A Cycle-Accurate Network-on-Chip Simulator with Support for Abstract Task Graph Modeling

Jan Moritz Joseph, Thilo Pionteck
Universität zu Lübeck
Institute of Computer Engineering
23562 Lübeck, Germany
Email: {joseph, pionteck}@iti.uni-luebeck.de

***Abstract*—This papers presents the design of a Network-on-Chip (NoC) simulator for design space exploration of router architectures. The simulator supports cycle-accurate router models and in addition allows the simulation of router architectures, which can adjust their processing according to the traffic type. Realistic traffic patterns are derived from task graph models of real-world applications that are simulated in parallel to the NoC at transaction level. Combining cycle-accurate router simulation and abstract task graph simulation circumvents the limitations of most NoC simulators, which either use synthetic traffic patterns or unrealistic and fixed router models. The proposed simulator architecture is presented in detail and its suitability is shown by means of a case study.**

***Index Terms*—Network-on-Chip, simulation, task models**

I. Introduction

So far, the number of components in System-on-Chips (SoCs) or of cores in multi-core processor systems has vastly increased. This trend asks for a communication infrastructure whose bandwidth and complexity scales linearly with the number of components. Network-on-Chips (NoCs) [1] meet this demand as the number of routers can easily be adjusted to the number of cores. Yet, with each new router the maximum number of hops and thus the maximum communication latency increases. Depending on the kind of application this can lead to significant performance degradations, e.g. [2]. To cope with this problem, task mapping, the routing strategy, or the actual router hardware architecture can be modified. As the effectiveness of these modifications depends on each other, extensive design space explorations by means of simulations are required to identify the optimal set of parameters for a dedicated application.

For performing a realistic design space exploration a simulator has to meet diverse demands. On the one side task mapping and routing algorithms ask for high abstractions levels and realistic traffic patterns. On the other side, router hardware architecture optimizations require low-level simulations. Therefore a simulation environment is needed that offers multiple levels of abstraction. The NoC itself must (at least) be simulated cycle accurate. NoC routing algorithms and the application can be modeled on a coarser level. Modeling transactions between tasks is sufficient as they represent traffic in the underlying communication infrastructure of the system. Most NoC simulators do not consider all these aspects at once. They either operate on a high abstraction level for evaluating algorithms [3] and use realistic traffic patterns, or on a low-level for testing router architectures using synthetic traffic patterns [4]. While the first neglects potential speed-ups caused by adaptive router pipeline architectures, e.g. varying latencies depending on the kind of data streams, the latter do not provide a realistic simulations due to the use of synthetic traffic patterns. The use of synthetic traffic also impedes the evaluation of router architectures that can adjust their processing according to the type of incoming traffic.

This work closes this gap by proposing a NoC simulation architecture, which addresses all relevant aspects for design of NoCs. The simulator is written in SystemC and makes use of the different modeling styles supported by SystemC. Timing for the NoC router is modeled cycle accurate while tasks are realized as modules that communicate via transaction level modeling (TLM) interface ports. The different modeling styles are connected via an interconnect unit that translates TLM-interface calls into cycle-accurate simulated packages and vice versa.

The rest of this paper is organized as follows. Section II discusses related work and summarizes the key features of the most popular NoC simulators. After deriving a requirement list in section III, the proposed NoC simulation model is introduced in section IV. Details on the SystemC implementation of the simulator are given in section V and section VI shows the suitability of the proposed simulator architecture by means of a case study. Finally, section VII concludes this paper.

II. Related Work

Beside diverse simulators specific for a dedicated NoC design, e.g. [5], there exists many universal NoC simulators for design space exploration. According to [6] the two most popular NoC-specific simulators are BookSim [7] and Noxim [4], which offer many features and are actively developed for years. The latter is implemented in SystemC. It offers customizations of network parameters such as sizes of buffers and packages. Furthermore, the routing algorithms can be exchanged. Different types of synthetic traffic patterns can be simulated in the NoC. Without modifying the source code, realistic traffic cannot be injected (for example by simulating an application). Noxim allows the evaluation of metrics such as throughput, packet delay, and estimated power consumption. The timing in the unmodified routers in Noxim is not fully suitable for

