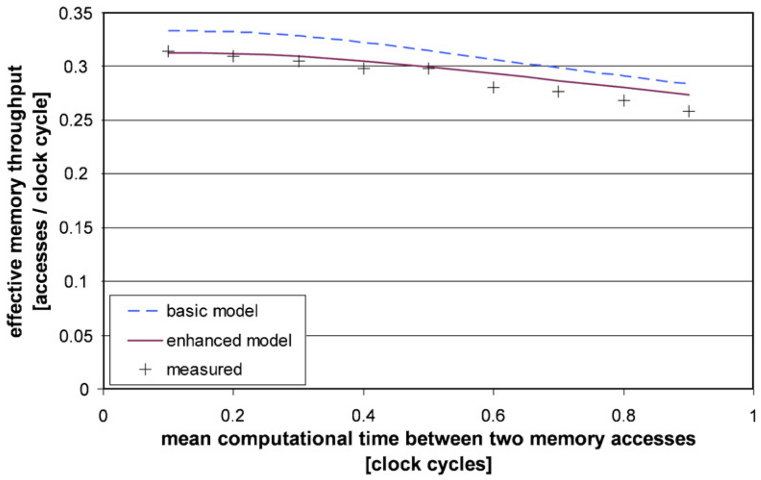

Fig. 5. Effective memory throughput comparison of the basic and the enhanced DSPN model as well as the measured values.

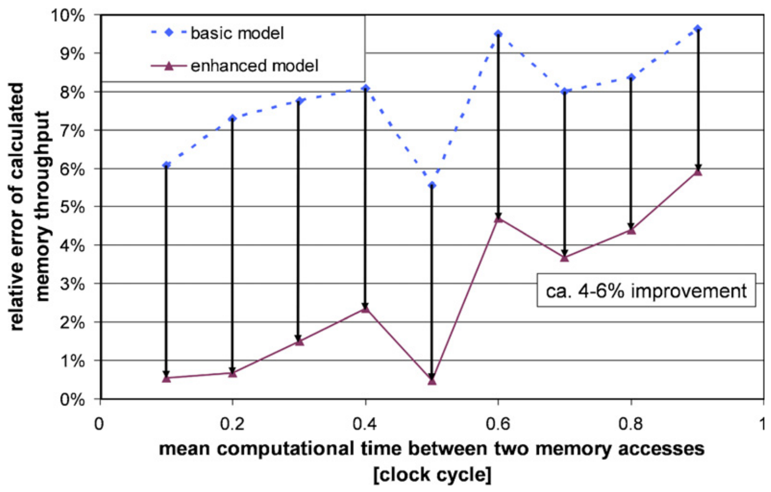

Fig. 6. Relative error of effective memory throughput for the basic and the enhanced DSPN model.

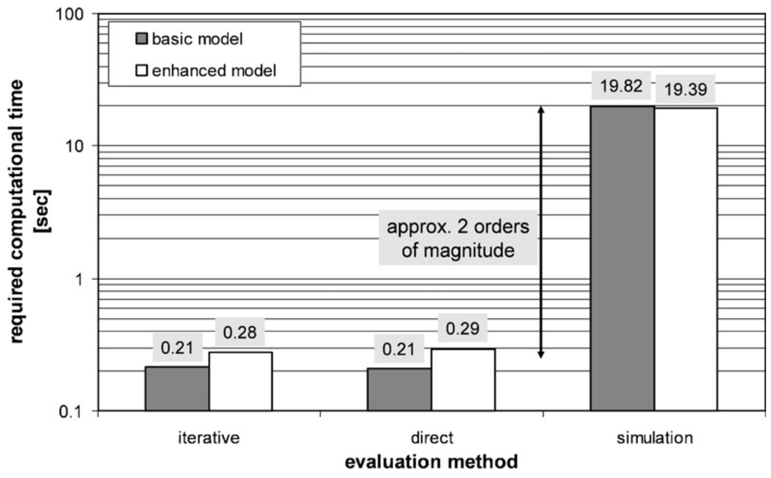

Fig. 7. Required computational effort for the different evaluation methods for the realized DSPN models.