all explorations of the design space. Routers are divided into a receive and a transmit process for pipeline behavior. If a flit arrives, it is handled by the receive process and stored in the input buffers. In the next clock cycle, the router's transmit process calculates the routing and the arbitration for a head flit. Furthermore, regardless of the type of the flit, it is transmitted to the next router one clock cycle after it is received. Thus, routers work on every flit an equal amount of time. This may not be the behavior one expects from a cycle-accurate router, since routing calculation and arbitration for head flits usually takes a minimum of two additional clock cycles. Although rather seldom, this time may vary for special routing algorithms or router's architectures. The simulator Booksim supports similar features as Noxim. It is written in C++. A wider range of network topologies are implemented and the router's architecture along with the routing algorithms can be adopted in comparison to Noxim. Here again, only synthetic traffic patterns can be injected. Summing up, both simulators operate on a low level of abstraction. Furthermore, they are restricted to synthetic traffic patterns, which are generated locally by processing elements attached to the local ports of routers.

Simulation of NoCs on a more abstract level with realistic traffic patterns can be done by using universal network simulation frameworks such as OMNetT++ [8] and NS-2/-3 [9]. Actually, both target the simulation of network traffic in physically wide-spanning networks, yet likewise can be used to simulate network traffic on a chip. A typical application of these simulators in the context of NoCs is the evaluation of routing algorithms. On the plus side, these simulators allow fast modeling of application based traffic. On the downside, both simulators are not primarily focused on on-chip simulations thus, for example, do not integrate interconnections to synthesizable designs by default. A less abstract representation of the network components itself is often desired.

A combination of low-level router simulation along with application-based data streams is done in the NocBench project [10]. The NocBench traffic generator simulates applications by means of task graphs and injects traffic into the actual NoC simulator. The traffic generator connects to the routers in the NoC via OCP-TLM (Open Core Protocol Transaction Level Modeling), which requires the OCPIP-Kit [11]. The kit is under proprietary license. The software model in the traffic generator is based on Kahn process networks [12]. Applications such as UMTS-modems, an H.263 compatible video encoder and decoder, or an audio-video benchmark have been implemented for the NocBench traffic simulator [13].

Another simulator offering parallel NoC simulation and traffic generation is DARSIM [14]. The NoC is modeled on cycle-accurate level. In addition, traffic can be injected into the network from cycle-accurate simulated cores. The cores execute compiled bit-code, which is highly accurate yet lacks in performance. Therefore, the simulator allows for traffic injection from trace-files instead. The files define the bandwidth, generation time and period of outgoing traffic-streams per core. The properties of the streams are static and do not change during the simulation. The architecture for a simulator as proposed in this paper does not have the performance degradations as the cycle-accurate simulation of cores in DARSIM. Yet it offers traffic patterns that are responsive to properties of the NoC since outgoing traffic is generated dynamically with respect to incoming data streams.

III. Requirements

As shown before, most NoC simulators either focus on realistic router architecture simulation or realistic traffic patterns. Yet, only the combination of both aspects allows a comprehensive design space exploration. This is especially true when simulating router architectures, that are capable to analyze the characteristics of data streams and show different behavior for different kind of traffics. For example, routers that are capable to detect and prioritize semi-static data streams such as proposed in [15] can only be tested when applying traffic patterns of applications in which semi-static data streams exist. These data streams are characterized by a number of consecutive packets following the same path for a longer period of time. Such scenarios can be found in several multimedia and signal processing systems [16] or processors with shared memory or caches [17]. In order to see the effect of prioritizing semi-static data streams in a simulation it is also required to simulate routers cycle accurate such that the delay in clock cycles for a flit can be altered. Thus, an ideal NoC simulator should provide the following features: Routers and routing algorithms must be simulated with cycle accurate timing to test different router architectures, designs, or algorithms. For realistic performance evaluation application-based on-chip network traffic is necessary. Thus an application must be simulated that injects traffic into the NoC. The description of the application should be abstract allowing for the implementation of a variety of use-cases. Well-known measured variables must be recorded for benchmarking. Furthermore, indicators are to be saved that are specialized for the indented use of the NoC.

None of the above mentioned simulators offers all these features at once since multiple levels of abstraction are required. Thus we propose an architecture for a NoC simulator that models both the on-chip communication infrastructure and an application layer to inject realistic traffic patterns into the chip. Therefore a simulation environment is needed that offers diverse abstraction levels: The NoC must be simulated cycle accurate. It is favorable if routers can partially use synthesizable design parts via interconnects. The application can be modeled on a coarser level. Only transactions between tasks are of interest as they represent traffic in the underlying communication infrastructure of the system. All these requirements are met by SystemC; it offers different modeling styles from bit accurate RTL-level, over cycle-accurate level up to transaction level (TLM) [18], here sorted by ascending abstraction. The more abstract styles offer better simulation performance, since less context switching is required or, as in case of TLM, context switches only occur when necessary.

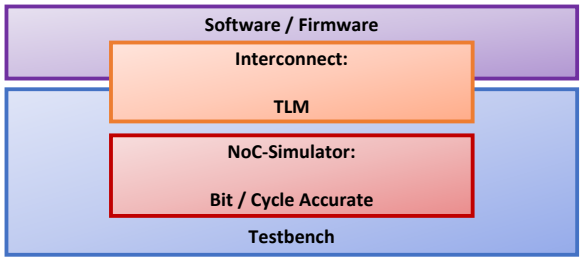

Fig. 1. An overview of the proposed architecture for a combined NoC and application simulator. The NoC simulator is depicted in red. It's architecture and properties is managed by a test-bench (blue). On top of the chip simulation there is an application simulation (violet). Both components are connected by the interconnect (orange).

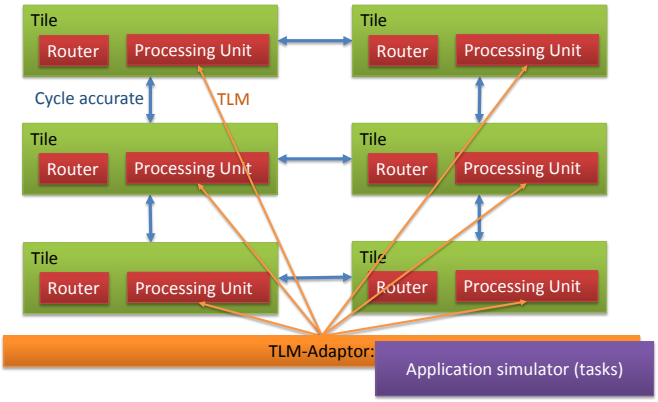

Fig. 2. Detailed view of the proposed architecture. Tasks in the application communicate via TLM (orange). The interface call is converted into a cycle accurate simulation via an adaptor. The NoC consists of processing elements that send packages via routers (blue).

IV. Simulator

In this section the simulator is introduced. It complies with the requirements as given in sec. III. At first, we will show key features of the architecture; we will especially focus on the interaction between the different abstraction levels and modeling styles. Secondly, a task model for this purpose is defined.

A. Architecture

An overview of the proposed architecture is given in fig. 1. In the bottom layer, the NoC simulator is depicted in red. It is on a low abstraction level and works cycle accurate. The NoC's architecture and properties are managed by a test-bench (blue). In the top layer there is an application simulation (violet), which simulates software using transaction level modeling. Both layers are connected by an interconnect (orange). The interconnect unit maps tasks to processing elements and vice versa. In addition it handles the different types of abstraction in the two layers of the system.

The data paths between the two layers are shown in fig. 2. The color-coding throughout the figures stays consistent. Routers and processing elements are paired in tiles. These can communicate according to the topology of the NoC. The application simulator can register packets at processing elements, which are the generalized representation of components of the system. Thus, the simulated software is running on a distributed system or a firmware that induces communication between different components of a SoC.

A detailed view on the data path between the two layers of the system is shown in fig. 3. The application simulation can function in two modes: To simulate the data flow on task level without realistic timing, the underlying communication infrastructure can be disabled. If timing information are required, tasks communicate indirect via the simulated communication infrastructure. Both modes have the following properties:

In the first mode (application only), a task will directly call a TLM-interface on another task. Routing is done by a single, global router. Buffered behavior must not necessarily be supported by the system. The interface call can be blocking since timing information is not relevant and the function call returns immediately. The resulting data flow can be used to debug the application simulation. Furthermore, information are provided to analyze the application. For example, emerging traffic patterns can be identified. This mode is shown in fig. 3 in the upper part (orange).