methods. The simulation parameters have been chosen in such a way that the simulation results match the results of the analytic approaches. The applied DSPN-Software provides an iterative method (Picard-iteration-method [16]) and a direct solution method (general minimal residual method [16]). Fig. 7 illustrates a comparison of the required computational time for the analysis and the simulation of the introduced basic and enhanced DSPN-models. For the example of the enhanced model the computation time of the DSPN analysis method only amounts to 0.3 s and the DSPN simulation time (10^7 memory accesses) amounts to 20 s on a Linux-based PC (2.4 GHz, 1 GByte of RAM) using the software environment DSPNexpress (version DSPNexpress-NG 2003 [11]). The difference between the iterative and direct analysis method is hardly noticeable.

4.2. DSPN model of a hierarchical NoC

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. 8).

Nios II processor kernels have been applied to realize the depicted processor network (version: Nios II/f, approximately 1800 LEs). The SOPC Builder only provides limited configuration options

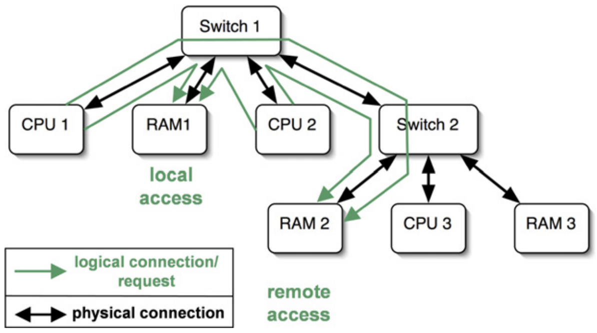

Fig. 8. Exemplary hierarchical NoC architecture.

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. Different custom-made communication components and structures (e.g. switches) have been realized in VHDL providing the possibility to implement various routing- and arbitration-schemes.

In the example 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 negative exponentially distributed. CPU 1 and 2 generate both local and remote memory accesses (accesses to a far away residing memory region within the network i.e. RAM 2). CPU 3 only accesses local memory sections (RAM 2/3). Fig. 8 sketches the network topology and the logical connections between components.

The DSPN developed to model this setup consists of 205 places, 178 transitions as well as 535 arcs, and provides several parameters defining the network behavior that have been used to set up various experiments. The modeling effort accounts for approximately 19 h.

Fig. 9 shows results of a comparison between measured values and values determined by application of the corresponding DSPN-model. The effective memory throughput (accesses per clock cycle) of CPU 1 and 3 is depicted over the probabiltiy of a local access of CPU 1 and 2 (instead of a remote access). The probability for local accesses is varied equally for CPU 1 and 2. Variation of local access probability affects the behavior of CPU 3 as higher local access probability results in less contention

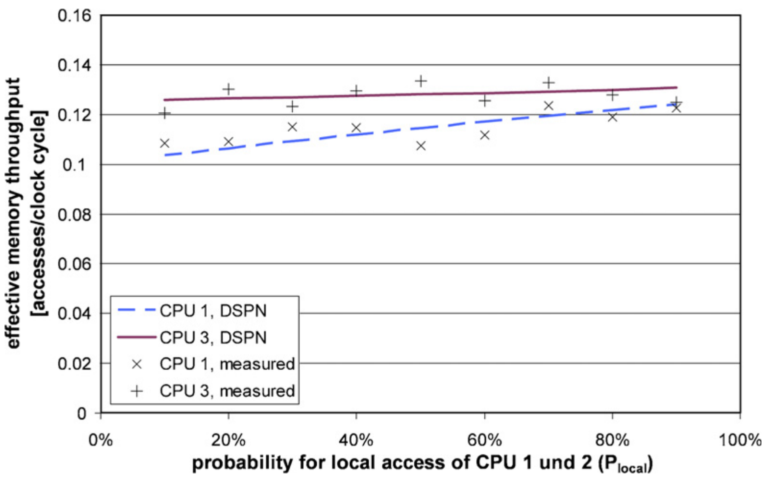

Fig. 9. Effective memory throughput of the hierarchical NoC-model.