In the second mode, the application is simulated along with the NoC. Thus, the model yields timing information. Parameters of the NoC can be adopted and their influence on the system's performance can be evaluated. In this mode, a task will call an interface in the interconnect unit. This interface call must be non-blocking since the communication infrastructure has buffering behavior. Otherwise tasks would stall until the destination task has received the data, which is obviously not desired. Again, the interconnect unit will transform between the different modeling styles. In addition it has to take care of the mapping of tasks to processing elements. This mapping is a (partial) function. The inverse mapping is not necessarily a function; thus scheduling in processing elements needs to be considered. The data flow between two tasks in this mode is more complicated: Tasks call the forward interface on the interconnect unit. The unit will register a packet at the processing element according to the task mapping. Processing elements separate packages into flits and send those via the NoC to the destination element (the destination is given by the mapping unit). After the packet was received, the processing element calls the backwards interface of the mapping unit. This call is forwarded to the corresponding task. The data path is shown in the lower part of fig. 3.

Finally, a detailed view of tiles is given in fig. 4. The grid of the NoC consists of tiles. The router's architecture for two-dimensional meshed NoCs is given here. In general, the proposed architecture supports all topologies if the routers are modified accordingly. Within a tile there is one router and processing element each. Both are connected via interface ports that can pass flits. There are data and valid ports in each direction. The routers are connected among themselves such that the desired topology is generated. A two-dimensional mesh was used here. Routers are simulated cycle-accurate

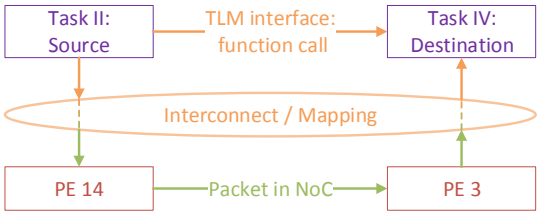

Fig. 3. The path of a transaction between two tasks is shown. The sending task can either send its data via an interface call directly to the destination task (orange) or it can use the simulated chip communication infrastructure. The interconnect unit maps tasks to processing elements and registers packets at their outwards buffers. Those are send via the simulated NoC (green).

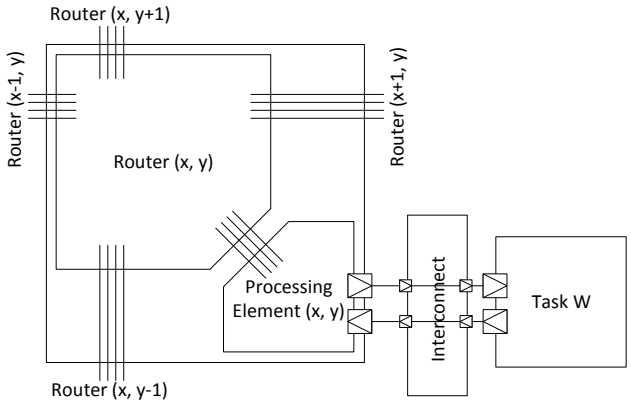

Fig. 4. Here the architecture of a simple router is shown. It is locally connected to a processing element and globally to its neighbors following mesh topology. As an example, a work task connects to the processing element.

and thus offer the flexibility to handle differing types of flits with a different number of cycles. The processing element offers interface ports, which are connected to the application simulator. These transaction level modeled ports are depicted in fig. 4 on the right side of the schematics. Since tasks and processing elements do not necessarily have the same address, the units are connected via an interconnect that handles the conversion of interface calls. The interfaces are modeled on transaction level beside the task mapping. Pipelining and buffering is achieved via separate in- and outports, where calls are non-blocking.

B. Task Graphs

In this section we introduce the task graph model used for our simulator that is capable of modeling real-world application data streams. Our work is inspired by Kahn process networks (KPNs) [12], in which processes communicate via unbounded buffers. Either synchronous models of computation (syn. MoCs) [19], [20] or data-flow process networks (DPNs) [21] can be used as a basis. The latter are defined for signal processing problems as a data flow centered communication model. Roughly speaking, synchronous MoCs divide the notation of time in rounds. In every round parallel processing units evaluate inputs and generate outputs. The results are deterministic and synchronous. In contrast,

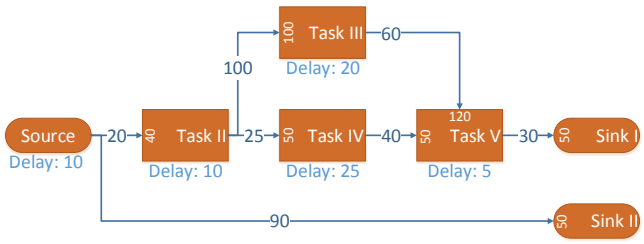

Fig. 5. Exemplary task graph with seven tasks. Special tasks (sinks and sources) are marked. Tasks are connected via ports and the amount of data along the vertexes is given. Furthermore, a delay is associated with tasks.

KPNs are asynchronous. DPNs are a special case of KPNs, in which processes communicate via unidirectional bounded buffers [22].

For our purposes a communication-centered and timed task model is needed. None of the models is fully suited for our purposes since a less abstract annotation of time is required. Limitations to the size of the buffers between processes are given by the simulated communication infrastructure and thus do not need to be considered in the task model. Tasks itself may communicate unbounded, yet the author of the application itself must take care that the infrastructure on the chip is capable to handle the bandwidth.

Derivation of data flow graphs of real-world applications is out of the scope of this work. Thus, we rely on data flow graphs published in literature, e. g. the data flow graph for a MPEG-4-decoder as presented in [23]. In general, data flow graphs are modeled as syn. MoCs or DPNs with typical bandwidth along the edges. The only addition needed to implement those into out task model is the delay of tasks, which has to be derived from the given data manually.

Our application model uses task graphs where vertexes represent tasks. These are connected via directed, weighted edges if they communicate. An exemplary task graph with seven tasks is shown in fig. 5. Special tasks (sinks and sources) are labeled. Tasks are connected via ports and the bandwidth along the edges is given as edge weight. A task sends data downstream if it received data from the upstream connections and processed those. Therefore, the processing time (delay) is associated to tasks. It defines the time a task is idle between receiving and sending data. The task will start processing if it has collected all required data, whose amount is given per incoming edge. Sources and sinks are modeled via tasks that exclusively have out- or in-going edges.

We extended the model such that a task may send data after receiving information from either all incoming connections or only one of these. Thus the model supports both flexible joining and forking of data streams. Sources send their data periodically with their delay. Thus applications will either execute periodically or terminate outright. To avoid periodic behavior of applications, we extend the task definition including an upper bound for the number of executions.

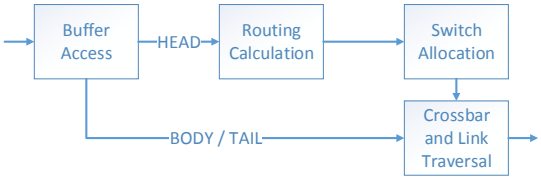

Fig. 6. A head flit requires four clock cycles per router (buffering, routing calculation, switch allocation, and link traversal). Body and tail flits each need two clock cycles, since routing and arbitration is already done in the head flit.

V. Implementation

The task-model was implemented in SystemC as well as a basic NoC-based communication infrastructure. The NoC offers dimension order routing with wormhole switching. The timing of the NoC is done cycle accurate such that handling a head flit requires four clock cycles per router (buffering, switch allocation, routing calculation, and link traversal). Body and tail flits each need two clock cycles, since routing and arbitration is already done in the head flit as shown in fig. 6. Activity in the communication ports between routers is indicated via a valid line. The routers have internally a hierarchical architecture. They are divided into buffers, units that manage arbitration and routing, and a state machine per input. This allows simple modifications for example of routing algorithms. Routers are divided into two separate processes. The first handles the inputs and stores valid flits in the buffers with one cycle of delay. The second process routes flits to the outputs. The delay is variable as given above. The outward process is implemented as a state machine that has individual states per input. There are no outward buffers in the router's design. Flits are transmitted to the buffers of the router downstream and then deleted in the current input buffers.

Tasks are realized as modules that communicate via TLM-interface ports. The interfaces define the message passing protocol. All transaction level modeled ports were implemented using custom interfaces. Messages consist of packages, that contain a payload along with basic information such as the source and destination. In the upper layer of the simulation, single packages with the size of the data streams are passed for better performance. Data streams are divided into single packages with constant size in the processing elements of the NoC. The task simulator runs on its own or parallel to a chip simulation as defined. The task scheduling in the processing elements was done event driven, as tasks are enabled via communication activity.