scenarios for the ports of switch 2 and vice versa. The results show that the effective memory throughput of CPU 3 is slightly increased with increasing percentage of local accesses of CPU 1 and 2, respectively. The memory throughput of CPU 1 increases with increasing probability of local accesses as expected. Only switch 1 has to be acquired when all memory accesses of CPU 1 and 2 are local. Therefore, contention cannot occur for switch 2.

Again the DSPN-model yields good estimation results as can be seen in Fig. 10 that depicts the relative error between measured values and values determined by application of the DSPN model. The maximal relative error is less than 6.5%. The results related to Figs. 9 and 10 have been determined by means of the iterative evaluation method.

Evaluation of this enhanced DSPN took approximately 15 s for the iterative method on the Linux PC mentioned in the previous section, whereas the direct evaluation method required about 25 s on the same machine.

4.3. DSPN model for a mesh-based NoC

In order to analyze further basic NoC architectures using DSPN models, a generic DSPN model allowing to efficiently vary various related NoC parameters i.e. NoC topology, arbitration scheme, line-switching scheme and routing scheme, has been built. Here, an experiment based on a regular mesh topology depicted in Fig. 11 is presented. This example has not been emulated on an FPGA-based platform but analyzed on the basis of the generic DSPN-model.

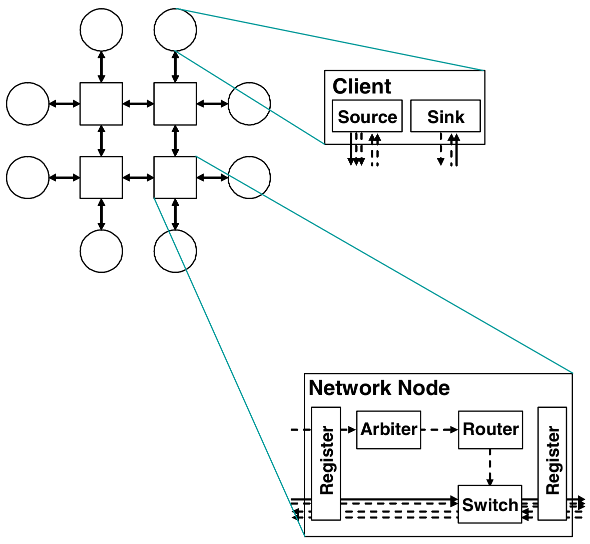

Fig. 11. Regular mesh NoC-topology featuring eight clients (sources/sinks) and four network nodes.

In this experiment two arbitrary clients out of eight clients have been configured as data sources and all eight clients are possible data sinks. In each network node connections to all four direct neighbours are realized. Routing inside this NoC is performed by a deterministic routing scheme (horizontal routing first, vertical routing second). The inputs of the network nodes are prioritized for arbitration purposes, according to the following scheme: west, east, north, south. Line-switching is applied as switching technology. The lengths of data

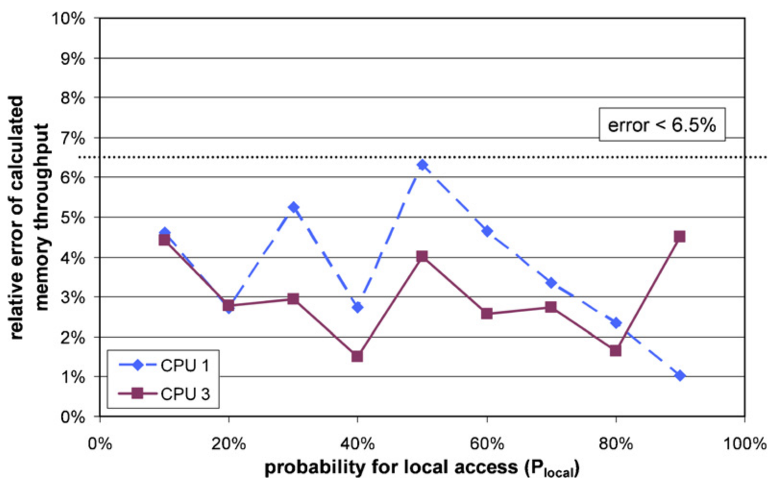

Fig. 10. Relative error of memory throughput of the hierarchical NoC model.