Both the task mapping and the definition of the application for the task model is done via Extensible Markup Language (XML) for easy user interaction. Tasks are declared as an element with name "task". The task-element is a complex type including a sequence of elements with both ingoing and outgoing ports, and parameters of the task itself. Sources and sinks are defined via the existence of ports. Furthermore, tasks have the following attributes in the parameters tag: The identifier of the task, the type of the task, and the maximum number of executions. The type determines if a task executes after collecting data from all connections or a single one. Thus a very flexible joining and forking of data streams can be simulated in the application. Connections between tasks are realized via the identifiers of the tasks. For outgoing connections the amount of data per execution is specified. This leads to a XML-file with the following structure per task:

```
<task id="0" max_executions="1" type="false">
        <inport id="0" data="1"/>
        ...
        <outport id="1">
                ...
        </outport>
        ...
        <parameters>
                ...
        </parameters>
</task>
```

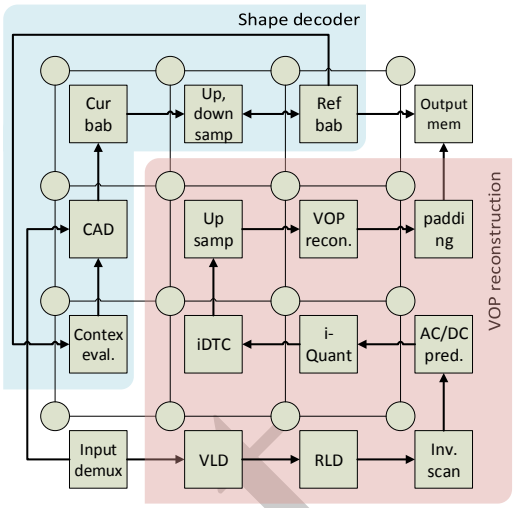

Fig. 7. Here the simulated NoC is shown with the mapping for the VOPD decoder. Processing elements are depicted as rectangles, router as circles. The first steps in each part of the calculation are shown. Sequential tasks have been mapped to adjacent elements. Functional units of the decoder are located as neighboring; in- and outputs units are located at the edges of the design.

VI. Case Study

An Video Object Plane decoder (VOPD) for a MPEG-4 compatible decoder was simulated using the proposed task model and implemented NoC. Data rates for the application are discussed in [23]. The VOPD runs on a simulated NoC with 2D-mesh topology and 16 processing elements (4x4). Since there are more than 16 tasks, multiple, not communication intensive tasks are mapped to single processing elements. Neither the task model nor the NoC simulator differentiate the type of the processing elements (e. g. between memory, CPU, or I/O). This is not necessary since we are interested in the data flow. The NoC design and task mapping is shown in fig. 7. There processing elements are depicted as rectangles and routers as circles. Mapped tasks are labeled with abbreviations consistent with [23]. Traffic utilization is minimized by mapping sequential tasks to adjacent elements. Functional units of the decoder are located as neighboring. In-

and outputs units are situated at the edges of the design.

The simulation of the VOPD-application can be done in two modes as introduced. In application-only mode, the simulation requires less CPU-time than with communication in the NoC. The difference is of magnitude 1/30, which is within the expected speed-up when using TLM [24]. This demonstrates that the application simulation yields a negligible overhead due to its high level of abstraction; the task model, as defined and implemented, has context switches only when required. This has a major impact on the simulation performance. However, avoiding context switches is not possible in all cases; for example, the routers have to be synchronized globally in a cycle-accurate simulation.

Besides the evaluation of standard NoC performance parameters, the proposed simulator architecture supports modifications within router models to monitor their internal processing. Thus, utilization of router components such as buffers or the arbitration process can be analyzed as well as the number and kind of intra-router transfers. These data can be used for a detailed performance evaluation of router architectures under realistic operating conditions. Furthermore, these data are the basis for an analysis of application-specific traffic patterns. As the emergence of special characteristics in traffic patterns depend on a realistic timing of the application model as well as on an accurate simulation of delays caused by each router, we think that our proposed simulator architecture offers a good basis for comprehensive NoC design space exploration.

VII. Summary and Outlook

In this paper an architecture for an simulator of application-based data flows in SoCs with Network-on-Chip infrastructure is proposed. The simulator unifies different levels of abstraction. Thus the resulting simulations are equally fast and accurate. Since the application model induces realistic traffic patterns into the NoC, the architecture distinguishes oneself by closing the gap between router design and application performance evaluation. The simulation of applications is modeled rather abstract so that this layer of the simulator has a low impact in the overall performance. The advantages of the simulator are its high accuracy in simulating timing and delay of the NoC for design space exploration and performance evaluation, the low impact of inducing application-based traffic patterns on the overall simulation performance, and its user-friendly interface via XML files.