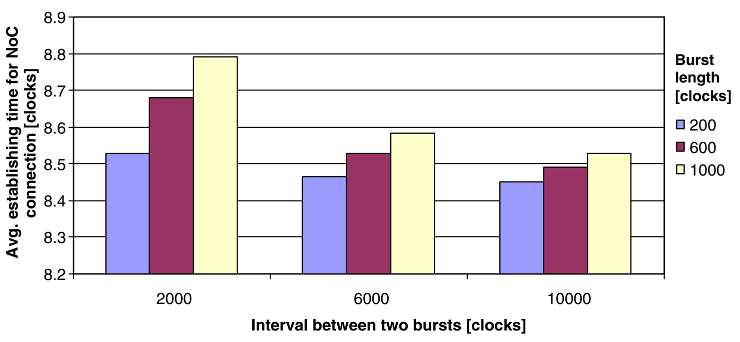

Fig. 12. Average establishing time for a requested NoC connection in dependency of the number of clocks between two bursts and the burst size.

bursts as well as the intervals between two bursts can be configured. The data sinks of each transfer are arbitrarily distributed. These parameters have been chosen in that way that a high amount of data transfer conflicts (data contention) arises. This example proves that even in cases with a very high amount of data contention a DSPN modeling of such a situation is a suitable and efficient means. The DSPN modeling effort of the single components of this NoC amounts to the following:

- client (incl. data source/sink): 17 places, 13 transitions,
- network node (incl. register, arbiter, router, switch): 133 places, 137 transitions.

In total this DSPN features about 500 places and 600 transitions. Using this DSPN model it can be examined for example what is the average establishing time for a requested connection inside this NoC. In Fig. 12 this establishing time (in clock cycles) is depicted in dependency of the burst length and the number of clocks between two bursts. As can be seen in this figure, the average establishing time of a connection in this NoC decreases with increasing length of the intervals between two bursts. At the same time, the average establishing time increases with increasing burst length due to a higher amount of routing conflicts inside the NoC.

The computational effort for this experiment on the Linux-PC described in the section before, amounts to 30 s for the direct solution and 90 s for the iterative solution. The structure analysis of the DSPN and the computation of the so called reachability graph which has to be performed once in order to determine characteristics of a DSPN model like the absence of deadlocks (non-solvable blockades), or the so called liveness of a graph etc. [16] requires about 30 min. The modeling effort for this DSPN model accounts to approximately 10 h. Based on this generic DSPN model it is possible to efficiently vary parameters like e.g. the routing algorithm (*x*–*y*-routing, adaptive routing, etc.), the topology (mesh, torus, ring etc.) or the applied arbitration schemes (prioritized, round robin, etc.) and to inspect their influence on the resulting NoC performance (for a discussion of main NoC parameters see e.g. [7]).

All the examples discussed in this paper show, 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 either connected via an Avalon communication structure or via a more complex custom-made hierarchical communication structure it could be shown that the modeling results are very close to the values measured on an FPGA testbed. For these examples 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. Analyzing NoC parameters like the influence of the distribution of read and write accesses, this example shows that DSPNs can be advantageously leveraged for early stage modeling of communication processes. By use of a generic DSPN model, the influence of parameters like the burst length or the intervals between two bursts on the average establishing time of an arbitrary connection on a mesh-based NoC has been studied. Our current work includes the extension of this DSPN modeling technique to a hierarchical approach which enables an even faster DSPN modeling of more complex NoC scenarios and facilitates easier reuse of subnets in new more complex DSPN models. Currently, we are modeling NoC topologies like more complex mesh, ring or torus topologies and analyze NoC features like arbitration schemes or aspects like quality of service.

References

[1] Altera: Nios Embedded Processor Software Development Reference Manual. 2001.

[2] Altera: SOPC Builder. <http://www.altera.com/products/software/products/sopc/sop-index.html>.

[3] Avalon: Bus specification manual. <http://www.altera.com/literature/manual/mnl_avalon_bus.pdf>.

[4] Avalon: Interface specification. <http://www.altera.com/literature/manual/mnl_avalon_spec.pdf>.

[5] L. Benini, G. DeMicheli, Networks on Chips: a new SoC paradigm, IEEE Computers (2002) 70–78.

[6] L. Benini, Application specific NoC design, DATE (2006) 491–495.

[7] T. Bjerregaard, S. Mahadevan, A survey of research and practices of Network-on-Chip, ACM Computing Surveys 38 (2006).

[8] H. Blume, T. von Sydow, T.G. Noll, A case study for the application of deterministic and stochastic Petri Nets in the SoC communication domain, Journal of VLSI-Signal Processing 43 (2-3) (2006) 223–233.

[9] H. Blume, T. von Sydow, T.G. Noll, Performance analysis of SoC communication by application of deterministic and stochastic Petri Nets. in: Proceedings of the Samos'2004 Workshop, Samos, Greece, July 19–21, 2004, Springer, LCNS 3133, pp. 484–493.

[10] G. Ciardo, L. Cherkasova, V. Kotov, T. Rokicki, Modeling a scalable high-speed interconnect with stochastic Petri nets, in: Proceedings of the Sixth International Workshop on Petri Nets and Performance Models PNPM'95 October 03–06, 1995, Durham, North Carolina, USA, pp. 83–94.

[11] DSPNexpress: <http://www.dspnexpress.de>.

[12] A. Jantsch, H. Tenhunen, Networks on Chip, Kluwer Academic Publishers, 2003.

[13] L. Kleinrock, Queueing Systems – Vol. 1: Theory, John Wiley and sons, 1975.

[14] T. Kogel, M. Doerper, et al., A modular simulation framework for architectural exploration of On-Chip interconnection networks, CODES + ISSS, October 2003.

[15] K. Lahiri, A. Raghunathan, S. Dey, System-level performance analysis for designing On-Chip communication architectures, IEEE Transactions on CAD of Integrated Circuits and Systems, June 2001.

[16] C. Lindemann, Performance Modeling with Deterministic and Stochastic Petri Nets, John Wiley and Sons, Berlin, Germany, 1998.

[17] J. Madsen, S. Mahadevan, K. Virk, Network-centric system-level model for multiprocessor SoC simulation, in: J. Nurmi et al. (Eds.), Interconnect Centric Design for Advanced SoC and NoC, Kluwer Academic Publishers, 2004, 2004.

[18] M.H. Mickle, Transient and steady-state performance modeling of parallel processors, Applied Mathematical Modelling 22 (7) (1998) 533–543.

[19] M. Neuenhahn, H. Blume, T.G. Noll, Quantitative analysis of network topologies for NoC-architectures on an FPGA-based emulator. in: Proceedings of the URSI "Advances in Radio Science – Kleinheubacher Berichte", Miltenberg, Germany, September 2006.

[20] J. Nurmi, H. Tenhunen, J. Isoaho, A. Jantsch, Interconnect Centric Design for Advanced SoC and NoC, Kluwer Academic Publishers, 2004.

[21] Petri Nets World: <http://www.informatik.uni-hamburg.de/TGI/PetriNets/>.

[22] J. Plosila, T. Seceleanu, K. Sere, Formal communication modeling and refinement, in: J. Nurmi, H. Tenhunen, J. Isoaho, A. Jantsch (Eds.), Interconnect Centric Design for Advanced SoC and NoC, Kluwer Academic Publishers, 2004.