The work on this project will continue as follows: We will further extend the routers' functionality. A central concern is the support of virtual channels and modifications in the routers' architecture to minimize delays. In addition we will implement more application models. Here, distributed cache systems are of special interest. Finally, we will test parameter alterations in the routers and their impact on the performance of the system.

Acknowledgements

This work is funded by the German Research Foundation (DFG) project PI 447/5-1.

References

[1] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[2] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *Computers, IEEE Transactions on*, vol. 54, no. 8, pp. 1025–1040, 2005.

[3] C. Grecu, A. Ivanov, P. Pande, A. Jantsch, E. Salminen, U. Ogras, and R. Marculescu, "An initiative towards open network-on-chip benchmarks," *OCI-IP White Paper*, 2007.

[4] F. Fazzino, M. Palesi, and D. Patti, "Noxim: Network-on-chip simulator," *URL: http://sourceforge. net/projects/noxim*, 2008.

[5] Z. Lu, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch, "Nnse: Nostrum network-on-chip simulation environment," *Proc. of SSoCC*, 2005.

[6] "Network-on-chip (noc) blog," http://networkonchip.wordpress.com/, accessed: 2014-08-11.

[7] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. Dally, "Booksim interconnection network simulator," *Online, https://nocs. stanford. edu/cgibin/trac. cgi/wiki/Resources/BookSim*.

[8] A. Varga *et al.*, "The omnet++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM2001)*, vol. 9. sn, 2001, p. 185.

[9] "ns-3," http://www.nsnam.org, accessed: 2014-08-14.

[10] S. K. Mandal, N. Gupta, A. Mandal, J. Malave, J. D. Lee, and R. Mahapatra, "Nocbench: a benchmarking platform for network on chip," in *Workshop on Unique Chips and Systems (UCAS)*, 2009.

[11] J. A. Colgan and P. Hardee, "Advancing transaction level modeling (tlm): Linking the osci and ocp-ip worlds at transaction level," *White paper. http://www. opensystems-publishing. com/whitepapers*, 2004.

[12] G. Kahn, "The semantics of a simple language for parallel programming," in *In Information Processing 1974: Proceedings of the IFIP Congress*, vol. 74, 1974, pp. 471–475.

[13] E. Pekkarinen, L. Lehtonen, E. Salminen, and T. Hamalainen, "A set of traffic models for network-on-chip benchmarking," in *System on Chip (SoC), 2011 International Symposium on*. IEEE, 2011, pp. 78–81.

[14] M. Lis, K. S. Shim, M. H. Cho, P. Ren, O. Khan, S. Devadas *et al.*, "Darsim: a parallel cycle-level noc simulator," in *MoBS 2010-Sixth Annual Workshop on Modeling, Benchmarking and Simulation*, 2010.

[15] T. Pionteck and C. Osterloh, "Prioritizing semi-static data streams in network-on-chips for runtime reconfigurable systems," in *High Performance Computing and Simulation (HPCS), 2013 International Conference on*, 2013, pp. 229–232.

[16] P. Wolkotte, G. J. M. Smit, G. Rauwerda, and L. Smit, "An energy-efficient reconfigurable circuit-switched network-on-chip," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005, pp. 155a–155a.

[17] I. Kotera, R. Egawa, H. Takizawa, and H. Kobayashi, "Modeling of cache access behavior based on zipf's law," in *Proceedings of the 9th workshop on MEmory performance: DEaling with Applications, systems and architecture*. ACM, 2008, pp. 9–15.

[18] P. R. Panda, "Systemc-a modeling platform supporting multiple design abstractions," in *System Synthesis, 2001. Proceedings. The 14th International Symposium on*. IEEE, 2001, pp. 75–80.

[19] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[20] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. De Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[21] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–801, 1995.

[22] D. Baudisch, J. Brandt, and K. Schneider, "Translating synchronous systems to data-flow process networks," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference on*. IEEE, 2011, pp. 354–361.

[23] E. B. Van Der Tol and E. G. Jaspers, "Mapping of mpeg-4 decoding on a flexible architecture platform," in *Electronic Imaging 2002*. International Society for Optics and Photonics, 2001, pp. 1–13.

[24] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2003, pp. 19–24.